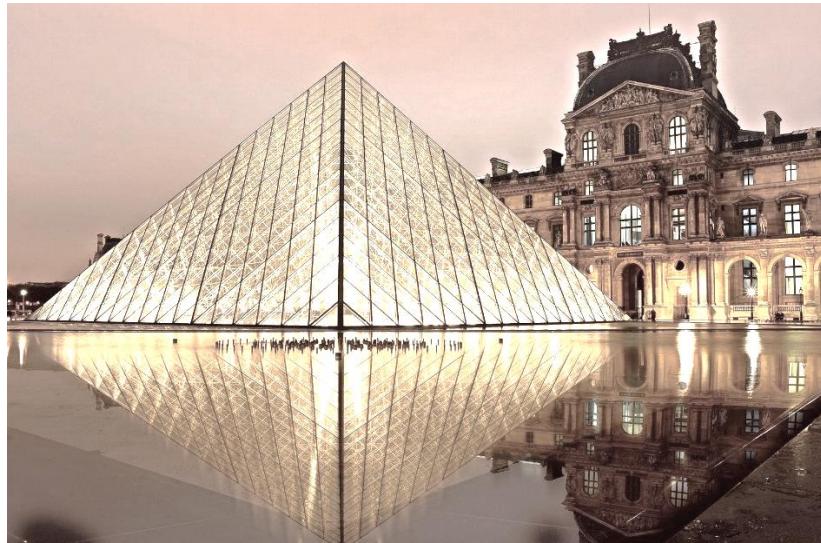


Analyse de Twitter en temps réel avec Kafka, Spark et mongoDB



Nom de l'UE	UASB03	Version	1.0
Auteur	Jean-christophe Feraudet	Date de mise à jour	21 Février 2020
Destinataires	Ndeye NIANG KEITA Michel CRUCIANU Philippe RIGAUX	Référence	UASB03_Projet_Feraudet_v1

Table des matières

Introduction	1
1. Description des étapes du projet	1
Etape N°1: Collecte et exploration des données	2
Etape N°2: Préparation des données textuelles	8
Etape N°3: Création des jeux de données	10
Etape N°4: Choix du modèle d'analyse	10
Etape N°5: Entrainement des modèles	13
Etape N°6: Prédiction et évaluations des Résultats	15
Etape N°7: Visualisation des résultats	18
2. Architecture du Projet	20
2.1 Architecture Technique	20
2.1.1 Architecture générale	20
2.1.2 Liste des progiciels	21
2.1.3 Liste des traitements	22
2.2 Passage à l'échelle	23
2.2.1 Passage à l'échelle de Kafka	23
2.2.2 Passage à l'échelle de Spark	25
2.2.3 Passage à l'échelle de MongoDB	28
Conclusion	30
Annexes	31
Annexe 1 : Liste des figures	31
Annexe 2 : Liste des tableaux	32
Annexe 3 : Liste des extraits de code	32
Annexe 4 : Contenu du fichier archive du projet	33
Annexe 5: Test de charge pour l'analyse en temps réel des tweets	34
Annexe 6: Copies d'écrans sur l'analyse en temps réel	36
Annexe 7: Exemple d'Analyse de Sentiments sur 200.000 tweets	38
Annexe 8: Comparaison des modèles sur la base d'apprentissage	40
Annexe 9: Ordonnancement des traitements	42
Annexe 10: Copies d'écrans du site Web	43

Introduction

Avec le développement d'internet et la mise à disposition d'un grand nombre de données textuelles, sont apparus de nouveaux besoins: l'analyse en temps réel des flux d'information et l'analyse automatisée des sentiments pour comprendre la polarité (positive, neutre ou négative) des avis exprimés dans ces textes.

Mon projet est une exploration en temps réel des données massives issues du réseau social Twitter, complétée d'une analyse statistique du contenu des tweets. Les données analysées sont des tweets portant sur des sorties culturelles à Paris telles que les musées, les expositions ou les spectacles.

Les objectifs de cette étude sont les suivants :

- Répertorier les activités culturelles les plus partagées sur Twitter avec une visualisation en « temps réel » des sujets échangés (#hashtags) sur chacune de ces activités.
- Procéder à une analyse de sentiments du contenu des échanges afin d'évaluer le niveau de satisfaction des visiteurs. Les tweets seront ainsi classés automatiquement en trois catégories (tweets négatifs, neutres et positifs) en se basant sur un jeu de données d'apprentissage composé de 3000 tweets déjà annotés.
- Vérifier le passage à l'échelle pour l'analyse et la fouille de ces données avec l'utilisation d'un jeu de données important de plus de 230.000 tweets collectés au cours des 6 derniers mois.

Ce document présente tout d'abord les grandes étapes de ce projet et les résultats obtenus de l'analyse en temps réel des tweets ainsi que de l'analyse de sentiments sur l'ensemble des données d'apprentissages et de test. La seconde partie décrit l'architecture technique mise en place pour un passage à l'échelle des traitements et des données. Enfin, je conclurai ce document par une synthèse des points abordés au cours de ce projet et par un bilan plus personnel de ce que m'a apporté cet enseignement.

1. Description des étapes du projet

Le schéma suivant présente les étapes du projet qui sont détaillées dans la suite de ce document. Un exemple d'exécution de chaque étape avec le code qui lui est rattaché est disponible sous la forme de Notebooks jupyter dont le contenu est décrit en annexe 4.

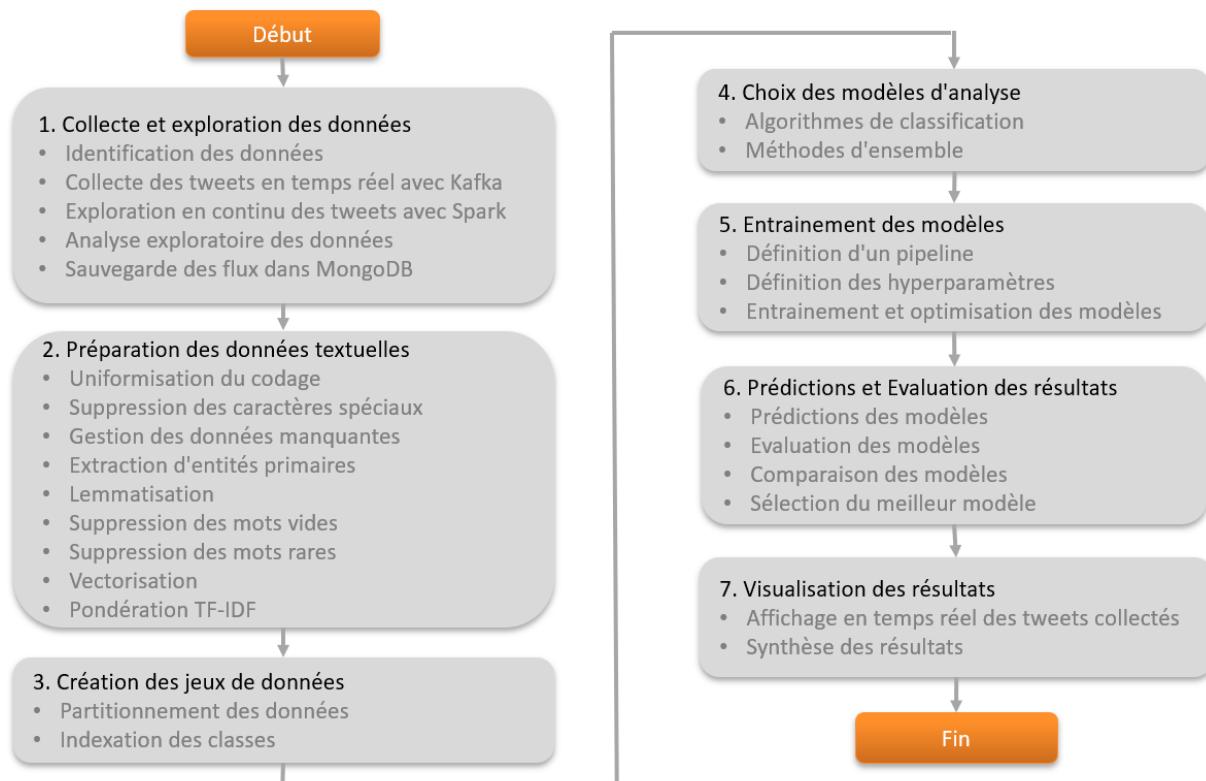


Figure 1: Liste des étapes du projet

Etape N°1: Collecte et exploration des données

1.1 Identification des données

Les données textuelles utilisées pour ce projet sont des tweets qui sont collectés en temps réel depuis Twitter et qui portent sur les sorties culturelles suivantes à Paris :

- Musées (ex : Musée du louvre)
- Monuments (ex : Tour Eiffel)
- Expositions (ex : exposition Léonard de Vinci au Louvre)
- Théâtres (ex : Comédie Française)
- Attractions (ex : Disneyland Paris)

Chaque Tweet comporte de nombreux attributs qui décrivent le contexte dans lequel le tweet a été publié (ex : géolocalisation du tweet, profil de l'auteur du tweet, nombre de followers, etc..). Pour ce projet, j'ai sélectionné les attributs de base d'un tweet qui sont les suivants :

- Id : Identifiant du tweet
- User : Auteur du tweet
- Created_at : Date de création
- Lang : Langue utilisée dans le tweet
- Full_Text : Message du tweet (contenu textuel)
- Hashtags : Liste des mots-clés se rapportant au sujet du tweet
- Media : Photos référencées dans le tweet
- Retweeted_status : Indicateur que ce tweet est un tweet Retweeté
- Possibly_sensitive : Indicateur que des données sensibles sont peut-être dans le tweet

La variable à prédire est le sentiment rattaché au tweet : négatif, neutre ou positif

- Ex tweet Négatif: « *Il y a vraiment trop de monde. Voir l'exposition relève du miracle* »
- Ex.tweet Neutre: « *Exposition Nous les Arbres à la Fondation Cartier du 12 juillet au 10 novembre* »
- Ex.tweet Positif : « *beaucoup de très belles oeuvres, une mise en scène intelligente. Un vrai régal* »

La sélection des tweets à analyser est réalisée au moyen de filtres basés sur des mots-clés (#hashtags). Ces mots-clés permettent d'identifier précisément la thématique supposée du tweet et peuvent être récupérés sur les sites internet dédiés à la vie culturelle de Paris (ex : site officiel de l'office du tourisme de Paris « Paris Info », « Journal des arts », « Sortir à Paris », etc...). Afin d'obtenir une base de données conséquente pour la fouille de données, les tweets ont été historisés pendant les 6 derniers mois, ce qui représente un total de 230 000 tweets à analyser.

Pour les données d'apprentissage, le choix initial de la base de données « sentiment140 » composée de nombreux tweets déjà annotés de l'université de Stanford s'est avéré non efficient. En effet, elle ne contient pas d'exemples d'apprentissage pour les tweets neutres qui sont majoritaires dans le cadre des sorties culturelles à Paris. J'ai donc choisi de créer une base d'apprentissage composée de 3000 tweets que j'ai annotés manuellement et qui ont été extraits au hasard depuis la base des tweets collectés précédemment.

1.2 Collecte des tweets en temps réel avec Kafka

La collecte des tweets est réalisée en temps réel au moyen des librairies python « kafka » et « Tweepy ». Cette dernière librairie permet de communiquer facilement avec Twitter et de recueillir les tweets en streaming au format JSON. Ce flux est ensuite transféré dans Kafka qui est un système distribué de messagerie par abonnement. La publication et la souscription du flux des tweets s'effectuent au travers d'un topic kafka. Il permet ainsi de rassembler et stocker tous les tweets pour ce projet dans une même catégorie. Une description détaillée de son architecture et le passage à l'échelle sont présentés au chapitre suivant.

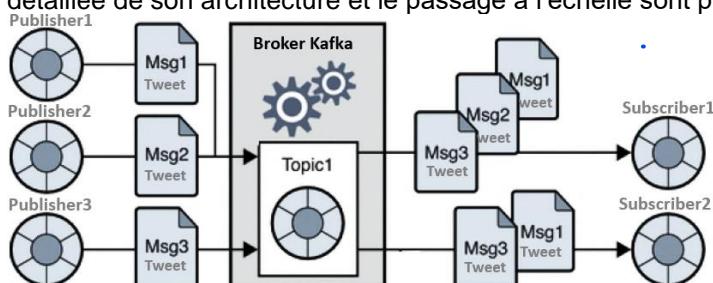


Figure 2: Système de messagerie Kafka

Le code python suivant montre comment le flux des tweets est créé et redirigé vers le topic Kafka « tweets01 » au sein d'un cluster composé des deux serveurs kafka01 et kafka02 :

```

#
# Code pour le Streaming de tweets avec Kafka
#
from tweepy.streaming import StreamListener, OAuthHandler, Stream
from kafka import KafkaProducer
# Encodage utf-8 des tweets
producer = KafkaProducer(
    value_serializer=lambda m: dumps(m).encode('utf-8'),
    bootstrap_servers=['kafka01:9092', 'kafka02:9092'])
# Envoi des données de streaming sur le topic kafka « tweets01 »
class StdOutListener(StreamListener):
    def on_data(self, data):
        producer.send("tweets01", value=data)
        date = datetime.now()
        print(date,"New Tweet ! (Len:",len(data),")")
        return True
# Démarrage du Streaming avec Tweepy et Kafka
listener = StdOutListener()
auth = OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
twitter_stream = Stream(auth, listener)
twitter_stream.filter(follow=list_follow,track=list_track,languages = ['en','fr'])

```

Code 1: Extrait du programme Twitter-03-Streaming-KafkaProducer-Loop.ipnbj

1.3 Exploration en continu des tweets avec Spark

Apache Spark est un framework open source de calcul distribué qui permet de réaliser des analyses complexes à grande échelle et dont son fonctionnement est détaillé au chapitre suivant. Spark est utilisé ici pour son moteur de **streaming** qui est une extension de l'API principale de Spark. Il permet le traitement rapide, scalable et tolérant aux pannes des flux de données extraits de Kafka. Ainsi, lorsque Spark streaming reçoit les données de Kafka, il les divise en mini batch RDDs qui sont à leur tour traités par le « Spark Engine » pour générer en sortie des résultats sous la forme de batchs :



Figure 3: Spark Streaming

Pour ce projet, **Spark Streaming** est utilisé pour extraire en continu les sujets de discussion (#hashtags) les plus fréquents dans les tweets collectés sur une période donnée. J'ai utilisé la fonction Spark de fenêtrage « `reduceByKeyAndWindow` » pour analyser les tweets sur une période de temps qui est paramétrable (ex: 20 secondes) et indépendante de la fréquence de lancement des mini batchs (ex: 5 secondes):

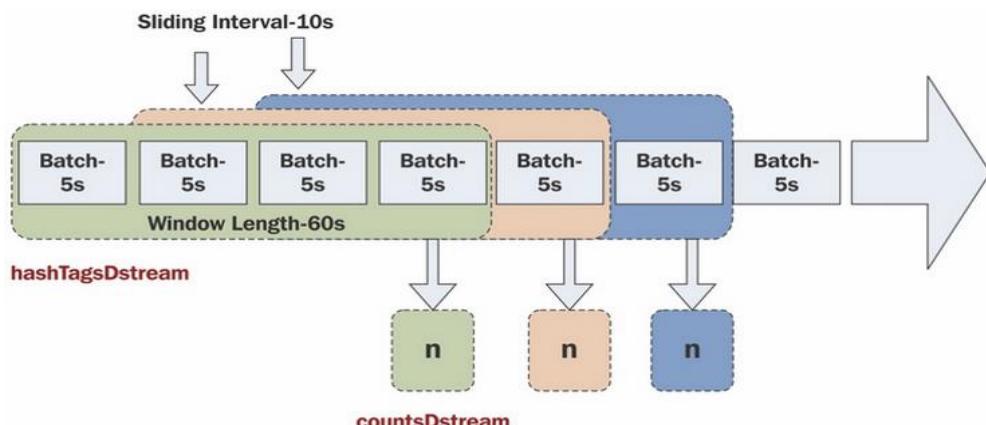


Figure 4: Opérations de fenêtrage avec Spark Streaming

Le code de la page suivante montre comment **Spark Streaming** se connecte au topic Kafka pour collecter en continu les tweets. Les mots-clés (#hashtags) sont ensuite extraits des tweets, puis agrégés et triés en fonction du nombre d'occurrences trouvé. Le résultat de l'analyse est imprimé puis sauvegardé pour un affichage futur.

```

# Code pour l'analyse en continu des tweets avec Spark Streaming
from pyspark.streaming.kafka import KafkaUtils
# Configuration de Spark Streaming toutes les 5 secondes
sc = SparkContext(conf=conf)
stream_nbseconds=5
ssc = StreamingContext(sc, stream_nbseconds)
# Connexion au topic Kafka dédié au streaming des tweets
topic="tweets01"
kafkaStream = KafkaUtils.createDirectStream(ssc,[topic],{"metadata.broker.list": "kafka01:9092,kafka02:9092"})
parsed = kafkaStream.map(lambda v: json.loads(v[1]))
# Analyse en continu des hashtags dans les tweets
words = parsed.flatMap(lambda line: line.replace("'",' ').replace('"',' ').replace
(",",' ').replace(")",' ').replace("\\",' ').replace(".",' ').split())
hashtags=words.filter(lambda w: re.findall(r'\B#\w*[a-zA-Z]+\w*',w)).map(lambda x:
(x, 1))
# fenêtre d'analyse des tweets toutes les 20 secondes
stream_window_nbseconds=20
stream_slide_nbseconds=20
lambda x, y: x - y, stream_window_nbseconds, stream_slide_nbseconds)
hashtags_reduced = hashtags.reduceByKeyAndWindow(lambda x, y: x + y,
hashtags_sorted = hashtags_reduced.transform(lambda w: w.sortBy(lambda x: x[1],
ascending=False))
# Affichage et sauvegarde des résultats
hashtags_sorted pprint(5)
hashtags_sorted.foreachRDD(lambda rdd: rdd.foreach(insertHashtags))
# démarrage de la boucle de traitement des tweets
ssc.start()

```

Code 2: Extrait du programme Twitter-04-Streaming-SparkConsumer-loop.ipnbj

Le résultat est affiché sous la forme d'un nuage de mots qui change au fur et à mesure de l'analyse de nouveaux tweets. J'ai utilisé pour cette partie la librairie python « WordCloud » qui permet d'afficher une liste des mots-clés les plus utilisés sous la forme d'un nuage de mots.

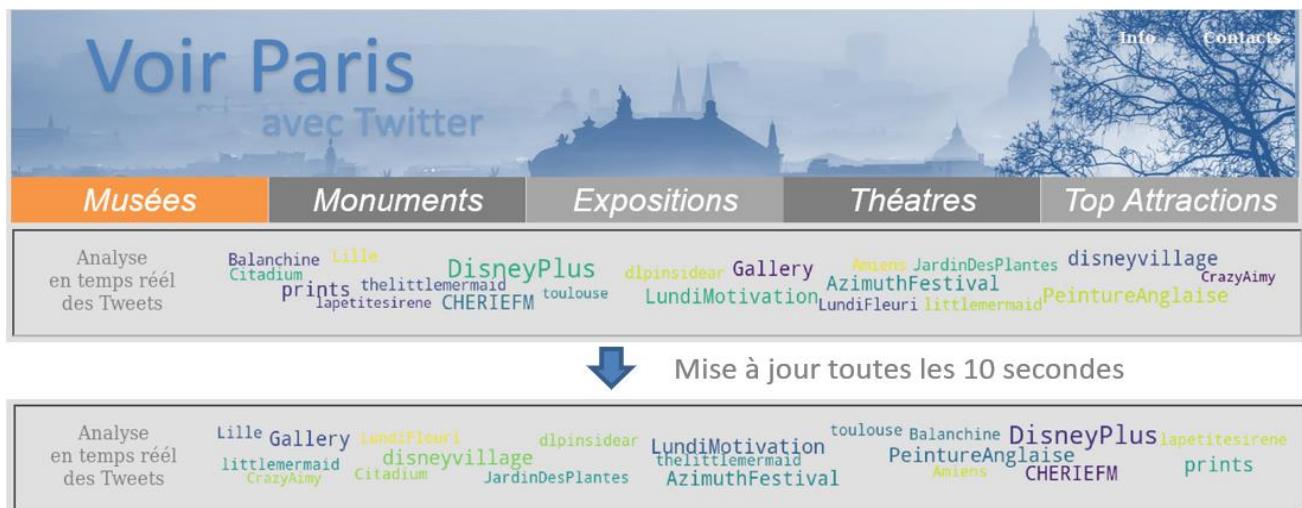


Figure 5: Affichage en continu des sujets les plus discutés sur Twitter

Note :

Le rafraîchissement automatique de la page Web a été réalisé au sein d'un « iframe » qui permet d'isoler cet affichage du reste d'une page Web et en utilisant la fonction html « refresh » :

```

<HEAD>
    <META HTTP-EQUIV="Content-Type" CONTENT="text/html;CHARSET=iso-8859-1">
    <META http-equiv="refresh" content="10" />
    <TITLE>Spark Streaming</TITLE>
</HEAD>

```

Code 3 : Extrait du code source pour l'affichage du nuage de mots (bagofwords.html)

1.4 Analyse exploratoire des données

L'analyse exploratoire des données a pour objet de préparer les données pour répondre au mieux aux contraintes imposées par les différents modèles de data-mining sur les données (normalité, linéarité,...).

Pour ce projet, l'analyse exploratoire des données reste limitée, compte tenu du faible nombre de caractéristiques. Ce sont principalement les mots-clés caractérisant les tweets (variable « hashtags ») pour une exploration en temps réel des tweets collectés et le contenu textuel des tweets (variable « full_text ») pour l'analyse de sentiments.

1.4.1 Données d'apprentissage

Répartition des données

La prise en compte de classes déséquilibrées par les modèles de data-mining est souvent problématique. J'ai donc procédé à une répartition équilibrée des trois classes de sortie dans la base d'apprentissage comme le montre l'histogramme suivant :

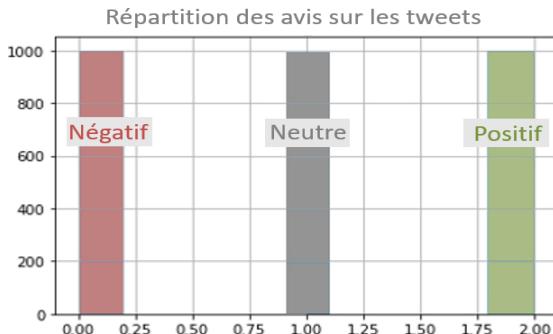


Figure 6: Répartition des avis dans la base d'apprentissage

Note :

Twitter est par nature très bruité avec de nombreux tweets qui sont dupliqués. Je n'ai retenu pour la constitution des données d'apprentissage que les tweets originels. J'ai donc supprimé tous les tweets retweetés pour limiter les effets d'emballage sur un sujet particulier comme celui des grèves de décembre dernier.

Fréquence des mots

Il est intéressant de visualiser les mots composant le jeu de données qui sera analysé par les algorithmes d'apprentissage machine. Les « nuages de mots » suivants montrent que les mots exprimant des sentiments sont relativement fréquents dans les tweets (ex : mauvais, terrible, hommage,..) et qu'ils sont différents en fonction du message que l'on souhaite faire passer dans les tweets :



Figure 7 : mots les plus fréquents dans les tweets négatifs (en rouge), neutres (en gris) et positifs (en vert)

Ceci permet de confirmer qu'une analyse des sentiments est possible à travers une analyse statistique des mots employés par les internautes pour classer les tweets en trois catégories (négatif, neutre ou positif).

Longueur des tweets

La longueur maximale d'un message posté sur Twitter est de 280 caractères. Pour ce corpus, la taille des tweets est plus réduite et se situe autour de 175 caractères pour les textes bruts et de 100 caractères après nettoyage des données. Cette longueur est très courte contrairement à celles utilisées dans d'autres corpus pour la classification de sentiments (comme les critiques de films).

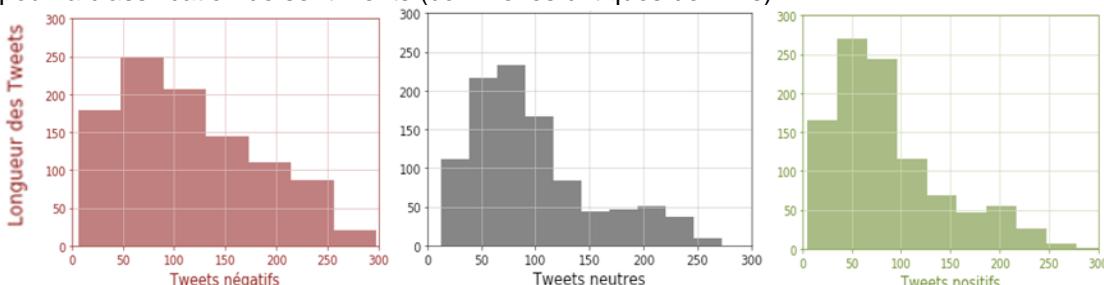


Figure 8: Histogramme des longueurs des tweets selon le sentiment exprimé

1.4.2 Données de test

Des analyses statistiques de base ont été réalisées pour chacune des activités culturelles sélectionnées pour ce projet. Ceci afin de vérifier si les données recueillies correspondent ou non à une réalité sur le terrain et sont donc pertinentes ou non pour ce projet .

Volume des tweets collectés

Une comparaison a été réalisée pour vérifier si le nombre de tweets collectés dans le jeu de données est un bon indicateur ou non de la fréquentation réelle et de l'intérêt porté par le public sur les activités culturelles à Paris. Les deux exemples suivants montrent que le jeu de données de test est bien corrélé à la réalité terrain.

Exemple N°1: Classement des activités culturelles selon le nombre de tweets collectés

Le tableau suivant présente le classement des activités culturelles à Paris selon leur thématique (Musées, Monuments, Expositions, Théâtres et Attractions) triées par rapport au nombre de tweets collectés.

Table 1: Classement des principales activités culturelles par thématique

Classement	Musées	Monuments	Expositions	Théâtres	Attractions
1	Louvre Visiteurs : 10.200.000 Tweets : 4405	Tour Eiffel Visiteurs : 6.000.000 Tweets : 7736	FIAC Visiteurs : 75.000 Tweets : 891	Chatelet Places : 2010 Tweets : 2980	DisneyLand Visiteurs : 9.800.000 Tweets : 21980
2	Orsay Entrées : 3.300.000 Tweets : 3958	Notre Dame Visiteurs : 12.000.000 Tweets : 2563	Léonard Vinci Visiteurs : +260.000 Tweets : 643	Opéra Comique Visiteurs : 1100 Tweets : 1763	Versailles Visiteurs : 6.900.000 Tweets : 3125
3	Histoire Naturelle Entrées : 2.000.000 Tweets : 3330	Opéra garnier Visiteurs : 400.000 Tweets : 1812	Tolkien Visiteurs : -100.000 Tweets : 603	Montparnasse Visiteurs : 915 Tweets : 1718	Arena Paris Visiteurs : 1.000.000 Tweets : 3059

On peut voir qu'il y a bien une corrélation entre le nombre de tweets collectés et la fréquentation réelle d'un site touristique. Ainsi, le musée du Louvre avec plus de 10.000.000 d'entrées par an, la Tour Eiffel (6.000.000 de visiteurs), le théâtre du Châtelet (plus grand théâtre de Paris avec 2000 places) et le parc DisneyLand Paris (10.000.000 de visiteurs) sont premiers dans leurs catégories respectives et recueillent le plus grand nombre de tweets (le classement détaillé est donné en annexe 10).

Note :

Il existe cependant quelques exceptions (en orange). L'exception la plus notable est la FIAC (plus grande foire internationale d'art contemporain de Paris, 75000 visiteurs sur 5 jours) qui surclasse en nombre de tweets l'exposition Léonard de Vinci (+260.000 visiteurs). On peut expliquer cette singularité par la notoriété internationale de la FIAC, son activité de marché et ses expositions « Hors les murs » sur la voie publique pour attirer l'attention médiatique (ex : Citrouille géante de Yayoi Kusama sur la place Vendôme) .

Exemple N°2: Evolution du nombre de tweets d'une activité culturelle en fonction du temps

L'exemple suivant montre l'évolution du nombre de tweets collectés pour l'exposition dédiée à Léonard de Vinci au musée du Louvre. On peut voir l'engouement du public au démarrage de cette exposition le 24 octobre dernier, avec un pic le dimanche qui a suivi son démarrage (jour traditionnel de visite) et qui s'est atténué au fur et à mesure du temps. On peut noter aussi un pic de retweets pendant cette même période.

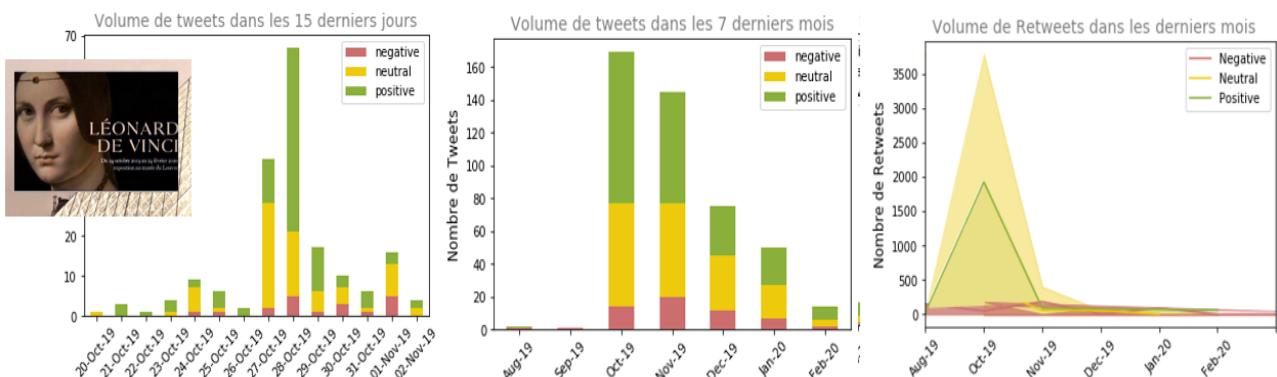


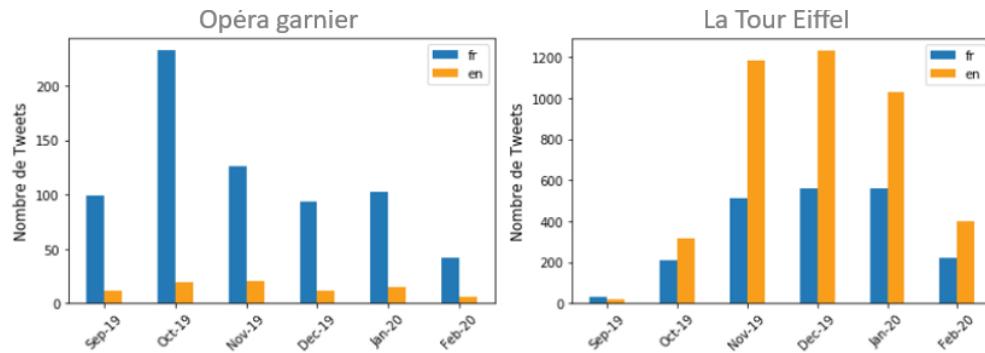
Figure 9: Volume de tweets échangés au démarrage de l'exposition de Léonard de Vinci

Note :

Ces données ont été croisées avec l'analyse de sentiments qui a été réalisée par la suite. On note que le pic de tweets le dimanche 27 octobre s'est accompagné d'un pic de tweets positifs qui confirme les avis favorables entendus dans les médias traditionnels sur cette exposition qui célèbre les 500 ans de la mort de Léonard de Vinci .

Répartition du nombre de tweets selon la langue.

La proportion de tweets en langue anglaise par rapport aux tweets en langue française est un autre indicateur qui permet de vérifier la renommée internationale d'un lieu touristique. Les deux exemples suivants montrent la proportion de tweets en anglais échangés sur la tour Eiffel et l'opéra Garnier.



Ces graphiques correspondent bien à une réalité de terrain. En effet, la tour Eiffel est un symbole incontournable de la France et un lieu à visiter pour tous les touristes séjournant à Paris. A l'opposé, l'opéra Garnier, bien qu'un autre symbole de la capitale, reste un lieu élitiste et peu visité par les touristes étrangers qui n'y rentrent pas.

Longueur des tweets

Les graphiques suivants montrent des exemples d'évolution des longueurs des tweets pour différentes activités culturelles. On remarque que la longueur des tweets est d'environ 175 caractères avec une particularité pour les tweets neutres qui sont généralement plus courts que les tweets négatifs et positifs. Ces résultats sont conformes avec les histogrammes calculés sur les données d'apprentissage :

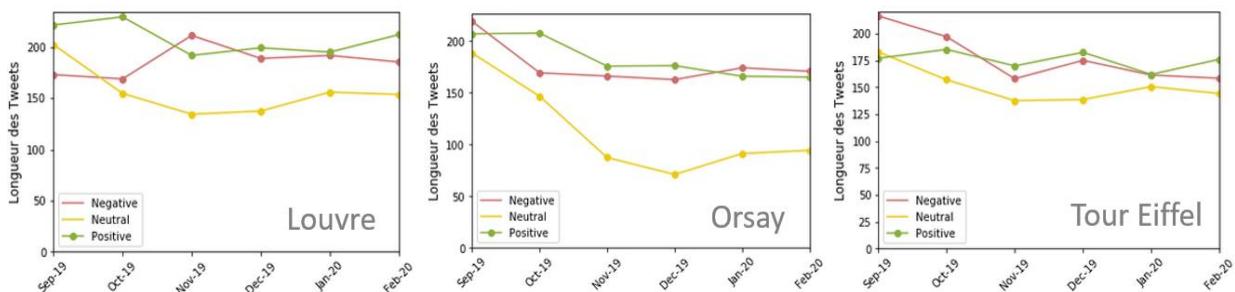


Figure 11: Exemples d'évolution des longueurs des tweets

1.5 Sauvegarde des flux de données dans MongoDB

Les flux de tweets traités par Spark Streaming sont archivés dans une base NOSQL à l'aide de la librairie "Pymongo" et de la fonction Spark Streaming "foreachRDD" qui permet de pousser les données de chaque RDD vers un système externe. Le choix s'est porté sur MongoDB pour les raisons suivantes :

- MongoDB est une base documentaire qui gère les données au format JSON. Il est donc particulièrement adapté pour le stockage des tweets qui sont au format JSON.
- MongoDB possède un mode distribué qui permet de répartir le stockage des tweets et ainsi répondre à la problématique du passage à l'échelle. En effet, le nombre de tweets à gérer peut devenir extrêmement grand selon les recherches effectuées (pour rappel, 500 millions de tweets sont générés chaque jour).
- MongoDB gère la réplication et les reprises sur panne. Ce point est particulièrement important lorsque de nombreux serveurs sont utilisés pour le stockage et le traitement de données massives.
- MongoDB possède de nombreux connecteurs comme "Pyspark" pour le connecter à un cluster Spark

Le code python suivant montre la sauvegarde des tweets collectés dans une collection MongoDB avec la fonction « foreachRDD » :

```
#  
# Code pour la sauvegarde en temps réel des tweets collectés  
#  
# Archivage des tweets dans une collection mongodb  
def insertTweet(rdd):  
    connection      = MongoClient('mongos01', 27017) # client  
    test_db         = connection.get_database('parisdb') # db  
    stream_tweets  = test_db.get_collection('stream_tweets')  
    stream_tweets.insert({"tweet": rdd, "status":'new'})  
    connection.close()  
# Boucle de sauvegarde des tweets reçus  
parsed = kafkaStream.map(lambda v: json.loads(v[1]))  
parsed.foreachRDD(lambda rdd: rdd.foreach(insertTweet))
```

Code 4: Extrait du programme Twitter-04-Streaming-SparkConsumer-loop.ipnbj

Etape N°2: Préparation des données textuelles

Une étape essentielle dans la fouille des données textuelles est la préparation des données collectées afin qu'elles puissent être efficacement traitées par les algorithmes d'apprentissage machine. Cette préparation consiste en un certain nombre de prétraitements qui visent à standardiser et nettoyer le contenu des tweets qui sont archivés :

- Uniformisation du codage
- Suppression des caractères spéciaux
- Gestion des données manquantes
- Extraction d'entités primaires
- Suppression des mots vides de sens
- Suppression des mots rares
- Lemmatisation
- vectorisation
- Pondération TF-IDF

2.1 Uniformisation du codage

Les tweets contiennent une diversité de codage, en particulier au niveau de l'utilisation des minuscules et des majuscules. Ainsi le mot « musée » est souvent écrit en minuscules ou avec une seule majuscule « Musée ». Il peut cependant apparaître tout en majuscules comme dans « LA NUIT DES MUSEES ». L'uniformisation a consisté à passer le texte en minuscules pour pouvoir regrouper tous les mots identiques, quel que soit leur codage. J'ai utilisé pour cette opération d'uniformisation la fonction python « lower() ».

Note :

Le texte analysé étant en français, je n'ai pas supprimé les accents afin que les étapes suivantes comme la lemmatisation puissent s'exécuter correctement.

2.2 Suppression des caractères spéciaux

Les textes des tweets contiennent de nombreux caractères spéciaux dont certains sont liés à du code html (ex:
), à des caractères de ponctuation (ex : !?..) ou à des émoticônes (ex :-)) = 😊). J'ai supprimé grâce à la librairie python « RE » l'ensemble des caractères liés à du code html ainsi que tous les chiffres et les caractères de ponctuation suivants :

- ([:#@%/\$()~_?+=\.&]#!?*)

J'ai par contre gardé tous les émoticônes inclus dans les tweets car il peuvent donner pour certains une indication sur les sentiments liés au contenu du tweet. Ainsi l'émoticône « clappinghands » se retrouve principalement dans les tweets positifs :

- Ex de tweet : « Une belle année de création, d'enrichissement et de progrès 🙌 »

2.3 Gestion des données manquantes

Les données manquantes sont un problème très répandu dans le data-mining. On trouve ce problème dans les tweets collectés dans lesquels le texte peut être vide. J'ai donc procédé à leur suppression grâce à la fonction spark « filter("xxx is not null") » qui permet de filtrer les tweets vides.

2.4 Extraction d'entités primaires

Le but de ce prétraitement est d'extraire d'un texte brut les différents mots le composant afin de pouvoir les analyser séparément. Cette étape est un préalable pour de nombreux traitements comme la lemmatisation. J'ai utilisé la fonction Spark « Tokenizer » pour cette étape particulière.

2.5 Lemmatisation

La lemmatisation consiste à remplacer chaque mot (ex : reçoit) par sa forme canonique (recevoir). Cette étape est utile pour la classification des textes car elle permet de traiter comme un mot unique les différentes variantes issues d'une même forme canonique. J'ai utilisé la librairie python « spacy » qui intègre cette fonctionnalité pour la langue française.

2.6 Suppression des mots vides de sens

Le but de ce prétraitement est de supprimer les mots qui n'ont pas de signification particulière comme des conjonctions (ex : et, que , ...) et qui peuvent être ignorés dans l'analyse de sentiments. J'ai utilisé la fonction Spark « StopWordsRemover » pour supprimer ces mots vides de sens (stopWords) ainsi que la librairie « nltk » qui fournit par défaut une liste de mots en français pouvant être supprimés. Cette liste a été complétée par une liste de mots qui apparaissaient plus de 1000 fois au sein des 200.000 tweets à analyser et qui n'ont n'ont pas de sens particulier dans l'analyse de sentiments.

2.7 Suppression des mots rares

Le but de ce prétraitement est de supprimer des mots qui sont très rarement employés dans les tweets. Pour ce projet, tous les mots qui n'apparaissent qu'une seule fois ont été supprimés car ils rendent moins efficaces les algorithmes utilisés pour l'analyse de sentiments.

2.8 Vectorisation

La représentation vectorielle des données vise à transformer l'ensemble des textes des tweets sous la forme de vecteurs afin qu'ils puissent être traités par les méthodes de fouille conçues pour des données vectorielles.

On utilise l'API HashingTF qui permet de construire des vecteurs creux de longueur fixe reflétant l'importance des mots dans chaque tweet en calculant la fréquence à laquelle ces mots apparaissent. Ainsi, plus un terme est fréquent dans un même document, plus il a de chances d'être révélateur de l'information contenue dans celui-ci (sauf si c'est un mot vide de sens). L'API HashingTF incorpore une notion de « hashing », qui est utilisé pour passer d'une chaîne de caractères à une valeur numérique. Elle sera utilisée comme indice pour la construction de la matrice d'appartenance des termes à un document.

2.9 Pondération TF-IDF

La méthode de pondération « IDF » (Inverse document frequency) est souvent utilisée en complément du « HashingTF » afin de réduire l'importance des termes qui apparaissent dans de trop nombreux documents. Il est calculé selon les formules suivantes :

Tableau 1: Calcul de la pondération TF-IDF

Coefficient	Description	Formule
TF	Fréquence des termes: Nombre de fois qu'un terme apparaît dans un document	$TF(t, d)$ (t=terme, d=document)
IDF	Inverse Document Frequency: Coefficient inverse proportionnel au nombre de documents	$IDF(t, D) = \log \frac{ D + 1}{DF(t, D) + 1}$ (DF=fréquence du document, D=corpus)
TF-IDF	Pondération TF-IDF: multiplication des deux précédents coefficients	$TFIDF(t, d, D) = TF(t, d) \cdot IDF(t, D)$.

Le code python suivant réalise la vectorisation et la pondération TF-IDF des données textuelles des tweets :

```
# Code de vectorisation des textes des tweets
#
from pyspark.ml.feature import HashingTF, IDF
# Définition du traitement de HashingTF
nbfeatures=6000
hashingTF = HashingTF(inputCol="tokens",outputCol="raw_features",
numFeatures=nbfeatures)
features_df = hashingTF.transform(base_tweets_sdf)
idf = IDF(inputCol="raw_features", outputCol="features")
idf_model = idf.fit(features_df)
scaled features df = idf_model.transform(features_df)
```

Code 5 : Extrait du programme Twitter-09-NLP-Sentiment-Analysis.ipnbj

Etape N°3: Crédit des jeux de données

3.1 partitionnement des données

Cette étape consiste en la création de deux jeux de données qui seront utilisés pour l'entraînement et la mesure de performance des différents modèles sélectionnés. Pour maximiser la performance des modèles prédictifs, j'ai opté pour une séparation aléatoire des tweets avec une proportion de 80% des données réservées à la phase d'entraînement et de 20% des données réservées pour la mesure de performance des modèles. Le code python suivant a été utilisé pour réaliser cette répartition des données :

```
#  
# Code de création des jeux de données 80/20% (Entrainement/Validation)  
#  
from pyspark.ml.feature import HashingTF, IDF  
train_sdf, test_sdf = base_tweets_sdf.randomSplit([0.8, 0.2], seed=1234)
```

Code 6: Extrait du programme Twitter-09-NLP-Sentiment-Analysis.ipnbj

3.2 Indexation des classes

Les algorithmes de classification nécessitent dans Spark de travailler sur des classes qui sont encodées en tant que valeurs numériques, en partant de la valeur zéro et avec un incrément de 1. J'ai donc indexé la classe de sortie comme suit :

- Tweet négatif: 0
- Tweet neutre : 1
- Tweet positif : 2

Etape N°4: Choix du modèle d'analyse

Choisir un modèle revient à choisir l'algorithme à utiliser pour construire le modèle. Il existe de nombreux algorithmes d'apprentissage machine et plusieurs d'entre eux seront testés et comparés afin de trouver le meilleur modèle.

Ces algorithmes peuvent être catégorisés selon le mode d'apprentissage et le type du problème traité :

- Apprentissage supervisé (supervised learning)
- Apprentissage non supervisé (unsupervised learning)
- Apprentissage par renforcement (reinforcement learning)

Pour ce projet, le choix s'est porté sur l'apprentissage supervisé qui est particulièrement bien adapté à la prédiction de résultats en s'aidant d'une base de données déjà étiquetées, comme le montre le schéma suivant :

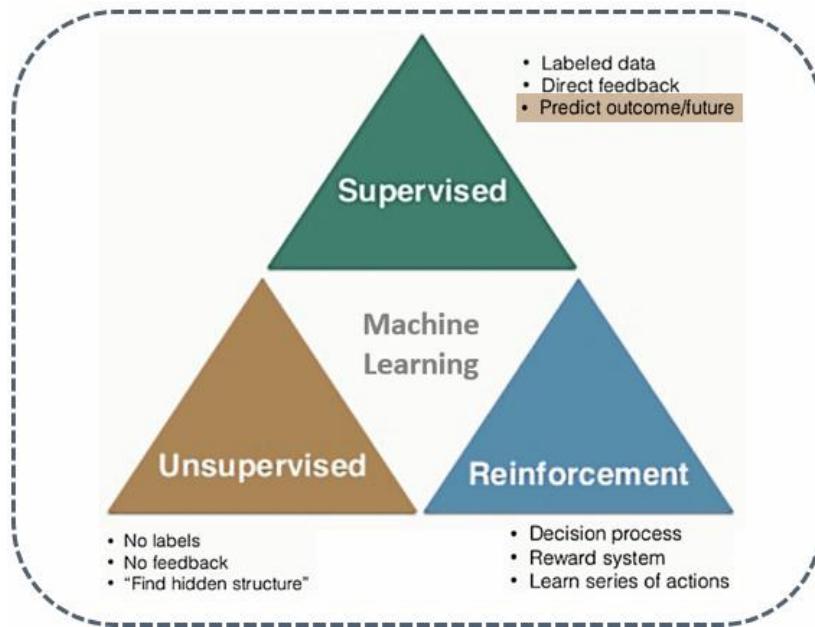


Figure 12: Synthèse des méthodes d'apprentissage machine

On distingue ensuite au niveau de l'apprentissage supervisé deux types d'algorithmes, selon que la variable à prédire est quantitative ou qualitative :

- Les algorithmes de régression
Ils sont adaptés aux problèmes pour lesquels le résultat à prédire est une valeur numérique
- Les algorithmes de classification
Ils tentent d'affecter une étiquette à une observation faite sur un ensemble de variables. L'ensemble des valeurs possibles pour cette étiquette doit être dénombrable et défini au préalable.

Pour ce projet, la variable de sortie est qualitative (tweet négatif, neutre et positif). J'ai donc choisi de tester des algorithmes de classification parmi ceux qui sont disponibles dans la librairie Spark MLlib afin de bénéficier du passage à l'échelle de Spark. Le choix s'est porté sur les algorithmes suivants qui sont compatibles avec la classification multinomiale et qui sont décrits brièvement dans les pages suivantes:

- Arbres de Decision
- Forêts aléatoires
- Naïves Bayes
- Régression logistique multinomiale
- One-vs-Rest

Notes :

- Les modèles basés sur SVM, le boosting et Perceptrons multicouches ont été écartés car leur implémentation avec la version utilisée de Spark (v2.4.4) reste limitée aux problèmes de classification binaire.
- A ces modèles d'apprentissage automatique proposés dans Spark, j'ai ajouté un modèle d'une autre famille de méthodes, dites symboliques, qui est très utilisé pour l'analyse de sentiments dans les tweets. Ce modèle est basé sur l'outil « TextBlob » qui est décrit dans la suite de ce document.
- Il est rare qu'un seul modèle suffise à produire de bonnes prédictions pour l'ensemble des données. J'ai donc testé la combinaison de plusieurs modèles qui permet souvent d'améliorer la performance globale de prédiction des modèles sous-jacents.

4.1 Arbres de décision

Les arbres de décision font partie des méthodes supervisées, ils ont pour but d'expliquer et/ou prédire une variable réponse « Y », à partir d'un ensemble de variables explicatives (X_1, \dots, X_p). On distingue les arbres de régression lorsque la variable de sortie Y est quantitative, et les arbres de classification lorsque Y est qualitative.

Le principe est de partitionner l'espace des variables explicatives en plusieurs régions et d'associer un modèle simple à chacune d'entre elles (généralement une constante), permettant ainsi de prédire une sortie « Y ». De nombreuses méthodes de partitionnement existent (C4.5, C5.0, CHAID, AID, etc.) mais la plus couramment utilisée reste la méthode CART (Classification and Regression Trees) qui consiste à partitionner de façon récursive chaque région en deux autres régions (on parle alors d'arbre binaire). La valeur prédite pour chaque région est la moyenne des valeurs de Y dans le cas d'arbre de régression ou la modalité la plus fréquente dans le cas d'une variable qualitative.

Les arbres de décision sont des modèles facilement interprétables et qui peuvent gérer des variables de plusieurs natures. Ils ont cependant l'inconvénient d'être des apprenants faibles, instables avec en général une capacité de modélisation trop limitée pour avoir de bonnes performances en pratique.

4.2 Forêts aléatoires

La stabilité des arbres de décision peut être grandement améliorée par l'utilisation des forêts aléatoires (Random Forests en anglais). Au lieu d'essayer d'optimiser l'arbre sur l'ensemble des données en une seule fois, les forêts consistent à générer plusieurs arbres différents en utilisant des échantillons indépendants issus des données, et à agréger ensuite les différentes prédictions fournies par ces différents arbres au moyen d'un vote dans le cadre de la classification.

Les forêts aléatoires sont une procédure de bagging particulière dans le sens où non seulement les individus sont rééchantillonés, mais où en plus les variables explicatives utilisées pour construire les noeuds sont choisies parmi un sous-ensemble de variables tirées au hasard. Ceci a pour effet de diminuer la liaison entre les différents arbres construits et améliore de ce fait les performances du prédicteur « baggé ».

Le schéma suivant montre le fonctionnement des forêts aléatoires :

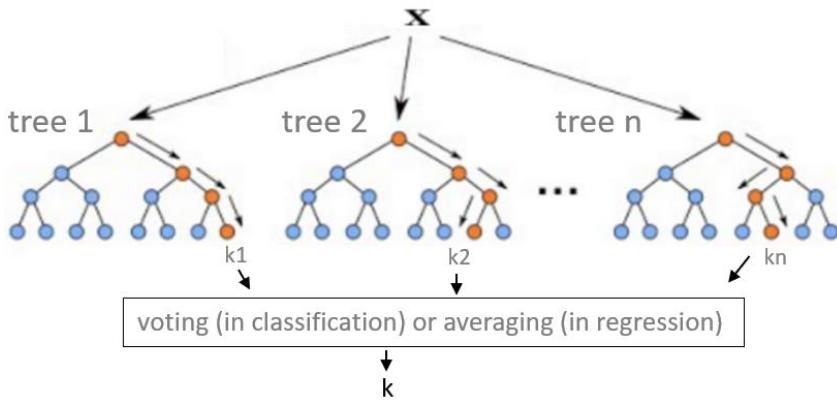


Figure 13: Algorithme des forêts aléatoires

4.3 Naïves Bayes

Naïve Bayes est un algorithme de classification basé sur les probabilités. Son utilisation est très populaire dans les applications du Machine Learning, notamment dans les problématiques de classification de texte. Naïve Bayes se base sur le théorème de Bayes fondé sur les probabilités conditionnelles, c'est-à-dire la détermination de la probabilité qu'un évènement se produise en fonction d'un évènement qui s'est déjà produit.

Le théorème de Bayes s'exprime sous la formule suivante :

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

Le théorème de Naïve Bayes s'appuie sur une hypothèse forte que chaque variable est indépendante, d'où le terme de « naïve ». Dans le cadre d'une classification de texte à l'aide de l'algorithme de Naïve Bayes, celle-ci se base sur le nombre d'occurrences d'un mot dans une phrase pour en déterminer sa catégorie. Chaque mot est alors pris de façon indépendante dans l'analyse.

4.4 Régression logistique multinomiale

La régression logistique binaire est basée sur la régression et utilise une fonction appelée sigmoïde ou courbe en S, qui la distingue de la régression linéaire :

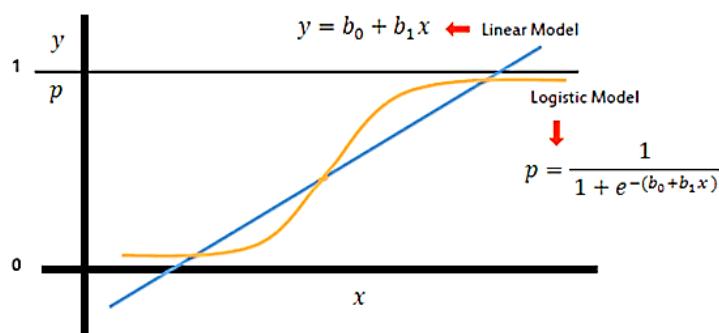


Figure 14: Différences entre la régression linéaire et la régression logistique binaire

La régression logistique multinomiale est une généralisation du modèle binaire lorsque la variable dépendante possède plus de deux modalités. Elle utilise en effet la fonction softmax qui elle une extension de la fonction sigmoïde au cas des multi-classes.

Dans le modèle multinomial, une valeur de la variable dépendante est distinguée comme catégorie de référence. On compare alors la probabilité de succès dans les autres catégories de la variable avec la probabilité de succès de la catégorie de référence. Cet algorithme produit une matrice de coefficients de dimension K x J où K est le nombre de classes attendues et J le nombre de caractéristiques (features). Les probabilités conditionnelles des K classes de résultats sont modélisées à l'aide de la fonction softmax suivante:

$$P(Y = k | \mathbf{X}, \boldsymbol{\beta}_k, \beta_{0k}) = \frac{e^{\boldsymbol{\beta}_k \cdot \mathbf{X} + \beta_{0k}}}{\sum_{k'=0}^{K-1} e^{\boldsymbol{\beta}_{k'} \cdot \mathbf{X} + \beta_{0k'}}}$$

Pour chaque nouvelle donnée à traiter, (K-1) modèles seront exécutés et la classe avec la plus grande probabilité sera choisie comme la classe prédictive pour cette donnée.

4.4 One-vs-Rest

L'algorithme One-Vs-Rest (ou One-vs-All) proposé dans Spark permet d'utiliser la régression logistique pour la classification multi-classes. Le principe consiste à découper le problème de classification multi-classes en une multitude de problèmes de classification binaire où l'on classe une par une chaque classe par rapport à toutes les autres :

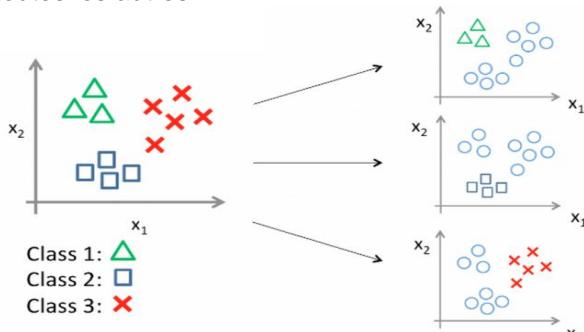


Figure 15 : Algorithme One-vs-Rest

4.6 Textblob

On distingue deux familles de méthodes pour l'analyse de sentiments dans un texte. Celles utilisant l'apprentissage automatique et celles, dites symboliques, basées sur un lexique de tonalités et des règles. TextBlob fait partie de cette seconde famille et est très utilisé pour l'analyse de sentiments sur les tweets. Basé sur la librairie NLTK, il fournit une API simple et extrêmement rapide pour effectuer des tâches courantes d'analyse textuelle en anglais et en français (tokenisation, extraction de données, analyse des sentiments, classification, traduction, vérification et correction orthographiques).

TextBob procède à l'analyse de sentiments d'un texte en affectant à chaque mot un score lié à sa position dans le texte et à sa signification en matière de sentiment et de subjectivité. TextBlob moyenne ensuite les résultats obtenus pour fournir un indice entre 0 et 1 qui permet de classer les tweets en trois catégories : négatif, neutre et positif.

Note :

Cet algorithme est extrêmement rapide mais ne bénéficie cependant pas du passage à l'échelle des traitements proposés dans Spark. Une analyse plus poussée des performances de cet algorithme reste donc à réaliser avant toute mise en production à grande échelle.

4.7 Méthodes d'ensemble

Un moyen très simple de créer un classificateur encore meilleur que ceux proposés précédemment consiste à agréger les prédictions de chacun de ces classificateurs et de prédire la classe qui obtient le plus grand nombre de votes, selon un vote majoritaire ou pondéré.

Notes :

- Les méthodes d'ensemble fonctionnent d'autant mieux que les prédicteurs utilisés sont réellement indépendants les uns des autres. En effet, plus ils sont indépendants entre eux, plus il y a de chances qu'ils fassent des erreurs très différentes, améliorant de ce fait l'exactitude de l'ensemble.
- Pour vérifier cette indépendance, on peut utiliser une matrice de corrélation des résultats obtenus par chacun des modèles.
- Pour obtenir des classificateurs suffisamment variés, une solution consiste à utiliser des algorithmes d'entraînement très différents.

Etape N°5: Entrainement des modèles

Cette étape a pour objectif d'appliquer un modèle d'apprentissage aux données d'entraînement selon le processus suivant :

- Définition d'un pipeline pour le modèle concerné
- Définition des hyperparamètres du modèle
- Entraînement et optimisation du modèle

L'exemple de la page suivante montre l'entraînement du modèle basé sur l'algorithme de classification logistique multinomiale qui s'est révélé être l'algorithme le plus performant parmi tous ceux qui ont été testés dans le cadre de ce projet

5.1 Définition d'un pipeline

Spark permet de définir des pipelines qui sont des flux de travail qui enchaînent différentes étapes de traitement des données se trouvant dans des DataFrames. Un pipeline se compose des éléments suivants :

- Les transformateurs (transformers)
Ils transforment un dataframe en un autre dataframe, en rajoutant en général une ou plusieurs colonnes au dataframe
- Les estimateurs (estimators)
Un estimateur est un algorithme qui permet de construire un modèle à partir de données d'un DataFrame. Une fois construit, ce modèle devient un transformateur.

Pour ce projet, une partie des traitements de préparation des données a été intégrée au pipeline.

5.2 Définition des hyperparamètres

Chaque algorithme d'apprentissage comporte un certain nombre de paramètres qui permettent d'optimiser la performance du modèle sélectionné. Ces paramètres numériques dont dépend la modélisation sont appelés « hyperparamètres ». Spark fournit des valeurs par défaut pour l'ensemble des hyperparamètres. Cependant, certains hyperparamètres ont un impact important sur les performances finales du modèle et nécessitent d'être optimisés.

5.3 Entrainement et optimisation des modèles

L'apprentissage du modèle sur le jeu de données d'entraînement s'effectue par la fonction Spark « fit » avec en paramètres le jeu de données concerné et le modèle à entraîner.

Pour optimiser un modèle donné, on réalise des tests successifs d'apprentissage du modèle sur le même jeu de données d'entraînement mais avec différentes combinaisons de valeurs des hyperparamètres. Cette recherche est automatisée avec l'utilisation de la fonction Spark de validation croisée CrossValidator(), basée sur une technique d'échantillonage, et d'une fonction d'évaluation du modèle qui pour cet exemple est la classe « MulticlassClassificationEvaluator ».

Le code python suivant montre l'entraînement du modèle basé sur la régression logistique multinomiale avec l'optimisation des hyperparamètres suivants: MaxIter (Nombre d'itérations), RegParam (terme de régularisation), ElasticNetParam (Pénalité pour éviter le surapprentissage) :

```
# Code python d'entraînement et d'optimisation du modèle basé sur la régression logistique
#
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
# Définition de l'algorithme de régression logistique
lr = LogisticRegression(featuresCol = 'features', labelCol = 'label')
# Evaluation du modèle
evaluator = MulticlassClassificationEvaluator(metricName="accuracy")
# Définition des hyperparamètres
paramGrid = (ParamGridBuilder()
    .addGrid(lr.regParam, [0.01,0.02,0.03])
    .addGrid(lr.elasticNetParam, [0.1,0.2,0.3])
    .addGrid(lr.maxIter, [10,20])
    .build())
# Définition de la validation croisée et de l'évaluation du modèle
cv = CrossValidator(estimator=lr, estimatorParamMaps=paramGrid, evaluator=evaluator,
numFolds=5)
# entraînement du modèle
cvModel = cv.fit(train_sdf)
# prédictions du modèle sur les données de test
test_predictions_sdf = cvModel.transform(test_sdf)
# Extraction des meilleurs paramètres
bestModel = cvModel.bestModel
bestModel.extractParamMap()
# Calcul Accuracy
cm = test_predictions_sdf .select("label", "prediction")
accuracy = cm.filter(cm.label == cm.prediction).count() / cm.count()
print("Model accuracy: %.3f%%" % (accuracy * 100))
```

Code 7: Extrait du programme Twitter-08-NLP-Models-Comparaison.ipnbj

Etape N°6: Prédiction et évaluations des Résultats

Cette étape consiste à évaluer chacun des modèles sur les même données de validation afin de trouver le modèle ayant les meilleures performances pour la prédiction des sentiments liés aux contenus des tweets.

6.1 Prédiction des modèles

Une fois qu'un modèle est entrainé, il est possible d'appliquer ce modèle sur de nouvelles données en utilisant la fonction Spark « transform ». Les prédictions obtenues sur le jeu de validation permettent de mesurer la performance de généralisation de chacun des modèles.

6.2 Evaluation des modèles

L'évaluation d'un modèle consiste à mesurer les écarts entre les prédictions du modèle et les résultats attendus. Spark propose les mesures suivantes pour les algorithmes de classification:

Table 2: Mesures d'évaluation des modèles de classification

Mesure	Description	Formule
TP	Vrai Positif (True Positive) : Nombre de documents pertinents qui sont inclus à raison dans le résultat	--
TN	Vrai Négatif (True Negative) : Nombre de documents non pertinents et non inclus à raison dans le résultat	--
FP	Faux Positif (False Negative) : Nombre de documents pertinents qui sont inclus à tort dans le résultat	--
FN	Faux Négatif (False Negative) : Nombre de documents pertinentes qui ne sont pas inclus à tort dans le résultat	--
Precision	La précision (weightPrecision) mesure la proportion de documents pertinents sur l'ensemble de documents retrouvés, pour une requête donnée	$PPV = \frac{TP}{TP + FP}$
Recall	Le rappel (weightedRecall) mesure la proportion de documents pertinents retrouvés par rapport à l'ensemble de documents qu'il fallait retrouver	$TPR = \frac{TP}{TP + FN}$
Accuracy	La justesse (Accuracy) est une mesure qui représente le pourcentage de prédictions correctes de l'algorithme	$ACC = \frac{TP + TN}{TP + TN + FP + FN}$
F1	F-mesure (F-socre ou F1 en anglais) est l'estimateur par défaut de Spark et est défini comme la moyenne harmonique de la précision et du rappel.	$F_1 = 2 \cdot \frac{PPV \cdot TPR}{PPV + TPR} = \frac{2TP}{2TP + FP + FN}$

Notes :

- Dans le cadre du multi-classes (n classes > 1), les moyennes globales de la précision et du rappel sur l'ensemble des classes i sont évaluées en calculant d'abord la précision et le rappel sur chaque classe i suivie d'un calcul de la moyenne des précisions et des rappels sur les n classes.
- Pour ce projet, l'ensemble des modèles a été évalué selon la même mesure, basée sur la justesse (accuracy).

Le code suivant a été utilisé pour calculer ces mesures pour le modèle basé sur la régression logistique:

```
# Code pour évaluer les modèles(Accuracy, F1, Precision, Recall)
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
# Accuracy
evaluator = MulticlassClassificationEvaluator(metricName="accuracy")
LR_accuracy = evaluator.evaluate(test_predictions_sdf)
print("LR classifier Accuracy (test) = %g" % (LR_accuracy))
# F1
evaluator = MulticlassClassificationEvaluator(metricName="f1")
LR_f1 = evaluator.evaluate(test_predictions_sdf)
print("LR classifier F1 (test) = %g" % (LR_f1))
# weightedPrecision
evaluator = MulticlassClassificationEvaluator(metricName="weightedPrecision")
LR_precision = evaluator.evaluate(test_predictions_sdf)
print("LR classifier weightedPrecision (test) = %g" % (LR_precision))
# weightedRecall
evaluator = MulticlassClassificationEvaluator(metricName="weightedRecall")
LR_recall = evaluator.evaluate(test_predictions_sdf)
print("LR classifier weightedRecall (test) = %g" % (LR_recall))
```

Figure 16: Extrait du programme Twitter-08-NLP-Models-Comparaison.ipynb

6.3 Comparaison des modèles

6.3.1 Comparaison de la performance des modèles testés

Le tableau suivant montre la performance des différents modèles testés en se basant sur le taux de bonne classification. Ce taux correspond au nombre de vrais positifs plus le nombre de vrais négatifs divisé par le nombre total de prédictions (voir détails en annexe 8).

Table 3: Comparaison des modèles testés

N°	Modèle	Accuracy% Données d'entraînement	Accuracy% Données de validation
1	Régression logistique Multinomiale	98,76%	85,25%
2	Classifieur Baysien Naïf	93,82%	82,80%
3	One-vs-Rest	99,62%	81,90%
4	Classifieur TextBlob (*)	83,69%	80,80%
5	Forêt aléatoire	87,44%	80,65%
6	Arbres de Décision	82,96%	73,73%

(*) Modèle non supervisé

On peut remarquer les points suivants:

- Le meilleur modèle est basé sur la régression logistique multinomiale. Ce modèle obtient en effet le meilleur taux de classification pour les données de validation avec plus de 85% de bonnes prédictions.
- Le meilleur modèle sur les données d'entraînement est basé sur One-vs-Rest avec un taux proche de 100% qui laisse penser à du sur-apprentissage. Son mauvais classement sur les données de validation vient confirmer ce point.
- Le dernier modèle est basé sur les arbres de décision. Ce résultat montre les capacités relativement faibles de prédiction de ce type d'algorithme.
- Le modèle basé sur les forêts aléatoires donne de meilleurs résultats que les arbres de décision. Ce résultat est conforme à ce que l'on peut attendre car les forêts aléatoires ont été concues pour pallier les défauts des arbres de décision (apprenants faibles, instables)
- Le modèle basé sur textBlob offre des performances qui se situent dans le bas du classement. Ceci montre toute la difficulté pour analyser correctement un corpus de textes lié à une thématique particulière comme la culture. Ainsi certains mots comme « noir » ou « sombre » sont incorrectement connotés comme très négatifs par le modèle textBlob alors qu'ils font simplement référence à la couleur d'une oeuvre. C'est le cas par exemple pour l'artiste Soulage qui ne peint qu'avec du noir. Le lexique et la notation de Textblob sont donc trop généralistes et nécessitent une adaptation pour mieux appréhender le monde de la culture.
- Il est à noter que les autres mesures d'évaluation (F1, Précision et Recall) donnent le même classement (voir annexe 8).

6.3.2 Courbes ROC

La courbe d'efficacité du récepteur (Receiver Operating Characteristic ou ROC) est un outil communément utilisé pour visualiser la performance des différents modèles basés sur des classificateurs binaires. Cette courbe donne le taux de vrais positifs (fraction des résultats positifs qui sont effectivement détectés) en fonction du taux de faux positifs (fraction des résultats négatifs qui sont incorrectement détectés).

Dans un contexte multi-classes, les courbes doivent être calculées indépendamment pour chacune des classes de sortie. Les graphiques suivants montrent les courbes ROC pour les meilleurs modèles trouvés :

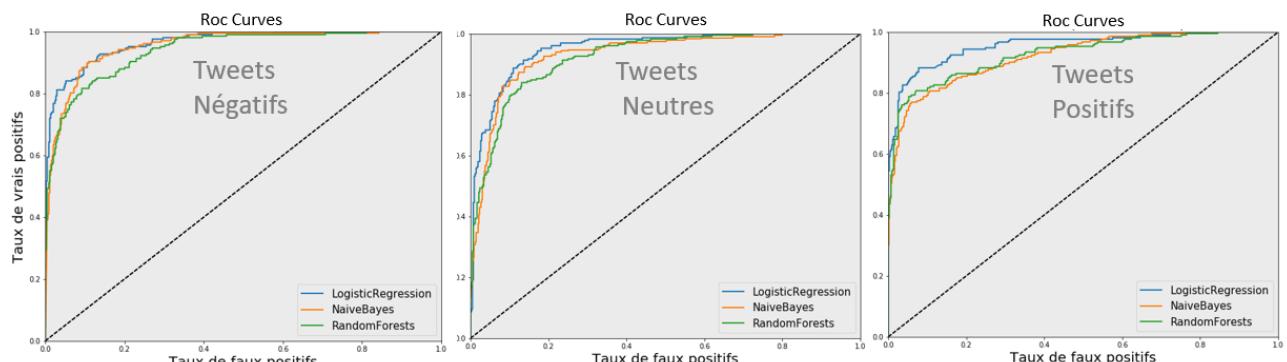


Figure 17: Comparaison des courbes ROC pour plusieurs modèles

Notes :

- Les courbes ROC de la page précédente confirment que le modèle basé sur la régression logistique multinomiale surclasse les autres modèles, quelque soit la classe de sortie (tweet négatif, tweet neutre et tweet positif). La différence de performance entre les deux premiers modèles est particulièrement visible au niveau de la prédiction des tweets positifs.
- Si on superpose les courbes obtenues sur les trois classes de sortie, on constate que les modèles ont plus de difficulté à identifier les tweets neutres que les tweets positifs ou négatifs.

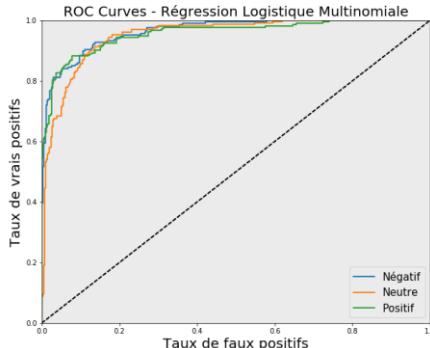


Figure 18: Superposition des courbes ROC de la Régression Logistique Multinomiale

6.3.3 Corrélation entre les modèles testés

Les méthodes d'ensemble fonctionnent d'autant mieux que les méthodes qui sont agrégées sont réellement indépendantes les unes des autres. Il est donc intéressant de pouvoir comparer les corrélations des modèles testés. J'ai utilisé la fonction Seaborn HeatMap pour visualiser ces corrélations. Les graphiques suivants montrent les corrélations des modèles testés pour chacune des classes de sortie (Tweet négatif, neutre et positif).

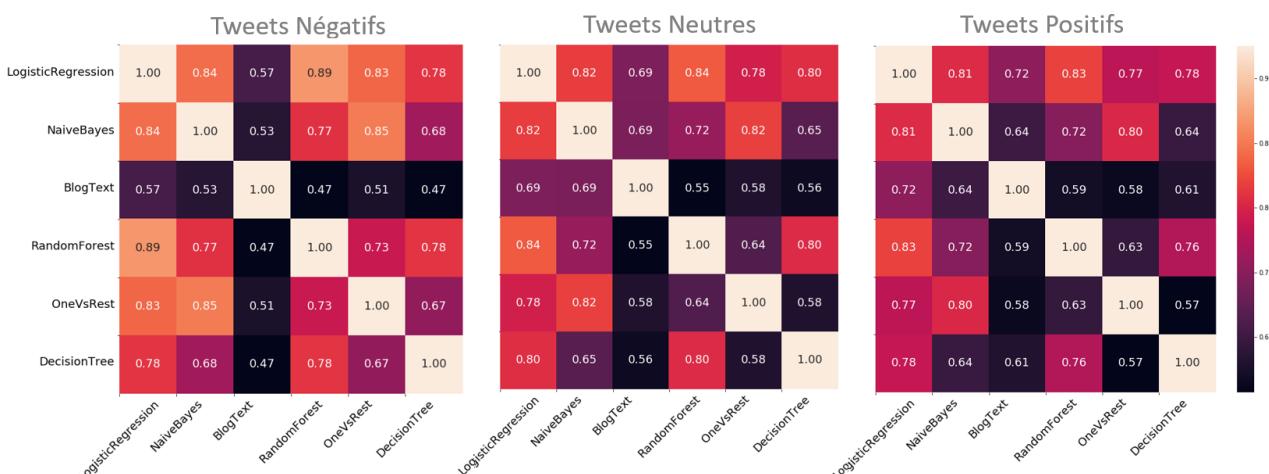


Figure 19: Corrélation entre les modèles testés

Note :

- On constate que le modèle basé sur « textBlob » est décorrélé des autres modèles. La raison est que sa conception est totalement différente des autres modèles.
- Les autres modèles sont globalement tous corrélés entre eux. Il existe toutefois d'autres dans Spark qui sont potentiellement moins corrélés comme les perceptrons multi couches. Ils ne sont cependant pas encore compatibles avec le classement multi-classes.

6.3.4 Combinaison des meilleurs modèles

Cette étape consiste à combiner les meilleurs modèles trouvés dans l'étape précédente afin d'améliorer la performance globale de prédiction de ces modèles. J'ai utilisé l'algorithme du vote majoritaire et du vote souple (utilisation de moyennes pondérées) pour combiner les prédictions de plusieurs modèles.

Le tableau suivant montre la combinaison de plusieurs modèles qui permet d'améliorer globalement les prédictions sur le jeu des données de validation (voir détails en annexe 8).

Tableau 2: Combinaison de plusieurs modèles avec le vote souple ou majoritaire

N°	Combinaison de modèles	Vote	Natif Spark	Accuracy %
--	LogisticRegression (meilleur modèle seul)	--	oui	85,25%
1	LogisticRegression + NaiveBayes + OneVsRest + RandomForest	majoritaire	oui	85,56%
2	LogisticRegression + NaiveBayes + OneVsRest	majoritaire	oui	85,71%
3	LogisticRegression + NaiveBayes + OneVsRest + RandomForest + textBlob	majoritaire	non	87,71%
4	LogisticRegression + NaiveBayes + textBlob	majoritaire	non	88,32%
5	LogisticRegression + NaiveBayes + RandomForest + textBlob	souple	non	89,09%

Notes :

- On constate que beaucoup de combinaisons de modèles font de meilleures prédictions que les modèles pris seuls. Cela montre tout l'intérêt d'utiliser des méthodes d'ensemble.
- La meilleure combinaison de modèles n'est pas celle des 4 meilleurs modèles. Ainsi, le modèle basé sur les forêts aléatoires est supérieur au modèle one-vs-rest, même s'il est moins performant individuellement. La raison vient du fait qu'il est moins corrélé avec les autres modèles.

6.3 Sélection du meilleur modèle

Le meilleur modèle pour ce projet est celui dont les prévisions sont les plus proches des résultats attendus et qui permet le passage à l'échelle. J'ai sélectionné le modèle ensembliste suivant qui combine selon un vote majoritaire les résultats des trois meilleurs modèles Spark :

- LogisticRegression
- NaiveBayes
- OnevsRest

Notes :

- J'ai écarté le meilleur modèle ensembliste N°5 car il fait appel à une méthode (textBlob) qui est très rapide mais n'est pas nativement implémentée dans Spark. Elle ne bénéficie donc pas directement du passage à l'échelle de Spark.
- Le vote souple pour le modèle N°5 a consisté à pondérer les résultats du modèle textBlob afin de lui donner plus de poids par rapport aux trois autres modèles issus de l'apprentissage automatique.
- Les résultats de l'analyse de sentiments avec le meilleur modèle sont présentés dans l'annexe 7.

Etape N°7: Visualisation des résultats

7.1 Affichage en continu d'un nuage des mots-clés trouvés dans les tweets collectés

Le nuage des mots-clés (#hashtags) est intégré dans des pages Web dynamiques. Ces pages sont générées en python avec l'interface CGI (Common Gateway Interface) qui permet de créer du contenu Web dynamique.

L'affichage en continu des tweets ainsi que la carte interactive des sites culturels sont isolés du reste des pages Web grâce à l'utilisation des « iframes ». Cette balise faisait déjà partie de la spécification HTML4 mais a été développée dans la version 5. Elle permet d'insérer du contenu Web plus facilement qu'en utilisant Ajax qui requiert des fonctions « XMLHttpRequest ».

L'exemple suivant montre la création d'un iframe pour l'affichage des mots-clés (#hashtags) en continu :

```
# 
# Code python d'insertion d'un iframe
#
print('<TABLE BORDER="0" CELLPADDING="0" CELLSPACING="0" WIDTH="100%" BGCOLOR="#B6B687">')
print('<TR><TD BGCOLOR="#E0E0E0" ALIGN="CENTER">')
mapname='<iframe src="stream/bagofwords.html" height="100" width="1300"></iframe>'
print(mapname)
print('</TD></TR></TABLE>')
```

Code 8: Extrait du programme Twitter-11-WebSite-Creation.ipnbj

7.2 Synthèse des résultats

Je présente la synthèse des résultats sous la forme de graphiques générés à partir des librairies python spécialisées suivantes :

- Matplotlib : Création de graphiques de base y compris en temps réel
- Seaborn : Ajout de fonctionnalités à Matplotlib pour une meilleure qualité graphique et une compatibilité avec les dataframes de la librairie Pandas.
- PIL (Python Imaging Library) : Traitement d'images
- Wordcloud : Librairie dédiée à la création et l'affichage de nuages de mots
- Folium/Leaflet : Création de cartes interactives à partir des coordonnées géographiques des éléments à afficher

L'exemple suivant montre la visualisation des résultats pour les 5 musées les plus présents sur Twitter, avec une synthèse des analyses réalisées pour le musée du Louvre, lorsqu'on clique sur son lien (d'autres exemples sont donnés en annexe 10):



Figure 20: Visualisation des résultats sous la forme de pages Web

2. Architecture du Projet

2.1 Architecture Technique

2.1.1 Architecture générale

L'architecture générale du projet s'articule autour des frameworks suivants :

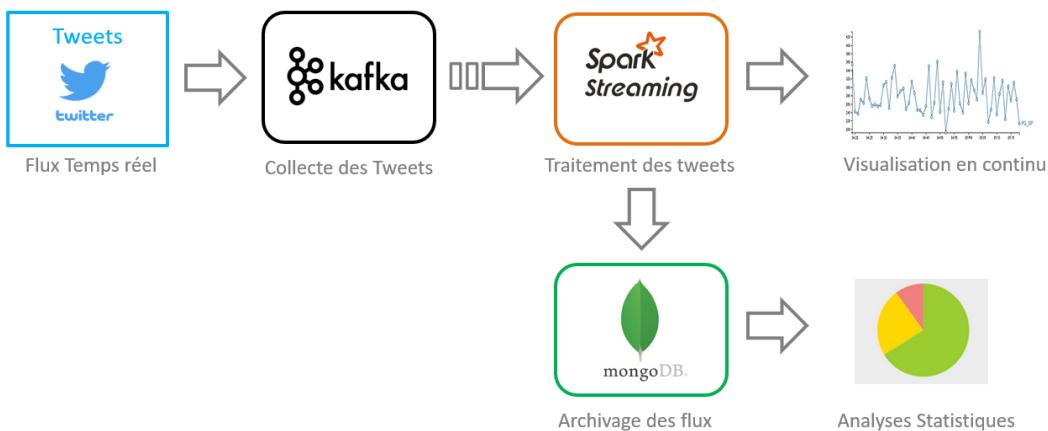


Figure 21: Architecture générale du projet

Le schéma suivant montre l'architecture technique du projet avec l'utilisation de clusters Kafka, Spark et mongoDB pour gérer le passage à l'échelle, la réplication ainsi que la reprise sur panne :

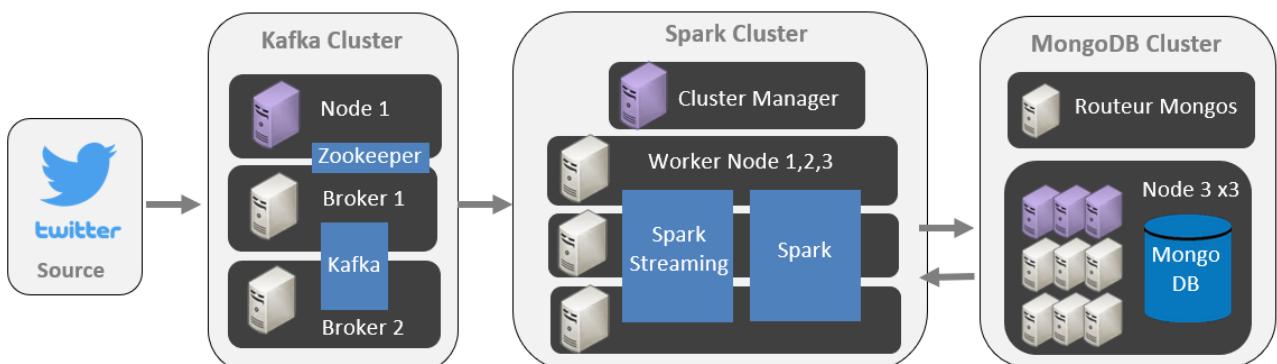


Figure 22: Architecture Technique du projet

Notes :

Le cluster Kafka est composé de 3 serveurs :

- 2 Brokers Kafka qui est le minimum requis pour gérer la haute disponibilité du cluster. Ils permettent aussi la scalabilité du système.
- Pour faciliter la configuration et la coordination des brokers Kafka, j'ai ajouté un serveur zookeeper (en violet). Pour une mise en production, il est recommandé d'avoir au minimum 3 serveurs zookeeper pour gérer la haute disponibilité.

Le cluster Spark est composé de 4 serveurs :

- 1 serveur « cluster manager » (en violet) qui est le gestionnaire du cluster et de ses ressources
- 3 serveurs (Worker nodes) pour la scalabilité du système.

Le cluster MongoDB est composé de 10 serveurs :

- 3 serveurs de configuration organisés en un replicaset (en violet), qui permet de bénéficier de la haute disponibilité.
- 6 serveurs pour la base de données, regroupés en deux replicaset pour gérer la scalabilité (shards) et la haute disponibilité.
- 2 routeurs mongos pour le routage des requêtes. Il faut un minimum de deux routeurs mongos pour bénéficier de la tolérance aux pannes du système.

2.1.2 Liste des progiciels/frameworks

Pour ce projet, j'ai utilisé les progiciels/frameworks suivants :

- Réseau social Twitter à partir duquel on collecte les tweets à analyser
- Kafka pour la mise à disposition en temps réel des tweets collectés
- Zookeeper comme gestionnaire de configuration du cluster Kafka
- Spark comme framework de calcul distribué (utilisation de Spark Streaming)
- MongoDB comme base de données NoSQL pour l'archivage des tweets
- Virtualbox pour la création d'un serveur Linux sous Windows 10
- Docker pour la création des conteneurs gérant les différents composants du projet
- Docker Compose pour l'administration des conteneurs Docker
- Centos 7, Alpine 3, et Debian 10 pour les systèmes d'exploitation au sein des conteneurs Docker
- Apache comme serveur Web
- Python comme langage de programmation du projet
- Jupyter Notebook comme outil de développement

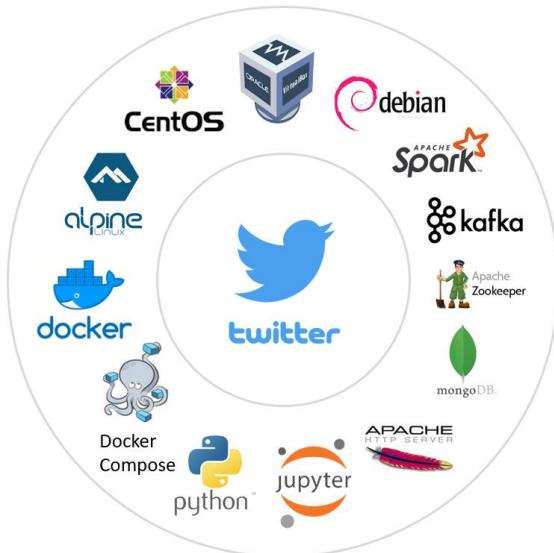


Figure 23: Liste des outils progiciels/frameworks utilisés dans le projet

Tous les composants de cette architecture ont été déployés au travers de conteneurs Docker qui nécessitent que très peu de ressources. Cette solution a permis de lancer l'ensemble des conteneurs Docker suivants sur un seul ordinateur pour tester le projet :

Table 4 : Liste des conteneurs Docker lancés pour la mise en oeuvre du projet

N°	Docker	Conteneur	IMAGE	PORTS
1	Cluster kafka	kafka01	twitter/kafka	9091->9092
2		kafka02	twitter/kafka	9092->9092
3		zookeeper01	twitter/zookeeper	2181->2181
4	Cluster Spark	sparkworker1	twitter/sparkmaster:2.4	8091->8081
5		sparkworker2	twitter/sparkmaster:2.4	8092->8081
6		sparkworker3	twitter/sparkmaster:2.4	8093->8081
7		sparkmaster	twitter/sparkmaster:2.4	8090->8080
8	Cluster MongoDB	mongocfg11	twitter/mongodb:4.2	--
9		mongocfg12	twitter/mongodb:4.2	--
10		mongocfg13	twitter/mongodb:4.2	--
11		mongodb21	twitter/mongodb:4.2	--
12		mongodb22	twitter/mongodb:4.2	--
13		mongodb23	twitter/mongodb:4.2	--
14		mongodb31	twitter/mongodb:4.2	--
15		mongodb32	twitter/mongodb:4.2	--
16		mongodb33	twitter/mongodb:4.2	--
17		mongos01	twitter/mongodb:4.2	30117->27017
18		mongos02	twitter/mongodb:4.2	30217->27017
19	Serveur Web	httpd	twitter/httpd:2.4	8080->80, 8443->443
20	Jupyter	jupyter01	twitter/jupyter:1.0	9888->8888

2.1.3 Liste des traitements

Ce projet s'articule autour des traitements suivants:

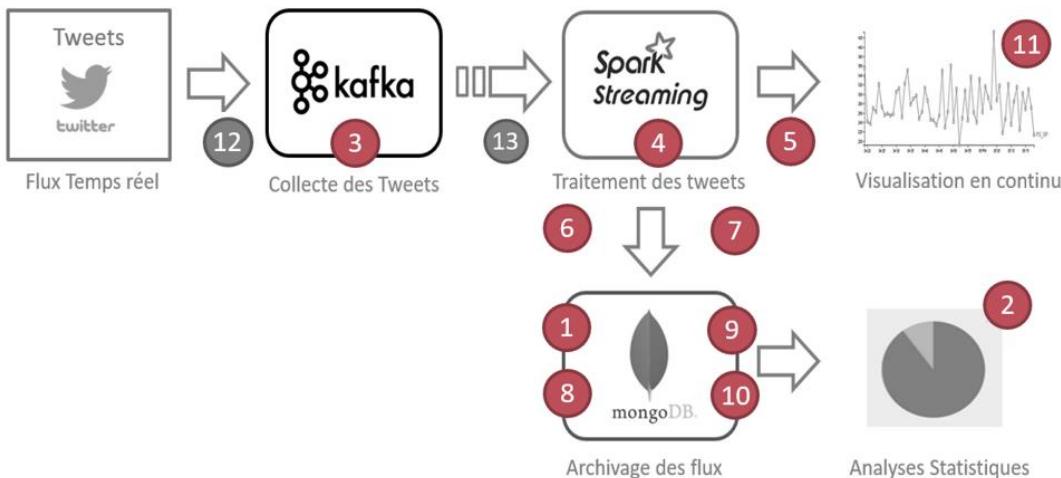


Figure 24: Traitements du projet

Table 5: Description des traitements du projet

N°	Programme	Description
1	Twitter-01-Apprentissage-Upload-Data.ipynb	Chargement des données d'apprentissage dans MongoDB (3000 Tweets annotés)
2	Twitter-02-Apprentissage-Exploration-Data.ipynb	Exploration des données d'apprentissage (Histogrammes, nuages de mots, etc.)
3	Twitter-03-Streaming-KafkaProducer-Loop.ipynb	Streaming des Tweets issus de Twitter avec kafka
4	Twitter-04-Streaming-SparkConsumer-loop.ipynb	Analyse en continu des Tweets collectés dans Spark Streaming et archivage dans MongoDB
5	Twitter-05_Streaming-WordCloud-Loop.ipynb	Génération d'un nuage de mots avec les résultats de l'analyse des tweets collectés de puis Kafka
6	Twitter-06-Streaming-Tweets-Data-Extraction.ipynb	Extraction et nettoyage des données textuelles des Tweets pour les insérer dans une collection MongoDB
7	Twitter-07-Batch-Tweets-Upload-Loop.ipynb	Initialisation de l'historique des Tweets avec des Tweets provenant des sites culturels officiels
8	Twitter-08-NLP-Models-Comparaison.ipynb	Comparaison de plusieurs modèles pour l'analyse de sentiments afin de sélectionner le meilleur modèle
9	Twitter-09-NLP-Sentiment-Analysis.ipynb	Analyse de sentiments des tweets collectés avec le meilleur modèle sélectionné précédemment
10	Twitter-10-WebSite-Summary-Creation.ipynb	Synthèse des analyses statistiques pour un affichage dans des pages Web
11	Twitter-11-WebSite-Creation.ipynb	Affichage dynamique et en continu des nuages de mots analysés dans Spark Streaming
12	Twitter-12-Test-de-Charge-Kafka-SparkStreaming-MongoDB-Producer	Test de charge : Streaming de 1000 tweets en 1 minute avec Kafka
13	Twitter-13-Test-de-Charge-Kafka-SparkStreaming-MongoDB-Consumer	Test de charge: Analyse en continu des 1000 tweets avec Spark Streaming et archivage des tweets dans mongoDb

Notes:

- Un exemple d'exécution de chaque traitement ainsi que son code sont disponibles dans les Notebooks Jupyter qui ont été téléchargés dans Moodle (voir détails en annexe 4)
- L'ordonnancement des traitements est détaillé en annexe 8
- Les traitements 1,2, 7 et 8 sont des traitements qui ne sont pas récurrents et ne sont à lancer qu'une seule fois (Chargement de la base d'apprentissage, exploration des données, Initialisation de l'historique des tweets, comparaison des modèles)
- Les traitements 4,5,6 peuvent être fusionnés en un seul traitement
- Les traitements 9 et 10 peuvent être aussi fusionnés en un seul traitement
- Le traitement 1 est optionnel et peut être remplacé par l'utilisation du fichier csv comportant les données d'apprentissage

2.2 Passage à l'échelle

2.2.1 Passage à l'échelle de Kafka

Présentation de Kafka:

Kafka est un système distribué de messagerie par abonnement qui possède de nombreuses qualités :

- Fiabilité: Kafka est distribué, partitionné, répliqué et tolérant aux fautes.
- Scalabilité: Kafka se met à l'échelle facilement et sans temps d'arrêt.
- Durabilité: Kafka utilise un commit log distribué, ce qui permet de stocker les messages sur le disque le plus rapidement possible.
- Performance: Kafka a un débit très élevé pour la publication et l'abonnement de messages.

Son architecture est basée sur les principaux composants suivants :

- Topic: C'est un flux de messages appartenant à une catégorie particulière. Les données y sont stockées
- Partitions: Chaque topic est divisé en partitions. Chaque partition contient des messages dans une séquence ordonnée immuable.
- Offset: Les enregistrements d'une partition ont chacun un identifiant séquentiel appelé offset, qui permet de l'identifier de manière unique dans la partition
- Répliques: Les répliques sont des backups d'une partition, en cas de perte de données.
- Brokers: Les brokers sont responsables de maintenir les données publiées, au niveau des partitions
- Cluster: Un système Kafka ayant plus qu'un seul Broker est appelé cluster Kafka. L'ajout de nouveaux brokers se fait de manière transparente et sans temps d'arrêt.
- Producers: Les producteurs sont les éditeurs de messages à un ou plusieurs topics Kafka. Ils envoient des données aux brokers Kafka. Chaque fois qu'un producteur publie un message à un broker, ce dernier rattache le message au dernier segment de la partition sélectionnée.
- Consumers: Les consommateurs lisent les données à partir des brokers. Ils souscrivent à un ou plusieurs topics, et consomment les messages publiés en extrayant les données à partir des brokers.
- Leaders: Le leader est le noeud responsable de toutes les lectures et écritures d'une partition donnée.
- Follower: Si le leader tombe en panne, l'un des followers deviendra le nouveau leader
- Zookeeper : Kafka travaille de concert avec Apache ZooKeeper qui est un logiciel de gestion de configuration pour des systèmes distribués et un service de coordination distribuée.

Passage à l'échelle

Le passage à l'échelle s'effectue par :

- l'augmentation du nombre de partitions pour distribuer les données d'un même topic. Lors de la création d'un topic, on indique le nombre de partitions souhaitées (voir exemple page suivante)
- L'augmentation du nombre de consumers pour un même topic. Ceci est rendu possible par la création de groupes de consumers dans Kafka
- L'augmentation du nombre de Brokers

La figure suivante montre la répartition de la charge entre plusieurs brokers Kafka qui gère chacun une partition d'un topic donné :

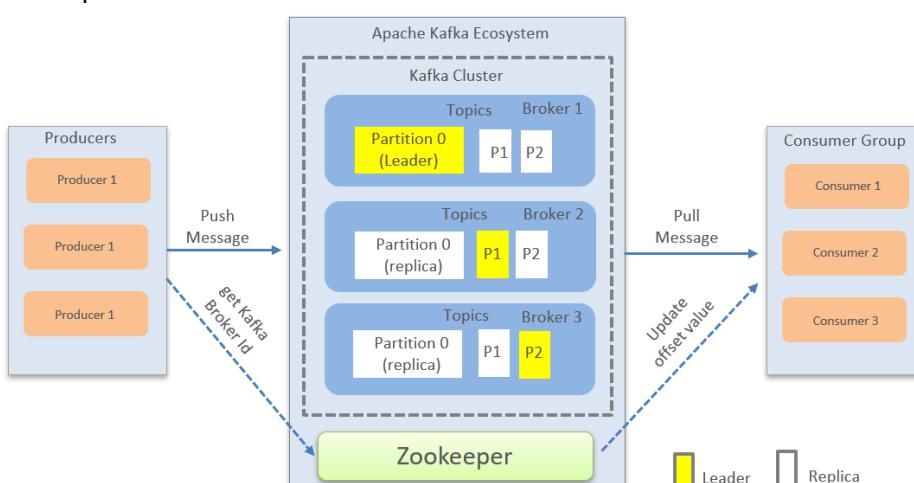


Figure 25: Passage à l'échelle de Kafka

Les commandes suivantes montrent le passage à l'échelle de Kafka en augmentant le nombre de partitions (de 1 à 3) et le nombre de replicas (de 1 à 2) pour le topic « tweets01 » :

```
# Vérification des conteneurs docker kafka et zookeeper
$ docker ps|grep -e kafka -e zookeeper
d79047371dfc "start-kafka.sh" Up 0.0.0.0:9091->9092/tcp kafka01
c54c7e04b786 "start-kafka.sh" Up 0.0.0.0:9092->9092/tcp kafka02
f221825364d6 "/bin/sh -c ..." Up 0.0.0.0:2181->2181/tcp zookeeper01

# Vérification de la création des topics dans les logs de kafka
$ docker logs kafka01| grep topics
creating topics: tweets01:1:1
creating topics: tweets02:1:1

# Connexion au serveur zookeeper
$ docker exec -it zookeeper01 /bin/bash

# Lancement du client zookeeper
# /opt/zookeeper-3.4.14/bin/zkCli.sh -server localhost:2181
2020-02-17 [myid:] - INFO  [Client environment:zookeeper.version=3.4.14
2020-02-17 [myid:] - INFO  [main- Session establishment complete on server 127.0.0.1:2181
WATCHER:::

# Liste des Brokers actifs gérés par Zookeeper
[zk: localhost:2181(CONNECTED) 0] ls /brokers/ids
[1002, 1001]

# Liste des topics Kafka
[zk: localhost:2181(CONNECTED) 1] ls /brokers/topics
[tweets02, tweets01]

# Détails sur les brokers gérés par zookeeper
[zk: localhost:2181(CONNECTED) 2] get /brokers/ids/1001
{"host":"192.168.1.80","port":9092,"version":4}
[zk: localhost:2181(CONNECTED) 3] get /brokers/ids/1002
{"host":"192.168.1.80","port":9091,"version":4}

# Connexion au serveur kafka
$ docker exec -it kafka01 /bin/bash

# Vérification du nombre de partitions et de replicas pour le topic tweets01
# ./kafka-topics.sh --describe --zookeeper zookeeper01:2181 --topic tweets01
Topic: tweets01      PartitionCount: 1      ReplicationFactor: 1
Configs:             Partition: 0    Leader: 1001   Replicas: 1001 Isr: 1001

# Augmentation du nombre de partitions du topic tweets01 à 3 partitions
# ./kafka-topics.sh --alter --zookeeper zookeeper01:2181 --topic tweets01 --partitions 3
Adding partitions succeeded!

# Vérification que le topic à 3 partitions réparties sur les deux brokers 1001 et 1002
# ./kafka-topics.sh --describe --zookeeper zookeeper01:2181 --topic tweets01
Topic: tweets01      PartitionCount: 3      ReplicationFactor: 1  Configs:
    Topic: tweets01      Partition: 0    Leader: 1001   Replicas: 1001 Isr: 1001
    Topic: tweets01      Partition: 1    Leader: 1002   Replicas: 1002 Isr: 1002
    Topic: tweets01      Partition: 2    Leader: 1001   Replicas: 1001 Isr: 1001

# Augmentation du nombre de replicas du topic tweets01 à 2 replicas
# vi increase-replicas.json
{
    "version":1,
    "partitions":[
        {"topic":"tweets01","partition":0,"replicas":[1001,1002]},
        {"topic":"tweets01","partition":1,"replicas":[1001,1002]},
        {"topic":"tweets01","partition":2,"replicas":[1001,1002]}
    ]
}
./kafka-reassign-partitions.sh --reassignment-json-file ./increase-replicas.json --execute
Successfully started reassignment of partitions.

# Vérification que le topic à 3 partitions et 2 replicas sur les brokers 1001 et 1002
# ./kafka-topics.sh --describe --zookeeper zookeeper01:2181 --topic tweets01
Topic: tweets01      PartitionCount: 3      ReplicationFactor: 2
Configs: Topic: tweets01 Partition:0 Leader:1001 Replicas:1001,1002 Isr:1001,1002
          Topic: tweets01 Partition:1 Leader:1002 Replicas:1001,1002 Isr:1002,1001
          Topic: tweets01 Partition:2 Leader:1001 Replicas:1001,1002 Isr:1001,1002
```

Code 9 : Passage à l'échelle de Kafka en augmentant le nombre de partitions

Note :

Un test de charge est présenté en Annexe 5. Il consiste à transférer dans Kafka un nombre important de tweets sur une courte période (1000 tweets en 1 minute) et de vérifier que Kafka est capable de les gérer sans perte d'information.

2.2.2 Passage à l'échelle de Spark

Présentation de Spark:

Apache Spark est un framework open source de calcul distribué qui permet de réaliser des analyses complexes à grande échelle. Une de ces principales qualités est sa vitesse d'exécution. Il est en effet très rapide car il peut exécuter des opérations d'analyse des données en mémoire et en temps réel, aussi bien qu'avec des données sur disque.

Son architecture est basé sur les principaux composants suivants :

- Apache Spark Core : Moteur d'exécution du framework. Il permet la répartition des tâches distribuées, le scheduling et des fonctionnalités de lectures/écritures de base.
- RDD. Ce sont des tables de données distribuées résilientes (Resilient Distributed Dataset) Chaque partition de données reste en mémoire sur son serveur de calcul entre deux itérations tout en gérant les principes de tolérance aux pannes. L'API RDD est implémentée sur Spark Core
- Spark SQL. C'est un composant qui vient au-dessus de la couche Core et qui introduit une nouvelle abstraction de données SchemaRDD, des données structurées et semi-structurées.
- Spark Streaming. C'est la partie traitement en pseudo temps réel d'Apache Spark en traitant les données sous forme de mini-batchs espacés par un instant T. L'API de Spark Streaming est basée sur Spark Core et construit des RDD de type spécifique (des Dstreams) mais les traitements qui seront faits sur les Dstreams sont identiques à ceux de Spark Core classique.
- MLLIB : C'est une librairie de Machine Learning distribuée et ses algorithmes sont conçus pour être exécutés sur un cluster de machines d'une manière distribuée.
- GraphX est un framework de traitement de graphe distribué sur Spark.

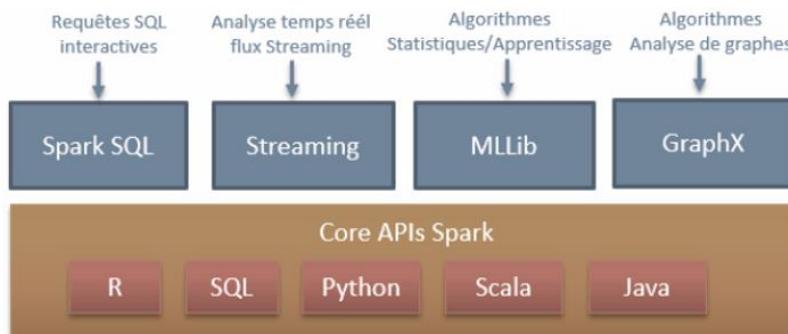


Figure 26: Composants de la stack technique de Spark

Spark s'exécute en mode maître esclave, c'est-à-dire un master et un ou plusieurs workers. Lorsqu'un traitement est lancé sur le framework Spark, on passe par le Driver qui communique avec le cluster manager. Celui-ci gère les ressources des workers qui vont exécuter par la suite le traitement demandé via des executors. Le Driver s'exécute dans une JVM qui héberge notamment le SparkContext qui est le point d'accès à toutes les fonctionnalités de Spark ainsi qu'une interface Web pour monitorer les traitements Spark. Le Driver pilote aussi la construction du graphe DAG (Directed Acyclic Graph) qui va optimiser les opérations de traitement Spark. Le schéma suivant montre l'architecture interne de Spark.

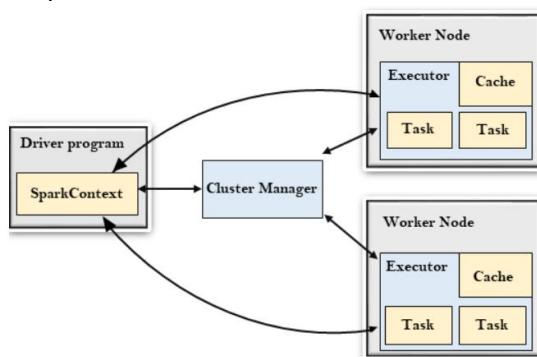


Figure 27: Architecture de Spark

Passage à l'échelle

Le passage à l'échelle peut se faire en augmentant les ressources par l'ajout de workers ou en optimisant les ressources déjà existantes.

- Optimisation :

Spark gère de nombreux paramètres qui permettent d'optimiser les ressources existentes d'un cluster Spark. Ils sont stockés en partie dans des fichiers de configuration « SparkConf ». On peut ainsi les modifier facilement pour customiser par exemple, le niveau de parallélisme des jobs Spark, les ressources qui leur sont allouées, etc. On peut aussi modifier certains paramètres directement au moment de l'initialisation du contexte Spark. Ainsi au lancement d'une session Spark Streaming, il est possible d'indiquer une durée de discréétisation des mini batchs en millisecondes. Cette durée indiquera la cadence à laquelle les minis-batchs seront produits. Couplés avec les fonctions avancées de fenêtrage de Spark, il est possible de passer à l'échelle rapidement. Enfin, Spark permet une allocation dynamique des ressources lorsque l'on positionne le paramètre « spark.dynamicAllocation.enable » à « true ». On laisse alors Spark définir lui-même les ressources à allouer dynamiquement à chaque job.

L'exemple suivant montre un extrait du fichier de configuration spark-defaults.conf avec les principaux paramètres qui permettent d'augmenter les ressources allouées aux executors qui exécutent les jobs sur les workers :

```
# Exemple de paramétrage des ressources allouées aux executors
spark.executor.memory=4g
spark.executor.cores=3
spark.executor.instances=3
```

Code 10: Extrait du fichier de configuration spark-defaults.conf

- Ajout de workers

Un cluster Spark permet d'exploiter la puissance de traitement de plusieurs machines. Le passage à l'échelle consiste à ajouter un ou plusieurs workers au sein du cluster Spark. Cet ajout est relativement simple puisqu'il suffit de déployer les nouveaux serveurs en indiquant le type de serveur souhaité et l'adresse du noeud maître. Il est à noter que pour une mise en production, il est recommandé de rendre le master résilient en utilisant par exemple ZooKeeper.

L'exemple suivant montre les commandes pour démarrer un serveur "Master" et un serveur "Worker":

```
# Déploiement d'un serveur Spark "Master"
./spark-2.4.4/bin/spark-class org.apache.spark.deploy.master.Master -h sparkmaster'

# Déploiement d'un nouveau serveur Spark "Worker"
./spark-class org.apache.spark.deploy.worker.Worker spark://sparkmaster:7077'
```

Code 11: Extrait du programme Spark docker-entrypoint.sh

- Failover

Lorsque l'un des workers tombe en panne, le cluster Spark est capable de rediriger tous les traitements sur les workers restants de manière transparente. La copie d'écran suivante montre comment une application reste toujours opérationnelle (State: Running), même après l'arrêt brutal de plusieurs workers (via la commande "docker kill") :

ExecutorID	Worker	Cores	Memory	State	Logs
0	worker-20200219101312-172.23.0.24-8881	4	1024	RUNNING	stdout stderr

ExecutorID	Worker	Cores	Memory	State	Logs
1	worker-20200219101312-172.23.0.22-8881	4	1024	LOST	stdout stderr
2	worker-20200219101312-172.23.0.23-8881	4	1024	LOST	stdout stderr

Figure 28: Exemple d'utilisation du Failover

Les commandes suivantes montrent les logs de démarrage du cluster Spark avec 3 workers :

```
# Vérification des conteneurs docker kafka et zookeeper
$ docker ps|grep -e spark
e2898ecd8dc6 "docker-entrypoint.sh" Up 0.0.0.0:8090->8080/tcp sparkmaster
e43451c5b1b2 "docker-entrypoint.sh" Up 0.0.0.0:8091->8081/tcp sparkworker1
55d01d30cb15 "docker-entrypoint.sh" Up 0.0.0.0:8092->8081/tcp sparkworker2
4ae482c17faa "docker-entrypoint.sh" Up 0.0.0.0:8093->8081/tcp sparkworker3

# Vérification des logs de démarrage
$ docker-compose logs|grep -e worker -e master
sparkmaster | 20/02/17 10:40:31 INFO Master: I have been elected leader! New state: ALIVE
sparkworker1 INFO Worker: Successfully registered with master spark://sparkmaster:7077
sparkworker2 INFO Worker: Successfully registered with master spark://sparkmaster:7077
sparkworker3 INFO Worker: Successfully registered with master spark://sparkmaster:7077
```

Code 12 : Passage à l'échelle de Spark en utilisant un cluster avec 3 workers

Spark possède aussi une interface graphique pour l'administration du cluster. Les copies d'écrans suivantes montrent la scalabilité dans Spark. On peut voir ainsi que l'application “viewparis” utilise bien l'ensemble des ressources du cluster Spark avec une répartition des calculs sur les 3 workers (executors 0,1 et 2) :

Figure 29: Console graphique d'administration du cluster Spark

Notes:

- Un test de charge est présenté en annexe 5. Il consiste à transférer dans Kafka un nombre important de tweets sur une courte période (1000 tweets en 1 minute) et à vérifier que Spark Streaming est capable de les traiter sans perte d'information.
- Un autre exemple est présenté en annexe 7 avec l'analyse de sentiments sur 200.000 tweets dans un temps très bref grâce au framework Spark (96 secondes).

2.2.3 Passage à l'échelle de MongoDB

Présentation de mongoDB:

MongoDB est une base documentaire qui gère les données au format JSON. MongoDB gère la réPLICATION et les reprises sur panne et possède un mode distribué pour le passage à l'échelle. Les principaux composants de mongoDb sont les suivants :

- Mongod: c'est le démon principal de mongoDB. Il gère l'accès aux données et effectue en tâche de fond des opérations de gestion de données.
- Mongo: c'est une ligne de commande interactive (mongo shell) pour exécuter des opérations sur la base de données.
- Mongos: c'est un routeur qui gère l'acheminement des requêtes dans environnement distribué. Lorsque les données sont distribuées sur des shards, il gère l'accès à celles-ci.
- ConfigServers: ils gèrent l'information sur la base de données (routeurs, shards, ReplicaSets ...)
- Shards: ils contiennent l'ensemble des données sous la forme de paquets appelés "chunks". La distribution des données sur plusieurs shards s'effectue au moyen du hashage d'une clé qui doit être présente dans l'ensemble des documents à distribuer.
- Replicaset : ce composant a pour but de garantir la haute disponibilité des données ainsi que leur réPLICATION. Un replica set est constitué d'un nœud maître (dit primaire) et de plusieurs nœuds esclaves (dits secondaires) et potentiellement un nœud arbitre. Un nœud du replica set devient le nœud primaire si il est élu à la majorité des nœuds secondaires qui composent avec lui le replica set. Une élection peut notamment avoir lieu si le nœud qui jusque-là était le nœud primaire tombe en panne, tarde à répondre, qu'un nouveau membre a été ajouté ou qu'une reconfiguration du replica set a lieu.

Le schéma suivant montre l'architecture MongoDB mise en oeuvre pour ce projet :

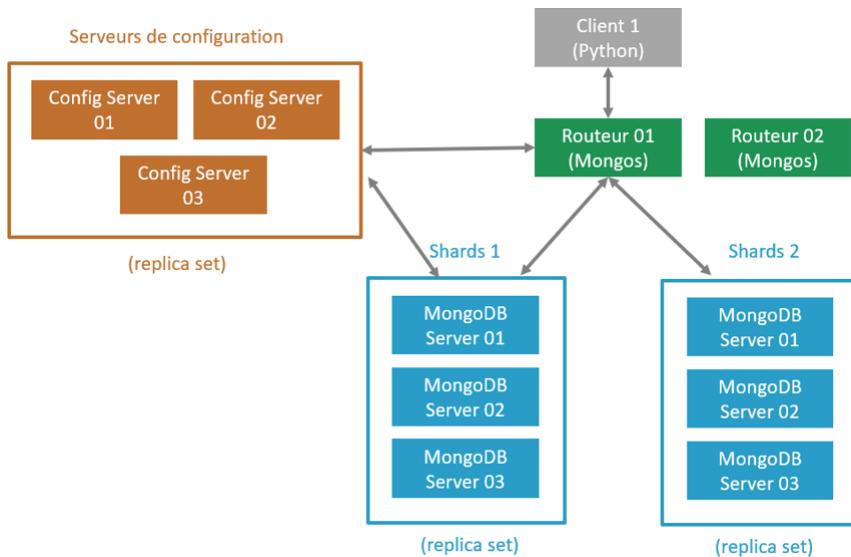


Figure 30: Architecture MongoDB

Passage à l'échelle

MongoDB utilise le sharding pour scalar la base de données (scalabilité horizontale). Le sharding distribue les données à travers les partitions physiques (shards) dont le cluster est composé. Chaque shard est composé d'un Replica Set.

Il est important de bien choisir la clé de sharding pour une répartition égale des documents insérés dans les différents shards. Ainsi les requêtes se feront sur de plus petits jeux de données et pourront être parallélisées, ce qui permet d'obtenir des temps de réponse plus rapides. Suivant l'augmentation des besoins, il est possible d'ajouter des serveurs supplémentaires sans interruption du service.

L'exemple suivant montre le statut du sharding du cluster mongoDB mis en oeuvre pour ce projet. Il montre aussi la bonne répartition des données d'apprentissage entre les deux shards (49%/51%) grâce à l'utilisation de l'identifiant du tweet comme clé de sharding.

```
# Vérification des conteneurs docker composant le cluster mongoDB
$ docker ps|grep mongo
6e39352e0022    "docker-entrypoint.sh" Up 0.0.0.0:27117->27017/tcp mongocfg11
251cf1e5f41d    "docker-entrypoint.sh" Up 0.0.0.0:27217->27017/tcp mongocfg12
7bae04685312    "docker-entrypoint.sh" Up 0.0.0.0:27317->27017/tcp mongocfg13
5815d6fc3356    "docker-entrypoint.sh" Up 0.0.0.0:28117->27017/tcp mongodb21
6c6e052a3d24    "docker-entrypoint.sh" Up 0.0.0.0:28217->27017/tcp mongodb22
4428016b7256    "docker-entrypoint.sh" Up 0.0.0.0:28317->27017/tcp mongodb23
ff98fb0e5f90    "docker-entrypoint.sh" Up 0.0.0.0:29117->27017/tcp mongodb31
1b53335c4654    "docker-entrypoint.sh" Up 0.0.0.0:29217->27017/tcp mongodb32
aada99d270dd    "docker-entrypoint.sh" Up 0.0.0.0:29317->27017/tcp mongodb33
6e1abaedb27    "docker-entrypoint.sh" Up 0.0.0.0:30117->27017/tcp mongos01
ea3c75f28934    "docker-entrypoint.sh" Up 0.0.0.0:30217->27017/tcp mongos02

#lancement du client mongo
$ docker exec -it mongos01 /bin/bash
$ mongo
mongos>

#vérification du statut du cluster
mongos> sh.status()
--- Sharding Status ---
shards:
  { "_id" : "rs2", "host" : "rs2/mongodb21:27018,mongodb22:27018", "state" : 1 }
  { "_id" : "rs3", "host" : "rs3/mongodb31:27018,mongodb32:27018", "state" : 1 }
active mongoses: 2
autosplit:
  Currently enabled: yes
balancer:
  Currently enabled: yes
databases: { "_id" : "parisdb", "primary" : "rs2", "partitioned" : true }

# connexion à la base parisdb
mongos> use parisdb
switched to db parisdb

# Vérification que la collection "apprentissage" est bien distribuée sur deux shards
mongos> db.apprentissage.getShardDistribution()

Shard rs3 at rs3/mongodb31:27018,mongodb32:27018
data : 416KiB docs : 1483 chunks : 2
estimated data per chunk : 208KiB
estimated docs per chunk : 741

Shard rs2 at rs2/mongodb21:27018,mongodb22:27018
data : 422KiB docs : 1517 chunks : 2
estimated data per chunk : 211KiB
estimated docs per chunk : 758

Totals
data : 838KiB docs : 3000 chunks : 4
Shard rs3 contains 49.61% data, 49.43% docs in cluster, avg obj size on shard : 287B
Shard rs2 contains 50.38% data, 50.56% docs in cluster, avg obj size on shard : 285B
```

Code 13: Passage à l'échelle de MongoDB avec deux shards

Conclusion

Bilan du projet

Ce projet m'a permis de mettre en pratique les concepts généraux de l'apprentissage machine et de vérifier leur efficacité sur l'analyse des tweets des sorties culturelles à Paris. Ainsi, j'ai pu tester de nombreux algorithmes de classification. La meilleure solution s'est révélée être la combinaison des trois modèles Spark suivants :

- LogisticRegression
- NaiveBayes
- OnevsRest

Le taux de réussite (accuracy) est supérieur à 85% sur les données de validation, ce qui est remarquable. Il faut cependant nuancer ces bons résultats :

- La capacité des outils à détecter automatiquement le caractère positif, neutre ou négatif d'un message dépend fortement de la qualité intrinsèque des messages analysés. Pour ce projet, tous les tweets traitent d'une même thématique. Ils forment ainsi un corpus de données cohérent qui permet d'identifier plus facilement la polarité des messages.
- On trouve des prédictions anormales. Les opinions des internautes s'expriment non seulement par les mots employés, mais aussi par le ton et l'utilisation d'une typographie particulière. Cela rend difficile l'analyse fine des sentiments.

La préparation des données, la qualité de la base d'apprentissage et le choix du modèle prédictif sont donc déterminants pour obtenir un haut niveau de prédiction.

J'ai pu aussi découvrir de nouvelles technologies comme Kafka, MongoDB et Spark Streaming. Elles permettent aujourd'hui de travailler en temps réel sur des données massives en optimisant au mieux les ressources utilisées (cpu, mémoire et disque).

En utilisant un outil de conteneurisation Docker, j'ai pu tester le passage à l'échelle ainsi que la haute disponibilité de ces trois technologies. Ceci m'a permis d'exécuter des traitements en temps réel ainsi que des traitements de fond dans des temps très réduits, malgré la grande quantité de textes à analyser (voir exemples en annexes 5 et 7).

Perspectives

Ce projet est un bon exemple de ce que peut permettre les algorithmes d'apprentissage machine couplés à des technologies pouvant gérer le passage à l'échelle des traitements et des données. Avec l'avènement des réseaux sociaux et la mise à disposition d'un nombre toujours croissant de données textuelles sur le Web, il est certain que l'intérêt pour l'analyse en temps réel de données va s'accroître rapidement.

Bilan personnel

Au cours de ce projet, j'ai pu travailler sur de nombreuses technologies particulièrement intéressantes. L'optimisation de certains modèles s'est révélée quelquefois ardue mais au final l'expérience a été très enrichissante et j'espère pouvoir continuer dans ce domaine à l'avenir.

Annexes

Annexe 1 : Liste des figures

Figure 1: Liste des étapes du projet	1
Figure 2: Système de messagerie Kafka.....	2
Figure 3: Spark Streaming.....	3
Figure 4: Opérations de fenêtrage avec Spark Streaming	3
Figure 5: Affichage en continu des sujets les plus discutés sur Twitter	4
Figure 6: Répartition des avis dans la base d'apprentissage	5
Figure 7 : mots les plus fréquents dans les tweets négatifs (en rouge), neutres (en gris) et positifs (en vert) .	5
Figure 8: Histogramme des longueurs des tweets selon le sentiment exprimé	5
Figure 9: Volume de tweets échangés au démarrage de l'exposition de Léonard de Vinci.....	6
Figure 10: Exemples de répartition des tweets selon la langue	7
Figure 11: Exemples d'évolution des longueurs des tweets	7
Figure 12: Synthèse des méthodes d'apprentissage machine.....	10
Figure 13: Algorithme des forêts aléatoires	12
Figure 14: Différences entre la régression linéaire et la régression logistique binaire.....	12
Figure 15 : Algorithme One-vs-Rest	13
Figure 16: Extrait du programme Twitter-08-NLP-Models-Comparaison.ipynb	15
Figure 17: Comparaison des courbes ROC pour plusieurs modèles	16
Figure 18: Superposition des courbes ROC de la Régression Logistique Multinomiale	17
Figure 19: Corrélation entre les modèles testés.....	17
Figure 20: Visualisation des résultats sous la forme de pages Web	19
Figure 21: Architecture générale du projet	20
Figure 22: Architecture Technique du projet	20
Figure 23: Liste des outils progiciels/frameworks utilisés dans le projet	21
Figure 24: Traitements du projet	22
Figure 25: Passage à l'échelle de Kafka	23
Figure 26: Composants de la stack technique de Spark	25
Figure 27: Architecture de Spark.....	25
Figure 28: Exemple d'utilisation du Failover	26
Figure 29: Console graphique d'administration du cluster Spark	27
Figure 30: Architecture MongoDB	28
Figure 31: Résultat du programme Twitter-12-Test-de-Charge-Kafka-SparkStreaming-MongoDB-Producer	34
Figure 32: résultat du programme Twitter-13-Test-de-Charge-Kafka-SparkStreaming-MongoDB-Consumer	34
Figure 33: Résultat du programme Twitter-05_Streaming-WordCloud-Loop.....	35
Figure 34: Affichage en continu des mots-clés au sein d'une page web	35
Figure 35: Liste des traitements du projet	42

Annexe 2 : Liste des tableaux

Table 1: Classement des principales activités culturelles par thématique	6
Table 2: Mesures d'évaluation des modèles de classification.....	15
Table 3: Comparaison des modèles testés	16
Table 4 : Liste des conteneurs Docker lancés pour la mise en oeuvre du projet.....	21
Table 5: Description des traitements du projet	22
Table 6: Liste des traitements	33

Annexe 3 : Liste des extraits de code

Code 1: Extrait du programme Twitter-03-Streaming-KafkaProducer-Loop.ipnbj.....	3
Code 2: Extrait du programme Twitter-04-Streaming-SparkConsumer-loop.ipnbj.....	4
Code 3 : Extrait du code source le page Web bagofwords.html	4
Code 4: Extrait du programme Twitter-04-Streaming-SparkConsumer-loop.ipnbj.....	8
Code 5 : Extrait du programme Twitter-09-NLP-Sentiment-Analysis.ipnbj	9
Code 6: Extrait du programme Twitter-09-NLP-Sentiment-Analysis.ipnbj	10
Code 7: Extrait du programme Twitter-08-NLP-Models-Comparaison.ipnbj	14
Code 8: Extrait du programme Twitter-11-WebSite-Creation.ipnbj	18
Code 9 : Passage à l'échelle de Kafka en augmentant le nombre de partitions	24
Code 10: Extrait du fichier de configuration spark-defaults.conf	26
Code 11: Extrait du programme Spark docker-entrypoint.sh	26
Code 12 : Passage à l'échelle de Spark en utilisant un cluster avec 3 workers	27
Code 13: Passage à l'échelle de MongoDB avec deux shards.....	29
Code 14: Résultat Twitter-09-NLP-Sentiment-Analysis.ipynb.....	39

Annexe 4 : Contenu du fichier archive du projet

Le fichier archive « UASB03_Projet_Feraudet_Fev2020.zip » déposé sur le site moodle contient les fichiers suivants :

- Le rapport de ce projet (format pdf) :
[*UASB03_Projet_Feraudet_v1.0.pdf*](#)
- La base de données d'apprentissages (3000 Tweets annotés) au format .csv :
[*base_apprentissage_3000tweets.csv*](#)
- Répertoire Docker: Ensemble des fichiers de création des conteneurs Docker du projet
(Seuls les fichiers Dockerfile et les scripts shell sont fournis, pour des problèmes de place)
- Répertoire jupyter_notebook : Code python du projet + exemples d'exécution intégrés dans les notebooks jupyter

Table 6: Liste des traitements

N°	Programme	Description
1	Twitter-01-Apprentissage-Upload-Data.ipynb	Chargement des données d'apprentissage dans MongoDB (3000 Tweets annotés)
2	Twitter-02-Apprentissage-Exploration-Data.ipynb	Exploration des données d'apprentissage (Histogrammes, nuages de mots, etc.)
3	Twitter-03-Streaming-KafkaProducer-Loop.ipynb	Streaming des Tweets issus de Twitter avec kafka
4	Twitter-04-Streaming-SparkConsumer-loop.ipynb	Analyse en continu des Tweets collectés dans Spark Streaming et archivage dans MongoDB
5	Twitter-05_Streaming-WordCloud-Loop.ipynb	Génération d'un nuage de mots avec les résultats de l'analyse des tweets collectés de puis Kafka
6	Twitter-06-Streaming-Tweets-Data-Extraction.ipynb	Extraction et nettoyage des données textuelles des Tweets pour les insérer dans une collection MongoDB
7	Twitter-07-Batch-Tweets-Upload-Loop.ipynb	Initialisation de l'historique des Tweets avec des Tweets provenant des sites culturels officiels
8	Twitter-08-NLP-Models-Comparaison.ipynb	Comparaison de plusieurs modèles pour l'analyse de sentiments afin de sélectionner le meilleur modèle
9	Twitter-09-NLP-Sentiment-Analysis.ipynb	Analyse de sentiments des tweets collectés avec le meilleur modèle sélectionné précédemment
10	Twitter-10-WebSite-Summary-Creation.ipynb	Synthèse des analyses statistiques pour un affichage dans des pages Web
11	Twitter-11-WebSite-Creation.ipynb	Affichage dynamique et en continu des nuages de mots analysés dans Spark Streaming
12	Twitter-12-Test-de-Charge-Kafka-SparkStreaming-MongoDB-Producer	Test de charge : Streaming de 1000 tweets en 1 minute avec Kafka
13	Twitter-13-Test-de-Charge-Kafka-SparkStreaming-MongoDB-Consumer	Test de charge: Analyse en continu des 1000 tweets avec Spark Streaming et mongoDB

Note:

- Une courte vidéo de 2m20 présentant les résultats d'un test de charge de Kafka, SparkStreaming et mongoDB (traitement en continu de 1000 tweets sur une période d'1 minute) a été déposée dans moodle :

[*video_test_de_charge_temps_réel_kafka_SparkStreaming_MongoDB.mp4*](#)

Annexe 5: Test de charge pour l'analyse en continu des tweets

Ce test consiste à générer rapidement un grand nombre de Tweets sur une courte période (1000 tweets en 1 minute) et à vérifier le bon traitement de ces tweets dans Kafka, Spark Streaming et mongoDB. Une vidéo présentant les résultats de ce test est disponible dans Moodle (cf Annexe 4)

Etape N°1: Génération de 1000 tweets en 1 minute et transfert des tweets dans le topic Kafka "Tweets01"

```
Programme: Twitter-12-Test-de-Charge-Kafka-SparkStreaming-MongoDB-Producer
*****
Date: : 2020-02-21 14:15:33

Début du test de charge : 1000 tweets en 1 minute
*****
1 Time: 2020-02-21 14h:15m:33s:249936 ==> New Tweet! len: 9757
2 Time: 2020-02-21 14h:15m:33s:288322 ==> New Tweet! len: 2417
3 Time: 2020-02-21 14h:15m:33s:322596 ==> New Tweet! len: 3626
4 Time: 2020-02-21 14h:15m:33s:340940 ==> New Tweet! len: 7388
5 Time: 2020-02-21 14h:15m:33s:405010 ==> New Tweet! len: 4530
6 Time: 2020-02-21 14h:15m:33s:423329 ==> New Tweet! len: 7715
...
...
999 Time: 2020-02-21 14h:16m:26s:549014 ==> New Tweet! len: 7136
1000 Time: 2020-02-21 14h:16m:26s:639655 ==> New Tweet! len: 3531

fin du traitement
*****
Temps d'exécution : 53 secondes
Nombre de tweets streamés: 1000
longueur totale de tweets streamés: 6891330 caractères.
```

Figure 31: Résultat du programme Twitter-12-Test-de-Charge-Kafka-SparkStreaming-MongoDB-Producer

Etape N°2: Collecte et analyse statistique en continu des mots-clés (#hashtags) contenus dans les tweets

```
Programme: Twitter-13-Test-de-Charge-Kafka-SparkStreaming-MongoDB-Consumer
*****
Date: : 2020-02-21 14:15:27

-----
Time: 2020-02-21 14:15:45
-----
('#MuseeRodin', 15)
('#pantheon', 15)
('#Rodin', 15)
('#EiffelTower', 13)
('#Paris', 7)
...
-----
Time: 2020-02-21 14:15:46
-----
Tweets in this batch: 19
-----
Time: 2020-02-21 14:15:47
-----
('#pantheon', 15)
('#AlUla', 12)
('#ExpoGreco', 7)
('#SainteChapelle', 6)
('#Vincennes', 6)
...
fin du traitement
*****
Temps d'exécution : 65 secondes
Nombre de tweets analysés: 1000
longueur totale des tweets collectés: 6891330 caractères.
```

Figure 32: Résultat du programme Twitter-13-Test-de-Charge-Kafka-SparkStreaming-MongoDB-Consumer

Etape N°3: Génération d'un nuages de mots-clés toutes les 5 secondes

```

Programme: Twitter-05_Streaming-WordCloud-Loop
*****
Date: : 2020-02-21 14:15:37

loop: 0 ( 21/02/2020 14:15:37 ) #StadeDeFrance #cityscape #cityscape #PetitPalais ...
loop: 1 ( 21/02/2020 14:15:40 ) #Tuileries, #Tuileries, #AlUla, #AlUla, #A ...
loop: 2 ( 21/02/2020 14:15:42 ) #instadisney #WinterAtTantora! #love #DisneyCharac ...
loop: 3 ( 21/02/2020 14:15:44 ) #Food #Food #Museums #Museums #Denmark #Denmark #V ...
loop: 4 ( 21/02/2020 14:15:47 ) #bibs2paris #tragique #tragique #BibParis #ExpoMag ...
loop: 5 ( 21/02/2020 14:15:49 ) #ArtGram #Kunst #Sant #Dessins #Accessibilit #Flas ...
loop: 6 ( 21/02/2020 14:15:52 ) #ducktales #ducktales #FreeNasrin #H #H #Tr #Tr #f ...
loop: 7 ( 21/02/2020 14:15:54 ) #CinemaChinoisFC #poesiephilosophique #THDV13EmeAr ...
loop: 8 ( 21/02/2020 14:15:56 ) #Num #5G #IA #VR #Histoire #Patrimoine #VeronicaMa ...
loop: 9 ( 21/02/2020 14:15:59 ) #Chaillot #CeJourLa #CeJourLa #Th #Coup #disneysid ...
loop: 10 ( 21/02/2020 14:16:01 ) #Tolkien, #Digital #AvecPoleEmploi #CNotrePerf #Em ...
loop: 11 ( 21/02/2020 14:16:04 ) #CIRCONSCRIPTION #CIRCONSCRIPTION #jardindestuiler ...
loop: 12 ( 21/02/2020 14:16:06 ) #costume #Valentine #ArtsPlastiques #ArtsPlastique ...
loop: 13 ( 21/02/2020 14:16:08 ) #Lifestyle #NaaniiGlobal #MayLeaveStarsTWC #MayLea ...
loop: 14 ( 21/02/2020 14:16:11 ) #UneRegionDAvenir #UneRegionDAvenir #UneRegionDAve ...
loop: 15 ( 21/02/2020 14:16:13 ) #Exposition #artvideo #Medias #LogementSocial #tea ...
loop: 16 ( 21/02/2020 14:16:16 ) #Brest #Poitiers #Saintbrieuc #saintherblain #Disn ...
loop: 17 ( 21/02/2020 14:16:18 ) #SpiderManFarFromHome #cadeau #BD2020 #BD2020 #BD2 ...
loop: 18 ( 21/02/2020 14:16:21 ) #franceverte #pagedustoner #paris10 #Sunday #Sunda ...
loop: 19 ( 21/02/2020 14:16:23 ) #AzimuthFestival #AzimuthFestival #AzimuthFestival ...
loop: 20 ( 21/02/2020 14:16:25 ) #ProGamer #ProGamer #ProGamer #ProGamer ...
loop: 21 ( 21/02/2020 14:16:28 ) #MoulinRouge; #LundiClassique #LundiClassique #Clu ...
loop: 22 ( 21/02/2020 14:16:30 ) #expoTolkienBnF, #expoTolkienBnF, #expoTolkienBnF, ...

fin du traitement
*****
Temps d'execution : 66 secondes

```

Figure 33: Résultat du programme Twitter-05_Streaming-WordCloud-Loop

Etape N°4: Affichage du nuage de mots-clés toutes les 5 secondes au sein d'une page Web

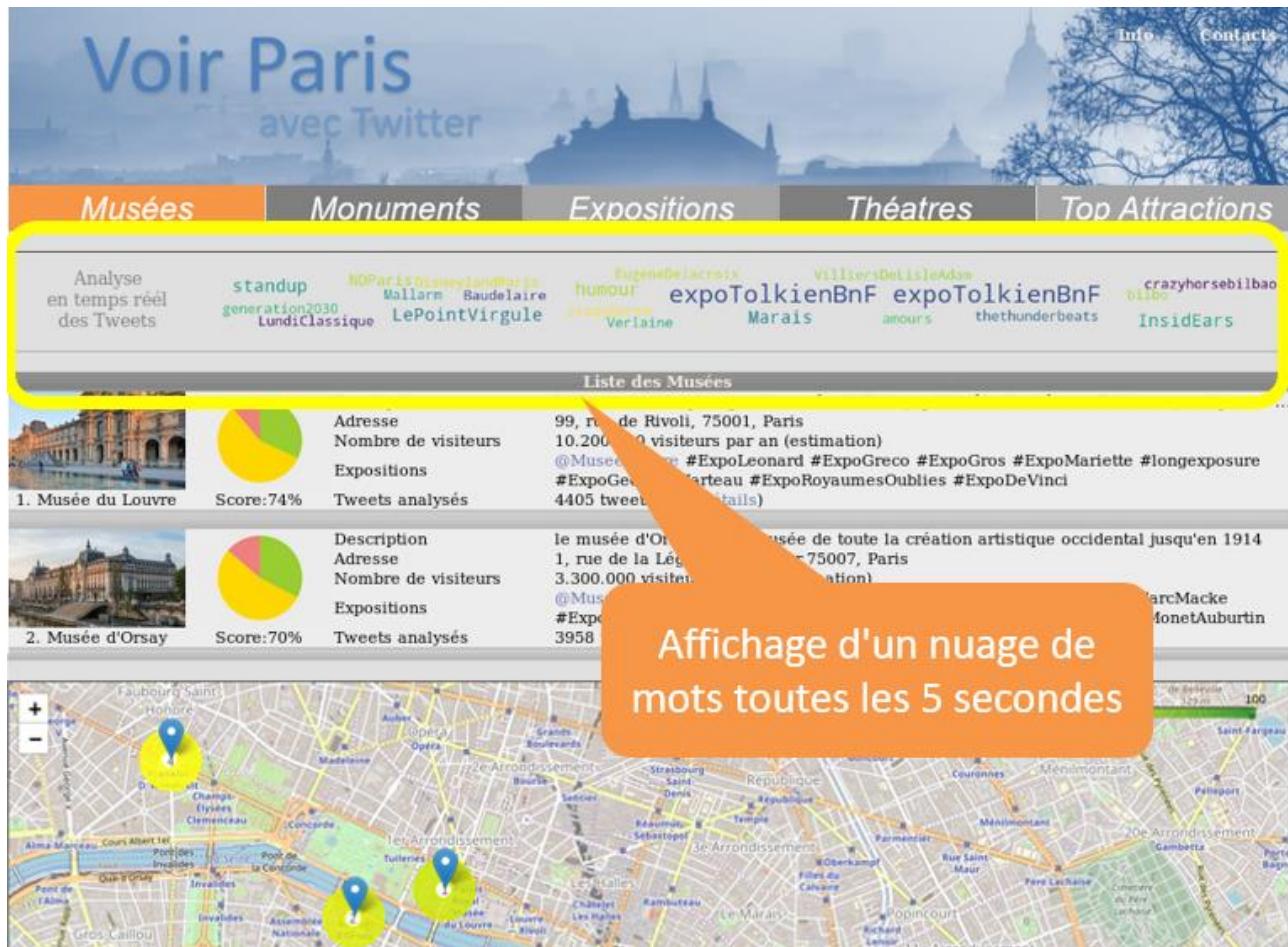
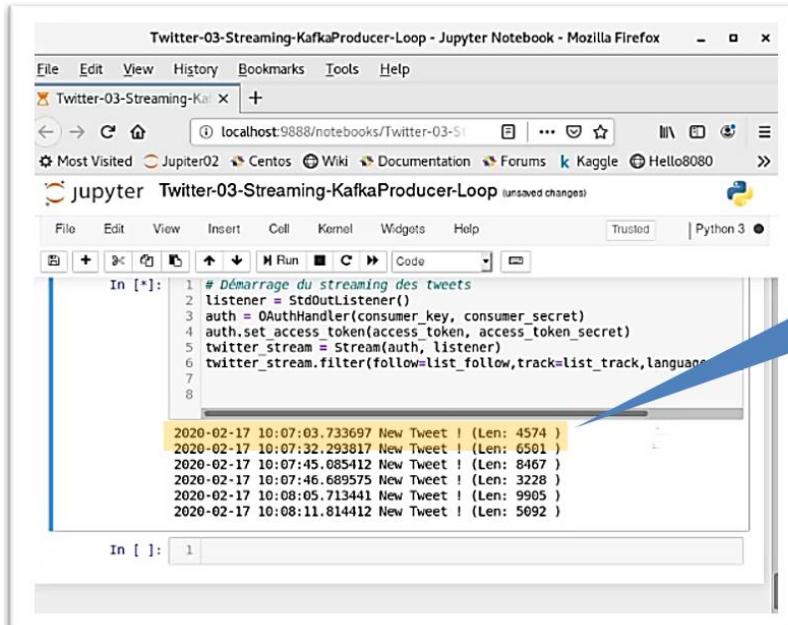


Figure 34: Affichage en continu des mots-clés au sein d'une page web

Annexe 6: Copies d'écrans sur l'analyse en continu des tweets

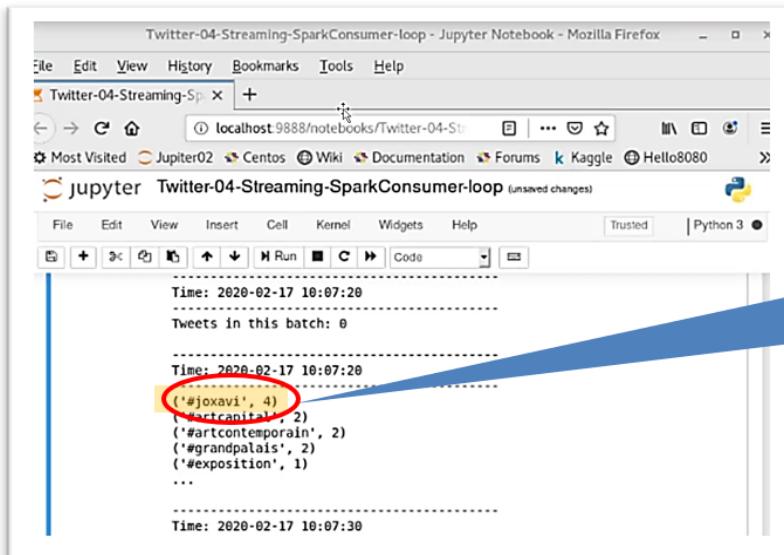
Les écrans suivants montrent l'analyse en continu des tweets collecté depuis Twitter:



```
Twitter-03-Streaming-KafkaProducer-Loop - Jupyter Notebook - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Twitter-03-Streaming-KafkaProducer-Loop x +
localhost:9888/notebooks/Twitter-03-Streaming-KafkaProducer-Loop
Most Visited Jupiter02 Centos Wiki Documentation Forums Kaggle Hello8080
Jupyter Twitter-03-Streaming-KafkaProducer-Loop (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
In [*]: 1 # Démarrage du streaming des tweets
2 listener = StdOutListener()
3 auth = OAuthHandler(consumer_key, consumer_secret)
4 auth.set_access_token(access_token, access_token_secret)
5 twitter_stream = Stream(auth, listener)
6 twitter_stream.filter(follow=list_follow, track=list_track, language='fr')
7
8
2020-02-17 10:07:03.733697 New Tweet ! (Len: 4574 )
2020-02-17 10:07:32.293817 New Tweet ! (Len: 6501 )
2020-02-17 10:07:45.085412 New Tweet ! (Len: 8467 )
2020-02-17 10:07:46.689575 New Tweet ! (Len: 3228 )
2020-02-17 10:08:05.713441 New Tweet ! (Len: 9905 )
2020-02-17 10:08:11.814412 New Tweet ! (Len: 5092 )

In [ ]: 1
```

Etape N°1
Collecte en temps réel
des Tweets dans Kafka



```
Twitter-04-Streaming-SparkConsumer-loop - Jupyter Notebook - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Twitter-04-Streaming-SparkConsumer-loop x +
localhost:9888/notebooks/Twitter-04-Streaming-SparkConsumer-loop
Most Visited Jupiter02 Centos Wiki Documentation Forums Kaggle Hello8080
jupyter Twitter-04-Streaming-SparkConsumer-loop (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
Time: 2020-02-17 10:07:20
-----
Tweets in this batch: 0
-----
Time: 2020-02-17 10:07:20
#joxavi: 4
#artcapital: 2
#artcontemporain: 2
#grandpalais: 2
#exposition: 1
...
-----
Time: 2020-02-17 10:07:30
```

Etape N°2
Analyse en continu des
hashtags contenus
dans les Tweets et
comptage

Twitter-05_Streaming-WordCloud-Loop - Jupyter Notebook - Mozilla Firefox

File Edit View History Bookmarks Tools Help

Twitter-05_Streaming-W x +

localhost:9888/notebooks/Twitter-05_...

Most Visited Jupiter02 Centos Wiki Documentation Forums Kaggle Hello8080

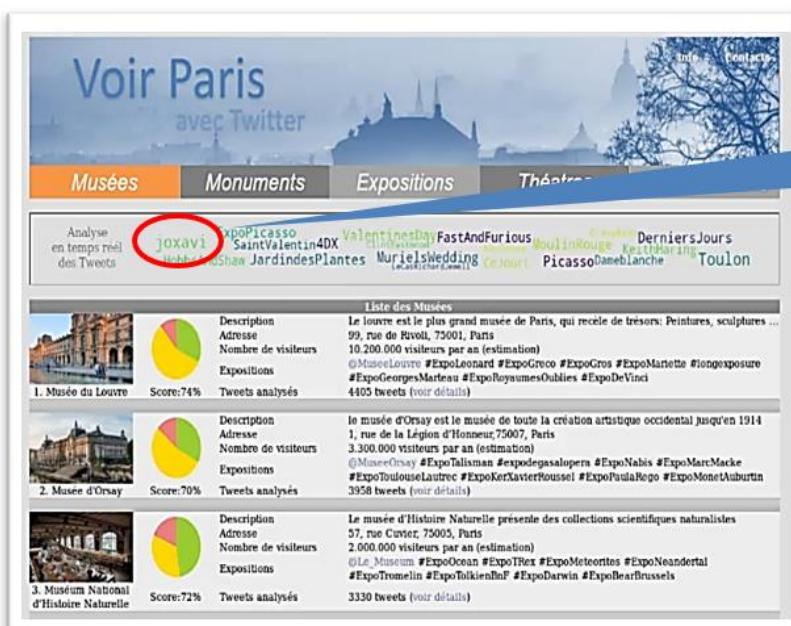
jupyter Twitter-05_Streaming-WordCloud-Loop (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
Trance #quest ...
loop: 7 ( 17/02/2020 10:07:27 )      #exposition #joxavi #joxavi #joxavi
#joxavi #artca ...
loop: 8 ( 17/02/2020 10:07:32 )      #exposition #joxavi #joxavi #joxavi
#joxavi #artca ...
loop: 9 ( 17/02/2020 10:07:37 )      #exposition #joxavi #joxavi #joxavi
#joxavi #artca ...
loop: 10 ( 17/02/2020 10:07:43 )     #PhotoDuJour #PhotoDuJour #dessin #
dessin #Renaiss ...
loop: 11 ( 17/02/2020 10:07:48 )     #PhotoDuJour #PhotoDuJour #dessin #
dessin #Renaiss ...
loop: 12 ( 17/02/2020 10:07:53 )     #PhotoDuJour #PhotoDuJour #dessin #
dessin #Renaiss ...
loop: 13 ( 17/02/2020 10:07:58 )     #PhotoDuJour #PhotoDuJour #dessin #
dessin #Renaiss ...
```

In []: 1

Etape N°3
Création d'un nuage de mots en continu



Etape N°4
Affichage du nuage de mots en continu

Annexe 7: Exemple d'Analyse de Sentiments sur 200.000 tweets

L'exemple suivant montre l'exécution du traitement d'analyse de sentiments sur environ 200.000 tweets :

```
Programme: Twitter-09-NLP-Sentiment-Analysis
*****
Date: : 2020-02-20 15:40:07

Versions des librairies utilisées
*****
python version: 3.6.9
Spark NLP version: 2.3.6
Spark version: 2.4.4

lecture collection tweets
*****
Nombre de tweets à analyser: 184884

lecture collection apprentissage
*****
negative: 1000 neutral: 1000 positive: 1000
Base d'apprentissage: 3000 tweets

concaténation des données
*****
Total Tweets: 187884

Préparation des données
*****
Passage en minuscules
Suppression des chiffres
Suppression des textes vides
Tokenisation du texte
Suppression des Stop Words
Suppression des mots les plus communs
Suppression des mots les plus rares
stopwordList_leastcommon: 39664 mots
Normalisation des données
Calcul TF/IDF
Split des données d'apprentissage (80/20%)
Données d'entraînement: 2348
Données de validation: 651
Données de test: 184884

fin de la préparation des données
*****
Temps pour la préparation des données: 75 secondes

Modele: Multinomial logistic regression
*****
DenseMatrix([[188., 22., 12.],
 [10., 185., 5.],
 [10., 27., 192.]])
LR classifier Accuracy (test) = 0.867896
LR classifier f1 (test) = 0.868502
LR classifier weightedPrecision (test) = 0.874265
LR classifier weightedRecall (test) = 0.867896
LR classifier Accuracy (train) = 0.98552
LR classifier f1 (train) = 0.985533
LR classifier weightedPrecision (train) = 0.985642
LR classifier weightedRecall (train) = 0.98552

Analyse des 184884 tweets avec le modèle de régression logistique multinomiale
*****
+-----+-----+-----+
| tweet_id| tweet_text| probability|prediction|
+-----+-----+-----+
|T100000232725489642|être trash être p...|[0.00962850611934...| 2.0|
|T1000002446930972673|immersion dans un...|[0.22726014039573...| 1.0|
|T1000002557979262976|quand réalise por...|[0.31728822710481...| 1.0|
|T1000006695597330432|promettre pense p...|[0.15299980774140...| 2.0|
+-----+-----+-----+

fin d'exécution du modèle de régression logistique multinomiale
*****
Temps d'exécution du modèle : 5 secondes
```

```

Modele: Naive Bayes
*****
Date: : 2020-02-20 22:39:33

NB classifier Accuracy (test) = 0.829493
NB classifier f1 (test) = 0.83004
NB classifier weightedPrecision (test) = 0.831441
NB classifier weightedRecall (test) = 0.829493

NB classifier Accuracy (train) = 0.930579
NB classifier f1 (train) = 0.9306
NB classifier weightedPrecision (train) = 0.930906
NB classifier weightedRecall (train) = 0.930579

Analyse des 184884 tweets avec le modèle Naive Bayes
*****
+-----+-----+-----+
| tweet_id| tweet_text| probability|prediction|
+-----+-----+-----+
|T1000002327254896642|être trash être p...|[0.00962850611934...| 2.0|
|T1000002446930972673|immersion dans un...|[0.22726014039573...| 1.0|
|T1000002557979262976|quand réalise por...|[0.31728822710481...| 1.0|
|T1000006695597330432|promettre pense p...|[0.15299980774140...| 2.0|
+-----+-----+-----+

fin d'exécution du modèle Naive Bayes
*****
Temps d'exécution du modèle : 3 secondes

Modele: One-vs-Rest
*****
Date: : 2020-02-20 22:42:02

OV classifier Accuracy (test) = 0.81874
OV classifier f1 (test) = 0.819088
OV classifier weightedPrecision (test) = 0.819899
OV classifier weightedRecall (test) = 0.81874

OV classifier Accuracy (train) = 0.995315
OV classifier f1 (train) = 0.995316
OV classifier weightedPrecision (train) = 0.995343
OV classifier weightedRecall (train) = 0.995315

Analyse des 184884 tweets avec le modèle OnevsRest
*****
+-----+-----+-----+
| tweet_id| tweet_text| probability|prediction|
+-----+-----+-----+
|T1000002327254896642|être trash être p...|[0.33852311344471...| 0.0|
|T1000002446930972673|immersion dans un...|[0.28255105372300...| 1.0|
|T1000002557979262976|quand réalise por...|[0.39774959708692...| 0.0|
|T1000006695597330432|promettre pense p...|[0.26439611479797...| 2.0|
+-----+-----+-----+

fin d'exécution du modèle OnevsRest
*****
Temps d'exécution du modèle : 11 secondes

Concaténation des résultats des 4 meilleurs modèles
*****
+-----+-----+-----+-----+-----+
| tweet_id| tweet_text|LR_prediction|NB_prediction|OV_prediction|class|
+-----+-----+-----+-----+-----+
|1000002327254896642|être trash être p...| 2.0| 2.0| 2.0| 2.0|
|1000002446930972673|immersion dans un...| 1.0| 1.0| 1.0| 1.0|
|1000002557979262976|quand réalise por...| 1.0| 1.0| 2.0| 1.0|
|1000006695597330432|promettre pense p...| 2.0| 2.0| 2.0| 2.0|
|1000011665239347200|nomination molièr...| 0.0| 0.0| 0.0| 0.0|
+-----+-----+-----+-----+-----+

fin du traitement de l'analyse de sentiments
*****
Temps d'exécution total : 96 secondes

```

Code 14: Résultat Twitter-09-NLP-Sentiment-Analysis.ipynb

Annexe 8: Comparaison des modèles sur la base d'apprentissage

L'exemple suivant montre le résumé de l'exécution du traitement de comparaison des modèles sur la base d'apprentissage composée de 3000 tweets déjà annotés :

```
Programme: Twitter-08-NLP-Models-Comparaison
*****
Date: : 2020-02-21 22:08:43

Versions des librairies utilisées
*****
python version: 3.6.9
numpy version: 1.18.1
pandas version: 0.25.3
Spark NLP version: 2.3.6

lecture de la base d'apprentissage
*****
3000 tweets

Préparation des données
*****
Passage en minuscules
Suppression des chiffres
Suppression des textes vides
Tokenisation du texte
Suppression des Stop Words
Liste des mots les plus communs:
Mots les plus communs: ['être', 'pour', 'vous', 'avoir', 'sur', 'dans',
'heure', 'son', 'par', 'tout', 'nous', 'notre', 'que', 'qui']
Suppression des mots les plus rares
Normalisation des données
Calcul TF/IDF
Split des données d'apprentissage (80/20%)

fin de la préparation des données
*****
Temps pour la préparation des données: 21 secondes

Comparaison des performances des modèles
*****
Modele: Multinomial logistic regression
*****
LR classifier Accuracy = 0.852535
LR classifier f1 = 0.852914
LR classifier weightedPrecision = 0.858473
LR classifier weightedRecall = 0.852535

Modele: Decision tree classifier
*****
DT classifier Accuracy = 0.737327
DT classifier f1 = 0.73521
DT classifier weightedPrecision = 0.778464
DT classifier weightedRecall = 0.737327

Modele: Random forest
*****
RF classifier Accuracy = 0.806452
RF classifier f1 = 0.806532
RF classifier weightedPrecision = 0.814948
RF classifier weightedRecall = 0.806452

Modele: Naive Bayes
*****
NB classifier Accuracy = 0.827957
NB classifier f1 = 0.827429
NB classifier weightedPrecision = 0.828496
NB classifier weightedRecall = 0.827957
```

```

Modele: One-vs-Rest
*****
OV classifier Accuracy = 0.817204
OV classifier f1 = 0.816861
OV classifier weightedPrecision = 0.819094
OV classifier weightedRecall = 0.817204

Modele: textBlob
*****
TB classifier Accuracy = 0.8079877112135176

Modèle Ensemble N°1 (LR+NB+OV+RF)
*****
ENS1 Accuracy = 0.855606758832565

Modele: Ensemble N°2 (LR+NB+OV)
*****
ENS2 Accuracy = 0.857142857142857

Modele: Ensemble N°3 (LR+NB+OV+RF+TB)
*****
ENS3 Accuracy = 0.877112135176651

Modele: Ensemble N°4 (LR+NB+TB)
*****
ENS4 Accuracy = 0.883256528417818

Modele: Ensemble N°5 (LR+NB+RF+TB)
*****
ENS5 Accuracy = 0.890937019969278

Synthèse des résultats
*****
DT Accuracy : ( 480 / 651 ) => 73.73%
RF Accuracy : ( 525 / 651 ) => 80.64%
TB Accuracy : ( 526 / 651 ) => 80.79%
OV Accuracy : ( 532 / 651 ) => 81.72%
NB Accuracy : ( 539 / 651 ) => 82.79%
LR Accuracy : ( 555 / 651 ) => 85.25%
ENS1 Accuracy: ( 557 / 651 ) => 85.56%
ENS2 Accuracy: ( 558 / 651 ) => 85.71%
ENS3 Accuracy: ( 571 / 651 ) => 87.71%
ENS4 Accuracy: ( 575 / 651 ) => 88.32%
ENS5 Accuracy: ( 580 / 651 ) => 89.09%

Fin du traitement
*****
Temps d'exécution total : 225 secondes
Temps d'exécution total : 4 minutes

```

Code 15: Résultat du programme Twitter-08-NLP-Models-Comparaison.ipynb

Note:

Le détail des performances pour chacun des modèles (accuracy, F1, recall, precision) sont donnés dans le notebook jupyter “Twitter-08-NLP-Models-Comparaison.ipynb”

Annexe 9: Ordonnancement des traitements

Le schéma suivant montre l'ordonnancement global des traitements du projet

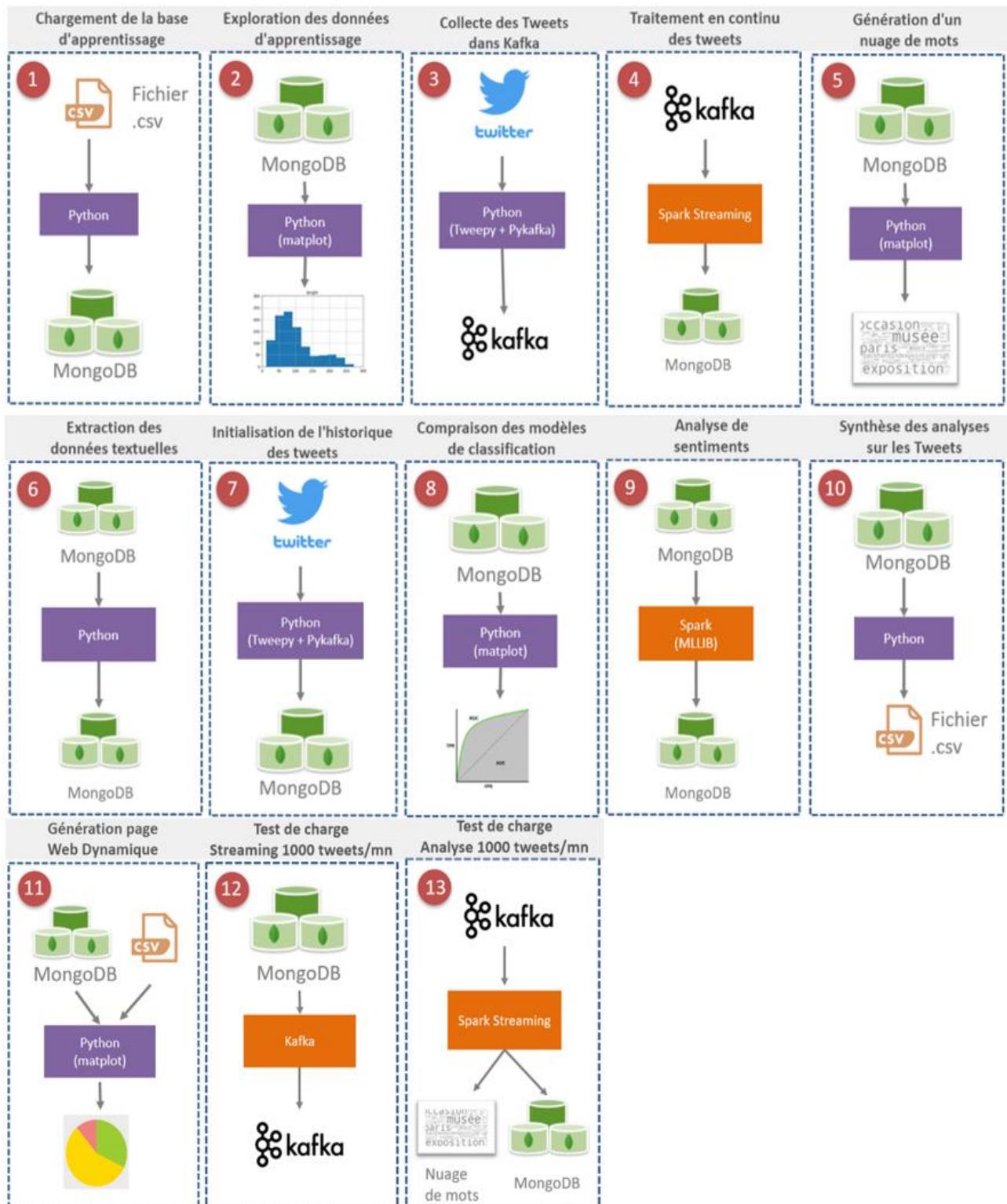
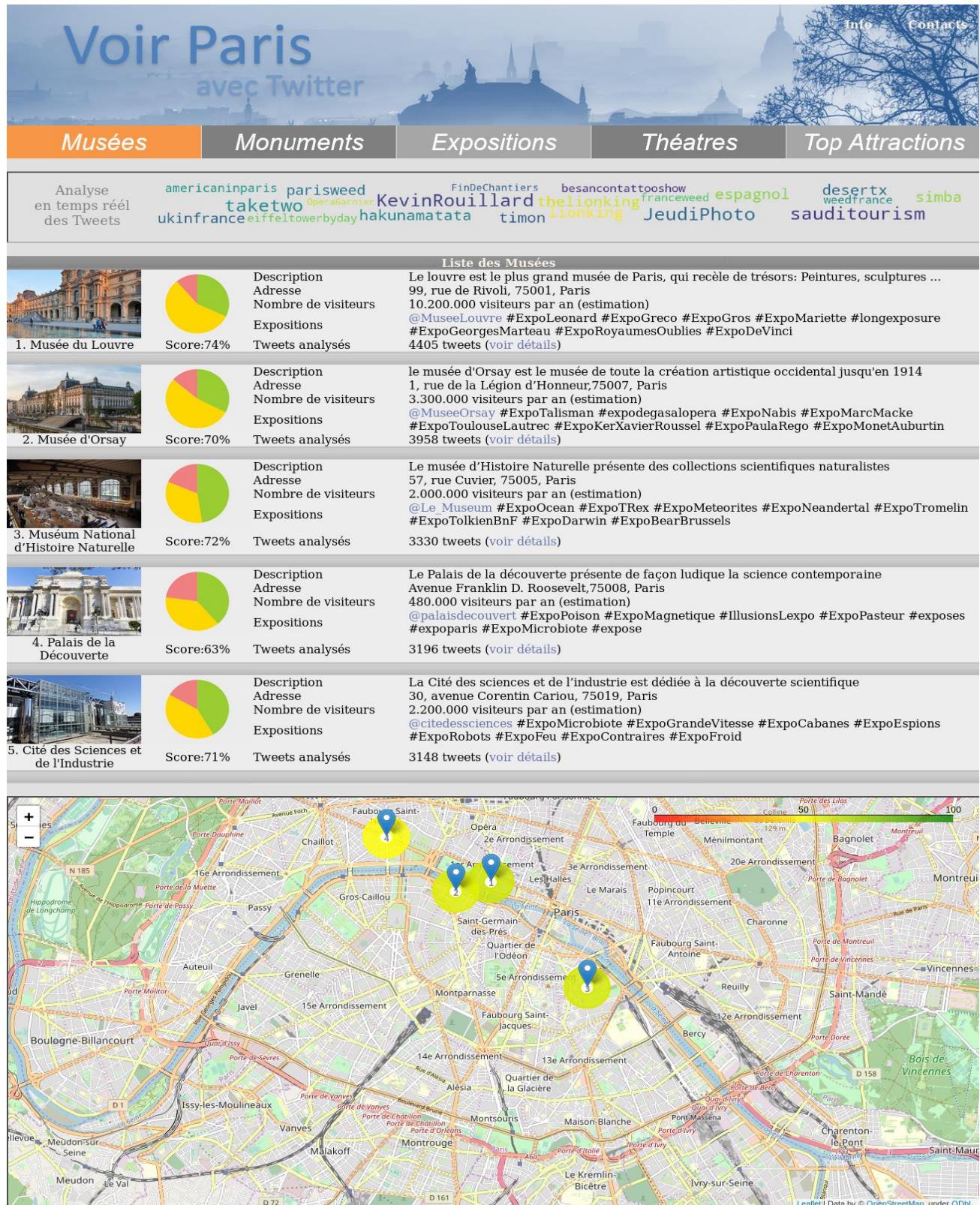


Figure 35: Liste des traitements du projet

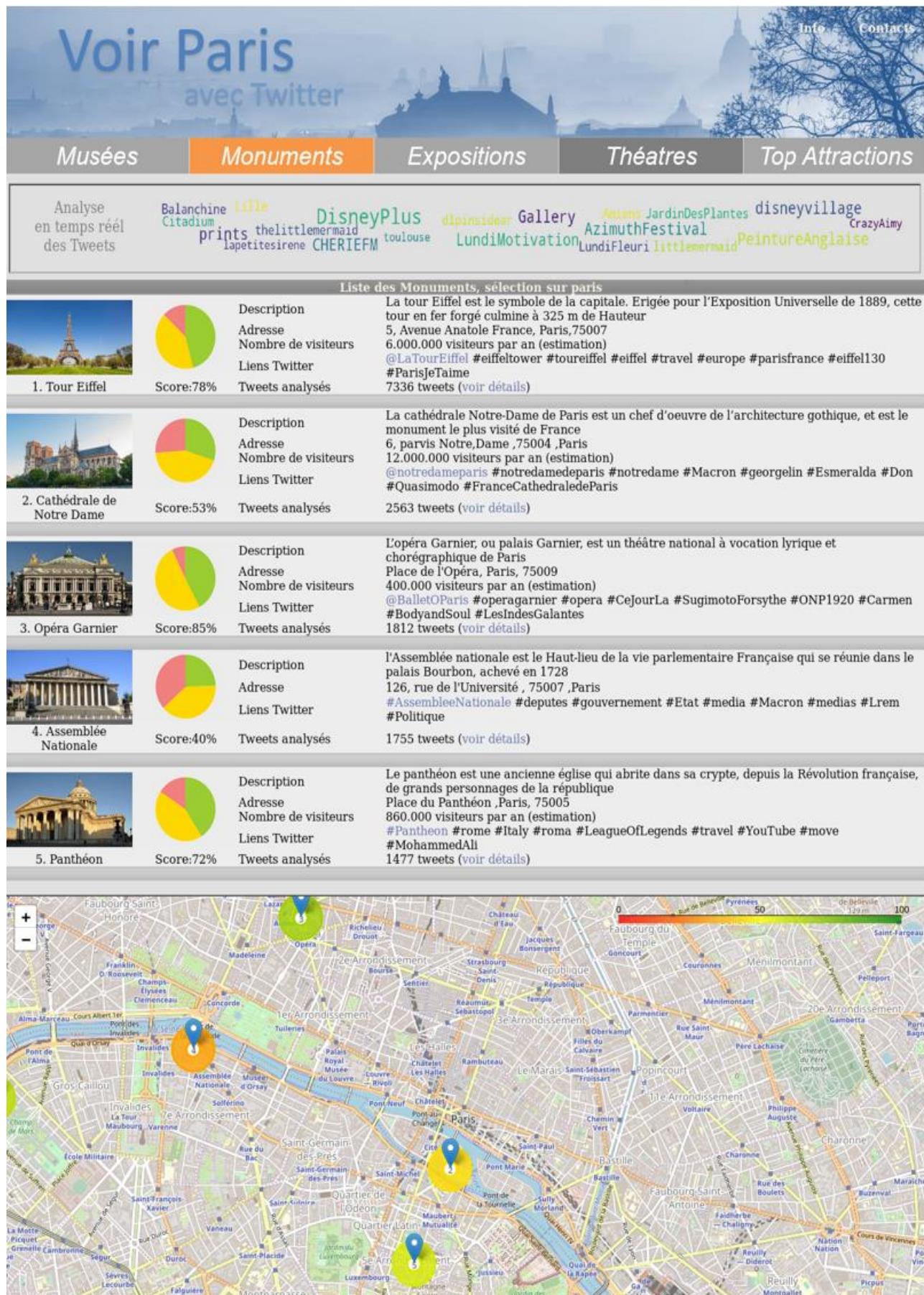
Annexe 10: Copies d'écrans du site Web

Page N°1 : Liste des 5 musées les plus présents sur Twitter

La copie d'écran suivante montre un exemple d'affichage des 5 musées les plus présents sur Twitter. Le nuage de mots évolue en continu. Une carte interactive de Paris permet de visualiser l'emplacement des différents musées ainsi que le niveau d'appréciation des visiteurs (rouge = avis négatif, jaune=avis neutre et vert=avis positif) :



Page N°2 : Liste des 5 monuments les plus présents sur Twitter



Page N°3 : Liste des 5 expositions les plus discutées sur Twitter

Voir Paris avec Twitter

Musées **Monuments** **Expositions** **Théâtres** **Top Attractions**

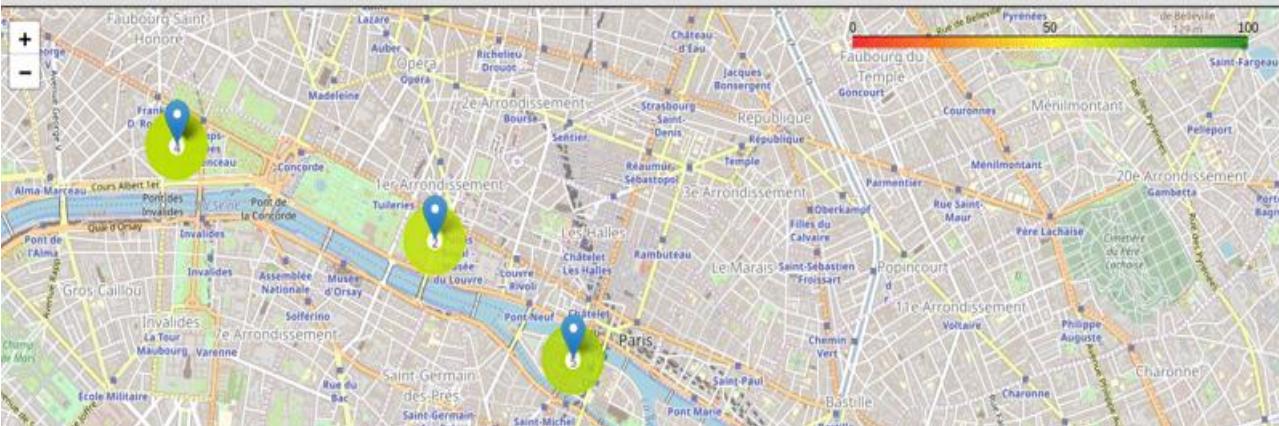
Analyse en temps réel des Tweets

PeintureAnglaise dlpinsidear AzimuthFestival LundiMotivation Citadium disneyvillage lille
Anglaise crazykym Festival Motivation Lundi citadium disneyvillage lille
LundiFleuri thelittlemermaid Gallery CHERIEFM prints littlemermaid Balanchine
the little mermaid gallery CHERIEFM prints little mermaid Balanchine
Amiens DisneyPlus JardinDesPlantes toulouse
Amiens Disney Plus Jardin Des Plantes toulouse

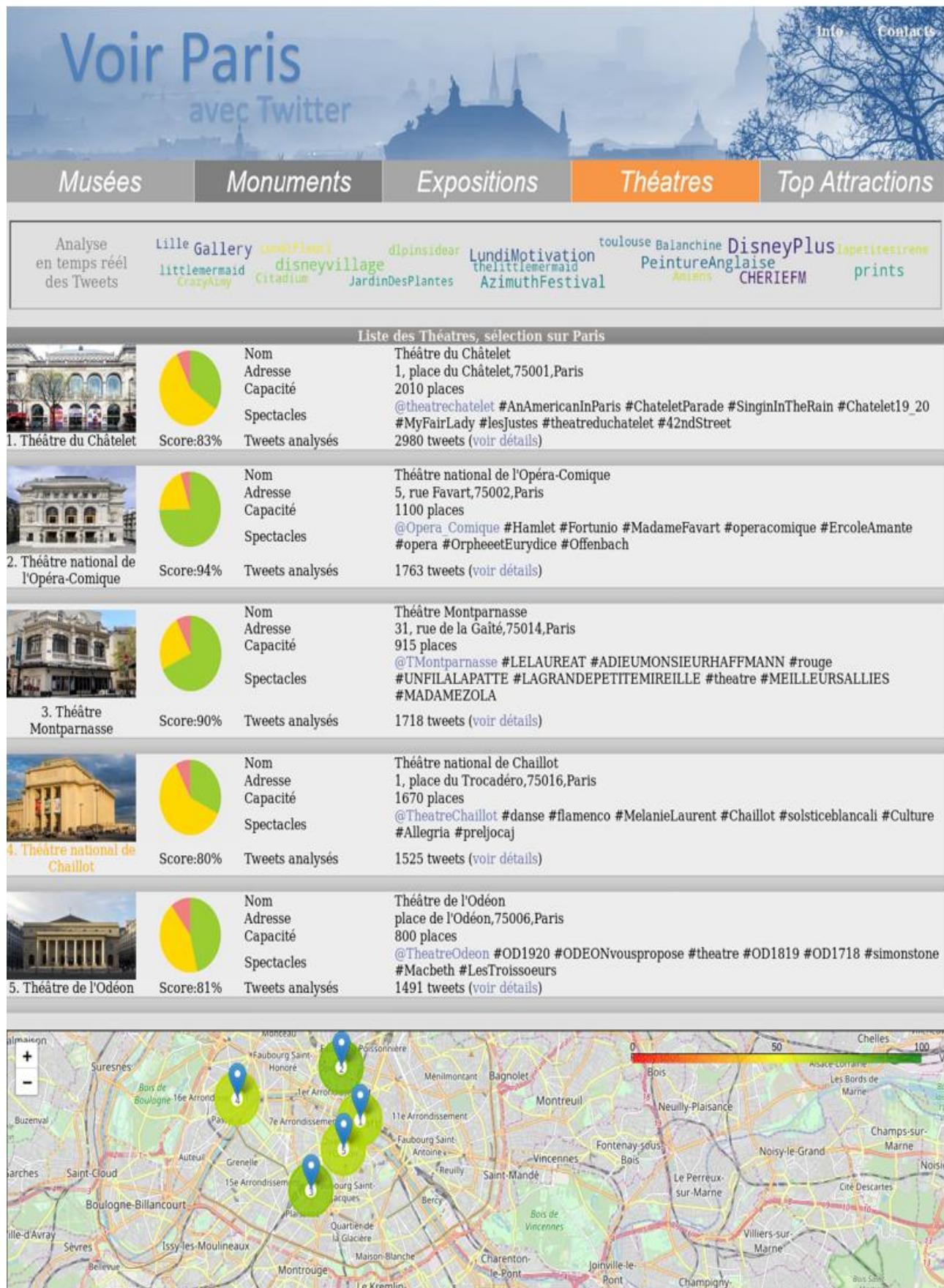
Liste des Expositions, sélection sur Paris

	1. FIAC		Description Adresse Dates Liens Twitter Score:74%	La FIAC est une Foire internationale d'art contemporain, qui se tient principalement au grand Palais, les jardins des Tuilleries, la place Vendôme et le petit Palais 3, avenue du Général Eisenhower, 75008, Paris 10-Oct-2019 au 20-Oct-2019 @FIAC #fiac #contemporaryart #ArtMarket #FRIEZE #ARTPRICE #fiac2019 #fiac2018 #biennale 891 tweets (voir détails)
	2. Léonard de Vinci		Description Adresse Dates Liens Twitter Score:77%	Cette exposition célèbre les 500 ans de la mort de Léonard de Vinci 99, rue de Rivoli, 75001, Paris 24-Oct-2019 au 24-Feb-2020 #ExpoLéonard #Louvre #leonardodavinci #LeonarddeVinci #MuseeduLouvre #TheSequence #leonardo500 #davinci #Renaissance 643 tweets (voir détails)
	3. expoTolkienBnF		Description Adresse Dates Liens Twitter Score:78%	Cette exposition est consacrée à l'auteur du Seigneur des Anneaux, l'écrivain britannique J.R.R. Tolkien Quai François Mauriac , 75013 ,Paris 22-Oct-2019 au 16-Feb-2020 #expoTolkienBnF #Tolkien #BnF #AbecedaireTolkien #hobbit #fantasy #TerreDuMilieu #litterature #LOTR 603 tweets (voir détails)
	4. Greco		Description Adresse Dates Liens Twitter Score:81%	Cette exposition est la première rétrospective en France d'un génie de la peinture de la Renaissance italienne 3, avenue du Général Eisenhower, 75008, Paris 16-Oct-2019 au 10-Feb-2020 #ExpoGreco #GrandPalais #Greco #parisjetaime #igersfrance #ig_france #ig_paris #ig_europe #painting 509 tweets (voir détails)
	5. Marie Antoinette, Métamorphoses d'une image		Description Adresse Dates Liens Twitter Score:83%	L'exposition aborde cinq thématiques qui permettent de comprendre les différentes images de la Reine Marie-Antoinette 2, Boulevard du Palais, 75001, Paris 16-Oct-2019 au 26-Jan-2020 #MarieAntoinette #Conciergerie #Versailles #LouisXVI #18thcentury #letthemeatcake #chateaudaversailles #louisxiv #history 412 tweets (voir détails)

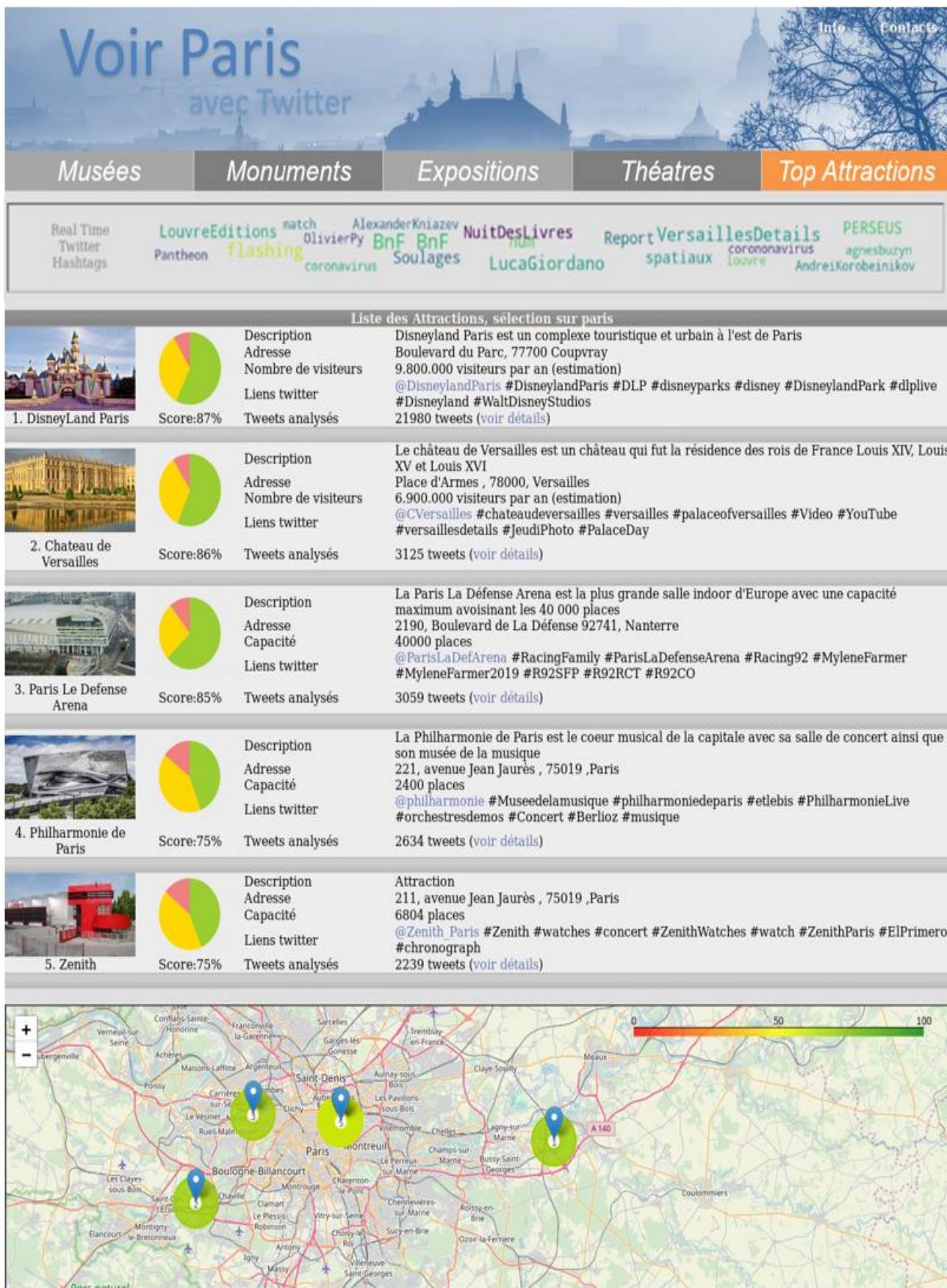
Map of Paris showing exhibition locations:



Page N°4 : Liste des 5 théâtres les plus présents sur Twitter



Page N°5 : Liste des 5 attractions les plus discutées sur Twitter



Page N°6 : Analyse statistique des tweets

La copie d'écran suivante montre les résultats des analyses sur le contenu des tweets collectés sur le musée du Louvre (Analyse de sentiments, volumétries, taille moyenne des tweets, nuages de mots, photos les plus partagées)

