2020

Analyse de sentiments à partir des avis postés sur le site Yelp

PROJET UASB03

CECILE MALHERBE

TABLE DES MATIERES

I. Choix d'un système de stockage passant à l'échelle par distribution

Dans cette partie, nous allons expliciter comment obtenir un environnement distribué complet et scalable tant au niveau des ressources de stockage que des ressources de calcul.

Dans un premier temps, nous étudierons l'architecture d'un cluster Spark et son fonctionnement (les Resilient Distributed Dataset et la reprise sur panne).

Puis, nous verrons l'architecture d'un cluster Cassandra et ses principales caractéristiques (scalabilité, cohérence, partitionnement et reprise sur panne).

Enfin, nous étudierons la bonne intégration des deux systèmes avec le Spark Cassandra Connector. Nous verrons comment déployer une telle architecture avec un cluster Datastax Entreprise et Docker.

A. Spark

Spark est un framework open source dédié au calcul distribué à grande échelle. Il permet de réaliser des traitements et des analyses complexes sur de gros volume de données. Spark n'est pas une base de données et ne stocke rien. Il est nécessaire de l'associer à un système de gestion de base de données pour le stockage des données.

1. Architecture d'un cluster Spark

Un cluster Spark est organisé selon une topologie maître-esclave, avec un nœud « master » et un ou plusieurs nœuds « workers ».

Le master communique avec l'application cliente qui lui transmet des demandes de traitement. L'exécution d'une demande de traitement sur un cluster Spark passe d'abord par un Driver. Le driver est créé soit sur la machine qui soumet l'application (mode client), soit à l'intérieur du cluster (mode cluster).

Le Driver divise une application Spark en tâches et les planifie pour s'exécuter sur les workers. Il coordonne les workers et l'exécution globale des tâches. Le Driver héberge aussi une interface web pour monitorer les traitements Spark. Le Driver communique avec le Cluster Manager qui gère les ressources des workers. Le Cluster Manager distribue équitablement les tâches entre les workers, et arbitre la quantité de CPU et de mémoire à allouer à chacun des traitements.

Les workers sont des nœuds d'un cluster Spark dans lesquels un ou plusieurs exécuteurs vont réaliser les tâches. Ces derniers effectuent le travail demandé en parallèle. Les workers établissent une communication bidirectionnelle avec le driver et le Cluster Manager.

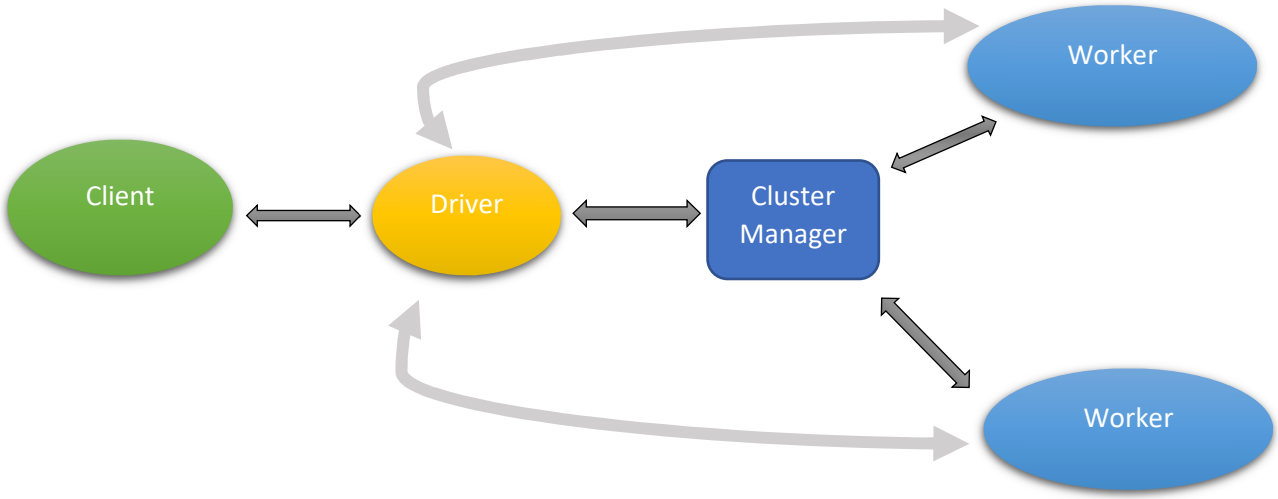


Figure 1 – Architecture d'un cluster Spark

2. Les Resilient Distributed Dataset ou RDD

Les RDD sont un concept au cœur du framework Spark. Un RDD est une collection distribuée sur différentes partitions, chaque partition étant traitée par un nœud du cluster Spark. Ils sont créés à partir d'une source de données (une base de données, un autre RDD ou un flux de données).

Les RDD sont dits résilients car Spark conserve l'historique des opérations qui a permis de les constituer. On peut donc recréer et recalculer l'ensemble des données d'un RDD.

Les RDD supportent deux types d'opérations :

- ➔ Les transformations : elles retournent un nouveau RDD en filtrant, enrichissant ou restructurant une collection. Rien n'est évalué lorsque l'on fait appel à une fonction de transformation car Spark ne l'exécute pas de suite mais garde un enregistrement de la fonction appelée. L'exécution est dite « lazy » ou paresseuse.
- ➔ Les actions : elles évaluent et retournent une nouvelle valeur. Au moment où une fonction d'action est appelée sur un objet RDD, l'ensemble des opérations d'un traitement Spark est exécuté et le résultat est retourné.

En plus de ces opérations, un RDD peut être rendu persistant afin d'être réutilisé dans d'autres chaînes. Il est alors stocké en RAM, et disponible comme source de données pour d'autres transformations

Lorsqu'une opération de type action est appliquée, une exécution distribuée se produit : le Driver détermine les étapes à exécuter et les tâches sont assignées aux nœuds « workers » du cluster. Chaque exécuteur d'un nœud worker travaille sur une partition à la fois de la collection. Lorsqu'ils atteignent une phase dite de « shuffle », les données sont redistribuées sur les nœuds « workers » afin de les regrouper selon certains critères.

3. Reprise sur panne

Dans un cluster Spark, une panne sur un noeud peut survenir en raison d'une défaillance d'un disque ou d'une panne de réseau.

Nous avons vu que les RDD, qu'ils soient persistants ou non, sont des collections partitionnées. Ils sont donc composés de fragments distribués sur les nœuds « workers » du Cluster Spark.

Spark conserve l'historique des opérations qui a permis de constituer un RDD. En cas de défaillance d'un nœud, Spark exploite la capacité à reconstruire des fragments de RDD par application de la chaîne de traitement sur les autres nœuds du cluster.

La défaillance d'un worker est donc, au final, sans incidence sur le résultat outre un temps d'exécution allongé du fait du retraitement. Ainsi les données perdues peuvent être récupérées souvent rapidement sans avoir à recourir aux mécanismes de réplication souvent couteux.

Le master, en revanche, est indispensable au fonctionnement global du système. C'est un point unique de défaillance pour lequel un système de redondance des services et de sauvegarde des données est indispensable.

B. Cassandra

Cassandra est un système de gestion de base de données (SGBD) à grande échelle conçu pour gérer des quantités massives de données sur un grand nombre de serveurs.

1. Architecture d'un cluster Cassandra

Dans la topologie d'un cluster Cassandra, il n'y a pas de notions de nœud maître et de nœud esclave. Tous les nœuds du cluster sont au même niveau et bénéficie de la capacité de lecture et d'écriture dans le cluster. Ainsi, un client qui interroge Cassandra contacte un nœud au hasard parmi tous les nœuds du cluster. Lorsque qu'un client veut lire ou écrire une donnée, il lui suffit de se connecter à l'un des nœuds du cluster, si le nœud en question ne contient pas la donnée, alors il la demandera au nœud qui la contient. Tous les nœuds du cluster sont capables de dire où se trouve la ressource recherchée. Le nœud interrogé par le client et qui redirige la requête au bon nœud, est appelé coordinateur. Ce mécanisme est transparent pour le client.

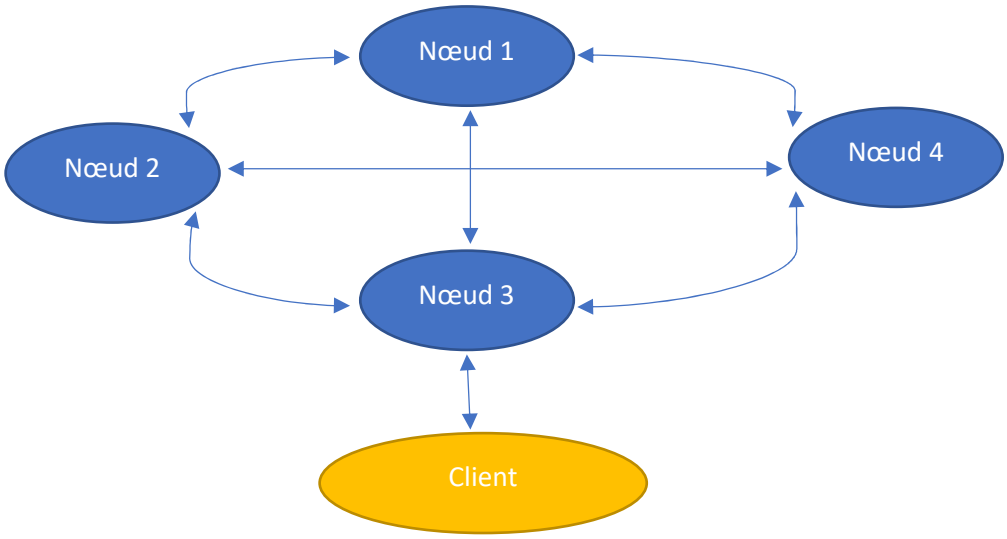


Figure 2 – Architecture d'un cluster Spark

2. Partitionnement

Dans un cluster Cassandra, les documents sont distribués via un partitionnement par hachage. Chaque nœud du Cluster est disposé, dans le sens des aiguilles d'une montre, sur un anneau ou « Hash Ring ». Une fonction de hachage permet d'associer à chaque nœud une position (« token ») sur l'anneau. Ce token va permettre de représenter un intervalle d'affectation des données (« token range ») pour chaque nœud. Chaque nœud du Cluster a la connaissance de la topologie du Hash Ring.

Le choix d'un nœud pour l'écriture dans le Hash Ring se fait grâce à un « partitioner » qui permet de répartir uniformément des documents dans le cluster afin d'obtenir un hachage cohérent.

Cassandra est basée sur une structuration en paire clé-valeur. Le premier élément de la clé primaire d'une collection de documents, appelé clé de partition, détermine sur quel noeud du cluster la donnée sera stockée. Cette répartition est effectuée via le hash de la clé de partition. Le document sera affecté au noeud qui gère le token range dans lequel est compris le hash de la clé. Les autres éléments d'une clé primaire composée, appelés « clustering columns », servent ensuite à ordonner les données sur un même noeud.

De plus, Cassandra s'appuie sur un système de nœuds virtuels qui sont assignés aux nœuds physiques et sont disposés sur le Hash Ring de manière à obtenir une distribution automatique et transparente des données. Ainsi, si la topologie du cluster change, Cassandra se charge de répartir les nœuds virtuels le plus équitablement possible sur le Hash Ring. L'équilibrage du partitionnement est ainsi facilité lors de l'ajout ou de la suppression d'un nœud du cluster.

3. Réplication

Les données Cassandra sont stockées dans un Keyspace global au cluster. Deux paramètres liés au Keyspace doivent être spécifiés dans le cadre de la réplication :

- ➔ La topologie de la réplication : logique selon laquelle les ressources seront répliquées au sein du cluster.

➔ Le facteur de réplication : Nombre final de copies de la ressource dans le cluster.

En choisissant la « stratégie simple » comme topologie de réplication, le nœud coordinateur qui reçoit la demande d'écriture du client va utiliser la méthode de hachage, le token des nœuds du cluster et la clé de répartition du document pour déterminer le nœud dans lequel il sera stocké. Le coordinateur redirige alors la requête vers le nœud choisi par la fonction de hachage. En introduisant un facteur de réplication de 3, on s'assure que le coordinateur redirige la requête en écriture vers les 2 nœuds consécutifs au nœud choisi sur l'anneau dans le sens du Hash Ring.

Cassandra permet de spécifier pour les requêtes en lecture et en écriture le niveau de cohérence souhaité, ce qui permet de régler l'équilibre entre sécurité et rapidité selon les besoins.

Les niveaux de cohérence les plus communément utilisés sont :

- ➔ ALL : le cluster attend la réponse de tous les nœuds devant écrire ou lire la donnée avant de répondre au client
- ➔ ONE : le cluster attend seulement la réponse d'un des nœuds avant de répondre au client
- ➔ QUORUM : le quorum est défini comme la majorité absolue des réplicas. Si la donnée est répliquée 3 fois, alors le cluster attendra la réponse de 2 nœuds avant de répondre au client. Si la donnée est répliquée 4 fois alors le quorum est constitué de 3 machines.

Avec la réplication, on obtient une architecture sans point individuel de défaillance : en cas de panne sur un nœud, un autre noeud identique gérant la même portion de donnée prend en charge les traitements demandés. Lors d'une défaillance prolongée ou d'une modification de la topologie du cluster Cassandra, la redistribution des données se fait de façon automatique et transparente.

4. Choix de Cassandra pour le stockage des données

Le choix de Cassandra pour le stockage des données s'est fait selon plusieurs critères :

- ➔ La scalabilité est apporté à deux niveaux. D'une part, la topologie multi-nœuds du cluster permet d'équilibrer la charge de la communication avec les applications clientes. D'autre part, le partitionnement permet d'équilibrer la charge du traitement et le stockage des données.
- ➔ Cassandra permet de paramétrer finement la tolérance aux pannes grâce à la réplication et présente l'avantage de ne pas avoir de point unique de défaillance du fait de sa topologie multi-nœuds.
- ➔ Il est possible de régler le niveau de cohérence des requêtes en lecture et en écriture, et ainsi de choisir entre prudence et performance.
- ➔ Le partitionnement par hachage cohérent et les nœuds virtuels permettent de maintenir la cohérence globale du partitionnement déjà effectué selon que l'on ajoute ou supprime un serveur du cluster.

Dans la suite, nous allons associer Spark à Cassandra comme source de données. Dans une telle architecture le calcul distribué est assuré par Spark et le stockage et le partitionnement sont assurés par Cassandra. Cela nous permettra de réaliser des traitements distribués très rapidement, Spark fournissant des facilités de calcul distribué que ne possède pas Cassandra.

C. Cassandra et Spark, un système distribué complet

Dans cette section, nous allons étudier la bonne intégration des systèmes Spark et Cassandra, grâce au Spark Cassandra Connector, et plus spécifiquement la correspondance entre le partitionnement du stockage Cassandra et le partitionnement des calculs de Spark. Nous étudierons ensuite l'architecture d'un cluster Data Stax Entreprise.

1. Le Spark Cassandra Connector

Datastax, l'entreprise fournissant Cassandra, a créé un connecteur permettant de connecter Spark sur un cluster Cassandra. Ce connecteur permet de :

- ➔ manipuler des collections Cassandra sous forme de RDD
- ➔ d'écrire des RDD Spark dans des collections Cassandra
- ➔ d'exécuter des requêtes CQL via les applications Spark

Le connecteur Spark Cassandra permet de créer des partitions Spark en prenant le token range de Cassandra et en le divisant en blocs. Le nombre et la taille des partitions Spark sont basés sur des métadonnées lues à partir du cluster Cassandra. Le connecteur évalue le nombre de partitions Spark en divisant le volume des données estimé de la table par le paramètre input_split_size_in_mb (volume des données d'une partition Spark).

Cette stratégie de partitionnement possède plusieurs avantages :

- ➔ Le connecteur sait optimiser les lectures, dès lors que Spark et Cassandra cohabitent sur un même noeud. En effet, le connecteur Spark-Cassandra amène chaque worker Spark à préférer lire les partitions Cassandra du nœud sur lequel il est, évitant ainsi de couteux transferts de données. C'est le principe de Data Locality.
- ➔ Il est possible de réaliser des opérations sans shuffle sur les partitions.
- ➔ L'ordre dans une partition de Cassandra est maintenu.

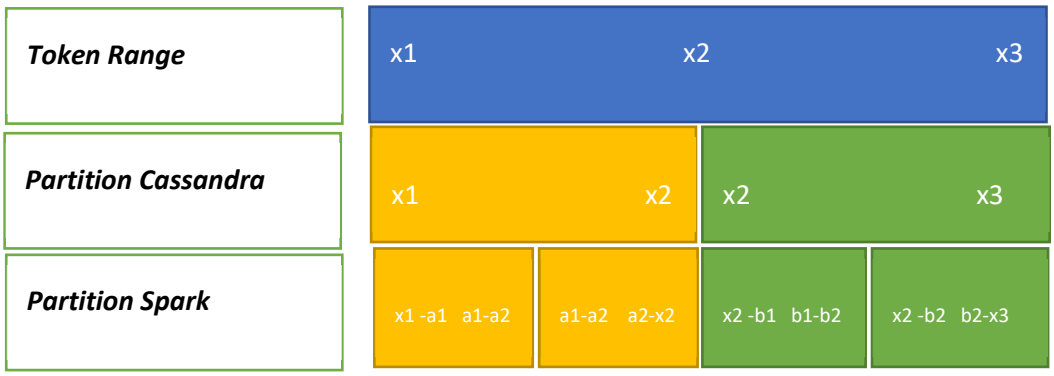


Figure 3 – Mapping des partitions Spark et Cassandra

En conclusion, le Spark Cassandra Connector grâce à sa stratégie de partitionnement et son optimisation des lectures par application du principe de Data Locality, permet lorsque les workers Spark sont placés sur les mêmes machines que les nœuds Cassandra, d'obtenir un système scalable tant au niveau des ressources de stockage que des ressources de calcul.

2. Cluster DSE

a) Architecture d'un cluster DSE

DataStax Entreprise (DSE) est un produit proposé par DataStax. Il s'agit d'une version certifiée de Cassandra qui intègre et utilise d'autres technologies Apache. Chaque nœud DSE peut intégrer des fonctionnalités d'indexation, de recherche, d'analyse et graphiques grâce à la combinaison de Cassandra avec Apache Solr, Apache Spark et DSE Graph.

DataStax Enterprise utilise le Spark Cassandra Connector pour connecter Spark et Cassandra.
Lors du déploiement d'un cluster DSE, le Spark Master s'exécute sur un des nœuds et les workers sur l'ensemble des nœuds du cluster. Le Spark Master sert uniquement de gestionnaire de ressources pour les applications Spark.

Dans un cluster DSE, le choix du nœud sur lequel s'exécutera le Spark Master s'effectue automatiquement sans configuration manuelle. Les nœuds du cluster communiquent entre eux pour élire un Spark Master ainsi qu'un Spark Master de réserve en cas de défaillance du premier. Si le nœud Spark Master tombe en panne, le Spark Master de réserve prend la relève et un nouveau Spark Master de réserve est choisi parmi les nœuds restants.
Si une panne survient sur l'un des nœuds du cluster DSE (autre que celui où s'exécute le Spark Master), le Spark Master réaffecte les tâches de ce nœud à un autre nœud worker. De plus, si le facteur de réplication paramétrer au niveau du Keyspace dans Cassandra est supérieur à 1, le Spark Master choisira un nœud contenant une réplique des données du nœud défaillant. Le principe de Data Locality est ainsi conservé.

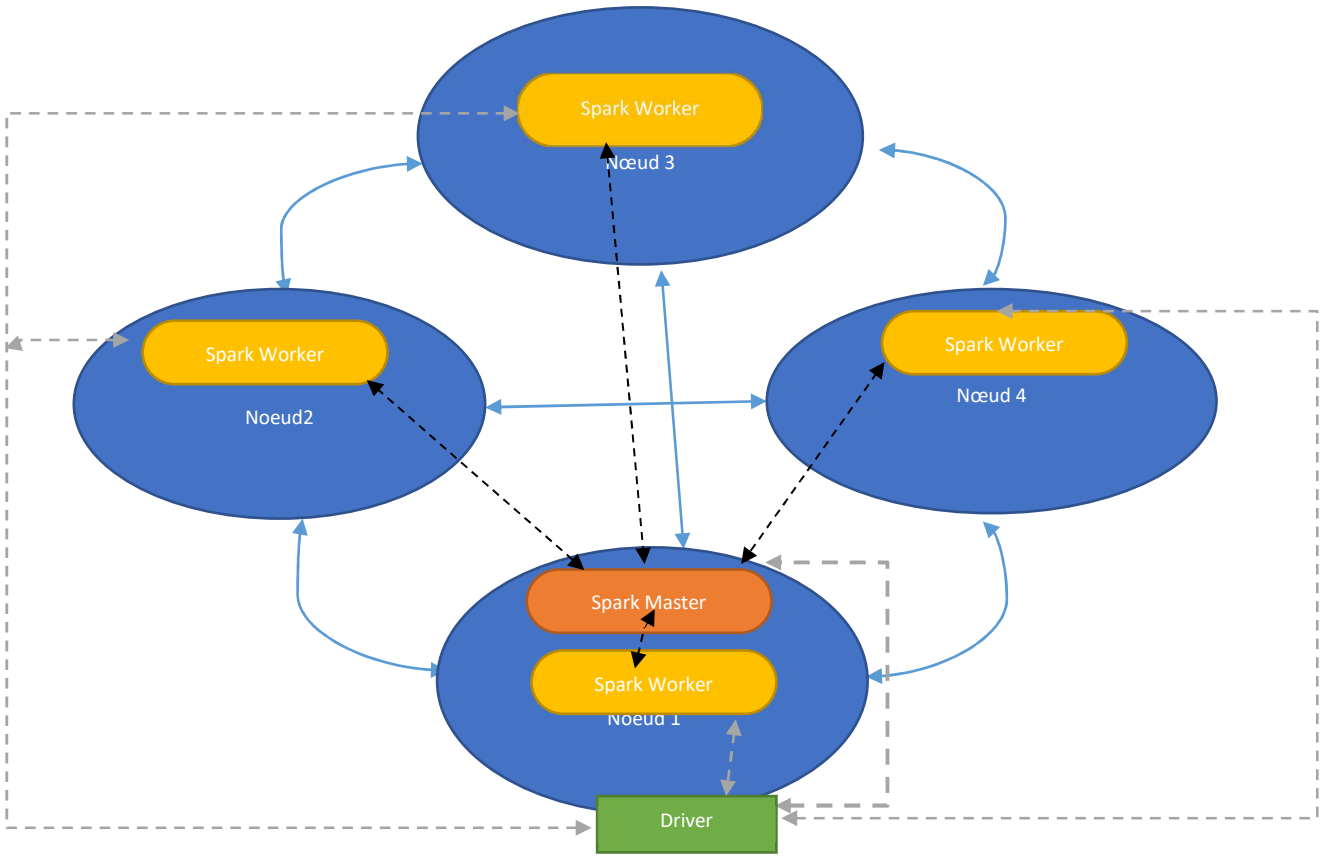


Figure 4 – Architecture d'un cluster DSE

b) Chargement du jeu de données dans Cassandra et lecture dans Spark

Dans la suite, nous allons effectuer une analyse de sentiment, à partir d'un jeu de données d'apprentissage et d'un jeu de données de test. Ces deux jeux de données sont constitués de 3 variables :

- ➔ Id : Identifiant de l'observation
- ➔ Cible : Polarité du sentiment (0 ou 1)
- ➔ Text : Avis postés sur le site Yelp

Le chargement de ces deux jeux de données dans Cassandra passe par la création d'un Keyspace dans Cassandra, avec spécification de la topologie de réplication et du facteur de réplication :

```
CREATE KEYSPACE IF NOT EXISTS YELP WITH REPLICATION = {'class' : 'SimpleStrategy',
'replication_factor': 3};
```

On crée ensuite les deux tables qui vont recevoir les données, en précisant que la clef primaire est l'identifiant de l'observation :

```
USE YELP;
CREATE TABLE yelp_500n (id INT, cible INT, text VARCHAR, PRIMARY KEY(id)) ;
CREATE TABLE yelp_testn (id INT, cible INT , text VARCHAR, PRIMARY KEY(id)) ;
```

Puis, on copie les données dans ces deux tables à partir des fichiers CSV :

```
COPY yelp_500n (id,cible, text) FROM '/yelp_500n.csv' WITH DELIMITER=';' AND HEADER=TRUE;
COPY yelp_testn (id,cible, text) FROM '/YELP_TESTn.csv' WITH DELIMITER=';' AND HEADER=TRUE;
```

Dans un cluster DSE, la lecture des tables Cassandra à partir de Spark, s'effectue avec les commandes suivantes :

```
val yelp = spark.read.format("org.apache.spark.sql.cassandra").options(Map( "table" -> "yelp_500n",
"keyspace" -> "yelp")).load()

val yelp_test = spark.read.format("org.apache.spark.sql.cassandra").options(Map( "table" ->
"yelp_testn", "keyspace" -> "yelp")).load()
```

On dispose alors de nos tables Cassandra sous forme de DataFrame Spark.

II. Analyse de sentiments

L'objectif de notre étude est la discrimination à deux classes (avis positif ou négatif) à partir du texte des avis et recommandations d'utilisateurs sur des restaurants, magasins, divertissements ou services, postés sur le site Yelp, qui est un site participatif d'avis sur des commerces locaux. Les données sont issues du Yelp DataSet Challenges 2015 [1], et ont été construites et utilisées comme benchmark dans la publication [2].
Les données d'apprentissage comptabilisent 560 000 avis et les données de test 38 000 avis au format csv et peuvent être téléchargées sur le site https://data.wluper.com/.
Les documents du corpus sont des textes courts en anglais. Une note (rating) allant de 1 à 5, donnée par l'utilisateur, est associé à chaque document. La polarité de chaque document est évaluée comme suit : pour les notes de 1 à 2 la polarité est négative, pour les notes de 4 à 5 la polarité est positive. Les documents avec un niveau de sentiment neutre (note égale à 3) ne sont pas inclus dans les jeux d'apprentissage et de test. La distribution de la variable cible est équilibrée : 50% des documents ont une polarité positive et 50% une polarité négative.

A. Pré-traitement du corpus

1. Lemmatisation

Nous procédons, dans un premier temps, à la lemmatisation des mots présents dans chaque document de notre corpus, grâce au lemmatiseur du projet Stanford Core NLP. La lemmatisation permet de diminuer le bruit sémantique. Elle fait appel à l'analyse lexicale avec étiquetage grammaticale, l'objectif étant de supprimer uniquement les terminaisons flexionnelles et de retourner la forme canonique du mot (ou lemme) : substantifs ramenés au singulier, flexions des verbes ramener à l'infinitif etc. Nous conservons les lemmes de chaque document, la ponctuation a été supprimée et les majuscules converties en minuscules.

2. Suppression des Stops Words

Certains termes comme les conjonctions de coordination, déterminants, particules, propositions et pronoms font partie des mots qui reviennent très fréquemment dans les textes et sont donc peut discriminants. Ils seront donc supprimés de notre corpus.

Après la lemmatisation et la suppression des Stop Words de notre corpus, nous obtenons pour chaque document une liste des lemmes. Ci-dessous le résultat pour les 5 premiers documents de notre corpus :

Text	**Lemme**
I found this gem on Yelp (go figure) and have taken numerous shirts, jackets, pants, and dresses etc.to be cleaned. This company lives up to its name. The staff is prompt, professional, and reliable. The owner takes customer service very seriously and values customer loyalty. I used the spend $30 for $50 coupon on the yelp site and redeemed it from my phone. It was a true breeze. The owner (I'm assuming) explained they have delivery service free of charge. I agree with his philosophy- dry cleaning should be hassle free. Quality work every time, no	[find, gem, yelp, figure, take, numerous, shirt, jacket, pants, dress, clean, company, live, name, staff, prompt, professional, reliable, owner, take, customer, service, seriously, value, customer, loyalty, use, spend, coupon, yelp, site, redeem, phone, true, breeze, owner, assume, explain, delivery, service, free, charge, agree, with, philosophy, dry, cleaning, hassle, free, quality, work, every, time, exception, necessarily, cheapest, absolutely, best, get, pay, cheers, one, request, owner, please, put, fan,

exceptions. Not necessarily the cheapest, but absolutely the best. You get what you pay for. Cheers! One request for the owner: please put a fan (or anything to cool) in the receiving area. That place can get toasty for those that choose to pick up in person.	anything, cool, receive, area, place, can, get, toasty, choose, pick, person]
I had no problem redeeming the 30-50 coupon. The staff was very friendly and knowledgeable. The clothes came out clean, except one that said they couldn't take the stains off without damaging the item, that item was charge which I don't think it should; since it was not cleaned. The price on my personal opinion it's very expensive. I took 11 pieces and paid with coupon over $80.	[problem, redeem, coupon, staff, friendly, knowledgeable, clothes, come, clean, except, one, say, take, stain, without, damage, item, item, charge, think, since, clean, price, personal, opinion, expensive, take, piece, pay, coupon]
Best customer service I've ever had at a cleaner! They were so friendly and fast. I've used them twice and will always use them.	[best, customer, service, ever, cleaner, friendly, fast, use, twice, will, always, use]
Excellent service in every way. Been using them for years. Always accommodating, pick up by Jeff always smiles and prompt. Dave is the best owner.	[excellent, service, every, use, year, always, accommodate, pick, jeff, always, smile, prompt, dave, best, owner]
I'm a little biased since I used to work here. Nonetheless, it's a great nursery. It's got quality plants, quality service and quality gardening supplies. The customer service is especially good and the care taken of the plants is outstanding. The only drawback is the price, but with the quality of everything so good, what else do you expect?	[little, bias, since, use, work, nonetheless, great, nursery, get, quality, plant, quality, service, quality, gardening, supplies, customer, service, especially, good, care, take, plant, outstanding, drawback, price, quality, everything, good, else, expect]

Notre corpus est composé de 188 366 lemmes distincts (hors stop words). Ci-dessous les 40 lemmes les plus fréquents du corpus :

Lemme	**Freq**
get	410 631
place	344 902
food	316 137
good	282 095
like	267 681
time	258 059
just	247 346
one	235 444
order	228 002
come	221 011
service	206 025
great	200 872
can	187 066
back	185 844
say	185 757
make	184 038
will	176 904
take	163 504
really	161 925
try	152 122

Lemme	**Freq**
even	136 007
want	129 541
look	129 292
give	125 603
wait	120 702
also	115 892
ask	115 493
eat	114 887
think	114 875
restaurant	114 568
know	114 096
tell	111 482
see	111 250
love	104 723
never	104 397
nice	101 266
price	99 843
people	98 862
well	98 269
drink	96 500

Parmi les lemmes les plus fréquents, on trouve de nombreux lemmes relatifs à la restauration : « restaurant », « eat », « drink » ou « food ». En effet, beaucoup des commerces notés sur le site Yelp sont des restaurants.

B. Représentation vectorielle

Afin d'exploiter des algorithmes d'apprentissage automatique sur nos données textuelles, il nous faut représenter le texte de nos documents sous la forme d'un vecteur de taille fixe, ceci afin de plonger la donnée dans un espace métrique. Nous utiliserons comme méthode de vectorisation, la matrice « Document x Termes » avec une pondération TF-IDF.

Pour ce faire, le corpus est représenté sous la forme d'une matrice contenant les documents en ligne et les lemmes en colonne. Chaque document correspond à un vecteur où la composante associée à un lemme vaut 0 si le lemme est absent du document, et une valeur pondérée selon l'importance du lemme, si le lemme est présent.

La pondération appliquée à cette matrice est la pondération TF-IDF. Il s'agit de la combinaison du « Term Frequency » et de l'« Inverse Document Frequency », c'est à dire la fréquence du terme dans un document par l'inverse de la fréquence du terme dans le corpus

Cette pondération conduit à considérer les mots fréquents à la fois dans un document et dans le corpus en entier avec une importance diminuée par rapport aux mots fréquents dans un document et rare dans le corpus. Un terme rare améliorera ainsi la pertinence lexicale. Avec cette représentation, deux documents seront proches s'ils ont de nombreux termes en commun et deux termes seront proches s'ils sont présents ensembles dans de nombreux documents.

On peut noter un inconvénient à cette représentation : les synonymes ne sont pas pris en compte dans la similarité entre documents alors que des mots polysémiques le sont. De plus, un terme rare peut se voir affecter un poids élevé, alors que pour la comparaison de documents il peut ne s'agir que de « bruit ». C'est une représentation qui ne tient pas compte du contexte des mots. Nous verrons plus tard comment la prise en compte des n-grammes permet de pallier, en partie, à cette difficulté.

Afin d'obtenir cette représentation vectorielle, nous utilisons l l'API « org.apache.spark.ml » basée sur les Dataset/DataFrame. La première étape consiste à créer une nouvelle instance « CountVectorizer » sur nos données lemmatisées. Le « CountVectorizer » effectue les deux opérations ci-dessous :

1) Au cours de la phase « fit », il trouve l'ensemble de mots dans tous les documents, puis compte les occurrences de ces mots dans chaque document. L'ensemble des mots constitue le « vocabulaire », dont la taille est paramétrable selon la fréquence d'apparition minimum d'un terme dans le corpus, le nombre minimum de documents dans lequel un terme doit apparaître ou le nombre maximal de mots retenus.

2) Au cours de la phase « transform », les occurrences d'un mot du « vocabulaire » sont comptabilisées pour chaque ligne du DataFrame en entrée et on obtient en sortie un vecteur creux pour chaque document, qui contient la taille du vocabulaire, l'index du mot dans le vocabulaire, puis le nombre d'occurrence dans le document pour ce mot.

Nous avons vu précédemment que nous avions 188 366 lemmes distincts dans notre corpus, soit autant de variables explicatives pour notre modèle. Nous limiterons notre vocabulaire aux 20 000 lemmes les plus fréquents dans le corpus, car utiliser un trop grand nombre de variables dans notre modèle peut entraîner un sur-apprentissage.

La deuxième étape, pour obtenir notre représentation vectorielle, consiste à calculer la fréquence inverse du document pour chaque terme du corpus en créant une nouvelle instance « IDF » et en appelant la méthode « fit » sur les vecteurs « TF » obtenus précédemment. Nous transformons ensuite les vecteurs « TF » en vecteurs « TF-IDF » grâce à la fonction de transformation d' « IDF ».

On obtient en sortie, pour chaque document, un vecteur creux qui contient la taille du vocabulaire, l'index du lemme dans le vocabulaire, puis le poids TF-IDF pour ce lemme.

La transformation du texte dans l'ensemble de données de test suit la même procédure que pour les données d'apprentissage : nous lemmatisons les textes des données de test, supprimons les stop words et transformons le corpus en matrice « Document x Termes » avec pondération TF-IDF. La pondération TF-IDF est obtenue avec le même vocabulaire et les mêmes pondérations IDF qu'en apprentissage. Il est important d'utiliser les pondérations IDF obtenues en apprentissage pour transformer les données de test, car cela crée une estimation plus réaliste des performances du modèle sur les nouvelles données.

C. Classification supervisée

Dans la suite, nous allons évaluer notre méthode de vectorisation « Documents x Termes » avec pondération TF-IDF en utilisant 3 modèles de classification supervisée : un classifieur bayésien naïf, les SVM et les forêts aléatoires. Pour chacun des algorithmes, afin de déterminer le meilleur modèle décisionnel, nous effectuerons une recherche en grille et une validation croisée. Nous comparerons les performances des modèles retenus à l'aide de leur AUC (Aire sous la courbe ROC), de la représentation de leur courbe ROC et de leur Accuracy.

1. Recherche par grille et validation croisée k-fold

Au sein de chacune des 3 méthodes choisies, plusieurs choix sont envisageables en fonction de la valeur des hyper-paramètres retenue. Les hyperparamètres sont déterminent la structure de base du modèle lui-même. Afin d'obtenir le « meilleur » modèle (pour un ensemble de données et dans une famille paramétrique fixée, comme les SVM linéaires), nous allons explorer l'espace des valeurs des hyper-paramètres à l'aide d'une grille multidimensionnelle ou grid-search (une dimension par hyperparamètre). Chaque combinaison des valeurs de la grille est caractérisée par le score de validation croisée obtenu pour les modèles qui possèdent ces valeurs des hyper-paramètres avec une validation croisée k-fold. La validation croisée k-fold consiste à découper l'échantillon d'apprentissage en k blocs de même taille, de manière à valider le modèle sur un bloc tout en entraînant le modèle sur les k-1 blocs restants. On obtient ainsi un premier taux d'erreur. On recommence ensuite k-1 fois l'opération. L'erreur est alors moyennée sur les k blocs et est appelée erreur de validation croisée, ce qui nous donne une estimation de l'erreur de test plus précise que celle obtenue avec un simple découpage test-apprentissage.

On choisit généralement k=10, mais dans la suite nous le fixerons à 5 afin de gagner en temps de calcul.

Nous estimerons les performances de généralisation des modèles retenus pour chacune des trois méthodes à l'aide des données de test mises de côté au départ. Pour chaque modèle retenu nous évaluerons l'AUC en test et en apprentissage et représenterons les courbes ROC obtenues sur les données de test.

La courbe ROC (Receiver Operating Characteristic) est un graphique représentant les performances d'un modèle de classification par la représentation du taux de vrais positifs en fonction du taux de faux positifs pour tous les seuils de classification.

L'AUC (Area Under Roc Curve), est l'aire sous la courbe ROC. Elle fournit une mesure des performances pour tous les seuils de classification possible.

Nous évaluerons aussi l'accuracy (le pourcentage d'observations correctement classées) en test et en apprentissage au seuil de 50%. En effet les données sont équilibrées (autant de positifs que de négatifs) et nous accordons autant d'importance au bon classement des positifs qu'au bon classement des négatifs.

2. Naïve Bayes, SVM et Forêts aléatoires

a) Naïve Bayes

Le classifieur bayésien naïf est une méthode de classification supervisée reposant sur le théorème de Bayes. L'hypothèse de base sur laquelle repose ce modèle est l'indépendance des variables explicatives (conditionnellement à la variable cible). Malgré la naïveté de ces hypothèses, le classifieur bayésien permet de fournir une base de référence par rapport aux performances des autres modèles et des temps de traitements beaucoup plus faibles que pour les autres méthodes.

L'API « org.apache.spark.ml » prend en charge les classifieurs bayésiens naïfs multinomiaux et les classifieurs bayésiens naïfs de Bernoulli. Il s'agit de modèle généralement utilisé pour la classification des documents. Dans ce contexte, chaque observation est un document et chaque variable explicative représente un terme dont la valeur est la fréquence du terme (pour les Bayes naïfs multinomiaux) ou, 0 ou 1 pour indiquer si le terme est présent dans le document (pour les Bayes naïfs de Bernoulli). C'est le modèle multinomial qui est sélectionné par défaut.

Ce modèle possède un paramètre d'apprentissage qui permet de lisser les données catégorielles et d'éviter de sur-adapter les données d'apprentissage en modifiant la probabilité attendue pour certaines classes. Nous conservons la valeur par défaut égale à 1.

On obtient une AUC moyenne, à partir des 5 modèles (validation croisée k-fold) sur leurs fractions de test, égale à 0.9140. Ci-dessous les valeurs de l'AUC et de l'Accuracy obtenues sur les données d'apprentissage et de test pour ce modèle :

	Apprentissage	**Test**
AUC	0,9177	0,9142
Accuracy	0,8639	0,8583

b) *Support Vector Machines (SVM)*

Le principe des Support Vector Machines est de réaliser une séparation linéaire des classes par plongement dans un espace de dimension plus grande, lorsque la séparation linéaire n'est pas possible dans l'espace initial. On recherche la séparation qui maximise la marge entre les classes, c'est-à-dire la distance entre la frontière de séparation et les observations les plus proches. Ces dernières sont appelées *vecteurs supports*. Dans les SVM, la frontière de séparation est choisie comme celle qui maximise la marge, car cette maximisation diminue la complexité du modèle et lui assure une meilleure généralisation.

Un des avantages des SVM est qu'ils supportent la colinéarité des variables explicatives.

L'API « org.apache.spark.ml » prend en charge les SVM linéaires. Ce modèle possède un hyper-paramètre :

- ➔ regParam : il définit le compromis entre les deux objectifs de minimisation de la perte (c.-à-d. l'erreur d'apprentissage) et de minimisation de la complexité du modèle pour éviter le surajustement. Il pénalise les erreurs et permet de contrôler l'ajustement du modèle. Plus il est grand et plus la sensibilité aux erreurs est forte et avec, l'ajustement. A cause de la pénalisation des erreurs et des individus à l'intérieur de la marge, une valeur plus élevée du paramètre diminuera la largeur de la marge et la capacité de généralisation du modèle, mais il sera plus précis sur l'échantillon d'apprentissage, avec moins d'erreur de classement. La valeur de ce paramètre doit être bien choisie pour garantir un bon compromis entre l'ajustement et la robustesse.

Nous testons les valeurs [0.001,0.01, 0.1, 1, 10] pour ce paramètre.

Pour chacune des valeurs, on obtient une AUC moyenne à partir des 5 modèles obtenus par validation croisée k-fold, sur leurs fractions de test :

(0.9695138607751133, 0.9702863093580703, 0.9722201807301654, 0.9700058080265842, 0.933906703866121).

Nous retenons la valeur 0.1 pour l'hyper-paramètre regParam, puisque c'est pour cette valeur qu'on obtient l'AUC moyenne la plus élevée.

Ci-dessous les valeurs de l'AUC et de l'Accuracy obtenues sur les données d'apprentissage et de test pour ce modèle :

	Apprentissage	**Test**
AUC	0,97917	0,97446
Accuracy	0,93339	0,92361

c) *Les forêts aléatoires*

L'algorithme des forêts aléatoires propose une optimisation des arbres de décision. Les forêts aléatoires font partie des méthodes dites d'agrégation. Elles utilisent le même principe que le bagging, mais avec une étape supplémentaire de randomisation dans la sélection des variables explicatives à la scission de chaque noeud afin de réduire la variance de l'estimateur obtenu. La

randomisation de la sélection des variables permet d'éviter de voir toujours apparaître les variables avec le plus fort pouvoir discriminant, diminue la corrélation entre les arbres et diminue donc la variance du modèle agrégé.

Les forêts aléatoires possèdent les mêmes hyper-paramètres que les arbres de décision :

- ➔ maxDepth : profondeur de l'arbre. La valeur par défaut est 5.
- ➔ maxBins : précise le nombre de catégorie à retenir pour discrétiser les variables continues.
- ➔ Impurity : indice d'impureté « entropy » ou « gini » (par défaut) qui permet de déterminer à quel moment un arbre de décision doit se ramifier.
- ➔ minInfoGain : détermine le gain minimum pour qu'un nœud soit encore divisé
- ➔ minInstancesPerNode : nombre minimum d'observations présentes à l'étape d'un nœud pour envisager une scission (en dessous de ce nombre, le nœud devient alors une feuille). C'est une autre manière de contrôler la profondeur de l'arbre.

Et elles possèdent aussi leurs propres hyper-paramètres :

- ➔ numTrees : le nombre d'arbres agrégés
- ➔ featureSubsetStrategy : nombre de prédicteurs sélectionnés pour la scission de chaque nœud. Si le nombre d'arbres est supérieur à 1, la valeur par défaut retenue est la racine carrée du nombre total de prédicteurs.

La documentation Spark précise que les deux paramètres les plus susceptibles d'améliorer la performance des forêts aléatoires sont le nombre d'arbre agrégés (numTrees) et la taille des arbres (maxDepth).

Nous choisissons de tester une unique valeur, assez élevée (200), pour le nombre d'arbres agrégés afin de limiter le temps de calcul.

Nous testons, par contre, différentes valeurs du paramètre maxDepth : (5,7,8,10,15). Une valeur plus élevée de maxDepth assure une plus grande robustesse de chaque arbre, mais risque d'augmenter la corrélation entre les arbres et diminuer le gain de l'agrégation. Pour les autres paramètres, ce sont les paramètres par défaut qui sont retenus.

Pour chacune des valeurs du paramètre maxDepth, on obtient une AUC moyenne à partir des 5 modèles (validation croisée k-fold) sur leurs fractions de test :

(0.8990930549459026, 0.9074410266145037 , 0.9112226274253507 , 0.9168945307728416, 0.91635169421)

Nous retenons la valeur 10 pour l'hyper-paramètre maxDepth, puisque c'est pour cette valeur qu'on obtient l'AUC moyenne la plus élevée.

Ci-dessous les valeurs de l'AUC et de l'Accuracy obtenues sur les données d'apprentissage et de test pour ce modèle :

	Apprentissage	**Test**
AUC	0,9135	0,9118
Accuracy	0,8124	0,8102

d) Courbes ROC

Afin de pouvoir tracer les courbes ROC, nous avons fait appel à l'API « org.apache.spark.mllib » basée sur les RDD, car cette métrique n'a pas été reportée dans l'API « org.apache.spark.ml » basée sur les Dataset/DataFrame.

Ci-dessous les courbes ROC obtenues sur les données de test pour les 3 modèles retenus.

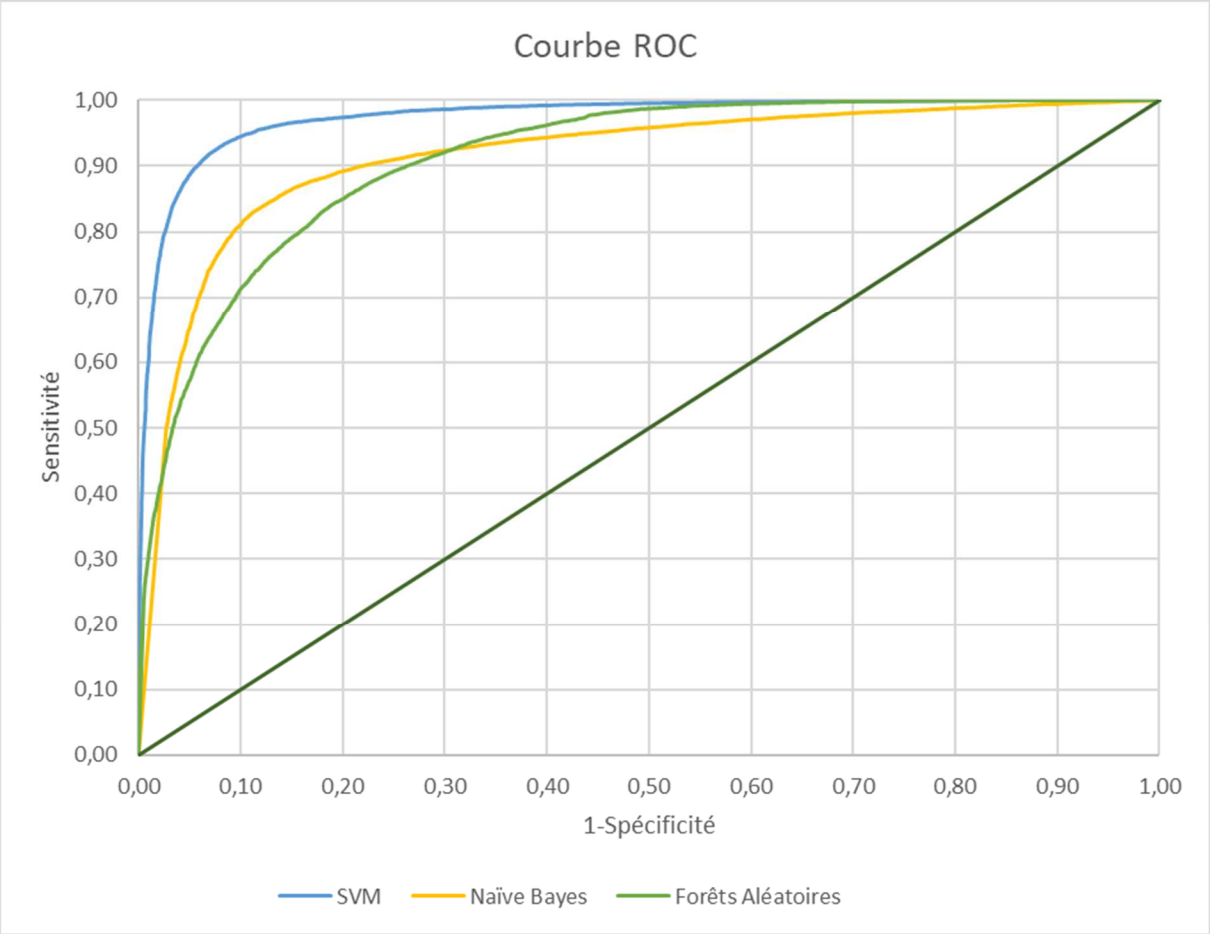


Figure 5 - Courbes ROC

Le modèle des SVM est toujours supérieur au modèle des Naives Bayes et au modèle des forêts aléatoires, tandis que le modèle des Naïve Bayes est tantôt supérieur, tantôt inférieur aux forêts aléatoires.

3. Sélection de variables

Dans la première partie, nous avons limité arbitrairement notre « vocabulaire » aux 20 000 lemmes les plus fréquents dans le corpus afin de limiter le nombre de variables explicatives. En effet, avec un trop grand nombre de variables explicatives, on peut se retrouver avec de nombreuses variables corrélées ou avec un modèle sur-ajusté.

Plutôt que de choisir les 20 000 lemmes les plus fréquents, Spark propose la sélection de variables à l'aide du test d'indépendance du χ^2 grâce à l'option ChiSqSelector de l'API « org.apache.spark.ml ».

ChiSqSelector identifie les variables explicatives qui ne sont pas indépendantes de la variable à expliquer et supprime les autres. Une table de contingence est construite à partir des variables explicatives et de la variable cible. Chaque entrée (i, j) correspond à une caractéristique i et à une classe j, et contient la somme des valeurs de la i ème caractéristique dans tous les échantillons appartenant à la classe j. On calcule ensuite la statistique du test χ^2. Cela fonctionne avec les fréquences de termes car la somme sera la fréquence totale d'un terme pour cette classe. Cela fonctionne aussi lorsque les valeurs sont des valeurs TF-IDF, car ce ne sont que des fréquences pondérées.

Nous appliquons ChiSqSelector à notre variable cible et à notre matrice « Document x Terme » avec la pondération TF-IDF et retenons les 20 000 variables avec le plus de pouvoir prédictif.

Ci-dessous, l'AUC et l'accuracy en apprentissage et en test pour les Naïve Bayes et les SVM.

	Apprentissage		**Test**	
	AUC	**Accuracy**	**AUC**	**Accuracy**
Naïve Bayes	0,9216	0,8671	0,9131	0,8582
SVM	0,9795	0,9346	0,9746	0,9237

La sélection de variable avec ChiSqSelector donne des résultats similaires aux précédents et n'a donc pas permis d'améliorer la performance de nos classifieurs.

4. Prise en compte des tri-grammes

Les mots de notre corpus sont des 1-grammes, qui peuvent se généraliser en n-grammes, c'est-à-dire en séquence de n mots consécutifs. Les n-grammes permettent une analyse plus subtile que la seule analyse des mots, en prenant en compte les négations et le contexte entourant les mots.

Dans la suite, nous testerons deux méthodes pour la prise en compte des tri-grammes :

- ➔ Sélection des tri-grammes les plus fréquents dans le corpus
- ➔ Sélection des tri-grammes avec les SVM

a) Tri-grammes les plus fréquents

Dans un premier temps, nous effectuons une lemmatisation du corpus mais sans suppression des stops words, nécessaires à la formation des tri-grammes. Nous choisissons ensuite les tri-grammes qui apparaissent plus de 100 fois dans le corpus afin d'éliminer les tri-grammes rares, peu porteurs d'information. Cela représente 30 381 tri-grammes distincts.

Ci-dessous les 40 tri-grammes les plus fréquents dans le corpus :

Tri-gramme	freq
the first time	12 475
have ever have	12 282
will not back	10 273
you can not	10 153
and the food	10 129
the fact that	9 661
love this place	9 532
this place have	9 136
about this place	7 803
the front desk	7 789
and they have	7 608
you look for	7 448
this place the	7 256
one the best	7 240
and the service	7 227
the only thing	7 108
give this place	7 076
they not have	6 812
for the price	6 764
that they have	6 584

Tri-gramme	freq
the rest the	6 550
recommend this place	6 421
you will not	6 228
the food and	6 150
you can get	5 951
this place and	5 863
the next day	5 710
the food not	5 687
have never have	5 508
but the food	5 490
they also have	5 378
try this place	5 371
the food good	5 365
for the first	5 322
not know what	5 306
not worth the	5 260
come here for	5 094
very friendly and	4 979
they have the	4 966
the only reason	4 789

On voit que les tri-grammes permettent de prendre en compte la négation avec des expressions comme « will not back », « you can not », « they not have » ou encore « not worth the ». On peut imaginer que ces expressions se réfèrent à un sentiment négatif, ce qui n'était pas détectable avec la séparation de chaque lemme.

Les lemmes composant les tri-grammes sont ensuite réunis avec un sous-tiret « _ », et sont remplacés dans le texte lemmatisé d'origine. On peut alors supprimer les Stop Words.

Ci-dessous le résultat pour les cinq premiers documents de l'échantillon d'apprentissage :

Text	1-gramme et tri-gramme
I found this gem on Yelp (go figure) and have taken numerous shirts, jackets, pants, and dresses etc.to be cleaned. This company lives up to its name. The staff is prompt, professional, and reliable. The owner takes customer service very seriously and values customer loyalty. I used the spend $30 for $50 coupon on the yelp site and redeemed it from my phone. It was a true breeze. The owner (I'm assuming) explained they have delivery service free of charge. I agree with his philosophy- dry cleaning should be hassle free. Quality work every time, no exceptions. Not necessarily the cheapest, but absolutely the best. You get what you pay for. Cheers! One request for the owner: please put a fan (or anything to cool) in the receiving area. That place can get toasty for those that choose to pick up in person.	[find, gem, yelp, figure, take, numerous, shirt, jacket, pants, dress, clean, company, live, name, staff, prompt, professional, reliable, owner, take, customer, service, seriously, value, customer, loyalty, use, spend, coupon, yelp, site, redeem, phone, true, breeze, owner, assume, explain, delivery, service, free, charge, agree, with, philosophy, dry, cleaning, hassle, free, quality, work, every, time, exception, necessarily, cheapest, ***absolutely_the_best***, get, pay, cheers, one, request, owner, please, put, fan, anything, cool, receive, area, place, get, toasty, choose, pick, person]

I had no problem redeeming the 30-50 coupon. The staff was very friendly and knowledgeable. The clothes came out clean, except one that said they couldn't take the stains off without damaging the item, that item was charge which I don't think it should; since it was not cleaned. The price on my personal opinion it's very expensive. I took 11 pieces and paid with coupon over $80.	[problem, redeem, coupon, staff, ***very_friendly_and***, knowledgeable, clothes, come, clean, except, one, say, ***they_could_not***, take, stain, without, damage, item, item, charge, think, since, clean, price, personal, opinion, expensive, take, piece, pay, coupon]
Best customer service I've ever had at a cleaner! They were so friendly and fast. I've used them twice and will always use them.	[best, customer, service, ***have_ever_have***, cleaner, ***they_friendly_and***, fast, use, twice, ***and_will_always***, use]
Excellent service in every way. Been using them for years. Always accommodating, pick up by Jeff always smiles and prompt. Dave is the best owner.	[excellent, service, every, use, year, always, accommodate, pick, jeff, always, smile, prompt, dave, best, owner]
I'm a little biased since I used to work here. Nonetheless, it's a great nursery. It's got quality plants, quality service and quality gardening supplies. The customer service is especially good and the care taken of the plants is outstanding. The only drawback is the price, but with the quality of everything so good, what else do you expect?	[little, bias, since, use, work, nonetheless, great, nursery, get, quality, plant, quality, service, quality, gardening, supplies, customer, service, especially, good, care, take, plant, outstanding, ***the_only_drawback***, price, quality, everything, good, ***what_else_you***, expect]

Nous transformons les nouveaux textes en matrice « Document x Termes » avec une pondération TF-IDF et limitons la taille du vocabulaire aux 20 000 termes (unigrammes et trigrammes) les plus fréquents.

Dans la suite, nous utilisons les 3 mêmes algorithmes de classification supervisée : un classifieur bayésien naïf, les SVM et les forêts aléatoires. Pour chacun des algorithmes, afin de déterminer le meilleur modèle décisionnel, nous avons effectué une recherche en grille et une validation croisée pour les mêmes paramètres et en testant les mêmes valeurs que dans la première partie. Nous comparerons les performances des meilleurs modèles retenus à l'aide de leur AUC (Aire sous la courbe ROC), de leur Accuracy et de la représentation de leurs courbes ROC.

AUC	**Apprentissage**	**Test**
Naive Bayes	0,9360	0,9330
SVM	0,9805	0,9763
Random Forest	0,9015	0,8992

Accuracy	**Apprentissage**	**Test**
Naive Bayes	0,8821	0,8767
SVM	0,9344	0,9253
Random Forest	0,7997	0,7976

Ci-dessous les courbes ROC obtenues sur les données de test pour les 3 modèles retenus.

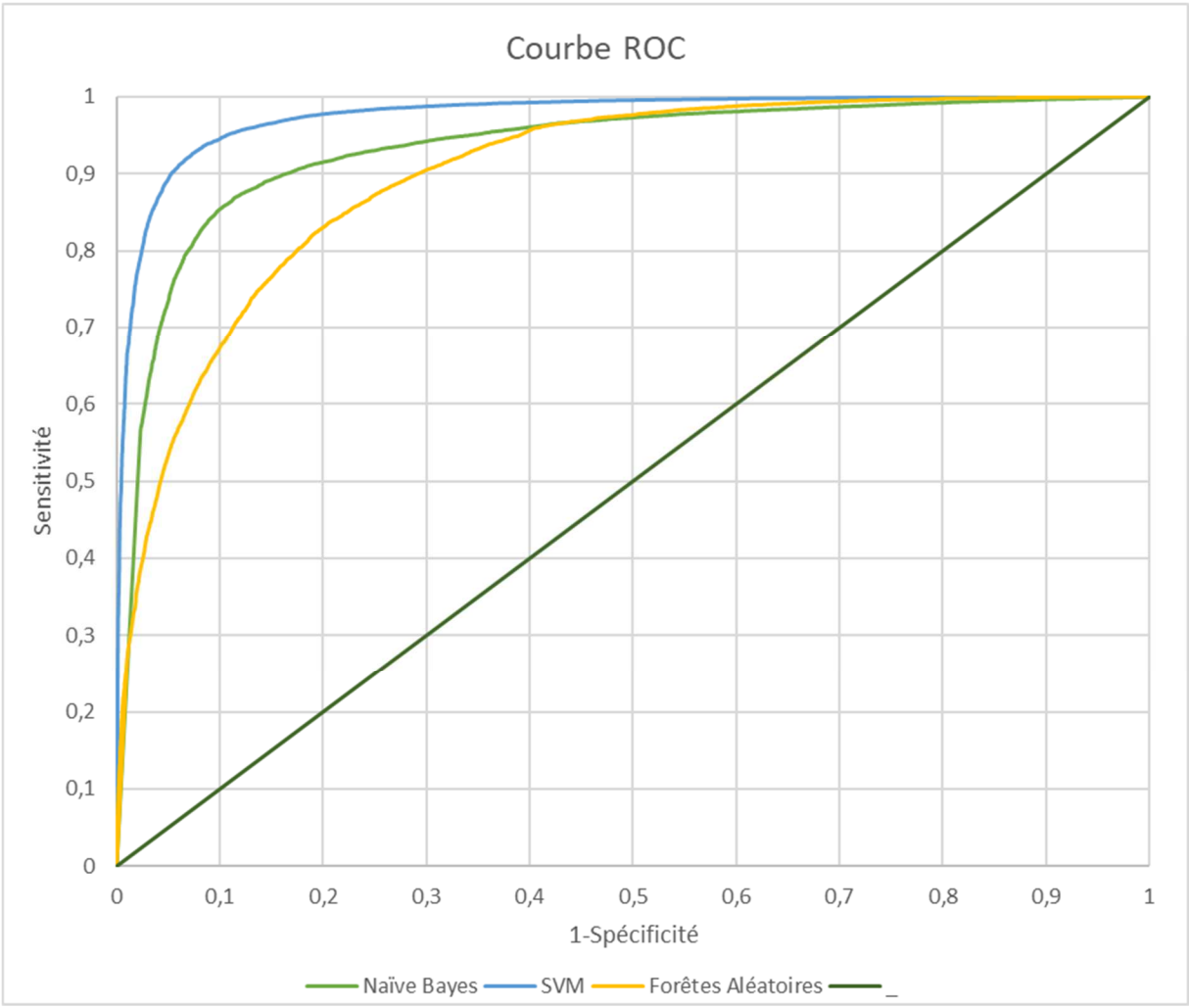


Figure 6 - Courbes ROC

Le modèle des SVM est toujours supérieur au modèle des Naives Bayes qui est toujours supérieur au modèle des forêts aléatoire

b) Sélection des tri-grammes avec les SVM

La sélection de variables consiste à identifier les variables pertinentes par rapport au problème considéré (dans notre cas la discrimination à deux classes, avis positifs ou négatifs, à partir des pondérations TF-IDF pour les uni-grammes et les tri-grammes). Si la sélection des variables explicatives est réalisée de manière appropriée, la performance du modèle de discrimination peut être fortement améliorée tout en réhaussant l'interprétabilité du modèle.

Précédemment, nous avons retenu les tri-grammes dont la fréquence était supérieure à 100 au sein de notre corpus.

Dans cette partie, pour sélectionner les tri-grammes les plus pertinents, nous déterminons la matrice « Document x Termes » avec la pondération TF-IDF pour tous les tri-grammes de notre corpus préalablement lemmatisé. Nous appliquons ensuite un modèle SVM à notre variable cible et aux variables explicatives de la matrice « Document x Termes » avec pondération « TF-IDF », obtenues à partir des tri-grammes (les 20 000 les plus fréquents dans le corpus).

Les coefficients obtenus représentent les coordonnées vectorielles orthogonales à l'hyperplan et leur direction indique la classe prédite. La valeur absolue des coefficients, les uns par rapport aux autres, peut ensuite être utilisée pour déterminer l'importance des prédicteurs pour la séparation des classes.

coeff	voc
0.6000930926925842	will not back
0.3654390323163257	will not return
0.3450294679731154	not worth the
0.2819685364226226	not come back
0.2581047666453786	will never back
0.23803864059085392	one the worst
0.23215377534341872	not very good
0.23119226753730993	not recommend this
0.22933982489176724	use love this
0.2122733197997213	this the worst
0.20398057436944428	would not recommend
0.20317274561846502	have have better
0.20299074564393882	this place suck
0.20145122750849892	would not back
0.2014307302194402	never come back
0.19672122463918065	horrible customer...
0.19564432546997135	avoid this place
0.19201194214040093	poor customer ser...
0.1886200815459369	avoid all cost
0.18853698793270593	save you money

coeff	voc
-0.32479208309176266	love this place
-0.25557761377667726	will definitely back
-0.24877397639659	one the best
-0.1998761826605621	highly recommend ...
-0.18893797433975665	definitely come back
-0.1881033768356991	they also have
-0.16470174008704289	can not wait
-0.16038473261803737	great place for
-0.16005202512280148	can not beat
-0.15980235197785808	great customer se...
-0.15970981016203142	only complaint that
-0.15703099671128998	great food and
-0.1548517107325457	not wait back
-0.15467880174392454	this the best
-0.15407901030711504	above and beyond
-0.15279237640145107	this place awesome
-0.14912747437851873	well worth the
-0.14845103546810032	would definitely ...
-0.1479269761135705	this place great

Nous retenons les 1 000 trigrammes avec les coefficients positifs les plus élevés et les 1000 trigrammes avec les coefficients négatifs les plus élevés. Ces trigrammes sont réintroduits dans notre corpus pour ne pas être pris en compte comme un seul mot mais plusieurs dans notre matrice TF-IDF finale. Nous limitons le vocabulaire de notre matrice TF-IDF aux 20 000 termes les plus fréquents du corpus.

	Apprentissage	Test
AUC	0,9849	0,9807
Accuracy	0,9445	0,9351

Nous pouvons utiliser les coefficients de notre modèle SVM final pour voir quels termes contribuent le plus à un avis positif ou négatif. Ci-dessous 2 nuages de mots, avec en rouge les 50 termes les plus négatifs et en bleu les 50 termes les plus positifs, d'après les coefficients de notre modèle SVM final :

Figure 7 - Nuage de mots « Termes les plus négatifs »

Figure 8 - Nuage de mots « Termes les plus positifs »

Les termes qui contribuent le plus aux avis négatifs concernent :

- le souhait de ne pas revenir: « will_not_back », « not_think_back », « not_come_back », « will_not_return »)
- les prix jugés trop élevés: «way_over_price », « too_expensive_for »
- la nourriture jugée décevante: « food_below_average », « the_food_mediocre », « get_food_poisoning »
- le fait de déconseiller : «skip_this_one », « try_somewhere_else »
- une mauvaise expérience: «disappointed», «just_not_impressed », « this_place_suck », « bland », « horrible »

Les termes qui contribuent le plus aux avis positifs concernent :

- le souhait de revenir: « will_definitely_back », « would_definitely_return», « would_definitely_stay», « definitely_come_back»
- une appréciation positive de la nourriture : « delicious », « lickin », « leckere », « great_food_great »
- le fait de recommander: « would_definitely_recommend », « highly_recommend_this»
- une bonne expérience: « great», « wondrous », « this_place_amazing », «this_place_awesome», « love_this_place», «great_service_great», «overall_good_experience»
- le fait d'avoir une unique doléance: « the_only_negative », « the_only_downside», « only_complaint_that»

5. Analyse de sentiment avec Spark NLP

Spark NLP est une bibliothèque développée pour Spark par la société John Snow Labs NLP et permettant de réaliser un assez grand nombre d'opérations de traitement automatique pour plusieurs langues au travers de traitements séparés ou « annotators ».

Il existe deux types d' «annotators »:

- ➔ AnnotatorApproach : ils doivent être entraînés sur des données avec une méthode fit()
- ➔ AnnotatorModel : ils sont destinés à transformer les dataframes avec une méthode transform ().

Spark NLP propose également des « pipelines » pré-entraînés, qui sont des regroupements de plusieurs « annotators ». Un de ces pipelines, « analyze_sentiment », a pour objectif d'associer à chaque phrase une étiquette indiquant si les sentiments transmis par une phrase sont positifs ou négatifs.

Ce pipeline, utilise plusieurs « annotators » de la bibliothèque Spark-NLP :

- ➔ Détection de phrases
- ➔ Tokenisation
- ➔ Vérification orthographique
- ➔ Détection de sentiments.

La vérification orthographique est basée sur le modèle Norvig et l'analyse du sentiment est basée sur l'algorithme Vivekn Sentiment. Cet algorithme a pour vocation d'améliorer la précision d'un classifieur bayésien naïf pour l'analyse des sentiments grâce à la gestion de la négation, la prise en compte des n-grammes et la sélection de variables via le critère de l'information mutuelles entre la variable cible et les prédicteurs.

Ce pipeline s'applique à des phrases et les documents de notre corpus sont des textes courts pouvant être composés de plusieurs phrases, nous supprimons donc la ponctuation de notre corpus avant l'application du pipeline pré-entrainé sur notre échantillon de test.

On obtient un taux de bon classement de 59% et la matrice de confusion ci-dessous :

		Classe estimée par le classifieur	
		0	**1**
Classe réelle	**0**	10 489	8 477
	1	7 039	11 932

L'application du pipeline pré-entraîné « analyze_sentiment » directement sur nos données ne donnent pas de très bons résultats. On peut imaginer que cela est dû au fait que le classifieur bayésien naïf utilisé dans ce pipeline, a été entraîné sur des données qui n'ont pas forcément les mêmes caractéristiques que les nôtres, en particulier concernant les sujets abordés dans les documents.

III. Conclusion

Nous avons testé plusieurs méthodes afin d'effectuer une analyse de sentiments à partir du texte des avis et des notes des utilisateurs postés sur le site Yelp. Dans un premier temps, nous avons effectué un pré-traitement des données afin des les rendre exploitables par des algorithmes de classification supervisée : lemmatisation, suppression des Stop Words, vectorisation des textes avec une matrice « Documents x Termes » et une pondération TF-IDF.

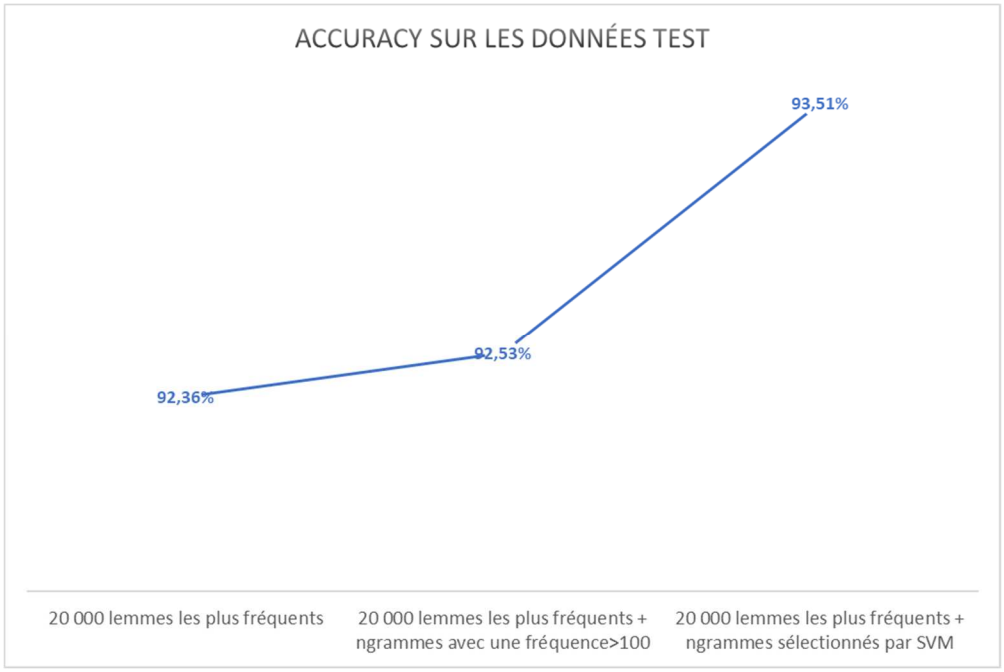
Nous avons limité arbitrairement notre « vocabulaire » aux 20 000 lemmes les plus fréquents dans le corpus afin de limiter le nombre de prédicteurs pour éviter un sur-apprentissage de nos modèles. Puis nous avons évaluer notre pré-traitement et notre méthode de vectorisation en utilisant 3 modèles de classification supervisée : un classifieur bayésien naïf, les SVM et les forêts aléatoires. Pour chacun des algorithmes, afin de déterminer le meilleur modèle décisionnel, nous avons effectué une recherche en grille avec validation croisée k-fold pour la sélection des hyper-paramètres. Nous avons comparé les performances des modèles retenus à l'aide de leur AUC (Aire sous la courbe ROC), de la représentation de leur courbe ROC et de leur Accuracy. Les meilleurs résultats ont été obtenu avec le modèle SVM.

Nous avons aussi testé la sélection de variables à l'aide du test d'indépendance du χ^2 grâce à l'option ChiSqSelector. Nous avons obtenu des résultats comparables aux précédents.

Par la suite, afin d'améliorer notre représentation vectorielle des textes des documents du corpus d'apprentissage, nous avons cherché à prendre en compte les tri-grammes. En effet, ils permettent une analyse plus subtile que la seule analyse des mots, en tenant compte des négations et du contexte entourant les mots. Nous avons sélectionné les tri-grammes les plus fréquents (freq>100) et nous les avons remplacés dans le corpus initial en un seul mot avant application de la vectorisation « Documents x Termes » avec pondération TF-IDF et recherche du meilleur modèle. On obtient de meilleur résultat qu'avec la représentation vectorielle précédente quelque soit le modèle et parmi ces modèles, les meilleurs résultats ont été obtenu avec les SVM.

Nous avons ensuite appliqué une deuxième méthode pour la sélection des tri-grammes les plus pertinents. Nous avons retenu les 1 000 trigrammes avec les coefficients positifs les plus élevés et les 1000 trigrammes avec les coefficients négatifs les plus élevés issus du modèle SVM appliqué à notre variable cible et aux variables explicatives de la matrice « Document x Termes » avec pondération « TF-IDF », obtenue à partir des 20 000 les trigrammes les plus fréquents dans le corpus. Cette sélection nous a permis d'améliorer encore les résultats obtenus avec les SVM.

Ci-dessous, l'évolution de l'accuracy avec le classifieur SVM et les différents pré-traitements :



IV. Références

- [1] http://www.yelp.com/dataset_challenge
- [2] Zhang, X., Zhao, J. and LeCun, Y., 2015. Character-level convolutional networks for text classification. In Advances in neural information processing systems (pp. 649-657).
- Cours CNAM NFE204 : http://b3d.bdpedia.fr/
- Cours CNAM RCP216 : https://cedric.cnam.fr/vertigo/Cours/RCP216/
- Cours CNAM STA211
- Vivek Narayanan, Ishan Arora, Arjun Bhatia 2013. Fast and accurate sentiment classification using an enhanced Naive Bayes model
- Spark, the definitive guide. Bill Chambers & Matei Zaharia
- Data Mining et Statistiques décisionelle. Stéphane Tufféry