2

Modélisation de la diffusion d'information sur les réseaux sociaux.

PROJET ARE DE L'ANNEE 2018/2019

Binga Thomas | Lacroix Yanis | Maupu Noah | Scart Gaetan

RESUME DU PROJET

Nous avons choisi de traiter un modèle dynamique d'actualité : la viralité des informations sur les réseaux sociaux. A travers l'outil de travail jupyter notebook et de plusieurs modules, nous avons tenté d'obtenir le modèle le plus réaliste possible. Typiquement, notre modèle permet de transmettre l'information qu'une personne « possède » et de la faire circuler dans un ou plusieurs groupes de personnes, tout cela en pouvant faire varier divers paramètres que nous développerons plus tard. Une importante part de notre travail a été d'obtenir un aperçu global de la transmission de l'information grâce à un GIF. De plus, si l'information se révèle être une Fake News, des détracteurs feront circuler une « contre-information » afin d'arrêter la diffusion de l'information fallacieuse.

PROJECT SUMMARY

We decided to work on a topical dynamic model: The virality of information on social media. We used the work tool jupyter notebook with different plug-ins to simulate social networks, we did the best we could to create a model close to reality. Typically, our model allows for someone to create an information and then make the information spread in one or several group of people. We would be able to change different parameters that we would take about later, and we will create graphs and GIFS of our results. Also, if the information happens to be a Fake News, some people would be able to spread a "counter information" which would explain that the information is fake and try to stop the spreading of the Fake News.

INTRODUCTION

Au début notre projet, nous avons chacun effectué des recherches sur les travaux déjà effectués sur le sujet afin de partir sur des bases solides. Malheureusement, aucuns travaux existants n'étaient réellement portés sur une modélisation d'un modèle de transmission sur les réseaux sociaux, la plupart traitaient des réseaux sociaux physiques (dans la vraie vie). Nous avons donc pensé que comme la thématique était récente, les travaux publiés n'étaient pas disponibles au public.

En cherchant des caractéristiques types, comme les proportions de personnes par tranches d'âge sur les réseaux sociaux, nous avons vite réalisé qu'il y avait de nombreuses valeurs auxquelles nous n'avions pas accès. Par exemple, les probabilités de renvoyer une information, de détecter si elle est fausse, ou juste d'avoir accès à l'information dépendent de beaucoup trop de facteurs. Nous n'avons donc pu retenir que certaines grandes idées, comme le fait que la part des utilisateurs de plus de 60 ans était importante, et que ces derniers partagent beaucoup d'informations sur les réseaux. Notre modèle ne sera donc pas une représentation fidèle à la réalité, mais nous avons tenté de nous en approcher au mieux.

Pour mener notre projet, nous nous sommes posé deux problématiques :

- Comment expliquer la diffusion de Fake News sur un réseau social comme Facebook ?
- Quels paramètres influent sur la rapidité de la diffusion d'une information ?

Nous détaillerons dans un premier temps les notions fondamentales sur lesquelles notre sujet se repose, puis nous présenterons les différents modèles que nous avons développés. Enfin, nous présenterons nos résultats et leurs interprétations, avant de conclure sur notre projet.

PRESENTATION DE LA THEMATIQUE

Depuis la révolution numérique de la fin de XXème siècle, internet a pris une place proéminente dans notre société. Cela a par exemple permis le développement de nombreuses plateformes facilitant grandement le passage d'informations à travers le monde. Aujourd'hui extrêmement populaire, le réseau social Facebook, regroupant plus de 2 milliards de comptes actifs est une des plateformes les plus utilisée d'internet. Cette plateforme permet à des personnes éloignées géographiquement de pouvoir discuter simplement. Quand une personne partage une publication sur son profil, tous ses amis peuvent potentiellement la voir, et peuvent s'ils le souhaitent la repartager à leur tour : c'est ce système que nous avons essayé de modéliser.

Néanmoins, un des récents problèmes de notre société est présent massivement sur les réseaux sociaux : la diffusion de Fake News. En effet, les réseaux sociaux permettent à n'importe qui de partager des informations, qu'elles soient vraies ou fausses. Sans esprit critique, on se retrouve donc submergé de Fake News.

Pour simuler notre réseau, nous nous sommes inspirés du modèle expliqué par Nicky Case. Dans celui-ci, les personnes sont fortement liées aux personnes qui appartiennent aux mêmes sous-groupes ou nœuds et liées de manière plus occasionnelle à des personnes de groupes différents. On peut cependant noter que dans la réalité ces groupes existent bien mais sont souvent impliquée l'un dans l'autre et chaque personne appartient à plusieurs groupes ce que nous avons négligé.

A l'aide de fonctionnalités comme la publication ou les messages privés, une information publiée sur internet peut facilement devenir virale. Dans notre modèle nous admettrons qu'une personne possédant l'information a une certaine probabilité de l'envoyer à chacun de ces amis. De plus, chaque ami a une probabilité de recevoir ou voir l'information, et il pourra donc repartager l'information selon le même système à la génération suivante. Une génération représente donc une unité de temps arbitraire durant laquelle toutes les personnes possédant l'information et ne l'ayant pas encore répandue décident si elles la propagent ou non. Comme expliqué dans l'introduction, aucunes données à propos de ces probabilités ne sont disponibles car elles dépendent de beaucoup trop de facteurs. Pour toutes nos simulations nous n'avons donc pas utilisé des probabilités réelles, mais plus des estimations probables.

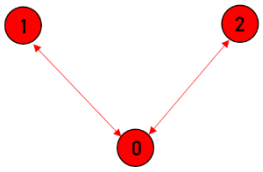
De plus, à cause de la manière dont notre code marche et des ordinateurs sur lesquels notre code tourne, nous pouvons au maximum simuler des groupes de l'ordre du millier de personnes, ce qui est très loin du nombre réel de personnes sur les réseaux sociaux. Nous avons donc décidé d'approximer avec des valeurs qui nous paraissaient réalistes par rapport à la quantité de personnes dans notre groupe (par exemple nous admettons que chaque personne est amis avec une personne sur 15 ou 20 ce qui ne serait pas du tout réaliste avec plus de 100/150 personnes). Finalement, notre modèle ne cherche pas à représenter fidèlement la réalité mais seulement à comprendre comment l'information se propage grâce à un modèle plus simple à analyser.

Avant d'avoir commencé nos recherches, nous avions déjà pensé à utiliser les matrices comme outil pour nos idées. Le principe est très simple : chaque personne du groupe est indexée, ce qui permet de créer une matrice de liaison, et une matrice de la ou les personnes ayant l'information. Le produit des deux matrices permet d'obtenir la matrice des personnes ayant l'information à la génération d'après.

La matrice présentant les agents porteurs de l'information (notée *info* dans nos fonctions) est très simple à concevoir. Pour une population de n personnes, la matrice sera une ligne, composée de 0 ou de 1 si la personne a ou non l'information. Par exemple, pour un groupe de 3 personnes, si seule la personne d'indice 0 a l'information, on obtient :

$$A = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix}$$

La matrice de liaison est légèrement plus complexe. C'est une matrice carrée de taille n, où chaque ligne permet d'identifier les amis d'une personne du groupe. On ajoute un 1 quand la personne est amie avec l'indice lui correspondant. Un exemple est plus parlant. Prenons un groupe trivial de 3 personnes, où seule une personne de ce groupe est amie avec les deux autres. On attribue à cette personne l'indice 0, et on a alors la matrice de liaison suivante :



$$B = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Maintenant, en multipliant la matrice A par B, on obtient la matrice C suivante : $C = \begin{pmatrix} 0 & 1 & 1 \end{pmatrix}$
Les deux amis de la personne 0 ont bien obtenu l'information.

Néanmoins, en Python le calcul matriciel n'est pas forcément simple, et nous n'avons donc pas pu utiliser cette méthode. Malgré tout, nous avons conservé le concept des matrices, que nous avons adaptées en langage de programmation.

Partie transmission simple (Yanis) :

Dès les premières heures, nous nous sommes attelés au développement d'un modèle très simplifié du passage d'une information dans un groupe. Le codage de ce programme s'est développé en 3 parties :

- Création d'un groupe et de sa matrice de liaison
- Passage de l'information dans ce groupe
- Visualisation graphique de l'évolution du nombre de personnes informées

a. Création d'un groupe aléatoire

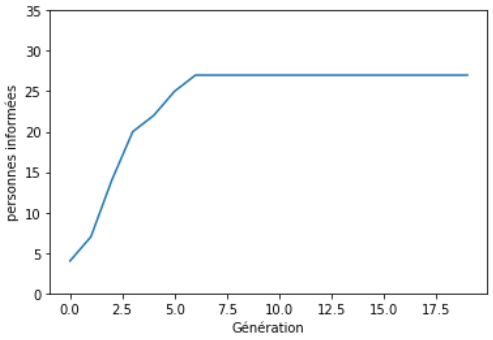
Pour cela, nous avons tout d'abord défini un nombre d'amis moyen du groupe, puis à l'aide d'une fonction simple, nous avons choisi pour chaque personne du groupe, avec qui cette personne est amie. Cette fonction, *choisir_amis*, nous retourne alors la liste des indices des amis de la personne. Ensuite, en réutilisant cette fonction, on créer pour chaque personne une liste indiquant les amis de cette dernière par un 1 ou un 0. La concaténation verticale de toutes ces listes nous forme la matrice de liaisons du groupe.

b. Passage de l'information

En utilisant la matrice de liaisons, nous avons construit une fonction permettant de transcrire le calcul matriciel expliqué ci-dessus, afin de passer l'information à la génération suivante. Cette fonction, basée sur le parcours de notre matrice de liaisons et de la liste *info*, sera la structure de tous nos modèles suivants. Il nous a été ensuite aisé d'étendre notre fonction de passage de l'info à n générations

c. Visualisation graphique

La matrice info indique le nombre de personnes venant d'être informée. En parcourant cette liste à chaque génération, nous avons pu construire une liste juxtaposant le nombre de personnes étant informées pour n générations. Le module MatPlotlib nous a ensuite permis de visualiser cette évolution.



Par la suite, tout notre travail s'est axé sur 3 grands vecteurs : la création d'un groupe plus ou moins complexe, le passage d'une information influencée par un nombre plus ou moins important de facteurs, et la représentation graphique du modèle.

Complexification du modèle : ajout des profils types

Afin de complexifier notre modèle d'un réseau social, nous avons pensé à introduire des profils types différents. En effet, sur un réseau social, tous les internautes n'ont pas les mêmes façons d'appréhender les informations et la même manière d'utiliser la plateforme. Evidemment, sur une population de n personnes Ces variations peuvent venir d'une multitude de facteurs, comme l'âge, le niveau de vie, le sexe … Même s'il est très réducteur de catégoriser des millions d'utilisateurs en 4 profils différents, nous avons décidé de commencer avec les profils suivants : **vieux, marginal, normal, et geek**. Chacun de ces profils présentent des caractéristiques spécifiques. On peut les retrouver dans le tableau récapitulatif en annexe.

Pour créer notre nouveau groupe d'ami, nous avons tout d'abord commencé par écrire une fonction, *listprofil,* qui créer une liste de n personnes avec leur profil, en tenant compte des proportions de chacun de ces profils au préalable. La liste rendue est de la forme suivante :

```
['marginal','marginal','geek','geek','classique','classique','marginal','classique
','classique','vieux']
```

Afin d'avoir un nombre entier de personnes quand l'on multiplie la proportion d'un profil avec le nombre de personnes totale, il a fallu arrondir le résultat, en utilisant l'opérateur round.

Pour générer la matrice de liaison, cela a commencé à se compliquer. Nous avons introduit un dictionnaire ProfilSpecs (ci-dessous) présentant l'ensemble des caractéristiques de chaque profil, reprenant les valeurs définies dans le tableau en annexe.

```
2 ProfilSpecs={"classique":(2/13,0.3,0.7),
3              "marginal":(1/15,0.1,0.4),
4              "vieux":(1/13,0.6,0.9),
5              "geek":(4/13,0.7,1)} #type: proportion d'amis, proba d'envoyer, proba de recevoir
```

En parcourant les indices pour créer les amis de chaque personne, on vérifie donc dans la liste créée auparavant avec *listprofil* quel type de profil est la personne, puis il faut aller vérifier la proportion d'amis qu'il doit obtenir. Ce programme reprend ensuite la structure du modèle sans profil pour obtenir la matrice de liaisons.

Enfin, la fonction *passage_info_simple_profils* permettant de passer l'information à la génération suivante est aussi très similaire au modèle simple, à la seule différence qu'il faut désormais tester la probabilité d'envoyer de la personne ainsi que la probabilité de recevoir de chaque ami en fonction de leur profil respectif.

Cette fonction retourne toujours la liste des personnes qui pourront transmettre l'information à la génération suivante (*info*), la liste regroupant le nombre de fois que chaque personne a reçu l'information (*stock_info*), ainsi que le nombre de personnes croyant à l'information (nbr).

A partir de ces résultats, on a pu de nouveau obtenir la liste de l'évolution du nombre de personne croyant à l'information et tracer la nouvelle courbe avec MatPlotlib.

En jouant sur les proportions de chaque type on a pu obtenir les cas critiques évidents :

- Un groupe composé majoritairement de Geek transmettra l'information en quelques générations
- Un groupe composé en majorité de Marginaux ne transmettra que très peu l'information voir pas du tout

Partie groupes (Gaëtan) :

Nous avons maintenant souhaité développer l'axe sur la complexification de nos groupes. Jusqu'ici, nous avons réussi à modéliser un réseau classique d'environ 100/200 personnes, cependant cela reste loin de la quantité présente sur les réseaux sociaux. De plus, les amis étaient choisis aléatoirement entre toutes les personnes tandis que, dans la réalité, les personnes sont plus souvent amies avec des personnes du même sous-groupes. La page web de Nicky Case "The wisdom and/or madness of crowd", simulant les interactions humaines dans des groupes, explique que les groupes sociaux peuvent être simplifiés en sous-groupes (ou nœud), dans lesquels les personnes sont très connectées, ainsi qu'en ponts, des liaisons plus rares entre des groupes. Nous avons donc établi que le réseau que nous avions créé serait constitué d'un seul nœud, nous allons donc en générer plusieurs en même temps puis les relier entre eux.

Tout d'abord nous avons créés une fonction *groupe_generator* qui nous permets de générer plusieurs groupes de tailles différentes. Nous pouvons donc accéder :

- aux matrices de liaisons de chaque groupe

- au nombre de personnes de chaque type par groupe

- à une liste du type de chaque personne (qui nous permettra de leur attribuer un numéro plus tard).

En effet, nous allons utiliser le fait que chaque personne est caractérisée par son type ainsi qu'un indice, qui le classe parmi les personnes de son type :

Par exemple dans la liste ['classique', 'marginal', 'vieux', 'geek', 'classique', 'classique', 'geek', 'marginal']

La 5ème personne sera « classique 1 » car il y avait un classique avant qui lui sera « classique 0 »

Une fois nos différents groupes générés, nous avons créé une fonction pour déterminer le nombre de liaisons entre les différents groupes. Pour cela nous avons établi, pour chaque type, la probabilité de faire une liaison avec quelqu'un d'un autre groupe. Nous parcourons donc les personnes de chaque groupe, après avoir déterminer leur type, puis nous créons un *random* et si celui tombe en dessous de la probabilité de liaisons ponts de ce type nous gardons l'information sur le type de la personne et le groupe dans lequel elle est. Finalement, nous obtenons un dictionnaire qui nous donne pour chaque type une liste du nombre de personne qui fait une liaison pont, pour chaque groupe.

Nous pouvons noter que nous allons créer une liaison pont pour chaque personne qui peux faire une liaison pont avec une personne aléatoirement choisie dans un autre groupe, c'est-à-dire que les personnes capables de faire une liaison pont dans 2 groupes distincts ne vont pas forcément se lier entre elles comme nous avions prévu au début.

Ensuite nous avons créé la fonction qui permet de déterminer quelle personne de chaque groupe fait les liaisons ponts et avec qui. En effet nous avions besoin de savoir qui faisait les liaisons avec qui afin de pouvoir faire passer l'information. Dans cette fonction nous avons utilisé deux autres fonctions que je vais décrire :

Type_liaisons : donne le type de la personne avec qui la personne fait une liaison (la probabilité de faire une liaison avec quelqu'un du même type

Choisir_groupe : détermine le groupe avec lequel va faire la personne fait la liaison

Notre fonction *liaisons_groupes* va donc parcourir les gens qui peuvent faires des liaisons ponts, pour chaque personne elle va déterminer son groupe et son numéro, ensuite pour chaque liaison que fait cette personne elle choisit un groupe et un type grâce au fonction précédente. De plus, nous allons déterminer aléatoirement le numéro de la personne en fonction des numéros disponibles évidemment. Finalement nous extrayons l'informations dans un dictionnaire.

J'ai ensuite créé une fonction permettant de faire passer l'information entre les groupes. Cependant, nous avons réalisé après l'écriture de la fonction qu'il était beaucoup plus simple de remettre les liaisons de groupes dans la même matrice. Je ne vais donc pas décrire la fonction *transmissions_info_groupes* et *transmissions_info_groupes_n* (qui propage l'information sur plusieurs générations) mais il y a des descriptions sur le code pour les curieux.

En revanche je vais décrire la fonction *grosse_matrice_liaison* : cette fonction va transformer un dictionnaire de matrice liaisons en un gros dictionnaire comprenant toutes les liaisons à l'intérieur des groupes (extrait du dictionnaire) mais aussi des liaisons entre groupes qu'on détermine grâce à la fonction *liaisons_groupes*.

Fake News et complotistes (Noah) :

Nous avons ensuite décidé de complexifier notre modèle au niveau de la diffusion de l'information en elle-même.

Tout d'abord, nous avons commencé par créer un nouveau profil qui se nomme « média », c'est le profil qui va pouvoir contredire la Fake News. Ce profil va avoir beaucoup d'amis avec une certaine chance de contredire l'information. Il va également avoir une grande probabilité de voir la Fake News mais une petite probabilité de la transmettre. Le « média » inclue également les influenceurs qui pourraient contredire l'information.

Pour cela, nous avons réutilisé la fonction *listprofil* précédemment créée. Cependant, il nous faut peu de *média* (environ 1 pour 100 personnes) donc nous avons dû rajouter une condition qui nous enlève un *classique* pour le remplacer par un *média* s'il n'y en a pas assez. Au final, nous voulons avoir le nombre personnes qui croient à la Fake News ainsi que le nombre de personnes qui croient à la contre-information. Pour cela nous avons créé la fonction *passage_info_profils_fake_news_n_generations*. Cette fonction est résumé en pseudo-code si dessous :

Tant que le média n'a reçu l'information :
    On transmet l'information simplement
Pour chaque média, on vérifie s'il la contredit

S'il l'a contredit :

On transmet l'information et la contre-information en parallèle

Sinon :

On continue de transmettre l'information simplement

On retourne une liste des personnes qu'on crue à l'information et ceux qui ont cru à la contre-information

La fonction qui transmet l'information et la contre-information en parallèle a été la plus compliqué à coder. A chaque génération, nous avons décidé de faire passer la contre-info avant l'information. Une hypothèse importante est que si une personne voit la contre-info, il la croit. Je vais également vous montrer cette fonction en pseudo-code :

Passer la contre-info

```
Pour chaque personne de la liste :
    S'il a la contre-info :
        Pour chaque personne :
            Si la personne n'a pas vu la Fake News et qu'il décide de l'envoyer :
                S'il a une liaison avec une personne (chaque amis) :
                    Si l'ami n'a jamais reçu la contre-info et qu'il la voit :
                        On note que l'ami l'a reçue
                        Si l'ami a déjà reçu la Fake News :
                            On note que l'ami ne croit plus la Fake News
                    Si l'ami a déjà reçu la contre-info et qu'il la voit :
                        On le note
            Si la personne qui envoie a déjà reçu la contre-info :
                    Alors sa probabilité d'envoyer augmente et
                    S'il a une liaison avec une personne (chaque amis) :
                        Si l'ami n'a jamais reçu la contre-info et qu'il la voit :
                            On note que l'ami l'a reçue
                        Si l'ami a déjà reçu la Fake News :
                            On note que l'ami ne croit plus la Fake News
                    Si l'ami a déjà reçu la contre-info et qu'il la voit :
                        On le note
On enlève les personnes qui ne croit plus à l'info
```

Passer la Fake News

```
Pour chaque personne de la liste :
    S'il a la Fake News :
        Si la personne n'a pas reçu la contre-info :
            Pour chacun de ses amis :
                S'il décide de l'envoyer :
                    Si l'ami n'a pas encore reçu la Fake News et qu'il la voit :
                        Si l'ami a déjà la contre-info :
                            On note simplement qu'il l'a reçu
                        Si l'ami n'a pas la contre-info :
                            On note que l'ami croit à l'info
                    Si l'ami a déjà reçu la Fake News :
                        On note qu'il l'a reçu une nouvelle fois
```

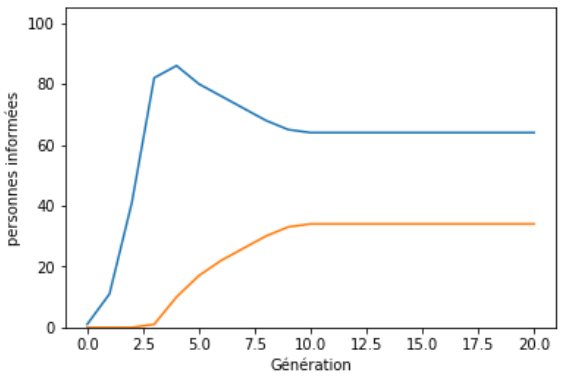
La fonction retourne la liste des nouvelles personnes qui ont reçu l'info/ la liste du nombre de fois qu'une personne a reçu l'info/ la liste des nouvelles personnes qui ont reçu la contre-info/ la liste du nombre de fois qu'une personne a reçu la contre-info/ le nombre de personne qui croit à la Fake news à cette génération/ le nombre de personne qui croit à la contre info.

Lors de la rédaction de cette fonction, nous avons rencontré un problème : nous voulions retirer les personnes qui ne croit plus à la Fake News de ceux qui y croient, cependant nous ne pouvions pas simplement retirer les personnes qui ont reçu la contre-info car il se peut que certaines personnes aient reçu la contre-info sans avoir vu la fake News. Nous avons donc dû définir une nouvelle variable qui va comptabiliser seulement ceux qui ont cru à la Fake News puis qui ont reçu la contre-info.

Complexification du modèle : ajout des complotistes

Pour complexifier le programme, nous avons rajouté un nouveau paramètre pour les profils : c'est le fait d'être un *complotiste*. Un *complotiste* est une personne qui ne va jamais croire à la contre-info ni la transmettre et qui va transmettre la Fake News même s'il a déjà reçu la contre-info. Son rôle est de répandre coûte que coûte la Fake News. Pour cela nous avons créé une nouvelle fonction qui va créer une liste en indiquant qui est un complotiste. Ensuite, dans la fonction lorsque nous faisons transmettre la contre-info, nous vérifions si la personne n'est pas un complotiste. Enfin, lors de la transmission de la Fake news, nous avons rajouté une boucle pour que les complotistes transmettent l'information quelles que soit les conditions.

Enfin, nous utilisons le module Matplotlib pour pouvoir tracer les courbes des personnes qui croient à la Fake News et ceux qui croient à la contre-info.



ProfilSpecs= {"classique":(10/100,0.3,0.7,0.16,0,0.2
"marginal":(8/100,0.1,0.4,0.06,0,0.2),
"vieux":(9/100,0.6,0.9,0.06,0,0.2),
"geek":(12/100,0.7,1,0.1,0,0.2),
"media":(30/100,0.2,0.8,1,1,0)}

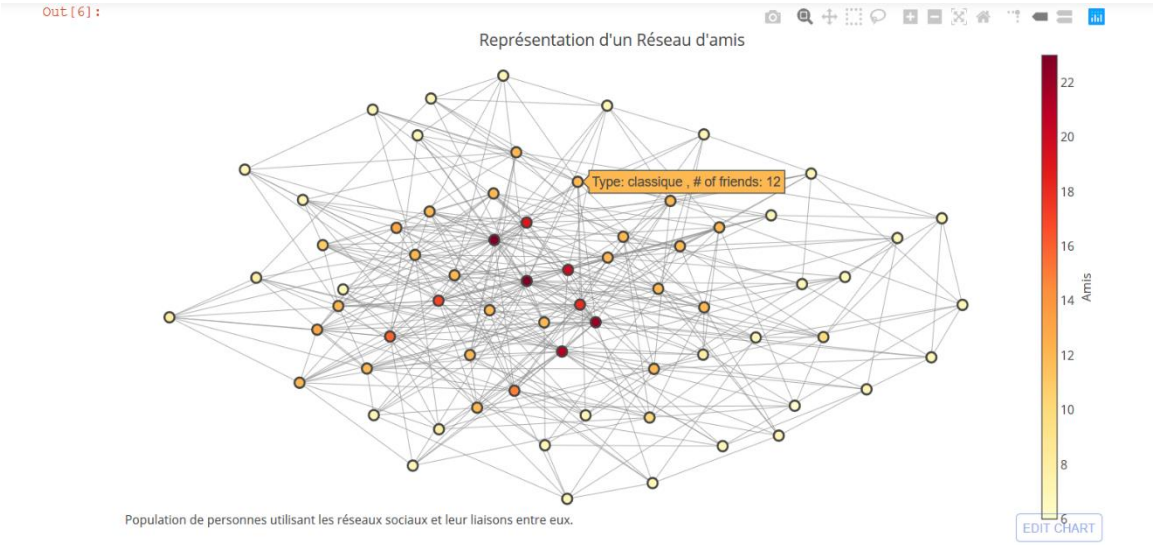
Profiltypes= {"classique" :0.39,"marginal":0.25,"vieux":0.15,"geek":0.2,"media":0.01}

Partie Graphique (Thomas) :

Afin d'avoir un rendu final plus esthétique qu'avec le simple module Networkx, nous avons décidé d'utiliser la bibliothèque de Plotly pour afficher les réseaux de personnes.

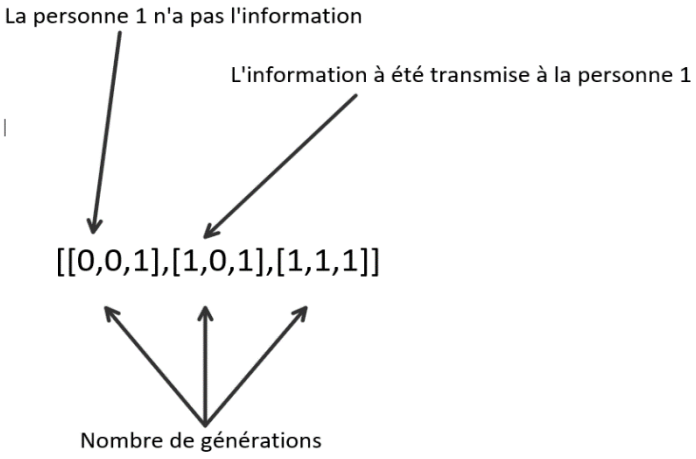
Plotly est une librairie complète et il m'a fallu quelques heures de lecture et de tests afin de comprendre son fonctionnement. J'ai en premier lieu essayer de tracer les liaisons entre les personnes dans le modèle simple avec profils. Cela fut déjà assez compliqué car, au lieu d'avoir un programme de 10 lignes comme pour tracer le même graphique avec Networkx, je me retrouvais avec plus de 50 lignes pour avoir le premier résultat convaincant.

Plotly offre plusieurs possibilités dans ses graphiques, notamment le fait de pouvoir afficher des « labels » (un texte s'affiche à l'écran quand on passe sa souris sur certains éléments tracés) dont je me suis servi pour afficher le type de chaque personne et le nombre d'amis qu'elle avait. En plus de cette fonctionnalité, Plotly permet de zoomer sur le graphique ou encore d'afficher seulement une partie du graphique. Voici un aperçu d'un tracé de graphique :



Le résultat est assez convaincant car on peut tout de suite distinguer que les personnes ayant le moins d'amis (en beige/blanc) se trouve en marge du graphique alors que les personnes ayant beaucoup d'amis comme les geeks sont concentrés au centre.

L'étape suivante fut de créer des GIF pour visualiser la propagation d'information au sein d'un groupe. Pour cela, j'ai dû créer d'autres programmes annexes afin de récupérer une liste de liste avec des 0 et de 1 organisée comme ci-dessous :



Grâce à cette liste j'ai pu colorer en rouge les points ayant l'information et laisser en blanc les ignorants. Pour créer le GIF, j'ai donc récupéré l'image du graphique avec les points colorés à chaque génération (à l'aide de Plotly). Il m'a ensuite suffi de regrouper toutes ces images dans un fichier GIF. Pour cela, j'ai utilisé les bibliothèque Imageio et Orca. Ce n'était globalement pas très compliqué, le plus dur étant de bien comprendre comment fonctionnent les différents modules pour les adapter à notre programme.

Annexe :

Tableau récapitulatif des caractéristiques des profils (définies et fixes)

	PROFILS				
	Geek	**Vieux**	**Marginal**	**Normal**	**Média**
Probabilité d'envoyer l'info	très élevée	élevée	très faible	faible	très élevée
Porbabilité de recevoir l'information	très élevée	très élevée	faible	moyenne	très élevée
Probabilité de faire des liaisons avec d'autres groupes d'amis	élevée	très faible	faible	faible	élevée
Proportion d'amis	élevée	moyenne	faible	moyenne	élevée
Probabilité d'envoyer la contre-info	moyenne	très faible	très faible	faible	très élevée

Remarque : les paliers (faible, élevée…) sont relatifs à chaque ligne. Par exemple, une probabilité élevée d'envoyer la contre-info est beaucoup plus faible qu'une probabilité élevée d'envoyer l'information (Fake News).

Tableau regroupant les paramètres variables

PARAMETRES VARIABLES
Proportion de chaque profil dans la population
Population totale
Proportion de complotistes
Profil lanceur de la Fake News

Ci-joint le tableau Excel reprenant l'ensemble de nos tests avec les interprétations. Nous avons testé l'impact de la variation de nos paramètres variables pour obtenir les cas critiques de notre modèle. La population totale n'a aucun impact.

Le document Excel est en copie dans le dossier sur GitHub

Conclusion

Durant notre projet, nous avons réussi à modéliser des réseaux sociaux ainsi qu'une dispersion d'informations comme voulu. Bien que nous n'ayons pas réussi à mélanger les deux modèles par faute de temps, nous avons atteint beaucoup d'objectifs. Tout d'abord nous avons montré que, malgré une contre-information qui dément la Fake News et dans certains cas, elle peut quand même devenir virale.

Néanmoins, ce modèle reste très élémentaire et n'est que peu représentatif de la réalité. Pour complexifier encore plus notre modèle final, nous avions pensé aux améliorations suivantes :

- Catégoriser les informations en différents types (politiques, divertissement, actualités …) et attribuer des probabilités d'intéressement pour chaque type d'informations à chaque profil
- Décliner les liens entre amis en liens plus ou moins fort (famille, collègue, ami proche)
- Prévoir le passage de plusieurs informations, avec un taux de crédibilité de chaque personne mémorisée par ses amis, en fonction de la véracité des informations partagées au préalable

En termes de travail de groupe, nous avons appris à collaborer en partageant nos points de vue, en choisissant des modèles qui nous convenaient à tous, ou encore en se répartissant des rôles en fonction de ce qui nous intéressait. Concernant les compétences acquises, nous avons tous gagné de l'aisance avec Python, notamment Networkx pour modéliser nos réseaux puis construire des GIF's. Pareillement, nous avons appris à utiliser Git Hub (notamment à soumettre notre travail avec des lignes de commandes).

Bien que nous nous soyons répartis le travail pour la rédaction du rapport, nous tenions à préciser que nous avons tous contribuer au développement du modèle dans son intégralité, sans réellement se définir des tâches précises.