Université de Cergy-Pontoise

Rapport de projet de synthèse

WEB SENSORS

Rapporteur : Dan VODISLAV
Tuteur : Dimitrios KOTZINOS
Encadrant de la gestion de projet : Tianxiao LIU

Réalisé par :
Mohamed Boudjemaa HAMICI
Mohamed II BAYO
Benoît FAGOT
Hong Phuc VU
Lahcen AIT BELLA

13 juin 2019

Table des matières

Table des figures

Chapitre 1

Introduction et présentation du projet

1.1 Contexte du projet

Actuellement, internet en général, et les réseaux sociaux plus particulièrement, sont devenus une source de données gigantesque, chaque seconde, 29.000 Gigaoctets (Go) d'informations sont publiées dans le monde, soit 2,5 exaoctets par jour soit 912,5 exaoctets par an.

Ces Gigaoctets de données sont une source d'information très précieuse, le "big data" est en croissance fulgurante dans le domaine de l'informatique, c'est ce qui pousse de plus en plus les entreprises et les universités à entreprendre dans ce domaine et à se préparer aux défis que celui-ci pose, notamment en terme de nouvelles façons de sauvegarder les données, de les analyser, d'y accéder etc...

Il y a aussi de plus en plus d'engouement envers les applications permettant l'obtention d'informations rapidement et intuitivement, un exemple serait Google Maps, qui permet non seulement de se géolocaliser mais aussi d'afficher, par exemple, les restaurants se trouvant aux alentours ainsi que leurs horaires d'ouverture.
C'est dans ce contexte que nous avons pensé que notre projet serait pertinent à réaliser et qu'il serait intéressant pour nous de nous y approfondir afin d'en apprendre un peu plus sur le domaine du big data, de l'analyse de données, ainsi que pour nous rapprocher des demandes du marché. Maintenant que nous avons défini le contexte, nous expliquerons dans la section suivante l'objectif de notre projet qui sera encore détaillé par la suite.

Ce projet nous a été proposé par Monsieur Dimitris Kotzinos et s'intitule « Web Sensors ».

Étant donné que notre équipe se dirige vers un parcours Systèmes Intelligents et Distribués (SID), nous sommes très intéressés par tout ce qui concerne les bases de données et la donnée plus généralement, de plus ce projet nous permet d'approfondir nos connaissances en fouille de données ainsi qu'en analyse de données.

Ce projet est aussi axé sur le traitement de données issues du web et des réseaux sociaux, ce sont des thèmes d'actualités et nous permettront d'avoir des connaissances plus poussées dans ces thèmes porteurs.

1.2 Objectif du Projet

La facilité d'accès à internet et aux smartphones fait que les informations et les événements sont partagés en temps réel sur le web et surtout sur les réseaux sociaux qui enregistrent une activité incroyable. Ceci nous permet d'obtenir des informations en temps réel sur ce qui se passe dans le monde et ainsi de découvrir tous les événements de l'actualité extrêmement rapidement.

Chaque seconde environ 5 900 tweets sont expédiés sur le site de micro-blogging Twitter. Cela représente 504 millions de tweets par jour ou 184 milliards par an.

La quantité de données est donc gigantesque et c'est dans ce contexte Big Data que s'insère notre projet.

Ce dernier à pour but de mettre en place des capteurs sur le web afin de détecter automatiquement des évènements dans un premier temps. Nous considérons comme événement, tout fait marquant de l'actualité, pouvant être décrit et situé dans le temps et l'espace, par exemple, un concert de musique, un débat télévisé ou une finale de ligue des champions. Un événement est caractérisé par :

- Sa description, ce qui permet de comprendre le sujet de celui ci,
- Son lieu, pour le situer géographiquement, sauf pour les événements abstraits comme les polémiques sur internet
- Sa date qui le situe dans le temps. Certains événements ont aussi une durée qui devra aussi être prise en compte

Considérant ces éléments, nous devrons dans un second temps rechercher les informations citées plus haut afin d'enrichir nos connaissances sur ces événements et les enregistrer en base de données. Une fois ce travail fait, toutes ces informations devront être restituées à l'utilisateur grâce à une interface homme-machine qui se présentera sous forme de pages web.

Nous restituerons aussi les tweets les plus pertinents selon l'événement choisi, ainsi que des articles de presse, des statistiques et des graphiques afin de permettre à l'utilisateur de mieux appréhender le déroulement de l'événement et son évolution sur le web.

Notre projet permettra donc au client de se mettre au courant des événements se produisant à travers le monde et de les analyser ce qui ces derniers grâce aux différents indicateurs cités plus haut.

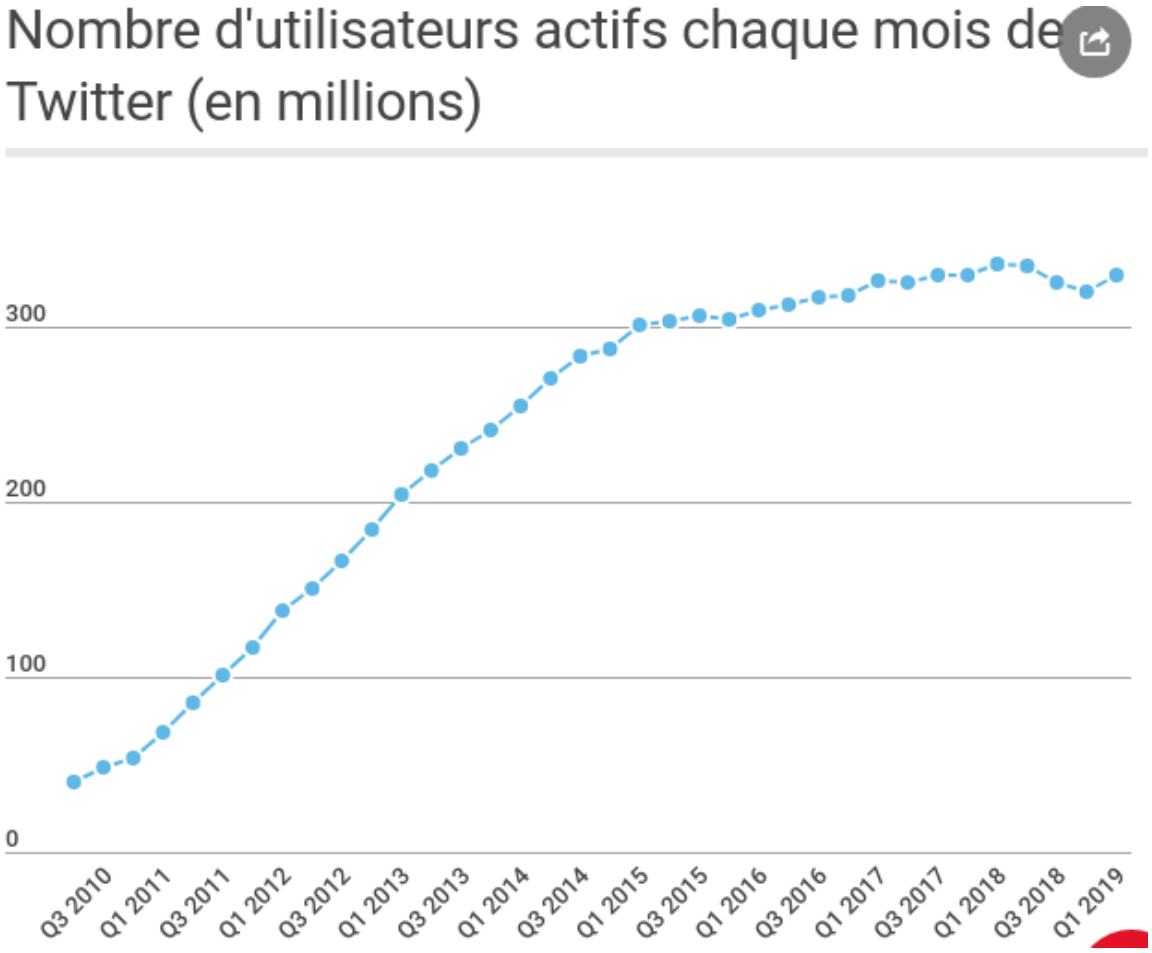


FIGURE 1.1 – statistique nombre utilisateur twitter par mois

Ces liens ci-dessous confirme ces formations : https ://www.numerama.com/tech/292891-pourquoi-twitter-se-prepare-a-abandonner-la-limite-des-140-caracteres-pour-passer-a-280.html

Web Sensors est regroupé dans plusieurs domaines notamment :

- Cloud Computing pour la recherche des mots clés dans un article, la recherche d'un événement grâce aux profils réalisés la première année
- Data Mining pour la classification

1.3 Mise en Scénario

Donnons un exemple de mise en place de notre projet. De nos jours, les données du Big Data sont primordiales pour toute entreprise se voulant compétitive sur le marché. En particulier, la détection automatique d'événements est très intéressante pour le milieu journalistique, qui est toujours à la course de l'information pour traiter des événements les plus intéressants.
Lorsqu'un événement a lieu, il est quasiment certain que des tweets à ce sujet vont avoir lieu.

Prenons la mort du footballer "José Antonio Reyes", ayant eu lieu il y a peu de temps.Grâce à notre produit, les journalistes sont informés qu'un événement a eu lieu sur la personne José Antonio Reyes.

La description estimée par nos "web sensors" va décrire que l'événement ayant eu lieu est (malheureusement) son décès. Ils estimeront aussi une date et un lieu. Le journaliste pourra ensuite sélectionner des critères de filtrage des tweets recueillis par nos "web sensors" afin d'obtenir des informations supplémentaires de ces tweets.

Ainsi, il aura pris une avance sur la concurrence car son travail de recherche a été grandement facilité par notre produit.

Le monitoring d'événements est très intéressant car il continue dans le temps. Si l'on prend cette fois-ci un événement plus long comme l'incendie de Notre-Dame : le monde entier est en attente d'informations en temps réel. Quoi de mieux que les témoignages réels des personnes sur place via leurs tweets ? Notre produit les récolte à intervalles fréquents, et on peut ainsi savoir quand l'incendie est éteint, et deux jours plus tards quelle est la conclusion des dégâts, l'estimation des travaux, etc.

Notre projet s'adresse à divers catégories de personnes, par exemple :

- Les journalistes qui désirent s'informer objectivement à partir des messages sur les réseaux sociaux. Ils pourront récupérer la story de l'événement qui contient les tweets importants de l'événement. Ces tweets importants pourront leur indiquer les faits importants de l'événement. Mais aussi un indice de la popularité de l'événement et une comparaison par rapport aux autres événements.
- Les sociétés qui veulent savoir précisément ce que pense le public vis-à-vis de leur produit ou de leur société.
  Prenons l'exemple d'une société qui est dans le milieu du Jeux Vidéo, elle pourra avoir des informations fournis par notre application à la sortie de son jeu comme :
  - Le nombre d'utilisateurs qui parlent de leur jeu
  - Avoir un retour sur les tweets les plus importants et les plus vus sur Twitter

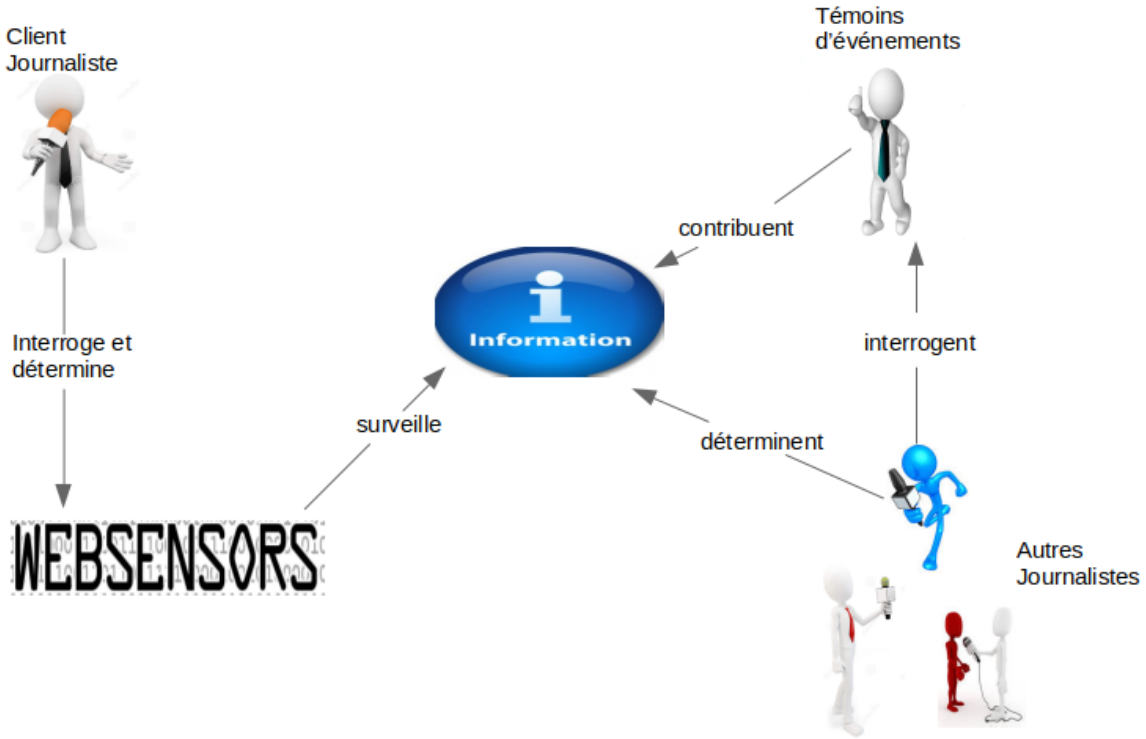


FIGURE 1.2 – mise en scénario utilisateur

1.4 Organisation du rapport

Ce rapport est organisé autour des chapitres suivants :

Chapitre 2 : Présentation et spécification du projet

Dans ce chapitre, nous expliquerons l'objectif et les attentes de notre projet. Quels sont ses spécifications, ses fonctionnalités attendues, sa conception générale, les outils utilisés, les problématiques rencontrées et comment tout cela s'assemble.

Chapitre 3 : Base de données :

Ici, nous reviendrons sur l'analyse effectuée pour choisir notre base de données.
Nous décrirons également comment nous avons choisi et décidé de récolter et intégrer les données issues de Twitter et des flux RSS.

Chapitre 4 : Découverte de la description :

Ce chapitre est consacré à l'analyse des tweets pour déterminer une description de l'événement sous la forme d'une phrase ayant un sens en Français. Nous expliquerons comment nous avons procédé pour arriver à cette fin.

Chapitre 5 : Découverte du lieu :

Cette partie est dédiée pour déterminer les lieux d'un document. Elle contient la mise en place de base de données de lieux, les méthodes pour interroger les données ainsi que les résultats obtenues lors du déploiement de système.

Chapitre 6 : Découverte de la date

Cette partie est dédiée pour détecter la date de l'événement. Nous analysons les tweets pour obtenir les dates et les durées qu'ils contiennent, soit elles sont écritent en chiffre ou bien en lettre, soit elles sont indiqué avec des mots qui exprime une date. Enfin nous déterminons celle qui se répète le plus.

Chapitre 7 : Rendu Final :

Cette partie est consacrée à l'implémentation de l'interface client. Nous avons opté pour une page Web afin de retranscrire les résultats de nos analyses de données. Nous reviendrons sur les diverses fonctionnalités de notre IHM et l'aboutissement de nos travaux.

Chapitre 8 : Gestion de projet :

Nous allons aborder dans cette partie à indiquer le planning global que nous souhaitons mettre en œuvre et respecter tout au long du projet.

Chapitre 9 : Conclusion et perspectives :

Dans ce chapitre, nous donnerons nos conclusions et nous proposerons différentes perspectives possibles afin d'améliorer le projet et lui donner plus de valeur.

Chapitre 2

Présentation et spécification du projet

Dans ce chapitre, nous présenteront les tenants et les aboutissants de notre projet ainsi que ses spécifications.

2.1 Solutions existantes

Dans cette partie, nous analysons les solutions déjà existantes liés à ce sujet de projet afin d'obtenir un état de l'art et visualiser au mieux ce qui est attendu. A partir de cet état de l'art nous pouvons ensuite effectuer nos choix de conception.

2.1.1 Comment l'USGS utilise les données de Twitter pour détecter les tremblements de terre

Ce document parle d'USGS (United States Geological Survey), une agence scientifique qui étudie les paysages des Etats-Unis. Après le tremblement de terre du Sichuan en 2008, les personnes ont commencé à utiliser Twitter pour parler de cet événement.

Dans un premier temps, les personnes travaillant à l'USGS pensaient que Twitter n'était pas utile pour détecter un événement sur les tremblements de terre, mais après quelques recherches, les données récupérées les ont satisfaits. Après cette recherche, l'agence a commencé de plus en plus à utiliser l'API Twitter pour détecter les tremblements de terre. L'API est notamment efficace en coût car celle-ci est publique. Ils cherchent maintenant comment récupérer des données basées sur des séismes et obtenir des tweets de en plus rapidement[1].

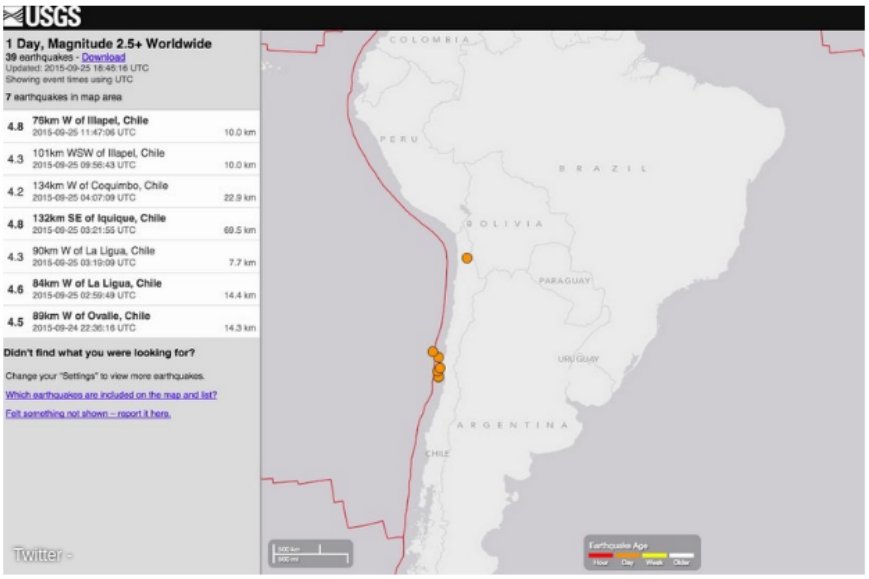


FIGURE 2.1 – les données de Twitter utilisées USGS pour suivre les tremblements de terre

2.2 Fonctionnalités attendues

2.2.1 Récupération des données

La première fonctionnalité attendue est la récupération des données. Par cela on entend le fait d'aller chercher des données pertinentes pouvant concerner des événements, et cela, en regardant les tendances des recherches par exemple.
Nous avons considéré diverses méthodes de récupération de données :

- récupération à l'aide d'APIs : il existe des APIs de type REST permettant à des utilisateurs externes l'accès à leur base de données (ou un échantillon restreint) à l'aide de fonctions prédéfinies par les concepteurs de ces dernières.
  Twitter est l'API que nous avons grandement utilisé pour notre produit.
- récupération par du web-scrapping : c'est une pratique assez mal vue car on peut la percevoir comme un pillage des données sans la permission des propriétaires.
  Elle est de plus très difficile à automatiser car les structures de sites d'informations ne sont jamais les mêmes.
- récupération par les flux-RSS : un peu semblable au web-scrapping, mais bien plus pratique. Les sites d'informations proposent un moyen de télécharger leurs articles via ce qu'on appelle les flux RSS et cela bien qu'exhaustif est plus envisageable à automatiser.
  Nous avons recueillis des flux de nombreux sites d'informations. Ces flux apportent par ailleurs une information sur le thème du sujet traité comme "Sport" ou "Politique" et peut nous aider lors des analyses.
- récupération par interrogation de la source DBpedia : cette méthode utilisée dans le projet d'Intégration et Entrepôts de Données aurait pu être pertinente de part la richesse de la source mais encore une fois, il est difficile d'automatiser un processus de recueil de données lorsque l'on ne sait pas prédire l'ontologie utilisée à l'avance.

2.2.2 Découverte des événements et des informations

La deuxième fonctionnalité est le fait de découvrir des évènements et d'obtenir les informations les concernant, que ce soit sa date, la personnalité importante de l'évènement, son type, sa localisation géographique, sa durée etc....

Pour cela, nous utilisons le traitement automatique du langage naturel ou en anglais NLP (natural language processing), ainsi que des APIs de type REST externes à notre produit pour obtenir une information plus précise pour un champ comme le lieu ou la date.

Les principales étapes du NLP :

- La tokenization du texte : il faut extraire tous les mots du corpus en retirant la ponctuation et et les majuscules.
- L'élimination des stop-words : on utilise une liste des mots de la langue du tweet qui sont jugés comme de la pollution de l'information et on les supprime des tokenized_words (exemple : "le", "a", "est", "du", "ou", etc.)
- Le stemming : en français racinisation ou désuffixation, cette étape est très importante et consiste à transformer des mots dérivés en leur racine. On peut par exemple ainsi conclure que "frontal" et "front" peuvent être acceptés comme étant identiques.
- Le processing : c'est le traitement effectué avec ce corpus de mots obtenus. Il est propre au but de l'algorithme dans lequel on l'utilise.
  Cas concret : obtenir la description d'un événement.
  A l'aide du corpus, on parcourt tous les flux RSS récoltés et on sélectionne la description du flux ayant obtenu le meilleur score de matching.

2.2.3 Sauvegarde des événements

Une fois ces informations recueillies, nous devons sauvegarder ces informations afin qu'elles soient accessibles facilement et rapidement.

2.2.4 Affichage des événements

Ceci consiste en l'affichage des événements à l'utilisateur, en considérant ses critères de recherche, et en allant trouver les informations précédemment sauvegardées.

2.3 Conception globale du projet

2.3.1 Vue pour l'utilisateur

Notre produit sera livré sous la forme d'une page web interactive avec le client. Il est destiné à des personnes à la recherche d'informations en temps réel sur les événements en cours, et discutés sur le web.

Les "web sensors" sont des algorithmes d'extractions et de traitements de données lancés quotidiennement, ainsi l'interface à laquelle accède l'utilisateur est constamment mise à jour et il n'a pas besoin de connaissances en informatique quelconques pour accéder aux informations élaborées par les "web sensors". Notre produit puise ses données dans l'API de Twitter (504 millions de tweets sont envoyés par jour sur la plateforme Twitter) et dans des articles de sites d'informations.

Le client, à la recherche d'informations concrètes et récentes, se verra proposer sur notre page plusieurs options de formulaires afin d'affiner la recherche pour répondre à ses critères.

Nous avons prévu deux options :

- une barre de recherche avec mots-clés
- une liste des tendances de Twitter les plus récentes

Ainsi l'utilisateur a maintenant accès aux résultats d'analyse de l'événement qu'il l'intéresse s'il a été détecté par nos "sensors".

Les résultats consistent en une interprétation globale de l'événement par une brève description, des statistiques sur l'activité de l'événement au cours du temps, et plusieurs options d'affichage de tweets pertinents de l'événement, avec article d'information correspondant à chacun de ces tweets s'ils existent. Les options d'affichage sont propres aux critères du client (si le client souhaite l'opinion des influenceurs, des tweets les plus populaires, des tweets correspondant aux pics d'activités, ...)
L'interface sera vu plus en détail dans le chapitre "Rendu final".

2.3.2 Architecture Techniques

Dans cette partie, nous détaillerons l'architecture et les rôles de chaque module composant le produit.

Architecture du produit :

Ci-dessous, on représente notre produit avec un schéma.

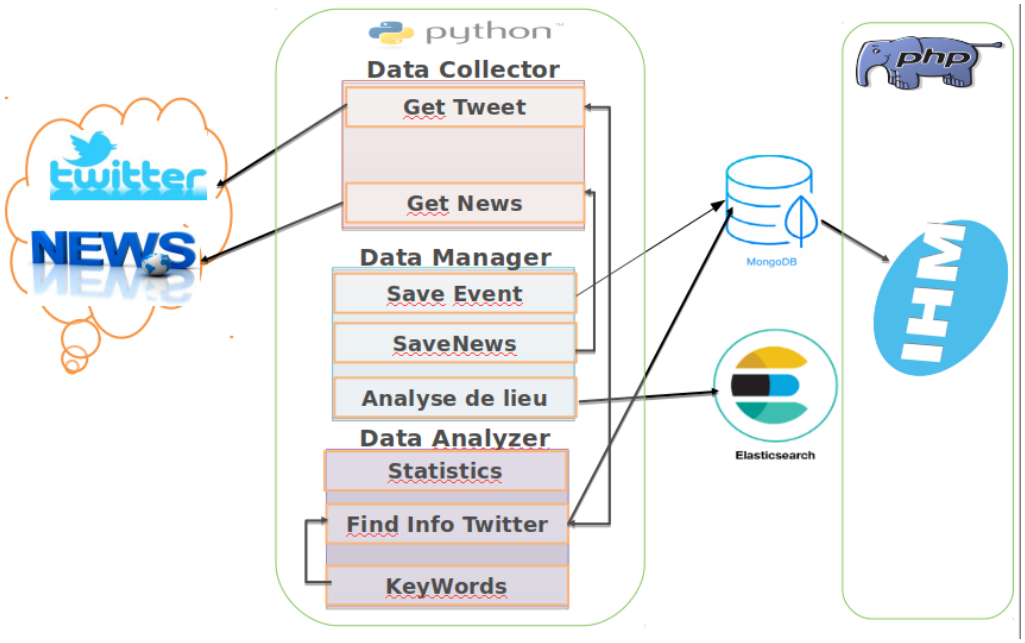


FIGURE 2.2 – architecture globale produit

Notre système est divisé en quatre modules, voici leurs fonctions :

Data Collector :

Récupère les flux RSS des sites qui sont entrés manuellement dans un fichier JSON Collecte les données (tweets, tendance, ect. . .)

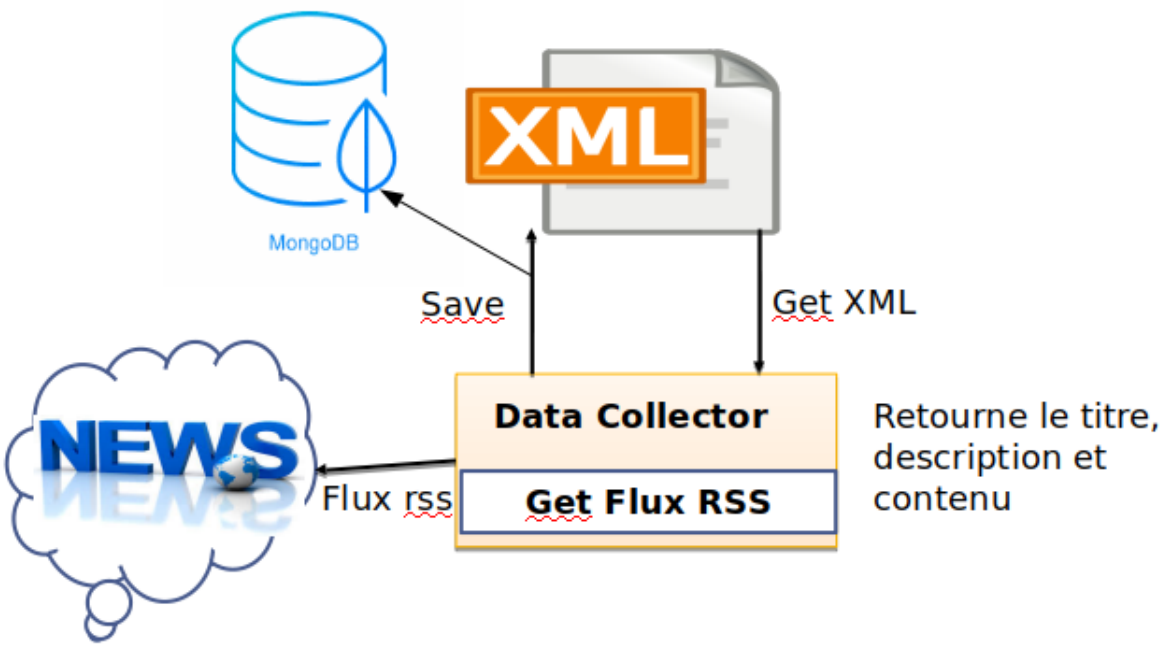


FIGURE 2.3 – architecture data collector

Data Manager

- Récupère les données nécessaire et stockées en base de données (événements, tweets. . .)
- Nous utiliserons une base de données adaptée à la volumétrie et aux données (MongoDB)

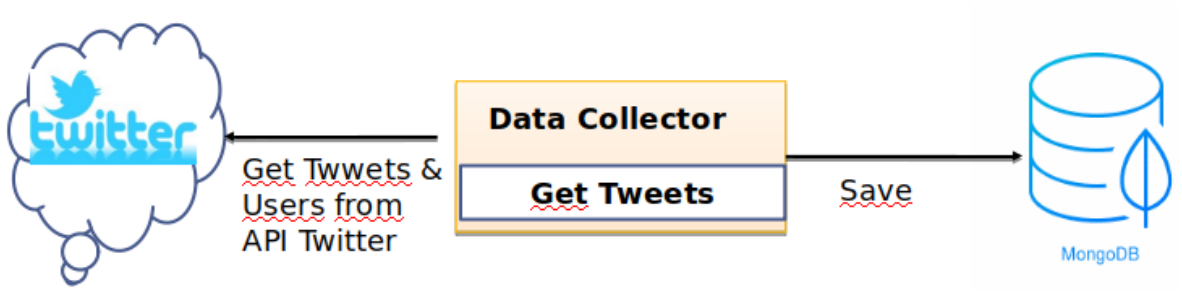


FIGURE 2.4 – architecture data manager

Data Analyzer

- Récupère les mots-clés depuis les articles / trendings Twitter
- Utilise les mots-clés pour requêter Twitter
- Extrait les informations utiles des tweets et flux RSS

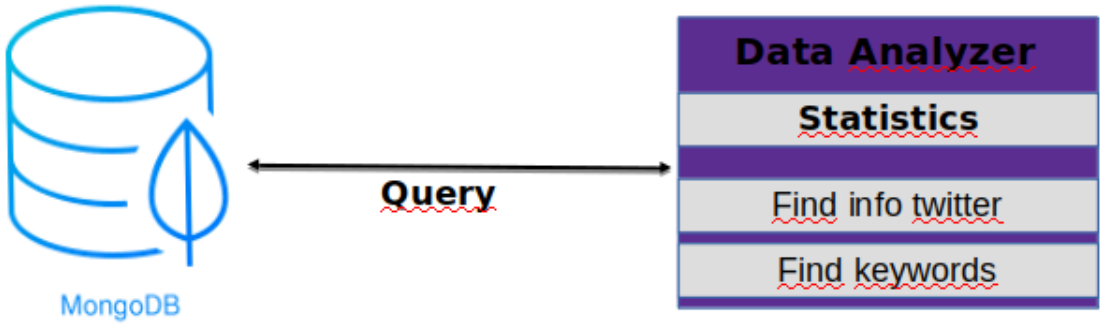


FIGURE 2.5 – architecture data analyzer

IHM : Vue de l'utilisateur

L'utilisateur pourra utiliser notre API pour découvrir des évènements en renseignant une date, un lieu ou un thème par exemple.
Une fois l'information renseignée, notre API lui retournera tous les résultats de notre base de données en format JSON ou XML.

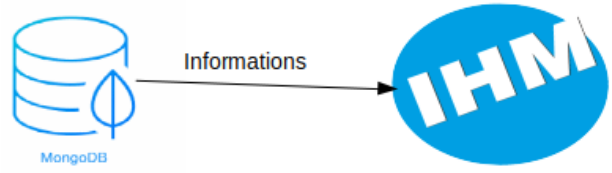


FIGURE 2.6 – architecture globale produit

2.3.3 Architecture Logicielle

Recherche des tendances

Pour ce faire, nous utiliserons des API de sites tels que Twitter ou Facebook afin de récupérer, dans un premier temps, les mots clés les plus recherchés, ce qui pourrait être indicatif d'une tendance.

Cette tendance pourrait être un mot clé s'apparentant à un évènement, mais quoiqu'il en soit, c'est représentatif d'une tendance importante et donc, dans le cas où c'est un évènement, le présenter dans les résultats est d'autant plus intéressant.

Découverte des événements et des informations

Une fois les mots clés repérés, nous utiliserons un web scrapper afin de chercher les informations sur ces mots clés, découvrir lesquels sont en lien avec un évènement, ensuite nous chercherons les informations essentielles le concernant.

Les informations importantes à rechercher sont les suivantes :

- La date
- Le lieu
- Le type

Les évènements seront transmis sous forme d'objets composés de différents attributs, ou d'une matrice de vecteurs, chaque vecteur étant un évènement.

Mémorisation en Base

Une fois ces informations recueillies, nous devons les enregistrer afin qu'elles soient accessibles.

Nous avons considéré qu'il était plus intéressant de les mettre dans une base de données NoSql.
En séparant les informations en deux bases de données différentes, il serait probablement plus intéressant de le faire ainsi, et de profiter de la vitesse d'accès d'une base de données NoSQL sur les informations récentes ou futures.

Restitution des données

Ceci constitue en l'affichage des évènements à l'utilisateur, en considérant ses critères de recherche, et en allant trouver les informations sur notre base de données. Si ces informations sont introuvables dans notre base de données, on pourra éventuellement refaire une recherche en utilisant ses mots clés comme base au lieu d'utiliser les tendances.
Pour résumer cette architecture, nous présentons le schéma suivant :

FIGURE 2.7 – résumer cette architecture restitution des données

2.4 Problématiques identifiées et solutions envisagées

Nous avons identifié deux problématiques.

2.4.1 Informations erronées

La première problématique posée à été la possibilité d'avoir des informations erronées, que ce soit sur le net ou dans les réseaux sociaux.

Nous avons décidé de palier à ce problème en vérifiant les données en croisant différentes informations recueillies sur plusieurs sites fiables, tels que des journaux par exemple.

2.4.2 Type de base de données à utiliser

Nous nous sommes ensuite questionnés sur l'enregistrement des données en base.

Étant donné que les API retournent des fichiers en JSON, nous avons décidé d'utiliser une base NoSQL tel que MongoDB afin de pouvoir enregistrer les fichiers dans leur format d'origine avec plus de facilité. Ceci est d'autant plus pertinent à partir du moment où les données deviennent très importantes en volume.

2.4.3 Temps de recherche

Étant donné le volume de données à retenir au fil des recherches, nous devrons faire face au problème de temps d'accès sur une base de données NoSQL, et pour cela, nous avons pensé qu'il serait plus viable de diviser nos données sur deux bases de données de types différents, une base de données SQL où seront stockées les données les plus récentes, et une base de donnée NoSQL où seront archivées les données plus anciennes et peu réutilisées.

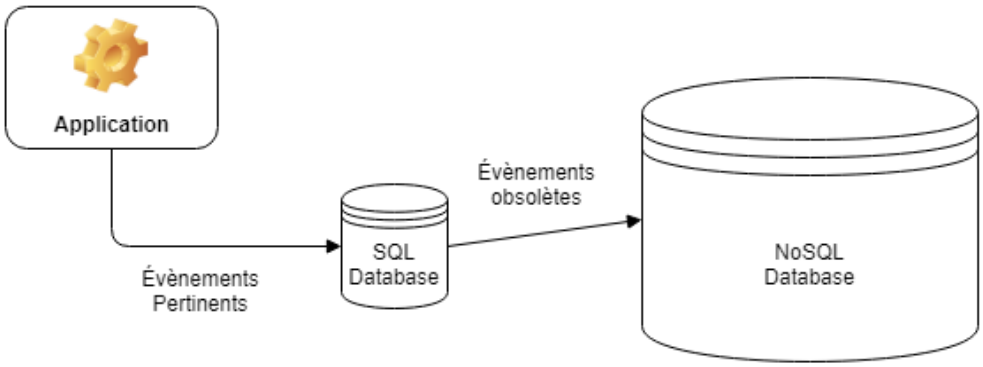


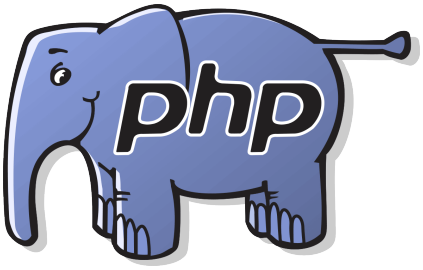
FIGURE 2.8 – résumer cette architecture restitution des données

2.5 Environnement de travail

Pour ce projet, nous n'avons nullement besoin d'outils matériels, seulement d'outils logiciels. Nous utiliserons donc seulement des langages de développement,les bases de données ainsi que quelques outils.

2.5.1 Langage de programmation

Python est un langage de programmation populaire. Il a été créé par Guido van Rossum et sorti en 1991. ce langage possède l'avantage de permettre la fabrication rapide de prototypes tout en proposant un nombre important de bibliothèques performantes développées par la communauté. De plus, la courbe d'apprentissage de ce langage est très basse. Nous utilisons la version 3 de python.



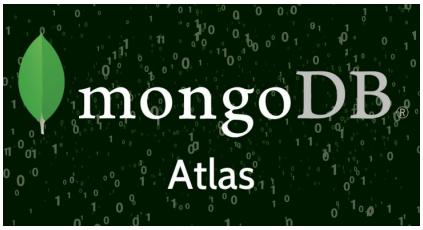
PHP Hypertext Preprocessor (PHP) est un langage de programmation qui permet aux développeurs Web de créer un contenu dynamique qui interagit avec des bases de données. PHP est essentiellement utilisé pour développer des applications logicielles basées sur le Web. Nous utilisons la version 7 de php

2.5.2 Base de données

ElasticSearch est un moteur de recherche RESTful open source reposant sur Apache Lucene et publié sous licence Apache. Il est basé sur le langage Java et permet de rechercher et d'indexer des documents dans divers formats. Nous utilisons la version 7.1

MongoDB est une base de données orientée document et multi-plateforme et à source ouverte, une sorte de base de données NoSQL. Nous utilisons la version 3.4.20

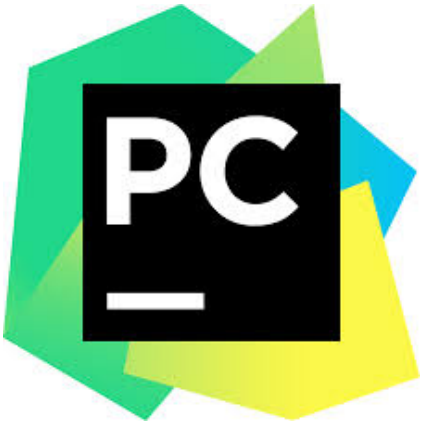
2.5.3 Outils



MongoDB Atlas est une base de données cloud entièrement gérée développée par les mêmes personnes qui construisent MongoDB.

Trello est un outil d'organisation collaboratif simple et gratuit. Nous l'utilisons pour définir nos différentes tâches et suivi des tâches.

2.5.4 IDE



PyCharm est un environnement de développement intégré (IDE) utilisé en programmation informatique , spécialement pour le langage Python . Il est développé par la société tchèque JetBrains .

Chapitre 3

Base de données

Dans ce chapitre, nous expliquerons le choix de notre base de données ainsi que son fonctionnement et son remplissage.

3.1 Analyse de la Problématique

D'abord la problématique est un ensemble de questions précises que l'on se pose au sujet d'une étude ou recherche spécifique.
En ce qui concerne notre projet, nous nous sommes posés les questions ci-après :

- **Quelle base de données serait la mieux adaptée pour notre projet ?**
- **Y-a-t'il un nombre limité de tweets à collecter ?**
- **Quels critères faut-il se baser pour le choix de la base de données ?**
- **Quelles sont les principales technologies à utiliser pour la base de donnée choisie ?**

Toutes ces questions sous-tendent notre problématique et ne manquent pas d'intérêt.

Notre projet étant axé sur la fouille des données, qui fait référence au volume de données, un nouveau domaine technologique a vu le jour : le Big Data. Inventé par les géants du web, ces solutions sont dessinées pour offrir un accès en temps réel à des bases de données géantes et performantes.

Plusieurs paramètres doivent être pris en compte dans un système classique lors de l'analyse des données :

- le temps de latence du disque qui est directement lié à la vitesse de rotation du disque de donnée.
- le temps de recherche sur les disques de données, si le disque contient un important volume de données.
- le temps de transfert, lorsque l'on veut manipuler de gros volume de données et surtout de nature différentes.

Pour optimiser les temps de traitement sur des bases de données géantes, plusieurs solutions peuvent entrer en jeu :
Des bases de données NoSQL (comme MongoDB, Cassandra ou Redis) qui implémentent des systèmes de stockage considérés comme plus performants que le traditionnel SQL pour l'analyse de données en masse (orienté clé/valeur, document, colonne ou graphe).

La base de donnée est l'une des parties les plus importantes de notre projet.
Dans le chapitre suivant, vous trouverez une étude des différentes bases de données que nous maîtrisons et notre choix définitif.

3.2 Étude de la base de données

Nous avons réalisé un certain nombre d'études sur trois types de base de données afin de trouver la plus adaptée à notre besoin selon certains critères.

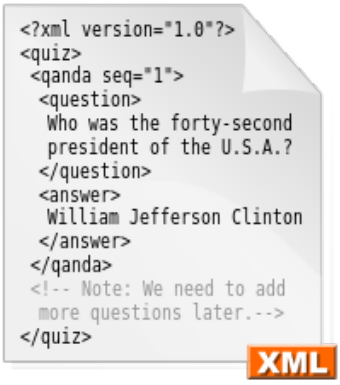
Tout d'abord, on doit être capable de réaliser un grand nombre d'insertions sans subir de ralentissements conséquents. Nous avons besoin de cela car nous allons devoir récupérer un grand nombre de tweets pour les insérer en base afin de réaliser ensuite notre fouille de données.

FIGURE 3.1 – choix base de données

De plus, pour réaliser cette fouille de données, le système doit proposer un moyen plus ou moins facile de requêter de façon complexe les données en faisant des liens entre plusieurs données de nature différente.

Les bases de données testées étaient les suivantes :

3.2.1 XML :



L'accès à un document au format texte est, par nature, lent et séquentiel. Il n'est pas possible de disposer d'index performant. De plus, se posent des problèmes de sécurité(dans certaines condition, limités), de gestion de l'intégrité , des transactions...
Dans quels cas on peut utiliser des fichiers XML dans une base de données ?

- on a des données fournies sous format XML qu'on souhaite traiter et publier(ex : open Data)
- Chercher un endroit pour stocker des pages web(fichier orientés structures)

Analogie à Twitter – Récupération des tweets :
Après quelques recherches, Twitter n'offre plus XML comme format de réponse pour leurs appels d'API depuis son passage à la version 1.1. Sur la version 1.0 il était possible de le faire, mais la récupération était jugée « maladroite » car y avait des incohérences qui ont poussées les développeurs à retirer cette fonctionnalité.

Les tweets sont actuellement récupérés au format « JSON », et il est possible de convertir les données de JSON en XML. Vu que nous travaillerons avec une grande quantité de données, nous ne pouvons pas nous permettre de perdre du temps à traiter les données de JSON à XML. Donc on a décidé d'exclure cette piste de base de données XML.[2]

3.2.2 Document – MongoDB

MongoDB est un système de gestion de base de données de type NoSQL. Ce système distribué sous licence AGPL est orienté document et est connu pour être facilement scalable (répartitions sur un nombre quelconque d'ordinateurs). MongoDB possède des pilotes dans de nombreux langages de programmation, incluant les langages suivants : PHP, Java, JavaScript, C, Python, Ruby ou Perl.

La particularité de mongoDB est sa flexibilité, on peut stocker des données de structures différentes dans une même collection(qui est équivalent à table pour SQL), contrairement à une base de données classique (MySQL par exemple) où il faut structurer les données selon les tables, les relation et les contraintes (ACID). Avec mongoDB, on construit les données de sorte que la sélection soit facile et permetet un grand nombre d'insertions de données ce qui est un point positif pour notre projet. De ce fait, une base de données MongoDB est amenée à changer au fur et à mesure de l'évolution de la demande du projet. Par exemple, si l'information de localisation des tweets devient plus importante, il est envisageable de revoir la structure des données stockées de sorte à ce que la requête s'exécute plus vite. Cela nous donne une certaine souplesse.

Drivers de base de données

Nous utiliserons Pymongo pour communiquer avec mongoDB en python.

Avantages

- **Performances :**
  Au niveau des performances, les créateurs ont eux-mêmes réalisé un benchmark de MongoDB [3] et montré que si le multithreading est activé, le nombre d'opérations (écriture ou lecture) par secondes peut monter jusqu'à 120 000 pour une latence d'environ 10 ms. C'est amplement suffisant, MongoDB est suffisamment robuste pour notre projet.
- **Format de données :**
  MongoDB est fait pour stocker des données sous un format analogue à JSON (JavaScript Object Notation) et c'est bien dans ce format que nous récupérerons les données de Twitter. Éviter une conversion de format de données est un avantage notable.

Inconvénients

- On pourrait penser que la souplesse offerte par MongoDB nous permet de travailler plus simplement avec les données mais on se rend vite compte que nous devons produire du code
- supplémentaire alors que la base de données aurait pu faire ces traitements. Pour chaque tweet, nous sommes obligés de vérifier son existence en base de données. Donc écrire du code supplémentaire pour assurer l'intégrité des données.

3.2.3 Relationnelle – MySQL

Mysql est un système de gestion de bases de données relationnelles l'un des (SGBDR) les plus utilisés au monde. Il est gratuit et très puissant. Il possède la double licence GPL et propriétaire depuis son rachat par Sun Microsystem eux-mêmes racheté par Oracle (concurrent direct de MySQL). Le logiciel reste cependant entièrement gratuit et libre. Il répond à une logique client/serveur , c'est à dire que plusieurs clients (ordinateurs distants) peuvent se connecter sur un seul serveur qui héberge les données.

Quels sont les avantages et les inconvénients d'utiliser MySQL ?

- **Avantages :**
  Code simple et facile à comprendre, populaire (documentations, aides en ligne, ...)
- **Inconvénients :**
  Très coûteux, pas compatible avec toutes les versions de python.

3.2.4 Décision prise : MongoDB

Étant donné que nous cherchons à récupérer des données en masse, nous devions choisir la base de données la plus optimisée parmi les 3 propositions pour nos besoins :

- Récupérer des données en grande quantité de façon constante
- Requêter facilement sur les données enregistrées

Vu que nous requêtons l'API de Twitter, nous avons éliminé XML comme choix puisque ce format n'est plus supporté.
Comme nous travaillons sur une grande quantité de données, MySQL n'était pas un choix valide dû à sa nature relationnelle. MySQL n'est pas optimisé pour le traitement et stockage des données en masse.

MongoDB est très performant en termes d'insertions et stockage de données et vu que l'API Twitter nous retourne les données au format JSON nous avons fini par choisir mongoDB, il correspond très bien à nos besoins pour ses hautes performances.

3.3 Hébergement de la base de données

Afin de travailler sur les mêmes données, il nous fallait une base de données commune, et donc il fallait l'héberger sur un environnement dédié.
Nous avons héberger notre base de données sur un serveur cloud de mongoDB gratuit en ligne « mongoDB Atlas » [4]
MongoDB Atlas est une base de données cloud entièrement gérée et développée par les mêmes personnes qui construisent MongoDB.
Inclus avec notre base de données cloud gratuite :

- 512 Mo de stockage
- RAM partagée
- Ensembles de réplicas hautement disponibles, cryptage de bout en bout, correctifs automatisés, API REST

FIGURE 3.2 – MongoDB Atlas

3.4 Collecter les tweets

Twitter propose une API(application programming interface) qui a pour but de proposer les données que Twitter a à sa disposition. Cette API est au format REST, c'est à dire qu'il faut envoyer une requête HTTP vers une URL précise en utilisant une méthode HTTP donné.

Pour commencer, nous avons faits des tests sur l'interface utilisateur de twitter « twitter.com/search » et à créer une version de l'API.

Par exemple, l'URL suivante nous permet de récupérer des tweets contenant le mot websensors : GET `https://api.twitter.com/1.1/search/tweets.json?q=websensors` De la même manière, il est possible de récupérer les données des utilisateurs

D'abord pour les collecter les données, il faut trouver des mots clés qui sont critiques au sein de notre processus car c'est grâce à eux qu'on va pouvoir collecter les tweets liés à un événement.

Nous avons commencé par rentrés les mots clés nous mêmes afin d'expérimenter manuellement la collecte de tweets. Ainsi afin d'automatiser la sélection des mots clés on a procéder comme suit :

3.4.1 Récupérer les tendances Twitter

trends = api.trends_place(615702) : le chiffre correspond à l'id de Paris.

api.trends_place(code lieu) : l'api twitter qui permet de récupérer les tendances twitter, en paramètre renseigner le WOEID(numéro d'identification Where On Earth).

L'api nous renvoie les 50 principaux sujets de tendances pour un code de lieu spécifique et les réponses sont dans un tableau objet de tendance. Pour notre projet on utilise le code pour Paris (615702)

3.4.2 Récupération des tweets

Une fois les tendances récupérées, il faut récupérer les tweets en utilisant l'API Twitter. Nous récupérons aussi d'autres informations comme les utilisateurs qui ont envoyés ces tweets, le nombre de followers, nombre de retweets, nombres de like de chaque tweets, la date de création du tweet etc...

Nous utilisons le curseur de l'API Twitter et également de la pagination, ci dessus l'instruction qui nous permet de chercher les tweets.
tweepy.Cursor(api.search,q=trend +'-filter :retweets',count=100, tweet_mode='extended', lang="fr").pages(10) :

- api.search :
- q : (champ obligatoire) contient la requête de recherche et un maximum de 500 caractères
- tweet_mode(extended) : le résultat inclut le texte intégral des longs tweets
- count : nombre de tweets à renvoyer par page, jusqu'à un maximum de 100. La valeur par défaut est 15.
- pages() :pour la pagination

Pour chaque tendance, nous avons paramétré l'API Twitter de la façon suivante :

- récupérer 100 tweets par pages : count = 100
- nombre de pages : pages = 10
- Nombre de tweets par tendance : count * pages = 100 x 10 = 1000 tweets

1000 tweets maximum à récupérer par tendance à chaque lancement du programme. Avec l'API Search de Twitter, nous collectons que les tweets des 2 dernières semaines.

Bien entendu, chaque tendance est réaprovisionnée de nouveaux tweets quotidiennement.

3.4.3 Processus

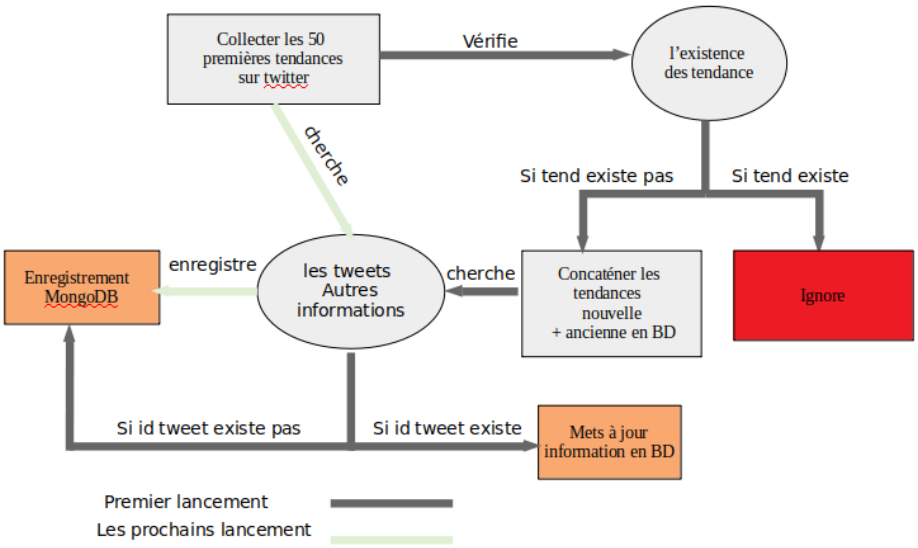


FIGURE 3.3 – processus de collecte des tweets

3.5 Flux RSS

3.5.1 Récupération des liens XML

Pour récupérer les XML, nous devons définir des URLs provenant de sites de news (Le Monde,L'Equipe ...) proposant des flux RSS. Nous catégorisons chaque URL en fonction du contenu qu'elle propose. Nous stockons les urls, manuellement, dans un fichier JSON.
Voici un aperçu du fichier de configuration « linkRss.json » :


```
{
   "politique":[
      {"rss": "http://www.lefigaro.fr/rss/figaro_politique.xml"},
      {"rss": "https://www.bfmtv.com/rss/politique/"},
      {"rss": "http://leplus.nouvelobs.com/tag/politique/rss.xml"},
      {"rss": "https://www.lepoint.fr/politique/rss.xml"}
   ],
   "sport":[
      {"rss": "https://www.lequipe.fr/rss/actu_rss.xml"},
      {"rss": "https://www.francetvinfo.fr/sports.rss"},
      {"rss": "https://www.france24.com/fr/sports/rss"},
      {"rss": "https://www.lemonde.fr/sport/rss_full.xml"},
        {"rss": "https://www.lemonde.fr/football/rss_full.xml"},
      {"rss": "http://sport24.lefigaro.fr/rssfeeds/sport24-accueil.xml"},
      {"rss": "https://www.lemonde.fr/jeux-olympiques/rss_full.xml"}
   ],
   "culture":[      ],
   "sante":[      ],
   "économie":[      ],
   "jeux videos":[      ],
   "education":[      ],
   "decouvertes":[      ],
    "cinema": [      ],
    "sciences": [      ],
    "planète":[      ],
    "series":[      ]
}
```


FIGURE 3.4 – fichier Json contenant les liens flux RSS

3.5.2 Récupération des fichiers XML

Nous cherchons à récupérer des ressources XML (disponible à partir des urls présentes dans «linkRss .json ») et extraire les titres, descriptions, la date de publication et la catégorie de l'article

Par exemple sur le lien : `http://www.lefigaro.fr/rss/figaro_politique.xml,onobtient:`

FIGURE 3.5 – exemple document XML

un exemple de données extrait du fichier xml ci-dessus :

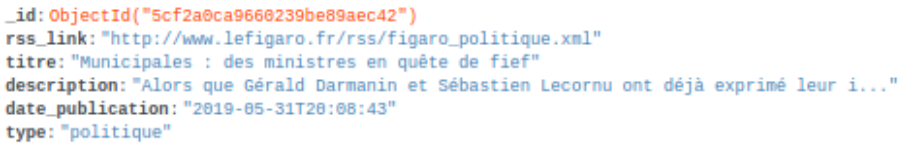


FIGURE 3.6 – exemple de donnée flux extrait des données XML

3.6 Collection :

une collection c'est l'équivalent d'une table pour une base de données relationnelle. Mais sauf dans notre cas, une collection n'a pas de schéma donnant la structure d'une collection.

Les trois collections utilisées pour le stockage des données

- tweets : collection pour stocker les informations sur les tweets liés à une tendance
- events : collection pour stocker les tendance(événements) et ses informations
- fluxRSS : collection pour stocker les données extraits du flux RSS

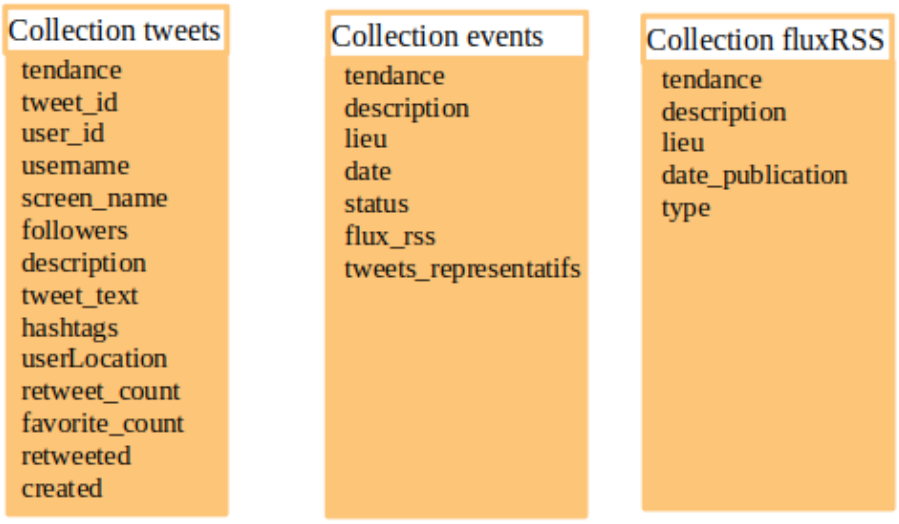


FIGURE 3.7 – les différentes collection pour notre base de données

Chapitre 4

Découverte de la description

Pour chaque évènement, il s'agira de découvrir une description adéquate le définissant.
Celle ci se base essentiellement sur les tweets récupérés par l'API et les Flux RSS dont nous disposons.

4.1 Analyse de la problématique

Notre projet étant un capteur d'événements sur le web, rappelons le contexte afin de mieux appréhender la suite de ce chapitre.

Un événement est un fait marquant de l'actualité et est défini par sa description, son lieu, sa date, et sa durée.
La description d'un événement doit permettre de comprendre les tenants et les aboutissants du fait traité grâce à une brève phrase, le titre d'un article de journal ou un tweet par exemple.

Pour illustrer ce qui vient d'être dit, donnons un exemple concret. Supposons que nous ayons la tendance twitter "Antonio Reyes", nous voyons clairement que ce nom ne signifie rien à lui tout seul, il faut donc rechercher une description pour cette tendance twitter dans les tweets liés à celle-ci et/où dans les articles de presse que nous détenons grâce aux flux RSS.

En effectuant une simple recherche google sur "Antonio Reyes", on se rend compte qu'il s'agit d'un joueur de football mort tragiquement et dont tout le monde parle, sur twitter et dans la presse. On trouve par exemple un article intitulé "Le monde du football rend hommage à José Antonio Reyes, disparu dans un accident de voiture".
Concrètement il s'agira de passer de la tendance "Antonio Reyes" à l'article cité précédemment, tout en évitant de relater un autre événement concernant Antonio Reyes, par exemple, sa titularisation lors d'un match précédent, ou la victoire de son équipe en championnat.
A travers ce schéma, nous pouvons apercevoir le cheminement de la tendance vers la description, qui fait comme suit :

- Récupération et envoi de la tendance vers l'application
- Traitements et échanges entre l'application et la base de données Obtention et ajout de la description dans la base de données

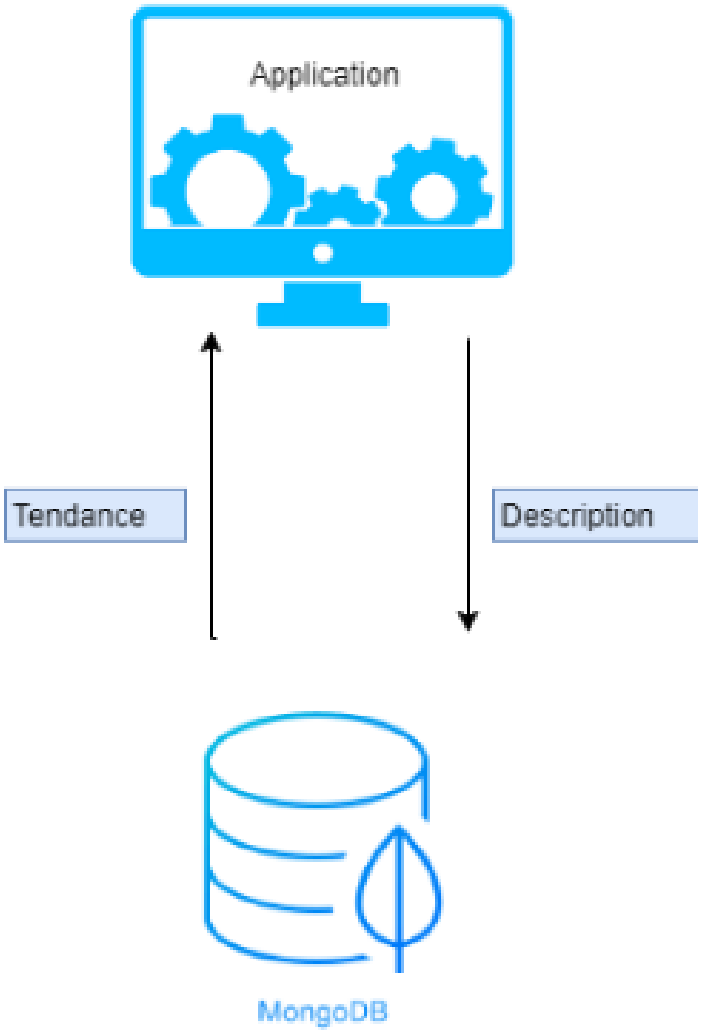


FIGURE 4.1 – Analyse de la problématique

4.2 Solution proposée et mise en oeuvre

4.2.1 Récupération des Tweets

La récupération des tweets se fait en deux étapes.

- Récupération des tendances
- Récupération des tweets de chaque tendance.

Ces informations sont récupérées de notre base de données, les requêtes se faisant en python et interrogeant notre base de données NoSQL MongoDB.

Nous ne faisons pas appel à l'API étant donné la limitation du nombre de requêtes par l'API et la latence d'obtention des résultats.
Il est donc bien plus efficace d'utiliser une base de données afin d'augmenter le temps de réponse et améliorer l'expérience utilisateur.

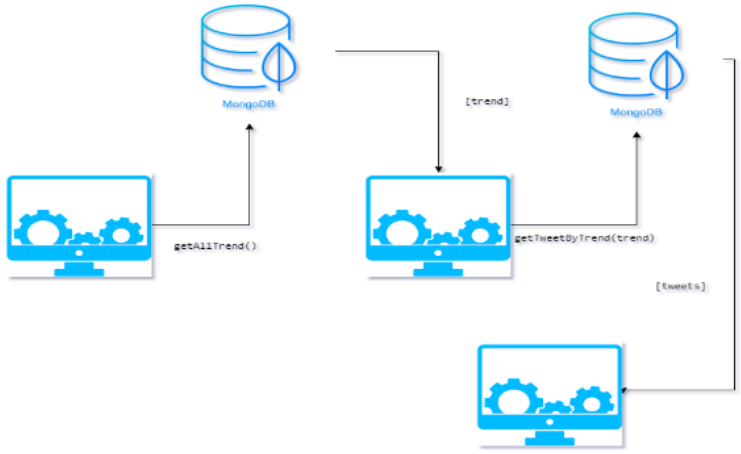


FIGURE 4.2 – Récupération des Tweets

4.2.2 Construction du texte à Analyser

Une fois les tweets récupérés, nous construisons un texte composé de la concaténation des tweets de chaque tendance.

Nous aurons pour résultat, un texte par tendance Twitter. Ce texte sera notre base d'analyse textuelle afin de détecter les descriptions des événements.

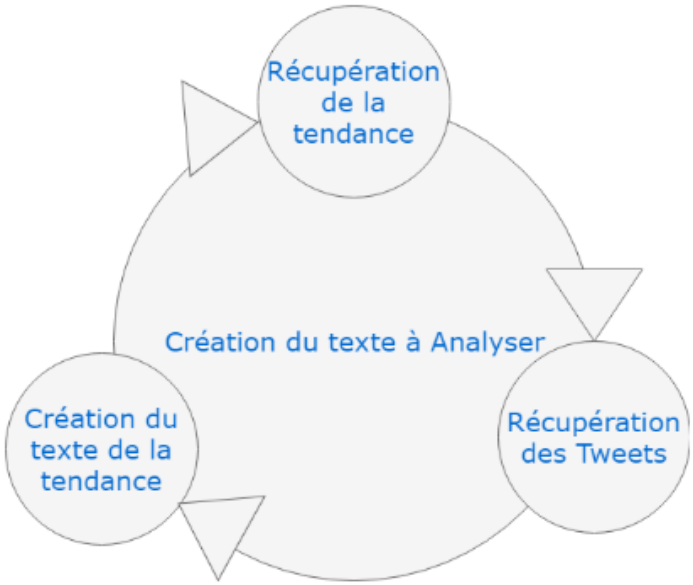


FIGURE 4.3 – Construction du texte à Analyser

4.2.3 Analyse du texte

Une fois ce texte récupéré dans une variable, il s'agira d'effectuer une analyse textuelle qui permettra d'en tirer un maximum d'information utiles et de construire une description de l'événement discuté dans ces tweets, si événement il y a.

Pré-processing du Text

A cette fin, nous avons utilisé une bibliothèque Python nommée NLTK [5] (Natural Language ToolKit) et pratiqué un certains nombre de traitements.

Tout d'abord, nous avons commencé par mettre tout le texte en minuscules afin d'éviter les problèmes de casse.
Ensuite, nous avons enlevé tous les mots vides (stop words) afin de ne pas obtenir dans nos résultats

de mots clés inutiles tels que ‘Le’, ‘La’, ‘ce’ et tout autre mot commun de la langue Française. Pour ce faire, nous avons utilisé la liste de mots vides fournie par la bibliothèque NLTK et en avons rajouté par nous même pour la compléter après avoir repéré empiriquement l’apparition de certains mots qu’on a considéré trop communs dans les résultats, comme : ‘Alors, Comment, Dedans’ etc...
Ce traitement nous a permis d’obtenir les mots clés les plus pertinents pour chaque évènements en évitant que ceux-ci soient étouffés par des mots vides.

Une fois ces traitements faits, nous avons ensuite enlevé tous les caractères non alphabétique ou non numérique afin de supprimer les émoticônes, les virgules, les underscore, tirets etc.. afin qu’ils ne soient pas pris en compte durant notre analyse comme étant des mots.

Nous avons ensuite décidé de faire de la racinisation [11], ce qui consiste à faire passer un mot de sa forme fléchie vers son radical (ou racine).
Les flexions sont les différentes formes fléchies d’un même mot. Les formes fléchies correspondent aux formes “conjuguées” ou “accordées” d’un mot de base non conjugué et non accordé : le Lemme.
Exemple : “Jouera” “Jouer”

La racine d’un mot correspond à la partie du mot restante une fois que l’on a supprimé son préfixe et son suffixe, à savoir son radical. Elle est aussi parfois connu sous le nom de stemme d’un mot. Contrairement au lemme qui correspond à un mot réel de la langue, la racine ou stemme ne correspond généralement pas à un mot réel.
Exemple : “Jouer” “Jou”
Ceci nous a permis d’obtenir de meilleurs résultats et perdre moins d’informations en prenant en compte les mots [‘jouer’, ‘jouera’, ‘joué’] comme étant un seul mot et une seule signification.

Nous avons aussi tester une lemmatisation, qui consiste à transformer les déclinaisons d’un lemme en un seul mot. En d’autres termes, le processus de lemmatisation transforme ’nettoyait’ et ’nettoyant’ en un seul mot : ’nettoyer’.
Les résultats de ce test ont été moins probants, ceci est explicité dans les tests et certifications.

Au delà de ces traitements, nous avons aussi effectué un filtrage statistique sur les tweets.
Nous avons éliminé les tweets ayant trop peu de retweets, ou issus de comptes ayant trop peu de followers, les considérant comme non fiables, et ceci afin d’avoir une meilleur qualité de données à traiter et diminuer des données à traiter, ce qui a pour effet d’augmenter la vitesse de calcul.

Dans la même optique, nous avons évalué l’orthographe des tweets et éliminé les tweets ayant un pourcentage de mots mal-orthographiés trop élevé.

FIGURE 4.4 – Pré-processing du Text

Une fois le pré-processing effectué, nous obtenons un texte nettoyé et prêt à être analysé.

Méthodes et Algorithmes d'analyse

L'objectif de notre analyse de texte dans cette partie est de découvrir la description de l'événement traité par le texte donné (concaténation des tweets).

Pour atteindre cet objectif, nous allons rechercher les mots les plus pertinents/importants du texte qu'on nommera 'mots clés', l'idée étant de les considérer comme étant les mots représentatifs du texte et donc nous permettre d'en découvrir le sujet et de le décrire.
Explicitons notre méthode de recherche de mots clés.

Nous avons recherché différents types de mots clés, des mots clés uniques, composés d'un seul mot comme 'Concert' ou des mots clés de longueur plus élevée, les n-grammes.
Un n-gramme est une sous-séquence de n éléments construite à partir d'une séquence donnée [7].

Exemple : "New York" ou "Grumpy Cat"
Utiliser des N-grammes nous permet de ne pas passer à côté de certains mots composés ou suite de mots pertinents et souvent répétés. S'est ensuite posé la question de la longueur des n-grammes, et nous avons défini empiriquement la longueur maximum d'un n-gramme à prendre en compte était de 5, car au delà de 5 mots, les n-grammes perdaient de leur pertinence alors que le temps de calcul augmentait fortement.

Pour découvrir ces mots clés, nous avons implémenté l'algorithme TF [12] afin de découvrir les mots apparaissant le plus dans le texte. Nous avons effectué la même chose avec les n-grammes.
Nous avons appliqué TF aux tokens de différentes longueurs séparément afin de trouver pour chaque longueur les mots clés apparaissant le plus.
Plus en détail, nous appliquons 5 fonctions sur le même texte, le rôle de chaque fonction est de retourner une liste de tous les tokens avec leur pourcentage d'apparition par rapport au nombre de tweets.

Chacune de ces fonctions prendra le texte et le nombre de tweets en entrée puis le divisera en groupe de mots de différentes longueurs, et calculera leur taux d'apparition dans le texte.
Les différentes fonctions sont :

- TF(text, nbTweets) pour les mots clés de longueur 1
- bigrams(text, nbTweets) pour les mots clés de longueur 2

- trigrams(text, nbTweets) pour les mots clés de longueur 3
- fourgrams(text, nbTweets) pour les mots clés de longueur 4
- fifthgrams(text, nbTweets) pour les mots clés de longueur 5

Concernant les tokens de longueur 1 il suffit de diviser le texte en fonction des caractères 'espace' et de la ponctuation, ce qui se fait facilement avec la fonction "word_tokenize()" de la librairie NLTK. Néanmoins, pour les mots de longueur supérieure à un, il a fallu implémenter nous même la façon de créer les tokens, nous avons donc créé nos tokens en concaténant les mots adjacents de la liste retournés par le tokenizer en les séparant par un espace.

Maintenant que nous avons expliqué comment nous divisons notre texte en tokens , explicitons comment fonctionne notre implémentation de TF.
L'algorithme se divise en quelques étapes simples

- Diviser le texte après pré-processing en Tokens
- Calculer la fréquence d'apparition de chaque Token
- Diviser cette fréquence par le nombre de tweets pour avoir un pourcentage d'apparition.

Exemple en python :

```
def TF(text, nbTweets):
    nb = nbWords(text)
    listWords = countEachWord(text)
    for word in listWords :
        listWords[word]=listWords[word]/nbTweets
    print("This is TF")
    return listWords
```

FIGURE 4.5 – Algorithmes d'analyse

Nous appliquons exactement le même algorithme aux tokens de longueurs différentes puis nous les ajoutons dans une seule liste contenant tous les tokens et leur pourcentage d'apparition multiplié par un coefficient pour éviter l'iniquité dûe à la longueur (les tokens de longueur plus petite apparaissant plus souvent).

Il est nécessaire de préciser que nous ne retenons que les mots clés ayant un taux d'apparition jugé suffisamment élevé, pour cela nous avons défini un seuil minimum d'apparition, choisi empiriquement à 10%.
C'est ce qui explique que certains tweets ne sont définis par aucun aucun mot clé.
Ce seuil nous permets de ne retenir de mots clés et donc de définition que pour les tendances ayant des tweets proches et décrivant donc un événement, car si ce seuil n'est pas dépassé, on considère que l'événement est mal défini au vu de la disparité des tweets.

Pour résumer, notre méthode s'applique comme suit :

- Récupérer le texte après pré-processing
- Appliquer les 5 méthodes pour les tokens de différente longueur
- Rassembler les résultats dans un même vecteur (contenant les tokens et leur pourcentage d'apparition)
- Utiliser ce vecteur pour un dernier nettoyage et une sélection de résultat

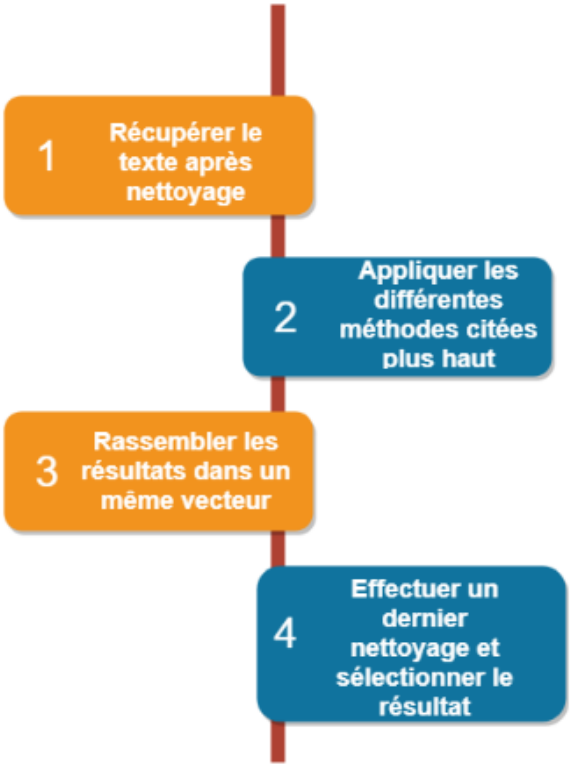


FIGURE 4.6 – analyse de text

Ce dernier point est explicité dans ce qui suit.

Post-Processing et choix du résultat

Une fois le vecteur contenant tous les résultats intermédiaires créé, nous allons appliquer quelques traitements afin de nettoyer ces résultats et pouvoir sélectionner les mots clés pertinents.
Nous commençons par choisir les 10 mots clés ayant le meilleur score dans le tableau des résultats, et les mettons dans une liste.

Nous remarquons que certains de ces mots clés en englobent d'autres par exemple, "Grumpy Cat" et "Cat", avec une fréquence d'apparition forcément plus élevée pour "Cat", il faut donc éliminer le mot "Cat" de la liste des résultats potentiels au privilège de "Grumpy Cat" qui l'englobe.

Pour ce faire, la fonction 'deletesubstr(list)' a été créé afin de supprimer les sous-chaînes apparaissant dans des chaînes plus grandes, ceci nous évite d'avoir des répétitions inutiles dans nos mots clés

résultants et permet une meilleure recherche d'articles à posteriori.

Une fois ces sous-chaînes supprimées nous supprimons les répétitions inutiles du résultat final, par exemple si on a le résultat suivant : "Grumpy Cat Dead Cat", nous supprimons la deuxième occurrence de cat, car elle est inutile et est nocive pour l'utilisation qu'on fera du résultat, nous obtenons donc "Grumpy Cat Dead".

Une fois ce résultat obtenu, nous avons des mots clés représentatifs de notre texte, néanmoins, cela ne forme pas forcément une description, pour ce faire, nous allons rajouter une étape à notre méthode.
Cette dernière étape est la recherche dans les tweets et dans les flux rss d'un article/tweet représentatif de ces mots clés, lequel nous servira de description pour notre texte.
Cette recherche se fait par des algorithmes de calcul de similarité comme l'algorithme Jaccard (ici 5), qui se base sur un calcul de distance, qui s'écrit comme suit :

$$\text{Jaccard}(X, Y) = \frac{|X \cap Y|}{|X \cup Y|}$$

FIGURE 4.7 – Jaccard

Soit le nombre de mots apparaissant dans les deux texte par rapport au nombre de mots total. Nous recherchons donc dans notre base de donnée le tweet et l'article de presse le plus proche de nos mots clés et nous l'utilisons comme description.

FIGURE 4.8 – Post-Processing

4.3 Tests et certification de la solution

En lançant notre recherche de descriptions nous remarquons que le résultat obtenu pour cet évènement est assez pertinent

Méthode	Bons	Mauvais	Vide	Fiabilité	Temps
Sans pré-processing	48	44	8	52%	6000
Avec pré-processing	70	20	10	78%	4500
En filtrant les tweets par pertinence	33	10	57	77%	3000
Avec le Stemming et le filtrage	38	7	55	84%	6000
En ajoutant le filtrage durant la recherche des tweets représentatifs	38	7	55	84%	2700
Avec le Lemmatizer	23	27	50	48%	3100
En utilisant un tweet représentatif pour la recherche d'article	10	40	50	20%	3180

Grâce aux tests effectués précédemment, nous voyons une évolution du temps de calcul ainsi que de la fiabilité qui atteint sa plus haute valeur lors du filtrage des tweets par pertinence.

Développons ce qui a été fait.
Tout d'abord, nous avons testé l'analyse de texte sans aucun pré-processing, le texte était traité tel qu'il était reçu au départ de Twitter.
Les résultats résultants étant peu fiables (50% de réussite), nous avons alors décidé de traiter le texte comme expliqué précédemment (Résultats avec pré-processing).

Les résultats une fois le texte traité nous obtenons un résultat plus intéressant au niveau de la fiabilité qui passe de 52% à 78% soit presque 50% d'amélioration ce qui est un progrès certain.
Cette amélioration est due au fait que nos tweets sont modifiés et son exempt de tout mot vide et de tout caractère non alphanumérique.
En ce qui concerne le temps de calcul nous avons gagné 1500 secondes soit 25 minutes (25% d'amélioration de gain de temps) pour les 100 tendances à traiter.
Ceci est possible grâce au fait que le nombre de mots est nettement plus bas suite à la suppression des mots vides et des caractères non alphanumériques, il y a donc moins de mots à parcourir pour comptabiliser les occurences.
Ensuite, nous avons tenté de perfectionner les résultats en précision et en temps de calcul en faisant une sélection des tweets comme expliqué ci-après.

Une fois le pré-processing fait, nous sommes passés à l'étape suivante en filtrant les tweets par pertinence, nous avons évalué la pertinence d'un tweet grâce à trois paramètres qui sont les suivants :

- Nombre de followers du compte ayant tweeté
- Nombre de retweets
- Pourcentage de mots mal orthographiés.

Nous avons empiriquement découvert que le meilleur compromis était :

- 100 000 followers
- 5 retweets
- 30% maximum de mots mal orthographiés

Nous remarquons que la fiabilité est la même, ceci dit, il y a beaucoup moins de descriptions trouvées, on passe de 10 descriptions non trouvées à 57 ce qui est causé par la sélectivité des tweets qui a été faite et au treshold d'apparition (pourcentage d'apparition d'un mot par rapport au nombre de tweet).
Néanmoins, au niveau du temps de calcul, nous gagnons encore 1500 secondes, ce qui est 50% d'amélioration cette fois ci.

Par la suite, nous avons implémenté la racinisation, pour celle ci, les résultats ont été moins probants.
Le taux de fiabilité à augmenté de 8% ce qui est du au stemming de certains mots qui fait que le radical à plus d'importance et qu'on retrouve plus facilement le sens des tweets.
Le temps de calcul est remonté à a peut près 6000 secondes.
On en conclut que le stemming combiné au filtrage n'a pas totalement apporté ces fruits. Nous avons

donc tenté de rajouter un filtre au niveau des tweets représentatifs

En ajoutant ce filtre le taux de fiabilité est resté le même, les résultats sont sensiblement identiques, ce qui est causé par le fait que les tweets représentatifs sont des tweets correspondants au filtre rajouté, soit, avec beaucoup de followers et de retweet. A contrario nous avons eu un gain de temps très élevé, ce qui est du que le calcul de similarité se fait avec beaucoup moins de tweets et donc, avec une vitesse plus élevée. On en conclut que cette solution est donc plus intéressante.

Une fois ceci fait, nous avons tenté d'utiliser un lemmatiseur au lieu d'un stemmeur, les résultats ont été peu probants, la fiabilité est très basse même si le temps de calcul est assez rapide.
Ce peu de fiabilité est du au fait qu'en lemmatisant les mots, il y a différentes possibilités de résultat, prenons un exemple, le mot "avions", ce dernier peut avoir comme résultat le verbe "avoir" ou le nom "avion" ce qui a pour conséquence nombre d'erreurs dans nos résultats de lemmatisation et donc, une baisse de la fiabilité.

Pour finir nous avons tenté de rajouter les tweet représentatif de l'événement comme paramètre de la recherche d'articles de presse (dans les flux RSS), ceci a donné de très mauvais résultats.

Voici un graphe résumant ces différents tests.

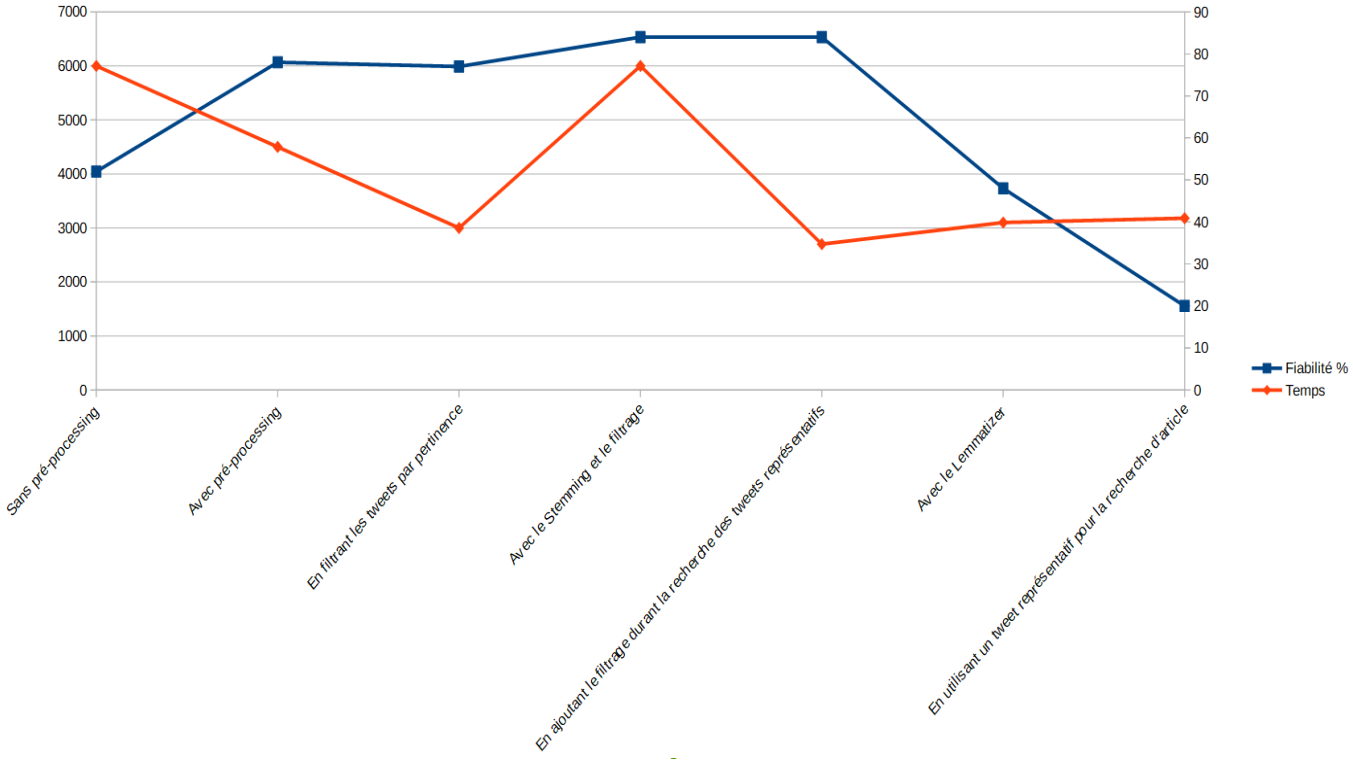


FIGURE 4.9 – Graphe de synthèse des tests de recherche de Description

On peut conclure de ce graphe assez rapidement que la meilleure option à prendre est celle du stemming, en utilisant un filtre sur les tweets avant la recherche de mots clés et avant la recherche du tweet représentatif.

Chapitre 5

Découverte du lieu

Pour chaque évènement, il s'agira aussi de découvrir un lieu adéquat le définissant.
Celle ci se base essentiellement sur les tweets récupérés et sur la base de données Geonames.

5.1 Analyse de la problématique

On souhaite de mettre en place un système permet de détecter les lieux d'un document textuelle. Il est capable de répondre aux utilisateur les informations liée aux lieux trouvés ainsi que assurer la fiabilité de résultat avec une performance de processing assez rapide. Il est aussi préférable de savoir l'effet de l'événement pour prédire que si l'événement se passe au niveau globale ou c'est un événement locale identifié que les gens en parlent. Par conséquent, la relation entre des information de lieux est un facteur important pour pouvoir donner une décision sur la globalité d'événement. Le stockage et l'analyse profondément de données devient donc les fonctionnements cruciaux de notre système
Les données de géolocalisation sont souvent complexes. En effet, pour un lieu, il contient non seulement les informations de lui-même mais aussi celles de son parents ou de ses fils, c'est à dire autre lieu qu'il appartient. L'analyse de résultat est fait avec prudence en raison de le volume de données qu'on obtient. D'ailleurs, certaines données situé dans le champ de résultat sont toujours pas évidents et compréhensible par humain, il s'agit tout simplement un clé permettant de lier aux autres base de données pour récupérer l'information manquée. Si on voudrait obtenir un résultat complète et lisible, il faut alors réaliser plusieurs opérations au dessus. Tel opération peut être très coûteux en terme de temps d'accès si on part avec une base de données relationnelle classique. Tout cela plus la quantité de données à traiter nous laisse pas le choix que aller avec la solution d'utiliser la base de données de type NoSQL
Le développement du programme s'effectue sous différents environnement. Particulièrement pour apprendre rapidement le fonctionnement de la base données géolocalisation proposée par des solutions existants, on préfère d'interroger directement la base de données par les requêtes GET qui se regroupe les paramètres et l'information cherché alors que durant le développement on utilise différent langage de programmation comme Python, php pour effectuer la requête. Il faut chercher un moyen flexible pour que la base de données soit accessible depuis n'importe quel application.
Pour les problématique mentionnés ci dessus, l'utilisation un web service existant semble le plus convenable. En plus, les webservices apportent un avantage sur le traitement de données sous prétexte qu'on peut choisir son format entre XML et JSON. Ce sont des formats standard, clair et bien structuré pour récupérer et analyser. Par contre, la solution de web service vient avec des contraintes qui pourra empêcher l'utilisation, c'est la limitation de requête qu'on peut effectuer dans une intervalle de temps. Ce contrainte existe parce que le programme de géolocalisation demande beaucoup de ressource sur le coté serveur ce qui coûte cher en maintenance. Le fait que les données sont très énormes et complexe pour accéder et récupérer nécessite une ressource assez grande en terme de stockage et de mémoire pour assurer la performance. Si on souhaite enlever ces limitations, il doit payer et le prix n'est pas justifiable pour ce qu'on a besoin. Même si on paie, il augmente juste le nombre de requête par jour, finalement le problème reste toujours là.

5.2 État de l'art

Geonames API

Au départ, on utilise l'API Geonames, il s'agit d'un webservice REST permettant d'effectuer les recherches de lieux. Les données de Geonames ont été construit de centaine de source différent pour proposer des informations riches et fiables. C'est un outil super puissant et complet pour notre besoin. Surtout l'utilisation de Geonames est complètement gratuit. Des fonctionnalités principaux de Geonames est comme suit :

- Chercher l'information d'un lieu à partir du mot clé
- Chercher la voisine de lieu indiqué (hiérarchique)
- Chercher les lieux connue (touristique)
- Chercher les villes en fonction de la population

Malheureusement pour l'utilisation non-commercial de la version open-source, Geonames met un contraint de 1000 requêtes par heure et 20000 requêtes par jour. Pour le traitement de données avec une quantité très large, clairement ça suffit pas et on atteint rapidement la limitation. C'est pour cela qu'il faut chercher une autre solution

5.3 Solutions proposées et démarche

5.3.1 Création d'une base de données personnalisée

Heureusement, Geonames [6] nous fournit les données de localisation partout dans le monde sous forme des fichiers texte. Ils sont catégorisés par plusieurs critères de recherche adaptés aux besoin différents. En revanche, les données trouvés dans ces fichiers ne sont pas structurées, on a besoin de transformer les données à un format plus uniformisé pour faciliter l'analyse et l'utilisation. Les résultats après le traitement de données sont ensuite envoyés vers un système de stockage de choix, particulièrement dans une base de données NoSQL. Cela permet non seulement d'avoir plus de contrôle sur la gestion de données mais aussi d'optimiser le temps d'accès et le temps de réponse pour les demandes de haute fréquence.
En outre, le choix de stocker dans une base de données remporte des grandes avantages, voici une liste non-exhaustive d'avantage par rapport aux base de données relationnelle :

- Les données sont distribuées sur plusieurs machines (sharding) de ce fait on évite les goulets d'étranglements lors de la récupération des données (fortes performances de lecture)
- L'évolutivité se fait de manière horizontale (pour augmenter les performances on ajoute des nouvelles machines)
- La base de données qu'on utilise dans le projet Elasticsearch [9] est de famille orienté document dont la particularité est de pouvoir indexer des documents fortement orientés textes. On peut profiter la puissance de l'indexation en ciblant les balises du document et ainsi bénéficier d'une interrogation simplifiée
- La représentation des données est notable par l'absence de schéma. On évite l'ennuie de définir le mapping de données comme la base de données NoSQL

On choisit la suite Elastic qui est basée sur le principe ETL(Extract, Transform, Load). Elastic se compose de plusieurs solutions, le plus connues c'est le fameux ELK (Elasticsearch, Logstash, Kibana). Chaque élément dans la suite est conçu pour un rôle différent.

- **Elasticsearch** : Un moteur de recherche et d'analyse RESTful distribué, conçu pour répondre à une multitude de cas d'utilisation. Il joue le rôle de système d'indexation, de moteur de recherche ou bien de collecteur de données
- **Logstash** : Il s'agit un pipeline de collecte de données, destiné au traitement des données. Il est capable de filtrer des données grâce aux plugins intégrés, structurer puis les envoyer vers un système de stockage ou bien un système d'indexation. On distribue bien sur vers Elasticsearch dans Cloud pour stocker la sortie structurée par Logstash dans les index.
- **Elastic Cloud** : Une solution de héberger et de gérer Elasticsearch en ligne [8]. Cela nous permet de déployer et d'accéder les données à distance

Le principe de fonctionnement de la suite Elastic

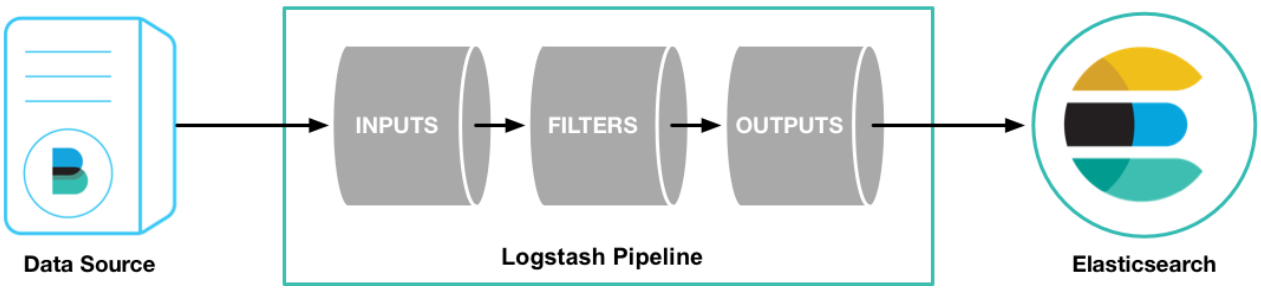


FIGURE 5.1 – Principe de fonctionnement de la suite Elastic

Pour fonctionner, Elasticsearch aura donc besoin de savoir quels mots sont employés dans chaque document. Pour cela, Elasticsearch intègre un moteur Lucene qui va s'occuper d'extraire les mots d'une collection de documents et de préparer des colonnes de mots. Un mot s'agit une colonne de documents avec le poids du mot dans chacun des documents. La couche de distribution effectuée par Elasticsearch permet de router les requêtes, paralléliser les traitements, répliquer les données en cas de panne et augmenter la capacité d'indexation de Lucene.

La photo ci-dessus décrit la conception. Chaque branche est un serveur Lucene s'occupe une partie des documentes.

FIGURE 5.2 – Architecture d'Elasticsearch

Prise en charge de données sur Elasticsearch
Le fichier texte téléchargé de Geonames n'est pas structuré. Chaque ligne contient les informations d'un lieu et elles se sépare par le caractère tab. Il suffit d'utiliser Logstash pour structurer les données. On apprend premièrement le mapping de champs indiqué dans la documentation de Geonames. Puis on prépare le fichier de configuration pour Logstash.
Le fichier de configuration défini les comportements de Logstash. En d'autre terme, les fonctionnalités se décompose en 3 partie : input, filter, output
On met l'emplacement de fichier Geonames dans input. La moyen de traiter et de transformer les données sera défini dans filter. Enfin la sortie de stockage de données transformées se trouve dans output.
La photo suivante représente la partie Input

```
input {
  file {
    path => "/Users/phucvu/Desktop/elastic/logstash-7.0.1/textFile/allCountries.txt
    start_position => "beginning"
  }
}
```

FIGURE 5.3 – Partie Input de Logstash

Filter de Logstash est illustré comme la photo ci dessus

```
filter {
  csv {
    separator => "	"
    columns => ["geonameid", "name", "asciiname", "alternatenames",
    "latitude", "longitude", "featureClass", "featureCode","countryCode",
    "cc2", "admin1Code", "admin2Code", "admin3Code", "admin4Code",
    "population", "elevation", "dem", "timezone", "dateMod"]
    quote_char => "£"
  }
}
```

FIGURE 5.4 – Partie Filter de Logstash

On utilise le plugin csv intégré dans Logstash. Ce plugin permet de transformer les fichiers suivi la séparation de contenu par un caractère. On déclare donc le caractère tab comme déliminateur et tous les champs renseignés de la documentation Geonames

Output de Logstash est comme suit

```
output {
  elasticsearch {
    hosts => ["https://487a3c18579448be987a324822344a49.eu-central-1.aws.cloud.es.i
    index => "geoname-%{+YYYY.MM.dd}"
    user => "elastic"
    password => "ZsCkeJa6qrPR6YICH8nwCRZR"
  }
}
```

FIGURE 5.5 – Partie Output de Logstash

Ici on précise tout simplement l'authentification de Elastic Cloud pour router les données sorties vers la base de données Elasticsearch hébergée en ligne

On effectue le même traitement pour la base de données alternatename. Cette base de données contient les noms alternatif d'un lieu.

5.3.2 Traitement de texte

On aperçoit que dans la vie réelle, les lieux est souvent écrit par les mots qui commence par une lettre en majuscule. On cherche alors dans un texte toutes les combinaisons suivies ce règle. L'utilisation de l'expression régulière [10] nous permet de récupérer toutes les combinaison facilement. Elle se décrit comme suit :

```
([A-Z][A-Za-zàéèêëîïôûüÿù]+(\s+[A-Z][a-zàéèêëîïôûüÿù]+)*)
```

Toutes les combinaisons satisfait l'expression :

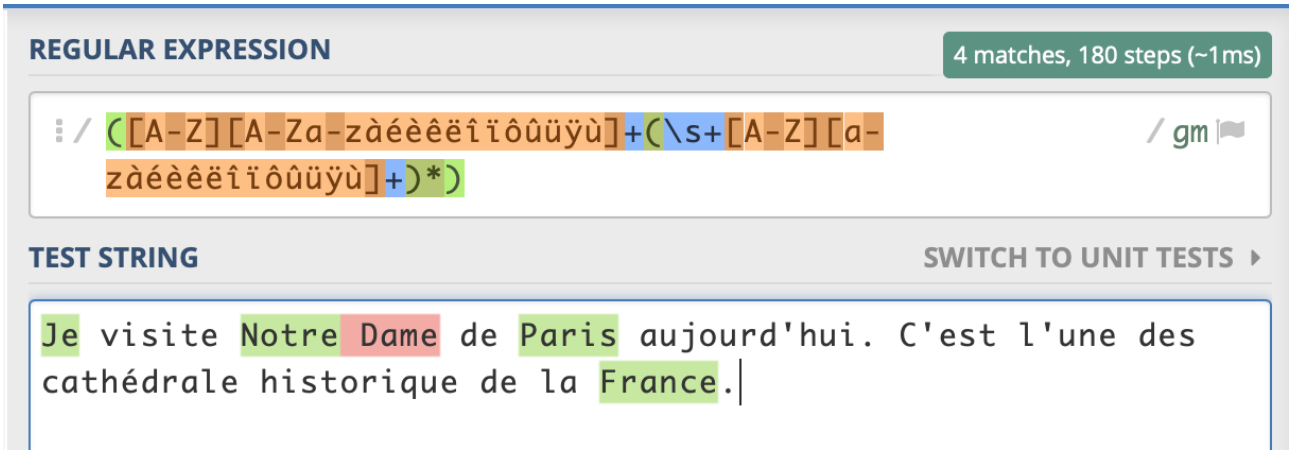


FIGURE 5.6 – Le matching de l'expression régulière

Cependant, il ne fallait pas prendre en compte les mots en majuscule après certains caractère tel que « . », « , ». Ces mots peuvent éventuellement fausser et affecter la pertinence de résultat. On met tout simplement les mots après les caractères spéciaux en minuscule.
Vu que le traitement de texte est en français, il y a sans doute des mots grammaticaux qui n'a pas utilité mais pollue le résultat de recherche des lieux. En effet, le sens de la phrase dans la recherche des lieux ne nous concerne pas, on évite donc ces mots en majuscule qui a avec une forte probabilité de répétition dans le document et qui prend la place le privilège des vrais mots de lieux
On s'intéresse également les hashtags lors d'un traitement de tweet pour Twitter. Les gens souvent indique les informations importantes notamment pour celles de lieux. D'ailleurs, Twitter l'utilise pour déterminer une tendance, même s'il n'y pas d'information de lieux, c'est bien de prendre en compte. La technique de traitement est assez simple, on supprime le hashtag puis on transforme la première lettre en majuscule. Il tombe alors dans le bucket de traitement des mots en majuscule.
A l'aide de l'expression régulière, on extrait les mots respectés la convention ci-dessus. Pour calculer la pertinence d'un mot, on enregistre l'occurrence d'un mot dans le document. Le mot et son occurrence seront stockés dans un tableau. On parcourt tous les mots trouvés par l'expression régulière, on incrémente une valeur de 1 pour l'occurrence si on tombe sur le même mot.

5.3.3 Analyse de lieux

Selon la définition de données de l'index geonames dans Elasticsearch, un lieu contient les informations suivantes

- geonameid : l'identifiant de lieu à travers toutes les indices
- name : le nom de lieu
- alternatenames : le nom alternatif de lieu
- featureClass : type de lieu (pays, villes, route, immeuble, monument, etc)
- featureCode : explication précise sur le type de lieu
- countryCode : code ISO de pays en deux lettre
- admin1Code : code pour la première division administrative
- admin2Code : code pour la deuxième division administrative

Sur Elasticsearch, on stocke également l'indice alternatename qui s'agit la base de données de nom alternatif en différents langages, la structure est la suivante :

- geonameid : l'identifiant de lieu à travers toutes les indices
- name : le nom de lieu
- isoLanguage : code ISO de langue dont qu'on souhaite de récupérer

Pour le pays, c'est possible qu'on n'ait pas d'information sur admin2Code.
D'après la structure d'un lieu, sauf pour les pays et les villes, on remarque qu'il possède l'information de villes et pays auxquels il appartient. La plupart du temps dans le tweet, les gens écrivent le nom de lieu et la ville où se trouve le lieu. Par contre, pour savoir s'ils parlent d'un lieu d'une même ville ou

dans des villes différentes, c'est à nous de vérifier. On peut très bien avoir par exemple « Notre Dame » comme nom de lieu discuté mais une personne parle de Notre Dame de Paris, une autre parle de Notre Dame de Rennes et encore une autre personne parle de Notre Dame qui ne se trouve pas dans le même pays, à autre continent.
L'objectif est donc de trouver les lieux et en même temps de repartir les lieux dans trois sections spécifiques : Pays, Ville, Place. Ça nous permet à la fin de trouver une sorte de relation entre les éléments, on déduit ensuite si le sujet aborde physiquement d'un lieu ou des lieux différents. L'indication pour la déduction est le pays et les villes de la place, c'est pour cela on favorise la recherche des pays et des villes en première pour les comparer avec celles extraites de places.

5.3.4 Recherche de pays et de villes

L'indice geonames sera utilisé pour la recherche de pays et de ville. En l'occurrence, il suffit de préciser « P » et « A » dans le champ de recherche featureClass. Cependant la base de données geonames fournit que des données standardisées en anglais alors que le langage écrit dans le document est en français, on a besoin une couche intermédiaire pour traduire des mots en paramètre de français en anglais pour que l'indice geonames comprenne l'entrée puis reforme la requête. Il consiste à utiliser l'indice alternatename et mettre « fr » comme iso pour la langue française dans le champ isoLanguage. Pour former la requête, on met également le nom de lieu cherché pour le champ alternatename. Le type de query est « term », il permet de prendre les résultats qui correspond exactement à 100% le mot cherché. Le code pour créer la requête est comme suit

```
alternatenameSearch = Search(using=es, index="alternatranslateSearch)
alternatenameSearch.query("term", alternatename__keyword=name)
translateSearch = translateSearch.query("term", isoLanguage__keyword='fr')
```

Les résultats obtenus par la requête de l'indice alternatename sont stockés dans un tableau et ensuite transférés comme paramètre pour faire la requête sur l'indice geonames. Pour assurer l'exactitude de résultat retourné par alternatename, on ne prend pas la traduction en anglais mais on prend plutôt le geonameid de chaque résultat trouvé. On s'effectue la recherche pour chaque geonameid en vérifiant la nature de lieu. Si elle n'est pas de type pays ou ville, on les ignore. La base de données est sans doute très complète mais en exchange elle contient beaucoup de données qui ne sont pas fiables. Il s'agit des lieux pas connus ou des lieux avec des informations incomplets. Ce problème peut être résolu facilement en filtrant la population qui est autre que 0. Le code décrit la requête de geonames est donc

```
geonameInfoSearch = geonameSearch.query("term", geonameid__keyword=geonameid)
geonameInfoSearch = geonameInfoSearch.query("terms",
featureClass__keyword=['P', 'A'])
geonameInfoSearch = geonameInfoSearch.exclude("match", population="0")
geonameInfoSearch = geonameInfoSearch.exclude("match", featureCode="ADM2")
geonameInfoSearch = geonameInfoSearch.exclude("match", featureCode="ADM4")
geonameInfoSearch = geonameInfoSearch.exclude("match", featureCode="ADM3")
```

On récupère enfin la réponse de l'API formaté en JSON et stocke dans un tableau de type dict dans Python.

Jusqu'à ici, on a un tableau de tous les pays identifiés et toutes les villes mélangées ensemble. Ces données sont utiles non seulement dans le sens de trouver la relation entre les places mais aussi de booster la pertinence de la recherche de place. En effet, l'information de pays et ville joue comme un facteur pour privilégier les places où se trouve les pays et les villes déjà trouvé

L'étape suivant consiste à séparer l'information de pays et de ville de table obtenue par la recherche. La donnée de pays sera stockée dans un tableau dédié uniquement pour le type pays, on fera le même travail pour les villes. On récupère l'information utiles comme nom, codePays, codeAdmin.

5.3.5 Recherche de place

Une fois la répartition de pays et villes est fait, on commence à chercher les places. On fait la requête tout d'abord sur l'indice geoname. Puisque on avait déjà l'information de pays et de ville, on met alors une condition pour filtrer les lieux autre que les places, c'est-à-dire le type n'est pas A ou P. La requête pour la recherche de place se constitue de trois principaux termes : must, mustnot et should. La terme must impose sur la requête doit satisfaire la condition alors que la terme should a pour le rôle d'augmenter la pertinence de résultat. Au contraire, la terme mustnot s'ignore les résultats dont matchent la condition du terme.
Comme on avait dit, le tableau de pays et le tableau de ville sert à booster la pertinence de résultat, l'information de ce tableau est considérée comme condition pour former la terme must not. Il faut parcourir le tableau et affecte la valeur au champ correspondant. L'exemple suivant représente la formation de terme should avec les informations trouvées pour les pays :

```
for i in range(len(dictCityCountry['country'])):
    query = Q('match', countryCode=dictCityCountry['country'][i])
    if query not in shouldQuery:
        shouldQuery.append(query)
```

La condition pour la terme must

```
mustQuery = [Q('term', name__keyword=name)]
```

La condition pour la terme must_not

```
%mustnotQuery = [Q('match', featureClass='P'), Q('match', featureClass='A')]
```

A la sortie de la requête, on a le résultat correspondant exactement au mot clé cherché. Et si on a des cas particuliers ? Par exemple :
La cathédrale Notre Dame de Paris, dans la vie courant quand les gens en parlent, ils mentionnent souvent le nom Notre Dame tout court, si on fait la recherche dans geonames, il va retourner un faux résultat qui porte le nom Notre Dame alors que le lieu qu'on voulait vraiment c'est Notre Dame de Paris. Pour éviter ce genre de problème, on vérifie si la requête avec la place cherchée donne un résultat dans l'indice alternatename, si c'est le cas on préfère de prendre plutôt celui-là au lieu de celui de geonames.

5.4 Tests et certification de la solution

5.4.1 Prise en charge de donnée text Geonames dans Elasticsearch

Le fichier text geonames contient 11908046 lignes qui présente les données de lieu disponible. On utilise Logstash de la stack Elastic pour lire le fichier text et transfère les informations structurée Elasticsearch Cloud pour l'indexation. On obtient le résultat suivant pour une bonne importation

```
{
  "_shards": {
    "total": 2,
    "successful": 2,
    "failed": 0
  },
  "stats": {
    "uuid": "JyIarL8EQvWpc74sokCpkA",
    "primaries": {
      "docs": {
        "count": 11908046,
        "deleted": 0
      },
      "store": {
        "size_in_bytes": 8457883657
      },
      "indexing": {
        "index_total": 11908046,
        "index_time_in_millis": 10995621,
```


FIGURE 5.7 – Statut de la prise en charge dans un index Elastic

Comme on peut voir dans la photo, il y a au total 11908046 documents dans l'index, il prend environ 12GB de stockage dans cluster

Pour savoir si les informations sont bien présentés dans la base, on effectue une requête qui va afficher tous les résultats trouvés dans l'indice. Cela se fait tout simplement avec la requête match_all.

```
{
  "took" : 3,
  "timed_out" : false,
  "_shards" : {
    "total" : 1,
    "successful" : 1,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 10000,
      "relation" : "gte"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "geoname-2019.05.30",
        "_type" : "_doc",
        "_id" : "hfnUCGsBzh5_JW7YKXz1",
        "_score" : 1.0,
        "_source" : {
          "admin4Code" : null,
          "dateMod" : "2019-04-10",
          "cc2" : null,
          "path" : "/Users/phucvu/Desktop/elastic/logstash-7.0.1/textFile/allCountries.txt",
          "countryCode" : "DE",
          "@timestamp" : "2019-05-30T13:00:50.377Z",
          "population" : "0",
          "latitude" : "50.86667",
          "timezone" : "Europe/Berlin",
          "alternatenames" : "Forst Oberaula",
          "admin2Code" : null,
          "featureClass" : "V",
          "admin3Code" : null,
          "name" : "Forst Oberaula",
          "geonameid" : "2861357",
          "dem" : "460",
          "host" : "server.phucvu.cf",
          "featureCode" : "FRST",
          "longitude" : "9.43333",
          "admin1Code" : "05",
          "@version" : "1",
          "elevation" : null,
```


FIGURE 5.8 – Représentation de données sous JSON

5.4.2 Traitement de text

Dans la fonction main principal du programme, pour une tendance donnée, on agrège tous tous les tweet et les retweet correspondants dans un seul variable. Puis, on chercher dans la valeur de variable tous les mots respectés la convention pour être un lieu. Autrement dit, on prend en compte les mots qui commencent par une lettre majuscule et calcule son occurrence. Toutefois, on garde que les mots avec une occurrence supérieur à un seuil définie en raison de la performance de traitement de texte.

La photo suivante représente le résultat de traitement de texte avec le nombre de comptage

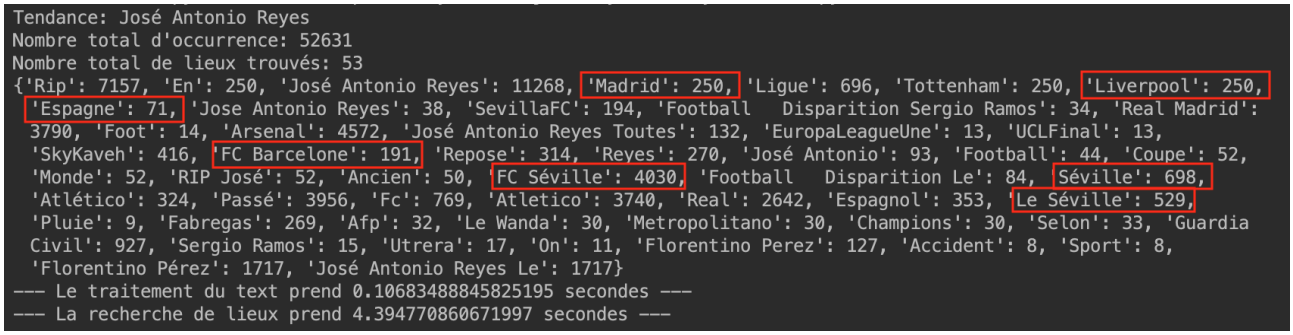


FIGURE 5.9 – Comptage d'occurrence de mots

Grâce à nos connaissances humaines, on pourrait identifier rapidement les mots : Madrid, Liverpool, Espagne, Barcelone, Séville sont des lieux entouré par le cadre rouge. Dans la photo, on avait Séville et Le Séville qui est censé d'un même lieu mais on garde en deux lieux séparés en raison du mots "Le" supplémentaire. C'est parce qu'on a par exemple le cas de "FC Barcelone" qui s'agit d'un équipe de foot alors que si on a Barcelone tout seul, ça sera bien un lieu. Même si une combinaison contient un mot de lieu, elle peut être avoir un sens totalement différent. Cette exception est également vraiment pour le nom de personne. Notre programme a donc bien détecté les combinaison de mots.

Prenons par exemple le mot "Espagne" dans le document résolu pour vérifier le bon fonctionnement de notre programme. On va utilise le système de comptage de l'éditeur Visual Studio qui est fiable pour comparer à notre résultat

Voici le résultat affiché par l'éditeur Visual Studio Code

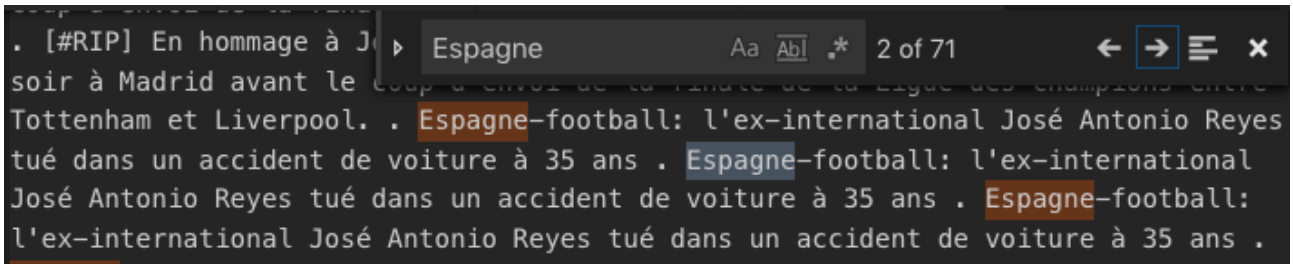


FIGURE 5.10 – Détection de mots en majuscule

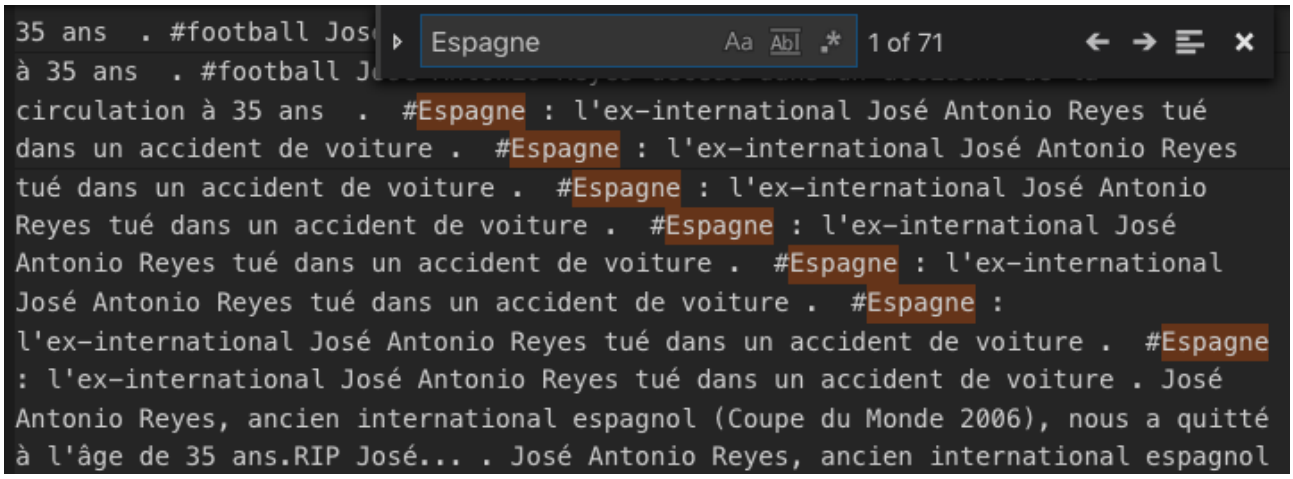


FIGURE 5.11 – Détection de hashtag

L'éditeur détecte qu'il y a 71 occurrences pour le mot "Espagne" d'ou le résultat obtenu par notre programme. Il traite correctement le mot collé avec un caractère non standard ainsi que celui commence par le hashtag.

5.4.3 Analyse de pays et de villes

Revenons sur le résultat pour la partie traitement de texte. A part de "FC Barcelone" qui n'est pas un lieu, les autres mots "Espagne", "Madrid", "Liverpool" sont évidement des lieux.
En effet, Espagne c'est un pays, Madrid et Séville sont des ville qui se situent dans l'Espagne et Liverpool est situé en Angleterre. Ce sont les lieux qu'on souhaite de voir pour la découverte de pays et de villes

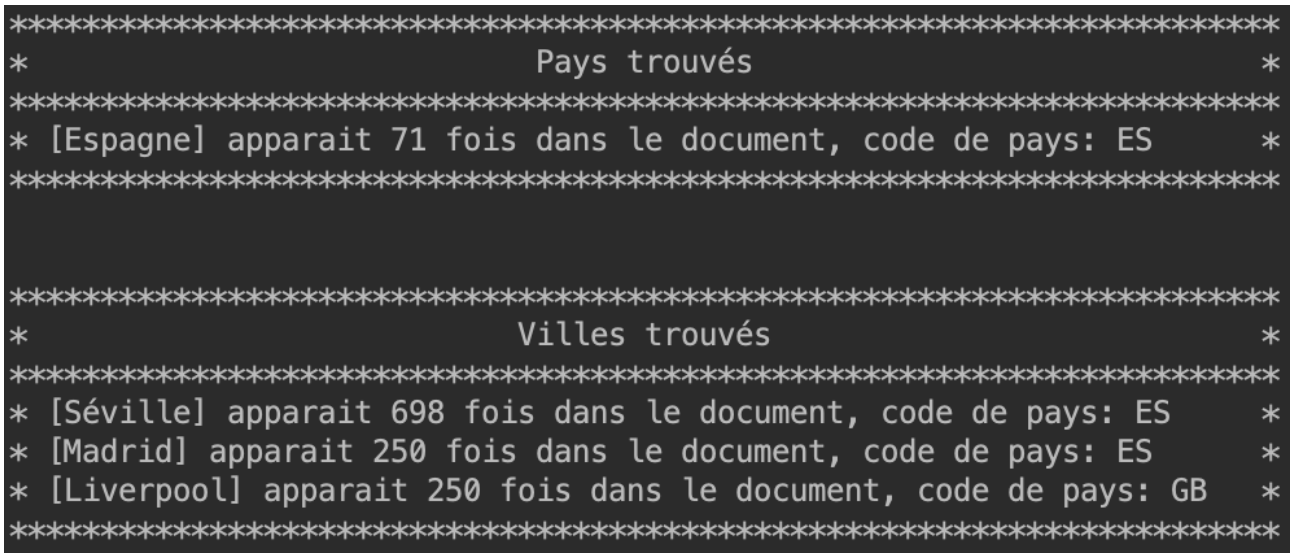


FIGURE 5.12 – Pays et villes trouvés dans le document

On trouve le résultat comme prévu dans la console. Pour avoir un résultat le plus pertinence que possible, on choisit que le top 3 de résultat pour chaque critère(pays, villes & places) et on s'ignore les autres avec les occurrences plus bas.
Une remarque qu'on voit toute de suite c'est qu'il existe une relation entre le pays Espagne et les deux villes Séville et Madrid. Ces deux villes appartient à Espagne. Par conséquent, le programme reconnaît cette relation et s'affiche dans la statistique de villes.

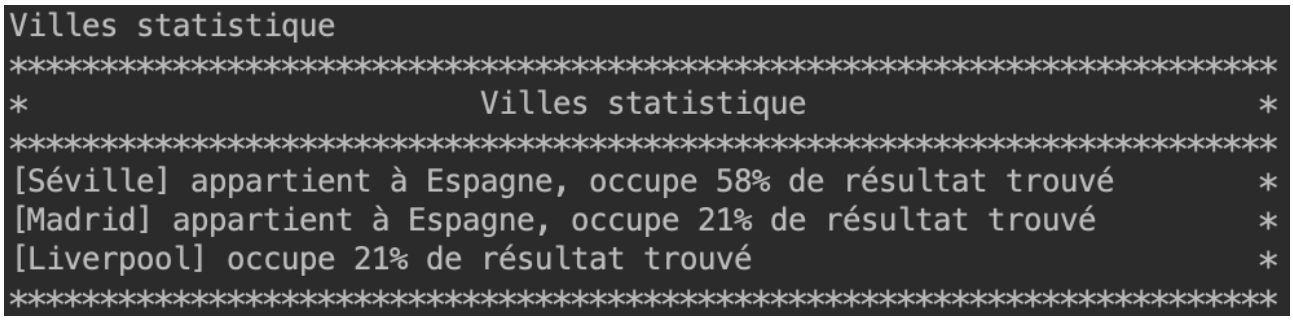


FIGURE 5.13 – Statistique de villes trouvées

Comme le résultat montré dans la photo, seulement les villes appartient à un pays trouvé a le pays correspondant, ce qui n'est pas le cas pour Liverpool. On affiche à la fois de taux d'occupation en pourcentage pour chaque ville entre le top 3.

5.4.4 Analyse de places

D'après le résultat de 5.9, on n'aurait aucun lieux de type place. Pourtant, on obtient des places ci-dessous existant dans la base Geoname

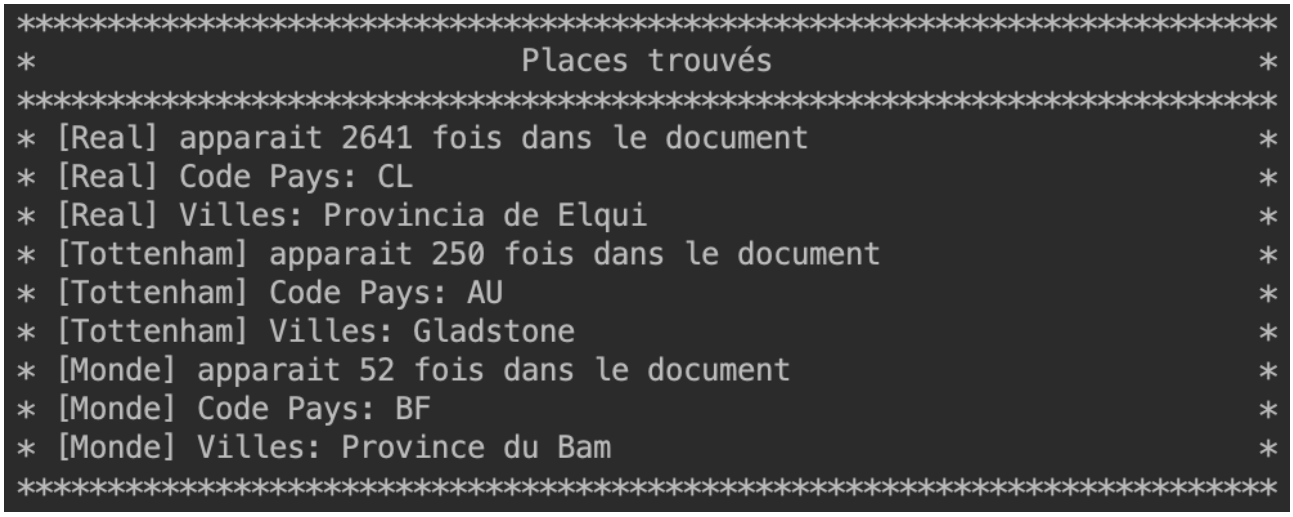


FIGURE 5.14 – Places trouvés dans le document

Tandis que les places situées dans le tableau sont bien existées, elles sont moins connues ou bien pas touristiques. Le fait que la base de données Geonames est très large, on peut parfois avoir des résultats qui sont étranges d'après notre connaissance. Par conséquent, l'importance de places trouvés est moins considérable que celle de pays et de ville.

5.4.5 Performance

Après la figure 5.9 on constate que le système tourne très rapide pour le traitement du texte. Concernant la performance d'extraction de lieux, cela dépend quasiment le débit de réseau et la capacité de traitement de données sur Elastic Cloud. En générale, le temps de traitement d'une requête sur Elastic Cloud prend en moyenne 10 ms. Selon la méthode conçu pour la recherche de lieux, le nombre de requête effectué est :

- **Recherche de pays et de villes** : 2 requêtes (fixé)
- **Recherche de places** : 3 requêtes (dans le pire des cas)

Test no et ref	Seuil occurrence	Nb lieux	Nb lieux trouvé	Temps théorique (sec)	Temps réel (sec)
1 (fig.5.9)	2	53	7	(53*5)*10(ms)=2,6	4
2 (fig.5.15)	2	21	5	(21*5)*10(ms)=1	1.9
3 (fig.5.16)	2	5	0	(5*5)*10(ms)=0.25	0.29

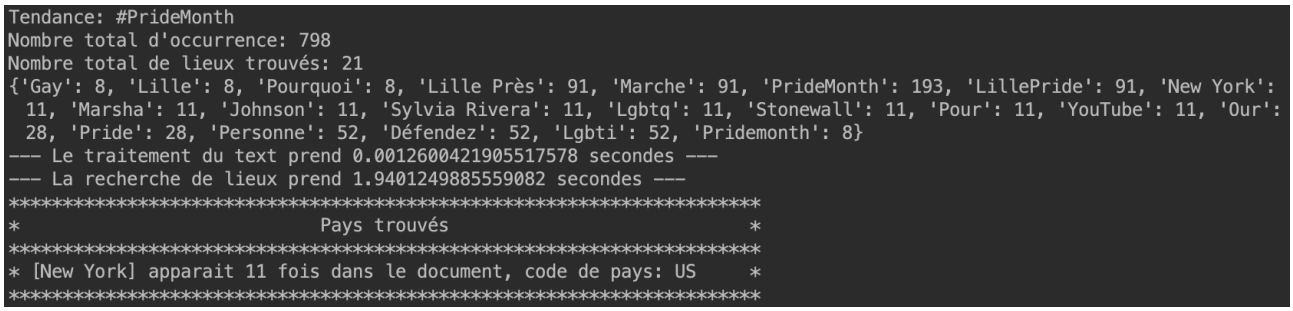


FIGURE 5.15 – Recherche de lieux pour la tendance #PrideMonth

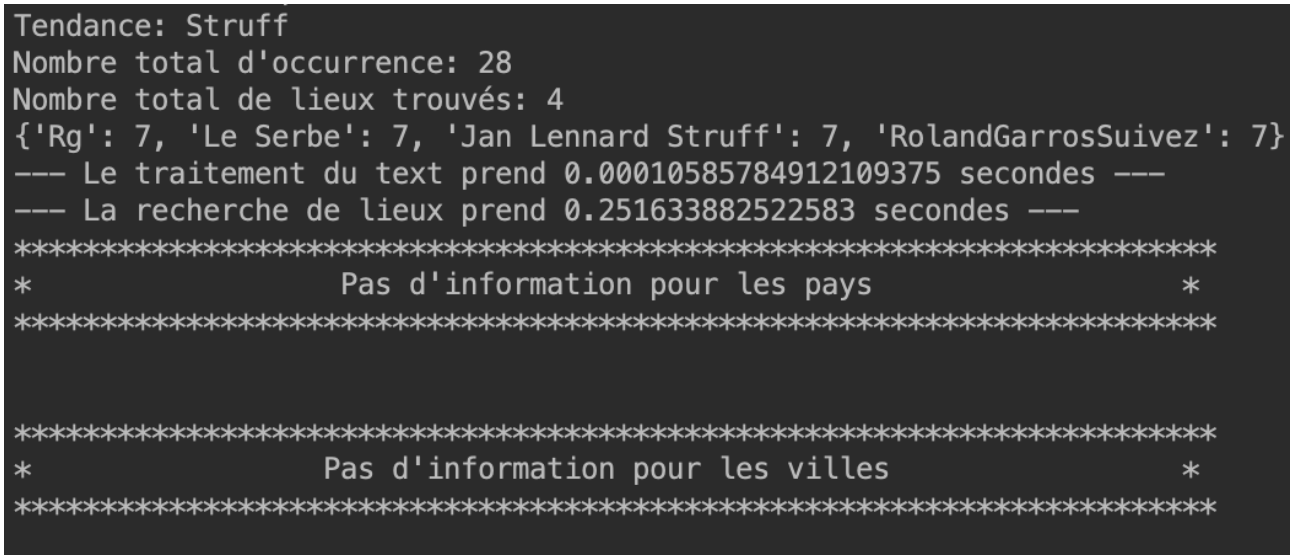


FIGURE 5.16 – Recherche de lieux pour la tendance Struff

La performance tout au long de tests est fiable et acceptable comparé à le temps de traitement moyenne théorique

Pertinence de résultat

Considérons l'exécution de programme pour la tendance #TOTLIV. Cet événement parle d'un match de football entre deux équipes Tottenham et Liverpool. On peut remarquer que Liverpool est bien une ville situé en Angleterre. Par contre dans ce contexte, Liverpool veut dire une équipe de foot. Si le programme trouve Liverpool comme résultat, ça ne nous concerne pas. De ce fait, une étude sur la pertinence de résultat lié à un événement est nécessaire.

Dans le but de vérifier la pertinence de résultat, on effectue une étude sur 25 tendances de Tweet. Le programme prend en paramètre la tendance et retourne les résultats catégorisés par : Pays, Villes et Places. D'ailleurs, il est possible que le lieu récupéré n'ait pas le rapport avec l'événement passé. Il suffit d'utiliser notre connaissances humaines pour vérifier si le résultat est vraiment parle d'événement. En l'occurrence, on a deux résultats pour chaque catégorise, un pour les lieux trouvés par le programme et l'autre contient les lieux filtrés de premier résultat en utilisant notre connaissance pour valider la relation auprès d'événement.

Le tableau ci dessus représente l'observation de nombre de résultat pour 25 tendances. Les colonnes sont les suivantes :

- Pays Trouvées : Le nombre de pays trouvés
- Bon Pays : Le nombre de pays lié réellement à l'événement
- Villes Trouvées : Le nombre de villes trouvées
- Bonne Ville : Le nombre de ville lié réellement à l'événement
- Places Trouvées : Le nombre de pays trouvés
- Bonne Place : Le nombre de place trouvées lié réellement à l'événement
- Le nombre de lieux qu'on est censé d'avoir selon la reconnais humaine

Test	Pays Trouvés	Bon Pays	Villes Trouvés	Bonne Ville	Places Trouvées	Bonne Place	Lieux correct
José Antonio Reyes	1	1	3	2	3	0	3
#PrideMonth	1	0	1	1	3	0	1
#TOTLIV	2	0	3	0	3	0	0
Naomi Osaka	0	0	1	0	0	0	0
#Acte29	2	1	2	2	0	0	3
#LOUMHR	0	0	0	0	0	0	0
Siniakova	0	0	0	0	0	0	0
Dimitrov	0	0	0	0	0	0	0
Julian Alaphilippe	0	0	0	0	0	0	0
#LIVTOT	0	0	0	0	1	0	0
RIP Reyes	0	0	0	0	0	0	0
Madouas	0	0	0	0	0	0	0
Massacre	3	3	3	0	3	0	3
Happy Pride Month	0	0	0	0	0	0	0
Reds	1	0	3	1	3	0	1
Fabio Quartararo	0	0	0	0	0	0	0
Lucas Moura	0	0	1	0	0	0	0
Fognini	0	0	0	0	0	0	0
les gilets	3	1	3	3	3	0	4
Novak Djokovic	0	0	0	0	0	0	0
Wawrinka	0	0	0	0	0	0	0
Célibataire	0	0	0	0	1	0	0
Delon	0	0	2	1	1	0	1
#LeMomentDeBriller	1	1	0	0	3	0	1
#LolOpenTour	1	0	0	0	0	0	0
Totale	15	**7**	22	**10**	24	0	**17**

TABLE 5.1 – Pertinence de résultat

La plupart du temps, le taux de réussite pour la recherche de pays et villes est respectivement élevé pendant que la recherche de place nous donne pas aucun résultat positif. On peut déduire que la recherche par pays et villes possède la meilleur pertinence

Chapitre 6

Découverte de la date

Pour chaque évènement, il s'agira, pour finir de découvrir une date et une durée adéquates le définissant.
Celle ci se base essentiellement sur les tweets récupérés par l'API.

6.1 Analyse de la problématique

Un événement est un fait marquant de l'actualité et est défini par sa description, son lieu, sa date, et sa durée.
On souhaite de mettre en place un système permet de détecter les dates et les durées d'un ou plusieurs text

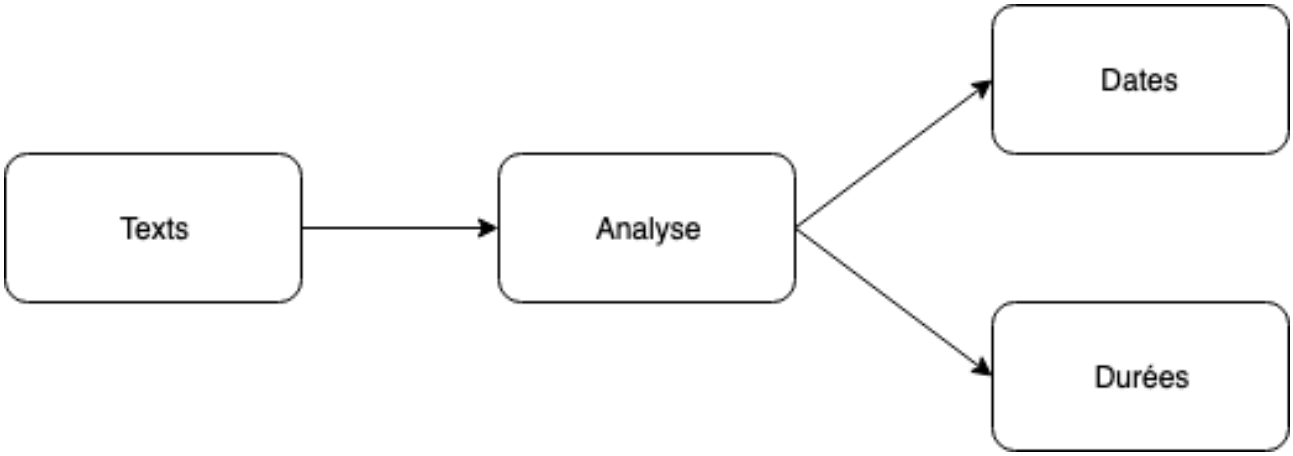


FIGURE 6.1 – Détection de la date/durée dans un texte

La date d'un événement doit permettre de savoir la date exact de cet événement ou bien sa durée s'il s'agit d'un événement de quelque jours.

Pour Bien comprendre ce qui vient d'être dit, donnons un exemple concret. Supposons que nous ayons la tendance twitter "Coup D'afrique", il faut donc rechercher une date ou bien une durée pour cette tendance twitter dans les tweets liés à celle-ci. Nous trouverons donc surement des phrases dans les tweets comme "la coupe commence après 13 jours", notre détecteur de date doit détecter que cette phrase exprime une date et la transformée en date, Nous allons avoir en sortie "21-06-2019"

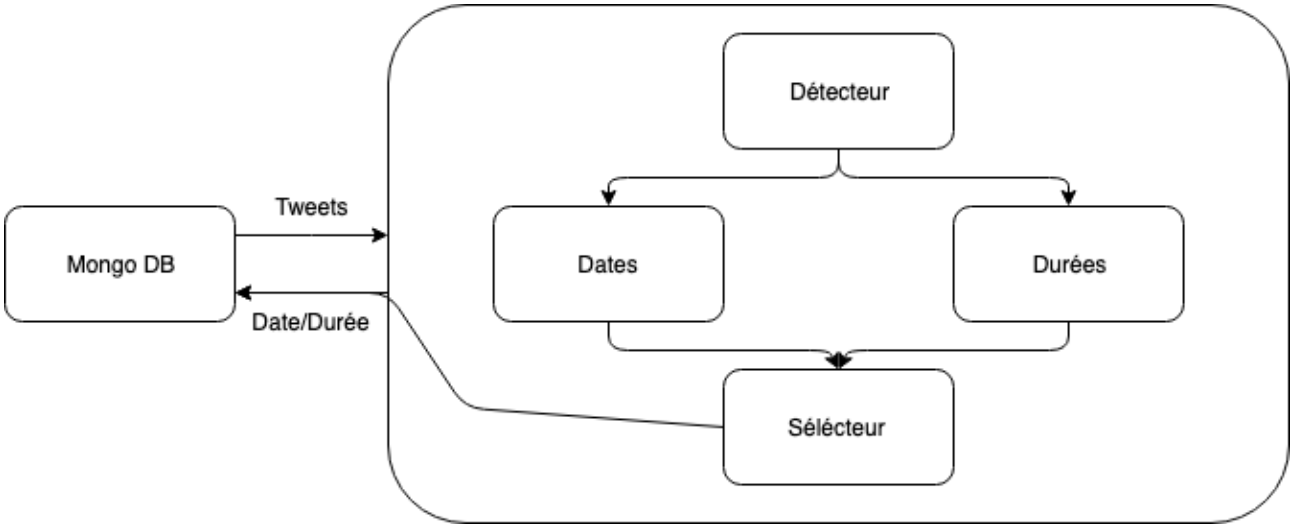


FIGURE 6.2 – Interaction de l'analyseur avec la base de donnée

A travers ce schéma, nous pouvons apercevoir le changement des tweets à des dates ou bien des durées, qui fait comme suit :

- Récupération et envoi des tweets vers l'application
- Détecter toutes les dates et les durée qu'ils contiennent
- Sélectionner la date ou la durée la plus pertinente
- Obtention et ajout de la date dans la base de données

6.2 Méthodes et Algorithmes de recherche

Datefinder : c'est un module python pour localiser les dates dans le texte. On utilise ce package pour extraire toutes sortes de dates, telles que des chaînes, d'un document et les transformer en objets datetime.
Ce module recherche les chaînes de date / heure probables puis utilise le package Dateparser pour convertir en objet date / heure.

FIGURE 6.3 – Détection de date avec DateFinder

Mot qui exprime une date : On a défini statiquement des mots qui exprime des dates, on a mis en place trois méthode algorithme :

- les jours/mois suivant : on a déclarer des mots statique qui exprime les prochaines dates comme "suivant prochain . . . ", à chaque qu'on trouve le mot on vérifie si le mots qui le précède ou bien celui qui le suit est un jour ou un mois, si c'est le cas on crée un objet heure qui lui correspond.

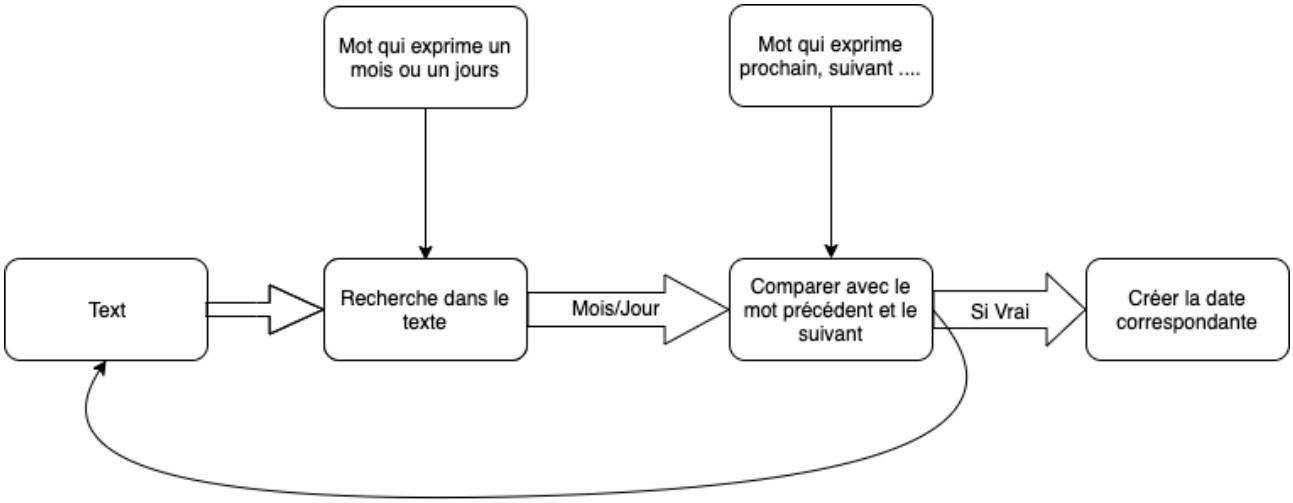


FIGURE 6.4 – Détecter les mots exprimant une prochaine date

- les jours/mois précédent : comme le point précédent on a déclarer des mots statique qui exprime les anciennes dates comme "précédent prochain . . . ", à chaque qu'on trouve le mot on vérifie si le mots qui le précède ou bien celui qui le suit est un jour ou un mois, si c'est le cas on crée un objet heure qui lui correspond.

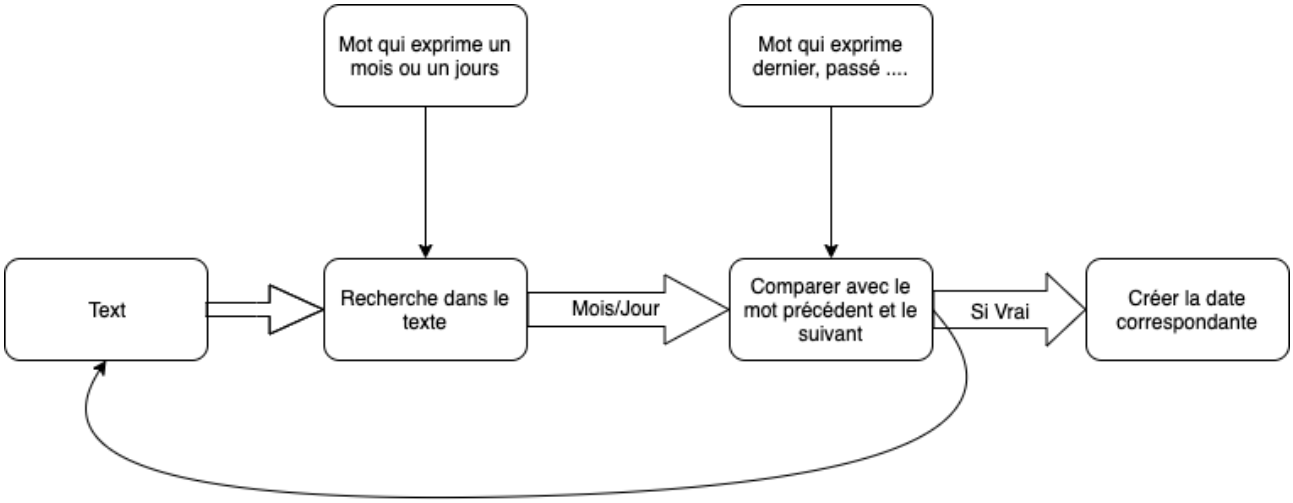


FIGURE 6.5 – Détecter les mots exprimant une ancienne date

- Les mots qui exprime le jours même, hier et demain.

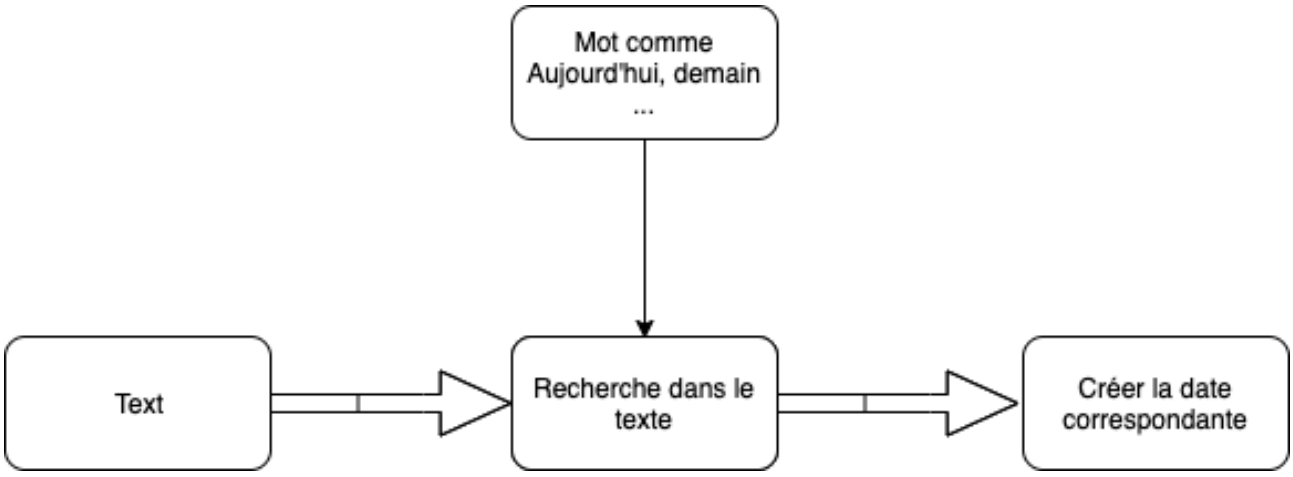


FIGURE 6.6 – Détecter les mots exprimant une date

Les durées : comme dit auparavant, sauf que pour les durées on doit trouver deux mots pour que les deux dates soit une durée, la façon de recherche change aussi, pour la date de début doit être

entre deux mots et celle de fin doit être après le deuxième mot par exemple “de Date_début à Date_fin”.

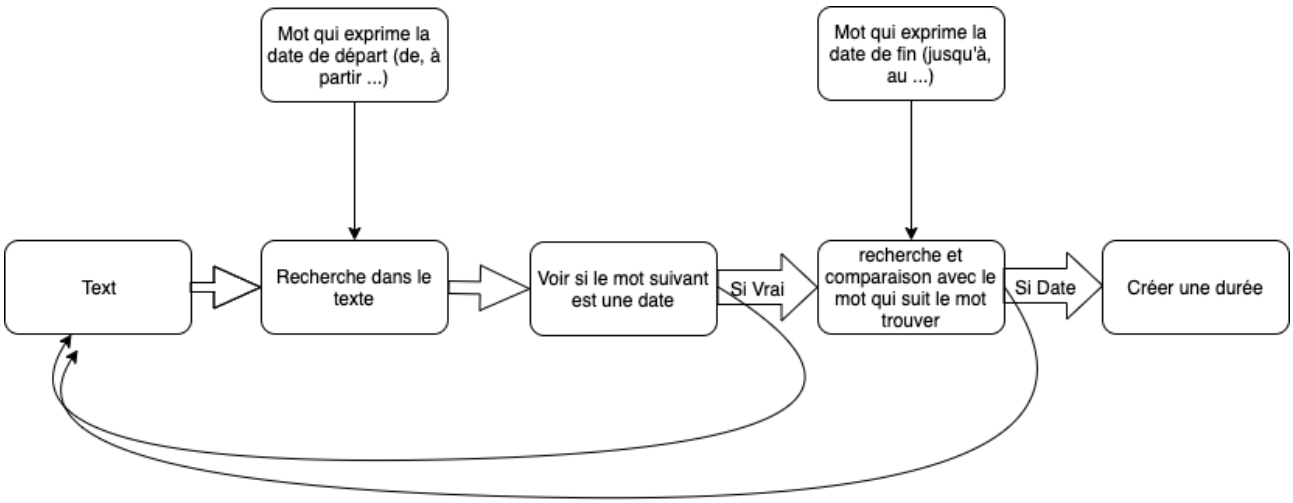


FIGURE 6.7 – Détecter les mots exprimant une durée

6.3 Analyse

La recherche de la date de l’événement se fait en deux étapes :

- on doit parcourir tous les tweets de chaque tendances, puis on analyse chaque tweet avec les algorithmes qu’on a cité au dessus. On cherche la date la plus fréquente, de même pour la durée, la phase finale pour cette étape c’est de comparer le nombre de fois dans lesquelles la date et la durée qu’on a choisie sont présentes, on choisit celle qui s’est répété plus de fois, le cas où on a une égalité on prend la durée.

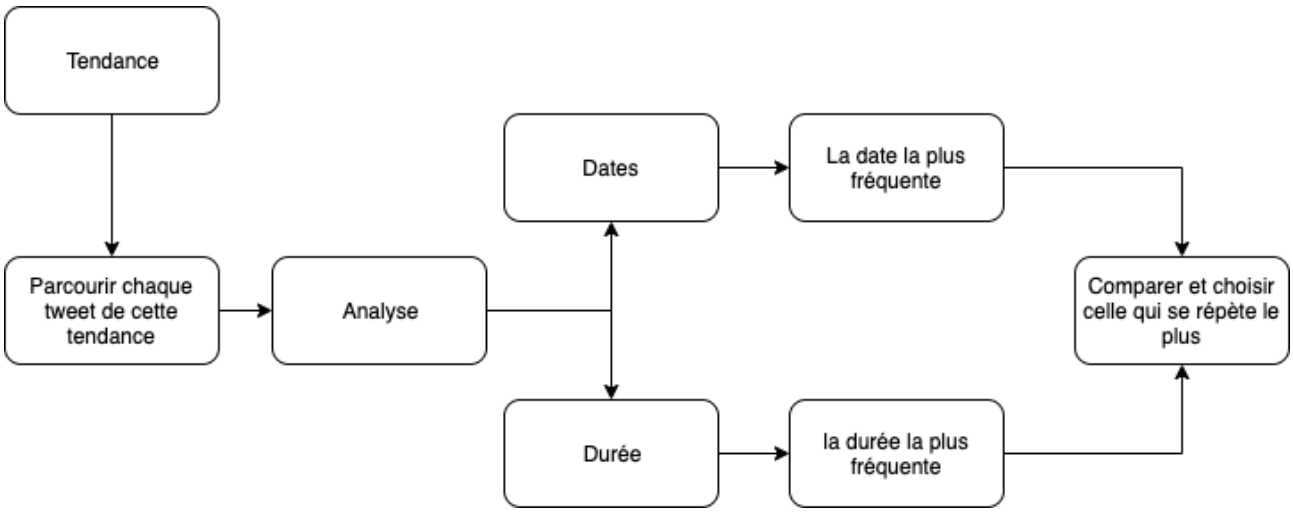


FIGURE 6.8 – Analyse et sélection de la date finale

- la deuxième étape c’est de rassembler les dates et les durées des tendances, faire le même processus que le point précédent, la date qu’on obtient c’est la date de l’événement.

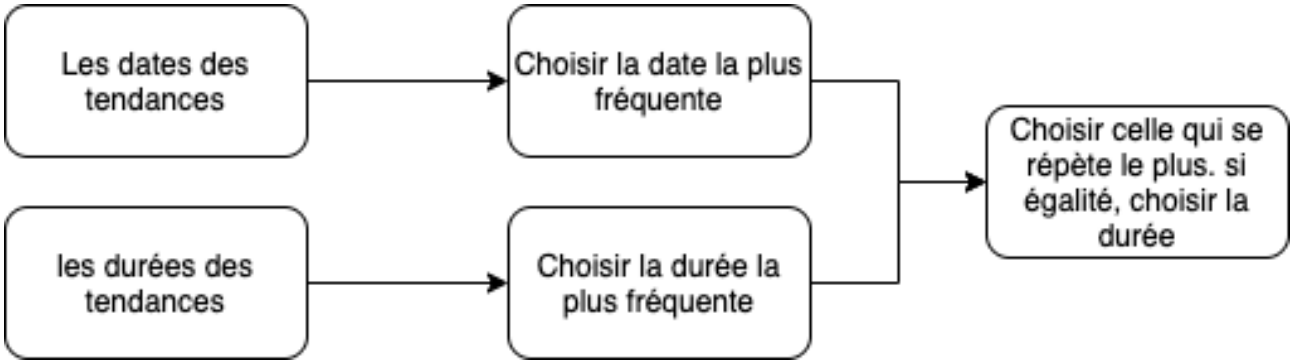


FIGURE 6.9 – Comparaison entre la date et la durée

6.4 Tests et certification de la solution

En lançant notre algorithme de détection de date nous remarquons que les résultats obtenu sont assez correct :

Nombre d'événement	Bons	Mauvais	Vide
50	28	19	3

Prenant la tendance "Gilles Sergent" de l'événement "Gilles Sergent s'en va, Fabrice Clément nouveau président de Caen", voilà les résultats trouvés :

```
**************************************************
**********Statistique*********************
**************************************************
Nombre de Date trouvée: 28
Nombre de Durée trouvée: 3
**************************************************
La date choisit est :  2019-06-02 00:00
Elle est présente dans la liste : 12 fois
```

FIGURE 6.10 – Statistique de détection de date

En faisant des tests nous avons trouvé que la détection d'une date ça nous prend au minimum quatre secondes et 150 secondes en grand maximum

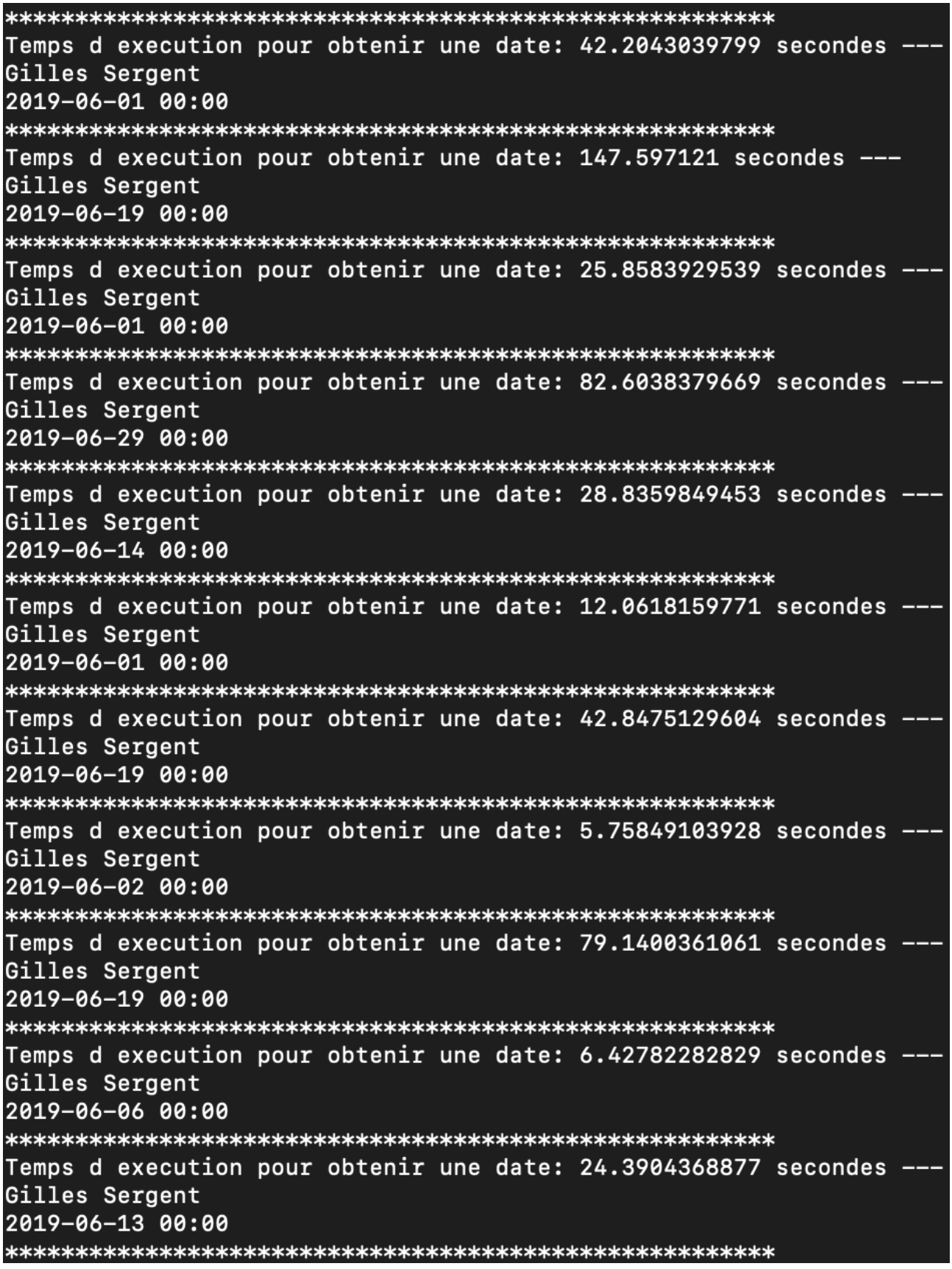


FIGURE 6.11 – Temps d'exécution

En lançant notre détecteur de date voilà quelques résultats qu'on a pu trouver.

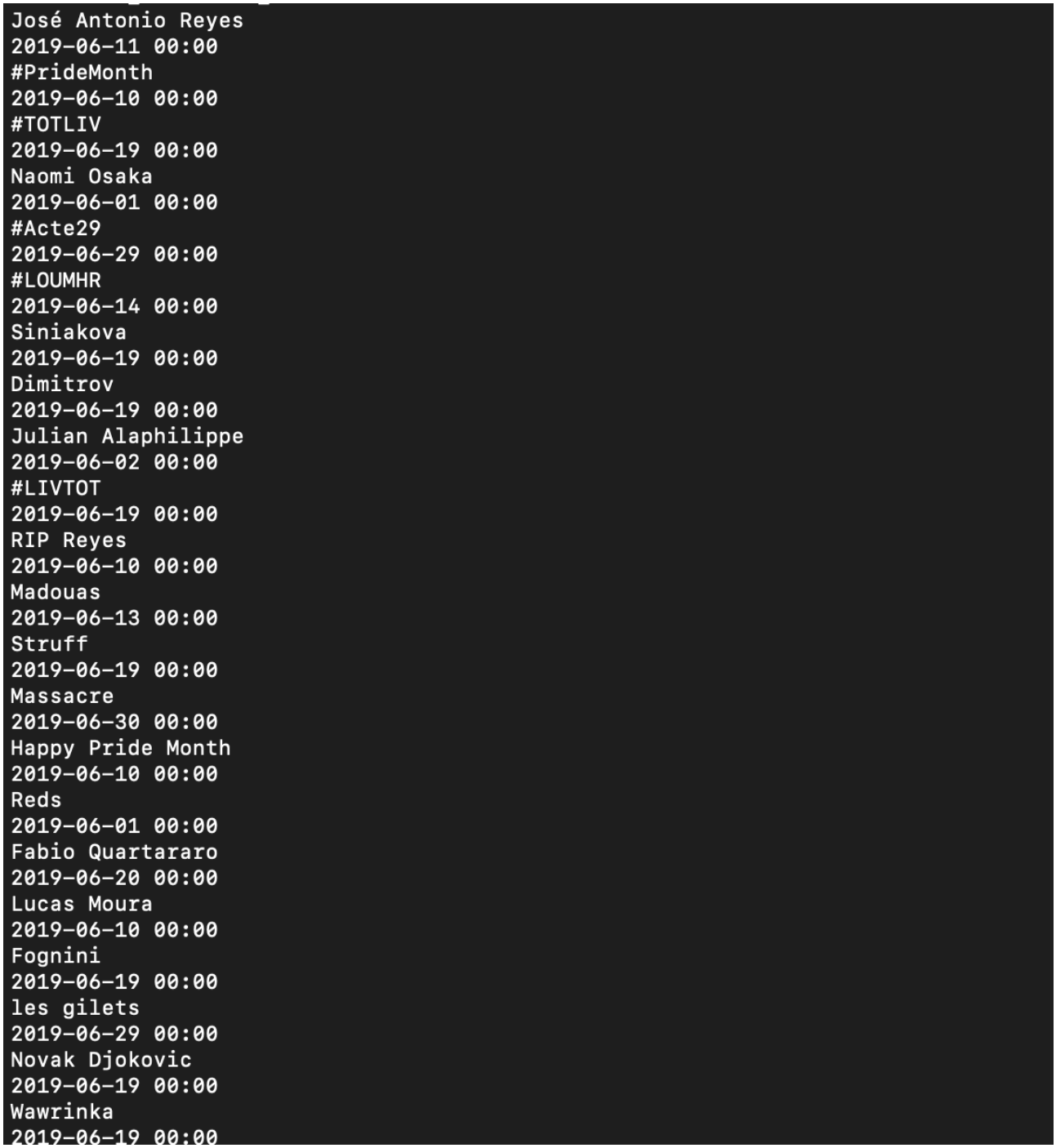


FIGURE 6.12 – Résultat final

Chapitre 7

Rendu Final

Dans ce chapitre, nous présenterons le rendu final vu par l'utilisateur, soit une IHM, ainsi que les tests et certifications que nous avons effectués.

7.1 IHM

Maintenant que nous avons établi et construit les principales fondations de notre projet "Web Sensors", à savoir l'approvisionnement automatique de tweets et d'articles provenant de flux RSS vers notre base de données, puis lle traitement de ces informations pour en conclure des événements précis, nous sommes désormais dans la position de pouvoir proposer à un client un interface graphique et interactif pour que ce dernier puisse naviguer vers le sujet qui l'intéresse et extirper les informations pour son utilisation personnelle.

7.2 Les choix d'interfaces possibles envisagés

La framework Python Django qui a l'avantage d'utiliser le même langage que la plupart de nos scripts, permet de créer des interfaces web. Elle utilise le modèle MVC. Un des membres de l'équipe (Mohamed II BAYO) s'est plongé dedans mais nous a informé qu'elle demande un processus d'apprentissage coûteux en temps pour reproduire des fonctionnalités triviales d'une page web classique

Dans la même ligne que Django, on a aussi la possibilités de bibliothèques Java comme JSF. Par contre, on a ici l'inconvénient de devoir utiliser un langage complètement différent de Python

Enfin, il y a aussi l'option la plus évidente : une page web codée en Javascript/PHP.
Ces langages ont l'avantage d'etre tres simples et manipulent facilement le format JSON utilisé par notre base de données MongoDB
De part l'expérience du responsable de l'IHM dans ces langages (utilisés dans son alternance), il dépasse rapidement les avancées effectuées sur Django et dans l'optique d'être le plus efficace dans le temps qui nous est donné, c'est l'option que les membres s'accordent à retenir

Afin que tous les membres puissent accéder à l'interface pour leurs tests, nous avons choisi d'héberger notre travail sur la plateforme https ://www.alwaysdata.com/fr/, cet hébergeur a déjà été utilisé pour des projets des années précédentes et propose diverses base de données, notamment MongoDB ; cependant, les membres utilisant la base MongoDB localisée sur le cloud Atlas, nous ne sommmes pas servie de cette dernière.

7.3 Fonctionnalité de l'IHM

L'interface graphique est la conclusion finale de notre projet de synthèse. C'est l'aboutissement de nombreux traitements de données, d'algorithmes de Natural Language Processing complexes, et d'acquisition d'informations en continu.

FIGURE 7.1 – Choix du trend disponible

Lorsqu'un utilisateur arrive sur notre page, il se verra proposer une barre de recherche s'il a un événement en tête, ou bien une liste des événements ayant eu lieu récemment.
Une fois un événement sélectionné, il peut alors choisir parmi plusieurs options de filtrage l'affichage de tweets et articles de flux RSS qui "match" ces tweets :

- Afficher selon les utilisateurs dits "influenceurs". Ce sont les utilisateurs qui ont le plus de followers parmi le lot d'utilisateurs dont nous avons récolté les tweets. Pour ce critère, on a imaginé deux scénarios :
  - le premier est celui où ces influenceurs émettent une sorte de débat entre eux. Cela donne donc lieu à des échanges qui peuvent être intéressants à retranscrire pour le client. (Ex : s'il y a eu un échange entre Emmanuel Macron et Donald Trump sur un sujet comme l'écologie, nous choisissons d'afficher exactement les tweets où ces messieurs s'interpellent.)
  - le deuxième cas est l'inverse du premier. Dans le cas où les influenceurs n'échangent pas entre eux et émettent tout simplement leurs opinions ou des informations sur les faits, dans ce cas nous affichons les tweets les plus populaires (évaluation du nombre de retweets et de likes avec des coefficients mettant plus en valeur les retweets).

- Afficher selon les plus référencés ou retweetés. Ce sont les tweets qui suscitent un accord total des autres utilisateurs et possèdent donc une qualité spéciale qui peut être une information pertinente tout comme une formulation humoristique. Comme effleuré précédemment, pour déterminer le score de popularité d'un tweet, nous devons appliquer des coefficients pour mettre d'avantage en valeur le nombre de retweets par rapport au nombre de likes. Nous utilisons le rapport (3*RT)/LK, où 3 est un coefficient choisi après avoir étudier grosièrement la variance de l'écart retweets/likes.

Ce filtrage, dont l'intersection avec le précédent est très probablement non nulle permet néanmoins de donner plus de valeur à des tweets issus de personnes non populaires et peut amener des informations supplémentaires lorsque le premier filtrage pourrait répéter la même information

- afficher selon la période de forte activité. En effet, un événement va nécessairement connaître une activité suivant une courbe de la forme d'un ou plusieurs U renversés si l'on décrit grossièrement la courbe (évidemment il peut arriver que la courbe soit plus variable cela est plus rare). Dès lors il peut être très pertinent de proposer une sélection de tweets selon ces seuils d'activité afin de retracer l'événement dans le temps.

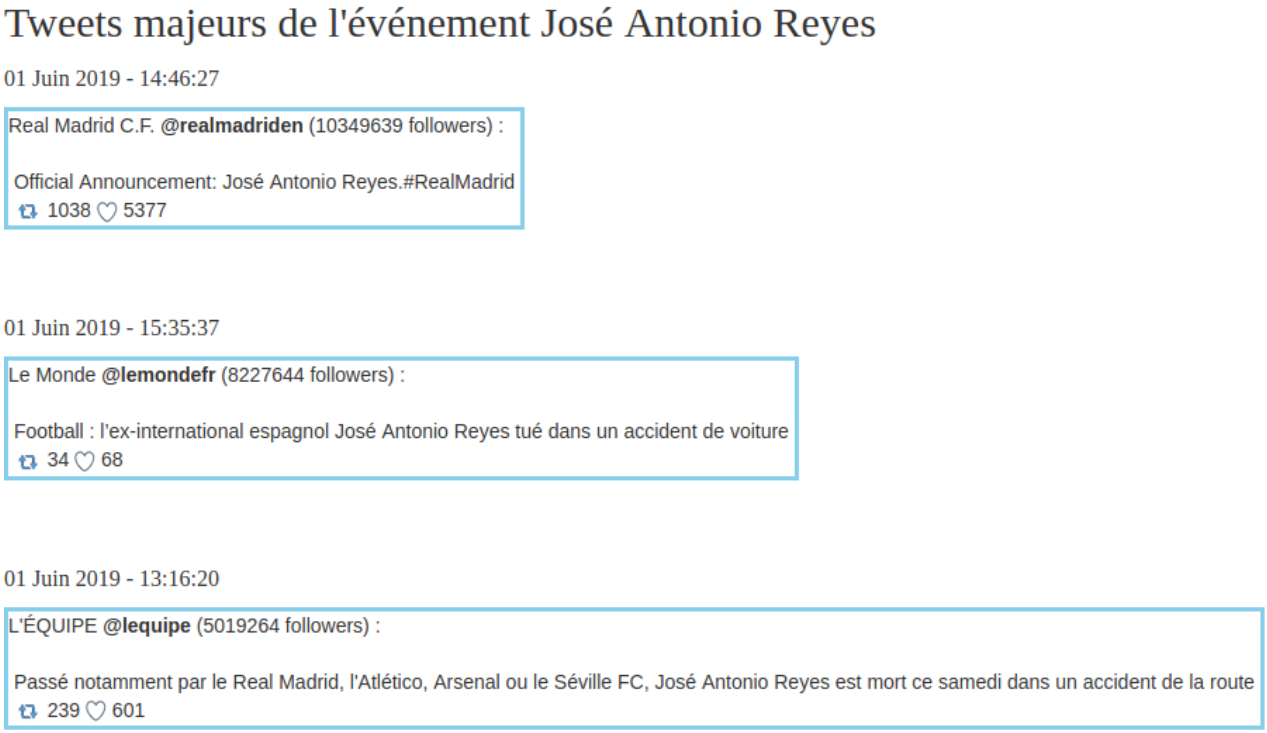


FIGURE 7.2 – Tweets d'influenceurs

7.4 Fonctionnalités supplémentaires

En plus des tweets, nous proposons à l'utilisateur le résultat de toutes nos analyses :

- Description de l'événement : en effet, les tendances de Twitter ne sont pas suffisantes pour déterminer ce qui suscite cette afflux de tweets. Il faut aller plus loin et déterminer sous la forme d'une phrase compréhensible par le client, une description de l'événement si la tendance correspond bien à un événement et non à une simple discussion.
  Pour cela, nous déterminons à partir des flux RSS que nous avons récoltés le ou les articles obtenant un score de "matching" satisfaisants, et nous prenons la description du meilleur afin d'obtenir une phrase rédigée par un humain, claire et compréhensible pour l'utilisateur.

- Estimation du ou des lieux : information supplémentaire, qui est tout autant pertinente. Le client doit pouvoir déterminer rapidement où se passe l'événement.

- Estimation de la date du début de l'événement : en faisant l'analyse des mots ayant une connotation temporelle et des dates présentes dans les tweets, nous pouvons soumettre à l'utilisateur la date la plus logique obtenue par l'algorithme.

FIGURE 7.3 – Informations de l'événement

En plus des informations, nous proposons à l'utilisateur les meilleurs articles en rapport avec l'événement afin de compléter les informations choisies par nos algorithmes.

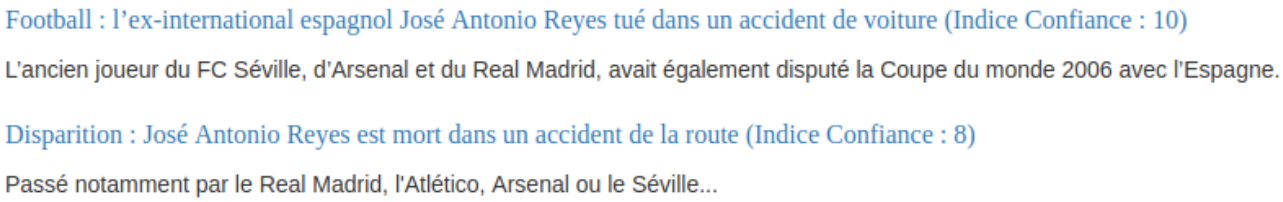


FIGURE 7.4 – Articles à propos de la tendance José Antonio Reyes

Avec cela, il est important de pouvoir soumettre des statistiques à l'utilisateur. Nous affichons une courbe d'évolution dans le temps de l'activité d'une trend, cela permet clairement à vue d'oeil de conjecturer sur la période d'un événement.

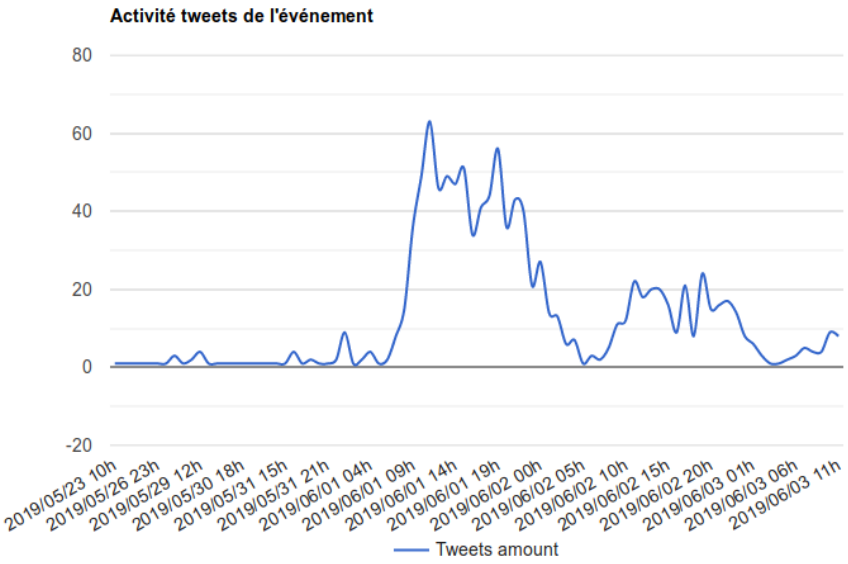


FIGURE 7.5 – Évolution de tweet

Nous proposons aussi à l'utilisateur un récapitulatif des statistiques d'une trend comme le nombre de tweets collectés, le total de retweets, ou bien le nombre d'utilisateurs différents ayant tweetés à ce sujet.

FIGURE 7.6 – Statistique de l'événement

Nous avions de base envisager d'utiliser la bibliothèque JS de twitter pour l'affichage d' "embedded tweets", c'est-à-dire le format de tweets découpés utilisés par les sites d'informations et qui conservent l'habillage original de Tweeter.

FIGURE 7.7 – Tweet original

Malheureusement, cette option s'avère non viable car certains tweets contenus dans la base Mongo ont été supprimés par Twitter ou l'utilisateur, et lorsqu'on essaye de créer un tweet avec un ID non existant dans.la base de données de Twitter, cela donne lieu à une HTTP Error 404 Object Not Found, et cela fait planter le script. Il n'y a de plus, pas d'options évidentes pour vérifier si un tweet existe toujours au préalable d'exécuter le script.
Puisque nous voulons conserver l'intégrité de notre système, toute information récoltée par notre système est pertinente. Il n'est pas envisageable de supprimer de notre base de données les tweets ayant été éliminés de celle de Twitter. Nous décidons donc d'opter pour un affichage plus basique en affichant les données extirpées depuis notre base, qui sont fréquemment mise à jour et reflètent donc les vraies valeurs de la base de Twitter.
Pour chaque tweet affiché, nous proposons des articles provenant de divers flux RSS de sites d'informations, en lien avec les mots contenus dans le tweet précédemment mentionné.

7.5 Récapitulatif de l'interface

En résumé, un client quelconque peut choisir en fonction de mots-clés, thèmes, ou par la sélection précise d'une Trend, un flux d'informations mis à jour périodiquement. Ce flux propose des statistiques, des tweets, et des articles issus de flux RSS.

La valeur ajoutée est notre analyse de ce flux, qui permet de proposer une description fiable de l'événement (en cours ou terminé), le ou les lieux probables des faits, et la date la plus pertinente du début de l'événement.
Nous ajoutons à cela plusieurs méthodes d'appréhension de l'événement (vu par des influenceurs, vu par des tweets populaires, vu par des tweets issus des pics d'activités...) Nous pouvons affirmer avoir implémenté une forme primaire de "web sensors" car nous avons automatisé la récolte d'informations issues de la toile, l'analyse et traitement des données après nettoyage, et l'affichage dynamique des résultats.

7.6 Tests utilisateur et certification

7.6.1 Récupération et sauvegarde des données

Pour la récupération des données, nous envoyons des requêtes HTTP vers des URLs proposés par Twitter afin de récupérer des données sous format JSON.
Le taux de réponse de Twitter est limité, nous avons le droit à 180 requêtes sur une fenêtre de 15 minutes

Cette méthode est donc utilisée pour récupérer des données récentes à partir de mots clés(dans notre cas nous utilisons des tendances = évènement), ces données sont ensuite sauvegardées en base de données comme expliqué dans les chapitres précédents. Une fois la description, la date et le lieu concernant ces évènements découverts nous actualisons la base de données avec ces informations.
Voici un exemple pour un évènement :

FIGURE 7.8 – Les données de l'évènement sauvegardées en collection Mongo

7.6.2 Découverte et Affichage des données

Prenons l'événement suivant : mort du footballer José Antonio Reyes.
Cet événement a été en tendance sur twitter sous plusieurs formulations ("José Antonio Reyes", "RIP Reyes", etc). Notre produit est donc censé être capable de déterminer une description précise, estimer les lieux possibles, et trouver la date du début de l'événement, c'est-à-dire les premiers témoignages du décès du joueur. Ci-dessous, voici le résultat affiché au client, on constate que les informations

sont pertinentes et que les algorithmes ont bien "sentis" la tendance "José Antonio Reyes" comme précurseuse d'un événement.

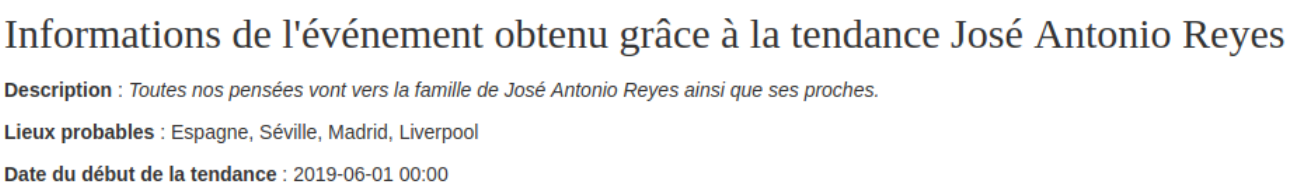


FIGURE 7.9 – Informations obtenues pour la tendance "José Antonio Reyes"

Par ailleurs, nous sommes censés proposer divers modes d'affichage des tweets de l'événement. Intéressons nous ici au cas des périodes où l'activité est élevée.

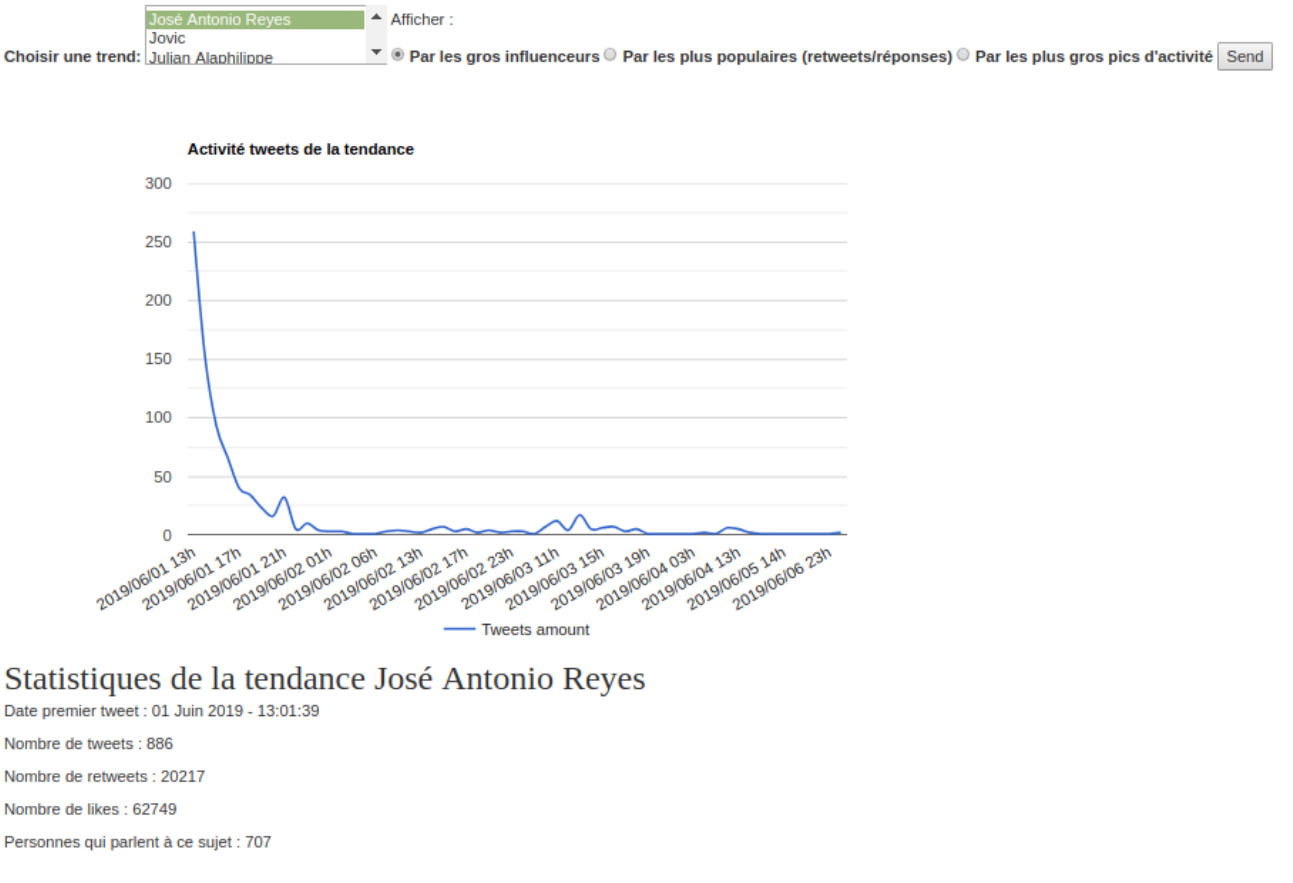


FIGURE 7.10 – Activité de la tendance "José Antonio Reyes"

On constate que l'événement était très discuté lors de l'après-midi. Aussi, nous montrons à l'utilisateur les meilleurs tweets extraits de ces périodes.

FIGURE 7.11 – Tweets affichés

Maintenant, prenons le cas d'une tendance qui ne reflète pas un événement mais simplement une discussion à propos d'un sujet. La tendance "Bruce Toussaint" sera notre exemple.

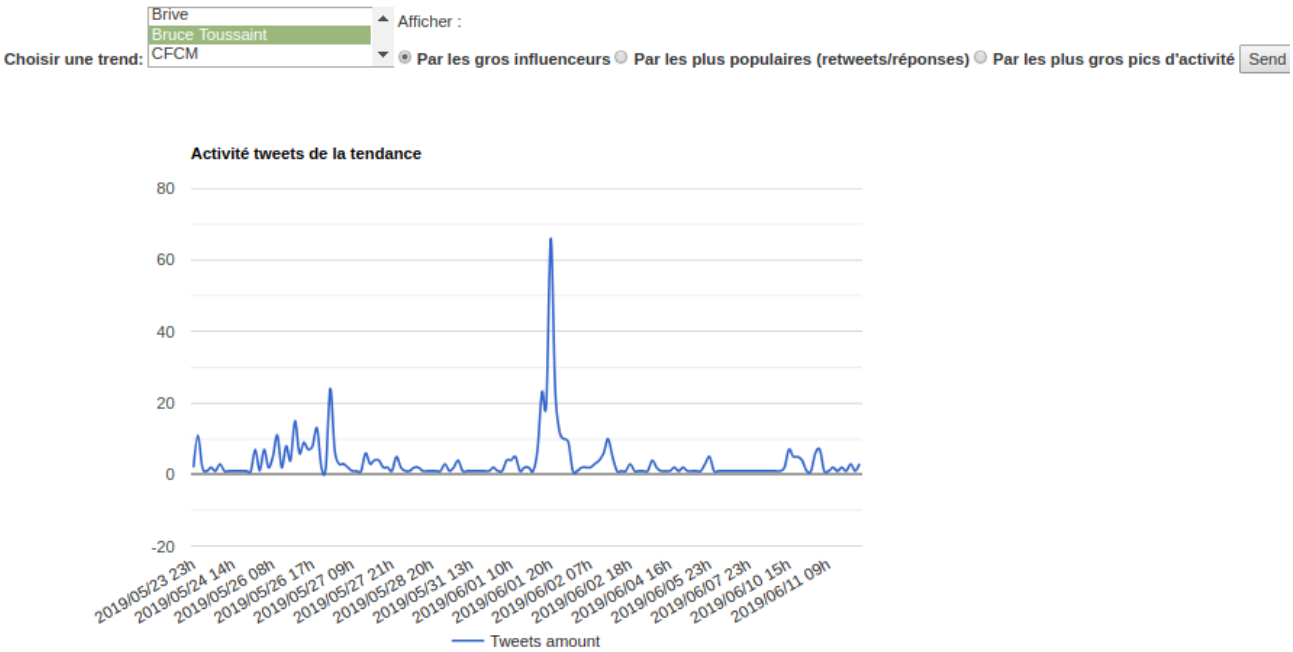


FIGURE 7.12 – Activité de la tendance "Bruce Toussaint"

Bruce Toussaint est un animateur radio/télé et ses spectateurs réagissent à ces émissions comme le montre l'activité retranscrite au dessus.
Le test est de voir si notre algorithme est capable d'éliminer cette tendance car ce n'est pas un événement.

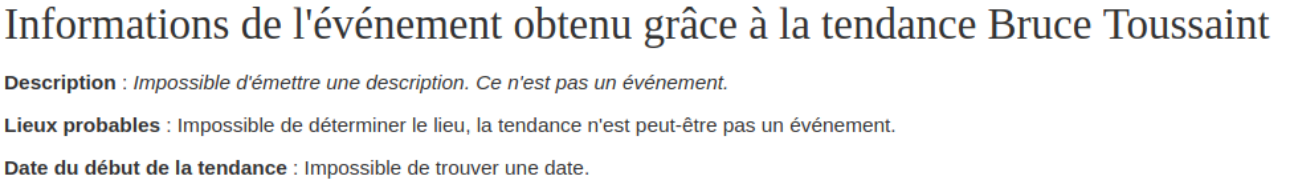


FIGURE 7.13 – Informations obtenues pour "Bruce Toussaint"

On observe que les algorithmes n'ont pas réussi à estimer des informations pour cette tendance, on conclut donc que ce n'est pas un événement. Notre produit est donc capable de différencier les tendances des événements. On considère qu'un événement est sûr si la description, le lieu, et la date ont réussi à être déterminés.

Néanmoins, il est toujours possible pour le client d'examiner les tweets selon le critère qui l'intéresse. Ci-dessous les tweets des plus grands influenceurs.

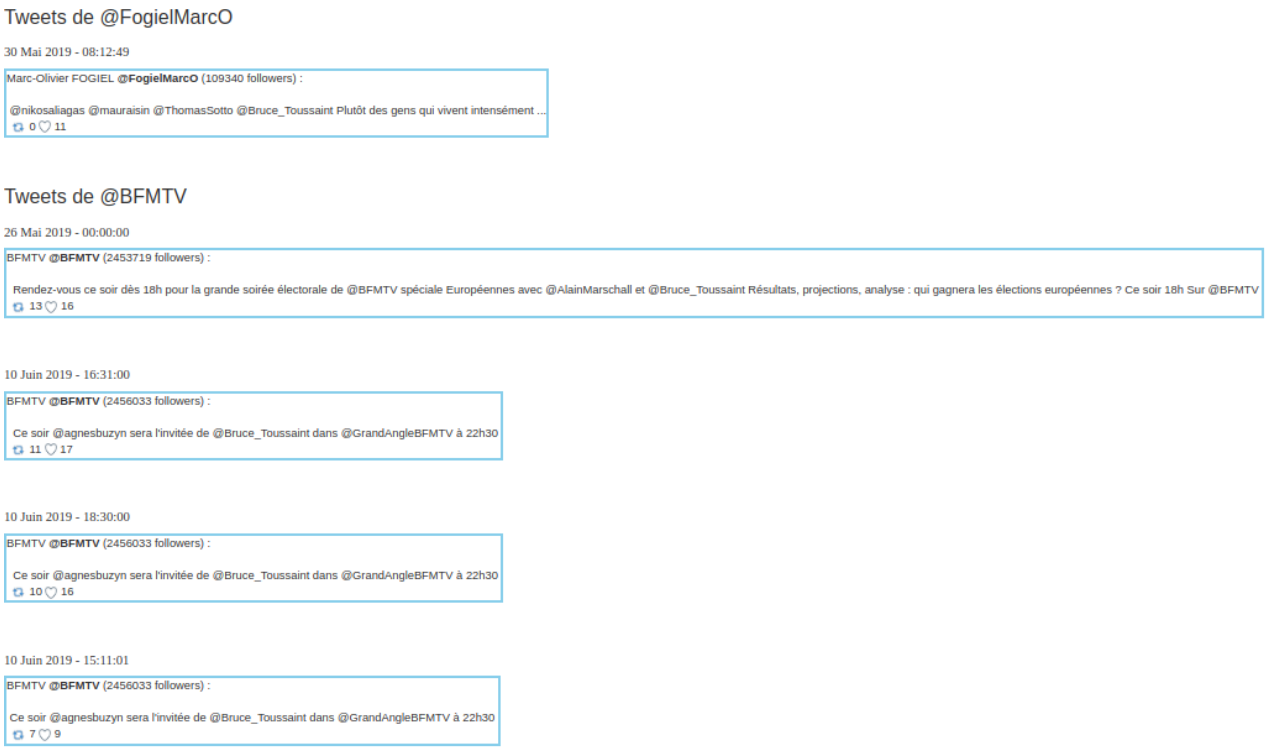


FIGURE 7.14 – Tweets selon le critère "grands influenceurs'

7.7 Autres tests et certification

Pour certifier notre solution, étant donné que nous avons déjà certifié les différentes parties de manière isolée avec des tests unitaires, nous vérifieront ici les connexions entre ces différentes parties. Soit, la connexion entre les algorithmes python et la base de données, ainsi que la connexion entre la base de données et l'IHM.

7.7.1 Connexion Python-Mongo

Nous avons donc vérifié la bonne exécution de notre algorithme de remplissage de la base de données qui consiste à récupérer les tendances depuis la base de données puis à l'actualiser avec les descriptions, dates et lieux des évènements.
De ce fait, nous remarquons que les accès sont disponibles que ce soit en lecture ou en écriture.
Nous notifions que cette base de données étant en ligne, nous dépendons d'un accès internet pour l'actualiser et y récupérer des données.

7.7.2 Connexion Mongo-IHM

Concernant la connexion Mongo-IHM, celle-ci à été vérifiée en requêtant la base de données avec PHP et en obtenant les résultats escomptés, il n'y a donc aucun problème de comptabilité ni de connexion et les deux modules s'imbriquent parfaitement.

7.7.3 Tests de performance de la rapidité du produit :

Nous avons utilisé l'extension google chrome "Page load time" proposée par Alexander Vykhodtsev afin de mesurer le temps de chargement de notre page. A noter que les mesures étaient différentes à chaque rafraîchissements avec une variance d'environ 3 secondes pour les temps les plus longs.

	A	B	C
1	Trend	Durée	Nombre de tweets
2	Laurent Ruquier	25,4	3136
3	Léonard	30,1	4028
4	Moussa Marega	6,88	57
5	Riolo	17,3	1693
6	Sign in with Apple	9,28	287
7	Square Enix	15,4	1247
8	Tribune de Marlene Schiappa	10,7	344
9	Wawrinka	32,4	4468
10	fabrice clément	5,54	80
11	josé antonio reyes	7,57	39
12	lariboisière	11,3	613
13	tableau	12,9	892
14	Mousquetaires	8,91	316
15	Fromage Agbonlahor	4,92	79
16	#snapchatdown	12,6	1484
17	#liverpoolvsTottenham	6,38	87
18	#frichtichallenge	10	941
19	#cheminots	8,22	230
20	Célibataire	44,8	7226
21			
22		Moyennes	
23		14,7684210526316	1434,05263157895
24			

FIGURE 7.15 – Mesures de performances

Nous avons choisi aléatoirement une vingtaine de tendances en prenant soin que l'échantillon comporte bien des extrema pertinents.
Ci-dessus, on obtient un temps de chargement moyen de 14 secondes, ce qui est plutôt lent, mais reste acceptable.

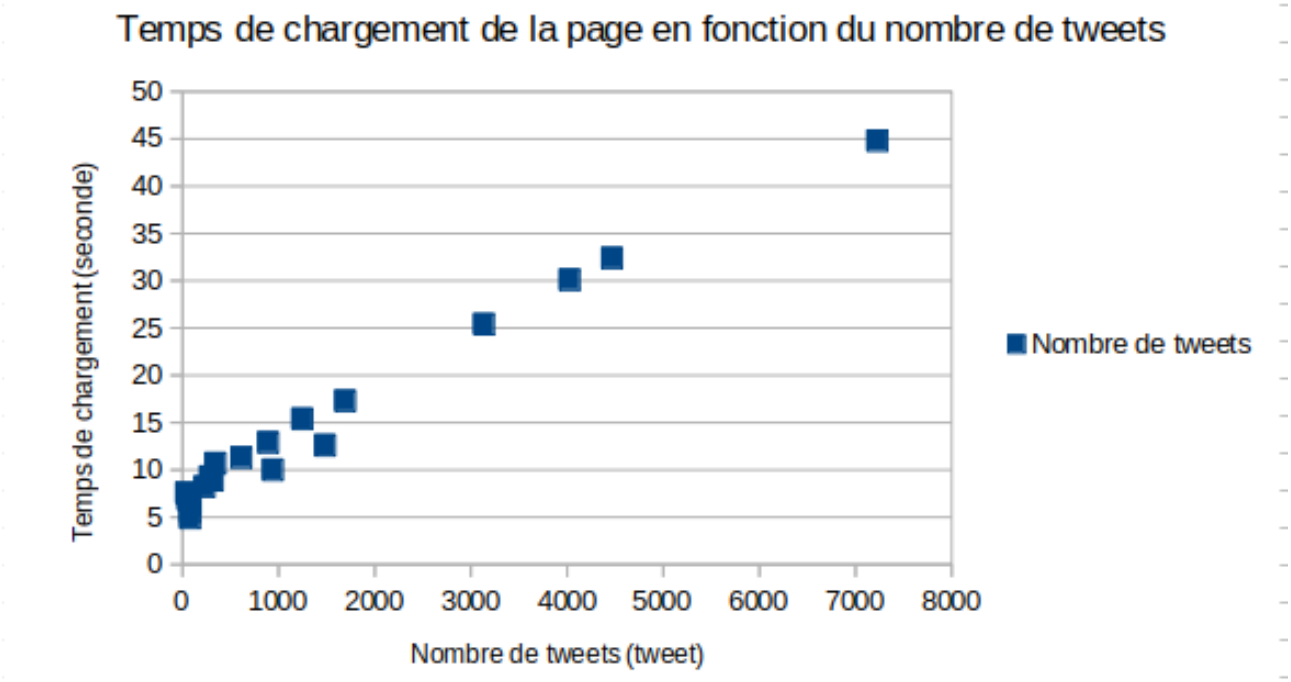


FIGURE 7.16 – Evolution de la performance

Avec l'ensemble de points obtenus par nos mesures, on constate tout de suite l'impact du nombre de tweets sur la performance du chargement de la page. Cependant, le souci de performance ne provient pas de la base de donnée en elle-même car MongoDB est un serveur qui est optimisé pour les grands

jeux de données. Après étude de la page, nous avons localisé la source du ralentissement dans un traitement en PHP post-récupération des données de la base Mongo.
Pour obtenir le graphe d'activité d'une tendance (cf figure ci-dessous), nous passons par un traitement PHP coûteux en mémoire afin de charger par la suite les données dans un tableau javascript qui donnera lieu avec l'API googlechart au graphique, et cela ralentit grandement le serveur. Si l'on considère la tendance "Célibataire" qui mettait plus de 40 secondes à s'afficher, si l'on se passe du graphique on passe alors à un affichage dès 8 secondes !

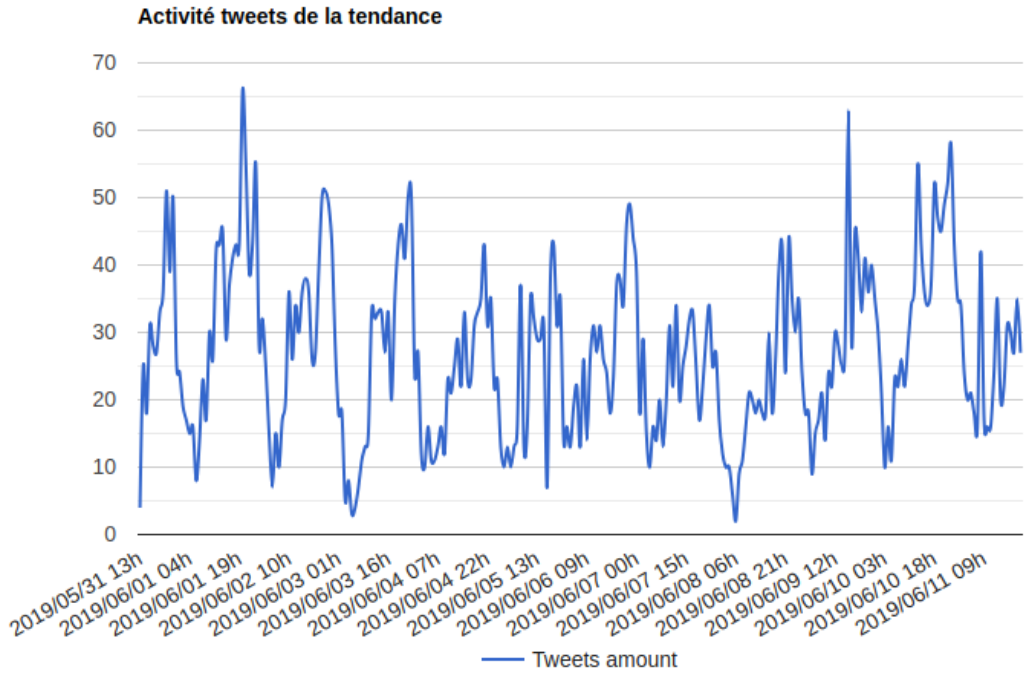


FIGURE 7.17 – Activité de la tendance "Célibataire"

L'origine du problème est qu'à chaque nouvelle clé rencontrée (ici une date + une heure précise), on doit parcourir le tableau et chercher si elle existe, ce qui devient rapidement coûteux. Il serait ici très intelligent d'utiliser un algorithme de type Map/Reduce car c'est le cas typique où l'on gagnerait énormément en performance.

Chapitre 8

Gestion de projet

Dans ce chapitre, nous présenterons la partie gestion de projet de ce dernier.

8.1 Organisation du projet, Planning et dépendance à réaliser entre les phases

Cette partie consiste à indiquer le planning global que nous souhaitons mettre en œuvre et respecter tout au long du projet.
Nous avons découpé notre projet des phases principales ayant des liens de dépendance entre les travaux à réaliser. En effet, chaque phase se suit, l'une ne pourra pas être faite sans avoir effectué la phase précédente.
Par la suite en tenant compte de l'ampleur de notre projet, du délai imparti et des ressources disponibles, les phases seront détaillées avec plus ou de moins de précision.

8.1.1 Phases du projet

Phase de pré – cadrage :

c'est la phase de découverte. Durant cette phase, deux réunions ont lieu avec le tuteur :

- la première pour une présentation globale du projet
- La deuxième pour montrer au tuteur que nous avons compris du sujet en s'échangeant les idées vis-à-vis du sujet.

Phase de cadrage :

est la phase de recherches. C'est l'étude approfondie du projet où chaque membre de l'équipe effectue des recherches : des articles, des documents ... Après un point avec le tuteur, il faut rédiger l'expression des besoins en indiquant les attentes du client (tuteur) et en expliquant les fonctionnalités que nous souhaitons mettre en place.

Phase de conception :

une phase de prise de décision et d'élaboration.
La phase de conception consiste à approfondir nos recherches en effectuant des expériences utiles à notre projet. Ces expériences permettront par la suite de nous aider à déterminer les éléments nécessaires au projet comme la base de données, langage de programmation, outils, etc...

Phase de développement/Test

C'est la phase de construction, c'est la production du rendu final. Nous développons le projet dans l'ordre des fonctionnalités que nous avons cité dans la phase précédente. Nous vérifierons au fur et à mesure la conformité du résultat obtenu à la fin de chaque fonctionnalité développée.

Phase de release Finale

Cette phase consiste principalement à se préparer à la présentation finale du projet. Nous préparerons les documents à rendre et la soutenance finale.

8.2 Listes des livrables par phases

- cahier des charges
- Planning du projet
- Rapport final
- Slide présentation du projet

8.3 Plan de charge humain

Un des risques des projets de longue durée est les ressources humaines lié à la démotivation ou à la surcharge des tâches. Pour cela, nous avons effectué un plan de charge au début de chaque tâche à réaliser.
A la fin de la tâche, nous pouvons ainsi calculer la différence entre le nombre de jour prévu et le nombre de jour réalisé.

8.4 Gestion du temps

Nous avions convenu de deux types de réunions, soit des réunions de l'équipe de développement seulement, soit avec la participation de notre tuteur.
Les réunions de l'équipe de développement étaient hebdomadaires afin de vérifier l'avancée du projet et son bon déroulement.
Le deuxième type de réunions se faisait plus rarement étant donné que notre tuteur était souvent indisponible, nous n'organisions de réunions qu'une fois que nous avions fait des avancées significatives et que nous avions besoin d'être dirigés dans notre projet. Ceci permettait aussi à notre encadrant de vérifier que nous avions bel et bien sur le projet ainsi que de jauger la qualité et la rigueur de notre travail.

8.5 Répartition des tâches

Le travail à été réparti tout d'abord en quatre puis en cinq à l'arrivée de Hong Phuc VU dans l'équipe.
Il s'agissait au départ d'un travail de recherche et de réflexion autour des outils à utiliser, du langage à privilégier ainsi que des méthodes à mettre en oeuvre pour solutionner nos problématiques.
Ensuite, une fois que tout ce travail fut fini, nous avons a réparti les nouvelles tâches de développement aux différents membres de l'équipe

8.5.1 Cycle de vie d'une tâche

FIGURE 8.1 – Cycle d'une tâche

8.5.2 Répartition lors de la phase de cadrage

Recherche des outils de travail et des langages : Mohamed II BAYO et Benoît FAGOT
Recherches concernant l'API Twitter : Lahcen AIT BELLA
Recherches concernant les méthodes d'analyse de texte : Mohamed Boudjemaa HAMICI

8.5.3 Répartition lors de la phase de développement

Hong Phuc VU ayant rejoint l'équipe au moment du début du développement, nous avons donc réparti les tâches en cinq.
Gestion de la base de données : Mohamed II Bayo
Découverte de la description : Mohamed Boudjemaa HAMICI

Découverte du lieu : Hong Phuc VU
Découverte de la date : Lahcen AIT BELLA
Interface Homme-Machine : Benoît Fagot

8.6 Outils de travail

8.6.1 Outils de communication

Pour communiquer en temps réel quand les membres de l'équipe sont distants, nous avons utilisé l'outil WhatsApp, cet outil nous a permis d'être en contact constant et de résoudre tout problème dans les plus brefs délais.

8.6.2 Outils de traçabilité

Github :

FIGURE 8.2 – gitHub

Equivalent à SVN, Github est un logiciel de versions décentralisé, nous l'utilisons principalement pour partager le code source avec PyCharm, notre IDE.

Activité et statistiques :
GitHub propose un système de statistiques sur l'activité du projet (contributions, nombre de commits, etc...) sur la branche « master » du projet.

Voici la vue globale de l'activité du projet tout au long de sa réalisation.

Excluding merges, **3 authors** have pushed **15 commits** to master and **15 commits** to all branches. On master, **170 files** have changed and there have been **16,731 additions** and **992 deletions**.

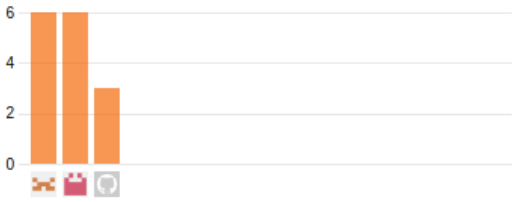


TABLE 8.1 – Nombre de commit Total

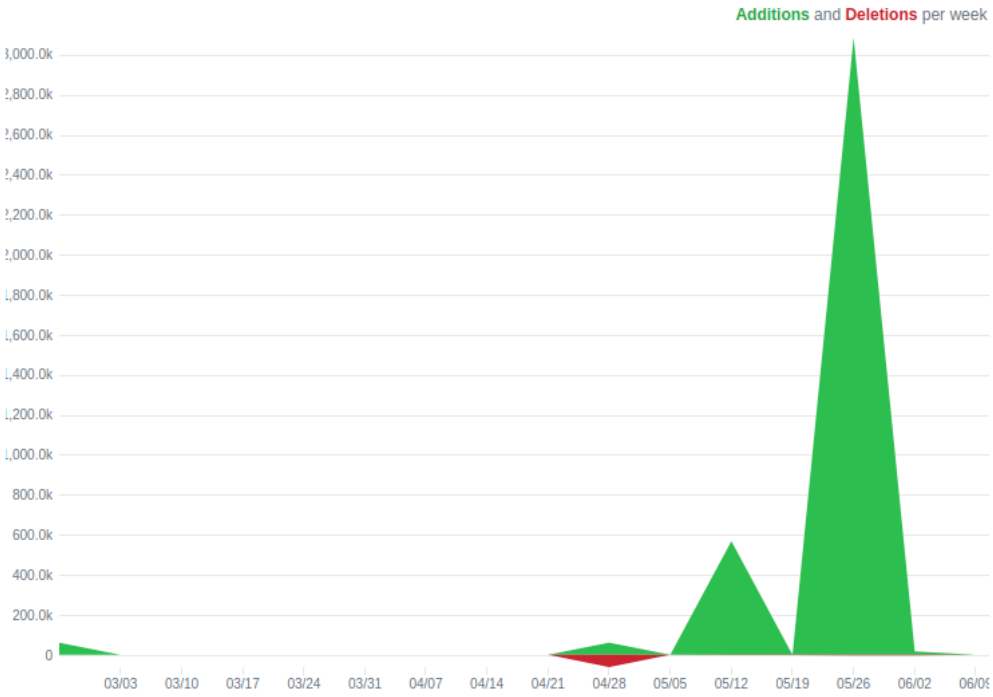


TABLE 8.2 – Nombre de commit

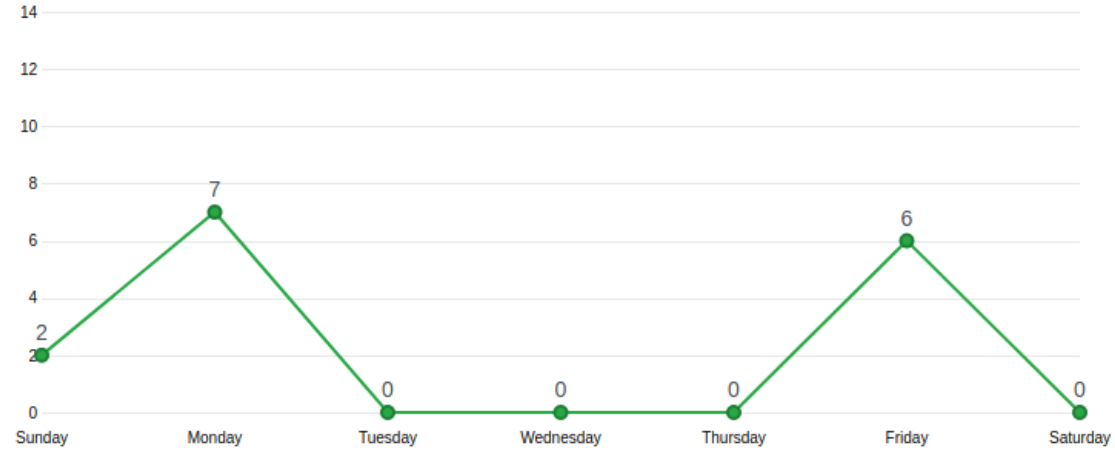


TABLE 8.3 – statistique graphique commit par semaine

TABLE 8.4 – statistique graphique commit par mois

8.6.3 Outil de gestion de projet

Trello :

Trello est un outil de gestion de projet en ligne. Il est basé sur une organisation des projets en planches listant des cartes, chacune représentant des tâches, ces cartes sont assignables à des utilisateurs et peuvent être déplacés d'une planche à l'autre.
C'est un outil puissant car il nous permet d'exposer et structurer nos idées et donc de soutenir le travail en groupe.

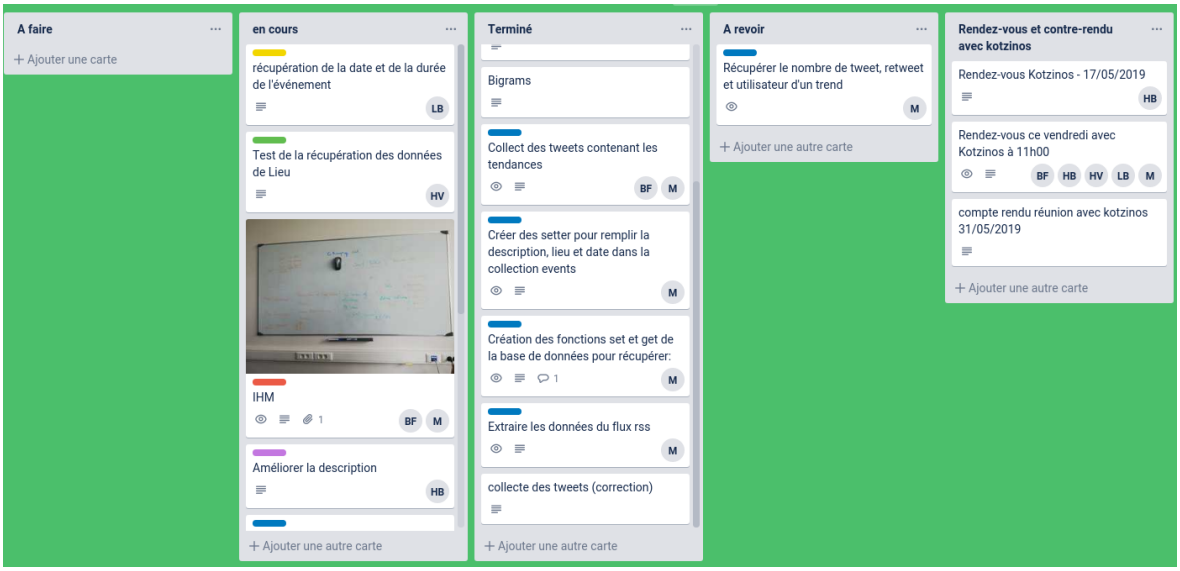


FIGURE 8.3 – Interface de Trello

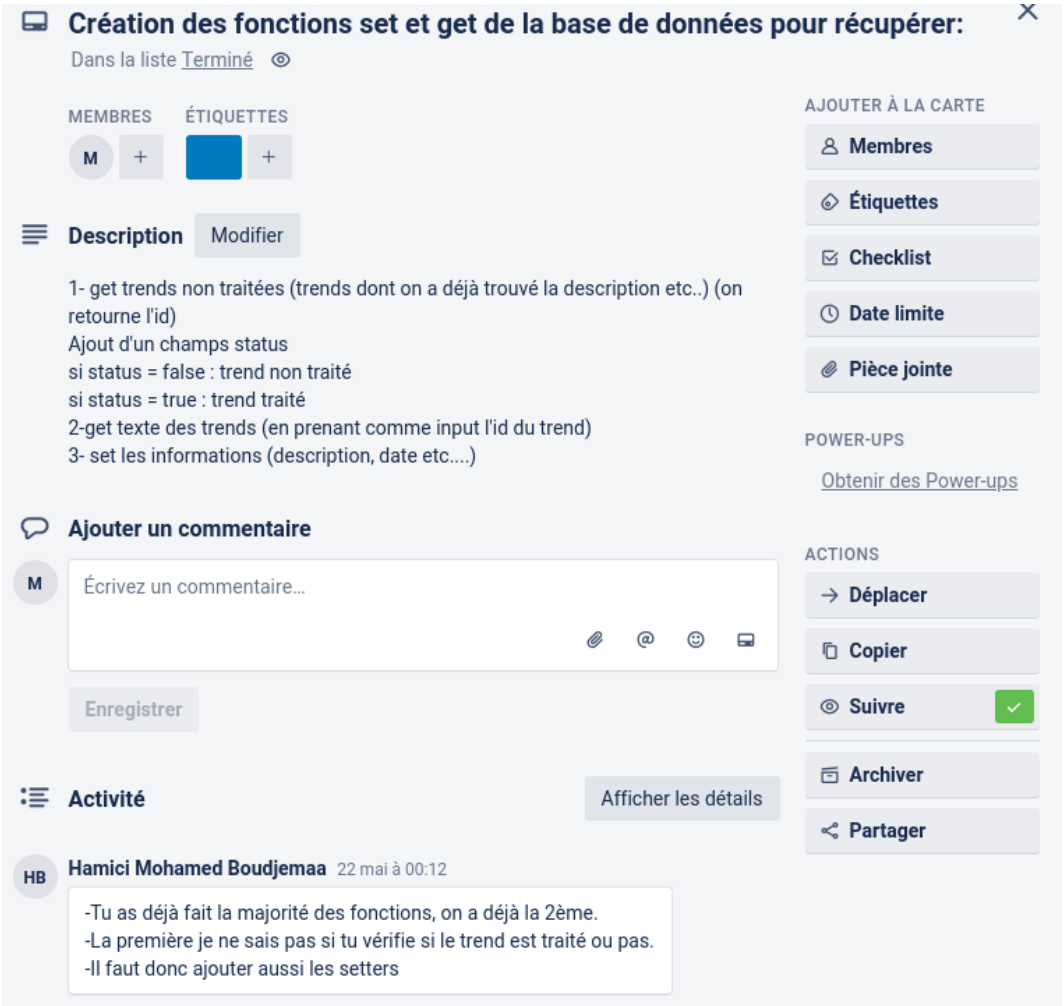


FIGURE 8.4 – Description d'une tache dans Trello(exemple de notre utilisation)

Chapitre 9

Conclusion et perspectives

9.1 Conclusion

La recherche automatique d'information sur le web est un sujet d'actualité qui nécessite la mise en oeuvre de différentes compétences informatiques, notamment en gestion de bases de données, en Intelligence Artificielle et en analyse de données. Il faut aussi prendre en compte différentes problématiques liées à la taille des données, à la vitesse de calcul et de recherche en base de données, à la fiabilité de ces données et des résultats que nous offrons à l'utilisateur.
Nous avons donc dû adresser ces différentes problématiques au fur et à mesure de l'avancée de notre projet, ce qui nous a permis de faire évoluer nos compétences sur ces différents sujets tout en mettant en application les connaissances apprises en cours tout en apprenant de nouvelles choses. Le projet nous a aussi permis de mettre en pratique notre récente formation en gestion de projet Agile afin de mener à bien ce projet.
Au terme de ce projet, nous avons réussi à utiliser Twitter et les différents flux RSS de la presse afin de découvrir automatiquement les évènements survenants dans le monde, nous avons réussi à définir ces évènements grâce aux tweets et aux flux RSS dont nous disposions, en plus de retrouver leur localisations, leur dates et durées. Nous avons donc répondu à toutes les attentes de départ du projet, nous expliciterons dans les paragraphes suivant le pourcentage de précision des informations extraites et les conclusions de l'analyse de nos résultats.

Au vu des résultats des tests unitaires effectués, nous remarquons que les résultats obtenus sont suffisamment fiables et ne posent pas de problèmes de vitesse étant faits à froid, ce qui répond aux contraintes de fiabilité de notre solution.

9.2 Perspective

Une fois notre projet terminé, nous remarquons qu'il reste beaucoup de perspectives de développement de ce dernier et qui permettraient de lui ajouter une plus-value certaine, nous allons les discuter dans ce chapitre.
La première amélioration que nous conseillons serait l'augmentation du nombre de données. Pour ce faire, on pourra utiliser d'autres réseaux sociaux comme Facebook par exemple, en plus d'augmenter le nombre d'articles grâce en prenant comme sources plus de flux RSS.
Cette amélioration serait assez simple à implémenter dépendamment de l'API fournie par le réseau social en question. En ce concerne les flux, il suffit de les trouver, ce qui ne prendra que le temps de la recherche.

Une seconde amélioration consisterait à créer une classification des événements selon leur type (culturel, sportif, éducatif) etc.. Pour cela, il faudrait utiliser des algorithmes de classification, supervisée ou non, ce qui suppose l'utilisation du machine learning notamment. Cette amélioration nécessite un travail assez soutenu de recherche afin de trouver le meilleur algorithme à utiliser et ensuite l'implémenter. Il est donc nécessaire d'avoir les compétences en Intelligence Artificielle et en Machine learning nécessaires à sa mise en oeuvre, cette amélioration est donc d'une difficulté assez élevée par rapport à la précédente.

Il faudrait penser améliorer la précision de la détection des informations que nous avons déjà en croisant celles ci avec les différentes sources ajoutées dans la première amélioration.
Pour finir, on pourrait aussi classer les événements par ordre d'importance, un tremblement de terre faisant des milliers de morts aurait par exemple plus d'importance que le concert d'un artiste régional. Pour mettre en oeuvre cette information, il faudra donc trouver le moyen de comparer les différentes sources d'informations concernant un même événement, soit découvrir les différents articles ou textes concernant l'événement, puis extraire les différentes informations et enfin les comparer. Ceci est d'une difficulté moyenne voire haute, car il faut rechercher les différents moyens existants pour comparer ces différentes sources puis les implémenter et tester leur efficacité. Nous pensons pour le moment à utiliser des algorithmes de comparaison de textes comme 'Jaccard' ce qui serait simple à réaliser, néanmoins, si cela ne donne pas de résultats suffisamment précis nous devrons utiliser d'autres moyens que nous ne connaissons pas encore, ce qui rendrait cette amélioration plus complexe à réaliser.

Bibliographie

[1] URL : https://hightech.bfmtv.com/internet/comment-twitter-detecte-les-tremblements-de-terre-920942.html.

[2] URL : https://mattdodge.codes/twitter-rss-xml/.

[3] URL : http://tinyurl.com/zcbqaox.

[4] URL : https://www.mongodb.com/cloud/atlas.

[5] URL : https://www.nltk.org/.

[6] Dirk AHLERS. "Assessment of the accuracy of GeoNames gazetteer data, Proceedings of the GIR Workshop". In : (2013), p. 74–81.

[7] Andrei Z. BRODER et al. "Syntactic clustering of the web. Computer Networks and ISDN Systems". In : 29.8 (1997), p. 1157–1166.

[8] ELASTIC. "Elastic Cloud : Hosted Elasticsearch, Hosted Search — Elastic". In : (2019). DOI : Elastic.co.

[9] ELASTIC. "Open Source, Distributed, RESTful Search Engine. Contribute to elastic/elasticsearch development by creating an account on GitHub". In : (2019).

[10] Jan GOYVAERTS. "Regular Expression Tutorial - Learn How to Use Regular Expressions". In : (2005). DOI : www.regular-expressions.info.

[11] Julie Beth LOVINS. "Development of a stemming algorithm. Mechanical Translation and Computational Linguistics". In : 11 (1968), p. 22–31.

[12] Vipin Kumar PANG-NING TAN Michael Steinbach. "Introduction to Data Mining". In : (2005).