Ezriel Steinberg
Nathaniel Hayoun
IED – Paris 8

Analyse linguistique de comptes tweeters

Rapport d'élaboration

Partie 1 – Ezriel Steinberg

J'ai décidé de mon coté d'utiliser python3, qui me permet d'utiliser plusieurs bibliothèques utiles. Cela permet aussi de « pickle/load » simplement les données dans un fichier afin de les transmettre.

Comme outils, j'ai tout simplement utilisé sublime text 3.

Du côté de l'algorithme, la majorité est du code entré dans l'interpréteur, assez simple et intuitif.

Au niveau des lib, voici ce que j'ai utilisé :

- pickle, sys, re tout simplement.
- argparse, qui permet de gérer facilement les arguments du programme.
- p_tqdm, qui offre des fonctions qui « map » des fonctions sur des listes, de manière concurrentielle, avec en prime une apparence sympathique de barre de chargement.
- got3 pour « Get Old Tweets », qui permet de télécharger des tweets en parsant les pages webs. A noter que j'ai modifié le code de celui-ci, car il y avait quelques erreurs, notamment au niveau du texte renvoyé lorsqu'il y avait une image dans un tweet, où lors de l'utilisation d'hashtags. Je n'ai pas encore pris le temps de proposer une correction sur Git-Hub. J'utilise donc le code modifié, que j'ai commenté afin de souligner les différences que j'ai introduite. A noter que modifié pour modifié, j'ai aussi enlevé des données lors de la récupération de tweets qui ne sont pas intéressantes chez nous (tous les méta-paramètres, nombre de retweets, de likes etc.), afin d'accélérer le chargement.
- french_lefff_lemmatizer qui est comme son nom l'indique un lemmatizer pour le français. A noter que j'utilise le nom commun en priorité (s'il existe), puis le verbe, puis l'adjectif, puis le mot lui même. Peut-être une approche stochastique serait elle plus intéressante ? D'un autre coté, les nom et verbes ou adjectifs qui sont semblable sont souvent proches syntaxiquement (ex : « marche » peut être « je marche » ou « en marche », cela ne change pas grand-chose), les cas extrêmes (ex : « avions » peut être : « des avion**s** » ou « nous av**ions** ») sont plus rare.

J'avais aussi à trouver les comptes Twitter intéressants à étudier. Ma méthode à été celle-ci : tout d'abord, voir les comptes des personnes connues, (candidats à la présidentielle, députés, maires etc.) J'ai observé à quelles listes elles appartiennent, en particulier les listes qui compte le maximum de politiques. J'ai récupéré tous les noms de toutes ces listes avec un simple Ctrl-A Ctrl-C Ctrl-V, puis ai utilisé la fonction rechercher-remplacer avec des regex, de sublime. J'ai donc une liste de compte, mais trop longue. J'ai donc créé un compte de développeur Twitter afin de faire des accès à l'API. J'ai utilisé le code python ci-dessous pour trier les comptes dans l'ordre du nombre de followers, en supposant que les plus suivis sont les plus intéressants.

```
import tweepy
auth = tweepy.OAuthHandler("_", "_")
auth.set_access_token("_", "_")
api = tweepy.API(auth)

def getFollNumb(u):
      try:
            return api.get_user(u).followers_count
      except:
            return -1

srted = sorted(users, key=getFollNumb)
```

En mettant bien sûr les ids d'accès et d'application et les codes aux emplacements ad hoc.
En est sortie la liste, d'où est le fichier « users.txt » qu'on utilise.

L'output se fait sous la forme d'un dictionnaire. Le niveau le plus élevé contient deux entrées : 'allWords' et 'users'. Le premier est lui même un dictionnaire, qui contient en clés tous les mots utilisés, et en valeur le nombre de fois où ils l'ont été. Cela permet une analyse rapide des mots trop utilisés, et ceux qui ne le sont pas assez.

Le second est un dictionnaire, qui à un compte twitter associe un dictionnaire, qui à un mot associe le nombre de fois que ce compte l'a utilisé dans notre échantillon.

Le tout est dumpé dans le fichier « output » donné en argument.

Partie 2 – Nathaniel Hayoun

Ma partie a été réalisé en python3.5, avec l'éditeur sublimeText. Les principaux outils externes utilisés sont :

- Le module python networkx, disponible via leur site https://networkx.github.io/, qui m'a permis de créer et tracer efficacement des graphes en python, sans devoir rentrer dans des casses-têtes de bibliothèque graphiques, ce qui ne m'intéressait pas outre-mesure dans le cadre de ce projet. Ce module en utilise lui-même un autre, plus répandu :

- Le module mathplotlib, disponible à l'adresse https://matplotlib.org/

- Le module argparse, disponible nativement avec python3, qui m'a permis de parser efficacement les arguments en ligne de commande pour que le programme soit facilement paramétrable.

La construction de ce programme ne fut pas trop compliqué, principalement parce que j'ai pris le temps de structurer son organisation avant d'écrire la moindre ligne de code. Je n'ai pas eu la nécessité d'utiliser un debugger externe, mais le programme contient une variable TEST_MODE, qui, quand elle vaut *True*, affiche toutes les informations qui peuvent être utiles pour du débogage (mots inclus dans la stoplist, liste des noms de compte analysés, leur nombre, etc.). Cette variable est maintenant paramétrable grâce à l'option -v, à l'appel du programme. La construction de mon programme obéit au schéma suivant, où sont surlignés en vert les fonctions du code :

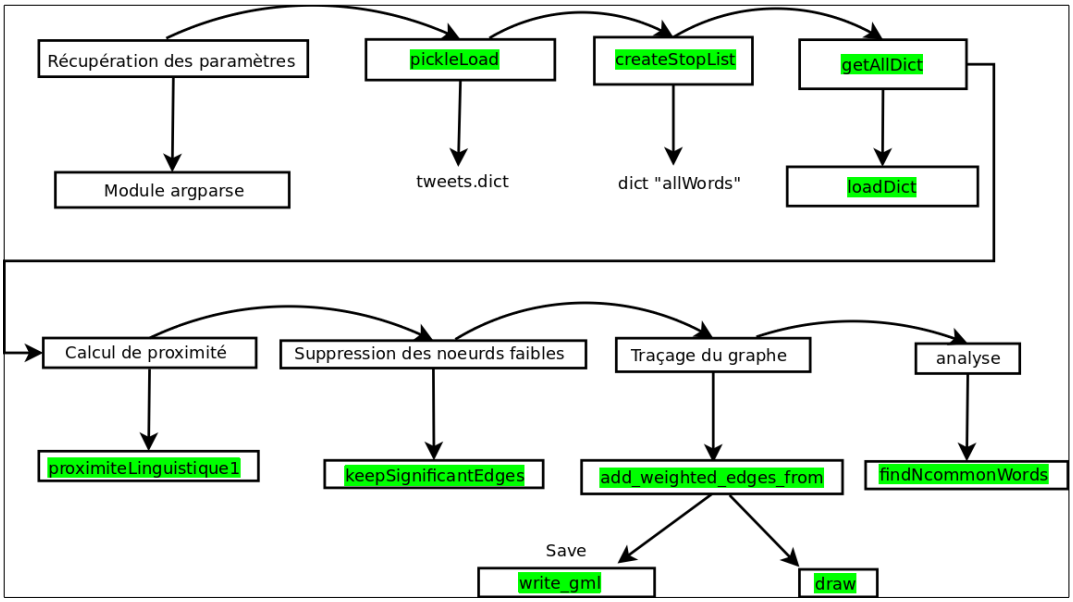


Illustration 1: Organisation du programme analyse.py

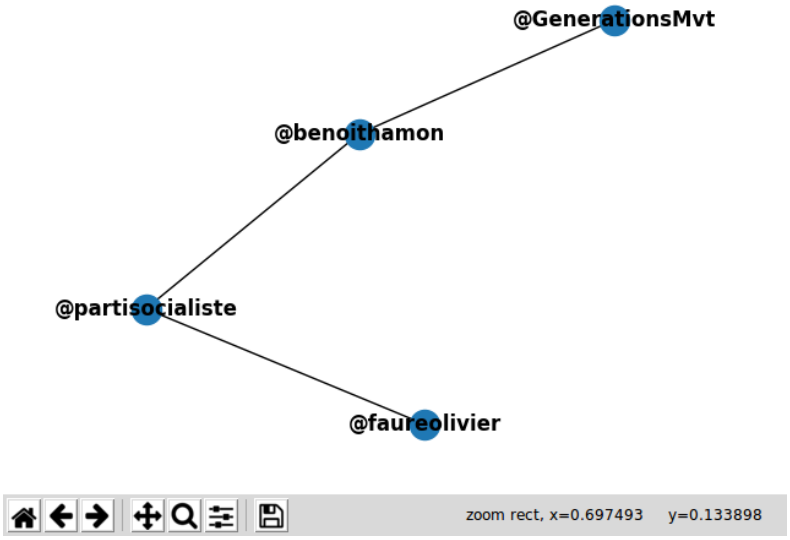
Les points intéressants d'un point de vue théorique furent les paramètres d'analyse linguistique, tels que la taille de la stopList (correspondant, dans mon programme, au nombre des mots les plus courants que l'on ne prend pas en compte), le nombre d'occurrences minimales d'un mot pour que celui-ci soit pris en compte, et enfin la fonction de calcul de proximité entre deux auteurs.

Concernant les deux premières variables, quelques tâtonnements m'ont permis de modifier peu à peu leur valeur pour trouver des résultats intéressants (il fallait supprimer en réalité bien plus de mots que ce que je pouvais imaginer), mais elles restent paramétrables à l'appel du programme, afin que l'utilisateur finale puisse tester d'autres valeurs.

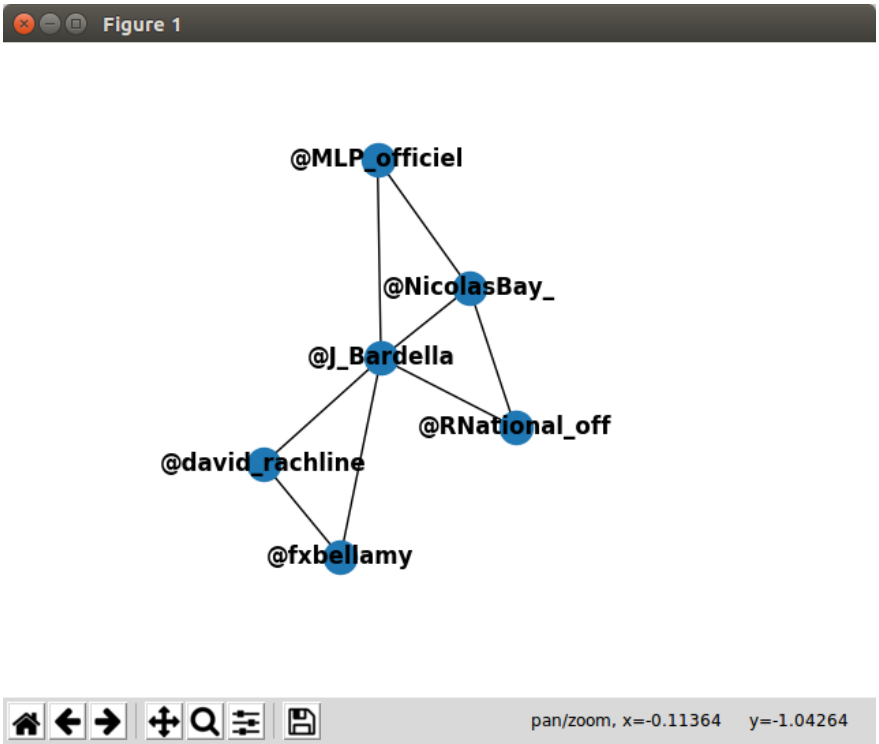
La solution que j'ai utilisé pour la fonction de proximité consiste à calculer la somme du minimum de la fréquence d'un mot donné entre deux utilisateurs, pour tous les mots utilisés par lesdits utilisateurs. Une autre méthode consistait à multiplier les fréquences de chaque occurrence de mots, mais, sans que je sache bien déterminer pourquoi, je ne retrouvais pas ainsi de *clusters* comme la première méthode me permettait de le faire.

Enfin, un des problèmes majeurs du module python networkx que j'ai utilisé, est qu'il ne permet pas de visualiser la *force* de deux liens, qu'il affiche simplement comme deux ligne reliant des personnages. J'ai dû donc créer la fonction *keepSignificantEdge*, qui permet de supprimer tous les liens qui ne dépassent pas un certain seuil. J'ai heuristiquement déterminé ce seuil par défaut à 0,331, mais, ici aussi, l'utilisateur peut faire varier sa valeur. Le graphe sauvé dans le fichier, contient lui *tous* les liens, même les plus faibles, qui peuvent tout de même s'avérer intéressant, dans la mesure où certains comptes tweeter ne sont liés que faiblement à d'autres.

D'un point de vue de méthodologie heuristique, j'ai fait varier les valeurs de mes paramètres de manière à retrouver les résultats que je m'attendais à voir (des clusters selon les diverses orientations politiques). Exemple, si je zoom sur la gauche de Benoit Hamon :



On peut critiquer cette méthode en avançant que je ne fais finalement que trouver les résultats que je veux trouver. Mais en réalité, en affichant les mots les plus communs des différentes arêtes du graphe, j'ai pu heureusement constater que je retrouvais des mots pertinents qui faisaient le lien entre les différents comptes. Par exemple, lorsque j'analysais le cluster lié au compte de Marine Le



Pen et Jordan Bardella, illustrée dans la figure suivante (je cherchais en réalité à comprendre pourquoi Nicolas Bay se retrouvait dans ce groupe) j'obtenais la liste de mots :

```
>>> findNcommonWords(["@NicolasBay_", "@MLP_officiel",
"@J_Bardella", "@Rnational_off"], 8)

emmanuel : 7.594409077242837
```

```
pays : 7.582792900407494
liste : 7.564011547590011
somme : 7.249463033625702
peuple : 7.081924640450012
seul : 6.336298060371602
union : 6.273054110155755
immigration : 5.92042774089543
```

Ces comptes sont les seuls à utiliser aussi massivement le mot emmanuel, pays, peuple, union ou immigration, caractéristiques d'une certaine manière de communiquer associée au Rassemblement National et de certaines thématiques politiques et/ou idéologiques récurrentes.

Partie 3 – Travail en commun

L'idée de ce projet revient à Ezriel, qui avait déjà utilisé l'API tweeter dans le cadre d'un autre projet, et en avait donc une certaine maîtrise.

Nous devions donc collaborer de manière à ce que je (Nathaniel), récupère les donnés des comptes téléchargés par Ezriel et les analyse. Au départ, ces comptes étaient téléchargés une seule fois sous la forme de fichiers dictionnaires individuels, mais nous avons convenu par la suite qu'il était plus intéressant de créer un programme qui, pour une liste de compte donnée, en télécharge un certain nombre de Tweets et les stocke dans un unique fichier dictionnaire. C'est ce dont s'est occupé Ezriel, et j'ai donc modifié ma partie (assez modulaire pour que cela se fasse rapidement) pour prendre en compte ce changement. Mis à part ça, puisque nos deux parties étaient très distinctes, nous n'avons pas eu de problèmes ni de nécessités de communication particulières.

Le fait d'utiliser python avec pickle a simplifié les choses, nous n'avons en effet pas besoin de renvoyer des fichiers textes mais directement des dictionnaires.