

Université de Cergy-Pontoise

M1 IISC

PROJET DE SYNTHÈSE

Web Sensor : système d'événements intelligent



Rapporteur :

Dan Vodislav

Tuteurs techniques :

Dimitris KOTZINOS

Wassim SWAILEH

Encadrant de gestion de projet :

Tianxiao LIU

rédigé par :

Mathieu VOISIN

Lydia KHELFANE

Gabriel CHEVALLIER

Martin GUILBERT-LEJEUNE

Juin 2020

Remerciements

Nous tenons à remercier M. Kotzinos ainsi que M. Swaileh pour nous avoir tutoré tout au long de ce projet, sans qui le projet n'aurait donc pas vu le jour, et qui ont pu nous aiguiller tout au long de celui-ci.

Nous tenons aussi à adresser nos remerciements à M. Vodislav, qui a accepté d'être notre rapporteur.

Et pour finir, nous voulons aussi adresser nos remerciements à M. Liu, qui nous a encadré du début à la fin du projet, et dont les remarques et conseils ont été très instructifs pour nous.

Table des matières

Remerciements	1
Glossaire	4
Chapitre 1 : Introduction	5
1.1 : Contexte du projet	5
1.2 : Objectifs du projet	6
1.3 : Mise en scénario	7
1.4 : Organisation du rapport	8
Chapitre 2 : Présentation et spécification du projet	9
2.1 : Étude du marché	9
2.2 : Fonctionnalités attendues	10
2.3 : Conception globale du projet	12
2.3.1 Vue utilisateur	12
2.3.2 Architecture technique	15
2.4 : Problématiques identifiées et solutions envisagées	16
2.5 : Environnement de travail	17
Chapitre 3 : Stockage des données	19
3.1 : Analyse de la problématique	19
3.2 : Etat de l'art : études des solutions existantes	19
3.3 : Solution proposée et sa mise en œuvre	21
3.3.1 : La récupération de tweets	21
3.3.2 : MongoDB: Une base de données orientée documents	22
3.3.3 : MySQL: Une base de données relationnelle	23
3.4 : Certification de la solution	24
Chapitre 4 : Traitement de texte brute	26
4.1 : Analyse de la problématique	26
4.2 : Etat de l'art du traitement de texte	26
4.3 : Solutions apportées et mises en œuvre	28
4.3.1 : Natural Language Processing (NLP)	28
4.3.2 : Vectorisation des données	29
4.4 : Tests et certifications de la solution	33
Chapitre 5 : Clustering des données	35
5.1 : Analyse de la problématique	35
5.2 : Etat de l'art : études des solutions existantes	36
5.3 : Solution proposée et sa mise en œuvre	42
5.4 : Tests et certifications de la solution	43
5.4.1 Jeu de données	43

5.4.2 Les phases de tests	44
5.4.3 Les résultats	45
Première phase	45
Deuxième phase	50
Troisième phase	54
Chapitre 6 : Prédiction d'événements populaires	58
6.1 : Analyse de la problématique	58
6.1.1 Pourquoi prédire.	58
6.1.2 Quoi prédire.	58
6.2 : Etat de l'art des prédictions	59
6.3 : Solutions apportées et mises en œuvre	66
6.4 : Tests et certifications de la solution	67
Chapitre 7 : Rendu final	68
7.1 : Interface utilisateur finale	68
7.2 : Tests utilisateur et certification	70
7.3 : Certification : grand volume de données	71
Chapitre 8 : Gestion de projet	73
8.1 : Méthode de gestion	73
8.2 : Répartition de tâches	74
8.3 : Gestion de réunions et organisation de l'équipe	75
8.4 : Utilisation des outils de gestion de projet au sein de l'équipe	76
8.5 : Décomposition du projet en features et stories	78
Chapitre 9 : Conclusion et perspectives	79
9.1 : Conclusion	79
9.2 : Perspectives	80
Bibliographies	82

Glossaire

1. **Event:** Un event est un événement auxquels nous allons nous intéresser dans le cadre de notre projet, cela peut-être factuel (Compétition sportive, etc.) ou un phénomène faisant parler de lui (Tremblement de terre, Epidémie d'un coronavirus, etc.). Il dépend de sa popularité sur Twitter (calculée avec le nombre de tweets et d'utilisateurs).
2. **Samples:** Un sample correspond à un échantillon de données.
3. **Cluster:** Un cluster est un regroupement d'un ensemble de données similaires.
4. **Vectorisation:** La vectorisation est l'action de transformer des mots en un vecteur.
5. **NLP (Natural Language Processing):** Ensemble d'outils permettant le traitement de la langue naturelle.

Chapitre 1 : Introduction

Ce chapitre va nous permettre d'introduire notre projet. Pour cela, nous allons commencer par le contextualiser, ce qui nous amènera à parler des objectifs du projet ainsi qu'à en faire une mise en scénario. Nous concluons cette partie par l'organisation du rapport.

1.1 : Contexte du projet

Le traitement des données joue un rôle de plus en plus important dans la vie de tous les jours, que ce soit pour pousser à l'achat avec des pubs visées, ou bien pour faire des recommandations sur netflix. Un grand nombre de données est traité chaque jour automatiquement dans le but de nous simplifier la vie. Parmi les données pouvant être utilisés, on retrouve les données personnelles pouvant être entrées sur certains sites, qui vont être stockés et traités (et peuvent être vendus), mais aussi sur les réseaux sociaux où un grand nombre d'utilisateurs postent des informations tous les jours.

Chaque jour, le volume de données numériques augmente à grande vitesse, ceci est dû à l'augmentation des activités numériques, notamment depuis l'apparition des réseaux sociaux, mais aussi avec l'augmentation du streaming sur différentes plateformes telles que Netflix ou Youtube.

Quelques chiffres pour se représenter l'évolution du big data :

- Selon McKinsey Global Institute, le volume mondial de données double tous les trois ans.
- IDC estime qu'en 2020 chaque personne sur Terre générera 1,7 megabytes de données par seconde.
- IDC estime aussi à plus de 203 milliards de dollars l'argent généré par le traitement des données d'ici fin 2020.

On voit donc bien que les données numériques sont de plus en plus présentes dans nos vies en suivant une évolution exponentielle, qu'elles soient professionnelles (dans le cadre d'une entreprise) ou bien personnelles. De plus en plus d'industries ont besoin de ces données de masse pour se développer, comme par exemple l'industrie médicale, qui grâce aux données collectées peut avoir une meilleure vision des différents traitements, ou encore l'énergie par exemple, qui, en utilisant les données d'utilisation énergétique des gens sur le long terme, peut déterminer comment adapter l'offre par rapport aux besoins des consommateurs.

Twitter, de par sa nature en temps réel, est devenu un média d'information qui permet de nombreuses recherches sur divers thèmes, tels que le sport, la politique ou encore les catastrophes naturelles. Par exemple, lorsqu'un séisme se produit quelque part dans le monde, de nombreux utilisateurs tweetent sur cet événement en temps réel, ce qui permet de suivre l'information au fur et à mesure du déroulement de l'événement.

Dans ce contexte, les informations postées sur des réseaux sociaux peuvent être considérées comme des données précieuses pouvant être utilisées pour de nombreuses choses. Nous allons donc nous essayer à l'exercice du traitement des données en masse pour détecter et prédire des événements depuis les données fournies par des utilisateurs.

1.2 : Objectifs du projet

L'objectif du projet est de développer une application, qui utilise les réseaux sociaux, en l'occurrence twitter, comme une source d'information à grande échelle, où chaque utilisateur est l'équivalent d'un capteur. En effet, on considère que chaque utilisateur qui poste du contenu sur twitter apporte une information, qui sera jugée pertinente ou non par l'application, et qui pourra être associée à d'autres tweets similaires afin de pouvoir repérer les différents événements dont parlent les utilisateurs de twitter. Chaque tweet est donc bien traité comme s'il s'agissait d'une mesure effectuée par un capteur, et l'ensemble des tweets récupérés permettent l'étude des tendances globales. On peut faire une analogie avec la météo par exemple, et comparer chaque tweet avec un relevé de température. Un relevé seul apporte peu d'informations et n'est pas forcément fiable, mais lorsqu'on a des milliers de relevés répartis à la fois sur une zone géographique (Qui sera pour nous la France) ainsi qu'une durée, on peut créer des modèles qui permettent un traitement de ces données fiables.

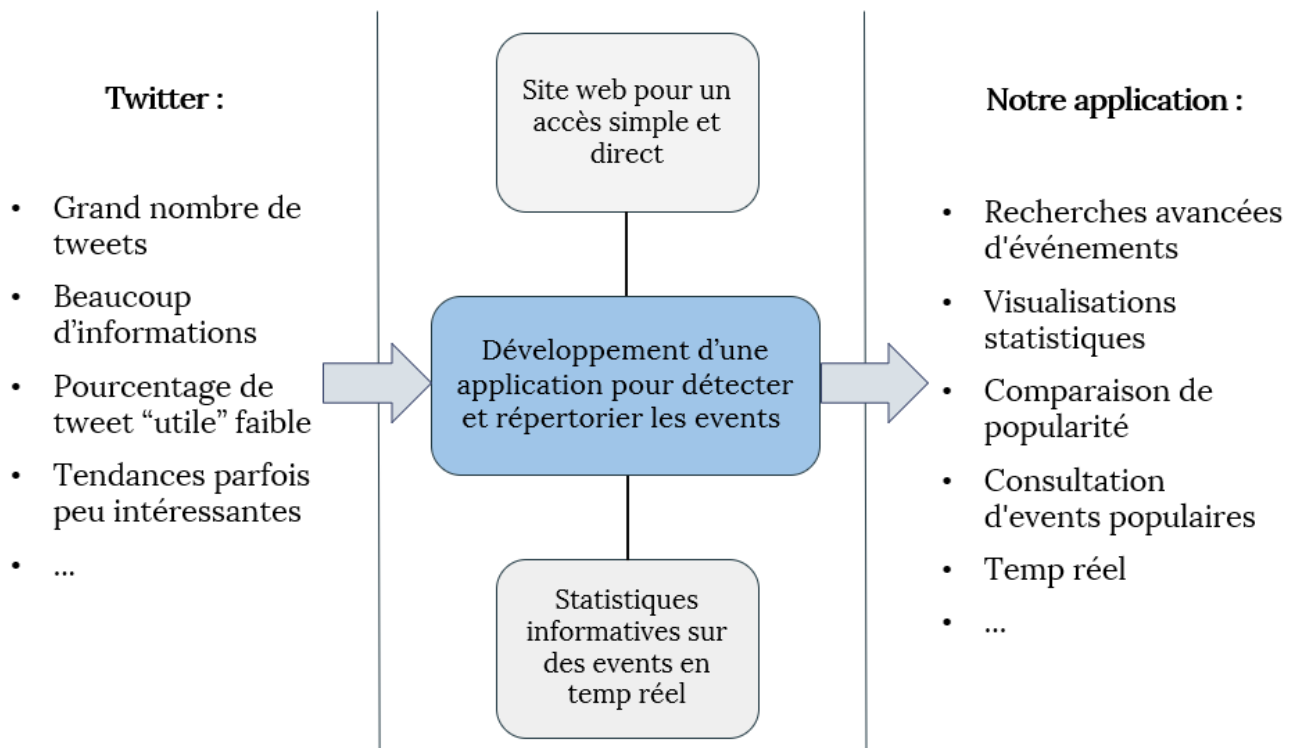


Figure 1: Objectif du projet

Comme indiqué sur le schéma ci-dessus, nous utilisons le site Twitter comme source de données. Ces données ont pour avantages d'être très nombreuses, car même avec la limitation

de récupération de tweets imposé par l'API de Twitter, nous pouvons récupérer un nombre de tweet très intéressant. Ces tweets nous apportent de nombreuses informations en tous genres. Cependant, un grand nombre de ces informations sont inutiles et les tendances twitter sont parfois peu intéressantes.

C'est pour cela que notre application, sous forme d'un site web, permettra la visualisation d'informations et de statistiques en temps réel. Ces informations et statistiques peuvent être sous différentes formes, comme par exemple l'affichage de tweets populaires sur une tendance, ou encore des comparaisons de popularité d'événement.

1.3 : Mise en scénario

Dans cette section nous allons voir un exemple d'utilisation de notre projet. Nous verrons donc les différentes actions possibles sur notre application web.

Sur la figure 2, nous pouvons voir l'importance d'avoir un site qui répertorie les événements en temps réel, ce qui permet aux utilisateurs de s'informer plus rapidement et d'être à la pointe de l'actualité.

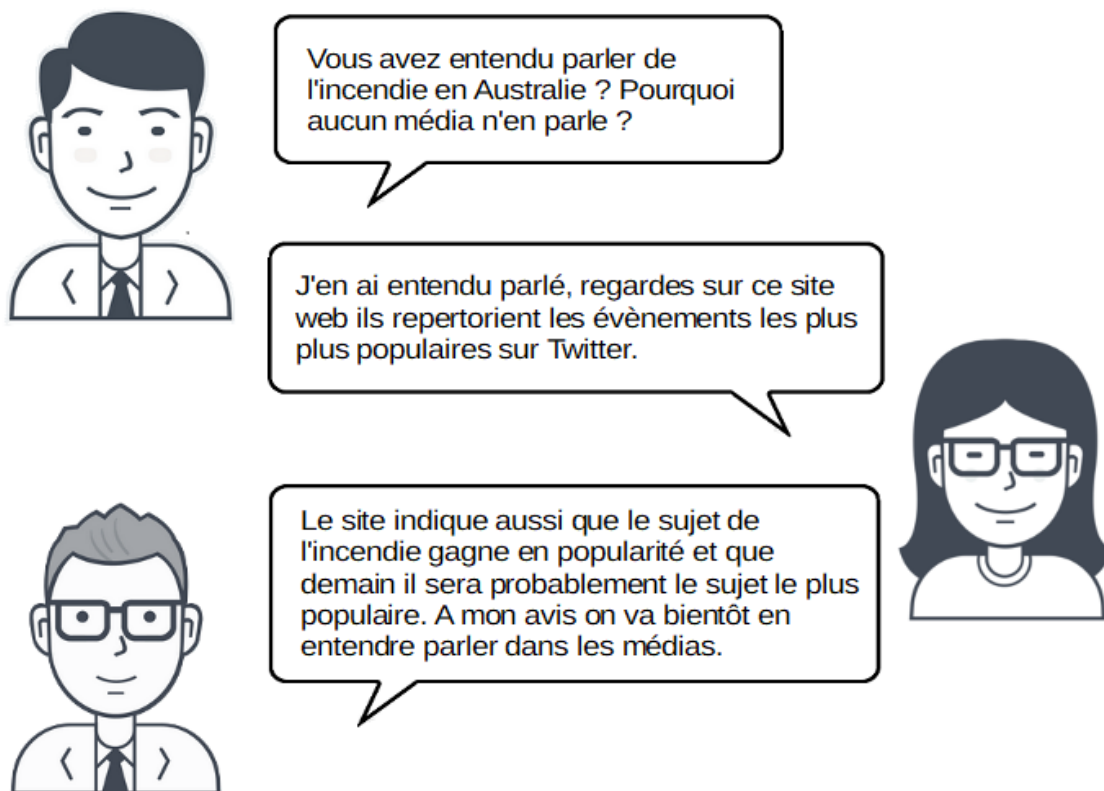


Figure 2: Exemple de scénario d'utilisation de l'application

L'utilisateur peut visualiser les events en temps réel avec leurs détails : nom, catégorie, popularité... etc. Il a aussi accès à un graphique représentant l'évolution de la popularité d'un event sur twitter. De plus, il peut exécuter les principales actions disponibles telles que :

- Faire une recherche par nom pour trouver les events liés.
- Consulter les events les plus populaires, c'est-à-dire des events avec un grand nombre de tweet / retweet.
- Comparer des events selon leurs popularités. Nous définissons la popularité d'un event par le nombre de tweet liée à cet event, plus il y en a, plus l'event est populaire.

Ces actions permettent à l'utilisateur d'observer les events sous plusieurs angles. L'utilisateur pourra voir précisément l'event qui l'intéresse avec la recherche ou voir directement les events les plus populaires. De plus il pourra, s'il le souhaite, voir les détails de chaque event, ce qui lui permettra de voir si l'event est populaire ou non, ainsi que sa localisation géographique et d'autres données liées. En comparant plusieurs events, il pourra observer l'évolution de la popularité de chacun, pour par exemple voir quels events attireront le plus de monde. Ces actions seront plus amplement décrites dans la [section 2.2](#).

1.4 : Organisation du rapport

Ce rapport est organisé en 7 chapitres, où chacun des chapitres aborde un aspect différent du projet.

Le chapitre 2 correspond à notre cahier des charges du projet. Il contiendra donc les détails de ce qu'il fallait faire pour ce projet, ainsi que les spécifications techniques nécessaires à la bonne compréhension du sujet.

Les chapitres 3, 4, 5 et 6 sont des chapitres techniques concernant respectivement le stockage de nos données, le traitement de ces données, le clustering ainsi que la prédiction. Nous y verrons les problématiques liées à ces domaines, et nous y détaillerons nos solutions apportées.

Dans le chapitre 7 nous aborderons le résultat final de notre projet, en présentant l'interface homme/machine ainsi que les informations concernant les tests de "certification" effectuées sur notre projet.

Le chapitre 8 nous permettra de présenter notre méthode de gestion de projet. Nous expliciterons donc notre mode de fonctionnement tout au long du projet, et nous allons y comparer nos prévisions (ex: planification de releases) avec ce qui a été réalisé.

Le chapitre 9 conclura ce rapport et permettra d'évoquer les différentes perspectives d'évolution du projet.

Chapitre 2 : Présentation et spécification du projet

Afin de bien commencer le projet, nous avons dû longuement réfléchir sur de nombreux points qui sont indispensables au bon développement du projet. Nous allons donc voir dans cette partie dans un premier temps les solutions semblables à notre projet déjà existantes, puis nous verrons les fonctionnalités et notre conception du projet qui permet de nous différencier et d'avoir une idée précise de ce que nous allons produire. Pour finir nous verrons les problématiques que nous avons identifiées, ainsi que les solutions que nous y apporterons.

2.1 : Étude du marché

Il existe de multiples projets utilisant Twitter comme source de données. En effet pour toutes les raisons vues dans le [chapitre 1](#), Twitter est un réseau social le plus fréquemment mis à jour environ 500 millions de messages sont postés quotidiennement sur sa plateforme. Il peut être vu comme un réseau de plusieurs milliers de capteurs qui, en temps réel, donnent différentes informations. Par exemple, Sahaki et al. ont créé un système d'alerte qui avertit lorsqu'un tremblement de terre ou une tempête frappe le Japon, en analysant les tweets des utilisateurs Japonais et en estimant avec, la position de la catastrophe [1]. Ainsi leur méthode pour estimer l'emplacement de l'épicentre d'un tremblement de terre est plus rapide et presque aussi précise que le système déjà mis en place au Japon.

Twitter dispose de plusieurs APIs permettant de requêter sa base de données, mais aussi de construire des services au-dessus de sa plateforme. Ces APIs sont particulièrement riches en retournant presque une centaine de variables par requête ; les données concernent les tweets (date de publication, le texte du message, etc.), l'auteur (date de création du compte, pseudo...), les entités contenus dans les messages (hashtags, mentions, urls...) et des informations de localisation (pays, timezone, longitude / latitude).

L'estimation de l'emplacement d'un event est une problématique fondamentale étant donné que les tweets forment un réseau de données dense et avec des positions géographiques variables. De plus pouvoir filtrer efficacement les tweets pour avoir une sélection qui correspond au sujet voulu est important, M. Joachim propose une méthode d'apprentissage automatique pour catégoriser du texte [2]. Il présente une machine à vecteurs de support (SVM) qui est utilisé par Sahaki et al. [1] pour catégoriser les tweets. Achrekar et al. présentent un framework qui affiche les tweets provenant des États-Unis qui mentionnent que l'utilisateur a un état grippal. Le framework va ainsi surveiller et prévoir la propagation de la grippe sur la population américaine [3]. Leur méthode de prédiction est ainsi très performante, car les résultats obtenus sont à 98% similaires aux résultats du centre de santé américain.

En lisant ces articles nous percevons certaines des problématiques que nous allons rencontrer au cours de ce projet, tel que la classification des données, ainsi que l'application de méthodes de prédiction d'events.

2.2 : Fonctionnalités attendues

Dans cette partie nous allons détailler toutes les fonctionnalités attendues dans notre projet.

→ Recherche d'événements:

L'utilisateur pourra rechercher un événement de plusieurs façons différentes. La plus simple concerne les événements populaires du moment. Ils seront affichés dans un fil d'actualité qui permet de suivre les tendances. (ex: pendant la coupe du monde, il est inutile de chercher par mot-clé coupe du monde, il s'agit d'un événement très populaire qui sera mis en avant automatiquement par le site web).

La seconde façon de rechercher un événement est via une barre de recherche. Cette barre de recherche nous permettra de faire une recherche, par mot-clé, sur n'importe quel événement qui a été enregistré par le site web. Cette recherche sera accompagnée de différents filtres, sur le thème de l'événement (Sport / Spectacle...) par exemple.

→ Visualisation d'événements:

L'utilisateur pourra visualiser les événements depuis le site web. Ces événements sont extraits à partir du réseau social twitter grâce à une analyse du texte de chaque tweet ainsi qu'à leur popularité. En effet les événements proposés sur le site seront les plus pertinents sur le réseau social.

L'utilisateur pourra donc avoir les détails concernant un événement. Ces détails seront divisés en deux catégories:

- Les informations textuelles: sur la page d'information, chaque fiche d'événement comportera ces informations : nom de l'événement, sa catégorie (Musique/Sport, etc.), sa popularité ainsi que le tweet le plus populaire par rapport à cet événement. Celle-ci va permettre à l'utilisateur d'avoir un accès direct aux informations les plus importantes concernant l'événement qui l'intéresse.
- Les graphiques: Pour chaque événement, on pourra visualiser des graphiques concernant l'évolution de la tendance, en fonction du temps ou par rapport à un lieu par exemple. Il y aura un choix de graphique prédéfini pour chaque événement, ce qui permet à l'utilisateur d'accéder à l'information qu'il veut, avec la vue qu'il veut sur cette information.

Cette visualisation permet donc un suivi d'un événement, ainsi qu'à l'obtention des informations en temps réel sur l'événement. S'il se passe quelque chose en direct (Par exemple un attentat, ou une catastrophe naturelle), les informations évolueront rapidement et permettront un réel suivi en direct de l'événement, notamment avec la notion de "tweet populaire" qui permettra de voir dans la partie d'information textuelle le tweet le plus représentatif de l'événement (news, photo, etc.).

→ Comparaison d'événements:

L'utilisateur peut comparer la popularité des événements sur une période donnée. Le résultat de la comparaison sera affichée sous forme de différents graphiques en courbe. Cette comparaison n'a de sens seulement si les deux événements à comparer ont un lien, plus ou moins fort, entre eux. Par exemple, comparer la popularité d'un événement sportif avec une nouvelle comme la mort d'un chanteur a peu d'intérêt.

La figure 3 représente un exemple de comparaison entre l'événement "Coupe du monde de football" et l'événement "Coupe du monde de Rugby". On peut y voir que sur le mois de juillet, correspondant à l'anniversaire de la victoire de la coupe du monde par la France il y a un pic de tweet. Tandis que l'événement "Coupe du monde de Rugby" atteint un nombre de tweet plus important sur la période de septembre. Si on compare les deux pics, on voit que la coupe du monde de rugby est bien plus populaire que la coupe du monde de football sur cette période. Ceci est dû au fait que l'événement concernant le rugby était d'actualité. Ce type de comparaison permet de présenter les résultats différemment, et est intéressant pour avoir une idée concrète de la popularité d'un événement par rapport à un autre événement.

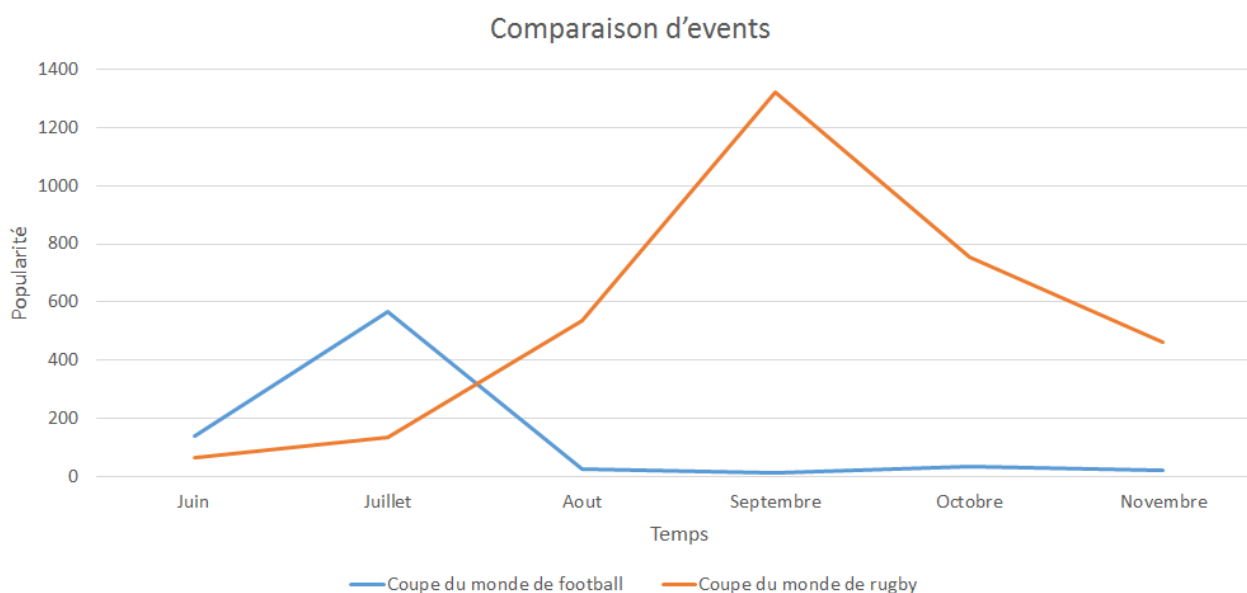


Figure 3: Exemple de comparaison entre deux événements

De plus, on pourra voir la popularité de ces événements sur une période donnée avec un graphique en barres, où chaque barre correspond au nombre de tweet total concernant un événement sur cette période, afin d'avoir un point de vue global sur les événements.

→ Visualisation et prédiction des événements les plus populaires:

Chaque jour, un "top 10" des événements les plus populaires sera mis en place, ce qui permettra à l'utilisateur de pouvoir voir un classement des événements du jour. De plus, un système de

prédiction de classement sera mis en place pour estimer quels sont les events qui peuvent prétendre au haut du classement pour la journée suivante.

La figure 4 est un diagramme qui résume les fonctionnalités que l'utilisateur pourra effectuer sur le site.

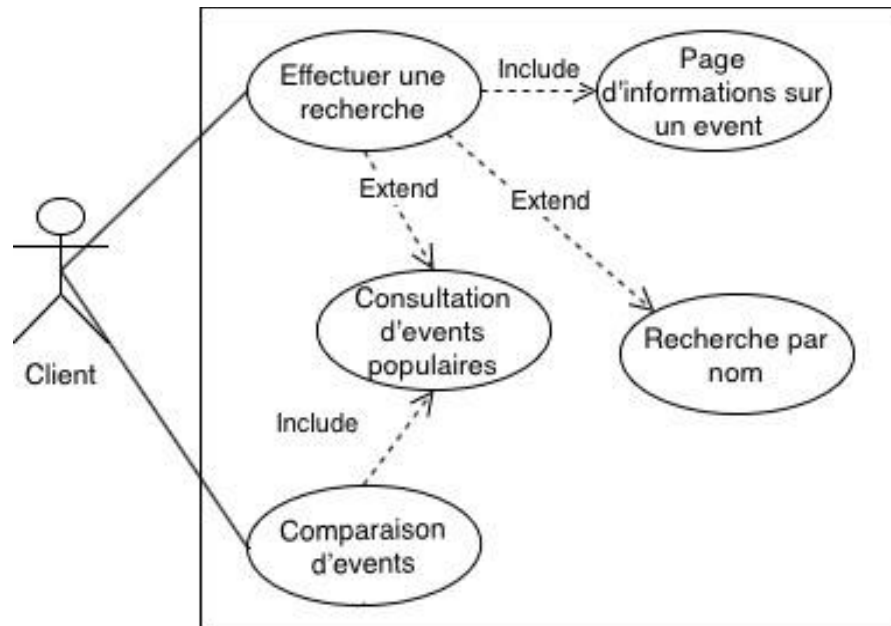


Figure 4: Diagramme d'utilisation du site web

2.3 : Conception globale du projet

Nous allons voir dans cette partie comment nous avons conçu notre projet pour intégrer ces fonctionnalités. Nous parlerons en premier lieu du point de vue de l'utilisateur, en expliquant certains choix que nous avons fait au niveau de l'interface utilisateur qui nous semblent importants. Ensuite nous verrons notre conception d'un point de vue plus technique.

2.3.1 Vue utilisateur

L'application sera accessible pour les utilisateurs via un site web. Nous avons décidé de mettre en place cette application sur un site web pour différentes raisons qui nous semblaient intéressantes:

- Accessibilité: Le fait de mettre en place un site web permet à l'application d'être facilement accessible de n'importe où, et par n'importe qui. De plus, un serveur web permet un accès multiple (non limité à 1 personne à la fois).

- Aucun téléchargement: L'application est directement utilisable du moment que l'on a un accès internet, il n'y a rien à télécharger ou à installer (hormis le navigateur web) pour profiter pleinement des fonctionnalités.
- Facilités de mise à jour: Les données peuvent être mises à jour facilement sans impacter l'utilisation de l'application. Cela permet de ne pas avoir (ou peu) de maintenances sur le site, coupant l'accès à l'application. De plus, cela force tout le monde à avoir la même version de l'application.

Le site Web contient 3 pages toujours accessibles durant la navigation comme sur le plan du site ci-dessous.

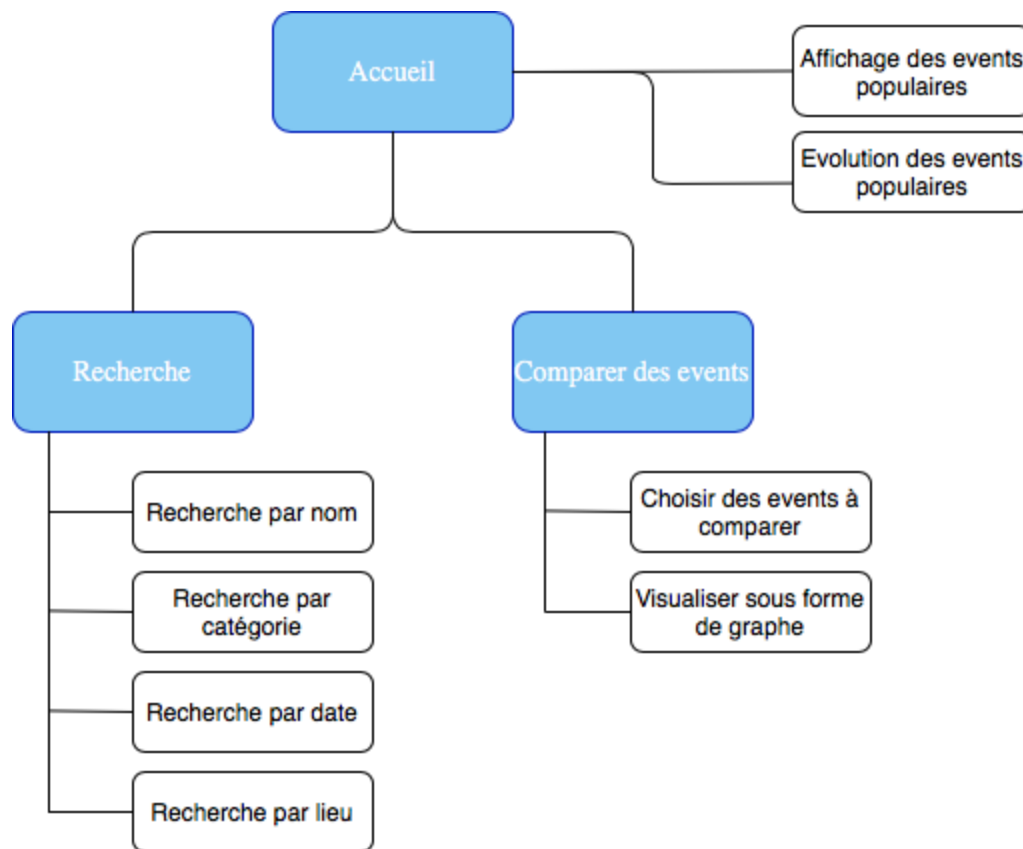


Figure 5 : Illustration général du site

En arrivant sur le site web, l'utilisateur aura comme première vue un fil d'actualité comportant les events les plus populaires du moment et l'évolution de leur popularité . Depuis cette page, l'utilisateur peut accéder aux pages permettant de faire une recherche ou de faire une comparaison.

Si l'utilisateur sélectionne la rubrique comparer des events, il sera redirigé vers une page qui sera divisée en 4 blocs principaux:

- Bloc 1 (en vert): Il s'agit du menu du site web, il est disponible depuis chacune des pages du site web.
- Bloc 2 (en rouge): Il s'agit des informations détaillées de chaque event avec lesquels on souhaite réaliser une comparaison. On y retrouve le nom de l'événement, sa catégorie (Sport / Musique etc.), sa popularité, ainsi que le tweet le plus populaire concernant cet event.
- Bloc 3 (en bleu): Il s'agit d'une représentation graphique de comparaison des events sélectionnés. Par défaut on pourra y voir l'évolution de la popularité sur une période donnée, mais ce graphique peut être changé via une liste en dessous qui permettra de sélectionner le graphique souhaité. (Voir [7.1 Interface utilisateur finale](#) afin de visualiser tous les types de graphiques possibles).
- Bloc 4 (en jaune): Contient quatre boutons qui permettent de choisir la période sur laquelle effectuer la comparaison, ajouter ou supprimer un event au graphique et enfin changer le type de graphe qu'on veut visualiser.

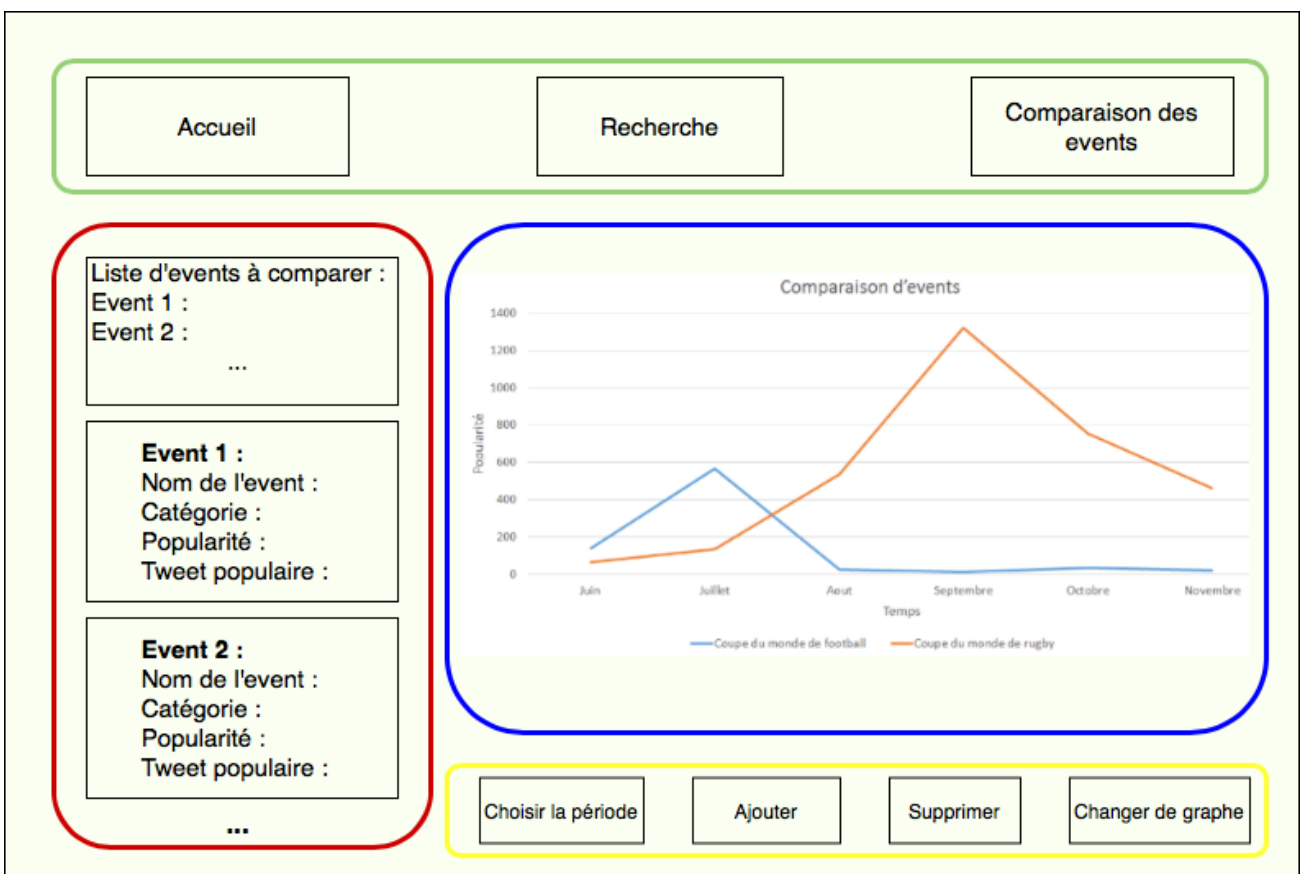


Figure 6: Schéma de l'interface concernant un Event

Ce choix de présentation semble pour nous très intuitif et ergonomique, et permet de repérer très facilement les éléments importants de la page. Le but est de simplifier un maximum l'interface pour améliorer l'expérience de l'utilisateur sur le site web.

Dans le cas où l'utilisateur clique sur un des boutons du menu, il sera redirigé vers la page correspondante à la fonctionnalité (Détailée dans la partie [2.2 Fonctionnalités attendus](#)). Chacune de ces pages sera présentée et détaillée dans la partie [7.1 Interface utilisateur finale](#).

2.3.2 Architecture technique

Pour réaliser les fonctionnalités vues précédemment, nous avons dû réfléchir à un ensemble d'aspects techniques. Ces aspects techniques permettent le bon fonctionnement du site web.

Le premier aspect technique est la récupération de tweets. En effet, notre base de données contiendra un grand nombre de tweets afin d'être plus complète et rendre le site web plus confortable. Pour cela, nous utiliserons une API fournis par Twitter qui permet de récupérer un fichier au format json qui contient les données liées aux tweets.

Une fois ces données récupérées au format json, allons les stocker dans une base de données de type MongoDB, ensuite nous aurons des algorithmes de Natural Language Processing (NLP) qui permettront de faire un pré traitement de texte. Une fois le pré traitement terminé, on va utiliser un réseau de neurones non supervisé afin de classifier chaque tweet, et de déterminer s'il s'agit d'un événement ou non.

On peut alors stocker l'événement dans une base de données SQL, et y associer les informations nécessaires (popularité, tweets associés...), afin de pouvoir permettre à l'application de récupérer ces données déjà traitées et de simplement les afficher sous différentes formes de graphiques.

On peut donc résumer tous ces aspects et leur déroulement à l'aide du diagramme d'activité suivant:

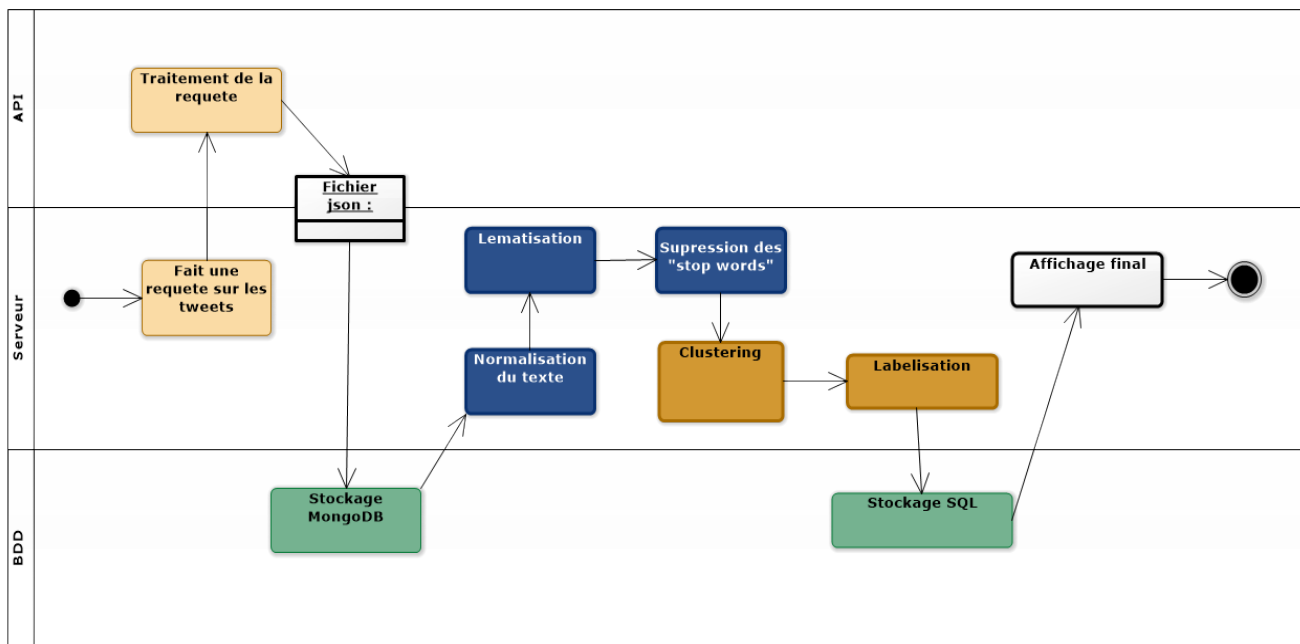


Figure 7: Diagramme de l'enchaînement des différentes parties techniques du système

Nous avons donc deux modules techniques qui vont entrer en jeu dans la conception de ce projet. Le premier est le module de Base de Données qui est nécessaire pour le stockage des informations, et le second est l'IA qui va nous permettre de déterminer quelles informations récupérées de Twitter sont utiles pour l'application (S'il s'agit d'un event ou non).

2.4 : Problématiques identifiées et solutions envisagées

Cette partie concerne les principales problématiques liées à la conception de notre projet ainsi que leurs solutions techniques pour répondre à nos besoins.

→ Quelle est notre définition d'un event ?

La première question que nous avons eu à nous poser était de savoir qu'est-ce qu'un event dans le cadre de ce projet. Pour cela, nous avons établi une liste de caractéristiques pouvant décrire un event et dans un premier lieu nous avons pensé déterminer un event par la reconnaissance textuelle d'un lieu et d'une date dans les discussions. Finalement à cause de la rareté de ces informations nous nous sommes concentré plutôt sur les phénomènes en repérant les similitudes de mots considérés importants dans les discussions.

→ Comment déterminer si un tweet concerne un event ?

Pour déterminer si le tweet nous intéresse nous allons classifier le tweet en question pour savoir s'il est associé à un évènement.

Pour cela, une fois les tweets récupérés, nous allons utiliser des méthodes de traitement de texte afin de nettoyer le texte des tweets pour ensuite les donner en entrée à un réseau de neurones qui créera des clusters, donc les classifieront en les regroupant par similarité. Nous déterminerons ensuite les catégories qui nous intéressent.

→ **Comment récupérer un contenu public sur le réseau social ?**

Nous allons utiliser une API (Application Programming Interface) Twitter qui nécessite un compte Twitter que nous allons créer pour le projet. Nous allons devoir le déclarer “compte développeur” en remplissant un formulaire et créer une interface de programmation via un autre formulaire Twitter.

Pour récupérer le contenu public du réseau social nous pourrons ensuite faire des requêtes grâce à un script python et la librairie Twython pour récupérer des tweets qui s'accumuleront par les nombreuses requêtes au cours du projet.

→ **Quel format de données utiliser pour que nos données soient traitables et compatibles avec différents systèmes et comment les stocker ?**

Le format Json (JavaScript Object Notation) permet de représenter de l'information structurée. L'API Twitter utilisée renvoie un objet comportant un nombre de tweets et leurs informations au format Json. Ce format est facilement traitable par un script python grâce à sa compatibilité avec les dictionnaires de données. Ce format est aussi utilisé dans les bases de données orientées document ainsi que dans de multiples bibliothèques graphiques pour le web. Nous utiliserons également le langage SQL pour les données traitées afin de faciliter la création de requêtes plus complexe depuis notre application web.

2.5 : Environnement de travail

Notre environnement de travail est basé sur un matériel simple, il est composé de logiciels et d'outils utiles au développement de l'architecture 3 tiers de notre projet qui compte plusieurs langages de programmation.

N'ayant pas obtenu de machines virtuelles pour ce projet, nous avons réalisé l'ensemble de cette architecture 3 tier en local sur une même machine. Le premier tier correspond au client, qui est sous forme de site web. Le second est le serveur, qui va contenir à la fois le code du site web, et les différents scripts pythons qui s'exécutent en temps réel (récupération de tweet, traitements des données etc.). Le 3ème et dernier tier correspond à la base de données qui contiendra toutes nos données, et qui sera en communication permanente avec le serveur. Ces 3 tiers sont donc joués sur une seule machine à l'aide d'un serveur local tel que WAMP.

Pour les différentes parties du système nous avons besoin de logiciels et d'outils pour la gestion et la relation des différentes parties entre elles dont voici le tableau illustratif ci-dessous.




	Client 	Serveur local 	Base de données 
Langages	JavaScript, HTML 5	Php, Python (3.6)	SQL (MySql)
Outils d'accès	Navigateurs	WampServer	PhpMyAdmin
Outils de développement	Editeurs de texte (Sublime Text, Notepad++)	IDLE Editeurs de texte	PhpMyAdmin

Tableau 1: Langages et outils utilisés selon la partie du système

Les langages Web s'adaptent parfaitement aux fonctionnalités de notre projet, la librairie D3.js (Data-Driven Documents) par exemple est une bibliothèque graphique utilisant les technologies SVG, JavaScript et CSS parfaite pour la construction de graphiques. Ensuite, le langage Python se prête très bien à notre projet pour le traitement de texte, la détection et le classement d'entités, etc. à réaliser sur les tweets.

Chapitre 3 : Stockage des données

Notre projet repose entièrement sur des données réelles que nous devons récupérer puis stocker. Nous allons voir dans ce chapitre quelles ont été les solutions étudiées, et quelles sont celles qui ont été mises en place.

3.1 : Analyse de la problématique

L'un des principaux enjeux du projet est de savoir comment récupérer des données en temps réel en grande quantité, et comment les conserver efficacement. Concernant la récupération de données en temps réel, utiliser le réseau social Twitter comme source de données semble être une bonne idée, car on peut y trouver de nombreux tweets postés chaque seconde, et ce dans tous les pays du monde. Il s'agit donc d'une source universelle en temps réel. Il y a des outils, que nous verrons par la suite, qui permettent de récupérer ces données sous forme de JSON.

Comment stocker ces données, fournies en tant que JSON? Il existe là encore différentes solutions, comme par exemple une base de données orientée document pour stocker les données brutes, ou bien une base de données relationnelle, qui impliquerait un petit traitement avant stockage. Quoi qu'il en soit, il faut faire en sorte de trouver une solution qui permettrait au système de ne pas être trop lourd, et qui éviterait de se retrouver avec un nombre de données mal stockées si important qu'on ne pourrait pas les exploiter.

De plus, nous pouvons nous demander quoi stocker? En effet, un tweet contient beaucoup de métadonnées, tel que le lieu du tweet, l'appareil utilisé, l'heure, le nom d'utilisateur etc., qui peuvent être des informations ayant une utilité. On peut alors se demander si utiliser deux types de stockages à la place d'un seul ne serait pas plus intéressant, une base de données pour les données brutes avec toutes les informations, et une base de données avec nos données traitées? Quel type de base de données utiliser finalement pour chacun de ces cas? Nous allons répondre à ces différentes problématiques dans ce chapitre.

3.2 : Etat de l'art : études des solutions existantes

Nous nous sommes intéressés aux différents types de stockage de données, principalement aux bases de données orientées documents (couchDB, MongoDB etc.), ainsi que les bases de données relationnelles plus classiques (MySQL etc.).

Base de données orientés documents :

Une base de données orientée document n'a pas de schéma de bases avec des contraintes sur les champs, elle est donc adaptée à un agencement simple de données qui lui apporte de la flexibilité dans sa gestion des données pour une meilleure rapidité d'exécution de ses requêtes.

En effet, il est plus rapide de requêter sur ce type de base de données par rapport à du SQL utilisé pour le relationnel.

Cependant, elle n'est pas adaptée pour un schéma de données structurées et l'écriture de requêtes complexes restes difficiles à mettre en oeuvre car ce type de base n'est pas utilisé pour l'existence de relations entre les différentes données comme la réalisation de jointures en relationnel. Notons de plus qu'il n'existe pas de langage standard sur les différentes bases de données orienté documents.

Etant donné que le stockage de données provenant de Twitter ne nécessite pas de schéma relationnel, nous pouvons nous pencher sur MongoDB qui est une de ces bases de données orientées documents, et plus précisément document JSON.

Pour le stockage de données brutes et en grosse quantité MongoDB est une base de données qui possède deux types de relations au lieu d'une : référence et imbriquée. C'est une base de données distribuée, universelle et basée sur des documents contenus dans des collections. Cela signifie qu'elle stocke les données au format de documents JSON, ce qui correspond bien au seul format de données récupérés depuis l'API twython. La base de données MongoDB est dotée d'un langage de requête riche et expressif qui permet d'effectuer un filtrage et un tri en fonction de n'importe quel champ, indépendamment du niveau d'imbrication dans un document. Les requêtes sont elles-mêmes au format JSON, et sont donc simples à paramétrer.

Base de données relationnelles:

Une base de données relationnelles permet l'exploitation du contenu structuré répartie dans différentes tables par application des opérations de l'algèbre relationnelle telle que la jointure, la sélection et la projection grâce aux différents Système de Gestion de Bases de Données Relationnelles (SGBDR). SQLite, MySQL, PostgreSQL adhèrent aux propriétés ACID pour la cohérence des transactions (atomicité, cohérence, isolation et durabilité), bien qu'ayant un fonctionnement similaire nous avons étudié les différences entre ces trois SGDBR.

PostgreSQL est plus particulièrement un système de gestion de base de données relationnelle et objet (SGBDRO), il aurait les meilleures performances des trois étudiées avec un large volume de données. Il peut stocker plus de types de données que les types simples traditionnels entiers, caractères, etc. en tant qu'utilisateur nous pouvons créer des types, des fonctions et utiliser l'héritage de type. PostgreSQL est reconnu pour ses possibilités de programmation étendues, directement dans le moteur de la base de données, via PL/pgSQL. Le traitement interne des données peut aussi être couplé à d'autres modules externes compilés dans d'autres langages. De plus Le bon support de JSON fait également de PostgreSQL une solution de base de données pour la mise à l'échelle des charges de travail NoSQL. Nous notons que ce SGDBR n'est pas directement installé avec la plateforme

WampServer et nécessite un paramétrage sur chacune de nos machines pour les requêtes php de notre serveur web local.

MySQL possède un avantage sur PostgreSQL pour les opérations simples en lecture lourde “Bulk inserts” (insertion de données en ligne (d’un csv par exemple) vers les tables, qui nécessite une grosse performance) en termes de débit et de performances. Ces performances seraient élevées en lecture, ce qui signifie qu’il est davantage orienté vers le service de données déjà en place que vers celui de mises à jour fréquentes. Nous pouvons noter que ce SGBDR est inclus avec WampServer, donc ne nécessite pas d’installation ou de paramétrage sur notre serveur web local.

Contrairement à MySQL et PostgreSQL, SQLite ne reproduit pas le schéma habituel client-serveur mais est directement intégrée aux programmes. L’intégralité de la base de données (déclarations, tables, index et données) est stockée dans un fichier indépendant de la plateforme. SQLite permet de réduire la latence des requêtes à une autre source.

Par contre, étant donné que SQLite est un SGBD basé sur des fichiers, il peut entraîner des problèmes de performances avec des ensembles de données plus volumineux en raison des limitations du système de fichiers. Nous notons que ce SGBDR n’est pas directement installé avec la plateforme WampServer.

3.3 : Solution proposée et sa mise en œuvre

Dans un premier temps, nous avons décidé de stocker tous les tweets collectés grâce à l’API de Twitter, au format JSON dans une base de données de type MongoDB, afin de conserver nos données dans un état brut, pour appliquer plusieurs traitements et trouver le plus efficace répondant à nos attentes. Ensuite, après l’exécution de toute la partie intelligence artificielle, nous stockons les résultats dans une base de données SQL plus pratique d’accès depuis notre application web.

3.3.1 : La récupération de tweets

La première problématique rencontrée a été la récupération de données. Pour cela, nous nous sommes orienté vers le réseau social twitter qui offre une API et sa librairie python de connexion, Twython, qui permet la récupération de données. Grâce à cette API, nous avons pu requêter twitter et obtenir en temps réel, des tweets sous un format JSON (voir partie [3.3.2: MongoDB](#)). Pour récupérer ces données, nous avons mis en place un script python qui permet la récupération à intervalles réguliers (toutes les 10 secondes) un certain nombre de tweets (100). La requête prend en paramètres:

- Le nombre de tweets voulus (limité à 100 par requête)
- Le type de résultat que l’on souhaite (choix entre récent et populaire)
- Le langage du tweet
- La date maximum du tweet le plus ancien

Nous avons donc décidé de récupérer, à chaque itération, les 100 tweets les plus récents en langage Français. Grâce à cela, nous pouvons nous concentrer par la suite uniquement sur les mots français, sans avoir à se soucier d'un autre langage. Nous avons décidé de traiter la langue française car il existe beaucoup de traitement sur la langue anglaise, mais très peu sur le français, cela nous a donc semblé plus intéressant.

Cependant, dans la situation actuelle de la covid19, nous n'avons pas pu avoir de machines virtuelles fournies par l'université, ce qui nous a demandé de lancer manuellement chaque jour la récupération de tweets, ne pouvant pas laisser l'une de nos machines allumées 24h/24 pour récupérer équitablement des tweets réparties sur toute une journée, tous les jours. De plus, nous avons une limitation de l'API qui à certains moments n'accepte plus les requêtes sur une période aléatoire, d'environ 2h généralement car nous ne bénéficions pas d'un accès payant prioritaire à l'utilisation. Nous avons donc au final une répartition très inégale des tweets sur chaque jour, ce qui nous a posé différents problèmes au cours des différentes étapes de traitement que nous verrons par la suite.

Une fois ces tweets récupérés, ils sont directement stockés, de manière brute, sans aucun traitement, dans une base de données mongoDB.

3.3.2 : MongoDB: Une base de données orientée documents

Notre base de données MongoDB ne contient que des collections de documents dont les données ont le même format. Les données récupérées grâce à Twython sont stockées dans la base de données MongoDB, grâce à Pymongo, sa librairie python officiel.

Voici un exemple des informations que nous conservons d'un tweet :

```
{ "_id": {
  "$oid": "5e9b431d3feab1e296cb150a"
},
  "user_id": "1062597708572188672",
  "user": "LeilaOusfane",
  "user_name": "Leila Ousfane",
  "followers_count": 289,
  "location": "Nîmes, France",
  "verified": false,
  "tweet_id": "1251558951562416136",
  "coordinates": null,
  "date": "Sat Apr 18 17:11:21 +0000 2020",
  "Urls": [
    {
      "url": "https://t.co/Zf5ENFHKsg",
      "expanded_url": "https://www.huffingtonpost.fr/entry/coronavirus-floride-plages-distanciation_fr_5e9b1b7cc5b635d25d6d4be8?ncid=other_twitter_cooo9wqtham&utm_campaign=share_twitter",
      "display_url": "huffingtonpost.fr/entry/coronavi...",
      "indices": [
        106,
        129
      ]
    }
  ]
},
```

```

    "favorite_count":0,
    "retweet_count":0,
    "geo":null,
    "in_reply_to_id":null,
    "retweeted":false,
    "iso_language_code":"fr",
    "text":"Malgré un nombre record de cas de coronavirus, la Floride rouvre ses plages et elles
    sont prises d'assaut https://t.co/Zf5ENFHKsg"}

```

Tous les tweets récupérées sont stockées dans une collection journalière. En effet, nous créons automatiquement une collection à chaque changement de jour dans la date du tweet pour partitionner nos données dans la base. Ainsi, il est plus facile de s'y retrouver, et les requêtes sont plus légères. Une requête sur une base de données MongoDB ne se termine qu'une fois que toutes les données ont été parcourus, ce qui peut être très long si beaucoup de données sont stockés dans la même collection. Les collections sont l'équivalent des tables dans une base de données relationnelle classique, elles permettent de trier les données.

3.3.3 : MySQL: Une base de données relationnelle

Pour le stockage des données traitées nous avons choisi une base de données MySQL. Les fonctionnalités basiques de ce SGBD conviennent parfaitement à nos besoins ainsi que les performances, qui ne sont pas affectées par la quantité de données peu élevée à chaque insertion et MySQL étant développé dans un souci de performances élevées en lecture notre choix s'est donc porté vers ce système de gestion facilement accessible sur les machines de chaque membre du projet.

Notre base de données MySQL contient 4 tables jointes entre elles. Ces tables nous permettent de stocker nos données de manière structurée une fois le traitement terminé.

→ Table event :

La table event contient nos events toutes périodes confondues, on attribue à chaque event un nom, il contient une popularité totale, ainsi que les features de son cluster (une liste de mots qui le définit), et les mêmes features contractées par similitudes.

→ Table tweet :

Cette table est composée de toutes les informations utiles des tweets populaires retenus pour l'affichage sur le site web comme le nom d'utilisateur de l'émetteur, le texte, sa date de publication, etc. Sa clé primaire est également une de ces informations récupérées : l'identifiant du tweet sur Twitter, qui est unique et nous assure de ne pas avoir de doublons.

→ Table popularity :

Cette table renseigne les valeurs de la popularité des events de la table event selon le jour, avec le rang du classement des plus populaires pour cette date.

→ Table prédiction :

La table prédiction permet d'enregistrer une prédiction sur les 10 events qui pourront potentiellement être les plus populaires les jours prochains. Cette prédiction renseigne la

popularité attendue et le classement de chaque event parmi les 10 plus populaires pour une période renseigné entre date_from et date_to.

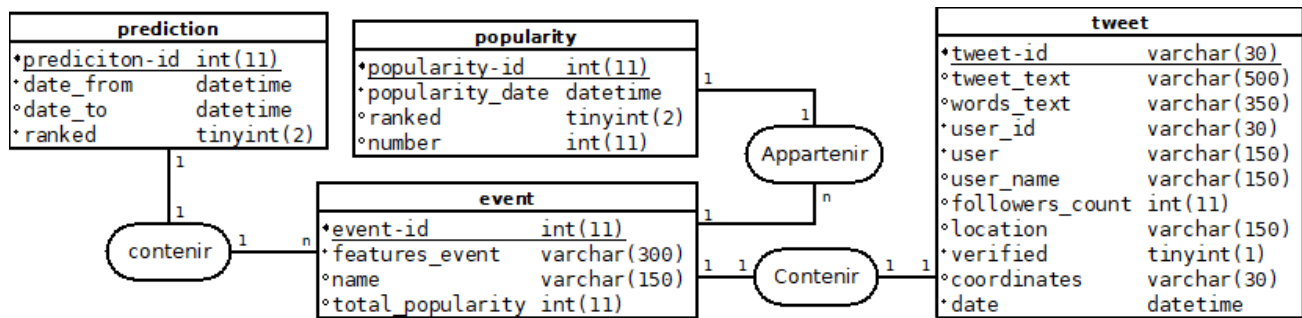


Figure 8: Schéma de la base de données relationnelle

3.4 : Certification de la solution

Depuis le 11 février 2020 nous avons récupéré plus de 32 millions de données du réseau social Twitter en temps réel, en lançant localement le script se connectant à l'API Twitter. Cependant, n'ayant pas de serveur fonctionnant sans interruption, nous n'avons pu lancer ce script que manuellement mais quasiment chaque jour.

Le graphique ci-dessous renseigne le nombre de données récupérées par jour.

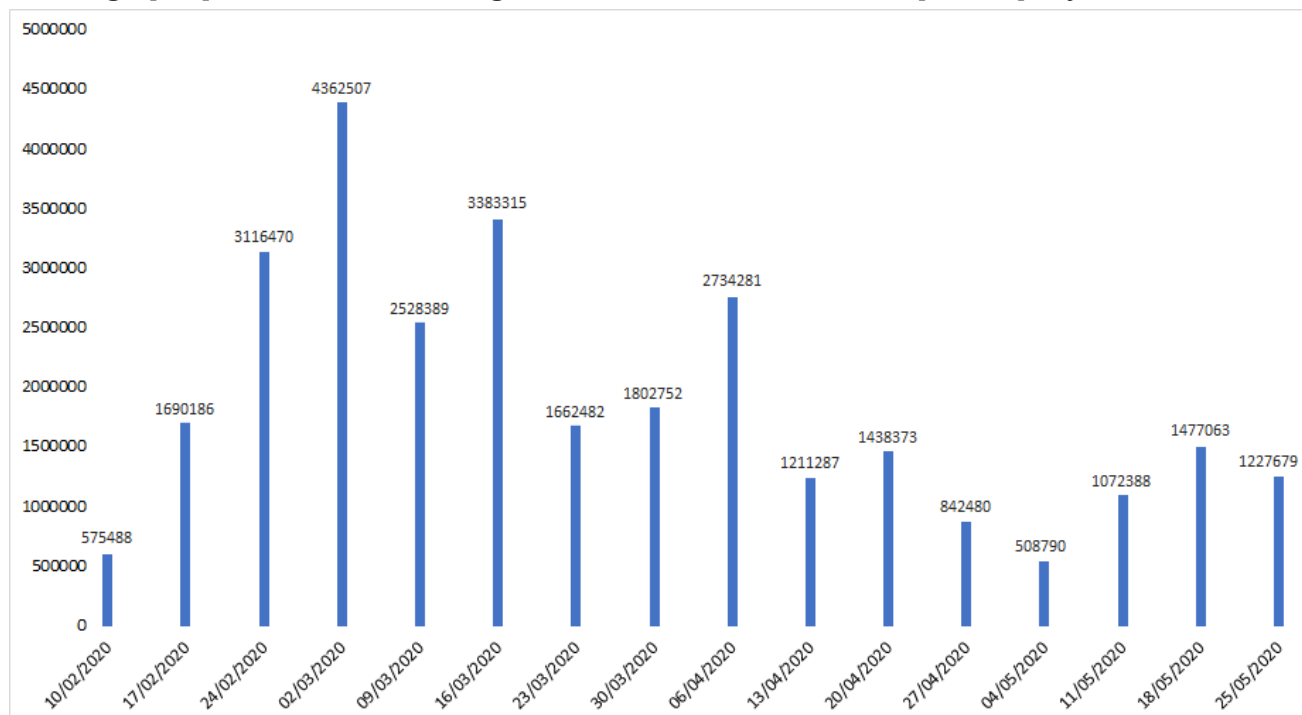


Figure 9: Graphique du nombre de tweet récupéré par semaine

Comme on peut le voir sur la figure 9, la répartition des tweets est très inégale, mais elle atteint généralement des pics à plus d'1 million de tweets (Jusqu'à 4,3 millions la semaine du 02/03/2020) récupéré par semaine. Notre solution de récupération de tweet est donc très

efficace et permet d'exporter un grand nombre de tweets. De plus, tous ces tweets ont directement été stockés en base de données (MongoDB dans un premier temps comme nous l'avons vu précédemment). **Chacun** des tweets récupérés a été stocké, sans jamais poser de problèmes lors de l'insertion. La solution d'une base de données orientée document s'est donc avérée un très bon choix permettant la gestion d'un grand nombre de données sans avoir de problèmes de performances.

Chapitre 4 : Traitement de texte brute

Dans ce chapitre, nous allons voir les premières étapes de traitement par lesquelles les tweets sont passés. Nous verrons dans un premier temps l'application du NLP, ce qui nous mènera à parler de la vectorisation de ces données.

4.1 : Analyse de la problématique

Une fois nos données brutes stockées dans une base de données MongoDB, il va falloir les étudier afin d'en tirer diverses informations qui sont nécessaires pour le traitement. Sans ce traitement de texte, il sera impossible de déterminer si un tweet parle d'un événement ou non. Chaque tweet publié sur Twitter par un utilisateur va être différent, tant sur le fond que sur la forme. Nous allons donc obtenir un texte brute, qui sera dépendant de chaque utilisateur.

Chaque utilisateur à sa façon d'écrire des messages. Certains ne mettent jamais de majuscules, d'autres en mettent uniquement en début de phrase, d'autres peuvent même mettre des mots entiers en majuscules. Il en est de même pour les symboles, tels que les smileys, les emojis ou les hashtags. Deux messages ayant le même fond n'auront pas toujours la même forme, même s'ils peuvent être très similaires. Par exemple on pourrait obtenir les messages suivants:

- "Je n'ai pas le coronavirus, je suis très content :D"
- "je n ai pas le #coronavirus je suis tres content"

Ces deux messages sont très similaires, disent exactement la même chose mais contiennent quand même des différences. Ces différences sont des détails d'un point de vue humain, mais du point de vue d'une machine la phrase n'est pas la même. En effet, elle n'a aucune idée du rôle d'une virgule, d'un hashtag ou d'un mot dans une phrase, il est donc impossible d'en tirer une information telle quelle.

Il faut donc transformer ce texte en une donnée que la machine pourra utiliser et comprendre afin de pouvoir les clusteriser par la suite. Sans cette étape de traitement, d'un point de vue machine à la moindre différence d'un symbole les deux phrases seraient considérées comme différentes, et nous n'obtiendrons pas le résultat souhaité.

L'objectif de cette étape est donc de faire en sorte que le texte qui a été écrit par l'utilisateur ne soit plus traité uniquement par sa forme, mais principalement par le fond.

4.2 : Etat de l'art du traitement de texte

Le traitement de texte est une technologie d'IA qui est en permanence en cours d'évolution. Cette notion a commencé à apparaître dans les années 1950, où la traduction automatique était devenu un enjeu principal en période de guerre froide. Alain Turing fera

même, dans un article [4] paru en 1950, ce qu'on appelle maintenant le "Test de Turing", qui consiste à calculer le degré d'intelligence d'une machine par sa capacité à se faire passer pour un humain. Ce test consiste en un échange, à l'écrit, entre un humain et une machine, et le sujet humain devait déterminer si son interlocuteur était humain ou non. S'il ne parvient pas à trouver la bonne réponse, alors la machine a "gagné".

Turing avait prédit qu'un jour, la machine pourra imiter à la perfection le langage humain. Pour le moment, cette prédiction n'est pas réalisée, mais les différents progrès dans le domaine ont permis de s'en rapprocher. Chaque année, un concours est réalisé aux états unis, le loebner-prize, qui a pour but de récompenser les meilleurs programmes informatiques qui satisfassent au mieux le test de Turing. La récompense est de 2 000\$ chaque année, mais il existe des récompenses spéciales qui n'ont pas encore été atteintes:

- Une récompense de 25 000\$ attend le premier programme qui arrivera à convaincre les juges que l'humain est l'ordinateur dans les conditions du test.
- Une récompense de 100 000\$ pour le premier programme qui ne pourrait pas être distingué d'un humain, et qui comprend le déchiffrement et la compréhension des entrées textuelles, visuelles et auditives.

Le concours s'arrêtera lorsque quelqu'un arrivera à avoir la récompense de 100 000\$, car on aura atteint le niveau optimal du traitement de texte, et la prédiction de Turing sera réalisée, le concours n'aura donc plus de sens.

En attendant, même si le test de Turing n'a pas trouvé de programme à la hauteur, le traitement de texte est développé à un niveau largement suffisant pour être utilisé dans nos vies de tous les jours. En effet, de nombreux outils de tous les jours utilisent différents types de traitement de texte, dans le but de nous simplifier la vie.

Parmi ces outils, on peut citer les traducteurs automatiques, les fonctionnalités de correction automatique de texte, les assistants vocaux (qu'il s'agisse de fonctionnalités dans un produit tel que Siri pour Apple, ou bien même d'un produit à part entière avec Google home). Ces outils sont de plus en plus efficaces, et répondent de mieux en mieux à nos besoins, à tel point qu'ils font maintenant partie de notre quotidien. Chacun de nos messages postés sur un réseau social est traité automatiquement afin d'éviter des messages non acceptables (racisme, incitation à la haine etc.), le traitement de texte se trouve partout autour de nous dans les différents systèmes informatiques sans forcément qu'on s'en rende compte, mais il s'avère nécessaire.

Certaines entreprises utilisent même le traitement de texte pour se faire de l'argent, en vendant par exemple des données personnelles, ou des informations tirées de formulaires. Chaque donnée est traitée automatiquement et revendu à d'autres entreprises, pour faire des publicités ciblées par exemple.

Comme nous l'avons vu, le traitement automatique du texte est un enjeu important dans notre société, et il en existe de nombreuses utilisations, qui ont chacune un objectif précis pour satisfaire les besoins d'un utilisateur. Dans le cas de notre projet, le but est de

“comprendre” les tweets envoyées par les utilisateurs pour déterminer leurs sujets de conversation.

4.3 : Solutions apportées et mises en œuvre

Dans le cadre de notre projet, nous n'avons qu'une partie du sujet à explorer, il concerne le traitement de texte brute. Le but de notre traitement de texte est de partir d'une phrase, compréhensible par un humain, à une représentation mathématique, “compréhensible” par la machine, afin de pouvoir par la suite la traiter. Pour cela, nous passons par deux étapes qui sont le NLP et la vectorisation.



Figure 10: Étapes de traitement de texte

4.3.1 : Natural Language Processing (NLP)

Les tweets récupérés sont bruts, et contiennent donc du texte qui est initialement peu exploitable. Le but du NLP va être de rendre ce texte exploitable pour pouvoir par la suite le passer sous forme d'un vecteur. Pour cela il faut normaliser le texte afin de mettre en commun ce qui est possible de mettre en commun dans le texte. Voici les différentes étapes que va connaître le texte en entrée durant le NLP:



Figure 11 : Les étapes du NLP

Nous allons maintenant expliciter chacune de ces étapes avec une phrase qui nous servira d'exemple : **"Il faut rester confiné à cause du méchant coronavirus. :("**

Lors de la première étape, on retire les majuscules du texte en entrée, notre texte devient donc : **"il faut rester confiné à cause du méchant coronavirus. :("**. Cette étape permet de rendre identiques deux mots dans le cas où sur un tweet l'un à une majuscule et sur un autre non (Pour diverses raisons: Premier mot de la phrase, erreur de frappe etc.).

La seconde étape permet de retirer ce qu'on appelle les "Stopwords" du texte. Les stop words sont un ensemble de mot considéré comme inutile car trop commun dans la langue française pour pouvoir apporter une information intéressante, par exemple tous les pronoms, ou bien des conjonctions de coordination par exemple. (ex: Je, tu, mais, donc etc.). Le résultat après cette étape est donc : **"faut rester confiné cause méchant coronavirus. :("**

La troisième étape consiste à donner à un mot sa forme "neutre canonique", et à un verbe sa forme à l'infinitif. Ceci permet de gérer par exemple les mots au pluriel ou les verbes conjugués. On obtient après cette étape : **"falloir rester confiner cause méchant coronavirus. :("**

L'étape suivante a pour but de retirer les accents, afin de pouvoir associer deux mots identiques, mais l'un écrit avec un accent et l'autre sans. Le fait de retirer l'accent permet d'uniformiser le mot, et de ne pas dépendre de l'écriture donnée par l'utilisateur. On a alors : **"falloir rester confiner cause mechant coronavirus. :("**

La dernière étape retire les caractères spéciaux du texte, afin de n'avoir plus que des caractères alphanumériques. On supprime alors les "#", les signes de ponctuation et les smileys par exemple, qui n'apportent pas d'information au traitement. Le résultat final de ces traitements est donc : **"falloir rester confiner cause mechant coronavirus".**

Ces différentes étapes permettent d'unifier les mots individuellement au sein de la phrase, ce qui nous permet par la suite de pouvoir vectoriser le texte du tweet.

4.3.2 : Vectorisation des données

L'une des problématiques de notre projet est de regrouper des données similaires dans un même groupe (clustering). Les algorithmes qui répondent à cette problématique, les algorithmes de clusterings, traitent des données numériques, des vecteurs et non pas directement des mots. Nous devons donc passer d'une phrase qui a subi le traitement NLP vu précédemment, à un vecteur contenant des valeurs. Mais comment arriver d'une phrase à un vecteur ?

Tout d'abord pour vectoriser une phrase, nous devons établir des features¹ qui forment un vocabulaire, qui représente le nombre de dimensions du vecteur. Chaque mot du vocabulaire

¹ Caractéristiques

est associé à une dimension. Un premier problème survient, comment considérer des termes comme : “Prince Charles” comme étant un seul token, pour que NLP ne le sépare pas en deux tokens. La solution est la prise en compte des N-gram. Un N-gram est une sous-séquence de N éléments construite à partir d’une séquence de mots donnée. La prise en compte des N-gram se fait par modélisations probabilistes qui prédisent le mot suivant dans la phrase en utilisant les N-1 mots précédents. Par exemple pour un 2-Gram (bi-gramme), on regarde pour un mot le mot qui précède le plus probable. Pour le 2-Gram : “Prince Charles”, on calcule :

$$P("Charles"|"Prince") = \frac{P("Prince charles")}{P("Prince")} = a, a \text{ est la fréquence de "Prince Charles" sur la fréquence de "prince"}. \text{ Et par exemple,}$$

$$P("Charles"|"Princesse") = \frac{P("Princesse charles")}{P("Princesse")} = b$$

Ici, on suppose, la probabilité $a > b$ donc si le mot “Prince” apparaît alors le mot qui le succédera sera “Charles”. En autorisant les N-Gram dans nos features, nous permettons à des mots qui vont généralement apparaître ensemble, comme “Prince charles” ou bien “Stade de France”.

Supposons maintenant un jeu de données D comportant n tweets. Le but est de trouver le vocabulaire à partir de D , pour notre problématique de reconnaissance d’événements, nous devons avoir les termes les plus fréquents en tant que feature, pour cela on calcule la TF (term-frequency) de chaque mot. Ce processus est appelé “Bag of word”. L’extraction de features est un processus qui se fait en plusieurs étapes, tel que :

1. Traiter chaque tweet avec le traitement NLP pour les nettoyer, puis les tokenizer.
2. Compter les tokens, pour voir les occurrences de chaque mot dans chaque tweet de D .
3. Calculer le TF pour chaque mot et prendre f features avec les meilleures valeurs.

La première étape a été décrite dans la partie précédente, comme on l’a vu elle permet de nettoyer les tweets, pour éviter d’avoir des mauvaises features et scinder les données pour la seconde étape. La seconde étape est simple, elle permet d’avoir le compte pour chaque mot de chaque tweet, ainsi nous savons quel mot apparaît combien de fois dans chaque tweet et son nombre d’apparition total dans D . Pour cela il nous suffit d’itérer dans D et d’enregistrer pour chaque tweet les mots associés et leur nombre sous forme d’une matrice Nombre de tweets x mots. On peut donc passer à la troisième étape, le calcul TF de chaque mot. Le TF ce calcul tel que :

$$TF(t, d) = \frac{n_{r,d}}{\sum_k n_{k,d}}$$

Formule 1: Calcul du TF

avec t le terme, $n_{r,d}$ le nombre d’apparition du terme dans le tweets d et $\sum_k n_{k,d}$ le nombre

total de termes dans les tweets d . Le TF de chaque terme est alors enregistré. Chaque token est alors considéré comme étant une feature, en prenant en compte les N-Gram. Maintenant

que nous avons nos features, nous pouvons vectoriser nos tweets. Mais comment attribuer une valeur pour chaque dimension d'un vecteur. Il existe une technique qui permet d'ajouter un poids si le mot est présent qui permet de voir la fréquence d'un mot et son unicité : le TF-IDF (term-frequency inverse document-frequency). Pour comprendre le TF-IDF, il faut voir comprendre ce qu'est le IDF (inverse document-frequency), la fréquence inverse de document. L'IDF est une mesure pour évaluer l'importance d'un mot, elle donne une valeur plus grande aux mots qui sont le moins fréquent, on la calcule tel que :

$$IDF(t) = \log \frac{n}{df(t)} + 1$$

Formule 2: Calcul du IDF

Avec $df(t)$ le nombre de tweets qui contiennent le terme t , et n le nombre de tweets dans D . On peut donc calculer maintenant le TF-IDF:

$$TF - IDF(t, d) = TF(t, d) \times IDF(t)$$

Formule 3: Calcul du TF-IDF

Un mot très fréquent dans un tweets mais qui sera moins fréquents dans les autres tweets de D , aura une valeur de TF-IDF. Ainsi pour tous les tweets de D , on calcule pour un tweet le TF-IDF de chaque token puis on normalise, tel que :

$$v_{Normalisé} = \frac{v}{\sqrt{v_1^2 + \dots + v_n^2}}$$

Formule 4: Normalisation du TF-IDF

Exemple de vectorisation :

Prenons comme exemple 3 phrases :

- Phrase 1 : "Le Prince Charles a le coronavirus !"
- Phrase 2 : "Le Prince Charles a un cheval blanc."
- Phrase 3 : "Le coronavirus."

Tout d'abord nettoyons ces phrases (suppressions des stopwords, de la ponctuation) et tokenisons les, tout en prenant en compte les 2-Gram, on obtient alors :

- Phrase 1 : [prince charles],[coronavirus]
- Phrase 2 : [prince charles],[cheval], [blanc]
- Phrase 3 : [coronavirus]

On calcul le TF pour chaque mot de chaque document , résultat représenté dans le tableau ci dessous :

	Prince charles	coronavirus	cheval	blanc
Phrase 1	0.5	0.5	0	0
Phrase 2	0.33	0	0.33	0.33
Phrase 3	0	1	0	0

Tableau 2: résultats TF pour nos 3 phrases d'exemple.

Maintenant calculons le IDF de chaque terme, tel que précédemment détaillé. Les résultats sont représentés dans le tableau ci-dessous :

	IDF(t)
Prince Charles	1.18
coronavirus	1.18
cheval	1.48
blanc	1.48

Tableau 3: résultats IDF pour nos 4 termes.

On peut donc dès maintenant calculer nos vecteurs avec le TF-IDF d'abord non normalisé puis normalisé. Les résultats sont représentés dans le tableau ci-dessous.

Features	Prince Charles	coronavirus	Cheval	Blanc
Vecteur phrase 1	0.59	0.59	0	0
Vecteur phrase 2	0.39	0	0.49	0.49
Vecteur phrase 3	0	1.18	0	0

Tableau 4: Résultat final de notre exemple de vectorisation non normalisé.

On normalise, tel que prince Charles pour la phrase 1 :

$$v = \frac{0.59}{\sqrt{0.59^2 + 0.59^2}} = 0.71$$

On fait ceci pour chaque vecteur, on obtient donc :

Features	Prince Charles	coronavirus	Cheval	Blanc
Vecteur phrase 1	0.71	0.71	0	0
Vecteur phrase 2	0.49	0	0.61	0.61
Vecteur phrase 3	0	1	0	0

Tableau 5: Résultat final de notre exemple de vectorisation normalisé.

Ainsi comme on peut le voir sur le tableau 5, “Cheval” et “Blanc” sont des mots plus uniques que “Prince Charles” car ils ne sont pas présents dans les 2 autres phrases de notre exemple.

Au final avec le “Bag of word”, N-Gram et TF-IDF, notre vectorisation de données textuelles permet de voir pour chaque tweet l'importance et la présence d'une feature, ce qui permettra un clustering optimal

4.4 : Tests et certifications de la solution

Pour tester notre solution, nous avons commencé par des tests individuels: Si nous n'avions pas la certitude que notre traitement NLP était efficace, nous n'aurions pas pu dire si la vectorisation l'était aussi.

Nous avons commencé par tester le NLP avec une liste de mots prédéfinis afin de voir ce qu'il se passait pour chacun d'entre eux. Cette liste de mots contenait les mots suivants:

- “coronavirus”, “pour”, “Seisme”, “Tremblement de terre”, “je suis”, “méchant”, “:D”, “Emmanuel MACRON”, “l'art”, “différents”.

Après avoir testé notre algorithme avec chacun de ces mots, 100% de nos attentes ont été réalisés. En effet, nous avons obtenu les résultats suivants:

- “coronavirus” => “coronavirus” : Rien ne change
- “Pour” => “” : Considéré comme un stop word
- “Seisme” => “seisme” : Perte de la majuscule
- “tremblement terre” : Perte de la majuscule + suppression du “de” considéré comme un stop word
- “je suis” => “etre” : Suppression du stop word “je”, et mise à l'infinitif du verbe “être” sans accent
- “méchant” => “mechant” : Plus d'accent
- “:D” => “” : Plus de symboles
- “Emmanuel MACRON” => “emmanuel macron” : Plus de majuscules, et le nom propre n'a pas été modifié

- “l’art” => “art” : Apostrophe retirée, et le ‘l’ mis sous sa forme “le” est supprimé car un stop word
- “différents” => “different” : Plus d’accent, et mot au singulier.

La seconde étape a été de tester cet algorithme sur des phrases entières, avec de la ponctuation. Pour cela, nous avons commencé par tester des phrases formées à l’aide des mots testés ci-dessus. Par exemple, nous avons formé les phrases “Je suis l’art différent !”, ou encore “Coronavirus méchant tremblement de terre”. Ces phrase n’a pas forcément de sens et peuvent comporter des erreurs grammaticales, mais elle nous permettent de bien tester l’application du NLP, sachant que les tweets que nous avons stockés comportent eux aussi des erreurs.

Le but de cette étape est de voir si appliquer le NLP à une suite de mots diffère de l’appliquer à un seul mot. Nous avons obtenu des résultats concluants car lors de ces tests, nous avons obtenu les mêmes résultats que pour les tests individuels. Ainsi, “Je suis l’art différent !” devient “etre art different”, qui peut être décomposé en “etre”, “art”, et “different” qui correspondent aux résultats individuels obtenus avec “Je suis”, “l’art” et “différent”.

Cependant, pour approfondir les tests nous avons utilisé de vrais tweets afin de nous assurer qu’il n’y avait pas de problèmes. Nous en avons rencontré un majeur, qui était que le mot “Coronavirus”, en fonction de sa position dans la phrase, pouvait devenir “coronavirus” ou “coronaviru”. Dans certains cas, il était considéré comme un nom au pluriel et le “s” était donc supprimé, mais pas dans tous les cas. Nous avons donc dû revoir l’algorithme que nous utilisions pour la lemmatisation, et nous avons pu corriger le problème en changeant la bibliothèque utilisée par cette méthode.

Ensuite, pour tester les calculs du TF-IDF, nous avons utilisé les phrases présentées en exemple dans la [sous section 4.3.2](#). Les calculs qui y sont présentés sont des calculs théoriques fait avant l’exécution du programme. Après avoir lancé l’algorithme, nous avons aperçu que les calculs théoriques se sont avéré exacts, et que donc l’algorithme fonctionnait bien comme prévu. De plus, lors de nos tests des étapes suivantes de traitement, nous avons fait des affichages consoles permettant de visualiser les résultats obtenus sur cette étape afin de pouvoir vérifier à chaque exécution que tout se passait bien, et nous n’avons à aucun moment relevé d’erreur.

Chapitre 5 : Clustering des données

Pour regrouper des tweets entre eux afin de trouver des events, nous avons dû mettre en place un algorithme de clustering. Dans ce chapitre, nous verrons sur quels algorithmes nous nous sommes penchés pour ce projet, quels sont leurs avantages et leurs défauts et comment nous les avons départagé.

5.1 : Analyse de la problématique

Comme nous l'avons vu dans la section 2.4, notre projet relève la problématique suivante : Comment déterminer si un tweet concerne un event ? C'est dans le domaine de l'intelligence artificielle, et plus précisément celui de "l'apprentissage automatique" (Machine Learning), que nous allons trouver une réponse. L'apprentissage automatique a pour but de classer des données qui sont labellisées (apprentissage supervisé) ou non (apprentissage non-supervisé). Pour faire un algorithme d'apprentissage automatique supervisé, il nous faudrait, une base massive de tweets (plus de 10 000), où l'on dit si chaque tweet est un événement, pour que l'algorithme s'entraîne dessus. Cette solution serait compliquée à mettre en place, au vu de la taille de la base d'entraînement à fournir. Un algorithme d'apprentissage automatique non supervisé permet aussi de classer les données mais sans avoir besoin de données labellisées. En alimentant l'algorithme de données, il va les regrouper en N groupes (clusters). Cette solution semble être optimal pour nous, car nous récupérerons les données (tweets) en temps réel.

Il y a en général deux de types d'algorithmes de clustering : à partitionnement et à hiérarchisation.

Les algorithmes de partitionnement construisent une portion d'une base de données de plusieurs échantillons, que nous appelons samples, dans un ensemble de k clusters (k est souvent paramètre de l'algorithme). Ce sont donc des algorithmes supervisés qui nécessitent une profonde connaissance du contenu de la base de données. Par exemple k-means est un algorithme connu faisant partie de cette catégorie. Les algorithmes de partitionnement fonctionnent généralement en deux étapes : déterminer le nombre de k clusters en minimisant une fonction objective, puis assigner chaque sample avec son cluster. Mais ces algorithmes de clustering supervisé nécessitent un paramètre que nous ne connaissons pas à l'avance, le nombre de clusters.

Les algorithmes hiérarchiques créent une décomposition hiérarchique de la base de données. Cette décomposition est représentée sous forme d'arbre (dendrogramme), un arbre, en partant de la racine (méthode agglomérative) ou par les feuilles (méthode divisive), qui sépare la base de données en sous-ensembles jusqu'à ce que chaque sous ensemble contienne qu'un seul sample. Chaque noeud représente un cluster. Contrairement aux algorithmes de partitionnement, les algorithmes hiérarchiques n'ont pas besoin de k comme entrée, mais nécessite un paramètre d'arrêt. Le principale problème de ces algorithmes est de trouver ce

paramètre d'arrêt qui n'est pas trivial. De plus les algorithmes hiérarchiques de clustering gèrent mal les grands nombres de données, à cause de la taille de l'arbre généré. Nous faisons alors le choix de deux algorithmes non-supervisés qui ne nécessitent pas le nombre de clusters, et qui sont performants avec un grand nombre de données : Mean Shift et DBSCAN [5]. Le but sera donc de les comparer pour savoir lequel est le plus performant pour notre projet.

5.2 : Etat de l'art : études des solutions existantes

Nous allons voir dans cette partie les différentes solutions existantes pour le clustering de données, ainsi le Silhouette score [6] qui nous servira à les comparer.

Tout d'abord, il existe de nombreux algorithmes de clustering, qu'on peut classer dans deux grosses classes: les algorithmes supervisés et non supervisés. Comme expliqué précédemment, dans le cadre de notre projet nous avons besoin d'algorithmes non supervisés qui détermineront si oui ou non un tweet appartient à un event. Dans cette catégorie d'algorithmes il existe encore une fois de nombreux algorithmes. Après avoir étudié certains d'entre eux, nous avons retenu deux algorithmes qui semblaient se prêter au mieux au projet. Il s'agit de Mean Shift et DBScan.

Mean Shift est un algorithme non supervisé basé sur l'estimation de densités des données (Sur le principe de KDE, Kernel Density Estimation). Le fonctionnement de cet algorithme est assez simple:

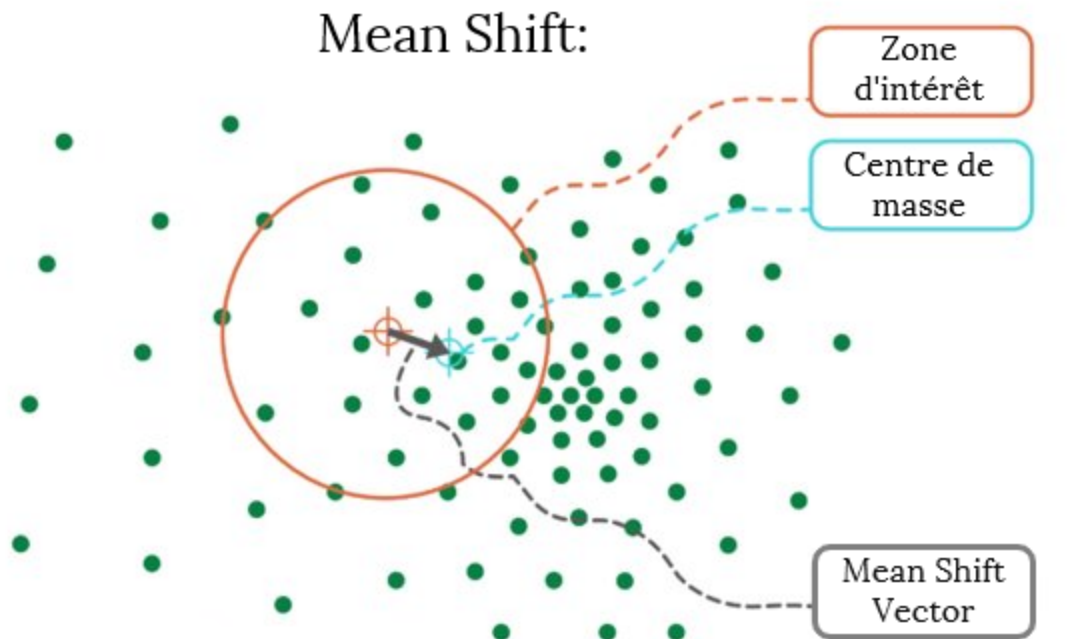


Figure 12: Illustration de Mean Shift

Pour chaque donnée, on regarde une zone d'intérêt (paramétrable) et on y calcule le centre de masse. On déplace la donnée sur ce centre de masse, en suivant ce qu'on appelle le Mean Shift Vector, et on répète l'opération jusqu'à ce que la donnée ne bouge plus, c'est-à-dire que son Mean Shift Vector devienne nul. La donnée est alors au centre de masse le plus proche de sa position initiale, et on l'associe donc à ce cluster.

On peut calculer le centre de masse à l'aide de la formule suivante:

$$m(x) = \frac{\sum_{x_i \in N(x)} K(x_i - x) x_i}{\sum_{x_i \in N(x)} K(x_i - x)}$$

Formule 5: Formule de Mean Shift

On a $m(x)$ le centre de masse de la fenêtre d'intérêt, $N(x)$ l'ensemble des données comprises dans la fenêtre d'intérêt, et $K(x)$ la fonction de densité utilisée (généralement gaussienne).

DBSCAN [1] (Density-based Spatial Clustering of Applications with Noise) est un algorithme de clustering non supervisé, qui permet de séparer dans notre jeu de données, sous forme d'espace vectoriel, les zones à hautes densités et les zones à densités plus faibles. Les samples qui appartiennent à une zone à hautes densités sont appelés les "core-samples". Les données "non core-samples" sont considérées soit comme étant du bruit, soit comme des "borders-samples"², ce sont des données en zones de densité faible.

Comme MeanShift, DBSCAN permet de regrouper des samples dans des clusters sans pour autant lui avoir donné le nombre exact de cluster au préalable, cependant l'algorithme possède deux paramètres qui ont un grand impact sur le résultat. Ses deux paramètres sont :

- " ϵ ", la distance maximum entre deux samples pour qu'ils soient considérés comme étant dans le même voisinage.
- "min samples", le minimum de samples qu'il faut considérer pour qu'un point soit un "core sample" et donc qu'il appartient à un cluster.

Les samples dans le rayon ϵ d'un autre sample font partie de son voisinage, on appelle cela l' **ϵ -voisinage** qui est défini comme suit.

Définition : [1], l' **ϵ -voisinage** d'un point p , noté N_ϵ est défini par :

$$N_\epsilon(p) = \{q \in D \mid dist(p, q) \leq \epsilon\}$$

Ainsi un "core sample" est défini si le nombre minimum de "core samples" autour de lui est à une distance (euclidienne) inférieure ou égale à ϵ , ce qui détermine son voisinage. Voyons maintenant avec le schéma ci-dessous le fonctionnement de DBSCAN.

² Échantillons qui appartiennent au bords d'un cluster.

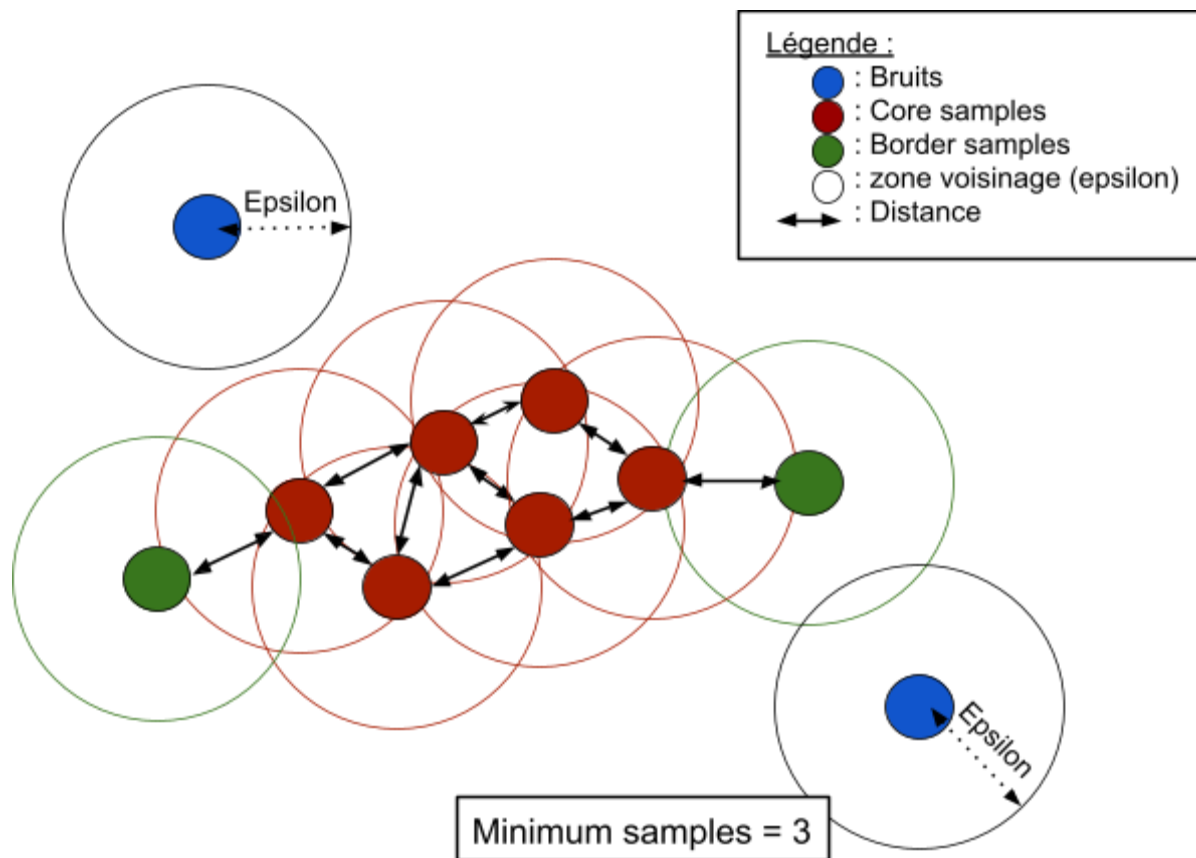


Figure 13: Schéma fonctionnement DBSCAN

Comme on peut le voir sur le schéma, tout point a une zone de voisinage défini par la valeur de ϵ . Si un sample a un nombre minimum d'autres samples dans son voisinage (3 dans l'exemple du schéma), alors il est considéré comme un "core-sample" (point rouge). Au départ, l'algorithme prend un point au hasard et va itérer sur le procédé suivant :

1. Analyser le voisinage du point.
2. Définir si c'est un "core-samples", en étudiant son voisinage.
3. Recommencer pour chaque autre sample
4. Création d'un cluster lorsque l'algorithme tombe sur un sample qui appartient au voisinage mais qui n'a pas assez de voisins (minimum de samples). Ce sample sera donc défini comme étant "border-sample" (points verts).
5. Arrêt lorsque tous les samples ont été visités.

On dit alors qu'un "border-sample" n'est pas **directement densité-atteignable** et plus globalement pas **densité-atteignable**, qui sont définis comme suit.

Définition : Un point p est **directement densité-atteignable** par un point q , si :

1. $p \in N_{\epsilon}(q)$, son voisinage.
2. $|N_{\epsilon}(q)| \geq \text{Min samples}$

Définition : Un point p est **densité-atteignable** d'un point q , si :

$\exists p_1, \dots, p_n, p_1 = q, p_n = p \mid p_{i+1} \text{ est directement densité atteignable à partir de } p_i$.

On considère un “border-sample” comme un “non-core-sample”, mais pas comme du bruit. Les bruits³ (points bleus) sont les samples qui ont un nombre de voisins inférieurs à “min samples”, qui ont une distance vers les autres samples supérieurs à ϵ . A la différence du bruit, les “border-samples” possèdent une propriété qui les rattache aux “core-samples”, la densité-connectivité, qui est définie comme suit.

Définition : Une point p est densité-connecté à un point q , s'ils sont tous deux densité-atteignables à partir d'un point o .

On peut donc maintenant définir ce qu'est un cluster pour DBSCAN :

Définition : Un cluster C est un sous ensemble non vide de la base de données qui satisfait les conditions suivantes avec ϵ et min samples :

1. $\forall p, q: \text{si } p \in C \text{ et } q \text{ est densité atteignable par } p, \text{ alors } q \in C.$
2. $\forall p, q: p \text{ est densité connecté avec } q, \text{ alors } q \in C.$

Les points rouges et verts forment donc un cluster et les points bleus seront labélisés comme étant du bruit. Le paramètre “min samples” contrôle principalement le degré de tolérance de l'algorithme au bruit, le paramètre ϵ est crucial pour obtenir un résultat qui correspond à notre problème. S'il est choisi trop petit, la plupart des données ne seront pas du tout regroupées (et. Au contraire si ϵ est choisi trop grand, les clusters proches fusionneront en un seul et unique cluster. Nous devons donc déterminer la meilleure combinaison de paramètres, afin d'obtenir un résultat qui satisfait notre projet.

DBSCAN et MEANSHIFT étant maintenant décrit, on voit en eux deux potentiels candidats pour répondre à la problématique de clustering de notre projet. Pour résumer nous avons deux algorithmes non supervisés : Mean Shift basés sur le calcul des centroïdes avec un seul paramètre, le bandwidth. DBSCAN basé sur le calcul de densité avec comme paramètres ϵ , min samples (cf. tableau ci-dessous).

Algorithme	Mean Shift	DBSCAN
Concept	basé sur les centres	basé sur la densité
Paramètre(s)	Bandwidth	ϵ , min samples

Tableau 6: descriptif des deux algorithmes

³ Le bruit est labellisé avec le numéro de cluster “-1” par DBSCAN.

Maintenant que nous avons vu en détail le fonctionnement de Mean Shift et DBSCAN. Nous allons voir avec quelle métrique nous choisis pour les départager, le Silhouette Score [6].

L'une des problématiques que nous rencontrons lorsque l'on traite des algorithmes d'apprentissage non supervisés, est que le résultat est au départ "imprévisible", ce qui implique que l'évaluation du résultat du clustering, la validité du clustering est non triviale. Comme décrit dans [7], on dénote 3 types de validation du cluster :

1. **Validation externe** : qui consiste à comparer la similarité (homogénéité, complétude...) entre le résultat obtenu et un résultat déjà défini, cela implique donc de connaître le résultat a priori.
2. **Validation interne** : consiste à utiliser les informations internes du résultat de l'algorithme de clustering pour évaluer la qualité du clustering. En effet un bon cluster, est un cluster qui possède des objets qui sont le plus possible similaires entre eux, et avoir une distinction entre les différents clusters. Pour déterminer cela, on calcule deux métriques :
 - a. **Compacité** : mesure la distance moyenne entre les objets d'un cluster, on parle de distance intra-cluster.
 - b. **Séparabilité** : mesure la distance moyenne entre les clusters de notre espace vectoriel, on parle de distance inter-cluster.
3. **Validation relative** : évaluation du résultat du clustering par comparaison aux autres résultats du clustering résultant du même algorithme mais avec des paramètres différents. Méthode utilisé pour trouver les paramètres optimaux.

Comme nous l'avons Mean shift et DBSCAN étant des algorithmes de clusterings non supervisé, ce qui fait que nous n'avons pas de résultat disponible pour comparer avec nos résultats obtenus. Le calcul des métriques issues de la validation externes n'est donc pas une bonne méthode pour calculer la validité du clustering. D'autre part, le calcul des métriques issues de la validation internes au clustering, semble être un choix judicieux pour l'évaluation des algorithmes de clustering non supervisés, étant donné qu'il ne nécessite pas de résultat prédéfini. De plus la méthode de validation relative, nous permettra d'optimiser les paramètres de Mean Shift et DBSCAN. Nous devons donc choisir un moyen de validation interne de notre résultat de clustering, qui soit simple à calculer et efficace pour nous démontrer la validité du clustering.

Dans [6] P.J Rousseuw s'est posé la question suivante : Quel objet apparaît comme étant bien classifiés, lesquels sont mal classifié et lesquels tendent entre deux clusters? Il propose alors une métrique de validation interne, le Silhouette Score. Le Silhouette Score est un score qui mesure la qualité de la clusterisation d'une donnée, ainsi que la distance (par défaut Euclidienne) moyenne intra-cluster et la distance moyenne inter-cluster. Le Silhouette Score est définie tel que :

Définition : $\forall i \in A$ et $A \subset D$, avec i un objet du cluster A inclus dans l'espace vectoriel D de n dimensions, on a :

$$a(i) = \text{dissimilarité moyenne entre } i \text{ et tout points de } A$$

Au final, $a(i)$ représente la distance moyenne inter-clusters. Mais a n'est pas suffisant pour calculer le silhouette score.

Définition : $\forall C \subset D$ et $C \neq A$ avec A et C cluster inclus dans D , on a :

$$d(i, C) = \text{dissimilarité moyenne entre } i \text{ et tous les objets de } C$$

et,

$$b(i) = \text{minimum } d(i, C)$$

$d(i, C)$ est donc la distance moyenne inter-cluster, et $b(i)$ correspond à la distance de i avec son plus proche voisin appartenant à un cluster auquel il n'appartient pas. Finalement nous avons tous les composants pour calculer le silhouette score s , que l'on définit tel que :

$$\text{Définition : } S(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \text{ avec } -1 \leq s(i) \leq 1$$

$$\text{Définition : } S = \frac{1}{n} \sum_{i=1}^n S(i), \text{ avec } -1 \leq s \leq 1$$

Un score $S(i)$ de 1 reflète que les données sont très bien regroupées. Une valeur proche de 0 montre qu'un objet tend entre deux clusters. Le score peut aller jusqu'à -1, qui est la valeur la plus basse, qui montre que l'objet est placé dans le mauvais cluster. Enfaite le Silhouette Score à trois cas possibles de valeurs :

- **Cas 1 :** $a < b \Rightarrow S > 0$, cela veut dire que la distance moyenne inter-cluster est supérieur à la distance moyenne intra-cluster, donc le clustering est jugé bon, on a des clusters bien séparé avec un petite distance entre les éléments de ces clusters.
- **Cas 2 :** $a = b \Rightarrow S = 0$, cela veut dire que la distance moyenne inter-cluster est égal à la distance moyenne intra-cluster, donc le clustering est jugé moyen, on a des clusters qui ne sont pas bien séparés avec des éléments qui tendent entre deux clusters. C'est une valeur donc à éviter.
- **Cas 3 :** $a > b \Rightarrow S < 0$, cela veut dire que la distance moyenne inter-cluster est inférieur à la distance moyenne intra-cluster, donc le clustering est jugé mauvais, on a des clusters pas bien séparé avec des éléments qui sont dans le mauvais clusters. C'est une valeurs donc à éviter.

S est la moyenne des silhouettes scores et permet d'avoir une vision de la qualité globale du clustering, c'est donc S qui sera calculé dans nos tests, et qui permettra d'optimiser les paramètres de Mean Shift et DBSCAN. Nous recherchons alors une valeur de S qui soit comprise entre 0 et 1.

Maintenant que nous avons choisi le silhouette score comme étant la métrique qui nous permettra de valider le clustering, nous devons établir un processus de test pour comparer Mean Shift et DBSCAN.

5.3 : Solution proposée et sa mise en œuvre

L'algorithme que nous avons décidé d'utiliser est DBSCAN, car nous l'avons trouvé plus adapté aux besoins du projet, nous verrons les tests et comparaisons que nous avons effectués entre ces deux algorithmes pour arriver à ce résultat dans la partie [suivante](#). Comme nous l'avons vu, DBSCAN est un algorithme basé sur la densité et la distance des données. Voici un rappel de son fonctionnement:

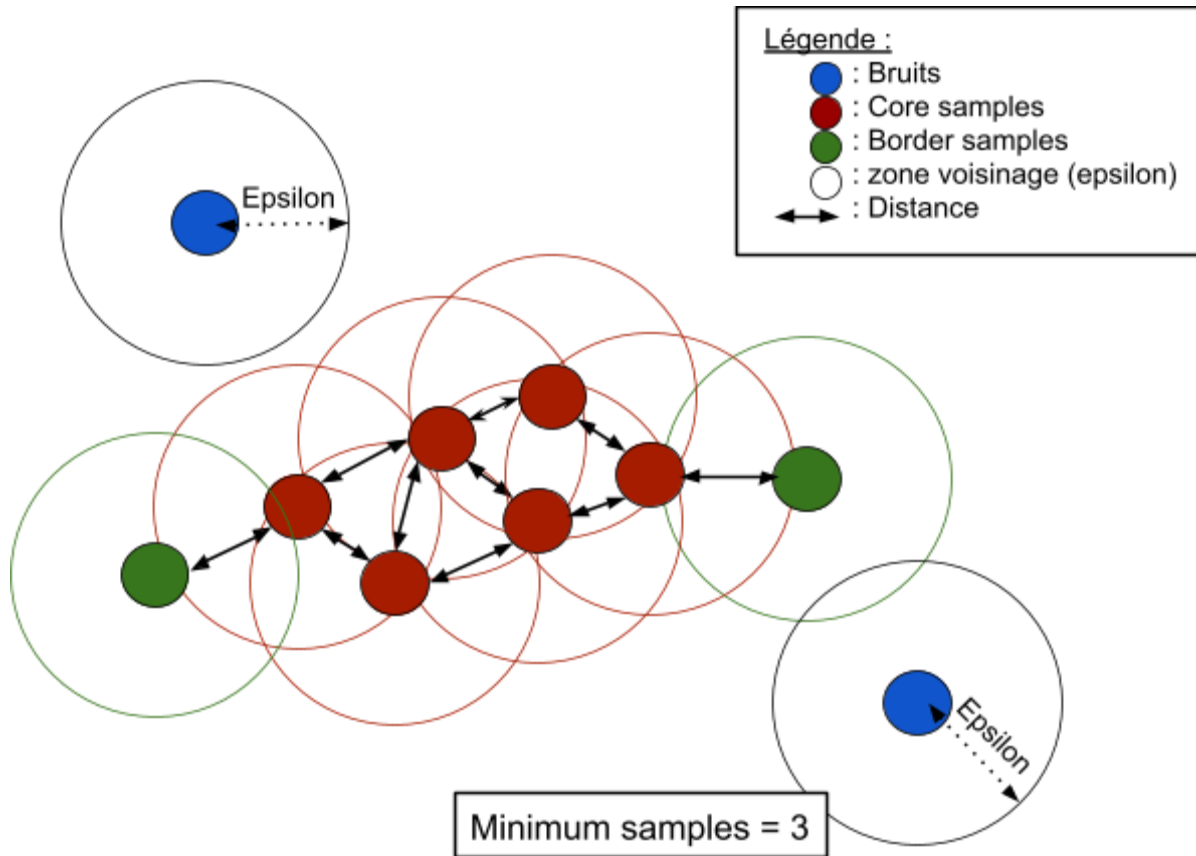


Figure 14: Schéma rappel du fonctionnement de DBSCAN

Pour appliquer cet algorithme dans notre projet, nous avons utilisé la bibliothèque sklearn pour python qui nous donne accès à un ensemble de méthodes qui permettent d'exécuter le clustering. La principale difficulté en utilisant ces méthodes déjà fournies a été de bien comprendre le fonctionnement de chaque paramètre afin de pouvoir déterminer leurs valeurs optimales. Après de nombreux tests ([Voir 5.4 : Tests et certifications de la solution](#)) nous avons déterminé que le paramètre optimal pour ϵ est de 0,4. Ce paramètre correspond au type de données que nous utilisons, et nous pouvons donc le fixer quel que soit le nombre de données.

Le second paramètre à fixer, qui est plus complexe, est le paramètre "min_sample" qui correspond au nombre minimum de valeurs voulues dans l' ϵ -voisinage d'un sample pour qu'il

soit considéré comme un core sample. Ce paramètre ne peut pas être fixé identiquement pour tous les clusters étant donné que nous n'avons pas le même nombre de tweets chaque jour. Pour certains jours nous avons des millions de tweets, pour d'autres seulement quelques milliers, nous ne pouvons donc pas les traiter de la même façon. Nous avons donc, après différents essais, fixé le `min_sample` à 0.5% du total de tweets par jour. En d'autres termes, pour qu'un event soit considéré par notre application sur un jour J, il faut qu'il contienne 0.5% des tweets de la journée. Ce pourcentage nous permet de faire un gros tri dans les petits events qui ne nous intéressent pas, par exemple, pour une base de 10 000 tweets, il faudra qu'un événement contienne au moins 50 tweets pour qu'il soit pris en compte. Ce paramètre est donc variable à chaque exécution de l'algorithme (Du moment que les données récupérées ne sont pas uniformément réparties sur les jours).

Afin de mettre en place un système de popularité le plus précis possible et récolter un maximum de données l'algorithme sera lancé chaque heure, ceci permettra d'avoir un spectre de données plus large pour les différents traitements qui seront effectués pour les différents affichages.

5.4 : Tests et certifications de la solution

Pour départager Mean Shift et DBSCAN, nous établissons une procédure de test comportant plusieurs phases afin de comparer les deux algorithmes. Le fait que ces algorithmes soient non supervisés ne facilite pas la tâche, car le nombre de clusters générés, peut être imprévisible. De plus nous devons optimiser les paramètres d'entrée de ces algorithmes afin d'obtenir un résultat satisfaisant, pour les tests et pour notre projet. Nous allons voir dans cette partie la procédure de test pour comparer Mean Shift et DBSCAN. En parlant tout d'abord des jeux de données créés pour ceux-ci, puis en détaillant les différentes phases de tests, pour finalement observer les résultats générés par ces phases et en tirer des conclusions.

5.4.1 Jeu de données

En premier lieu, pour générer des jeux de données avec lesquelles nous allons tester les algorithmes, nous définissons 7 thématiques :

- coronavirus
- peinture
- danse
- prince charles
- magasin
- stade france
- spectacle

Nos jeux de données comportent ces thèmes sous forme de différentes phrases, tweets. On s'attend à ce que ces thèmes soient des clusters. Pour effectuer nos tests, nous créons deux

jeux de données. Le jeu de données *D1*, créé de manière fictive comportant juste des mots, tel que :

- coronavirus x 100 samples
- peinture x 100 samples
- danse x 100 samples
- prince charles x 100 samples
- magasin x 100 samples
- stade france x 100 samples
- spectacle x 100 samples
- coronavirus danse x 10 samples
- prince charles coronavirus x 10 samples
- prince charles coronavirus stade france x 10 samples

D1 nous permet de tester de manière simple, et d'avoir une attente sur le résultat en sortie, en effet ici le résultat attendu est un ensemble de 10 clusters comportant chacun un type d'entrée différent. Maintenant que nous avons créé un jeu de données, simple pour comparer Mean Shift et DBSCAN, *D1*, nous allons créer un jeu de données plus complexe qui se rapproche plus des conditions réelles de notre projet, *D2*.

Le jeu de données *D2*, qui comporte toujours nos 7 thèmes, avec 9998 tweets qui proviennent directement de l'API Twitter. Ce jeu de données est moins contrôlable, car nous savons juste que ces tweets comportent entre 0 et 7 de nos thèmes. Le résultat, sera donc beaucoup moins prévisible, mais *D2* permet de simuler au mieux des conditions réalistes.

Maintenant que nous avons défini deux jeux de données, *D1* et *D2* avec lesquelles nous allons lancer Mean Shift et DBSCAN, et que nous avons choisis la métrique (le Silhouette Score), Nous allons décrire les différentes phases de tests qui compose notre procédure de test.

5.4.2 Les phases de tests

Pour pouvoir tester Mean Shift et DBSCAN, nous devons établir un processus qui se rapproche de l'une des problématiques de notre projet, qui est clusteriser une grande quantité de tweet pour déterminer de potentiels événements. C'est pour cela que notre processus de test comporte trois phases qui vont se rapprocher peu à peu de notre problématique. Les modalités des trois phases se présentent comme ceci :

- Phase 1 : Nous lançons le test pour chaque algorithme sur le jeu de données *D1* avec comme vocabulaire les 7 thèmes et quelque 3-gram qui compose *D1*.
- Phase 2 : Nous lançons par la suite le test des algorithmes sur *D2* avec le même vocabulaire d'entrée que pour la phase 1.
- Phase 3 : Ici nous lançons le test des algorithmes sur *D2* avec une auto-génération en calculant le Tf-Idf de tous les mots de chaque tweet de *D2* pour trouver les features potentielles, on choisit de générer 300 features avec

4-Gram (cf ...) . La phase 3 est celle qui se rapproche le plus des conditions réelles de notre projet.

Pour toutes phases de notre processus de test, nous itérons l'exécution des algorithmes en incrémentant à chaque fois le(s) paramètre(s) de ceux-ci (en commençant par la valeur minimum) et en calculant le Silhouette Score du résultat généré à chaque fois. Cela nous donne une courbe de la valeur du Silhouette Score pour chaque valeur du paramètre (une courbe par paramètre) que l'on compare au nombre de clusters générés pour chaque valeur du paramètre (graphique en dessous d'une courbe). Ce qui nous permet d'observer la meilleure valeur possible pour tous paramètres (Méthode de validation relative), c'est-à-dire là où valeur du silhouette score est à son maximum et si on a le choix, là où le nombre de clusters est minimal. Maintenant que nous avons établi notre procédure de test, nous pouvons observer les résultats obtenus pour les différentes phases de tests.

5.4.3 Les résultats

Nous allons maintenant voir ensemble les résultats obtenus et en tirer des conclusions. Comme nous l'avons vu, le but de ces phases de tests est de voir le comportement de Mean Shift et DBSCAN, face à des situations diversifiées, pour ainsi les comparer.

Première phase

Première phase avec Mean Shift :

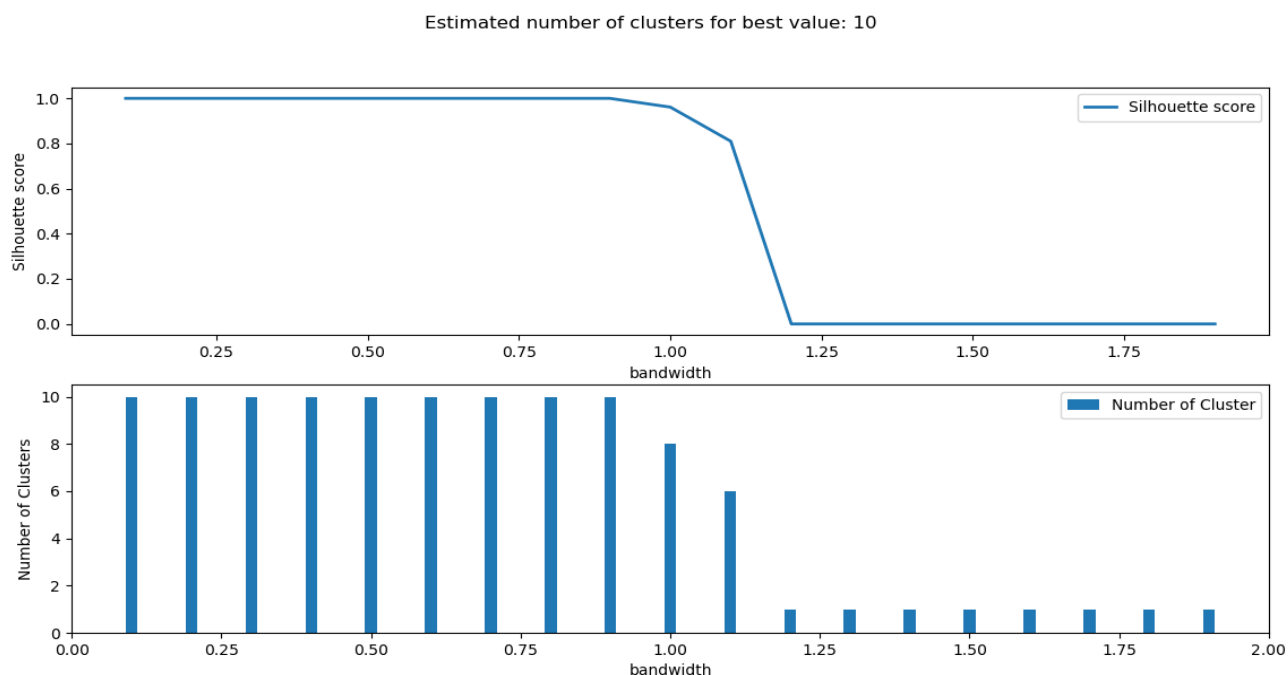


Figure 15 : résultat du test de Mean Shift sur son paramètre bandwidth avec D1.

La figure 15 nous présente donc deux schémas, l'un représentant le silhouette score de Mean Shift sur son paramètre : la bandwidth, appelons le G_1 , et en dessous le nombre de clusters

généralisé pour la même valeur de son bandwidth, appelons le $G2$. Analysons les résultats obtenus sur $G1$. On voit que l'on a trois parties sur la courbe :

- Partie avec les meilleures valeurs : De 0.10 (valeur minimum pour la bandwidth) et ce jusqu'à environ 0.992 nous avons un silhouette score de 1, ce qui est preuve, comme nous l'avons vu, d'un bon clustering, de clusters bien séparés et bien compacts.
- Partie Décroissante : Dès que l'on dépasse 0.90 et jusqu'à 1.20 le silhouette score va commencer à décroître. Ce qui est dû au fait que plus la bandwidth est élevée plus Mean Shift générera des clusters englobant plusieurs samples différents.
- Partie avec valeurs basses : De 1.20 jusqu'à 2.0 (on ne va pas plus loin, car on ne s'intéresse pas à un silhouette score inférieur ou égal à 0), la valeur stagne à 0.

$G2$ nous montre quant à lui que plus la valeur de bandwidth augmente (donc la valeur de Silhouette score diminue), moins Mean Shift générera de clusters. Les phases de $G2$ coordonnent avec celles de $G1$:

- Partie avec les meilleures valeurs : De 0.10 et ce jusqu'à environ 0.99 le nombre de clusters est de 10 ce qui correspond parfaitement à $D1$, un cluster pour toutes entrées différentes.
- Partie décroissante : Dès que l'on dépasse 0.99 et jusqu'à 1.20 le nombre de clusters va commencer à décroître.
- Partie avec valeurs basses : De 1.20 jusqu'à 2.0, la nombre de clusters stagne à 1, un cluster qui englobe tous les samples, ce que nous voulons éviter.

Pour savoir pourquoi l'on obtient ces valeurs de silhouette score, on doit repenser au principe de fonctionnement du bandwidth de Mean Shift. Le bandwidth, comme nous l'avons vu précédemment, définit la zone d'intérêt. Or les vecteurs générés pour les phrases avec plusieurs features, donnent des valeurs normalisées, qui peuvent être inférieures à 1.0, et donc si la valeur de la bandwidth dépasse 0.90, alors la zone d'intérêt va prendre en compte plus d'éléments, et donc le silhouette score va baisser. Comme on peut le voir sur $G2$ le nombre de clusters va lui aussi diminuer jusqu'à former 1 seul qui regroupe toutes les données. La meilleure valeur possible pour le bandwidth se trouve donc entre 0.1 et 0.8, des valeurs avec lesquelles le Silhouette score est à 1. On choisit arbitrairement 0.1 qui génère 10 clusters, ce qui correspond parfaitement à $D1$.

- Première phase avec DBSCAN :

Pour tester DBSCAN, le procédé reste le même que vu plus haut, on teste tous paramètres de DBSCAN (ϵ et minimum de samples) et on incrémente chaque paramètre tout en regardant la valeur du silhouette score, ainsi que le nombre de clusters générés pour cette valeur.

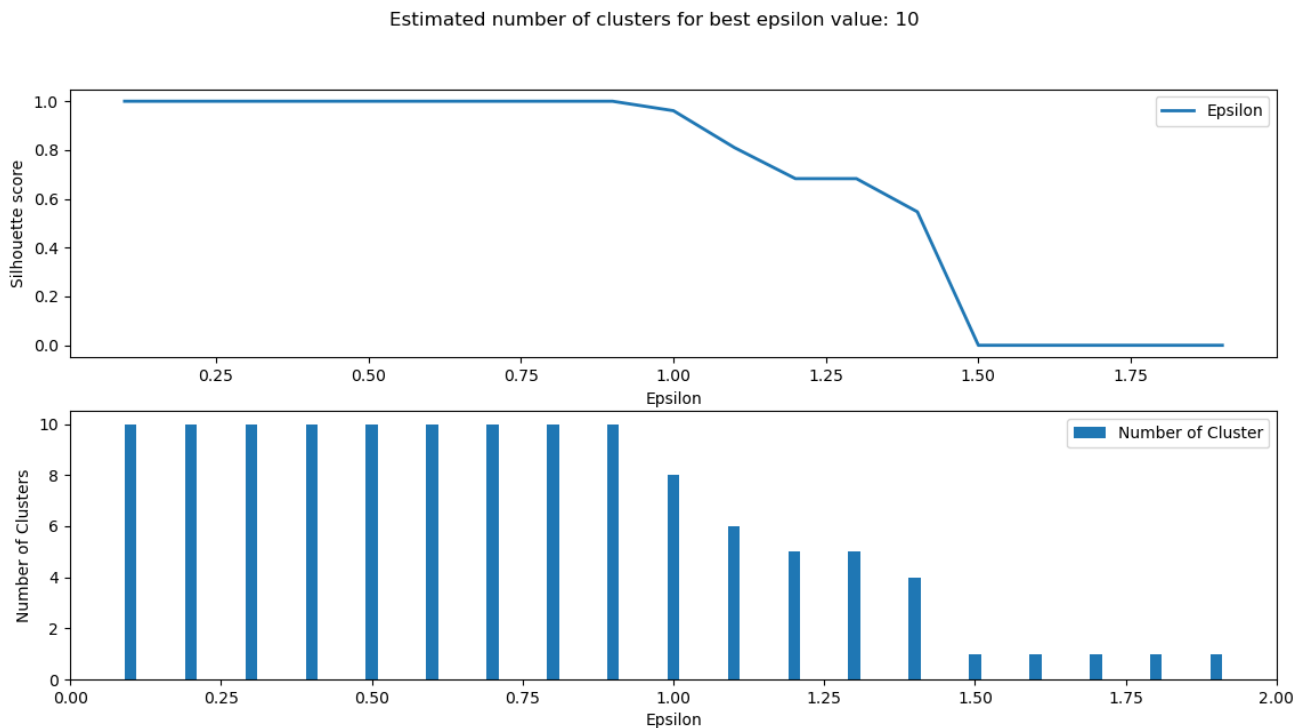


Figure 16 : résultat du test de DBSCAN sur son paramètre ϵ avec $D1$.

La figure 16 nous présente donc deux schémas, l'un présentant le silhouette score de DBSCAN sur son paramètre : ϵ , appelons le $G3$, et en dessous le nombre de clusters générés pour la même valeur, appelons le $G4$. Analysons les résultats obtenus sur $G3$, on voit que l'on a trois parties sur la courbe :

- Partie avec les meilleures valeurs : De 0.10 (valeur minimum pour ϵ) et ce jusqu'à environ 0.99, nous avons un silhouette score de 1, ce qui est preuve, comme nous l'avons vu d'un bon clustering.
- Partie décroissante : Dès que l'on dépasse 0.99 et jusqu'à 1.50 le silhouette score va commencer à décroître. En effet, ϵ a un comportement similaire au bandwidth, plus il augmente plus il prendra en compte de samples dans le voisinage d'un sample, et donc générera des clusters avec des valeurs diversifiées.
- Partie avec valeurs basses : De 1.50 jusqu'à 1.92 (on ne va pas plus loin comme nous l'avons vu pour $G1$), la valeur stagne à 0.

$G2$ nous montre quant à lui que plus la valeur de ϵ augmente (donc la valeur de Silhouette score diminue), moins DBSCAN générera de clusters. Les phases de $G2$ sont :

- Partie avec les meilleures valeurs : De 0.10 et ce jusqu'à environ 0.99 le nombre de clusters est de 10 ce qui correspond parfaitement à $D1$, un cluster pour toutes entrées différentes soit 10 clusters.
- Partie décroissante : Dès que l'on dépasse 0.99 et jusqu'à 1.50 le nombre de clusters va commencer à décroître.
- Stagnation : De 1.50 jusqu'à 1.9, la nombre de clusters stagne à 1, ce que nous voulons éviter.

À la différence de Mean Shift, on voit clairement sur $G1$ que le silhouette score va descendre de manière moins significative avec une pente plus douce, mais qui décroît au même moment (0.99). On peut donc supposer que ε a un impact moins décisif sur la qualité du clustering. on choisit donc 0.1 comme valeur de ε . Voyons maintenant la courbe et le graphe pour le paramètre min samples.

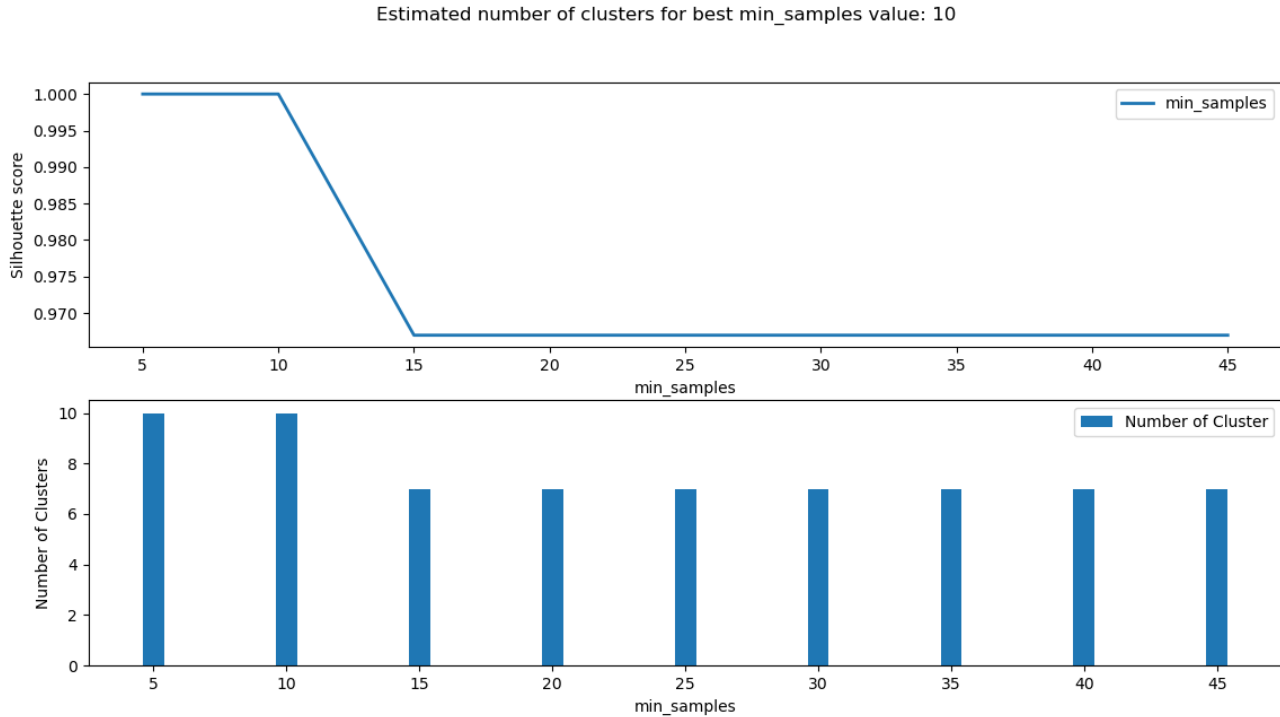


Figure 17 : résultat du test de DBSCAN sur son paramètre min samples avec $D1$.

La figure 17 nous présente donc deux schémas, l'un présentant le silhouette score de DBSCAN sur son paramètre : min samples, appelons le $G5$, et en dessous le nombre de cluster générés pour la même valeur, appelons le $G6$. Analysons les résultats obtenus sur $G5$:

- Partie avec les meilleures valeurs : De 5 (valeur minimum pour min samples) et ce jusqu'à 10, un silhouette score de 1, ce qui est preuve d'un bon clustering, nous choisirons alors une valeur entre 5 et 10.
- Partie Décroissante : Dès que l'on dépasse 10 et jusqu'à 15 le silhouette score va commencer à décroître, mais pas de manière significative.
- Stagnation : De 15 jusqu'à 45 (on ne va pas plus loin car la valeur maximale pour le silhouette score a été trouvé), ensuite la valeur silhouette score stagne à 0.967.

$G5$ nous montre que plus la valeur de min samples augmente (donc la valeur du Silhouette score diminue), moins DBSCAN générera de clusters. Les parties du graphique $G6$ sont :

- Partie avec les meilleures valeurs : De 5 et ce jusqu'à 10 le nombre de clusters est de 10 ce qui correspond parfaitement à $D1$, un cluster pour toutes entrées différentes soit 10 clusters.
- Décroissance et stagnation : Dès que l'on dépasse 15 et jusqu'à 45 le nombre de clusters va commencer à décroître et rester à 7.

On remarque donc que le paramètre min samples n'a que peu de valeur possible (entre 5 et 10), on prend donc comme valeurs 5 pour min samples. Comme nous l'avons vu min samples est un paramètre qui fait varier la taille du voisinage nécessaire pour qu'un point soit considéré comme "core-sample", un membre d'un cluster, il est donc normal qu'avec sa modification le silhouette score ne varie pas énormément, car en augmentant min samples, on augmente aussi la taille possible d'un cluster, cela fera juste varier le nombre de clusters. De plus il est important de notifier que DBSCAN n'a pas généré de cluster contenant le bruit car $D1$ contient seulement des mots qui sont dans les features.

Conclusion de la phase 1 de test : Pour l'instant les résultats de la première phase nous montrent deux choses. Tout d'abord les deux algorithmes fonctionnent parfaitement pour un jeu de données simples, dans les deux cas on obtient le résultat attendu (10 clusters). Ensuite on remarque une légère sensibilité du paramètre bandwidth de Mean Shift comparé à ϵ de DBSCAN, qui nous fait dire DBSCAN est plus optimal pour cette première phase de test dans le sens où la variation du paramètre ϵ prend plus de valeurs à donner un silhouette score de 0. Le clustering reste le même, avec un cluster pour chaque mot différent de $D1$, soit 10 clusters. Regardons le contenu de ces clusters pour chaque algorithme, dans le tableau 7.

	Mean Shift bandwidth = 0.1		DBSCAN $\epsilon=0.1$ et min samples = 5	
	Contenu et proportion (%)	nombre de tweets	Contenu et proportion (%)	nombre de tweets
Cluster 1	prince charles : 100 %	100	coronavirus : 100 %	100
Cluster 2	coronavirus : 100 %	100	peinture : 100 %	100
Cluster 3	danse : 100 %	100	danse : 100 %	100
Cluster 4	stade france : 100 %	100	prince charles: 100 %	100
Cluster 5	magasin : 100 %	100	magasin : 100 %	100
Cluster 6	peinture : 100 %	100	stade france : 100 %	100
Cluster 7	spectacle : 100 %	100	spectacle : 100 %	100
Cluster 8	prince charles coronavirus : 100 %	10	coronavirus danse : 100 %	10
Cluster 9	prince charles coronavirus stade france : 100 %	10	prince charles coronavirus : 100 %	10

Tableau 7 : Comparatif quantitatif des clusters de Mean Shift et DBSCAN avec $D1$.

Le tableau 7, nous montre donc que le clustering avec les paramètres optimaux, est parfait. Il est intéressant de montrer que DBSCAN garde l'ordre des entrées de $D1$. On a donc la confirmation que les deux algorithmes fonctionnent bien. Maintenant rajoutons un peu de difficultés à nos tests avec la seconde phase.

Deuxième phase

La phase 2 de notre processus de test se rapproche de la phase 1 en utilisant cette fois si $D2$ qui comporte 9998 tweets collectés avec le même vocabulaire que la phase 1, on a donc des vecteurs de 10 dimensions. Le but est de lancer Mean Shift et DBSCAN avec un jeu de données plus complexe et en lien avec notre projet toutes en gardant le contrôle car nous utilisons ici les mêmes features que pour la phase 1. Testons en premier lieu Mean Shift.

- Deuxième phase avec Mean Shift :

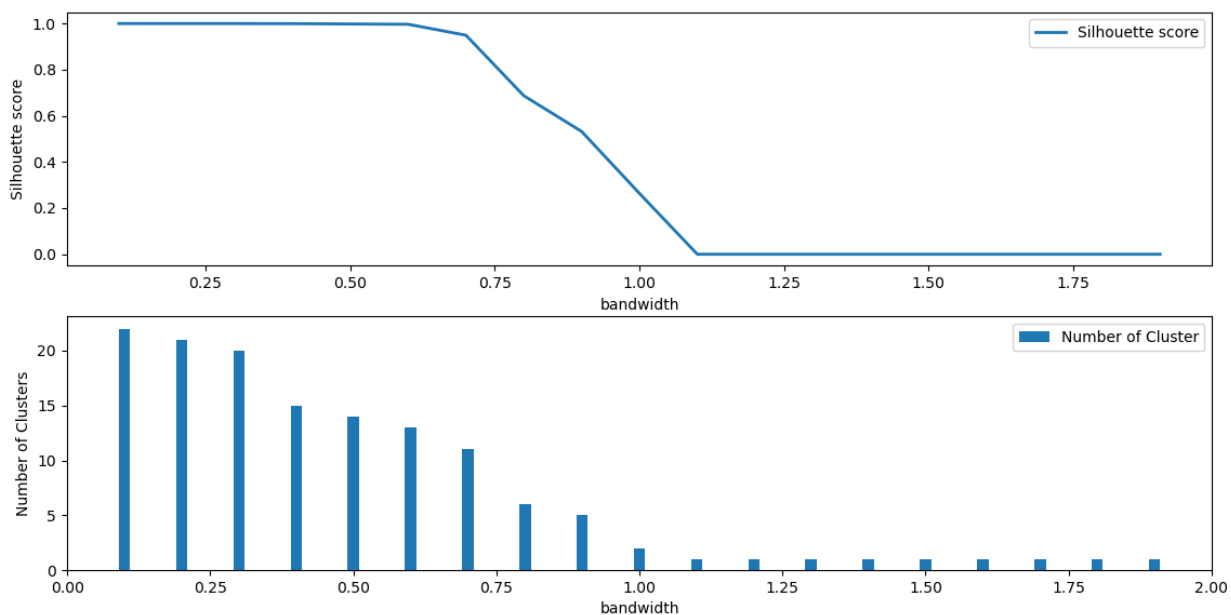


Figure 18 : résultat du test de Mean Shift sur son paramètre bandwidth avec $D2$.

La figure 18 nous présente donc deux schémas, l'un présentant le silhouette score de Mean Shift sur son paramètre : la bandwidth, appelons le $G1$, et en dessous le nombre de cluster généré pour la même valeur de son bandwidth, appelons le $G2$. Analysons les résultats obtenus sur $G1$, on voit que l'on a trois parties sur la courbe :

- Partie avec la meilleure valeur & décroissance : De 0.10 on obtient la meilleure valeur du silhouette score de 1 jusqu'à environ 0.60 de bandwidth, puis la valeur va décroître jusqu'à une valeur du bandwidth de 1.10.
- Partie avec valeurs basses : De 1.10 jusqu'à 2.0 la valeur stagne à 0.

$G2$ nous montre quant à lui que plus la valeur de bandwidth augmente (donc la valeur de Silhouette score diminue), moins Mean Shift générera de clusters, comme nous l'avons vu dans la phase 1. Les parties de $G2$ coordonnent avec celles de $G1$:

- Meilleur valeur & décroissance : De 0.10, là où l'on obtient la meilleure valeur du silhouette score, le nombre de clusters varie. Nous devons donc faire un choix pour le bandwidth pour réduire le nombre de clusters et se rapprocher du nombre de features (10).
- Stagnation : De 1.10 jusqu'à 2.0 , la nombre de clusters stagne à 1. Ce sont donc des valeurs à éviter.

Une valeur de 0.6 de bandwidth semble être un choix judicieux, car le silhouette score est 1 ce qui est excellent et le nombre de clusters est le plus petit comparé aux autres valeurs de bandwidth où le silhouette score est de 1. On voit donc que comme lors de la phase 1, le bandwidth va faire diminuer la valeur du silhouette score beaucoup plus rapidement, et donc le nombre de clusters va diminuer plus rapidement. Comme nous l'avons vu dans la partie sur la [vectorisation](#), les tweets contenus dans *D2* produisent des vecteurs plus complexe qu'avec les tweets de *D1*. En effet les tweets peuvent contenir plusieurs features et même plusieurs fois cette même feature, on a donc des vecteurs aux valeurs très variées. Le fait que lorsque la bandwidth augmente le silhouette score diminue (et le nombre de clusters diminue), est due au fait que la valeur de la distance moyenne inter-cluster diminue petit à petit, car la zone d'intérêt augmente, l'algorithme fusionne alors plusieurs clusters, pour ne former au final qu'un seul cluster entre 1.10 et 2 de bandwidth. Finalement pour la meilleure valeur de silhouette score on obtient un bandwidth de 0.1 et un nombre de clusters de 13.

- Deuxième phase avec DBSCAN :

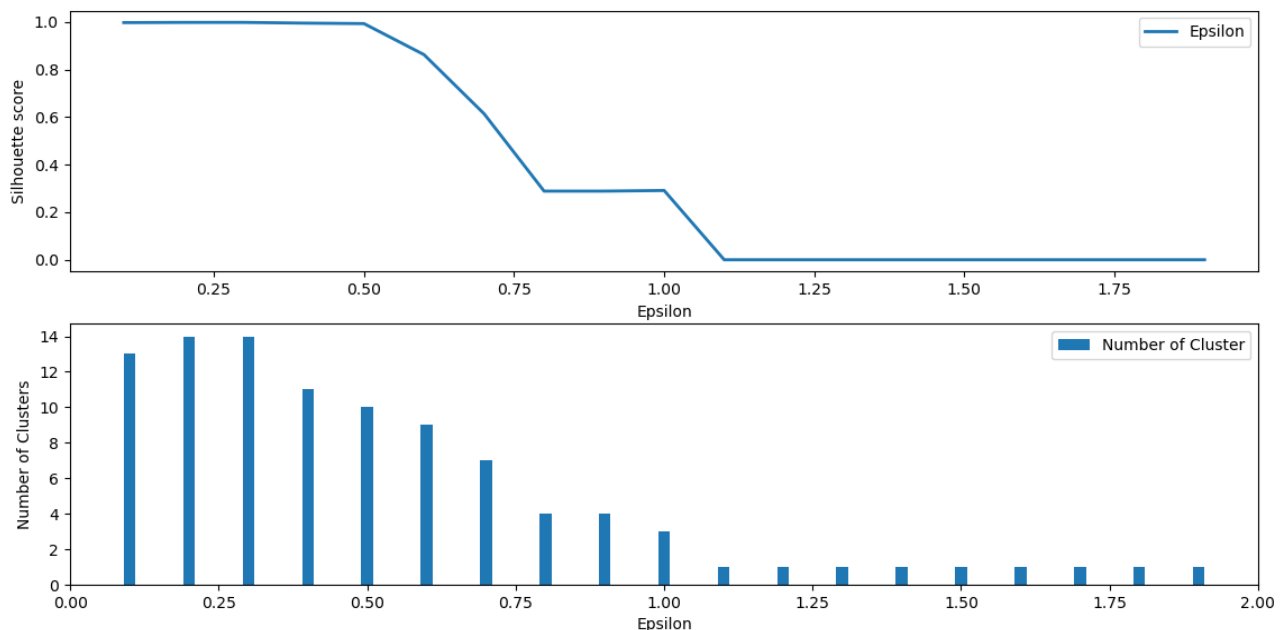


Figure 19 : résultat du test de DBSCAN sur son paramètre ϵ avec *D2*.

La figure 19 est similaire au schéma *G1* et *G2* de la phase 1. On voit que la valeur optimale de ϵ se trouve entre 0.1 et 0.5 avec un silhouette score d'environ 1. Pour $\epsilon = 0.5$, la valeur de *S* est égal à 1 et le nombre de clusters est minimal, on choisit donc cette valeur pour ϵ .

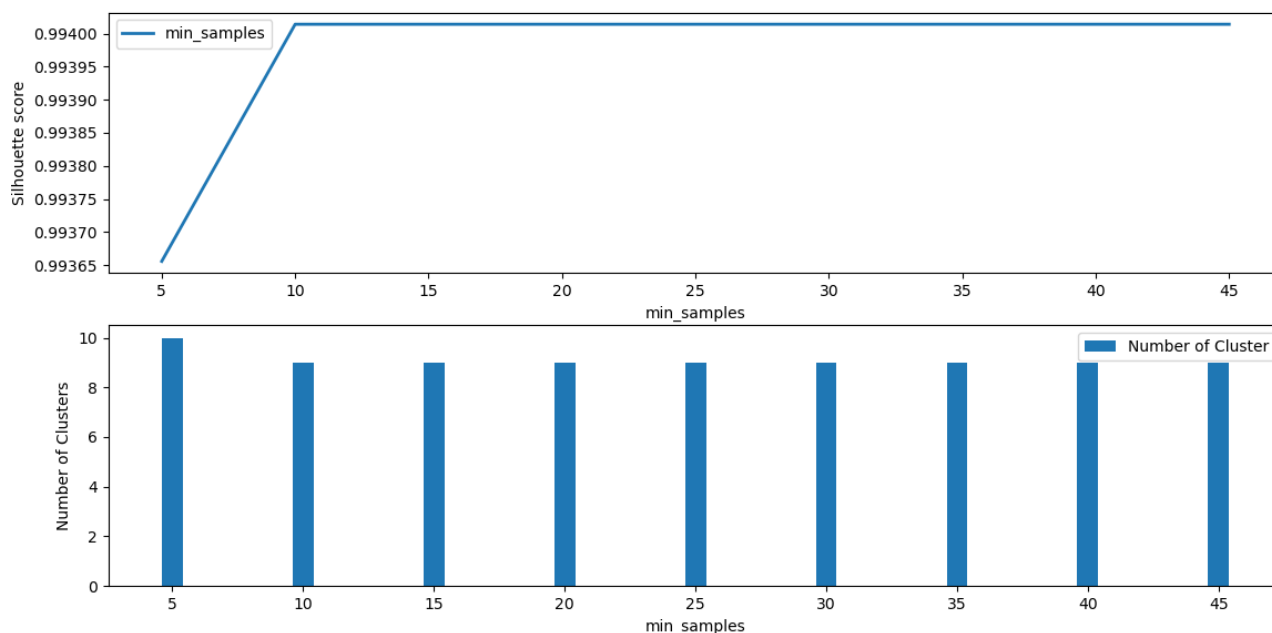


Figure 20: résultat du test de DBSCAN sur son paramètre min samples avec D2.

La figure 20 nous présente donc une courbe et un graphique sur le paramètre min samples en prenant en compte la valeurs optimale de $\epsilon=0.5$, précédemment trouvée.

On remarque 2 parties sur la courbe :

- **Augmentation** : Chose qui diffère de la phase 1, on trouve une augmentation en début de courbe, entre 5 et 10, cela nous montre qu'avec $\epsilon=0.5$ une valeur de 5 ne serait pas un choix judicieux car G4 nous montre que pour 5 le nombre de clusters est à son maximum.
- **Partie à valeur basse** : Après un min samples de 10 la valeurs du silhouette score stagne à 0.994, on s'arrête donc de calculer le silhouette score à min samples de 45 car le maximum à été trouvé.

Comme dans la phase 1, la variation du paramètre min_samples, ne fera pas beaucoup changer le silhouette score, mais plutôt le nombre de clusters. On choisit donc la valeur min samples où la valeur de silhouette score est maximale et où le nombre de clusters est minimal, c'est-à-dire pour 10.

Nous avons donc les valeurs optimums pour tous paramètres de nos algorithmes, on peut donc maintenant comparer les clusters générés par Mean Shift et DBSCAN (cf tableau 8).

	Mean Shift bandwidth = 0.6		DBSCAN $\epsilon=0.4$ et min samples = 10	
	Contenu et proportion (%)	nombre de tweets	Contenu et proportion (%)	nombre de tweets

Cluster 1	coronavirus : 100 %	1791	coronavirus : 100 %	1791
Cluster 2	{aucunes features ⁴ } : 99.7% stade france: 0.3%	3095	{aucunes features} : 99.7% stade france: 0.3%	3095
Cluster 3	magasin : 99.5% , peinture magasin : 0.5%	1213	magasin : 97.7%, coronavirus magasin: 1.4%, peinture magasin : 0.9%, coronavirus peinture magasin : 0.1%	1236
Cluster 4	peinture : 99.6%, peinture magasin : 0.4%	1184	peinture : 100 %	1179
Cluster 5	danse : 100 %	1055	danse : 100 %, danse spectacle : 0.3%	1058
Cluster 6	coronavirus spectacle : 100 %	849	coronavirus spectacle: 100 %	849
Cluster 7	spectacle : 99.8%, stade france spectacle : 0.2%	450	spectacle : 99.6%, danse spectacle : 0.2% stade france spectacle : 0.2%	451
Cluster 8	{aucunes features} : 94.5% ,spectacle: 5.5%	275	{aucunes features} : 94.5%, coronavirus : 5.5%	275
Cluster 9	Stade france : 100 %	57	Stade france : 100 %	57
Cluster 10	coronavirus magasin : 94.4%, coronavirus peinture magasin : 5.6%	18		
Cluster 11	peinture danse : 100%	6		
Cluster 12	danse spectacle : 100%	4		
Cluster 13	danse magasin: 100%	1		
Total de donnée	9998		9991 donc 7 tweets considérés comme étant du bruits.	

Tableau 8 : Comparatif quantitatif des clusters de Mean Shift et DBSCAN avec D2

Lorsqu'on regarde le tableau 8, on se rend compte de plusieurs choses. Tout d'abord DBSCAN génère moins de cluster (9 clusters) que Mean Shift (13 clusters), ce qui est une bonne chose. Lorsque l'on regarde le contenu des clusters, on remarque que pour Mean shift on trouve des clusters qui contiennent plusieurs fois les mêmes features, par exemple

⁴ Aucunes features équivaut à un vecteur nulle.

“peinture magasin” qui se retrouve dans les clusters 4,5 et 10. De plus lorsque l’on regarde les clusters générés par Mean shift, les clusters de 10 à 13 sont des clusters avec très peu de membres, qui pour nous ne . DBSCAN ne générera pas ces clusters, notamment grâce à son paramètre min samples qui comme nous l’avons vu ne générera pas de clusters avec un nombre de tweets inférieur à sa valeur : 10. On remarque aussi que DBSCAN fusionne le cluster 10 de Mean Shift, dans le cluster 3, ce qui est une bonne chose car le cluster est trop faible pour être un cluster (seulement 18 tweets).

Conclusion de la phase 2 de test : Avec les différentes courbes obtenues et le tableau X comparatif, on se rend compte que Mean Shift aura tendance à clusteriser même de très faibles quantités de tweets, alors que DBSCAN, par son paramètre min samples combineras ces tweets dans un seul cluster qui contient une plus grosses quantité de tweets. Ce qui est un bonne choses car on fait le choix d’éviter d’avoir dans notre clustering des clusters avec une quantité trop faible de tweets, qui ne seront pas considéré comme des événements. D’où l’importance de choisir un valeur pour minimum samples qui génère un nombre acceptable de clusters.

Troisième phase

La phase 3 de notre processus de test se rapproche de la phase 2 en utilisant cette fois si $D2$ mais en générant le vocabulaire de manière automatique avec le calcul du Tf-Idf (cf [4.3.2 : Vectorisation des données](#)). pour chaque mot. On décide de générer 300 features allant jusqu’à 4-Gram à partir de $D2$, on aura donc des vecteurs de 300 dimensions. On s’attend donc à plus ou moins 300 clusters. Voyons maintenant les courbes et les graphes générés pour la phase 3.

- Troisième phase avec Mean Shift :

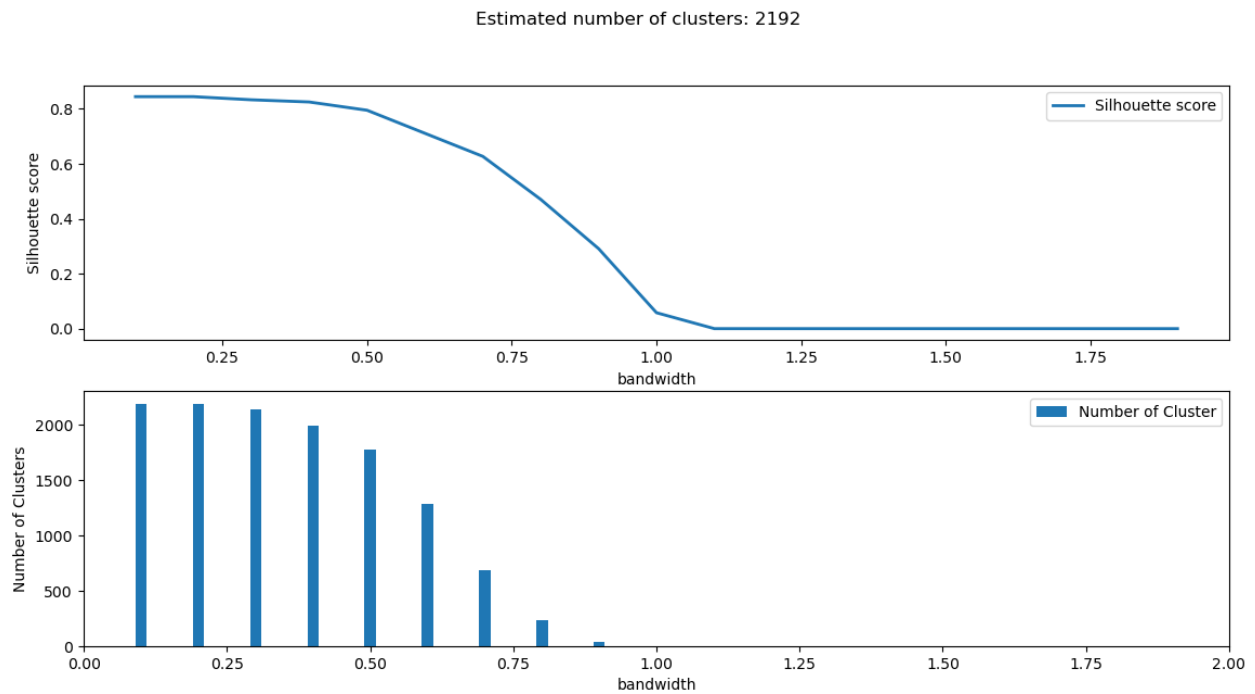


Figure 21 : résultat du test de Mean Shift sur son paramètre bandwidth phase 3 du test.

Comme on peut le voir sur la figure 21, la courbe est divisée en trois phases :

- Meilleure valeur : De 0.10 et ce jusqu'à environ 0.4 un silhouette score de 0.77, ce qui n'est pas optimal mais qui reste très bon.
- Décroissance : Dès que l'on dépasse 0.4 et jusqu'à 1.10 le silhouette score va commencer à décroître.
- Stagnation : De 1.20 jusqu'à 2.0, la valeur stagne à 0 .

Pour la première fois le silhouette score maximal ne dépasse pas les 0.9, car les 300 features générées avec des 4-Gram, ce qui créera des vecteurs aux valeurs plus variées que pour la phase 1 ou 2, une valeur pour la bandwidth implique moins de tweets dans un cluster, mais avec un bon silhouette score car la distance moyenne intra-cluster sera petite. Une grande valeur pour la bandwidth implique plus de tweets dans un cluster. On choisit donc une bandwidth de 0.4, ce qui nous génèrent 2192 clusters, ce qui est mauvais car cela veut dire que même si le silhouette score est bon, Mean Shift va clusteriser de très petites quantités de tweets, comme nous l'avons vu dans la phase 2.

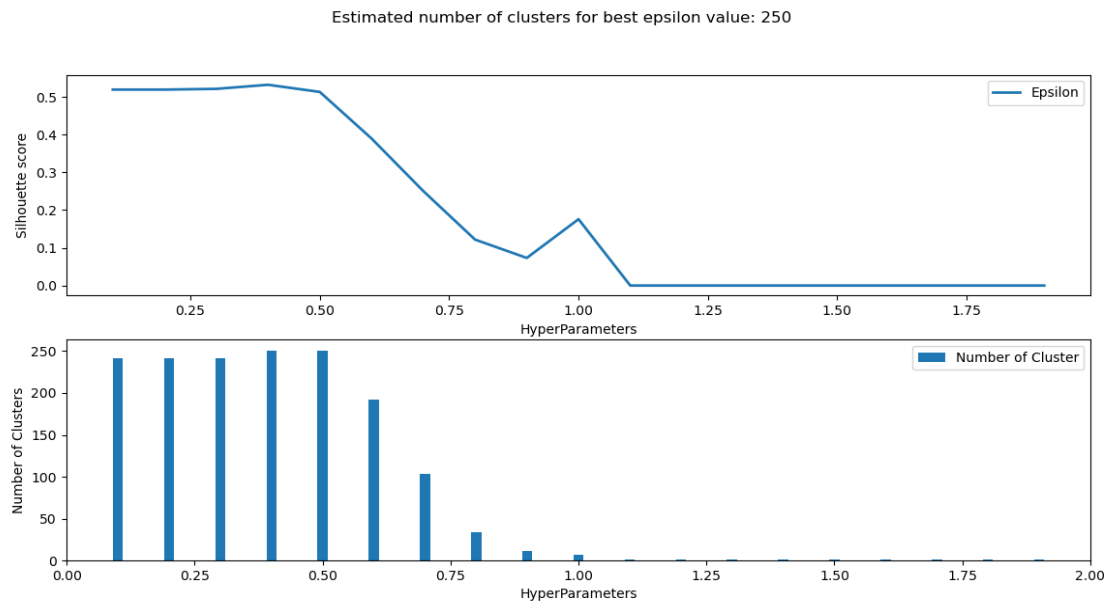


Figure 22 : résultat du test de DBSCAN sur son paramètre Epsilon phase 3 du test.

Comme on peut le voir sur la figure 22, la courbe nous montre que comme Mean Shift, le silhouette score pour DBSCAN est plus bas que pour les autres phase. En effet le silhouette score de 0.55 nous montre que la distance moyenne intra-cluster, surpasse presque celle inter-cluster, on a donc des clusters moins compacts. La valeur maximum est atteinte pour $\epsilon = 0.4$, qui génère 250 clusters. Le nombre de clusters est quant à lui très satisfaisant car il se rapproche du nombre de features généré.

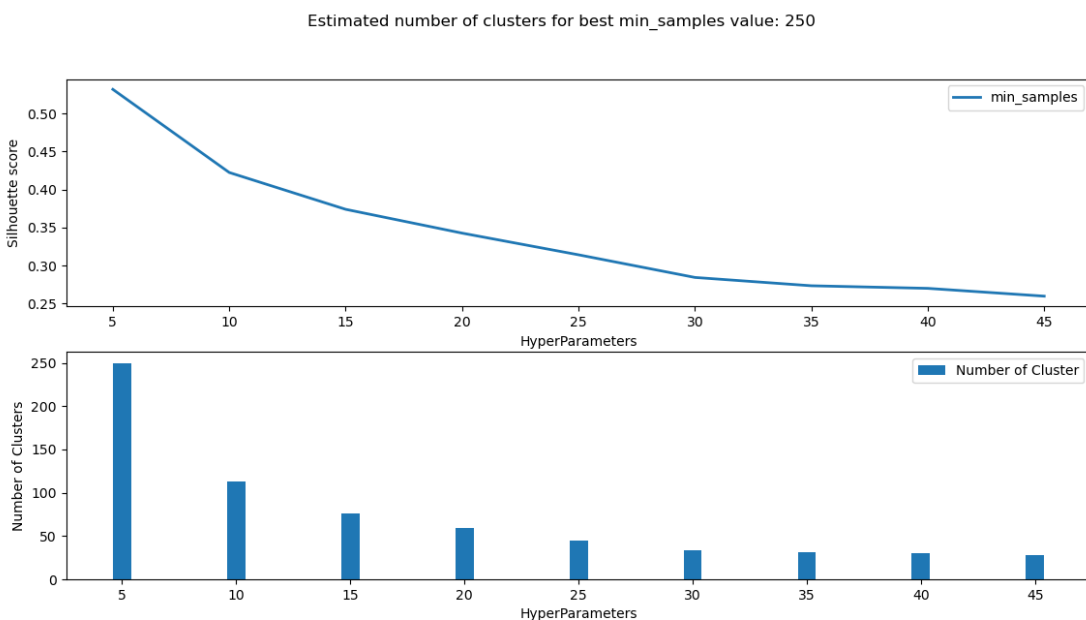


Figure 23 : résultat du test de DBSCAN sur son paramètre min samples phase 3 du test.

Comme nous le montre la figure 23, le silhouette score maximum est atteint pour $\text{min samples} = 5$, car cette valeur de 5 permet à DBSCAN de générer des clusters de petite taille, donc de réduire la distance moyenne intra-cluster, ce qui améliore le Silhouette Score. Cette valeur est donc très bien pour repérer un grand nombre de clusters. Dans le cadre de ce projet il vaut mieux prendre une valeur plus grande afin de réduire le nombre de clusters "parasite". Le silhouette score sera donc plus bas dans le cadre général du clustering, mais correspondra mieux aux besoins de notre projet.

Conclusion de la phase 3 et conclusion final des tests : Finalement avec cette phase finale, qui simule au mieux notre projet, on se rend bien compte que Mean Shift ne répond pas aux attentes, en générant environ 3000 clusters pour 300 features, alors que DBSCAN n'en génère que 250. Avec la phase 2 et 3, Mean Shift a montré qu'il regroupait des petites quantités de tweets, ce que nous voulons éviter. DBSCAN avec son paramètre minimum samples nous permettra de plus contrôler le nombre de clusters généré. De plus, les temps d'exécution sont plus longs avec l'algorithme mean-shift. On choisit donc DBSCAN comme étant l'algorithme de clustering de notre projet.

Chapitre 6 : Prédictions d'événements populaires

Nous avons donc vu dans les parties précédentes les réponses aux problématiques suivantes : comment traiter des données textuelles (NLP), et les numériser (Vectorisation), ainsi que comment regrouper ces données (DBSCAN) en clusters? Mais maintenant que nous avons vu cela, il nous reste une dernière problématique à laquelle il nous faut répondre : Comment prédire les possibles événements du lendemain ?

6.1 : Analyse de la problématique

Nous allons voir dans cette partie, comment répondre à cette problématique : Comment prédire les possibles événements du lendemain. Dans un premier temps nous verrons pourquoi prédire les événements du lendemain est une chose intéressante, et dans un second temps qu'est que nous devons vraiment prédire.

6.1.1 Pourquoi prédire.

Comme nous l'avons dit précédemment, nous voulons ajouter à notre application un côté prédiction d'événement. Voyons ensemble ce que cela signifie. Jusqu'alors notre projet détectait des événements, l'utilisateur de notre application, pouvait alors consulter ces événements. Même si ces événements peuvent différer du système de tendance proposé par Twitter qui affiche les sujets les plus populaires, le principe reste globalement le même. Là où notre application WebSensor, peut faire la différence, c'est en proposant un système de prédiction d'événement pour les jours futurs, ce qui pour l'utilisateur serait une aubaine, car il pourrait voir avec cela, les événements qui vont possiblement être populaires ou pas, et donc au final maximiser la popularité de son compte twitter en devançant les tendances Twitter. Maintenant que nous avons défini pourquoi prédire les futurs événements, nous allons voir qu'est que nous devons prédire, tout en définissant des notions de notre projet qui seront utilisées à cela, tel que la popularité.

6.1.2 Quoi prédire.

Nous avons vu dans le [chapitre 5](#) que les regroupements de nos données, les tweets, formaient des clusters, qui sont identifiables dans un premier temps par la valeur des vecteurs dans un cluster, puis par les features (dimensions du vecteur). En effet, les dimensions des vecteurs correspondent chacune à une feature ([partie 4.3.2](#)), on retrouve alors facilement, les features qui définissent un cluster. De plus comme nous l'avons vu DBSCAN donne un label aux données en gardant l'ordre d'entrée, nous savons donc les labels associés pour tous tweets. Une brève analyse des features pour chaque tweet et pour tous les labels, permet de savoir quelles features définit un cluster. On a donc maintenant, pour un cluster i , $C_i(f, d, t, u)$ avec i allant de 0 au nombre de clusters générés, ses features f et les tweets qui le composent, donc

sa taille t , d la date du cluster, ainsi que le nombre d'utilisateurs u^5 qui ont tweetés les tweets du cluster. Nous avons maintenant la définition claire d'un cluster dans notre projet, mais comment les départager ?

L'une des problématiques de notre projet est de détecter les Event à partir de Twitter, or nous avons défini un event comme dépendant de sa popularité. La popularité est une notion dont nous parlons beaucoup dans ce rapport sans l'avoir vraiment défini, remédions à cela. La popularité d'un événement, permet de définir si un sujet (feature) crée des interactions sur Twitter. Il y a de multiples interactions sur Twitter, écrire un tweet, commenté, retweeté, aimer un tweet, rien qu'un simple clique sur un tweet est quantifiable par l'application. C'est donc par la quantité de ces interactions que nous allons calculer la popularité d'un cluster $C_i(f, d, t, u)$ de tweets.

Le calcul de la popularité P pour un cluster C_i est le suivant :

$$p(C_i) = \frac{\alpha \times t + \beta \times u}{T + U}$$

Formule 6 : Calcul de la popularité

Avec t la taille du cluster, u le nombre d'utilisateurs qui ont tweetés les tweets du cluster, T le nombre total de tweet dans le jeu de données et U le nombre d'utilisateurs qui ont tweetés dans le jeu de données, et α, β deux coefficient réel. La popularité étant maintenant définie, c'est donc avec celle-ci que nous allons classer les clusters et donc prédire cette valeur pour les jours suivants.

6.2 : Etat de l'art des prédictions

Prévoir des événements a toujours été un enjeu important. Afin de pallier à cette problématique, des scientifiques ont mis en place ce qu'on appelle les séries temporelles. Ces séries ont pour but de calculer (ou prévoir) une chose précise à l'aide de données recueillies par le passé. Cette technique n'est pas récente, car elle existait déjà au prémices de l'astronomie vers 500 après J.C. L'astronomie est basée en grande partie sur ces séries temporelles, il s'agit de prédire les mouvements d'un astre en utilisant les données recueillies de ses mouvements précédents. Plus récemment, cette méthode est utilisée pour le système de météorologie que nous connaissons actuellement. À l'aide de données récupérées depuis des dizaines d'années on essaye de prédire la météo qu'il fera la semaine d'après.

Stationnarité :

La plupart des modèles de séries temporelles se basent sur le principe que la série est stationnaire, c'est-à-dire que ses propriétés statistiques (espérance, variance...) ne varient pas

⁵ En faisant une simple recherche par l'identifiant d'un tweet on retrouve l'utilisateur, le nombre correspond au nombre utilisateurs uniques .

dans le temps, et qu'elle a un comportement particulier sur un temps donné, car on peut alors penser que ce comportement se reproduira à un moment ultérieur.

Définition : Une série à valeurs réelles et en temps discret $Z_1, Z_2 \dots Z_t$ est dite stationnaire si :

- (1) Son espérance $E[Z_i]$ est constante au cours du temps i .
- (2) Sa variance $Var[Z_i]$ est constante et ne varie pas à l'infini au cours du temps i .
- (3) L'auto-corrélation⁶, c'est à dire la corrélation de la variable Z_i par rapport à une version décalé de la variable Z_{i-k} , que l'on note $Cov[Z_i, Z_{i-k}]$ avec i le temps et k le décalage. L'auto-corrélation, ne doit pas dépendre du temps, tel que $Cov[Z_i, Z_{i-k}] = f(i, k) = p_k$ mais seulement de l'ampleur du décalage k , tel que $Cov[Z_i, Z_{i-k}] = f(k) = p_k$.

La définition de la stationnarité nous informe alors que pour faire une prédiction sur une série temporelle ne doit pas avoir de tendance (courbe croissante ou décroissante avec le temps), tel que dit dans les conditions (1) et (2), et ne doit pas dépendre du temps, comme dit dans la condition (3). Il est alors difficile de s'imaginer que dans des conditions réelles ces trois hypothèses soient validées. En effet les séries temporelles ont souvent, des tendances et des variations qui surviennent de manières répétées (saisonnalités). Nous devons donc transformer une série temporelle non-stationnaire en une série temporelle stationnaire, ou du moins s'en rapprocher le plus possible. Nous devons donc trouver un moyen de tester la stationnarité d'une série.

Test de Dickey-Fuller :

Le test Dickey-Fuller [8] ou test de la racine unité, est un test statistique qui permet de vérifier la stationnarité d'une série. Il estime l'hypothèse H de non-stationnarité avec p une constante réelle, tel que :

$$H_0 : p = 1$$

$$H_\alpha : |p| < 1$$

Formule 7 : hypothèse de non-stationnarité.

L'hypothèse nulle implique qu'il existe une racine unitaire dans le modèle auto-régressif. Considérons une série à valeurs réelles et en temps discret $Z_1, Z_2 \dots Z_t$, représentée sous forme autorégressive d'ordre 1 avec ε l'erreur, tel que :

⁶ L'auto-corrélation est un outil mathématique qui permet de détecter des régularités sur une courbe.

$$Z_t = p \times Z_{t-1} + \varepsilon_t$$

Formule 8 : forme autorégressive d'ordre 1.

La manière la plus simple pour réaliser le test est d'utiliser une statistique de Student associée à H_0 . Nous calculons la p-valeur, qui est la probabilité pour un modèle statistique sous l'hypothèse nulle, d'avoir une valeur supérieure ou égale à la valeur observée, et nous la comparons aux niveaux de significativités (valeurs critiques 1%, 5%). Si la p-valeur est inférieure à l'un de ces niveaux, alors l'hypothèse nulle est rejetée. Notre série est alors stationnaire.

$$p = P(x | H_0)$$

Formule 9 : Calcul de la p-valeur du résultat observé x

Rendre stationnaire une série temporelle :

Voyons ensemble les techniques pour rendre la série stationnaire. Comme nous l'avons vu, généralement une série peut être décomposée en 3 composantes qui sont:

- Une tendance générale
- Une variance saisonnière (ex: Augmentation de la température en été)
- Du bruit, résidus ou bien erreurs de nature irrégulières, aléatoires et inexplicables.

Prenons en exemple comme la vente de billet d'avion sur une période de 145 mois en tant que série temporelle (donc une donnée tous les mois), si on affiche cette série on obtient la figure 24.

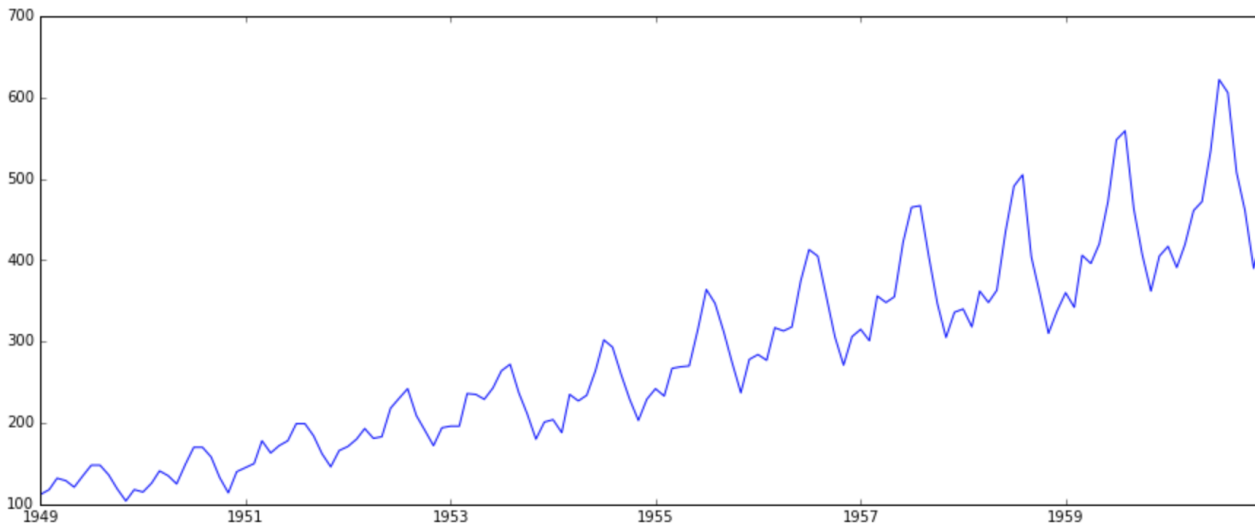


Figure 24 : série temporelle de vente de billet d'avion

Sur la figure 24 on voit clairement les composantes d'une série temporelle, une tendance qui augmente de mois en mois, ainsi que des saisonnalités, ces piques plus petits qui sont le résultat de paramètres saisonniers, et qui se répètent. En effet on vend plus de billet en période de vacances, ce qui explique ces saisonnalités. Il est intéressant de séparer de la série temporelle, les tendances et les saisonnalités, ainsi que ça résiduelle comme dans la figure 25.

Pour ainsi supprimer celles-ci de la série et obtenir une série stationnaire. Il arrive que le bruit soit stationnaire, on peut donc l'utiliser pour faire des prédictions.

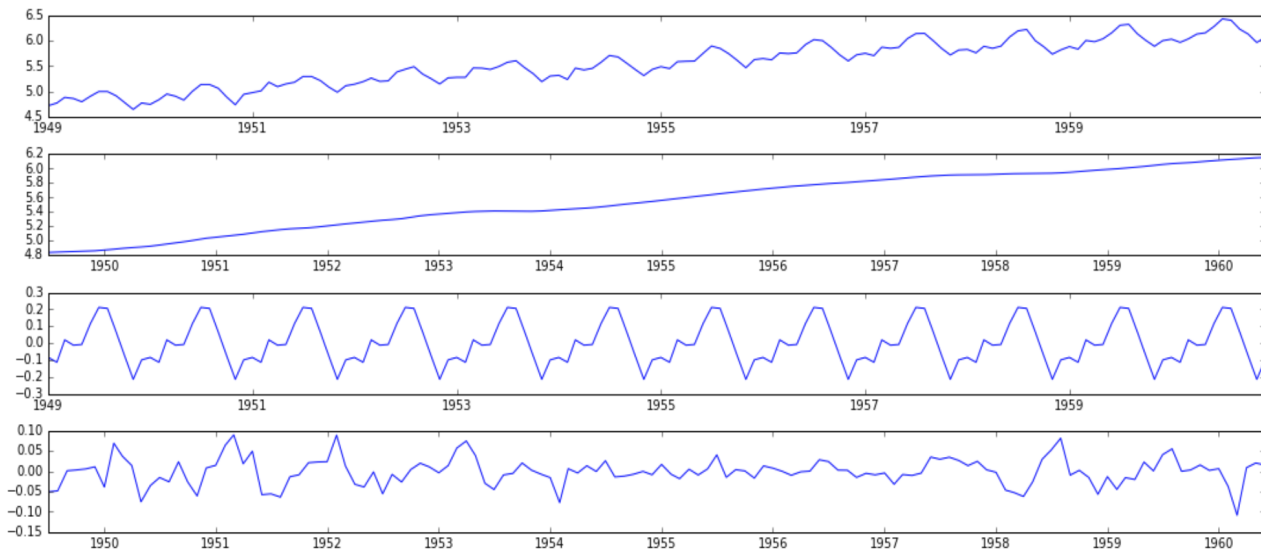


Figure 25 : série temporelle séparée

Dans la figure 25, nous avons de haut en bas : (1) le logarithme de la série, (2) la tendance, (3) les saisonnalités et (4) le bruit (résidus). L'intérêt du logarithme appliqué sur la série, avant de séparer ses composantes, est d'adoucir la tendance, qui va avoir pour effet de baisser les valeurs hautes. Le fait de séparer de la série temporelle de sa tendance, sa saisonnalité et de son bruit (résiduel), s'appelle la méthode de décomposition. Cette méthode, en plus d'offrir un visuel sur les composantes de la série temporelle, permet de supprimer la tendance et les saisonnalités ce qui corrigerait une série non stationnaire en série stationnaire. Une autre technique qui permet de rendre une série non stationnaire en série stationnaire est la méthode dite de différenciation. Cette méthode consiste à prendre la différence de l'observation à un instant particulier avec l'observation à un l'instant précédent de celle ci. Plus la série temporelle est complexe plus le nombre de différences est nécessaire.

Définition : Une série X est **stationnaire en différence** si la série obtenue en différenciant les valeurs X_t de la série originale est stationnaire, avec t un instant de la série. On note la différence, telle que :

$$\Delta X_t = X_t - X_{t-1}$$

Formule 10 : opérateur de différence

Maintenant que nous avons vu comment rendre une série stationnaire, Nous allons voir un algorithme qui permet de prédire, tout en rendant la série stationnaire.

ARIMA :

Il existe des algorithmes qui permettent d'améliorer les prédictions contre certains prérequis. Le plus connu d'entre eux est le modèle ARIMA [9]. Ce modèle se base sur des séries stationnaires, et est incapable de traiter simultanément plus d'une série. Le modèle ARIMA fonctionne avec trois paramètres:

- P : partie autorégressive du modèle, qui permet d'intégrer l'effet des valeurs passées de la série à la prédiction. Plus précisément, P réfère au nombre de décalages nécessaires de la série pour être utilisés comme prédicateurs.
- D : le nombre de différenciation nécessaire pour transformer la série non stationnaire en série stationnaire.
- Q : partie moyenne glissante du modèle, définit l'erreur du modèle, comme une combinaison linéaire des valeurs d'erreurs observées à des instants précédents. Q réfère au nombre d'erreurs de prédiction retardée.

La démarche est donc de trouver une combinaison correcte de ces trois paramètres, afin d'obtenir une prédiction optimale.

Choix de D :

Comme nous l'avons le but de D est de rendre la série stationnaire, plus précisément c'est le nombre de différenciation à faire pour rendre la série stationnaire. D peut donc être égal à 0 si la série est déjà stationnaire. Il nous faut alors effectuer les différenciations et tester leurs stationnarités avec le test de Dickey-Fuller, jusqu'à obtenir une série temporelle stationnaire.

Choix de P :

L'étape est maintenant de savoir si notre modèle a besoin de termes autorégressif. Pour cela nous allons utiliser une dérivée de la fonction d'autocorrélation (ACF), la fonction d'autocorrélation partiel (PACF). Décrivons tout d'abord la fonction d'autocorrélation (ACF).

Fonction d'auto-corrélation (ACF) :

L'analyse de la fonction d'autocorrélation est un outil mathématique permettant de trouver des modèles répétitifs, comme la présence d'un signal périodique obscurcit par le bruit. Comme nous l'avons vu, l'auto-corrélation d'une série temporelle, p_k , fait référence au fait que la mesure d'un phénomène à un instant t peut être corrélée aux mesures précédentes. On calcule l'auto-corrélation p_k pour tout instant t et pour une variable Z_t de moyenne μ et de variance σ^2 telle que :

$$p_k(Z, t) = \frac{E((Z_t - \mu)(Z_{t+k} - \mu))}{\sigma^2}$$

Formule 11 : calcul de p_k

La valeur de p_k nous informe donc sur la corrélation entre un point et son k-ième point précédent. Plus la valeur est élevée plus les points sont corrélés (valeurs comprises entre -1 et 1). Par définition $p_0 = 1$, ce qui est logique car avec un décalage de 0, toutes les observations sont corrélées à elles-mêmes. L'affichage de la fonction d'auto-corrélation s'appelle un corrélogramme. Le corrélogramme nous permet d'observer la valeur de la corrélation (entre 1 et -1) par rapport au nombre de décalages, ainsi on peut voir pour quelle valeur de décalage nous avons la meilleure corrélation. Tout corrélogramme commence à 1, car avec 0, on la compare avec elle-même.

Fonction d'auto-corrélation partiel (PACF) :

Tout comme l'ACF, il s'agit de la corrélation partielle (comme la corrélation, elle mesure le degré d'association entre deux variables aléatoires, mais avec l'effet d'un contrôle sur les variables aléatoires qui pourraient influencer les deux variables analysées), entre les valeurs avec un décalage k et la valeur à l'instant "présent", mais cette fois-ci, en prenant aussi en compte les retards plus courts (inférieur à k). En analysant le corrélogramme de PACF, on peut trouver le nombre optimal pour P , car PACF montre la corrélation entre un décalage et la série et nous montre si un décalage est nécessaire. La corrélation partielle d'un décalage k d'une série, est le coefficient α de ces décalages dans l'équation auto-régressive y , tel que :

$$y_t = \alpha_0 + \alpha_1 y_{t-1} + \dots + \alpha_k y_{t-k}$$

Formule 12 : équation auto-régressive

Donc la corrélation partielle d'un décalage k est son coefficient α_k . Toutes les autocorrélations dans une série stationnaire peuvent être rectifiées en ajoutant suffisamment de termes autocorrélés, P . La valeur de P donne l'ordre du modèle. Ainsi, nous prenons initialement l'ordre des termes autocorrélés pour qu'ils soient égaux au nombre de retards qui franchissent la limite de l'intervalle de confiance dans le tracé de PACF calculé, avec n le nombre d'observations dans la série, tel que :

$$\left[\frac{-1.96}{\sqrt{n}} ; \frac{1.96}{\sqrt{n}} \right]$$

Formule 13 : Intervalle des valeurs critiques

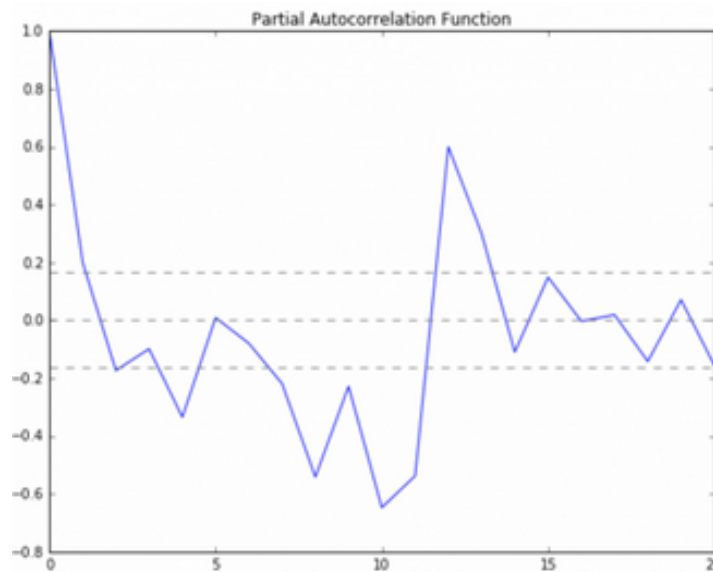


Figure 26 : corrélogramme PACF

Comme on peut le voir dans la figure 26, le corrélogramme du PACF de la vente de billet d'avion de la figure 24, nous permet d'observer la valeur de la corrélation (entre 1 et -1) par rapport au nombre de décalages, ainsi que l'intervalle de confiance (Formule 13). Ici pour P nous prendrons 2 comme valeur car c'est à ce décalage (abscisse) que la valeur de la corrélation va rester à la limite supérieure de l'intervalle de confiance. C'est donc la valeur la plus haute de corrélation avec une assez bonne confiance. On ne prend pas en compte les autres valeurs avec un décalage plus grand, car la valeur de corrélation reste la même sur la limite supérieure de l'intervalle de confiance.

Choix de Q :

Pour rappel Q est le nombre de termes moyenne-mobile, qui sont les erreurs des prédictions décalées dans une équation de prédiction. Les modèles à moyenne mobile suggèrent que la série présente des fluctuations autour d'une valeur moyenne. L'équation d'une observation y_t , est représentable tel que :

$$y_t = \theta_1 \varepsilon_{t-1} - \dots - \theta_k \varepsilon_{t-k} + \varepsilon_t$$

Formule 14 : équation moyenne-mobile

On comprend alors que chaque observation est composée d'une composante d'erreur aléatoire ε et d'une combinaison linéaire des erreurs aléatoires passées. $\theta_1, \dots, \theta_k$ sont les coefficients de moyenne mobile du modèle. La valeur de Q donne l'ordre du modèle. Tout comme nous avons fait avec le PACF pour les termes autocorrélés, nous pouvons utiliser l'ACF pour obtenir le nombre de termes q et en affichant l'intervalle de confiance.



Figure 27 : Corrélogramme ACF

Comme on peut le voir dans la figure 27, le corrélogramme du PACF de la vente de billet d'avion de la figure 24, qui nous permet d'observer la valeur de la corrélation (entre 1 et -1) par rapport au nombre de décalages, ainsi que l'intervalle de confiance. Ici pour Q nous prendrons 2 comme valeur car c'est à ce décalage que la valeur de la corrélation va rester à la limite de l'intervalle de confiance. C'est donc la valeur la plus haute de corrélation avec une assez bonne confiance.

Avec le ACF pour q, le PACF pour p et le test de Dickey-Fuller pour d, nous avons vu trois méthodes pour optimiser les paramètres du modèle ARIMA, qui nous permet de faire une prédiction. On peut donc conclure, qu'un modèle ARIMA est un modèle dans lequel la série temporelle a été modifiée au moins une fois pour la rendre stationnaire et dans lequel nous combinons les termes autocorrélées et de moyenne mobiles. Ainsi, l'équation devient :

$$y_t = \alpha_0 + \alpha_1 y_{t-1} + \dots + \alpha_k y_{t-k} + \theta_1 \varepsilon_{t-1} + \dots + \theta_k \varepsilon_{t-k} + \varepsilon_k$$

Formule 15 : équation d'ARIMA pour un décalage k

6.3 : Solutions apportées et mises en œuvre

Maintenant que nous avons développé l'aspect théorique de la prédiction, nous allons voir comment les solutions ont été apportées et mises en œuvre pour répondre à la problématique de prédiction. Quand on compare ce que nous avons vu dans la partie précédente et ce que nous avons avec le clustering, nous voyons directement des liens. En effet, comme nous l'avons vu en partie 6.1.2, à la fin du clustering nous obtenons les clusters $C_i(f, d, t, u)$ et leur popularité p . On peut donc établir pour tout cluster, une série temporelle définie par la

popularité du cluster par jour*d*. Pourvu qu'un event se répète sur plusieurs jours, cela créera donc une série temporelle de cet événement. Nous avons décidé d'utiliser le modèle ARIMA pour établir nos prédictions. Le modèle nous donnera la possible popularité d'un jour j , nous décidons de limiter la prédiction à seulement 3 jours, pour garder une certaine précision (cf. partie 6.4).

Nous avons maintenant une série temporelle et un modèle de prédiction, ces deux éléments nous permettent d'obtenir des prédictions sur la popularité d'un event. nous allons voir comment le projet utilise les résultats des prédictions. Tout d'abord, nous ajoutons un classement des 10 clusters les plus populaires (top 10) pour le jour actuel, ensuite nous établissons une prédiction, pour chacun des clusters, ce qui nous permet d'établir le classement pour les jours suivants. Ainsi l'utilisateur de l'application pourra voir les tendances changer.

Cependant, comme vu dans la [partie 3.3.1](#), nos données sont mal réparties dans le temps, et certains jours il n'y a pas eu de récupération de données. De ce fait, les prédictions se basent sur une semaine entière pour déterminer les tendances d'un jour, alors qu'il faudrait idéalement utiliser les données uniquement de la veille, voir d'un jour de plus, car les discussions sur twitter évoluent très vite. Prédire une tendance pour un jour J en utilisant les données datantes de 15 jours avant est donc très peu pertinent, mais il s'agit de la seule solution possible au vu de nos données.

6.4 : Tests et certifications de la solution

Les tests sont encore en cours, mais nous connaissons déjà la démarche avec laquelle nous allons procéder :

En première étape, nous allons prendre comme série temporelle la popularité de l'event "coronavirus" sur plusieurs jours (du 11 février 2020 au 26 février 2020). Puis vérifier la stationnarité de la série avec test Dickey Fuller et la rendre stationnaire si besoin. Ensuite nous estimerons les trois paramètres d'ARIMA comme nous l'avons vu dans la partie précédente. Au final, faire la prédiction et l'observer les résultats, ainsi que calculer les erreurs de prédiction.

Chapitre 7 : Rendu final

Dans ce chapitre, nous parlerons de la version finale de notre projet. Pour cela, nous verrons dans un premier temps l'interface utilisateur finale, ainsi que les tests et certifications sur le bon fonctionnement de l'application.

7.1 : Interface utilisateur finale

L'interface utilisateur de notre application est mise sous la forme d'un site web qui permet une utilisation en ligne sans besoins de téléchargements. Comme vue dans la [partie 2.3.1](#), notre site est composé de trois pages, voici un rappel du plan du site décrit plus tôt:

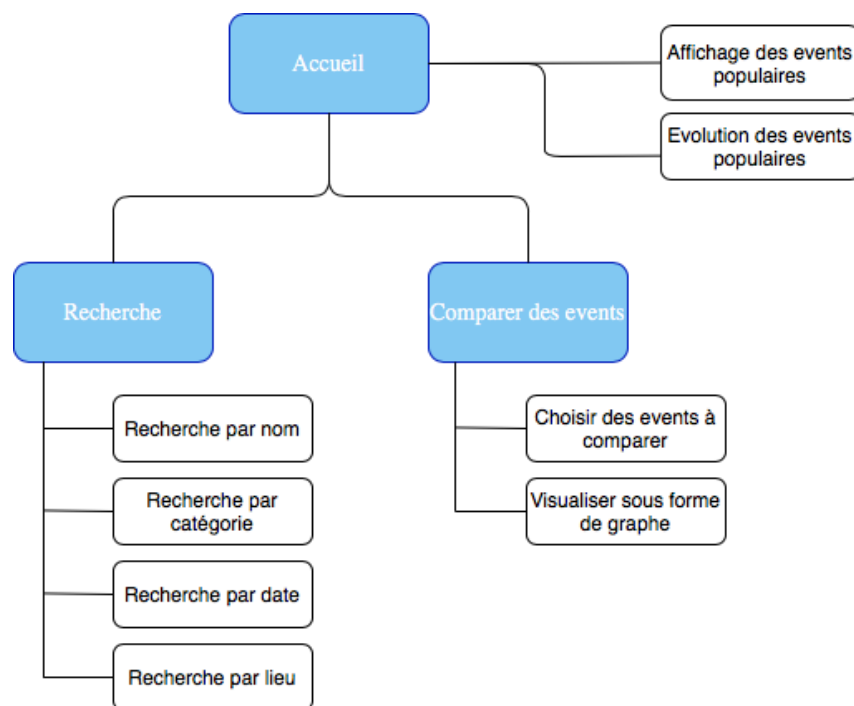


Figure 28: Rappel du plan du site

La première page sur laquelle l'utilisateur arrive est la page d'accueil. Cette page contient une vue sur les events les plus populaires du moment. On y retrouve le top 10 des événements populaires, ainsi que des tweets qui y sont associés. De plus, on peut voir une prédiction des futurs events les plus populaires.

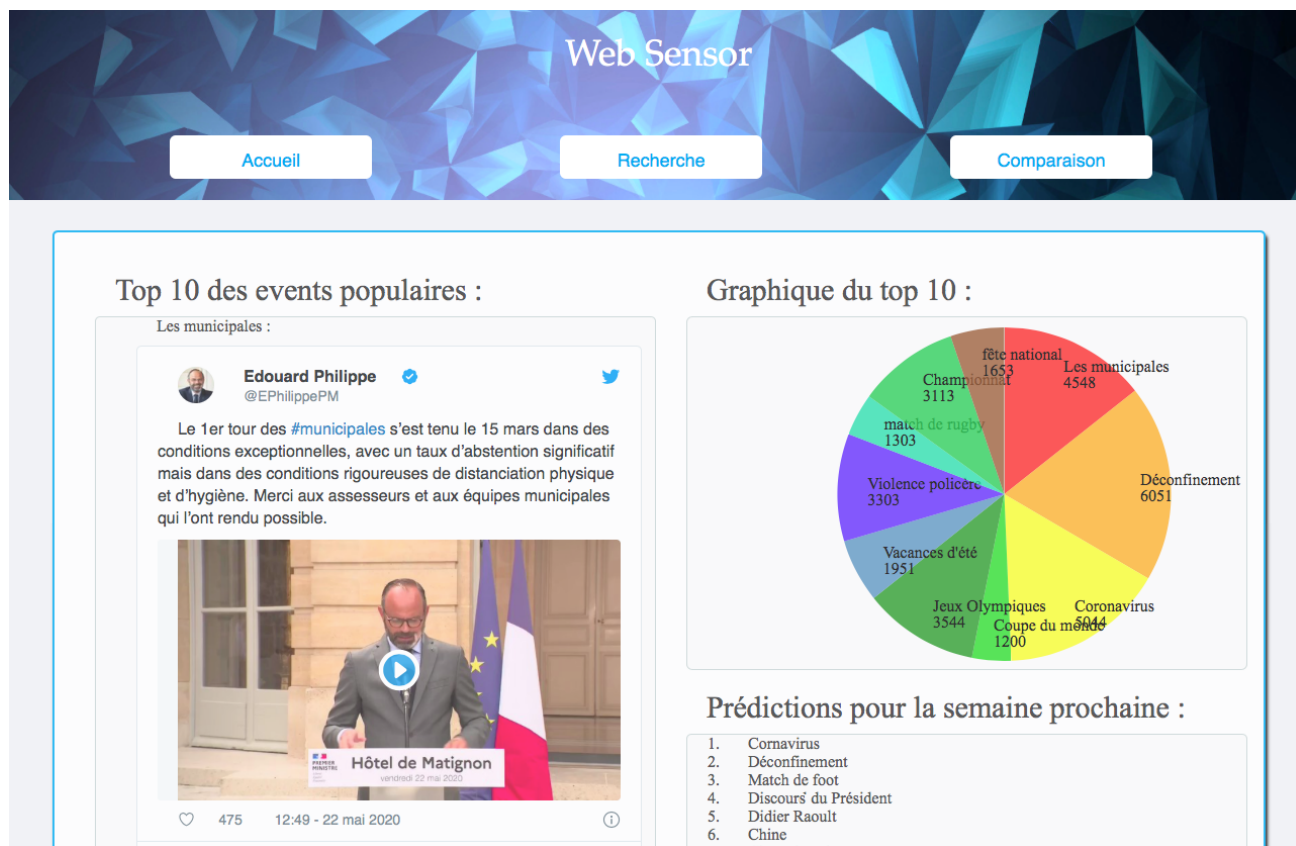


Figure 29: Page d'accueil du site

La seconde page, la page de recherche permet une recherche d'événements. Certains événements populaires ne sont pas figurés parmi les plus populaires du moment, ou bien certains événements ne sont plus populaires, et cette fonctionnalité permet de les retrouver. On peut donc avoir des informations sur ces différents événements plus en détail même s'ils ne sont pas affichés sur la page d'accueil. Il est à noter qu'un événement sur la page d'accueil peut tout aussi bien être recherché pour obtenir plus d'informations dessus.

La recherche se fait par mots-clés ou par nom d'événement, et retourne une liste des résultats correspondants à la recherche. Si plusieurs résultats correspondent à la recherche, alors ils seront triés par ordre de popularité.

La page de comparaison permet une visualisation de l'évolution de deux tendances différentes. Cette comparaison n'a de sens que si les deux tendances ont un lien entre elles. Comparer l'évolution d'une tendance sur un événement sportif avec la tendance du confinement peut être intéressant pour voir l'impact qu'a eu le confinement. En revanche, comparer la coupe du monde de football avec un incendie en Australie n'aura pas d'intérêt. L'utilisateur peut, à son envie, ajouter ou retirer des événements à comparer entre eux. La page se présente ainsi:

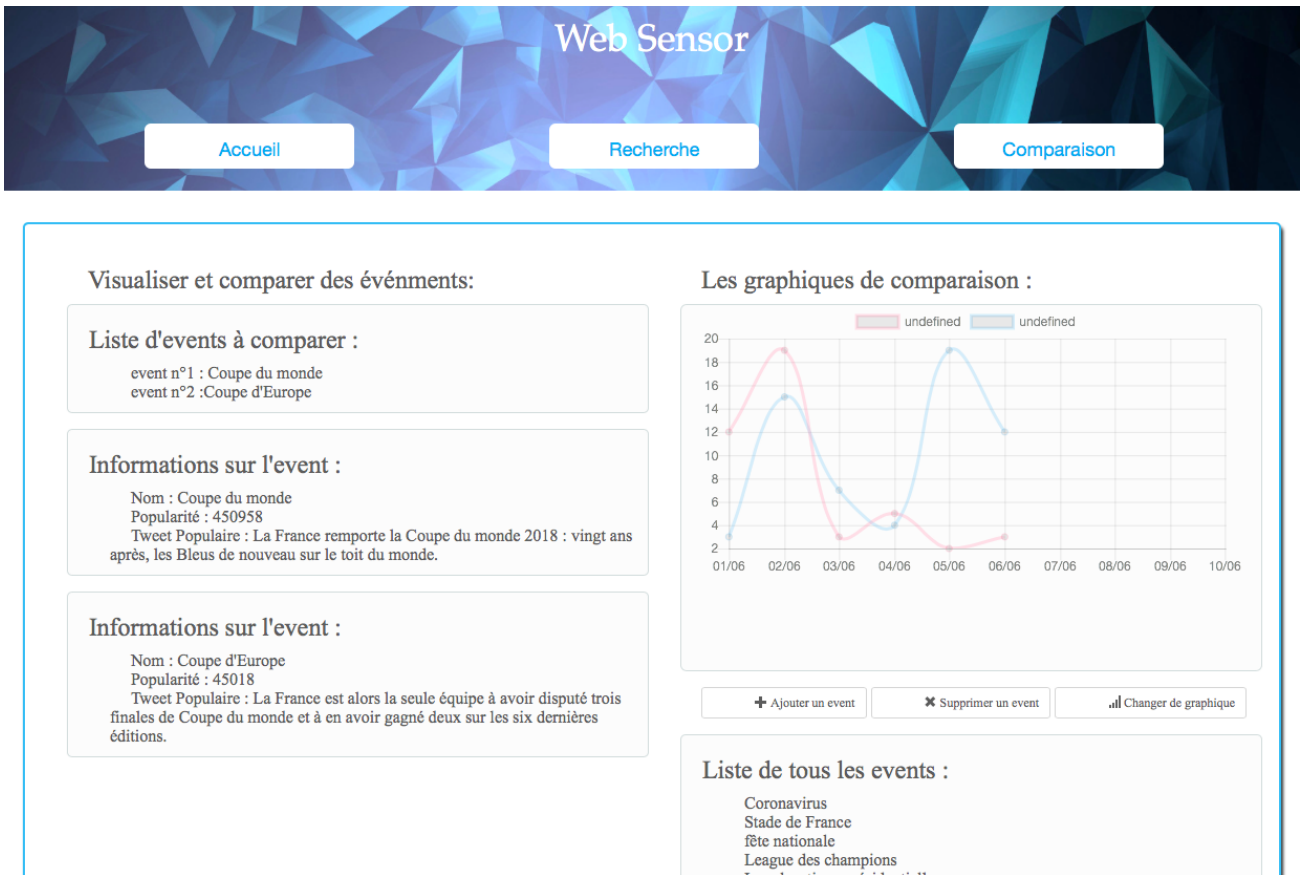


Figure 30: Exemple de comparaison d'événements

7.2 : Tests utilisateur et certification

Afin de certifier nos résultats, nous avons procédé tout d'abord à des vérifications manuelles des tendances obtenues par rapport aux tendances actuelles. Les tendances étant très évolutives très rapidement sur twitter (rotation permanente tout au long de la journée), il est difficile pour l'application d'obtenir des résultats "parfaits" en effectuant le traitement 1 fois par jour, sur des données récupérées sur une plage horaire de quelques heures par jour seulement. Cependant, les tendances globales de notre application respectent bien les tendances twitter, par exemple on retrouve très régulièrement la tendance "Coronavirus" en top de nos tendances, tout comme sur twitter.

Cependant, nous ne pouvons pas nous baser uniquement sur twitter pour vérifier nos résultats, car un certain contenu de notre application est basée sur nos résultats précédents, et pas ceux de Twitter. Il s'agit de la prédiction. En effet, nous ne pouvons pas nous baser sur les tendances évolutives de twitter pour vérifier nos résultats. Pour vérifier ces résultats, nous avons mis en place une comparaison de nos prédictions avec le résultat obtenu le jour J de la prédiction. Plus nos prédictions sont proches de la réalité, plus l'algorithme est performant.

Encore une fois, nous avons rencontré un problème lié à notre récupération de données inégale et mal répartie sur une journée.

À cause de cela, nos prédictions ne peuvent pas s'avérer très fiables, cependant elles peuvent permettre d'avoir quand même une idée sur les futurs événements populaires. De plus, nos tests sur l'algorithme, ainsi que notre documentation sur le sujet, nous permettent d'affirmer qu'avec plus de données (Traitement par heure par exemple) on aurait des résultats bien plus fiables, sans changer quoi que ce soit dans notre partie de prédiction (à part le nombre de données en entrées). Comme dit précédemment, des événements peuvent apparaître en cours de journée, comme par exemple une catastrophe naturelle, un attentat etc., ce qui est donc malheureusement impossible à prédire pour un algorithme qui fonctionne par jour. Afin d'évaluer et de reconnaître la montée de ces tendances en temps réel, il faudrait appliquer le traitement bien plus fréquemment.

7.3 : Certification : grand volume de données

La récupération de tweets s'est vue être inégale au cours du projet, ce qui nous donne un nombre de valeurs très mal réparties dans le temps. Cependant, pour certifier que notre traitement peut gérer ces écarts de valeurs, ainsi qu'une grande quantité de tweets par jour, nous avons effectué le même traitement chaque jour, quel que soit le nombre de tweets récupérés. Pour le calcul de la popularité nous avons opté pour une normalisation des résultats obtenus pour ne pas biaiser les statistiques. En effet, sans cette normalisation on verrait de gros écarts d'une journée à l'autre sur la popularité d'un même événement, qui serait en réalité toujours aussi populaire. La figure 27 nous montre le taux de récupération de tweets entre le 11 février 2020 et le 26 mars 2020.

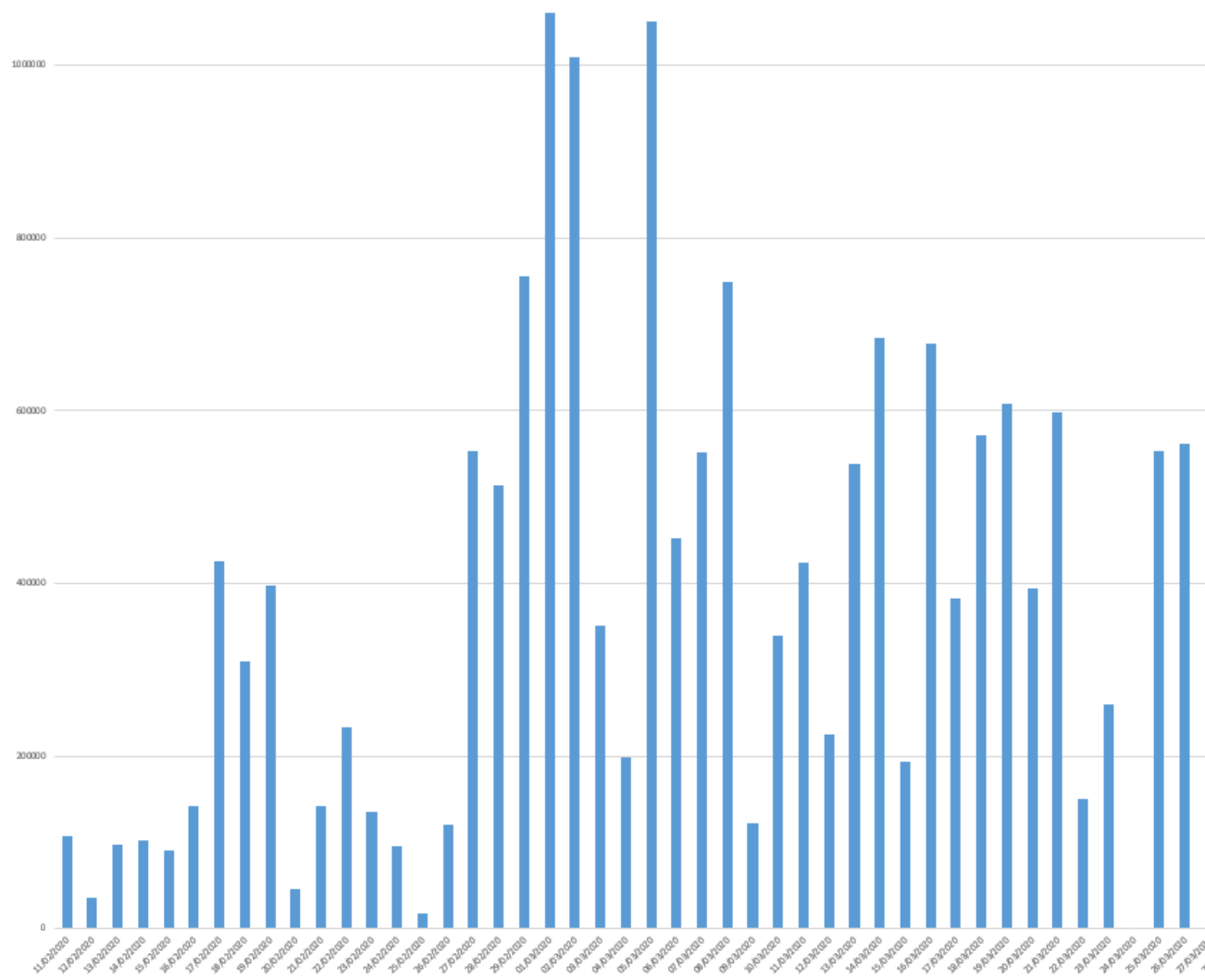


Figure 31: Récupération de tweets/jours entre le 11/02/2020 et le 26/03/2020

Nous pouvons voir que certains jours, plus de 1 000 000 de tweets ont été traités. Grâce à notre normalisation qui prend en compte le nombre de tweets sur un event, le nombre d'utilisateurs distincts ayant tweeté à ce sujet et le nombre total de tweets de la journée, nous pouvons assurer que ces pics de données n'influent que très peu sur les résultats finaux

Nous avons aussi avec ça, la preuve que notre système peut gérer un nombre très élevé de données par jour, bien que tout cela se passe en local sur nos postes. Nous avons fait en sorte de réduire les temps d'exécution de nos différents algorithmes (en passant par exemple par des traitements matriciels plutôt qu'individuels) afin qu'un grand nombre de données ne soit pas un problème. De ce fait, notre système pourrait être lancé sur des machines en temps réel 24h/24. La quantité de données récupérée serait gérable par nos algorithmes, et pourrait même être traitée chaque heure (plutôt que chaque jour) afin de rendre plus fiable nos prédictions.

Chapitre 8 : Gestion de projet

Afin d'arriver au résultat final, nous avons dû gérer notre projet de façon agile. Nous allons voir dans ce chapitre quelle a été notre méthode de gestion, comment les tâches ont été réparties, comment nous avons mis en place nos réunions internes/avec nos tuteurs au cours du projet, ainsi que les différents outils qui nous ont permis de gérer au mieux le projet.

8.1 : Méthode de gestion

Pour gérer ce projet de façon agile, nous avons choisi d'utiliser la méthode de gestion scrum, adaptée au format du projet de synthèse (en autres à cause de l'alternance). Nous avons donc fonctionné non pas par sprint, mais par release selon le schéma suivant :

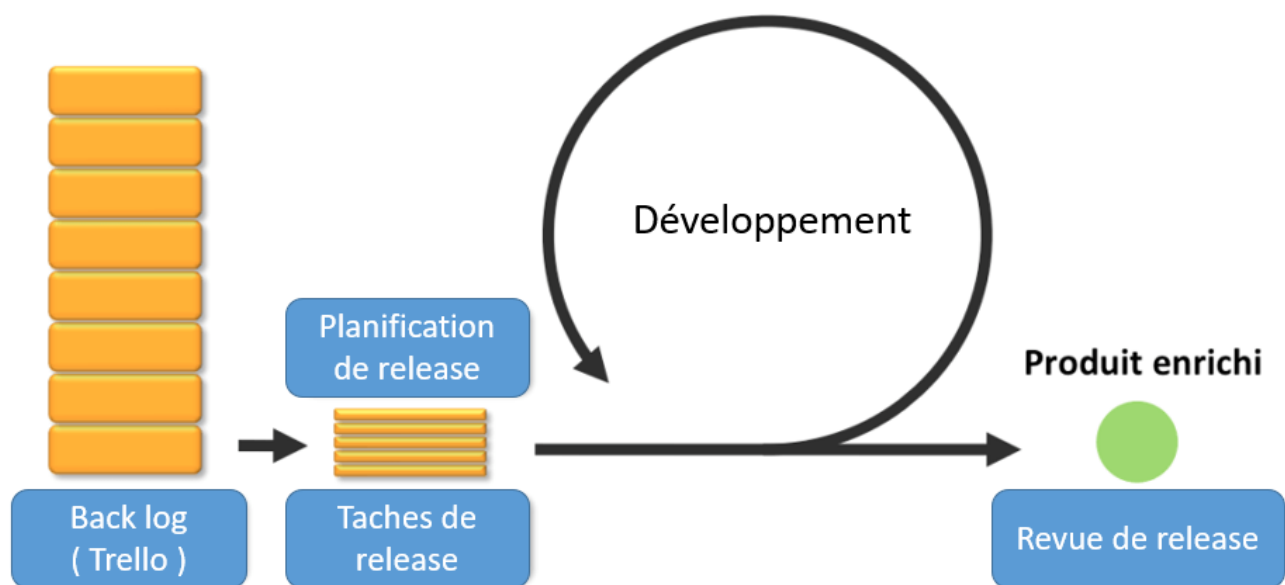


Figure 32: Cycle de développement du projet

Notre Backlog, contenant les différentes stories à développer, a été faite sous Trello, qui est une alternative gratuite mais efficace à Jira. Nous y avons divisé notre tableau Trello en différentes features, et chacune de ses features en différentes user stories, nous reviendrons plus en détail sur cette décomposition dans la partie 8.5 : Décomposition du projet avec l'approche agile : features + user story.

Concernant nos releases, nous en avons eu 2 intermédiaires avant le rendu final, placé arbitrairement les semaines du 9 Mars 2020 et du 24 Avril 2020. Au début de chaque release, nous avons déterminé certaines stories, réparties à travers les différentes features, dans le but de les réaliser durant la release, sans pour autant "fermer" la release à ces stories. En effet, au cours de nos différentes réunions avec nos tuteurs techniques, nous avons ajouté, retiré ou bien modifié certaines stories, et qui ont impacté le cycle de développement de la release.

Release	Date	Fonctionnalités prévus (faites / non faites / presque finis)
Release 1	09/03/2020	Prévoir la liste des étapes de traitement nécessaires Mise en place des BDs Récupération des tweets Traitement de texte Tests sur le traitement de texte Squelette du site web
Release 2	24/04/2020	Clustering sur un ensemble de données fixes Clustering à intervalle de temps avec des données en temps réel Labellisation Premiers affichages web (Statistiques)
Rendu Final	19/06/2020	Voir partie “Rendu final”

Tableau 9 : Division des releases

8.2 : Répartition de tâches

Afin d'avancer le projet dans de bonnes conditions, nous nous sommes réparti les tâches de la manière suivante:

Partie IA:

- NLP : Martin et Mathieu
- Clustering : Martin et Mathieu
- Tests et certifications: Martin

Partie BD:

- Choix des BD: collégiale
- Mise en place de la base MongoDB: Gabriel
- Stockage des tweets: Gabriel
- Schema SQL: Lydia
- Requêtes et stockage SQL: Lydia et Gabriel

Partie Web:

- Application web: Lydia
- Graphiques: Gabriel

Gestion de projet:

- Mise en place des outils: Mathieu
- Gestion des outils (mises à jour de Trello etc.): Mathieu

Cette répartition s'est mise en place en fonction des forces de chacun. En effet, au début les stories étaient dans la backlog et chacun a pu choisir sur quoi s'orienter et travailler en début de projet, selon ses affinités. Par la suite, le fait d'avoir commencé sur une certaine tâche nous a amené à continuer dans le même domaine. Ceci est dû à plusieurs raisons:

- **Une meilleure connaissance des bases.** En effet on peut se représenter les tâches comme un arbre, avec pour racine la première tâche du domaine (le NLP par exemple pour la partie traitement des données). Le fait d'avoir commencé par cette tâche nous pousse à nous documenter un maximum sur le domaine, ce qui nous permet de mieux avancer pour les tâches suivantes.
- **Une "expertise" sur le domaine.** Chaque personne travaillant principalement sur un domaine, en cas de problème lié à une étape précédente du développement, il est plus simple de retourner soit même voir qu'est-ce qui cause le problème, ce qui permet de le régler facilement.
- **Un gain de temps de documentation.** La documentation est probablement ce qui nous a pris le plus de temps sur ce projet. Cette répartition des tâches permet que chacun se documente sur ce qu'il a à faire, et non sur ce qui a été fait par les autres. Ainsi, avec une bonne communication d'équipe, on a pu expliquer (en simplifiant) chaque étape afin que seul 1 à 2 personnes n'aient à se documenter entièrement sur un sujet, et que chacun comprenne au moins le fonctionnement de la partie.

Même si on peut voir une répartition assez hétérogène, chaque membre de l'équipe avait des connaissances sur tout ce qui était fait par les autres membres en permanence, grâce notamment à nos réunions d'équipes.

8.3 : Gestion de réunions et organisation de l'équipe

Au début du projet, nous avons prévu de faire des réunions avec notre tuteur technique toutes les deux semaines environ, mais ceci s'est vu difficile étant donné les disponibilités de notre tuteur. En effet, notre tuteur étant très occupé, nous avons eu un peu de mal à communiquer jusqu'à environ fin Janvier. Nous avons donc dû nous adapter à cette situation en augmentant les réunions interne au groupe afin de nous concerter sur l'avancement.

Pour cela, nous avons mis en place le rituel de scrum "Daily meeting Scrum" tous les jours où on avait des créneaux de projet en autonomie. Ceci nous a permis dans un premier temps d'éviter de nous perdre dans le sujet du projet, en prenant en compte l'avancement de chacun et les différents blocages. De plus, chaque fin de semaine de cours, nous avons organisé des réunions type "revue de sprint" pour revenir sur ce qui a été fait durant la semaine et évaluer notre avancement. Cette situation avec très peu de rendez vous avec notre tuteur ne pouvait pas durer, et grâce à l'arrivée du second semestre, nous avons pu le voir plus régulièrement.

Nous avons donc réussi à nous adapter, et le fait d'avoir deux tuteurs techniques s'est révélé très important, car nous pouvions prendre des rendez vous plus fréquents avec l'un ou l'autre selon leurs disponibilités, ce qui nous a permis d'avancer dans de meilleures conditions. Nous avons donc pu au final avoir au moins une discussion avec l'un de nos deux tuteurs toutes les deux semaines (que ce soit par mail, rendez vous, ou question dans les couloirs de l'université). Avec l'arrivée du confinement, nous avons pu obtenir plus de rendez vous avec nos tuteurs, qui étaient plus accessibles, et nous avons donc pu nous réorganiser, jusqu'à obtenir un rendez vous par semaine à la fin du projet.

8.4 : Utilisation des outils de gestion de projet au sein de l'équipe

Durant le projet, nous avons utilisé différents outils qui ont chacun un but précis, et qui se complètent entre eux. L'objectif d'utiliser ces outils est de nous "simplifier" la gestion du projet et nous assurer de conserver une trace de tout ce qui a été fait.

Discord:

Nous avons mis en place un serveur discord, qui a été l'outil le plus utilisé pour le projet. Il ne s'agit pas d'un outil de gestion de projet, mais d'un outil de communication. Nous l'avons mis en place dès le début du projet, et il nous a permis de communiquer ensemble à tout moment, tout en divisant nos canaux de discussion afin d'organiser nos messages, et de retrouver les informations importantes si besoins. En plus des canaux écrit, nous avons un canal oral qui nous permet de faire nos réunions, ce canal permet aussi les partages d'écran afin de procéder au pair programming à distance. Cet outil est devenu d'autant plus indispensable avec le confinement.

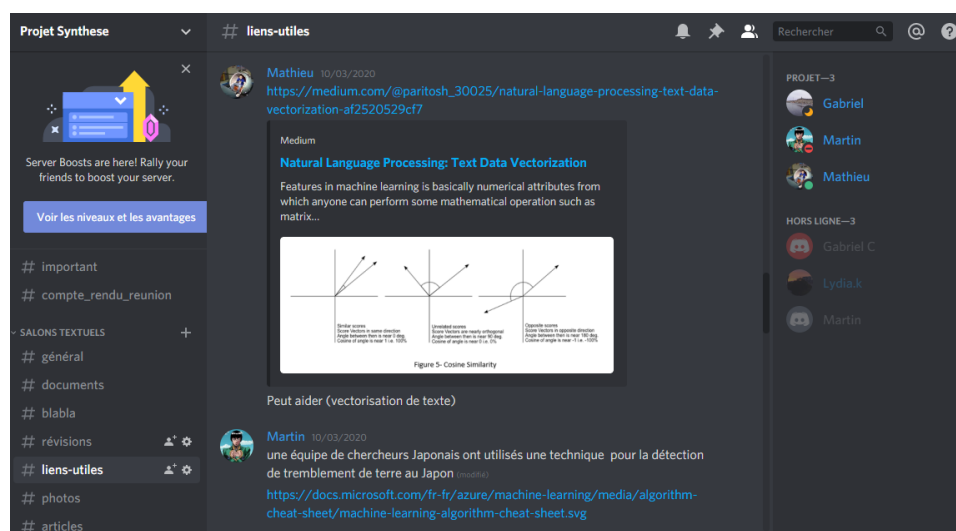


Figure 33: Serveur discord mis en place pour le projet

GitHub :

Afin de partager entre nous les différents fichiers réalisés au cours du projet, nous avons mis en place un git. Nous avons choisi Github car il s'agit de l'outil que nous connaissons le mieux pour faire cela. Nous y avons placé nos fichiers pour le site web ainsi que nos scripts pythons sur ce git, afin que tout le monde ait accès aux fichiers à jour à tout moment.

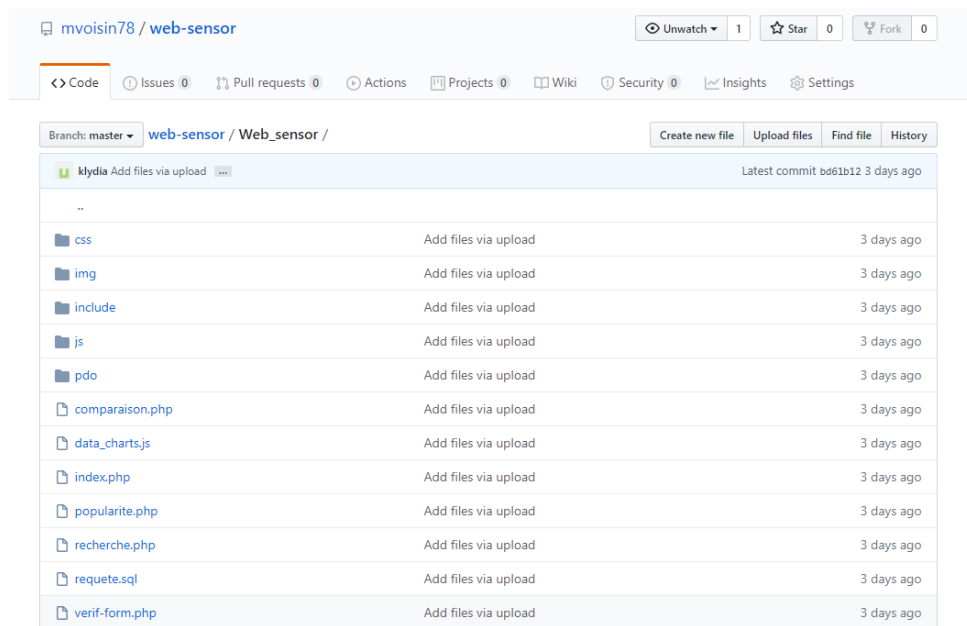


Figure 34: Github

Office 365:

Nous avons utilisé les outils fournis par office 365 pour documenter notre projet. En effet, nous avons utilisé la version en ligne de word afin de rédiger divers documents, tel que le rapport de projet, ou encore des documents que nous avons utilisé en interne afin de rédiger nos idées sur certains points, notamment aux phases de conception du projet. De plus, nous avons utilisé la version en ligne de powerpoint afin de réaliser nos présentations, mais aussi différents schémas qui nous ont été utiles au cours du projet.

Trello:

Afin de gérer nos différentes tâches, nous avons utilisé Trello, une alternative à Jira. C'est un outil moins complet que Jira, mais suffisant dans le cadre du projet de synthèse. Cet outil

est basé sur un système de “liste”. En effet, on peut créer des listes qui ont chacune un thème, dans notre cas une liste correspond principalement à une feature (Nous avons aussi utilisé des listes pour représenter les échéances du projet, ou bien pour tenir un carnet de bord des réunions avec leurs comptes rendus). Dans chaque liste on peut mettre un certain nombre de tâches, généralement des tâches les plus “simples” possibles, qui correspondent aux users stories. Nous allons maintenant voir comment nous avons décomposé notre tableau Trello.

8.5 : Décomposition du projet en features et stories

Dans le cadre de notre gestion agile du projet avec la méthode Scrum, nous avons utilisé l'outil Trello pour mettre au clair nos besoins au fur et à mesure de l'avancement du projet. Notre tableau est décomposé en plusieurs listes, qui correspondent à chacune des features:

- Feature 1: Administration qui correspond aux choix techniques, à la conception du projet et aux besoins fondamentaux (récupération de tweets etc.).
- Feature 2: Registre des événements qui est la feature la plus importante du projet. Elle comprend toutes les étapes de traitement des données qui permettent un affichage.
- Feature 3: Affichage des events qui consiste en la mise en place du site web ainsi que des pages de visualisations d'events
- Feature 4: Statistiques qui comprend tout ce qui est création de graphes, de top d'events etc. qui seront affichés sur les pages dédiés aux events.
- Feature 5: Tests est la feature qui permet de certifier que nos solutions sont des solutions qui fonctionnent bien dans le cadre de notre projet.

Chacune des stories qui composent ces features ont été faites afin d'être réduites au minimum dans l'objectif d'avoir 'une storie = une tâche'. De plus, nous avons ajouté des listes qui ne sont pas des features, elles correspondent à des parts importantes du projet tel que les différentes deadlines et les comptes rendus de réunions.

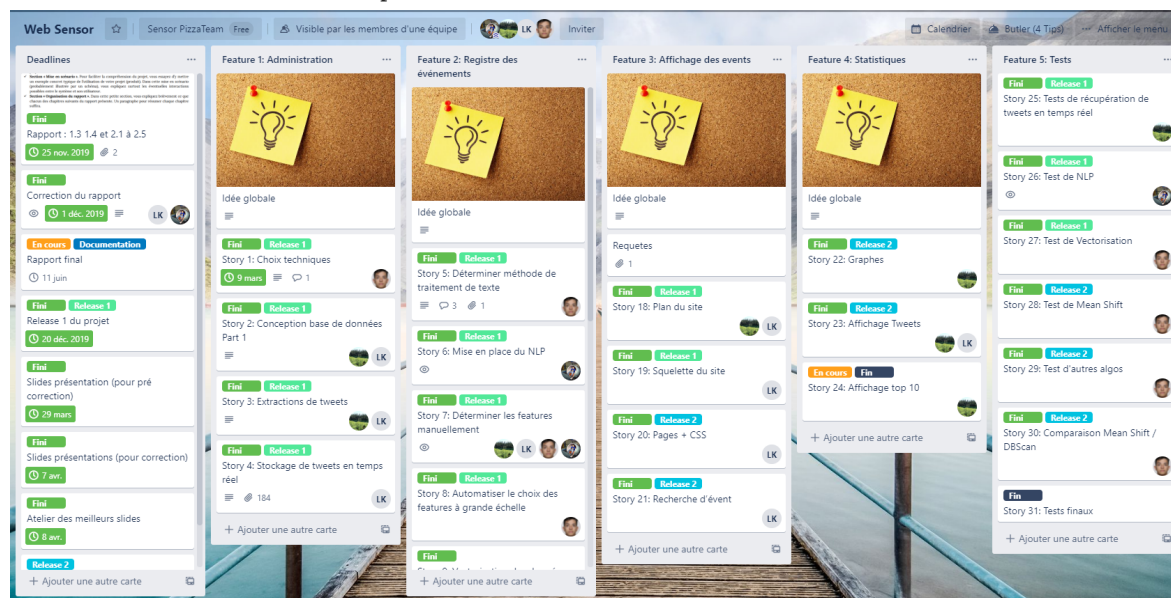


Figure 35: Tableau Trello

Chapitre 9 : Conclusion et perspectives

Dans ce chapitre, nous allons conclure sur notre projet, et nous verrons les différentes perspectives qui s'offrent à nous pour de futures améliorations.

9.1 : Conclusion

Pour conclure, nous avons trouvé ce projet très intéressant et surtout enrichissant sur de nombreux points techniques. En effet, nous avons trouvé ce projet beaucoup plus basé sur la recherche que sur la technique, car nous n'avions pas assez de connaissances dans le domaine pour mener à bien ce projet sans passer par de grosses phases de documentations. Cette recherche de connaissances qui a eu lieu tout au long du projet nous a permis d'améliorer nos compétences de recherche et de documentations sur différents sujets. Ces sujets n'étant pas tous abordés en cours, nous avons par la même occasion eu la chance de pouvoir découvrir certains algorithmes (DBScan par exemple) que nous ne connaissions pas, et qui se révèlent très utiles dans le domaine.

Base de données: Avec nos recherches de bases de données, nous avons pu en apprendre davantage sur les différentes solutions existantes en dehors des BD relationnelles classiques ne nous avons utilisé jusqu'à maintenant. En effet, nous avons pu apprendre à manipuler une base de données orientée document dont nous n'avions pas encore vu l'utilité avant ce projet, et elle s'est avéré très performante et efficace.

Traitement de texte: La compréhension des différentes méthodes utilisées pour le traitement de texte est vraiment un plus pour nous, car cela nous a permis de comprendre comment fonctionnaient (en surface) de nombreux systèmes utilisés chaque jour. Nous avons rencontré certaines difficultés liées à la langue française, qui comporte moins d'outils et qui sont moins performants que la langue anglaise, mais cela nous a permis de plus approfondir le sujet pour aller plus loin.

Clustering: Lors de ce projet, nous avons découvert le clustering. Il s'agit d'une notion que nous n'avons vu que très brièvement en cours d'IA, sans même y mentionner le mot "cluster". Nous avons donc dû, pour cette partie encore, partir de 0 au niveau des connaissances, ce qui nous a mené à nous documenter énormément. Sans aucune connaissances sur le sujet, nous avons pu découvrir les différents types d'algorithmes, les comparer entre eux et déterminer lequel est le meilleur pour nous. Cela s'est donc révélé très intéressant car nous avons pu voir des algorithmes que nous n'aurions peut-être jamais vus si nous savions directement sur quoi nous tourner au début du projet.

Prédiction: Prédire des données futures avec des données passées n'est pas forcément quelque chose de bien compliqué. Cependant de par la nature mal répartie de notre jeu de données, cela nous a posé quelques difficultés. Nous avons donc dû travailler sur différents

points pour essayer de rendre exploitable un maximum nos données afin de pouvoir faire une prédiction sur les jours suivants.

Chaque recherche effectuée dans le cadre de ce projet s'est vue être utile. Même si nous avons pris une grosse quantité de temps pour découvrir et comprendre les différentes étapes de traitement qui étaient nécessaires pour le projet, nous en sommes sorti grandi, car nous avons maintenant un oeil plus critique sur ces algorithmes, leurs points forts et leurs points faibles, leurs cas d'utilisation etc. que nous n'aurions pas eus, du moins pas si poussé, s'il s'agissait de notions uniquement vues en cours.

De plus, grâce à toute cette documentation, nous avons pu comprendre le fonctionnement de beaucoup "d'outils" que nous utilisons tous les jours sans connaître leur fonctionnement. Par exemple:

- Pour le NLP: La traduction automatique de texte. De nombreuses applications utilisent le NLP pour traduire un texte en entrée vers un autre langage, et nous pouvons comprendre maintenant comment se déroule ce traitement de texte plus en détail.
- Pour le clustering: Les tendances twitter. Il s'agit là du point de départ de notre sujet, même si nos tendances ont pour but d'être différentes de celles de twitter, le principe pour déterminer ces informations reste le même et est intéressant à comprendre.
- La prédiction: La météo. Le système de prévision mis en place dans notre projet est semblable, dans le fonctionnement, à celui utilisé pour prédire la météo d'un jour à l'autre.

Chacune de nos recherches nous a donc apporté des connaissances vastes, et non uniquement sur le thème précis que nous avons traité, et nous savons que ces connaissances peuvent donc être appliquées à de nombreux domaines et y sont même indispensables.

Pour certains d'entre nous, nous n'avons jamais fait de projet ensemble lors de notre parcours universitaire, et nous avons pu mettre en place une cohésion d'équipe assez rapidement.

9.2 : Perspectives

Dans le but de faire évoluer le projet, il y a différentes perspectives qui nous ont semblé intéressantes. Il s'agit de différents points qui pourraient être développés par la suite, que ce soit par nous, ou par une personne tierce qui aurait décidé de reprendre le projet.

- Traitement en temps réel

Étant donnée la situation actuelle du covid-19, nous n'avons pas pu obtenir de machine virtuelle fournie par l'université en temps voulu. Nous avons donc dû récupérer les tweets sur nos postes, mais cela n'a pas pu être réalisé 24h/24. Ceci a impliqué différents choix tels que le fait de faire les prédictions sur une période d'une semaine et non d'un jour, qui est donc moins précis et moins pertinent.

- Optimisation du système de prédictions

Comme vu précédemment, notre système est basé autour de nos données qui n'ont pas été récupérés 24h/24. Notre système de prédiction n'est donc pas optimal car les tendances twitter peuvent évoluer énormément d'un jour à l'autre, et donc encore plus d'une semaine à l'autre. La prédiction serait donc bien plus précise dans le cas où les données seraient réparties sur 24h chaque jour.

- Ajout de vues sur les données

Nous avons mis en place un certain nombre de vues sur les données, sous différentes formes (Graphes, tweets, top 10 etc.). Cependant, chacun peut voir une utilité différente à chaque vue, et aimerait peut-être voir ces données d'une façon non mises en place sur le site. Ajouter des vues (différents graphiques, ou bien encore ajouter une image représentative de chaque événement) pourrait être intéressant selon ce que les utilisateurs aimeraient voir apparaître sur le site.

- Ajout d'une carte des tweets

On pourrait par exemple, pour chaque événement, mettre en place une carte du monde, et placer un point à chaque endroit où un utilisateur a parlé de cet événement. On pourrait donc y observer quelles zones parlent plus de telle ou telle tendance. Cela pourrait s'avérer utile lors par exemple d'un tremblement de terre, où on pourrait observer en voyant la carte l'étendue de la zone touchée.

Pour conclure ce rapport, nous dirons que ce type d'application est assez libre, et chacun peut avoir sa vision de ce qui pourrait être ajouté ou non, il n'y a jamais de version finale qui pourrait correspondre aux besoins de chacun.

Bibliographies

- [1] T. Sakaki, M. Okazaki, et Y. Matsuo, "Earthquake shakes twitter users: real-time event detection by social sensors", 2010.
- [2] T. Joachims, "Text categorization with support vector machines", 1998.
- [3] H.Achrekar, A.Gandhe, R.Lazarus, S.Yu et B.Liu, "Predicting Flu Trends using Twitter Data", 2011.
- [4] A. Turing, "Computing Machinery and Intelligence", 1950
- [5] M.Ester, H.Kriegel, J.Sander et X.X, "A Density-Based Algorithm for Discovering Clustering Large Spatial Databases with Noise", 1996.
- [6] Peter J. ROUSSEEUW, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis ", 1987.
- [7] Maria Halkidi, Yannis Batistakis et Michalis Vazirgiannis, "Cluster Validity Methods : Part I", 2002.
- [8] Dickey, D. A. Fuller, W. A. , "Distribution of the Estimators for Autoregressive Time Series with a Unit Root", 1979.
- [9] Asteriou Dimitros, Hall Stephen G. "ARIMA Models and the Box-Jenkins Methodology", 2011