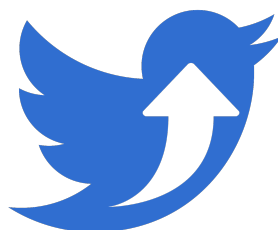




FACULTÉ DES
SCIENCES

RAPPORT DE TRAVAIL D'ÉTUDE ET DE RECHERCHE

Analyse de Tweets



Réalisé par
MOURAD INAN
CLÉMENT TASSART
SOLAL VERNIER

Sous la direction de
PASCAL PONCELET

ANNÉE UNIVERSITAIRE 2017-2018

Table des matières

1	Introduction	1
2	Présentation du contexte du projet	2
3	Analyse du projet	4
3.1	Analyse des besoins	4
3.2	Choix des outils	5
4	Partie Technique	6
4.1	Récupération	6
4.1.1	Tweets passés	6
4.1.2	Tweets en direct	7
4.2	Stockage	10
4.3	Visualisation	12
4.3.1	Nuage de Mots	12
4.3.2	Histogramme	12
4.3.3	Autres visuels	13
4.3.4	Affichage des tweets par intervalle de temps	13
5	Organisation	14
5.1	Répartition des tâches	14
5.2	Communication	14
5.3	Programmer ensemble	14
	Références	30

1 Introduction

Le réseau social *Twitter*, à l'instar d'autres réseaux sociaux, a permis, depuis son essor au début de cette décennie, de faciliter la création et l'échange d'informations entre individus. Par l'envoi de courts messages textuels et instantanés appelés *tweets*, les utilisateurs de ce réseau diffusent donc énormément de données, et de *métadonnées*, et de part le concept même de celui-ci, de manière immédiate. Ainsi, tout ou presque fonctionne en direct.

Pour les acteurs de la *fouille de données*, aussi connu sous l'anglicisme de *data mining*, Twitter est devenu un lieu idéal pour suivre des tendances, des actualités ou tout sujet qui puisse être intéressant à travailler.

Durant ce travail d'étude et de recherche de fin de Licence Informatique, nous avons pour objectif de réaliser une application web aidant à l'analyse de ces courts messages, les tweets. En premier lieu, l'application devra être capable d'en récupérer automatiquement un certain nombre, filtrés au préalable par l'utilisateur, puis de sauvegarder ceux-ci. En deuxième lieu, elle devra être capable de fournir à celui-ci des visuels permettant de mettre en exergue certaines données et métadonnées qui seront utiles à l'analyse de ces mêmes tweets.

Dans un premier temps, nous présenterons plus en détail le problème sur lequel nous nous sommes penchés lors de ce projet, ainsi que les besoins liés à un tel sujet et nous nous attarderons sur une analyse de l'existant. Dans un second temps, nous verrons comment la solution au besoin a été mise en place dans sa phase de réalisation. Enfin, nous terminerons sur une revue de notre organisation, sur ce que nous a apporté un tel projet et nous conclurons de manière plus large.

2 Présentation du contexte du projet

Aujourd'hui, le nombre de données que nous, utilisateurs, laissons sur internet, ne cesse d'augmenter. Il y a plus de 350 000 tweets par minute dans le monde. Et, plus il y a de données sur internet, plus il y a de questions qui se posent. Afin d'élaborer des réponses à ces questions, il faut examiner et exploiter ces données.

L'analyse des données se fait en plusieurs étapes,

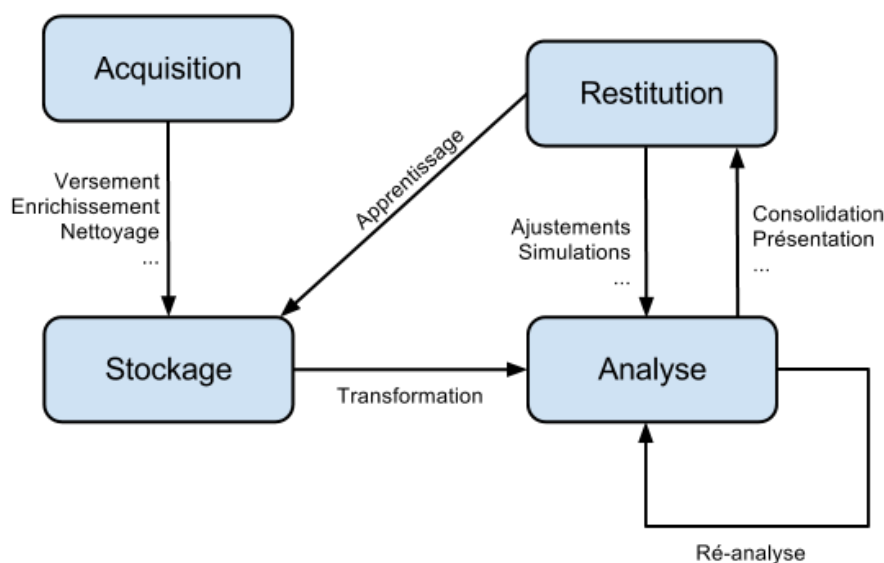


FIG. 1 – *Etapes d'une analyse des données*

Nous allons, dans ce projet, développer la partie sur l'acquisition et le stockage pour que l'utilisateur puisse, avec les données nettoyées, faire des analyses plus poussées.

Premièrement, l'acquisition des données. Ces données là peuvent être rassemblées via, par exemple, des achats des utilisateurs sur des plates-formes web ou encore via des partages de données via internet. Cependant, il y a tout de même une limite à la récolte de ces données. En effet, il existe des encadrements qui interdisent les entreprises d'exploiter certaines données des clients.

Ensuite, vient la récupération de ces données. C'est ici que l'analyste doit choisir les données qui répondent aux questions posées. Dans la plupart des cas, on choisit de stocker ces données dans une base de données. En effet, avoir une base de données facilite beaucoup les prochaines étapes. Il faut que l'analyste

trie les attributs de ces données pour ne garder que l'essentiel dans sa base de données.

Après avoir stocké les données qu'il a récupéré, l'analyste doit pouvoir illustrer ces données qui sont pour l'entreprise de simples données non verbales. Pour cela, il y a par exemple les graphiques. Cela peut aller de simples courbes aux diagrammes explicites où se trouve clairement les réponses aux questions posées. Ces graphiques permettent aux personnes extérieures au domaine informatique de comprendre l'analyse et ainsi avoir des critiques constructives.

Enfin, grâce aux visuels que l'analyste aura développés, la création des modèles et l'apprentissage de l'algorithme peut commencer. On parle du *machine learning*. Ici, l'analyste devra répondre à des problèmes précis à l'aide de son programme qui devra être complet : tous les cas possibles devront être connus par le programme, et ceci est possible grâce à des jeux de données que l'analyste devra tester sur son programme. Plus le jeu de données est complet plus le programme aura des réponses précises et moins il y aura d'erreurs de précision.

Voilà les différentes étapes de l'analyse des données. Après avoir étudié la théorie de cette analyse, passons à la pratique avec l'analyse technique. Comme dit plus haut, notre projet consistait à préparer les données pour que le data-analyste puisse commencer l'apprentissage de la machine.

3 Analyse du projet

3.1 Analyse des besoins

Lors de notre première réunion avec Pascal PONCELET, le 18 janvier 2018, nous avons pu prendre connaissance du sujet et des différentes attentes. L'idée est de proposer une plateforme d'analyse et de statistiques de tweets. Notre projet étant de créer intégralement une nouvelle application, nous avons jugé important de prendre le temps de poser les bonnes questions afin d'avoir une visualisation parfaite de ce qui était attendu.

Les tweets sont récupérés selon des critères spécifiques que l'utilisateur aura imposés. En effet, l'utilisateur peut récupérer des tweets en appliquant un ou plusieurs filtres : par mots-clés, par utilisateur twitter, par langue ou encore par géolocalisation.

Il doit être possible de traiter des tweets en temps réel mais aussi des tweets passés, il faut donc offrir à l'utilisateur différentes interfaces en adéquation avec ses besoins.

Après avoir extrait les tweets, il faut pouvoir découvrir ce qu'ils contiennent, c'est pourquoi l'utilisateur souhaite avoir accès à des graphiques, diagrammes et nuages de mots. Grâce à ces indicateurs, il pourra prendre connaissance d'un grand nombre de données statistiques et poursuivre cette étude sur le long terme.

De plus, il est important de pouvoir sauvegarder ces analyses. C'est pourquoi nous avons dû mettre en place un système de session, permettant à l'utilisateur de revenir sur des opérations effectuées dans le passé. De ce fait, chaque récupération de tweets sera associée à une session unique : il faudra remplir un formulaire de création de session, dans lequel sera renseigné les filtres apportés aux tweets. On doit également pouvoir importer et exporter une session afin d'échanger nos travaux avec d'autres utilisateurs.

Tout au long de ce projet, nous avons la liberté de prendre des initiatives concernant l'ajout de nouvelles fonctionnalités visant à perfectionner l'application. Les réunions à fréquence de deux semaines nous ont permis de faire part de nos avancées à Pascal PONCELET et ainsi obtenir des retours réguliers sur les choix que nous prenions.

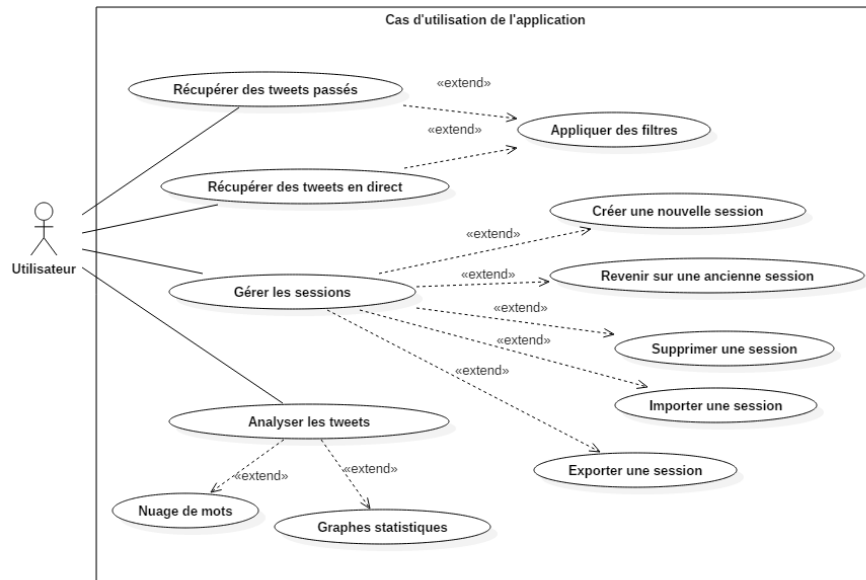


FIG. 2 – Diagramme de cas d'utilisation de notre application

3.2 Choix des outils

Concernant la récupération des tweets, nous avons été invités à utiliser une bibliothèque développée en *Python*. En effet, il en existe de nombreuses permettant d'utiliser l'API de Twitter. Nous avons pris la décision d'utiliser la bibliothèque *Tweepy* car elle possédait une bonne documentation et semblait répondre à l'ensemble des besoins.

L'ensemble des données de l'application seront stockées dans une base *MongoDB*. Aucun d'entre nous n'avait utilisé de base *NoSQL*, architecturée sous forme de collections dans lesquelles les objets sont des documents. Cette architecture nous permettait de stocker et manipuler facilement les objets *json* renvoyés par l'API de Twitter.

La partie visualisation a été facilitée grâce à la bibliothèque *D3.js*. Cette bibliothèque graphique *Javascript* permet l'affichage de données numériques sous une forme graphique et dynamique.

Enfin, l'architecture du site web sera également codée en *Python* grâce au microframework *Flask*. Nous avons hésité avec le framework *Django* mais celui-ci était plus compliqué à prendre en main. De plus, *Flask* a l'avantage de proposer une extension permettant d'interagir avec une base *MongoDB*.

4 Partie Technique

4.1 Récupération

Comme dit précédemment, une grande partie de notre application réside dans la récupération des tweets. Ces derniers doivent être choisis selon les filtres appliqués par l'utilisateur.

D'après les besoins de notre encadrant, nous avons mis en place deux types de récupération :

- **Les tweets passés** : l'ensemble des tweets publiés dans un intervalle de temps donné.
- **Les tweets en direct** : l'ensemble des tweets postés à partir du lancement du processus de récupération. C'est à l'utilisateur de définir lorsqu'il souhaite arrêter ce flux d'écoute.

Nous avons décidé de créer le dossier `./retrieve_tweets`, consacré à la récupération de l'ensemble des objets `.json` renvoyés par *Tweepy*. Dans cette partie, nous nous placerons dans le fichier `./retrieve_tweets/filters.py`, plus particulièrement au niveau de la fonction `filter(keywords, geocode, dates, user, language)` (cf. *Annexe A*).

Cette fonction traitera l'ensemble des types de récupération, listés précédemment. Elle sera invoquée une fois que l'utilisateur aura rempli le formulaire de création de la session (cf. *Annexe B et C*).

Nous utiliserons aussi, puisque nous avons besoin de récupérer des tweets par localisation géographique, la bibliothèque Javascript *Leaflet*, et particulièrement la bibliothèque *leaflet-areaselect* qui offre aux utilisateurs la possibilité de sélectionner une zone géographique sur une carte du monde (par l'intermédiaire d'*OpenStreetMap*, alternative open-source de *Google Maps*). La zone sélectionnée par l'utilisateur sera de la forme `"-74,2589, 40,4774, -73,7004, 40,9176"` pour l'application, qui correspond respectivement à la latitude sud, la latitude nord, la longitude ouest et la longitude est. On appelle cela une boîte ajustée (*bounding box* en anglais). Elle nous sera primordiale pour la suite.

4.1.1 Tweets passés

Voici la partie de la fonction `filter()` qui récupère les tweets passés :

```
32 query = keywords
33 if startdate != "" and stopdate != "":
34     query = query + " since:" + startdate + " until:" + stopdate
35 if user != "":
36     query = query + " from:" + user
37 if language == "fr":
38     language = "fr"
39 for tweet in tweepy.Cursor(api.search, q=query, tweet_mode="extended", geocode=geocode, lang=language).items(int(tweets_batch)):
40     stock_tweets(tweet)
```

FIG. 3 – Récupération des tweets passés

Les lignes 32 à 36 permettent de préparer la variable *query* dont on expliquera le rôle dans le paragraphe suivant.

L'appel vers l'API Twitter se fait par l'intermédiaire de la fonction *Cursor()* de notre objet *tweepy*. Cette fonction prend plusieurs paramètres :

- *api.search* : correspond au noeud *"/search"* de l'API Twitter.
- *query* : la chaîne de caractères contenant les informations suivantes :
 - les mots-clés.
 - la plage de dates dans laquelle les tweets ont été postés.
 - l'auteur des tweets récupérés (tout le monde par défaut).

Exemple : "hello since:2018-03-21 until:2018-03-22 from:@realDonaldTrump"

Ici, on récupérera tous les tweets postés entre le 21 et le 22 mars 2018 par le compte de Donald Trump, contenant le mot-clé "Hello".

- *tweet_mode* : reçoit la valeur *extended* car on souhaite, pour chaque tweet, récupérer l'intégralité du texte du tweet. Par défaut, l'API ne retourne qu'un extrait du texte, ce qui est insuffisant à nos besoins.
- *geocode* : la position géographique où les tweets ont été publiés.
- *lang* : la langue dans laquelle le tweet a été rédigé.

Exemple : "fr" pour les tweets en français, "en" pour ceux en anglais etc...

Il est également nécessaire de renseigner le nombre maximum de tweets à récupérer à travers la fonction *items(nbTweets_à_récupérer)*. Ici, on en récupère un nombre entier variable, que l'utilisateur aura renseigné dans le formulaire de création de session. Le résultat renvoyé par cet appel est un tableau de *status_object* : des objets au format *.json* contenant l'ensemble des informations de chaque tweet. On itère sur ce tableau, pour chacun des tweets, on appelle la fonction *stock_tweets(status_object)* que l'on explicitera dans la partie consacrée au stockage.

4.1.2 Tweets en direct

L'objectif de cette partie est de répondre aux besoins suivants :

1. Après avoir créé sa session, l'utilisateur clique sur un bouton afin de lancer le processus de récupération.
2. Dès lors, un flux d'écoute est démarré. On stock chaque tweet répondant aux filtres de l'utilisateur.
3. L'utilisateur peut arrêter le *stream* en cliquant sur un bouton. Le flux est alors terminé.

Ci-dessous, la partie de la fonction *filter()* qui récupère les tweets en direct :

```

global stream_stop
stream_stop = False

if user is not None:
    user = getIdByUser(user)

stream_o = tweepy.Stream(auth=api.auth, listener=Stream())
stream_o.filter(locations=geocode, track=[keywords], languages=[language], follow=[user])

```

FIG. 4 – *Récupération des tweets en direct*

Ce type de récupération nous semblait être une fonctionnalité plutôt compliquée à implémenter aux premiers abords. En effet, il fallait gérer le fait qu’énormément de tweets allaient être récupérés par l’application (en direct) et qu’il fallait les stocker aussitôt, un par un ; on se retrouvait rapidement avec un “décalage” : après avoir lancé une récupération d’une centaine de tweet, même après arrêté le stream, des dizaines voire des centaines de tweets venaient se stocker dans notre base de données. Pour faire plus simple, il fallait faire en sorte que cette récupération se fasse en multitâche. Par défaut, le framework *Flask*, qui est la base de notre application, est en mono processus. Il suffisait alors de la configurer en mode *multi-threadé*.

De plus, l’objet *tweepy* utilisé pour la récupération de tweets passés, possède une fonction *Stream()* prévue pour cela.

Cette dernière nécessite deux paramètres :

- *api.auth* : les clés d’authentification à l’API de Twitter.
- *listener* : une instance de la classe *Stream*. Cette classe est fournie par *tweepy* et nous allons surcharger la méthode *on_status()* de cette même classe :
 - Cette fonction sera appelée dès qu’un tweet répondant à certains critères (les filtres) est posté. En réalité, il se peut que des tweets ne soient pas “entendus” par le *listener* (contrainte liée à Twitter directement), mais on sait qu’une très grande majorité l’est.
 - Pour arrêter le stream, il suffit de retourner *False* dans cette même fonction ; sinon il boucle à l’infini.

L’objet retourné par *tweepy.Stream()* est un objet où l’on va pouvoir spécifier les filtres voulus par l’utilisateur lors du formulaire via la fonction *filter()* :

- *locations* : on renseigne la *bounding box* (variable *geocode*) préparée par la bibliothèque *Leaflet*.
 - On peut voir que la variable *geocode* est réécrite dès le début de la fonction. En effet, on passe effectivement une chaîne de caractères à la fonction, juste après le formulaire ; il faut convertir cette chaîne en un tableau de floats pour *tweepy*.
- *track* : les mots-clés.
- *languages* : la langue dans laquelle les tweets sont écrits.
- *follow* : les utilisateurs (donc tous les tweets de ceux-ci).

Une fois fait, le lancement se déroule bien, mais le stream ne peut pas être arrêté par l'utilisateur. Il existe sûrement plusieurs façons d'implémenter ceci, voici la nôtre :

Nous allons mettre une variable globale, *stream_stop* qui sera *False* par défaut. A l'appui du bouton *Stop*, une route est appelée en Ajax et sa fonction a pour but de passer cette variable à *True*. Dans la fonction *on_status()* de la classe *Stream*, on décidera de couper le stream si la variable est *True*, sinon de stocker le tweet "attrapé" par le *listener*.

4.2 Stockage

Comme vu plus haut, nous utilisons *MongoDB* pour le stockage des tweets. Ce système de gestion de base de données a été notre choix pour ce projet car il est simple d'utilisation et comme les objets que l'on stocke ne sont pas des objets complexes (pas des jointures) le choix a été évident.

Le langage de l'application étant le *Python*, nous devions trouver la bibliothèque permettant d'utiliser au mieux *MongoDB*.

Pymongo est une bibliothèque permettant d'utiliser facilement MongoDB, directement dans un projet *Python*.
(Vous trouverez l'API de *Pymongo* directement dans la bibliographie).

Diagramme représentant les liens entre Python Web et MongoDB

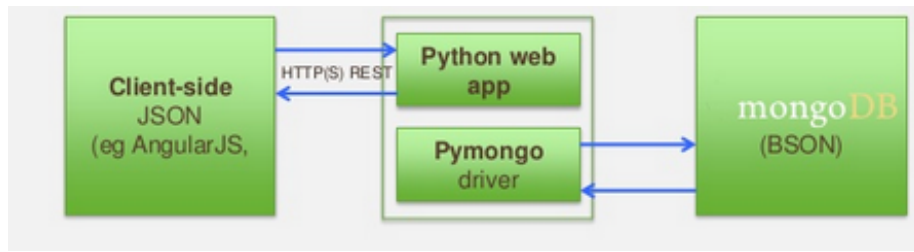


FIG. 5 – *Pymongo*, le lien entre *Python* et *MongoDB*

Au sein de notre projet, l'utilisation de *Pymongo* se trouve lorsque nous voulons insérer des tweets dans notre base de données et lorsque nous voulons récupérer ces mêmes tweets.

```
from flask_pymongo import PyMongo
```

FIG. 6 – *Importation de Pymongo depuis Flask*

Nous utiliserons, par la suite, directement la variable *mongo* qui est initialisé comme ceci :

```
mongo = PyMongo(app)
```

Nous allons analyser la gestion du stockage avec *Pymongo*. Comme vu précédemment, la fonction de récupération depuis *tweepy* appelle une fonction *stock_tweet* qui permet, elle, de stocker les tweets.

```
def stock_tweets(tweet):
    tweets_table = mongo.db.tweets
    tweets_table.insert({'session_id': session['last_session'], 'tweet_object': tweet._json})
```

FIG. 7 – Fonction qui permet de stocker les tweets récupérés via la fonction de filtrage

La première ligne récupère l'objet de stockage *mongo* dans une variable et la deuxième ligne permet d'insérer dans cet objet deux attributs: l'*id* de la session qui est un objet session récupéré à la création de la session par l'utilisateur et l'objet *tweet* récupéré au filtrage. Pour plus de facilité dans l'utilisation de cet objet, nous le convertissons en json.

L'utilisation de cet objet *tweet* se fait dans toutes les fonctions de récupération de tweets dans la base de données. Pour plus de facilité, nous avons créé une fonction qui récupère l'objet de stockage pour la récupération des tweets dans la base de données.

```
def tweets_by_session_id(session_id):
    return mongo.db.tweets.find({'session_id': session_id})
```

FIG. 8 – Fonction de récupération de tweets

Dans cette fonction, nous utilisons l'objet de stockage *mongo.db.tweets* (*tweets* est le nom de la base de données) et nous lui demandons de chercher tous les tweets dans la base de données où l'*id* de la session est l'*id* passée en paramètres. Cette fonction est utilisée dans les autres fonctions de récupération.

```
def retrieve_all_tweets_text():
    if 'last_session' in session:
        tweets_table = tweets_by_session_id(session['last_session'])
        tweet_text = ""
        for tweet in tweets_table:
            if 'full_text' in tweet['tweet_object']:
                tweet_text = tweet_text + tweet['tweet_object']['full_text']
            elif 'text' in tweet['tweet_object']:
                tweet_text = tweet_text + tweet['tweet_object']['text']
        return word_splitter(tweet_text)
```

FIG. 9 – Fonction de récupération des textes de tous les tweets dans la base de données

La première ligne de cette fonction est l'appel de la fonction *tweets_by_session_id()* vue précédemment. L'*id* de la session est récupérée depuis la création de la session par l'utilisateur. La suite de la fonction est la récupération des textes des tweets contenus dans la base de données pour la création du nuage de mots que vous verrez juste après.

4.3 Visualisation

4.3.1 Nuage de Mots

Un nuage de mots est un outil qui permet à l'utilisateur de voir les tweets récupérés de la meilleure façon. Simple, précis et complet, cet outil était un outil indispensable pour notre plate-forme d'analyse de tweets. Pour l'implémentation de ce *wordcloud*, nous avons utilisé la bibliothèque *D3.js* comme vu plus haut.

Comme pré-requis pour la visualisation, nous avons un projet *git* assez complet sur les nuages de mots *D3*. (Le lien se trouve en bibliographie).

Ce projet nous a allégé la visualisation et a permis un joli résultat en peu de temps.

Premièrement, après la récupération des tweets et leur stockage, la fonction *retrieve_all_tweet_text()* permet de récupérer les tweets déjà stockés et de construire la chaîne de caractères où les textes de tous les tweets récupérés seront concaténés. Ces tweets passent, après, par la fonction *wordsplit()* (cf. Annexe D) où le texte de tous les tweets vont passer en filtrage et les fréquences des mots dans les tweets vont être rassemblés pour, enfin, avoir une liste de mots avec ses fréquences dans les tweets.

Cette liste ira directement dans la fonction *draw()* du Javascript via cette fonction:

```
@app.route('/result-wordcloud/')
def wordcloud():
    words = retrieve_all_tweets_text()
    return json.dumps(words)
```

FIG. 10 – Fonction de lien entre le côté client et le côté serveur du site

La ligne 32 permet au Javascript de récupérer la liste construite précédemment et de pouvoir l'afficher. Cette fonction utilise plusieurs fonctionnalités du *D3* via soit par le fichier *d3-cloud* venu directement de *Github* soit de l'*API D3*. Cette fonction contient plusieurs paramètres d'affichage comme la longueur et la largeur de l'objet que va contenir le nuage de mots. Le résultat final est visible en annexes (cf. Annexe E). Le nuage de mots attends la fin du filtrage pour pouvoir s'afficher.

4.3.2 Histogramme

En ce qui concerne, l'histogramme présentant le nombre de tweets en fonction d'intervalle de temps, l'organisation est semblable à celle des nuages de mots.

Nous commençons à récupérer les tweets de la base de données par ordre chronologique pour créer plus tard, des intervalles de différents créneaux. Après avoir récupéré les tweets, on compte le nombre de tweets par intervalle de temps sélectionné par l'utilisateur (par défaut, l'intervalle est automatisé c'est à dire,

cela dépend du début et de la fin des dates des tweets contenus dans la base de données). Le résultats de ce comptage se trouve dans une liste avec comme clés (nombre de tweets, date de début, date de fin).

Cette liste est envoyée dans la fonction *histogram()* en Javascript. (cf. *Annexe F*) Cette fonction gère les paramètres d’affichage (comme dans les nuages des mots, nous utilisons plusieurs fonctions de *D3.js*). Le rendu final de l’histogramme est disponible en annexes. L’histogramme a une fonctionnalité qui permet au clic sur un élément de l’histogramme, d’afficher un nouveau nuage de mots qui aura comme intervalle l’intervalle de l’élément. (cf. *Annexe G*)

4.3.3 Autres visuels

D’autres fonctionnalités font la beauté du visuel du site comme par exemple le chargement du nombre de tweets lors du filtrage toutes les secondes ainsi que, par exemple, l’affichage d’une boucle visuelle qui fait patienter l’utilisateur le temps de l’affichage du nuage de mot. (cf. *Annexe H*)

L’affichage de ces deux fonctionnalités, à savoir, le nuage de mot et l’histogramme sont gérées en *Ajax*. C’est à dire se charge dynamiquement pour que l’utilisateur puisse être au courant de l’avancée de sa recherche.

4.3.4 Affichage des tweets par intervalle de temps

Enfin, une autre fonctionnalité de la page des statistiques du site est l’affichage des tweets par ordre croissant du nombre de retweets.

Cette fonctionnalité ressemble aux deux précédentes. La récupération dans la base de données des tweets se fait en filtrant les tweets par dates. Cette fonctionnalité se lance en même temps que le petit nuage de mots et dans la même intervalle de temps (l’intervalle de temps de l’élément de l’histogramme). Le résultat est mis dans un tableau et est trié par ordre croissant des retweets. Ce tableau est envoyé sous forme json à la fonction côté client pour l’affichage. La possibilité de voir les tweets sous la même forme que sur Twitter permet à l’utilisateur de comprendre le contexte de sa recherche et de se sentir à l’aise avec les différents visuels que l’application lui propose.

Le résultat de cette affichage vous sera montré lors de la soutenance.

5 Organisation

5.1 Répartition des tâches

Toutes les deux semaines, après chaque réunion avec notre encadrant, nous listions les nouvelles tâches à implémenter. L'outil de gestion de projet *Trello* nous a permis de diviser et repartir les fonctionnalités à développer de manière équitable.

Nous avons également adopté d'autres aspects de la méthode agile, comme le *dailymeeting*. Tous les trois jours, nous discutons ensemble afin de prendre connaissance de l'avancée du projet, des problèmes rencontrés et des solutions à apporter.

5.2 Communication

En plus des réunions avec Pascal PONCELET, nous n'avons pas hésité à lui envoyer des e-mails afin qu'il nous éclaire sur certains questionnements.

Quant à la communication entre les membres de l'équipe, nous utilisons des applications de messagerie instantanée.

5.3 Programmer ensemble

Les développements apportés par chaque membre de l'équipe ont pu être mis en commun grâce au gestionnaire de source *Github*. Il nous a permis d'archiver l'ensemble des codes implémentés et de gérer les conflits causés par cette mise en commun.

Remerciements

Nous tenons à remercier tout particulièrement Pascal PONCELET, notre encadrant de TER, pour sa grande disponibilité, son appui ainsi que son temps accordé à notre projet durant toute sa durée.

Nous tenons aussi à remercier toutes les personnes qui ont pu, de près ou de loin, contribuer à la concrétisation de celui-ci.

Conclusion

Durant ce travail d'étude et de recherche de fin de Licence Informatique, notre encadrant, Pascal PONCELET, nous a confié le sujet suivant : Analyse de tweets. Une idée intéressante se dégage rapidement d'un tel sujet : proposer aux utilisateurs de notre application d'observer, par l'intermédiaire de visuels, la manière dont un sujet est considéré sur Twitter ainsi que son évolution au cours du temps. En étudiant le problème posé avec notre encadrant tout au fil des réunions, en répertoriant ce que nous avons besoin pour réaliser une telle application et en concevant une analyse de ce qui se faisait sur le marché actuel, nous avons pu mettre en lumière l'orientation de notre projet.

Après plusieurs semaines de développement, nous sommes tous fiers de ce que nous avons pu faire lors de ce TER. Dans un premier temps, notamment par le choix du sujet, l'analyse et le développement fut un réel plaisir pour nous tous. En effet, le sujet abordé est en adéquation parfaite avec les poursuites d'études et les métiers que nous visons, c'est-à-dire le domaine de la donnée et des statistiques. Dans un second temps, nous avons appris à utiliser de nouvelles technologies ; par besoins techniques mais aussi par volonté de "sortir des sentiers battus". Pour finir, à l'instar d'autres projets en équipe, nous nous sommes d'autant plus familiarisés à la gestion d'un projet ; que ce soit par la répartition des tâches ou par l'utilisation d'outils de collaboration et de communication.

Ce TER aura été pour nous positif sur tous les plans ; aussi bien pour chacun de nous par la mise en abîme d'un domaine qui nous intéresse et qui fait écho aux poursuites d'études que nous envisageons, mais aussi sur le plan humain, dans les différentes interactions que nous avons été amenés d'avoir au sein du groupe et entre le groupe et notre encadrant. Un tout qui se présentera être un vrai tremplin pour une future carrière.

Table des figures

1	Etapes d'une analyse des données	2
2	Diagramme de cas d'utilisation de notre application	5
3	Récupération des tweets passés	6
4	Récupération des tweets en direct	8
5	Pymongo, le lien entre Python et MongoDB	10
6	Importation de Pymongo depuis Flask	10
7	Fonction qui permet de stocker les tweets récupérés via la fonction de filtrage	11
8	Fonction de récupération de tweets	11
9	Fonction de récupération des textes de tous les tweets dans la base de données	11
10	Fonction de lien entre le côté client et le côté serveur du site . . .	12

Annexes

A Fonction filter

```
def filter(keywords=None, geocode=None, stream=False, startdate=None, stopdate=None, user=None, language=None, tweets_batch=None):
    if geocode is not "":
        # Passe d'une chaîne de caractère en un tableau de floats (chaque élément séparé d'une virgule)
        geocode = [float(s) for s in geocode.split(",")]
    if stream:
        global stream_stop
        stream_stop = False
        if user is not "":
            user = getIdByUser(user)
        stream_o = tweepy.Stream(auth=api.auth, listener=Stream())
        stream_o.filter(locations=geocode, track=[keywords], languages=[language], follow=[user])
    else:
        geocode = None
        query = keywords
        if startdate != "" and stopdate != "":
            query = query + " since:" + startdate + " until:" + stopdate
        if user != "":
            query = query + " from:@" + user
        if language == "":
            language = "fr"
        for tweet in tweepy.Cursor(api.search, q=query, tweet_mode="extended", geocode=geocode, lang=language).items(int(tweets_batch)):
            stock_tweets(tweet)
```

B Interface de création de session tweets passés

Créez votre session

Nom de la session
MySessionName

Nombre de tweets récupérés par lancement
100

Filtres ?

Mots clés
Nutella

Date de début
21

Régions
3.78300108, 43.54854151, 3.97065725, 43.67333749


Utilisateur Twitter
@realDonaldTrump

Date de fin

Langue
Toutes

+

-



Leaflet

LANCER LA SESSION

C Interface de création de session tweets en direct

Créez votre session

Nom de la session
MySessionName

Filtres ?


Mots clés
Nutella

Régions
23
3.76500108, 43.54854151, 3.97065725, 43.67333749

Utilisateur Twitter
@realDonaldTrump

Langue
Toutes

+ -

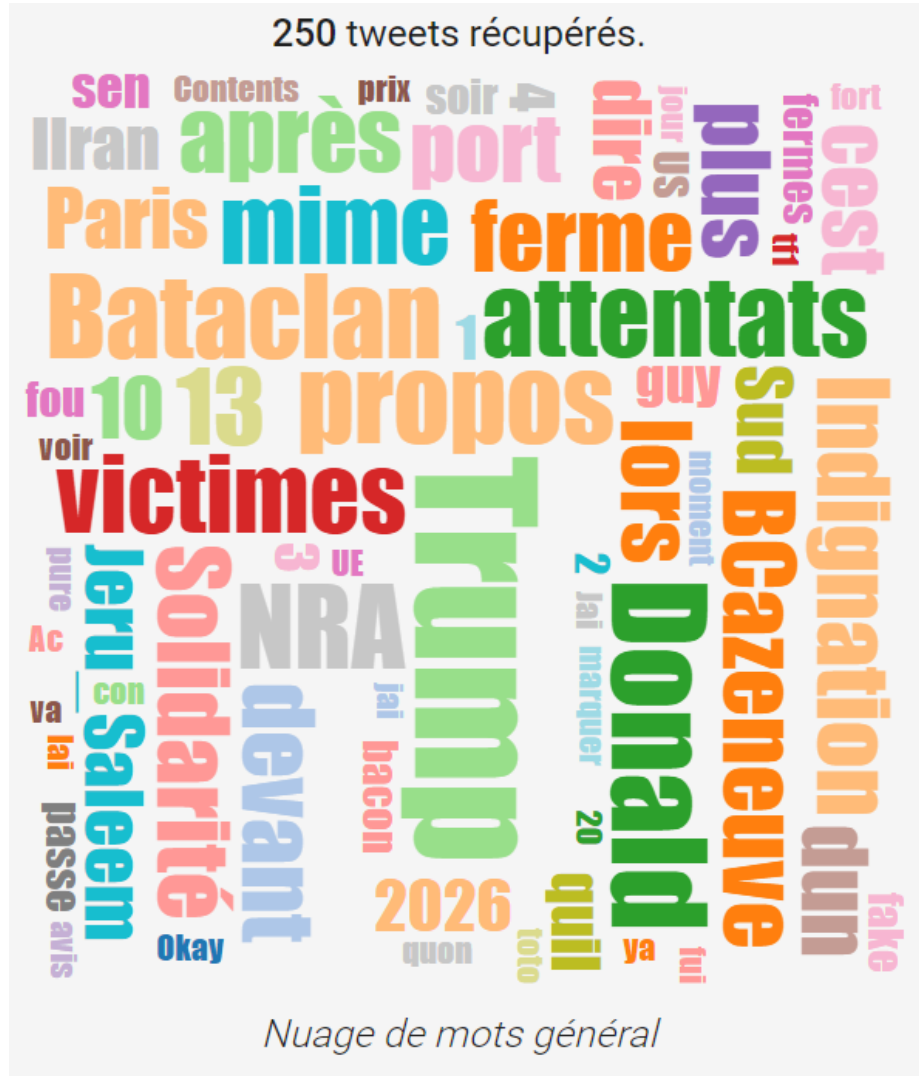


LANCER LA SESSION ▶

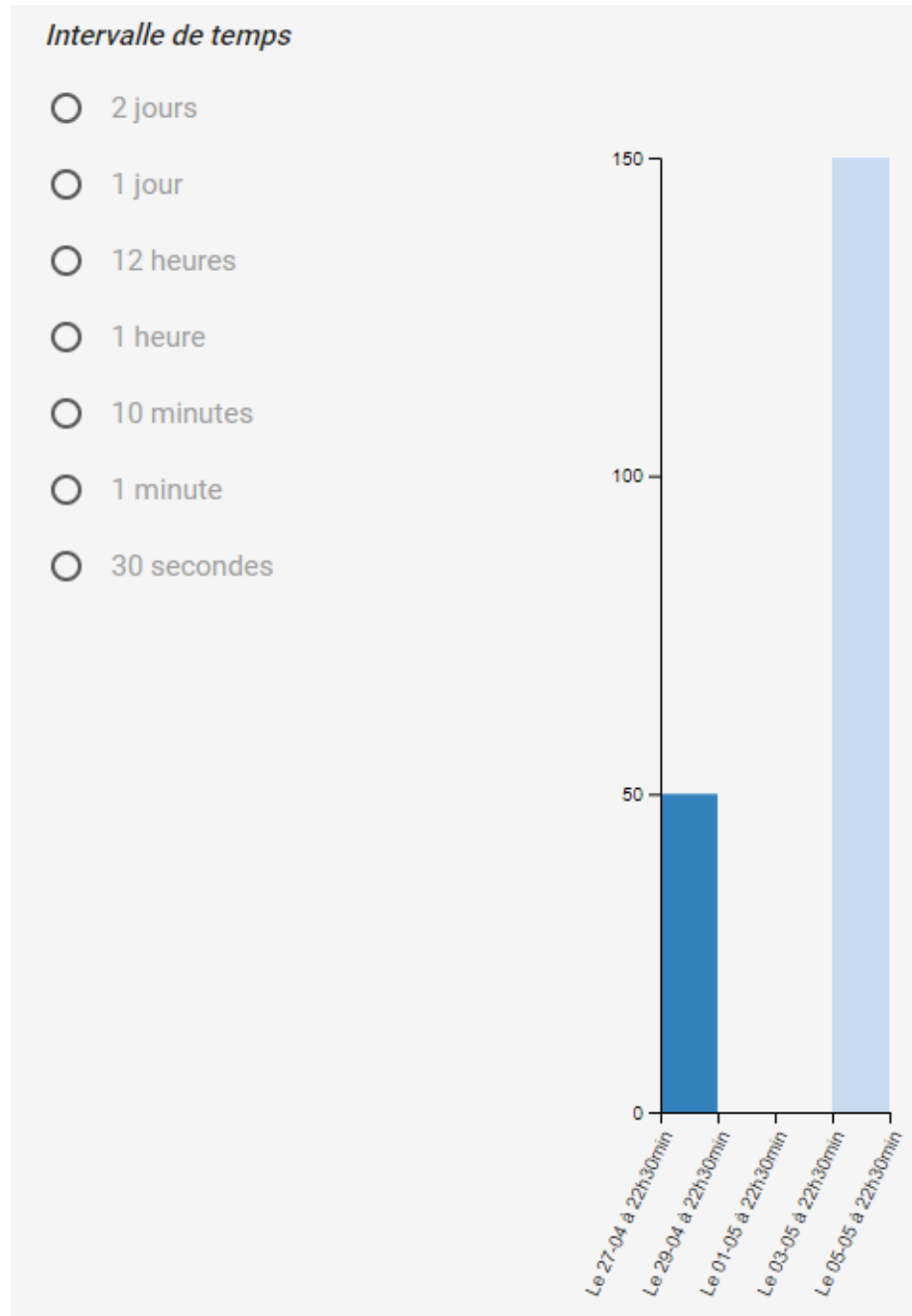
D Fonction wordsplit

```
def word_splitter(tweet_text):
    tweet_text = re.sub(r'(^\\w\\s|', '', tweet_text)
    tweet_text = re.sub(r'\\s|s+', ' ', tweet_text)
    pattern = re.compile('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|(!*\\(\\)|(?:%[0-9a-fA-F][0-9a-fA-F]))+')
    pattern.sub(' ', tweet_text)
    words = tweet_text.split(" ")
    new_words = []
    stop_words = get_stop_words('fr') + get_stop_words('en')
    words = [word for word in words if word.lower() not in stop_words and 'RT' not in word]
    word_counter = collections.Counter(words)
    for word in word_counter:
        if word_counter[word] >= 2:
            new_words.append({'text': word, 'size': word_counter[word]})
    return new_words
```

E



F L'histogramme



G



H Chargement des tweets

407 tweets récupérés.



Références

- [1] *Documentation Python 3* : <https://docs.python.org/fr/3/>
- [2] *Documentation Tweepy* : <http://docs.tweepy.org/en/v3.5.0/>
- [3] *Documentation Flask* : <http://flask.pocoo.org/docs/0.12/>
- [4] *Documentaion D3.js* : <https://d3js.org/>
- [5] *Documentation Pymongo* : <https://api.mongodb.com/python/current/>
- [6] *Documentation Materialize* : <https://materializecss.com/>
- [7] *Documentation Leaflet.js* : <https://leafletjs.com/reference-1.3.0.html>
- [8] *Documentation Overleaf* : <https://www.overleaf.com/devs>
- [9] *Dépôt Github de Tweetostats* : <https://github.com/Haytu/app-python-analyse-tweets>
- [10] *Dépôt Github Leaflet-areaslect* : <https://github.com/heyman/leaflet-areaselect>
- [11] *Dépôt Github google-image-download* : <https://github.com/hardikvasa/google-images-download>