QFM

Rapport du projet S3

A big data architecture for product reputation monitoring and tracking.

EWINSOU Damilola Roméo
NDAO Alioune Badara
TINA Djara Olivier

Professeur encadrant :
Pr. FAHD KALLOUBI

A tous les professeurs de l'UM6P. Merci pour la formation de qualité que vous nous avez donné tout au long de cette période d'études.

RESUME

Le projet "A big data architecture for product reputation monitoring" explore une approche complète pour le suivi de la réputation des produits. En utilisant une architecture big data intégrant divers outils tels que Apache Airflow, YouTube API,Apache Kafka, Spark, ONNX, Streamlit, et Hadoop HDFS, et le webScraping du site d'Amazon le projet collecte, traite, et analyse des données provenant de multiples sources. L'accent est mis sur la conception d'une architecture robuste permettant une gestion efficace des flux de travail, la mise en place de modèles de langage (LLMs) tels que BERT et DistilBERT pour l'analyse de sentiment, et le développement d'une application web interactive avec Streamlit. Le projet offre ainsi une solution complète pour la surveillance proactive de la réputation des produits.

ABSTRACT

The "A big data architecture for product reputation monitoring" project explores a comprehensive approach for product reputation monitoring. Using a big data architecture integrating various tools such as Apache Airflow, YouTube API, Apache Kafka, Spark, ONNX, Streamlit, and Hadoop HDFS, and webScraping of the Amazon website, the project collects, processes, and analyzes data from multiple sources. The focus is on the design of a robust architecture enabling efficient workflow management, the implementation of language models (LLMs) such as BERT and DistilBERT for sentiment analysis, and the development of an interactive web application with Streamlit. The project thus offers a complete solution for proactive monitoring of product reputation.

Table des matières

1 Chapitre I : Introduction et contexte du projet

1.1 Introduction générale

À l'ère numérique actuelle, la réputation des produits joue un rôle essentiel dans les décisions d'achat des consommateurs. La montée en puissance du commerce électronique a considérablement accru l'importance de surveiller et de suivre la réputation des produits en temps réel. Dans ce contexte, notre projet se concentre sur le développement d'une architecture Big Data dédiée à la surveillance et au suivi de la réputation des produits. La multitude de plateformes en ligne et de canaux de communication a créé un paysage complexe où les opinions des consommateurs, les avis en ligne, et les tendances sociales influent directement sur la perception des produits. Afin de capturer cette dynamique et d'en tirer des informations exploitables, nous avons conçu une architecture Big Data capable d'ingérer, de traiter et d'analyser massivement les données liées à la réputation des produits. Notre projet vise à répondre à des questions cruciales telles que la manière dont les commentaires sont analysés à l'aide de Large Language Models (LLMs), l'impact du web scraping pour collecter des données provenant du site Amazon, et comment les commentaires en temps réel peuvent améliorer la surveillance des produits. En exploitant les technologies Big Data, notre architecture offre une approche innovante pour anticiper et analyser le sentiment global du marché concernant un produit.

1.2 Context du projet

L'objectif dans ce rapport de défense est de proposer une architecture Big Data pour la surveillance et le suivi de la reputation des produits. Nous explorerons les différents composants de l'architecture, en nous concentrant sur la collecte, le stockage, le traitement et l'analyse des mégadonnées nécessaires pour surveiller l'évolution de la réputation d'un produit.

Le rapport de ce mémoire de projet de fin de semestre se présente comme suit :

Dans le chapitre II nous discuterons des différentes tehnologies et concepts utilisés pour la réalisation de ce projet et aussi l'approche qu'on a adopté pour faire la surveillance et le suivi de la réputation d'un produit en temps réel.

Le troisième chapitre est dédié aux résultats et discussion. Et enfin, on en terminera ce rapport avec une conclusion générale et des perspectives.

1.3 Questions de recherche

Nous cherchons à repondre aux questions suivantes :

— Comment exploiter une architecture Big Data pour surveiller en temps réel la réputation des produits d'Amazon ?

— En quoi l'analyse des commentaires des vidéos de youtube rélatifs à un produit peut-elle contribuer à une évaluation plus précise de la réputation

de ce dernier ?
- Comment les Large Language Models (LLMs) peuvent apporter une précision dans l'analyse des commentaires rélatifs aux produits ?
- Pourquoi mettre en place une architecture big data pour la suivi de la réputation d'un produit ?
- Comment l'utilisation du web scraping pour collecter les produits provenant du site Amazon ameliore t-il la structure de notre solution finale ?

C'est à ces questions que nous avons essayé de répondre à travers ce project et dans la suite de ce rapport, nous y apportons des réponses. En répondant à ces questions de recherche, le projet vise à donner une idée sur le sentiment du public concernant un produit, en utilisant des techniques d'analyse de données massives, d'analyse des sentiments avec les LLMs et du webScraping

1.4 Conclusion

En conclusion de ce chapitre, nous avons examiné en détail l'importance de l'utilisation du Big Data pour surveiller et suivre la réputation d'un produit en temps réel. L'architecture Big Data, combinée à l'analyse des sentiments, peut offrir de nouvelles perspectives prometteuses dans ce domaine.

Dans le prochain chapitre, nous explorerons en détail les différentes technologies utilisés pour la réalisation de ce projet, ainsi que l'approche spécifique que nous avons adoptée pour mettre en place notre solution de surveillance de reputation d'un produit.

2 Chapitre II : Technologies et approche

2.1 Introduction

Le présent chapitre se concentre sur les technologies et approches clés utiliséés dans notre projet pour la mise en place de l'Architecture Big Data et de l'interface.

Nous commencerons par discuter sur les surces de données et comment nous avons recupérer les données. Dans ce project nous avons plusieurs source de données allant du site de l'Amazon à Youtube pour la recuperation des commentaire.

Ensuite, nous allons explorer les concepts de Big data tels que Apacche Airflow, Apache spark, Apache Kafka, docker, hadoop HDFS, streamlit, et les métriques de performance de notre modèle comme l'accurary.

Nous aborderons aussi les modèles utilisées en évaluant leurs performances, leurs avantages et limites dans la l'analyse des sentiments.

Nous allons également parler de l'interface streamlit que nous allons mettre en place pour le monitoring de la reputation des produits.

Enfin nous expliquerons l'architecture de notre système pour la réalisation de notre projet.

Sans plus tarder, plongeons dans l'exploration des technologies et des concepts essentiels pour améliorer notre compréhension de la prédiction des cours des actions en temps réel.

2.2 Youtube API

L'API YouTube offre une interface puissante permettant aux développeurs d'accéder aux commentaires associés à une vidéo spécifique. Pour utiliser cette fonctionnalité, plusieurs étapes doivent être suivies.

2.2.1 Obtention de la Clé d'API

Avant toute interaction avec l'API YouTube, une clé d'API doit être obtenue à partir de la console de développement Google. Cette clé servira à authentifier les requêtes et à garantir un accès autorisé aux données.

2.2.2 Identification de la Vidéo Cible

Pour récupérer les commentaires d'une vidéo spécifique, l'identifiant unique de cette vidéo doit être déterminé. Cet identifiant peut être extrait de l'URL de la vidéo YouTube.

2.2.3 Construction de la Requête API

La récupération des commentaires se fait en envoyant une requête HTTP GET à l'URL approprié de l'API YouTube. Certains paramètres essentiels, tels que l'identifiant de la vidéo et la clé d'API, doivent être inclus dans la requête.

```
GET https://www.googleapis.com/youtube/v3/commentThreads
?part=snippet
&videoId={VIDEO_ID}
&key={YOUR_API_KEY}
```

Dans notre contexte, comme nous cherchons à récupérer plusieurs vidéos, nous avons écrit une fonction qui permet de collecter les identifiants des vidéos à partir du nom de produit que l'utilisateur entre dans l'interface Streamlit, dont nous parlerons un peu plus bas.

2.2.4 Réception de la Réponse JSON

Une fois la requête soumise avec succès, l'API renvoie une réponse au format JSON. Cette réponse contient des détails sur les commentaires, tels que l'auteur, le texte, la date de publication, etc.

Exemple de réponse JSON :

```
{
   "kind": "youtube#commentThreadListResponse",
   "items": [
      {
         "kind": "youtube#commentThread",
         "snippet": {
            "topLevelComment": {
               "snippet": {
                  "authorDisplayName": "Utilisateur123",
                  "textDisplay": "Un commentaire intéressant!",
                  "publishedAt": "2024-01-26T12:34:56Z"
               }
            }
         }
      },
      // Autres commentaires...
   ]
}
```

2.2.5 Traitement des Données dans l'Application

Les données JSON peuvent ensuite être traitées dans l'application pour extraire et afficher les commentaires de manière adaptée à l'interface utilisateur.

2.3 Web scraping des données d'Amazon

Le web scraping, est le processus de construction d'un agent capable d'extraire, d'analyser, de télécharger et d'organiser automatiquement des informations utiles à partir du Web.

Il nous a permis de recupérer les données d'Amazon. Nous avons essayé d'extraire les sous-catégories d'une catégorie spécifique d'un produit sur la page d'Amazon en utilisant Selenium et BeautifulSoup.

2.3.1 Selenium

Selenium est un framework open-source initialement développé pour exécuter des tests automatisés sur différents navigateurs (Chrome, Internet Explorer, Firefox, Safari,...). Cette librairie est également utilisée pour effectuer du web scraping, car elle permet de naviguer entre des pages internet et d'interagir avec les éléments de la page comme un utilisateur réel. Ainsi, il devient possible de scraper des sites web dynamiques, c'est-à-dire, des sites qui retournent un certain résultat à l'utilisateur en fonction de ses actions.

2.3.2 BeautifulSoup

BeautifulSoup est une bibliothèque Python qui facilite l'extraction d'informations à partir de pages web. Elle se place au-dessus d'un analyseur HTML

ou XML et fournit des idiomes Python pour l'itération, la recherche et la modification de l'arbre d'analyse.

2.3.3 Recupération des données

Pour la recupération des données d'Amazon, nous avons écrit une fonction qui extrait les sous-catégories d'une catégorie spécifique d'un produit :

1. Importe les modules nécessaires tels que time, requests, BeautifulSoup de bs4, et webdriver de selenium.
2. Définit un en-tête HTTP pour simuler une requête provenant d'un navigateur web standard.
3. Définit une fonction getSubCategorie qui prend une catégorie en tant qu'argument.
4. Utilise le module selenium pour automatiser un navigateur Chrome.
5. Navigue vers le site web Amazon.co.uk.
6. Sélectionne la catégorie spécifiée à l'aide du sélecteur d'ID 'searchDropdownBox'.
7. Clique sur le bouton de recherche pour afficher les résultats de la catégorie sélectionnée.
8. Récupère l'URL actuelle du navigateur après la sélection de la catégorie.
9. Utilise requests pour effectuer une requête HTTP à l'URL récupérée afin de récupérer le contenu de la page.
10. Utilise BeautifulSoup pour analyser le contenu HTML de la page.
11. Recherche et extrait les sous-catégories de la catégorie spécifiée à partir du HTML analysé.
12. Retourne une liste des noms de sous-catégories et l'URL actuelle.

En fin de compte, la fonction getSubCategorie est appelée avec la catégorie 'Beauty' pour démontrer son fonctionnement, imprimant la liste des sous-catégories et l'URL correspondante.

2.4 Docker

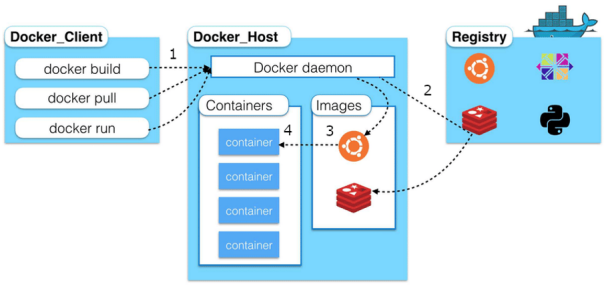
2.4.1 Définition

Docker est une plateforme de conteneurs lancée en 2013 ayant largement contribué à la démocratisation de la conteneurisation. Elle permet de créer facilement des conteneurs et des applications basées sur les conteneurs. Il en existe d'autres, mais celle-ci est la plus utilisée. Elle est par ailleurs plus facile à déployer et à utiliser que ses concurrentes.
C'est une solution open source, sécurisée et économique. De nombreux individus et entreprises contribuent au développement de ce projet. Un large écosystème de produits, services et ressources sont développés par cette vaste communauté. Initialement conçue pour Linux, Docker permet aussi d'exécuter des conteneurs

sur Windows ou Mac grâce à une " layer " de virtualisation Linux entre l'OS Windows / MacOs et l'environnement runtime Docker. Il est donc possible d'exécuter des conteneurs Windows natifs sur des environnements de conteneurs Windows ou Linux.

2.4.2 Fonctionnement interne de docker

Le fonctionnement interne de Docker est résumé à travers l'image suivante :



Fonctionnement interne de Docker

Une requête de construction d'image est envoyée (docker build) ou une requête d'utilisation d'image existante est envoyée (docker pull, docker run).
La deuxième étape consiste à trouver l'image qui va être exécuté ou bien construire l'image.
Dans le cas où l'image est non-disponible localement, alors le processus Docker va la trouver en ligne dans le registre configuré (par défaut docker.io). Un registre Docker contient les images pré-construite prêtes à être utilisées.
Un coup l'image construite ou téléchargée, alors on continue vers l'étape 3.
Si l'image est disponible localement, alors on ignore le registre en ligne de docker et on va à l'étape 3.
Dans le cas de la construction, le contexte (l'arborescence de fichier au même niveau que le fichier «Dockerfile») est envoyé au processus.
L'image est résolue et vérifiée (comparaison de checksum), puis stockée localement, de façon accessible au processus docker.
Cette partie se produit seulement lors de la commande docker run : un conteneur Docker est construit avec l'image de l'étape 2 et 3 puis est exécuté sur l'hôte, indépendamment du système d'exploitation.
Il y a bien sûr moyen de personnaliser les paramètres d'une image, notamment avec des variables d'environnement (option -e), en personnalisant les « Port-Forwarding » (option -p) ou en personnalisant le réseau de l'image (option –network). Docker est donc très flexible et permissible au niveau de la configuration.

2.4.3 L'orchestration des conteneurs avec Docker

Docker permet de faciliter la coordination des comportements entre les conteneurs, et de les connecter entre eux pour créer des stacks d'applications. Pour simplifier le processus de développement et de test d'applications multi-conteneurs, Docker a créé Docker Compose.
Il s'agit d'un outil de ligne de commande, similaire au client Docker, utilisant un fichier de description spécifiquement formaté pour assembler les applications à partir de conteneurs multiples et de les exécuter sur un hôte unique.
Lorsqu'une application est prête à être déployée sur Docker, il est nécessaire de pouvoir approvisionner, configurer, étendre et surveiller les conteneurs sur l'architecture de microservice.
Pour y parvenir, on utilise des systèmes d'orchestration de conteneurs open source tels que Kubernetes, Mesos et Docker Swarm. Ces systèmes fournissent les outils nécessaires pour gérer les clusters de conteneurs. Ces solutions permettent notamment de répartir les ressources entre les conteneurs, d'ajouter ou de supprimer des conteneurs, de gérer les interactions entre les conteneurs, de surveiller leur statut, ou d'équilibrer la charge entre les microservices.

2.4.4 Dockerfile

Un Dockerfile est un fichier de configuration utilisé dans Docker pour créer une image Docker. Il contient un ensemble d'instructions qui spécifient les étapes nécessaires pour assembler une image Docker, à partir de laquelle des conteneurs Docker peuvent être lancés. Un Dockerfile est essentiellement un script textuel qui définit l'environnement et les dépendances nécessaires à l'exécution d'une application ou d'un service spécifique.

2.5 Apache Airflow

2.5.1 Définition

Apache Airflow est une plateforme open-source de gestion de workflows développée par la Apache Software Foundation. Elle permet la programmation, l'ordonnancement, et la surveillance de flux de travail complexes constitués de tâches interdépendantes.

2.5.2 Composant d'Apache Airflow

1. **Dag (Directed Acyclic Graph)** : Au cœur d'Airflow se trouve le concept de DAG, qui représente le workflow que vous souhaitez orchestrer. Un DAG est une collection ordonnée de tâches qui s'exécutent selon une logique définie. Ces tâches sont représentées en tant que nœuds dans le graphique dirigé, et les dépendances entre elles définissent l'ordre d'exécution.

2. **Opérateurs :** Les tâches dans un DAG sont définies par des opérateurs. Les opérateurs déterminent le type d'action à effectuer, comme exécuter une requête SQL, lancer un script Python, ou transférer des données entre systèmes. Airflow propose une gamme d'opérateurs prédéfinis, et les utilisateurs peuvent également créer leurs propres opérateurs personnalisés.
3. **Scheduler :** Airflow utilise un composant appelé Scheduler qui planifie l'exécution des tâches en fonction des dépendances définies dans le DAG. Le Scheduler s'assure que les tâches sont exécutées dans le bon ordre, en tenant compte des dépendances entre les différentes tâches.
4. **Meta-base de données :** Airflow utilise une base de données (comme SQLite, MySQL, ou PostgreSQL) pour stocker les métadonnées sur les DAG, les tâches, les statuts d'exécution, etc. Cette base de données permet de suivre l'état des workflows et de stocker l'historique des exécutions.
5. **Web Server :** Airflow fournit une interface web conviviale qui permet aux utilisateurs de visualiser, surveiller et administrer les DAG et les exécutions en cours.
6. **Executor :** L'Executor est responsable de l'exécution réelle des tâches. Airflow prend en charge plusieurs types d'executors, tels que SequentialExecutor, LocalExecutor, CeleryExecutor, etc., qui déterminent comment les tâches sont exécutées en parallèle.

Dans l'ensemble, Apache Airflow offre une solution flexible et évolutive pour automatiser et orchestrer des workflows complexes, tout en fournissant une interface utilisateur conviviale pour la gestion et la surveillance des processus.

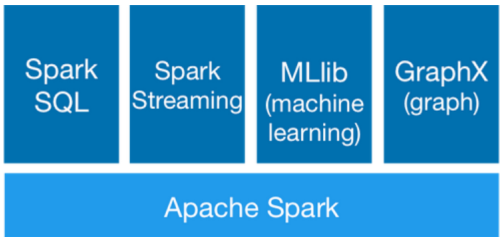
2.6 Streamlit

Streamlit est un framework open-source en Python qui permet de créer des applications web de manière simple et rapide. Streamlit simplifie le processus de création d'applications web interactives en permettant aux développeurs d'utiliser du code Python familier. Son approche déclarative et sa réactivité automatique font de lui un choix populaire pour la création rapide de prototypes, de tableaux de bord, et d'applications web simples.

2.7 Apache spark (Pyspark)

2.7.1 Définition

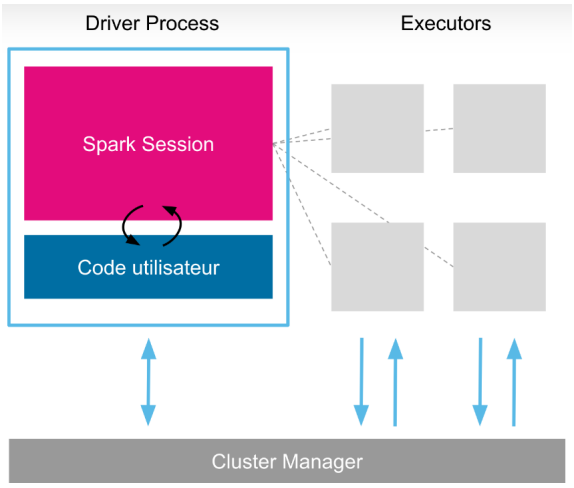
Spark (ou Apache Spark) est un framework open source de calcul distribué in-memory pour le traitement et l'analyse de données massives. Il s'agit d'un ensemble d'outils structurés selon une architecture définie. Ces composants font de Spark une plate-forme unificatrice riche en fonctionnalités : elle peut être utilisée pour de nombreuses tâches qui devaient auparavant être accomplies avec plusieurs frameworks différents.



Le groupe de machines que Spark utilisera pour exécuter des tâches sera géré par un gestionnaire de groupe comme le gestionnaire de groupe autonome de Spark, YARN, ou Mesos. Les demandes Spark sont ensuite soumises à ces gestionnaires de clusters qui accorderont des ressources à la demande afin que celle-ci puisse être effectuer.

2.7.2 Fonctionnement de Spark

Les applications Spark se composent d'un pilote (driver process) et de plusieurs exécuteurs (executor processes). Il peut être configuré pour être lui-même l'exécuteur (local mode) ou en utiliser autant que nécessaire pour traiter l'application, Spark prenant en charge la mise à l'échelle automatique par une configuration d'un nombre minimum et maximum d'exécuteurs.



Le driver (parfois appelé Spark Session) distribue et planifie les tâches entre les différents exécuteurs qui les exécutent et permettent un traitement réparti. Il est le responsable de l'exécution du code sur les différentes machines.
Lors de l'exécution de l'application, Spark crée des jobs, des stages et des tasks. Sans aller trop loin dans le détail, les jobs se composent de stages, et les stages

se composent de tâches (voir le schéma ci-dessous). Les stages sont généralement exécutés séquentiellement, tandis que les tâches peuvent être exécutées en parallèle dans le cadre d'un seul stage.

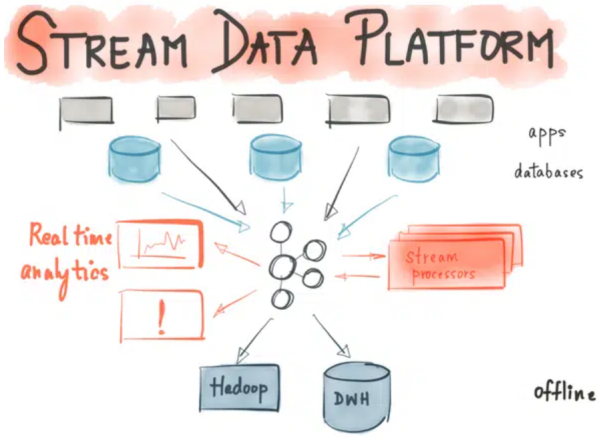
Le framework étant écrit en Scala, il s'agit du langage privilégié pour les nouvelles fonctionnalités et qui est le plus efficace à utiliser pour travailler avec Spark.

PySpark est l'implémentation de Spark pour Python contenant les différents composants de Spark.

2.8 Apache Kafka

2.8.1 Définition

En 2009, une équipe de LinkedIn a rencontré des difficultés d'intégration des données provenant de différents systèmes de l'entreprise lors d'un projet Big Data utilisant Hadoop. Pour résoudre ce problème, ils ont créé Apache Kafka, un système de messagerie distribuée en mode publish-subscribe, capable de gérer de gros volumes de flux de données provenant de divers systèmes. En 2011, Kafka est devenu une technologie open-source largement utilisée pour le traitement et la diffusion de messages en streaming. Ses principes de conception sont axés sur la scalabilité et la manipulation en temps réel des données et des journaux des systèmes complexes d'aujourd'hui. Kafka est désormais une plateforme centrale pour le stockage et l'échange de données en temps réel, et il est considéré comme une norme pour les pipelines de traitement de données.



2.8.2 Fonctionnement du système

Apache Kafka est un logiciel qui permet de définir des Topics (considérez un Topic comme une catégorie de donnée), d'ajouter des applications, de traiter et

de retraiter des enregistrements.

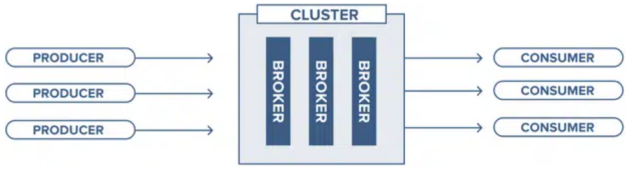
Les applications se connectent à ce système et transfèrent un message sur le Topic. Un message peut contenir n'importe quel type d'information, par exemple, des informations sur un événement qui s'est produit sur un site web, ou un événement qui est censé déclencher un événement. Une autre application peut se connecter au système et traiter ou retraiter les messages d'un topic . Les données envoyées sont conservées jusqu'à l'expiration d'une période de conservation déterminée (configurable).

Les quatre parties principales d'un système Kafka :

Broker : Traite toutes les demandes des clients (production, consommation et métadonnées) et conserve les données répliquées dans le cluster. Il peut y avoir un ou plusieurs brokers dans un cluster (tous les enregistrements Kafka sont organisés dans des topics).
Zookeeper : Conserve l'état du cluster (Brokers, Topics, Utilisateurs).
Producer : Envoie les enregistrements à un broker (écrit sur des topics).
Consumer : Consomme les messages en provenant du broker (extrait des enregistrements d'un topic).

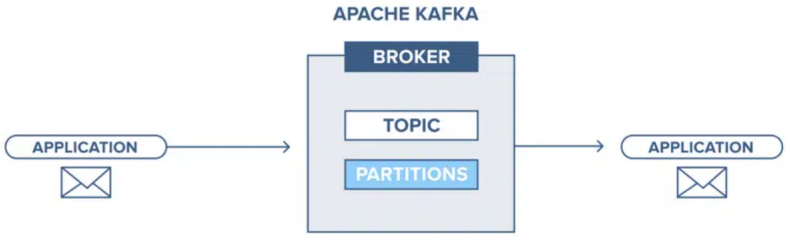
2.8.3 Broker et Topic Kafka

Broker Kafka Un cluster Kafka est constitué d'un ou de plusieurs serveurs (broker Kafka) fonctionnant sous Kafka.Il est possible de faire fonctionner un seul broker Kafka, mais cela ne donne pas tous les avantages que Kafka dans un cluster peut donner, par exemple, la réplication des données.La gestion des brokers du cluster est assurée par le Zookeeper et il peut y avoir plusieurs Zookeepers dans un cluster (de préférence de trois à cinq, en gardant un nombre impair).



Topic Kafka/Partitions Un Topic est une « catégorie » (similaire à une table dans une base de données sans les contraintes) dans lequel les messages sont stockés et publiés.Kafka conserve les enregistrements dans le log, ce qui rend les consommateurs responsables du suivi de la position dans le log, connu sous le nom de « l'offset ».Généralement, un consommateur avance l'offset de manière linéaire au fur et à mesure de la lecture des messages. Cependant, la position est en fait contrôlée par le consommateur, qui peut consommer les messages dans n'importe quel ordre. Par exemple, un consommateur peut revenir à un offset plus ancien lors du

retraitement des enregistrements.



2.9 ONNX

ONNX(Open Neural Network Exchange) est un format ouvert pour représenter à la fois les modèles d'apprentissage profond et les modèles traditionnels d'apprentissage automatique. Avec ONNX, les développeurs d'IA peuvent plus facilement déplacer les modèles entre les outils de pointe et choisir la combinaison qui leur convient le mieux.

2.10 Hadoop HDFS

2.10.1 Définition

Hadoop HDFS (Hadoop Distributed File System) est un système de fichiers distribué conçu pour stocker de grandes quantités de données sur des clusters de serveurs, offrant une tolérance aux pannes élevée et des performances optimisées pour le traitement parallèle de données. Il se compose de deux types de nœuds principaux : le NameNode et les DataNodes.

2.10.2 Fonctionnemnt de HDFS

HDFS comprend le Namenode qui est le cœur du système Hadoop HDFS. Il gère les métadonnées du système de fichiers, y compris les informations sur les fichiers, les répertoires et les emplacements des blocs de données. Le NameNode conserve ces métadonnées en mémoire, ce qui lui permet de coordonner les opérations d'accès aux données.

Ensuite nous avons les DataNodes qui sont les nœuds de stockage du cluster Hadoop HDFS. Chaque DataNode gère le stockage physique des blocs de données sur le disque local. Ils sont responsables de stocker les données, de les répliquer sur d'autres DataNodes et de rapporter régulièrement leur état au NameNode.

Lorsqu'un fichier est envoyé à Hadoop HDFS, il est divisé en blocs de taille fixe, généralement de 128 Mo. Le NameNode attribue des emplacements aux blocs et les DataNodes stockent les blocs de manière distribuée sur leurs disques locaux.

Hadoop HDFS réplique chaque bloc sur plusieurs DataNodes pour assurer la redondance et la tolérance aux pannes. Par défaut, chaque bloc est répliqué trois fois, avec une réplication initiale sur des nœuds différents pour améliorer la disponibilité des données.

Le NameNode communique avec les DataNodes pour gérer les opérations de stockage et de récupération des données. Les DataNodes rapportent régulièrement leur état au NameNode, y compris les blocs qu'ils stockent et leur disponibilité. Le NameNode utilise ces informations pour maintenir la cohérence des métadonnées et gérer les opérations d'accès aux données.

Les applications peuvent accéder aux données stockées dans Hadoop HDFS en utilisant des interfaces API spécifiques, telles que HDFS API ou des frameworks de traitement de données tels que Apache MapReduce ou Apache Spark. Ces interfaces permettent de lire, écrire et manipuler les données stockées dans le système de fichiers distribué.

2.11 Apache Zookeeper

2.11.1 Définition

Apache ZooKeeper est un service de coordination et de gestion distribuée open-source qui permet de coordonner et de synchroniser les nœuds d'un système distribué. Il fournit un ensemble de primitives pour la gestion des configurations, la synchronisation, la détection de pannes et la gestion des verrous dans les environnements distribués.

2.11.2 Fonctionnement de Zookeeper avec Kafka

Apache ZooKeeper joue un rôle essentiel dans le fonctionnement de Kafka. D'abord ZooKeeper est utilisé par Kafka pour stocker les métadonnées relatives aux brokers, aux topics et aux partitions. Ces métadonnées incluent des informations telles que les adresses IP des brokers, les partitions associées à chaque topic, les leaders des partitions, etc. ZooKeeper agit comme un registre centralisé qui permet à tous les brokers Kafka de connaître l'état actuel du cluster.

De plus Kafka utilise ZooKeeper pour gérer l'élection des leaders des partitions en cas de défaillance ou de reprise des brokers. Chaque partition a un leader qui est responsable de la réception et de la gestion des messages entrants. En cas de défaillance du leader, ZooKeeper est utilisé pour élire un nouveau leader parmi les brokers disponibles.

De même, ZooKeeper permet également aux brokers Kafka de s'enregistrer et de signaler leur disponibilité. Les brokers enregistrent leurs informations dans ZooKeeper lors de leur démarrage, et ZooKeeper est utilisé par les autres com-

posants Kafka pour découvrir les brokers actifs et établir des connexions avec eux.

En résumé, ZooKeeper joue un rôle crucial dans l'infrastructure Kafka en fournissant un système de coordination distribuée pour la gestion des métadonnées, l'élection des leaders de partition et le suivi de la configuration. Grâce à ZooKeeper, Kafka peut garantir la cohérence et la disponibilité du cluster, ainsi que la coordination entre les différents composants du système distribué.

2.12 Locust

Locust est une plateforme open-source de test de charge et de performance écrite en Python. Elle est utilisée pour émuler un grand nombre d'utilisateurs simultanés accédant à un système, service ou application, afin d'évaluer sa stabilité, sa robustesse, et sa capacité à gérer des charges importantes

2.13 Construction de l'image

Docker a révolutionné la manière dont nous déployons et gérons nos applications et nos outils d'analyse de données. Il permet une mise à l'échelle sans effort et la possibilité de personnaliser facilement les services pour répondre à des besoins spécifiques.

Dans notre projet nous avons déployé Apache Airflow, Apache Kafka, Apache Spark,Hadoop HDFS à l'aide d'un fichier docker-compose.yaml et tirer parti des volumes pour gérer les dépendances de données entre les services.

2.14 Métrique de Performance

L'évaluation des modèles d'analyse de sentiments repose sur diverses métriques permettant de mesurer leur performance. Dans notre contexte, nous nous concentrons principalement sur l'accuracy, une mesure classique et intuitive de la précision du modèle.

Accuracy (Précision Globale)

L'accuracy, ou précision globale, est une métrique fondamentale pour évaluer la performance d'un modèle d'analyse de sentiments. Elle mesure la proportion de prédictions correctes parmi l'ensemble total des prédictions. Mathématiquement, l'accuracy se définit comme :

$$\text{Accuracy} = \frac{\text{Nombre de Prédictions Correctes}}{\text{Nombre Total d'Échantillons}}$$

L'accuracy varie de 0 à 1, où 1 indique une performance parfaite (toutes les prédictions sont correctes) et 0 indique une performance nulle (aucune prédiction correcte). Cette métrique est particulièrement pertinente dans le contexte de l'analyse de sentiments, où l'objectif est de déterminer avec précision le sentiment associé à un texte donné.

Il est important de noter que bien que l'accuracy soit une métrique informative, elle peut ne pas être suffisante dans certains cas, notamment lorsque les classes ne sont pas équilibrées. Dans de telles situations, d'autres métriques comme la précision, le rappel et le F-score peuvent fournir une vision plus détaillée de la performance du modèle.

Pour notre analyse de sentiments, nous surveillerons principalement l'accuracy comme mesure principale de la qualité de nos prédictions.

2.15 Prétraitement des données

Le prétraitement des commentaires recueillis via l'API YouTube est essentiel pour garantir la qualité des données et faciliter une analyse efficace. Nous appliquons les étapes suivantes :

Tokenisation : Division du texte en unités (tokens) comme mots, ponctuations et émojis pour une représentation structurée. Cela est pris en charge âr le Tokeniseur de DistilBert.

Suppression des Stopwords : Élimination des mots courants sans grande valeur sémantique pour réduire le bruit.

Correction d'Orthographe (en option) : Correction d'orthographe pour améliorer la qualité du texte.

Ces étapes de prétraitement assurent que les commentaires sont prêts pour une analyse de sentiment précise, réduisant le bruit et améliorant la cohérence des données. Ce processus facilite la création d'un modèle d'analyse de sentiments robuste.

2.16 Large Language Models (LLM)

Les Modèles de Langage de Grande Taille (LLM) représentent une catégorie de modèles d'apprentissage automatique spécialement conçus pour la compréhension avancée du langage naturel. Cette section explore ces modèles en détail, en se concentrant sur leur définition, leur fonctionnement, le fine-tuning, et en réalisant une comparaison avec BERT, DistilBERT et RoBERTa.

2.16.1 Définition des LLM

Les Modèles de Langage de Grande Taille sont des architectures neuronales profondes entraînées sur de vastes corpus de texte. Leur objectif principal est de capturer des représentations sémantiques riches et contextualisées du langage, ce qui les rend particulièrement adaptés à une variété de tâches liées au traitement du langage naturel (NLP).

2.16.2 Fonctionnement des LLM

Ces modèles fonctionnent en utilisant une architecture de réseau de neurones, généralement basée sur des transformers. Ils exploitent des mécanismes d'attention pour comprendre la relation contextuelle entre les mots dans une séquence, permettant ainsi une meilleure compréhension du sens global du texte.

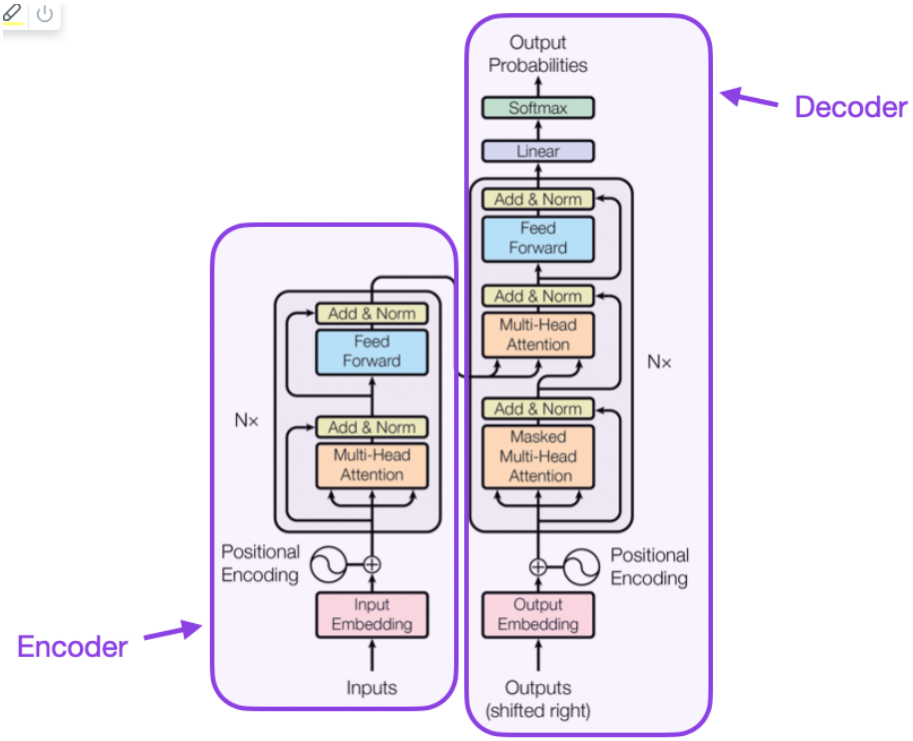


FIGURE 1 – Bert : Model Testing

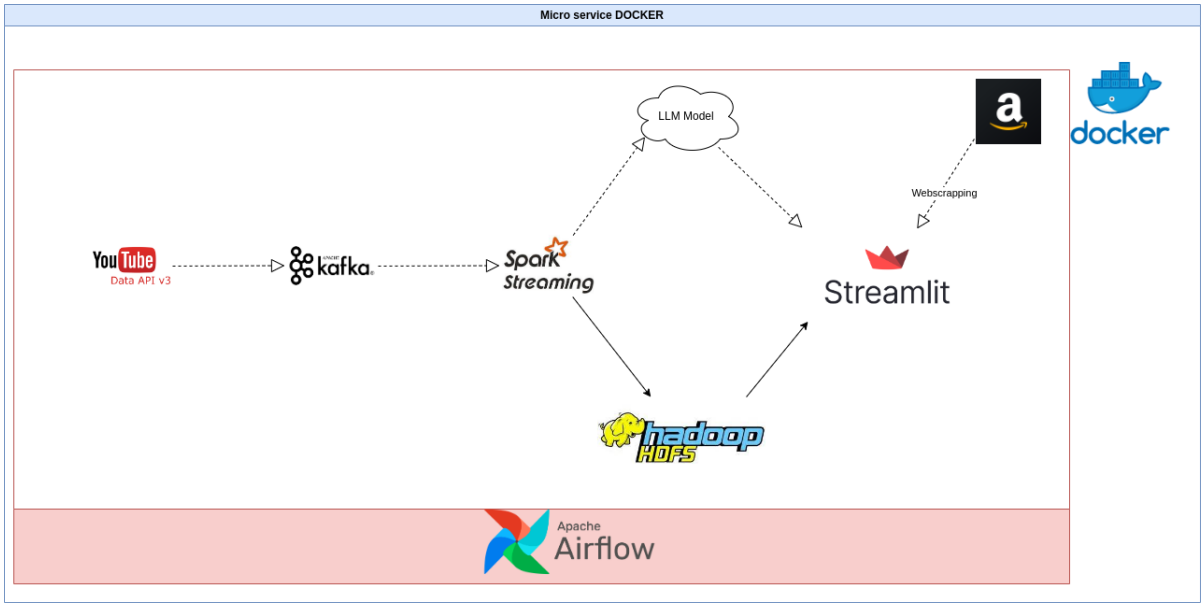
2.16.3 Fine-tuning des LLM

Le fine-tuning des LLM consiste à ajuster un modèle pré-entraîné sur des tâches spécifiques, adaptant ainsi ses capacités à des domaines ou des problématiques particuliers. Cette étape est cruciale pour maximiser les performances du modèle dans des contextes spécifiques.

Dans notre cas, le fine-tuning consistait à entrainer le modèle sur le Dataset Amazon-review afin de le familiariser avec les avis tout en prédisant leur polarité.

2.17 Architecture technique / Workflow de notre système

Les données proviennent de l'API de **youtube**. Il s'agit des commentaires des vidéos rélatives à un produit. La selection du produit a eu lieu sur l'interface de l'application **Streamlit**, obtenu à travers un **webScraping** dynamique d'**amazon**. Ces données passent par **kafka**, ensuite passées à **Spark** et sont stockées dans le système de fichiers **Hadoop HDFS**. Elles seront traitées par spark avant d'être passé au modèle pour l'analyse de sentiment. Les résultats de cette analyse seront ensuite passés à Streamlit pour la visualisation.



2.18 Conclusion

En conclusion, ce chapitre a examiné les technologies et approches clés utilisées dans notre projet de suivie de la réputation des produits. Nous avons commencé par décrire la source de nos données et comment les récupérées. Ensuite, nous avons exploré les concepts du Big Data ainsi que les métriques de performance utilisées. Nous avons également abordé l'analyse des sentiments avec les LLMs, une technique importante pour comprendre les tendances et les sentiments du public concernant un produit. Nous avons expliqué en détail le fonctionnement de chaque outil et des LLMs et leurs rôles dans notre approche. Enfin, nous avons décrit l'architecture de notre système, soulignant les composants clés et leur interaction pour la réalisation de notre projet. Cette architecture offre une base solide pour notre projet et constitue une étape importante vers le développement d'un système de surveillance d'un produit.

Dans le prochain chapitre, nous aborderons les résultats et les discussions de notre projet. Cette section sera dédiée à l'analyse des performances de notre modèle et la visualisation finale

3 Chapitre III : Résultats et discussions

3.1 Introduction

Ce chapitre présente les résultats et des discussions issus de notre projet de Monitoring de la reputation des produits. Après avoir exploré les technologies et

les approches clés dans les chapitres précédents, nous sommes maintenant prêts à évaluer les performances de notre modèle et à analyser les implications de nos résultats. On pourra examiner en détail les performances de notre modèle de sentiment analysis. Nous présenterons les résultats obtenus lors de nos expérimentations et discuterons des mesures de performance utilisées pour évaluer l'efficacité de notre approche. En plus des performances, nous discuterons des principales constatations et observations découlant de nos résultats.

3.2 Analyse Descriptive Exploratoire

Dans cette section, nous effectuons une analyse descriptive exploratoire de notre ensemble de données d'entraînement, qui est basé sur les critiques d'Amazon. L'objectif est de comprendre la distribution des classes et d'obtenir des insights préliminaires.

Distribution des Classes

La distribution des classes dans l'ensemble de données est un élément clé à examiner. Pour notre ensemble de données, les classes 0 et 1 correspondent à certaines caractéristiques. Voici la distribution des classes :

Classe	Nombre d'échantillons
0	5098
1	4902

TABLE 1 – Distribution des Classes dans l'Ensemble de Données d'Entraînement

La Table 1 montre la répartition équilibrée entre les classes 0 et 1, ce qui indique que notre ensemble de données est équilibré.

3.3 Comparaison des modèles de langages : BERT, DistilBERT et RoBERTa

Une comparaison entre les modèles BERT, DistilBERT et RoBERTa permet de mettre en lumière leurs différences essentielles. BERT excelle dans la compréhension contextuelle bidirectionnelle, c'est un modèle de 110 Millions de paramètres.

DistilBERT offre une version plus légère (66 Millions de paramètres)tout en conservant des performances considérables.

RoBERTa introduit des modifications spécifiques pour améliorer la formation et la performance sur certaines tâches, il a plus de paramètres que les deux précédent et donc peut être plus performant dans certaines taches en terme d'accuracy.

Les figures suivantes permettent de les comparer en fournissant le nombre de requettes par seconde et le temps de reponse des 3 modeles après le fine-tuning.

Pour le modèle Bert

FIGURE 2 – Bert : Model Testing

Pour le modèle RoBerta

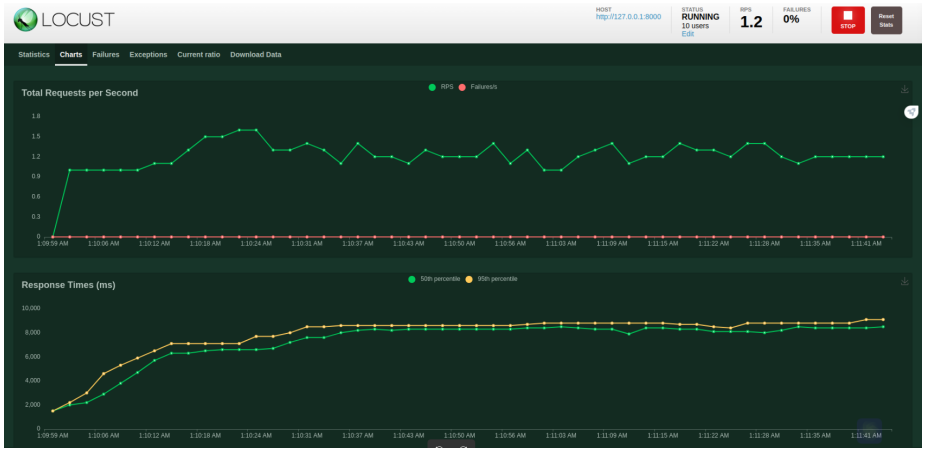


FIGURE 3 – RoBerta : Model Testing

Pour le modèle DistilBert

FIGURE 4 – DistilBert : Model Testing

De ces graphiques, on rétient que DistilBert est le model qui a le plus grand nombre de requettes par seconde tout en ayant le temps de réponse minimum des 3 modèles. D'autres part, nous avons obtenu les accuracy de la table 2 en fonction des modèles :

Modèle	**Accuracy (%)**
BERT	98
RoBERTa	98
DistilBERT	97

TABLE 2 – Résultats d'Accuracy avec Différents Modèles

il est donc évident que c'est celui-ci que nous avons déployé pour nos taches de sentiment analysis.

Cette comparaison souligne l'importance de choisir le modèle le mieux adapté à la tâche spécifique, en tenant compte des compromis entre la complexité du modèle, les ressources disponibles et le temps d'inférence.

3.4 Visualisation sur Streamlit

Une visualisation des résultats sur l'application **Streamlit** donne ce qui suit.

Nous pouvons voir sur l'application un **diagramme en donut** qui montre la répartition des sentiments : en vers le pourcentage des sentiments positifs et en rouge celui des négatifs. On peut également voir la repartition des sentiments en fonction du pays.

De même, on peut voir le sentiment dominant dans certains pays sur la **carte choroplèthe binaire**. Et enfin, un **WordCloud** pour montrer les expressions qui reviennent le plus souvent dans les commentaires.

A gauche dans l'application se trouve les **Filtres**. Veuillez remarquer le choix d'un produit est d'abord précédé par le choix de sa catégorie et de sous catégorie, comme présenté sur le site d'Amazon. On peut egalement faire une recherche du produit, de la catégorie ou de la sous catégorie.

3.5 Discussion

L'analyse approfondie des performances des modèles BERT, RoBERTa, et DistilBERT a apporté des éclaircissements significatifs qui guident notre choix pour le déploiement dans les tâches de sentiment analysis. Les graphiques mettent en évidence que DistilBERT excelle avec le plus grand nombre de requêtes par seconde, tout en maintenant un temps de réponse minimal par rapport aux autres modèles évalués.

En détaillant les résultats d'accuracy dans la Table 2, nous notons que BERT et RoBERTa ont atteint une excellente performance avec une accuracy de 98%, tandis que DistilBERT affiche une légère diminution à 97%. Cependant, l'évaluation ne peut pas se réduire à cette seule métrique. Des considérations pratiques, telles que le temps d'inférence et la consommation de ressources, jouent un rôle crucial dans la sélection du modèle optimal pour le déploiement en production.

La décision de déployer DistilBERT repose sur un équilibre stratégique entre des performances acceptables (97% d'accuracy) et des avantages pratiques. DistilBERT offre non seulement des résultats compétitifs, mais il excelle également en termes de rapidité d'inférence et d'efficacité en matière de ressources. Cette caractéristique devient particulièrement pertinente dans des environnements où la vitesse de traitement et l'économie de ressources sont des considérations cruciales.

Il est crucial de souligner l'importance de choisir le modèle le mieux adapté à la tâche spécifique. La complexité du modèle, les ressources disponibles, et le temps d'inférence sont des facteurs incontournables qui influent sur le succès du déploiement dans des applications du monde réel. Cette démarche réfléchie garantit une mise en œuvre efficace, répondant aux besoins spécifiques de nos tâches de sentiment analysis tout en optimisant les performances opérationnelles.

3.6 Conclusion

La section résultats et discussions a fourni un aperçu approfondi des performances des modèles BERT, RoBERTa et DistilBERT dans le contexte de notre projet de Monitoring de la réputation des produits. Les analyses détaillées, notamment la comparaison des performances, la distribution des classes, et les

visualisations sur Streamlit, ont jeté la lumière sur les forces et les compromis de chaque modèle.

La décision de déployer DistilBERT repose sur une évaluation minutieuse, prenant en compte les performances globales, la vitesse d'inférence, et l'utilisation efficace des ressources. Bien que DistilBERT ait affiché une légère baisse d'accuracy par rapport à BERT et RoBERTa, sa rapidité d'inférence et son efficacité en termes de ressources en font un choix judicieux pour nos tâches de sentiment analysis.

Cette section a également présenté des visualisations concrètes via l'application Streamlit, offrant une interface utilisateur intuitive pour l'analyse des sentiments en temps réel. L'ensemble des résultats et discussions fournit des bases solides pour la suite du déploiement et de l'utilisation opérationnelle de notre modèle dans le contexte du Monitoring de la réputation des produits.

4 Conclusion générale et perspectives

Le projet "A big data architecture for product reputation monitoring" propose une approche novatrice pour le suivi de la réputation des produits. L'utilisation de plusieurs outils tels que **Apache Airflow**, **Youtube API**, **Apache Kafka**, **Apache Spark**, **ONNX**, **Streamlit**, et **Hadoop HDFS** et le **webScraping** du site **Amazon** permet de collecter, traiter et analyser de grandes quantités de données sur les produits et feedbacks des clients. La motivation première de la mise en place d'une architecture big data est de fournir une plateforme robuste et évolutive capable de traiter des données provenant de diverses sources. La diversité des sources, notamment le web scraping pour les informations sur les produits Amazon, l'extraction de commentaires vidéo sur YouTube via l'API YouTube, et l'obtention d'avis Amazon à partir de Kaggle, témoigne de la polyvalence de l'architecture mise en place.

L'orchestration de cette architecture à l'aide d'outils tels que **Apache Airflow** a permis une gestion efficace des flux de travail, assurant une collecte, un traitement et une analyse des données cohérents et planifiés. L'utilisation de technologies Big Data comme **Apache Spark** et **Apache Kafka** a facilité le traitement et la gestion de grandes quantités de données de manière distribuée et évolutive.

La partie cruciale de l'analyse a été consacrée à la mise en place de modèles de langage (**LLMs**), notamment **BERT** et **DistilBERT**, pour réaliser une analyse de sentiment précise. Cette étape a été essentielle pour comprendre la perception des clients à l'égard des produits, fournissant ainsi des informations précieuses pour le suivi de la réputation des produits.

Enfin, la création d'une application web de visualisation avec **Streamlit** a permis de mettre en valeur les résultats de manière conviviale et accessible. Cette interface utilisateur simplifiée offre la possibilité d'explorer les résultats de l'analyse de sentiment de manière interactive.

Perspectives futures :

- Exploration de l'intégration de modèles d'apprentissage en continu pour une analyse en temps réel des commentaires des clients.
- Extension de la plateforme pour inclure des fonctionnalités supplémentaires telles que l'analyse d'images pour une compréhension plus approfondie des produits.
- Amélioration continue des modèles de langage pour garantir des performances optimales sur diverses catégories de produits et langues.
- Intégration d'autres sources de donnée comme **Twitter API** pour avoir d'autre commentaires sur les produits

En conclusion, ce projet démontre l'efficacité d'une architecture big data bien orchestrée dans le domaine du suivi de la réputation des produits. Les résultats obtenus et les perspectives futures illustrent le potentiel continu d'innovation et d'amélioration dans ce domaine. La combinaison de technologies Big Data et de modèles de langage ouvre la voie à une surveillance proactive et informée de la réputation des produits dans un monde numérique en constante évolution.

Références

[1] https ://towardsdatascience.com/bert-roberta-distilbert-xlnet-which-one-to-use-3d5ab82ba5f8 **[Dernière consultation : 30/01/2024]**

[2] https ://spark.apache.org/ **[Dernière consultation : 26/11/2023]**

[3] https ://microsoft.github.io/SynapseML/docs/features/onnx/about/ **[Dernière consultation : 12/11/2023]**

[4] https ://kafka.apache.org/ **[Dernière consultation : 24/12/2023]**

[5] https ://www.docker.com/ **[Dernière consultation : 08/01/2024]**

[6] https ://hadoop.apache.org/docs/r1.2.1/hdfs_design.html **[Dernière consultation : 16/11/2023]**

[7] https ://hub.docker.com/r/confluentinc/cp-zookeeper **[Dernière consultation : 10/10/2023]**

[8] https ://hub.docker.com/r/confluentinc/cp-kafka **[Dernière consultation : 10/10/2023]**

[9] https ://hub.docker.com/r/bde2020/hadoop-namenode **[Dernière consultation : 05/11//2023]**

[10] https ://hub.docker.com/r/bde2020/hadoop-datanode **[Dernière consultation : 05/11/2023]**

[11] wikipedia **[Dernière consultation : 29/01/2024]**

[12] https ://airflow.apache.org/ **[Dernière consultation : 15/01/2024]**

[13] https ://streamlit.io/ **[Dernière consultation : 28/01/2024]**