
Prédiction des Sentiments en Temps Réel

Intégration de Deep Learning, Big Data, JavaScript et Ingénierie des Connaissances

Résumé

Dans un monde où les technologies numériques et l'intelligence artificielle influencent profondément la prise de décisions stratégiques, ce projet s'inscrit dans cette dynamique en alliant des domaines comme l'ingénierie des connaissances, le deep learning, le traitement des big data, et les technologies JavaScript. L'objectif principal est de concevoir et développer une application innovante capable d'analyser les sentiments exprimés sur les réseaux sociaux pour prédire les tendances du marché.

Ce projet explore des méthodologies avancées pour extraire, modéliser, et traiter les connaissances issues des données massives, en utilisant des techniques d'apprentissage profond (deep learning) pour l'analyse de sentiment et des technologies front-end pour une visualisation intuitive des résultats. En plus de ses dimensions techniques, ce projet permet d'acquérir des compétences pratiques en résolution de problèmes complexes tout en répondant aux enjeux réels de l'analyse des données dans le contexte du marché.

Les résultats attendus incluent la création d'une ontologie des tendances de marché, le développement d'un module performant de deep learning, et la conception d'une interface utilisateur interactive. Ce rapport détaille les étapes du projet, les approches adoptées, et présente les contributions théoriques et pratiques en matière d'analyse prédictive basée sur les données massives.

List of Figures

1.1	Flask	4
1.2	React	4
1.3	MongoDb	5
1.4	Apache Spark	5
1.5	Apache PySpark	5
1.6	Apache kafka	6
1.7	Apache Jena	6
1.8	Apache Jena	7
2.1	Bibliothèques	12
2.2	Session Spark	14
2.3	Accès à la Configuration Spark	14
2.4	Affichage de la Configuration Spark	15
2.5	Chargement des données	15
2.6	Visualisation des données(code)	15
2.7	Visualisation des données(Affichage)	16
2.8	Supression des valeurs nulles	17
2.9	Fusion des jeux de données	17
2.10	Exécution du fusion des jeux de données	18
2.11	Supression des doublons	19
2.12	Répartition des sentiments	20
2.13	Encodage des sentiments	21
2.14	Conversion des sentiments	22
2.15	Equilibrage des classes	23
2.16	Visualisation 1: après équilibrage	23
2.17	Visualisation 2: après équilibrage	24

2.18	Prétraitement des textes	24
2.19	Prétraitement des textes	25
2.20	Suppression des Stopwords	25
2.21	Suppression des Stopwords	26
2.22	Word Cloud for Sentiment Positive	27
2.23	Word Cloud for Sentiment Neutral	27
2.24	Word Cloud for Sentiment Negative	27
2.25	Ensembles d'entraînement, validation et de test	28
2.26	Enregistrement des données sous format csv	29
3.1	Architecture GRU	41
3.2	Etapes LSTM	44
3.3	Architecture LSTM	45
3.4	Architecture - Partie 1	50
3.5	Architecture - Partie 2	51
3.6	Architecture - Partie 3	51
3.7	Producer: Partie1	53
3.8	Producer: Partie2	54
3.9	Producer: Partie3	54
3.10	Consumer	55
4.1	Architecture Kafka	59
4.2	Démarrage de Zookeeper	60
4.3	Démarrage de Kafka	61
4.4	Lancer le producer	62
4.5	Lancer le consumer	63
5.1	Création des classes de l'ontologie	66
5.2	Création des sous classes de l'ontologie	67
5.3	Création des propriétés de l'ontologie	69
5.4	Graphe de l'ontologie	70
5.5	Intégration de GPT-2	71
5.6	Extraction des entités	72
5.7	Mise à jour des entités	72

5.8	Sauvegarder l'ontologie	73
5.9	mise à jour automatique (partie 1)	74
5.10	mise à jour automatique (partie 2)	74
5.11	Planification et Exécution de la tâche	75
5.12	Résultat	75
5.13	Les règles d'inférence	76
5.14	Jena application	77
5.15	les Requêtes d'inférence	77
5.16	implémentation Java	78
6.1	Graphes de Visualisation (partie 1)	80
6.2	Graphes de Visualisation (partie 2)	81
6.3	Population et Commentaires par Pays	83
6.4	Carte Interactive des Tweets par Pays	84
6.5	Relation entre l'âge et la longueur des tweets & Distribution des âges	85
6.6	Répartition des sentiments et évolution du nombre de tweets au fil du temps	86
6.7	Répartition des genres parmi les auteurs des tweets	87
6.8	Aperçu du tableau des tweets en temps réel (partie 1)	87
6.9	Aperçu du tableau des tweets en temps réel (partie 2)	88

Contents

Résumé	i
Remerciement	ii
Liste des Figures	v
Table des Matières	x
Introduction Générale	1
1 Contexte Général du projet	2
Introduction	2
1.1 Description	2
1.2 Fonctionnalités	3
1.3 Outils	3
1.3.1 Flask	3
1.3.2 React	4
1.3.3 MongoDB	4
1.3.4 Spark	5
1.3.5 PySpark	5
1.3.6 Kafka	6
1.3.7 Protégé	6
1.3.8 Apache Jena	7
1.4 Modèles	7
1.4.1 CNN	7
1.4.2 RNN	7
1.4.3 LSTM	8

1.4.4 GAN (Generative Adversarial Networks)	8
2 Préparation des données	10
2.1 Description	10
2.1.1 Variables Textuelles	11
2.1.2 Variables Démographiques	11
2.1.3 Variables Temporelles	11
2.1.4 Variable Cible	11
2.2 Préparation des données	12
2.2.1 Bibliothèques	12
2.2.1.1 PySpark Libraries	12
2.2.1.2 Python Libraries	13
2.2.2 Configuration Spark	13
2.2.2.1 Création et Initialisation d'une Session Spark	13
2.2.2.2 Accès et Affichage de la Configuration Spark	14
2.2.3 Chargement et visualisation des données	15
2.2.3.1 Chargement des données	15
2.2.3.2 Visualisation des données	15
2.2.4 Prétraitement	16
2.2.4.1 Suppression des valeurs nulles et comptage des lignes	16
2.2.4.2 Fusion des jeux de données d'entraînement et de test	17
2.2.4.3 Suppression des doublons et analyse descriptive des données	18
2.2.4.4 Visualisation de la répartition des sentiments	19
2.2.4.5 Encodage des sentiments	20
2.2.5 Conversion et Visualisation des Sentiments	21
2.2.5.1 Équilibrage des Classes par Downsampling	22
2.2.5.2 Visualisation après équilibrage	23
2.2.5.3 Prétraitement des textes	24
2.2.5.4 Prétraitement des textes	24
2.2.5.5 Suppression des Stopwords	25
2.2.5.6 Génération de Nuages de Mots Basés sur les Sentiments	26
2.2.5.7 Division des données en ensembles de formation, de test et de validation	27

2.2.5.8	Conversion des colonnes de type ARRAY en STRING et sauvegarde des ensembles de données	28
3	Réalisation des modèles	30
3.1	CNN (Convolutional Neural Networks)	30
3.1.1	Introduction aux CNN	30
3.1.2	CNN 1D (Convolutional Neural Network 1D)	31
3.1.2.1	Architecture du modèle CNN 1D	31
3.1.2.2	Évaluation du modèle	33
3.1.3	CNN 2D (Deux Dimensions)	33
3.1.3.1	1. Préparation des données	33
3.1.3.2	2. Architecture du modèle CNN 2D	34
3.1.3.3	3. Compilation et entraînement du modèle	34
3.1.3.4	4. Résultats obtenus	35
3.1.3.5	5. Conclusion	35
3.1.4	CNN 3D (Trois dimensions)	35
3.1.4.1	Architecture d'un CNN 3D	35
3.1.4.2	Traitement des données	36
3.1.4.3	Évaluation d'un CNN 3D	36
3.1.4.4	Applications des CNN 3D	37
3.1.4.5	Avantages et défis des CNN 3D	37
3.1.4.6	Résultats de traitement	38
3.2	RNN	38
3.2.1	Workflow général	39
3.2.2	Préparation des données	39
3.2.3	Architecture du Modèle	40
3.2.4	Optimisation et Fonction de Coût	41
3.2.5	Entraînement du Modèle	41
3.2.6	Évaluation du Modèle	42
3.2.7	Résultats de l'Entraînement et de l'Évaluation	42
3.3	LSTM	43
3.3.1	Architecture du Modèle	44
3.3.2	Paramètres et Configuration du Modèle	45

3.3.3	Configuration des Paramètres d’Entrainement	46
3.3.3.1	Entrainement du Modèle	47
3.3.3.2	Evaluation du Modèle	47
3.4	GAN	48
3.4.1	Modéle GAN avec GPT2	48
3.4.1.1	Architecture de GPT-2	48
3.4.1.2	Fonctionnement de GPT-2	48
3.4.1.3	À quoi sert GPT-2 ?	49
3.4.1.4	Utilisation de GPT-2 dans le projet	49
3.4.2	Consumer et Producer avec KafkaPy	51
3.4.2.1	KafkaPy	51
3.4.2.2	À quoi sert KafkaPy ?	52
3.4.2.3	Fonctionnalités principales de KafkaPy :	52
3.4.2.4	Fonctionnalités de KafkaPy dans ce projet:	53
4	Intégration de kafka pour la Prédiction en Temps Réel	57
4.1	Objectifs	57
4.2	Prédiction des Sentiments en Temps Réel avec Kafka et RNN	58
4.2.1	Workflow des prédictions	58
4.2.1.1	Collecte des Données avec Kafka	58
4.2.1.2	Prétraitement des Textes	58
4.2.1.3	Chargement des Ressources	58
4.2.1.4	Prédiction des Sentiments	59
4.2.1.5	Affichage des Résultats	59
4.2.1.6	Étape 6 : Envoi des Résultats au Kafka Producer	59
4.2.2	Configuration de l’Infrastructure Kafka et Zookeeper pour l’Analyse de Textes	59
4.2.2.1	Démarrer Zookepper	60
4.2.2.2	Démarrer kafka	60
4.2.2.3	Créer un Topic kafka	61
4.2.2.4	Lancer le producer	61
4.2.2.5	Lancer le consumer	62

5 Ingénierie des connaissances	64
5.1 Conception d'une ontologie des tendances de marché et des sentiments	65
5.1.1 Création des Classes	65
5.1.2 Création des sous Classes	66
5.1.3 Création des Propriétés	67
5.1.4 Propriétés d'Objet	68
5.1.5 Propriétés de Données	68
5.1.5.1 Ajout des Individus	68
5.1.6 Validation avec le Raisonneur Hermit	68
5.1.7 Visualisation du Graphe	69
5.1.8 Exportation de l'Ontologie	70
5.2 Extraction et Enrichissement des Concepts avec des Données en Temps Réel	70
5.2.1 Intégration de GPT-2 pour Génération de Textes	71
5.2.2 Analyse et Extraction d'Entités et de Relations	71
5.2.3 Mise à jour de l'Ontologie avec les Relations	72
5.2.4 Sauvegarder l'ontologie	73
5.2.5 Fonction principale pour la mise à jour automatique	73
5.2.6 Planifier et Exécuter la tâche avec Schedule	74
5.2.7 Résultats de l'Extraction	75
5.3 Les règles d'inférence et création des requêtes d'inférence	75
5.3.1 Règles d'inférence	75
5.3.2 Requêtes d'inférence	76
6 Réalisation du Projet	80
6.1 Tableau de Bord Statique	80
6.2 Tableau de Bord en Temps Réel	85
Conclusion Générale	90

Introduction Générale

Dans un contexte où les technologies numériques évoluent rapidement, les données massives et aussi l'intelligence artificielle jouent un rôle clé dans la prise de décisions stratégiques. Le présent projet s'inscrit dans cette dynamique en intégrant des domaines variés tels que l'ingénierie des connaissances, le deep learning, le traitement des big data et les technologies JavaScript.

L'objectif principal de ce projet est de concevoir et développer une application innovante capable d'analyser les sentiments exprimés sur les réseaux sociaux afin de prédire les tendances de marché. Ce défi multidisciplinaire permet d'explorer des méthodologies avancées pour extraire, modéliser et traiter des connaissances tout en exploitant des techniques d'apprentissage profond et des technologies front-end pour une visualisation intuitive des résultats.

Au-delà des aspects techniques, ce projet offre une opportunité unique de consolider des compétences pratiques en résolution de problèmes complexes tout en répondant à des enjeux réels du domaine de l'analyse des données. Les résultats attendus incluent la création d'une ontologie des tendances de marché, le développement d'un module de deep learning performant, et la conception d'une interface utilisateur interactive.

Ainsi, ce rapport détaille les différentes étapes et approches adoptées pour atteindre ces objectifs, mettant en lumière les contributions théoriques et pratiques du projet à l'analyse prédictive basée sur les données massives.

Chapitre 1

Contexte Général du projet

Introduction

Dans un monde en constante évolution technologique, l'analyse des données massives est devenue un levier stratégique pour les entreprises cherchant à anticiper les comportements des consommateurs et à s'adapter aux tendances du marché.

Ce projet vise à concevoir une application permettant d'exploiter les données issues des réseaux sociaux pour analyser les sentiments exprimés et en déduire des tendances de marché émergentes. Cette initiative multidisciplinaire combine les dernières avancées en ingénierie des connaissances, apprentissage profond, big data et technologies JavaScript pour la visualisation.

1.1 Description

Le projet consiste à développer une application complète intégrant plusieurs modules complémentaires. Ces modules incluent un système de gestion des connaissances, un module d'apprentissage profond pour l'analyse des sentiments, un traitement des données massives via des outils big data comme Spark, Kafka, et une interface utilisateur interactive développée en React.

L'application repose sur une architecture qui permet de collecter, traiter et analyser des données textuelles provenant de diverses plateformes de réseaux sociaux. Les résultats de ces analyses sont ensuite visualisés de manière intuitive grâce à un tableau de bord interactif. L'approche repose sur l'utilisation de techniques avancées telles que l'extraction des connaissances, la création d'ontologies, et l'entraînement de modèles prédictifs pour identifier les tendances potentielles.

1.2 Fonctionnalités

L'application proposée est dotée des fonctionnalités suivantes :

- **Collecte de données en temps réel** : Extraction de données textuelles depuis des plateformes comme Twitter, Facebook ou Instagram à l'aide des API appropriées. Dans cette partie nous avons utilisés deux approches différentes: la première c'est l'utilisation d'une base de données avec des tweets, la deuxième approche c'est l'utilisation de GAN pour la génération des tweets.
- **Analyse des sentiments** : Utilisation de modèles de deep learning pour classifier les publications en fonction du sentiment exprimé (positif, négatif ou neutre).
- **Tableau de bord interactif** : Visualisation intuitive des résultats d'analyse avec des graphiques, des cartes géographiques, et des indicateurs clés de performance dans un tableau de bord interactif.
- **Adaptabilité et évolutivité** : Intégration de nouvelles données et mise à jour continue des modèles et connaissances pour rester en phase avec les évolutions du marché.

Ces fonctionnalités visent à fournir une solution complète et pratique, combinant analyse prédictive et visualisation interactive pour répondre aux besoins des décideurs dans des environnements dynamiques.

1.3 Outils

1.3.1 Flask

Flask est un micro-framework web écrit en Python, connu pour sa simplicité et sa flexibilité. Contrairement à d'autres frameworks comme Django, Flask adopte une approche minimaliste, permettant aux développeurs de choisir les composants nécessaires selon les besoins du projet. Il repose sur WSGI (Web Server Gateway Interface) via Werkzeug, garantissant une gestion efficace des requêtes, et Jinja2, un moteur de templates puissant pour générer des pages HTML dynamiques. Grâce à sa documentation bien structurée et sa communauté active, Flask est largement utilisé pour développer des applications légères, des API RESTful, et des prototypes rapides.



Figure 1.1: Flask

1.3.2 React

React est une bibliothèque JavaScript open-source développée par Facebook pour construire des interfaces utilisateur interactives et dynamiques. Elle repose sur une architecture déclarative basée sur les composants, permettant de découper une application en petites parties réutilisables. React utilise un DOM virtuel (Virtual DOM) qui améliore considérablement les performances en minimisant les mises à jour du DOM réel. Parmi ses fonctionnalités clés, on retrouve la gestion des états à l'aide de hooks, une compatibilité avec des outils comme Redux pour la gestion globale des états, et la possibilité d'intégration avec d'autres frameworks ou bibliothèques. React est particulièrement adapté pour les applications monopage (SPA) nécessitant une expérience utilisateur fluide et réactive.

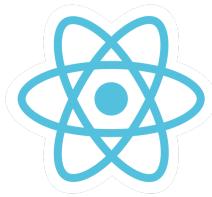


Figure 1.2: React

1.3.3 MongoDB

MongoDB est une base de données NoSQL orientée documents, conçue pour gérer des volumes de données importants et des structures non relationnelles. Contrairement aux bases de données traditionnelles, MongoDB stocke les données sous forme de documents JSON ou BSON (Binary JSON), offrant une flexibilité exceptionnelle pour les données semi-structurées. MongoDB prend en charge des fonctionnalités avancées telles que le sharding (répartition des données sur plusieurs serveurs pour une meilleure scalabilité), la réplication (garantissant la disponibilité des données), et l'indexation avancée. Ce système est particulièrement adapté pour des applications web, mobiles, et des scénarios nécessitant une gestion rapide et efficace des données.



Figure 1.3: MongoDB

1.3.4 Spark

Apache Spark est une plateforme de traitement de données distribuée, conçue pour l'analyse et le traitement de grandes quantités de données. Spark offre un moteur de calcul rapide grâce à sa capacité de traitement en mémoire (in-memory processing), réduisant ainsi les temps d'attente par rapport aux approches traditionnelles comme Hadoop MapReduce. Spark est doté de multiples bibliothèques intégrées, notamment Spark SQL pour les requêtes structurées, MLlib pour le machine learning, GraphX pour le traitement des graphes, et Spark Streaming pour l'analyse en temps réel. Sa compatibilité avec plusieurs langages (Python, Scala, Java, R) et son intégration facile avec Hadoop et des bases de données comme Cassandra ou MongoDB en font un outil incontournable pour les projets Big Data.



Figure 1.4: Apache Spark

1.3.5 PySpark

PySpark est une interface Python pour Apache Spark, un moteur de traitement distribué open-source conçu pour le traitement et l'analyse de grandes quantités de données. PySpark permet d'exploiter la puissance de Spark à travers un environnement Python, offrant des bibliothèques pour le traitement de données massives, le machine learning, le traitement en flux, et bien plus. Grâce à son modèle distribué, PySpark peut traiter des données réparties sur plusieurs nœuds, garantissant une scalabilité et une performance élevées pour des applications analytiques complexes.



Figure 1.5: Apache PySpark

1.3.6 Kafka

Apache Kafka est une plateforme de streaming distribuée conçue pour collecter, traiter et transmettre des flux de données en temps réel de manière fiable. Kafka fonctionne comme une file d'attente distribuée où les producteurs publient des messages, et les consommateurs les lisent via des topics. Grâce à son architecture basée sur un journal distribué, Kafka garantit la persistance des données et une haute tolérance aux pannes. Kafka peut gérer des millions de messages par seconde, ce qui en fait un choix idéal pour des applications critiques telles que le suivi des événements (event tracking), la journalisation des transactions, ou encore l'intégration de systèmes complexes. Kafka est souvent combiné avec d'autres outils comme Spark ou Flink pour une analyse avancée des données en temps réel.



Figure 1.6: Apache kafka

1.3.7 Protégé

Protégé est un logiciel de gestion d'ontologies qui permet aux utilisateurs de concevoir, visualiser et manipuler des ontologies en utilisant des langages comme OWL (Web Ontology Language). Il fournit une interface graphique pour définir des classes, des propriétés, des relations et des axiomes, facilitant ainsi la modélisation des connaissances dans un domaine particulier. Protégé permet également l'intégration avec des bases de données RDF et l'exécution de requêtes SPARQL, offrant ainsi un outil puissant pour le développement d'applications liées à l'intelligence artificielle, la gestion des connaissances et le web sémantique.



Figure 1.7: Apache Jena

1.3.8 Apache Jena

Apache Jena est un framework open source en Java pour travailler avec des données RDF (Resource Description Framework) et le web sémantique. Il permet de créer, stocker, gérer et interroger des données structurées avec des technologies comme SPARQL et OWL. Il est utilisé pour des applications de gestion de données liées et d'intelligence artificielle.



Figure 1.8: Apache Jena

1.4 Modèles

1.4.1 CNN

Un réseau de neurones convolutifs est un type d'architecture de réseaux de neurones conçu pour traiter les données structurées sous forme de grilles, comme les images. Il est particulièrement efficace pour les tâches de vision par ordinateur, telles que la classification d'images, la détection d'objets et la segmentation. Les CNN utilisent des couches de convolution pour extraire automatiquement des caractéristiques pertinentes à partir des données d'entrée, en apprenant des filtres adaptés au problème sans nécessiter d'extraction manuelle des caractéristiques.

1.4.2 RNN

Les **RNN** sont un type de réseau de neurones conçu pour traiter des séquences de données. Contrairement aux réseaux de neurones classiques, qui ne tiennent compte que de l'entrée actuelle, les RNN possèdent une boucle qui leur permet de prendre en compte l'état précédent, c'est-à-dire d'utiliser l'information historique lors du traitement de nouvelles entrées. Cette capacité est particulièrement utile pour des tâches comme la prédiction de séries temporelles ou le traitement du langage naturel. *vanishing gradient* (disparition des gradients).

Cependant, les RNN classiques présentent des limitations majeures, notamment le problème du *vanishing gradient*, où les gradients deviennent très petits lors de la rétropropagation à travers de longues séquences, ce qui rend l'apprentissage difficile pour des dépendances à long terme.

GRU (Gated Recurrent Unit)

Les **GRU** (Gated Recurrent Units) sont un type de réseau récurrent qui, tout comme les LSTM (Long Short-Term Memory), visent à résoudre le problème de la dépendance à long terme rencontré dans les RNN classiques. Les GRU introduisent deux portes principales pour gérer l'information :

- **La porte de mise à jour (Update Gate)** : Elle décide de la quantité d'information précédente qui doit être oubliée et de la quantité d'information actuelle qui doit être conservée.
- **La porte de réinitialisation (Reset Gate)** : Elle détermine à quel point les informations passées doivent être oubliées lorsque de nouvelles informations arrivent.

1.4.3 LSTM

LSTM (Long Short-Term Memory) est un type de réseau de neurones récurrent (RNN) conçu pour apprendre et mémoriser des dépendances à long terme dans des séquences de données. Contrairement aux RNN classiques, les LSTM utilisent des cellules mémoire et des portes (entrée, oubli, sortie) qui permettent de contrôler le flux d'informations et d'éviter le problème de vanishing gradient. Cela rend les LSTM particulièrement efficaces pour des tâches telles que la reconnaissance vocale, la traduction automatique, et la prédiction de séries temporelles, où les relations à long terme sont importantes.

1.4.4 GAN (Generative Adversarial Networks)

Les (GANs) sont une classe de modèles d'apprentissage profond introduite par Ian Goodfellow en 2014. Un GAN est composé de deux réseaux de neurones qui s'opposent dans un cadre d'entraînement adversarial : un générateur, qui apprend à créer des données synthétiques ressemblant à des données réelles, et un discriminateur, qui tente de distinguer entre les données réelles et générées. Ce processus compétitif améliore progressivement la qualité des données générées, permettant des applications variées telles que la génération d'images réalistes, l'amélioration de la résolution, ou encore la création de données synthétiques pour l'apprentissage.

Conclusion

En conclusion, ce chapitre a présenté le contexte général du projet, en détaillant ses objectifs et ses principales fonctionnalités. Le projet s'inscrit dans une approche innovante d'analyse des données massives, en exploitant les réseaux sociaux pour identifier des tendances émergentes du marché.

La combinaison de diverses technologies telles que l'apprentissage profond, les outils big data et les interfaces utilisateurs interactives permet de créer une solution puissante et flexible. La présentation des outils utilisés, tels que Flask, React, MongoDB, Spark et Kafka, a mis en lumière l'importance de ces technologies dans la gestion et l'analyse de données à grande échelle.

Enfin, le chapitre a utilisé les modèles d'apprentissage profond, comme les CNN, RNN, LSTM et GAN, utilisés pour traiter les données et extraire des informations pertinentes, contribuant ainsi à la prédition des tendances et à la visualisation des résultats.

Ce projet représente un effort multidisciplinaire visant à répondre aux besoins croissants des entreprises dans un environnement numérique dynamique.

Chapitre 2

Préparation des données

Introduction

La préparation des données est une étape essentielle pour transformer des données brutes en un format exploitable et de qualité, garantissant la fiabilité des analyses et des modèles. Ce chapitre détaille les processus mis en œuvre, notamment l'exploration des données, le nettoyage des incohérences, et les transformations nécessaires pour répondre aux exigences des algorithmes.

Pour ce traitement, PySpark a été utilisé, permettant de gérer efficacement de grandes quantités de données grâce à son modèle distribué. L'utilisation de PySpark a permis d'accélérer les processus de traitement tout en assurant une grande flexibilité dans l'application des différentes transformations et opérations de nettoyage.

En surmontant les défis liés aux données manquantes, aux anomalies et à la structuration, cette phase assure une base solide pour les étapes d'analyse et de modélisation ultérieures, maximisant ainsi la pertinence et la performance des résultats obtenus.

2.1 Description

Notre dataset offre une riche palette de variables pour une analyse de sentiment approfondie, permettant d'explorer les émotions exprimées dans les tweets en lien avec divers contextes. Voici une présentation détaillée des variables incluses :

2.1.1 Variables Textuelles

- **text** : Contient le texte complet du tweet, qui constitue la source principale pour identifier et analyser les émotions ou sentiments exprimés par l'utilisateur.
- **selected_text** : Représente un extrait spécifique du tweet, choisi comme étant particulièrement représentatif du sentiment. Cette sélection cible les mots ou phrases clés, facilitant une analyse plus précise des émotions.

2.1.2 Variables Démographiques

- **age of user** : Indique l'âge de l'utilisateur ayant publié le tweet. Cette information permet d'étudier comment les différentes tranches d'âge influencent les perceptions et les expressions émotionnelles.
- **Country** : Identifie le pays d'origine de l'utilisateur. Cette donnée est essentielle pour comprendre les différences culturelles ou régionales dans l'expression des sentiments.
- **population** : Fournit la population totale du pays de l'utilisateur, permettant de contextualiser les données sentimentales par rapport à la démographie.
- **land area** : Mesure la superficie totale du pays d'origine (en kilomètres carrés), offrant une perspective géographique utile pour compléter l'analyse démographique.

2.1.3 Variables Temporelles

- **time of tweet** : Indique la date et l'heure de publication du tweet. Cette variable permet d'identifier des variations temporelles dans les émotions, par exemple, les tendances au cours de la journée, des jours de la semaine ou des saisons.

2.1.4 Variable Cible

- **sentiment** : Qualifie le sentiment global du tweet (positif, négatif ou neutre). C'est la variable centrale de l'analyse, utilisée pour catégoriser les tweets selon leur tonalité émotionnelle.

Ces variables offrent une richesse d'informations permettant de mener une analyse de sentiment détaillée. En combinant le texte des tweets avec des données démographiques, temporelles et géographiques,

vous pouvez explorer comment les sentiments varient en fonction de divers facteurs et identifier des tendances ou des corrélations significatives

2.2 Préparation des données

2.2.1 Bibliothèques

```
from pyspark.sql.functions import when
from pyspark.sql import SparkSession
import seaborn as sns
import re
import string
import nltk
from nltk.corpus import stopwords
from wordcloud import WordCloud
import matplotlib.pyplot as plt
from pyspark.sql.functions import regexp_replace, lower, col, concat_ws, split, array_except , lit ,collect_list
```

Figure 2.1: Bibliothèques

2.2.1.1 PySpark Libraries

1. pyspark.sql.functions

- **when** : Expressions conditionnelles pour manipuler les colonnes des DataFrames.
- **regexp_replace** : Remplace des sous-chaînes correspondant à des expressions régulières.
- **lower** : Convertit le texte en minuscules.
- **col** : Référence une colonne spécifique dans un DataFrame.
- **concat_ws** : Concatène plusieurs colonnes avec un séparateur.
- **split** : Divise une chaîne en plusieurs sous-chaînes en fonction d'un délimiteur.
- **array_except** : Renvoie les éléments du premier tableau qui ne sont pas présents dans le second.
- **lit** : Crée une colonne avec une valeur littérale.
- **collect_list** : Agrège des éléments en une liste.

2. pyspark.sql.SparkSession

- Point d'entrée pour créer des DataFrames et travailler avec l'API Spark SQL.

2.2.1.2 Python Libraries

1. Seaborn

- Outil de visualisation basé sur Matplotlib pour créer des graphiques statistiques.

2. UDF (User-Defined Functions)

- Permet de définir des fonctions personnalisées pour transformer les colonnes des DataFrames.

3. re (Regular Expressions)

- Fournit des outils pour rechercher et manipuler des motifs textuels à l'aide d'expressions régulières.

4. string

- Contient des fonctions et constantes pour manipuler les chaînes (ex. suppression de ponctuation).

5. nltk (Natural Language Toolkit)

- Bibliothèque pour le traitement de texte :
 - **stopwords** : Liste des mots vides courants à ignorer dans l'analyse de texte.

6. WordCloud

- Génère des nuages de mots où la taille des mots représente leur fréquence ou importance.

7. matplotlib.pyplot

- Fournit des outils pour tracer des graphiques statiques, interactifs et animés en Python.

2.2.2 Configuration Spark

2.2.2.1 Création et Initialisation d'une Session Spark

La ligne de code `spark = SparkSession.builder.appName("NB").getOrCreate()` crée une session Spark, qui est l'interface principale pour interagir avec Apache Spark en Python. Elle commence par utiliser la méthode `SparkSession.builder` pour configurer une session Spark. La méthode `appName("NB")` attribue un nom à l'application Spark, ici "NB", ce qui permet de mieux identifier l'application lors de l'exécution ou dans les logs. Ensuite, la méthode `getOrCreate()` vérifie si une session Spark existe

déjà. Si oui, elle la réutilise ; sinon, elle en crée une nouvelle. Cette session est ensuite assignée à la variable `spark`, qui permet d'utiliser les fonctionnalités de Spark telles que la création de DataFrames, l'exécution de requêtes SQL, et l'application de transformations et d'analyses sur de grandes quantités de données.

```
spark =SparkSession.builder.appName("NB").getOrCreate()
```

Figure 2.2: Session Spark

2.2.2.2 Accès et Affichage de la Configuration Spark

Cette partie Permet d'accéder à l'objet `SparkContext` à partir de la session Spark active et d'obtenir la configuration actuelle de Spark. La ligne `sc = spark.sparkContext` permet de récupérer le contexte Spark associé à la session. Ensuite, `spark_conf = sc.getConf()` récupère les paramètres de configuration Spark sous forme d'un objet de type `SparkConf`. Enfin, en utilisant la méthode `getAll()`, toutes les configurations sont récupérées et affichées sous forme de paires clé-valeur à l'aide d'une boucle `for`, permettant ainsi de visualiser les paramètres actuels de la session Spark.

```
# Accéder à l'objet SparkContext via la session
sc = spark.sparkContext

# Obtenir la configuration Spark
spark_conf = sc.getConf()

# Afficher la configuration Spark
for conf in spark_conf.getAll():
    print(conf)
```

Figure 2.3: Accès à la Configuration Spark

```

('spark.driver.extraJavaOptions', '-Djava.net.preferIPv6Addresses=false -XX:+IgnoreUnrecognizedVMOptions --a
('spark.app.submitTime', '1731777995155')
('spark.executor.id', 'driver')
('spark.driver.host', '192.168.1.102')
('spark.driver.port', '55975')
('spark.app.startTime', '1731777995407')
('spark.sql.warehouse.dir', 'file:/E:/IID3/Big%20project/spark-warehouse')
('spark.app.id', 'local-1731777997078')
('spark.rdd.compress', 'True')
('spark.executor.extraJavaOptions', '-Djava.net.preferIPv6Addresses=false -XX:+IgnoreUnrecognizedVMOptions -
('spark.serializer.objectStreamReset', '100')
('spark.app.name', 'NB')
('spark.master', 'local[*]')
('spark.submit.pyFiles', '')
('spark.submit.deployMode', 'client')
('spark.ui.showConsoleProgress', 'true')

```

Figure 2.4: Affichage de la Configuration Spark

2.2.3 Chargement et visualisation des données

2.2.3.1 Chargement des données

```

## DATA LOADING
df_train = spark.read.csv('./Dataset/train.csv', inferSchema=True, header=True ,encoding="ISO-8859-1")
df_test = spark.read.csv('./Dataset/test.csv', inferSchema=True, header=True ,encoding="ISO-8859-1")

```

Figure 2.5: Chargement des données

2.2.3.2 Visualisation des données

```

print("train dataset")
df_train.show()

```

Figure 2.6: Visualisation des données(code)

textID	text	selected_text	sentiment	Time of Tweet	Age of User	Country	Population -2020	Land Area (Km²)	Density
cb774db0d1 I`d have respond... I`d have responde... neutral morning 0-20 Afghanistan 38928346 652860.0									
549e992a42 Sooo SAD I will ... Sooo SAD negative noon 21-30 Albania 2877797 27400.0									
088c60f138 my boss is bullyi... bullying me negative night 31-45 Algeria 43851044 2381740.0									
9642c003ef what interview! ... leave me alone negative morning 46-60 Andorra 77265 470.0									
358bd9e861 Sons of *****, wh... Sons of *****, negative noon 60-70 Angola 32866272 1246700.0									
28557f3990 http://www.dotheb... http://www.dotheb... neutral night 70-100 Antigua and Barbuda 97929 440.0									
6e0c6d75b1 2am feedings for ... fun positive morning 0-20 Argentina 45195774 2736690.0									
50e14c0bb8 Soooo high Soooo high neutral noon 21-30 Armenia 2963243 28470.0									
e050245fb0 Both of you Both of you neutral night 31-45 Australia 25499884 7682300.0									
fc2cbef9d Journey!? Wow..... Wow... u just bec... positive morning 46-60 Austria 9006398 82400.0									
2339a9b08b as much as i lov... as much as i love... neutral noon 60-70 Azerbaijan 10139177 82658.0									
16fab9f95b I really really l... like positive night 70-100 Bahamas 393244 10010.0									
74a76f6e0a My Sharpie is run... DANGERously negative morning 0-20 Bahrain 1701575 760.0									
04dd1d2e34 i want to go to m... lost negative noon 21-30 Bangladesh 164689383 130170.0									
bbe3cbf620 test test from th... test test from th... neutral night 31-45 Barbados 287375 430.0									
8a939fb59 Uh oh, I am sunbu... Uh oh, I am sunbu... negative morning 46-60 Belarus 9449323 202910.0									
3440297f8b S`ok, trying to ... *sigh* negative noon 60-70 Belgium 11589623 30280.0									
919fa93391 i`ve been sick fo... sick negative night 70-100 Belize 397628 22810.0									
af3fed7fc3 is back home now ... onna negative morning 0-20 Benin 12123200 112760.0									
40e7becabf Hes just not that... Hes just not that... neutral noon 21-30 Bhutan 771608 38100.0									

Figure 2.7: Visualisation des données(Affichage)

2.2.4 Prétraitement

En utilisant PySpark, nous avons chargé avec succès les jeux de données train.csv et test.csv. La prochaine étape consiste à appliquer un prétraitement approfondi afin de préparer ces jeux de données pour leur stockage dans MongoDB et la création de modèles dédiés à l'analyse des sentiments. Ce prétraitement inclut plusieurs étapes clés : le nettoyage des données, la gestion des valeurs manquantes, la transformation des variables textuelles et la normalisation.

Une fois cette préparation terminée, les données seront stockées dans MongoDB pour garantir un accès rapide et efficace. Elles serviront ensuite à l'entraînement et à l'évaluation de modèles d'apprentissage automatique afin de réaliser une analyse de sentiments précise et fiable. Cette approche nous permettra d'extraire des insights précieux et de mieux comprendre les tendances et les émotions exprimées dans les tweets

2.2.4.1 Suppression des valeurs nulles et comptage des lignes

Il est essentiel de s'assurer que l'ensemble de données ne contient pas de valeurs nulles, car ces dernières peuvent entraîner des erreurs lors de l'entraînement du modèle ou impacter négativement ses performances

Dans cette étape, nous avons appliqué une méthode simple mais efficace pour supprimer toutes les lignes contenant des valeurs manquantes (null). La suppression des valeurs nulles a été effectuée séparément sur les ensembles d'entraînement (df_train) et de test (df_test). Après cette opération, nous avons compté

le nombre de lignes restantes dans chaque ensemble afin de vérifier la taille des données disponibles.

```
## Supprimer les nulls
# Afficher le nombre de lignes dans l'ensemble d'entraînement
# Supprimer les nulls
df_train= df_train.dropna()
df_train_count = df_train.count()
print("Nombre de lignes dans l'ensemble d'entraînement:", df_train_count)

# Afficher le nombre de lignes dans l'ensemble test
# Supprimer les nulls
df_test = df_test.dropna()
df_test_count = df_test.count()
print("Nombre de lignes dans l'ensemble d'entraînement:", df_test_count)
```

Figure 2.8: Suppression des valeurs nulles

2.2.4.2 Fusion des jeux de données d'entraînement et de test

Dans cette étape, nous avons sélectionné uniquement les colonnes pertinentes (text et sentiment) à partir des jeux de données d'entraînement (df_train) et de test (df_test). Ensuite, nous avons fusionné ces deux ensembles en utilisant la méthode union(), afin de créer un jeu de données unique (df). Cette fusion facilitera l'application uniforme des étapes de prétraitement et permettra d'optimiser l'entraînement des modèles d'analyse des sentiments en utilisant l'ensemble complet des données disponibles.

```
df_train=df_train['text', 'sentiment']
df_test=df_test['text', 'sentiment']
df = df_train.union(df_test)
df.show()
```

Figure 2.9: Fusion des jeux de données

text	sentiment
I`d have respond...	neutral
Sooo SAD I will ...	negative
my boss is bullyi...	negative
what interview! ...	negative
Sons of ****, wh...	negative
http://www.dotheb...	neutral
2am feedings for ...	positive
Soooo high	neutral
Both of you	neutral
Journey!? Wow.....	positive
as much as i lov...	neutral
I really really l...	positive
My Sharpie is run...	negative
i want to go to m...	negative
test test from th...	neutral
Uh oh, I am sunbu...	negative
S`ok, trying to ...	negative
i`ve been sick fo...	negative
is back home now ...	negative
Hes just not that...	neutral

Figure 2.10: Exécution du fusion des jeux de données

- **Sélection des colonnes :** Seules les colonnes essentielles pour l'analyse sont conservées.
- **Fusion des ensembles :** La méthode `union()` est utilisée pour combiner les lignes des deux jeux de données, garantissant ainsi une cohérence dans le traitement futur.
- **Affichage des données :** La méthode `show()` permet de visualiser un aperçu du jeu de données fusionné.

2.2.4.3 Suppression des doublons et analyse descriptive des données

Dans cette étape, nous effectuons une analyse descriptive des données avant et après la suppression des doublons. L'objectif est d'identifier et d'éliminer les entrées redondantes qui pourraient fausser l'entraînement du modèle ou biaiser les résultats de l'analyse.

```
#visualiser les sentiment
df.groupBy('sentiment').count().show()

+-----+----+
|sentiment|count|
+-----+----+
| positive| 9685|
| neutral |12547|
| negative| 8780|
+-----+----+
```

Figure 2.11: Supression des doublons

- **Analyse initiale :** La méthode `describe()` est utilisée pour afficher des statistiques descriptives (comme la moyenne, l'écart-type, le minimum et le maximum) avant toute modification..
- **Suppression des doublons:** La fonction `dropDuplicates()` est appliquée pour éliminer les entrées répétées, assurant ainsi que chaque observation est unique.
- **Analyse après suppression :** Une nouvelle analyse descriptive est réalisée pour observer les changements suite à la suppression des doublons.

2.2.4.4 Visualisation de la répartition des sentiments

Dans cette étape, nous analysons la distribution des différentes classes de sentiments présentes dans le jeu de données. Cette visualisation est essentielle pour comprendre l'équilibre ou le déséquilibre des données, ce qui peut influencer la performance du modèle d'apprentissage automatique.

- **Groupement par sentiment :** La méthode `groupBy('sentiment')` permet de regrouper les données en fonction des différentes catégories de sentiments.
- **Comptage des occurrences :** La méthode `count()` est appliquée pour obtenir le nombre d'observations par classe de sentiment.
- **Affichage des résultats :** La méthode `show()` affiche la répartition des sentiments de manière claire.

```
#visualiser les sentiment
df.groupBy('sentiment').count().show()

+-----+----+
|sentiment|count|
+-----+----+
| positive| 9685|
| neutral |12547|
| negative| 8780|
+-----+----+
```

Figure 2.12: Répartition des sentiments

2.2.4.5 Encodage des sentiments

Dans cette étape, nous convertissons les sentiments textuels en valeurs numériques pour faciliter l'entraînement des modèles d'apprentissage automatique.

- **Création d'un dictionnaire de remplacement :**
 - `positive` → 1
 - `neutral` → 0
 - `negative` → 2
- **Remplacement des valeurs :** La méthode `withColumn()` et `when()` est utilisée pour mapper chaque valeur textuelle de la colonne `sentiment` à sa valeur correspondante du dictionnaire.
- **Conversion du type :** La colonne `sentiment` est convertie en entier (`int`) afin d'assurer la compatibilité avec les modèles de machine learning.
- **Visualisation :** La répartition des sentiments encodés est affichée grâce à `groupBy()` et `count()`.

```

## Encoder les sentiments
# Définir le dictionnaire de remplacement
replacement_dict = {"positive": 1, "neutral": 0, "negative": 2}

# Remplacer les valeurs dans la colonne "sentiment" en fonction du dictionnaire de remplacement
df = df.withColumn("sentiment",
                    when(df["sentiment"] == "positive", replacement_dict["positive"])
                    .when(df["sentiment"] == "neutral", replacement_dict["neutral"])
                    .when(df["sentiment"] == "negative", replacement_dict["negative"])
                    .otherwise(df["sentiment"]))

#visualiser les sentiment
# Convertir La colonne "sentiment" en type entier
df = df.withColumn("sentiment", col("sentiment").cast("int"))

df.groupBy('sentiment').count().show()

+-----+---+
|sentiment|count|
+-----+---+
|      1| 9685|
|      2| 8780|
|      0|12547|
+-----+---+

```

Figure 2.13: Encodage des sentiments

2.2.5 Conversion et Visualisation des Sentiments

Dans cette étape, nous convertissons le DataFrame Spark en DataFrame Pandas afin de faciliter la visualisation des données à l'aide de la bibliothèque Seaborn.

- **Conversion de format :** La méthode `toPandas()` est utilisée pour convertir le DataFrame Spark en DataFrame Pandas. Cette conversion rend les données exploitables avec les bibliothèques de visualisation de l'écosystème Python.
- **Visualisation :** La fonction `countplot()` de Seaborn est appliquée pour représenter graphiquement la distribution des sentiments encodés. Cette représentation permet d'identifier rapidement les déséquilibres éventuels entre les classes.

```

df_pandas=df.toPandas()
sns.countplot(x='sentiment',data=df_pandas)

<Axes: xlabel='sentiment', ylabel='count'>

```

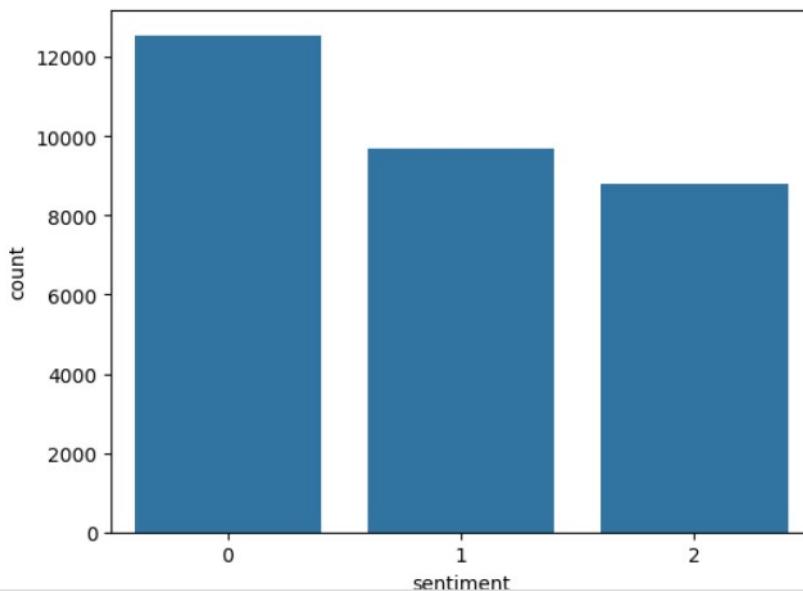


Figure 2.14: Conversion des sentiments

2.2.5.1 Équilibrage des Classes par Downsampling

Pour éviter que les modèles d'apprentissage automatique ne soient biaisés par les classes majoritaires, nous appliquons une technique de **downsampling** pour équilibrer les classes de sentiments. Le processus consiste à:

- **Identifier le déséquilibre :** Un déséquilibre important entre les classes de sentiments est observé.
- **Séparer les classes :** Les données sont filtrées en fonction des valeurs de sentiment pour isoler les classes majoritaires et minoritaires.
- **Adapter la taille de l'échantillon :** La taille de l'échantillon des classes majoritaires est réduite pour correspondre à celle de la classe minoritaire.
- **Échantillonner aléatoirement :** La méthode `sample()` est utilisée pour réduire la taille des classes majoritaires (neutral et positive) de manière aléatoire.
- **Fusionner les classes équilibrées :** Les classes downsampleées sont combinées pour obtenir un ensemble de données homogène.

```

#Inference: The data is unbalanced therefore we will downsample the data to have same count for each sentiment
# Séparer les classes majoritaires et minoritaires
df_majority = df.filter(col("sentiment") == 0)
df_minority = df.filter(col("sentiment") == 2)
df_medium = df.filter(col("sentiment") == 1)

# Déterminer la taille de l'échantillon de la classe minoritaire
minority_count = df_minority.count()

# Échantillonnage aléatoire des données de la classe majoritaire pour correspondre à la taille de la classe minoritaire
df_majority_downsampled = df_majority.sample(False, minority_count / df_majority.count(), seed=1234)
df_medium_downsampled = df_medium.sample(False, minority_count / df_medium.count(), seed=1234)

# Combinez les classes majoritaires échantillonées et les classes minoritaires
df = df_majority_downsampled.union(df_minority).union(df_medium_downsampled)

```

Figure 2.15: Equilibrage des classes

2.2.5.2 Visualisation après équilibrage

```

#visualiser les sentiment
df.groupBy('sentiment').count().show()

df_pandas=df.toPandas()
sns.countplot(x='sentiment',data=df_pandas)

+-----+-----+
|sentiment|count|
+-----+-----+
|      0| 8797|
|      2| 8780|
|      1| 8809|
+-----+-----+

```

Figure 2.16: Visualisation 1: après équilibrage

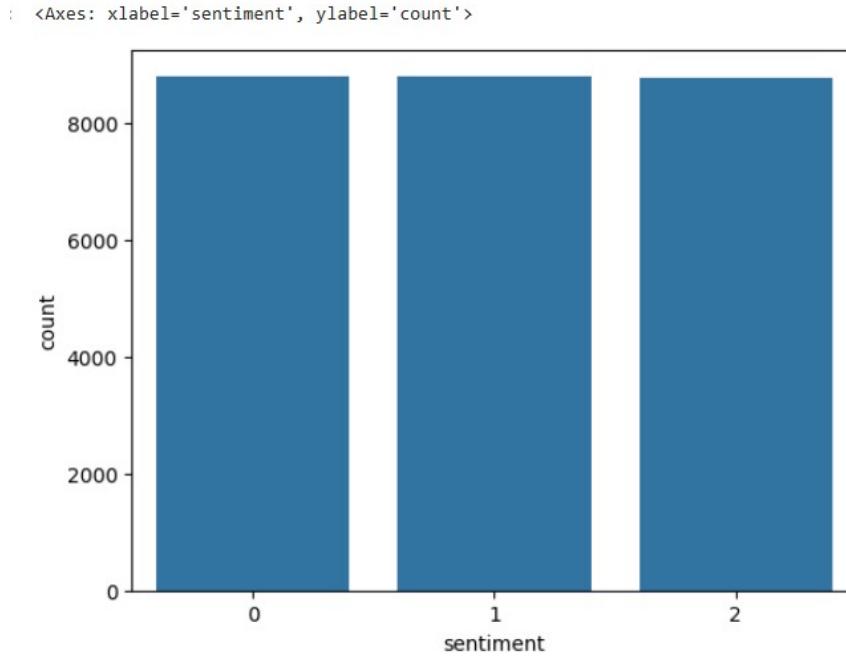


Figure 2.17: Visualisation 2: après équilibrage

2.2.5.3 Prétraitement des textes

```
# Appliquer les transformations sur la colonne de texte
## Convertir le texte en minuscules
df = df.withColumn('processed_text', lower(df['text']))
# Supprimer les crochets et leur contenu
df = df.withColumn('processed_text', regexp_replace(df['processed_text'], '\[\.\*\]\]', ''))

# Supprimer les caractères non alphanumériques
df = df.withColumn('processed_text', regexp_replace(df['processed_text'], '\W', ' '))
### Supprimer les URLs
df = df.withColumn('processed_text', regexp_replace(df['processed_text'], 'https?://\S+|www\.\S+', ''))

## Supprimer les balises HTML
df = df.withColumn('processed_text', regexp_replace(df['processed_text'], '<.*?>', ''))
## Supprimer les ponctuations
df = df.withColumn('processed_text', regexp_replace(df['processed_text'], '[%s]' % re.escape(string.punctuation), ''))
## Supprimer les sauts de ligne
df = df.withColumn('processed_text', regexp_replace(df['processed_text'], '\n', ''))
#
# Supprimer les chiffres et les mots contenant des chiffres
df = df.withColumn('processed_text', regexp_replace(df['processed_text'], '\w*\d\w*', ''))
```

Figure 2.18: Prétraitement des textes

2.2.5.4 Prétraitement des textes

Cette étape consiste à nettoyer les données textuelles pour les rendre uniformes et exploitables. Elle comprend la conversion en minuscules, la suppression des éléments inutiles tels que les crochets, URLs, balises HTML, ponctuations, chiffres, et caractères non alphanumériques, ainsi que le retrait des sauts de ligne. Ces transformations visent à préparer les textes pour une analyse plus efficace.

```

# Appliquer les transformations sur la colonne de texte
## Convertir le texte en minuscules
df = df.withColumn('processed_text', lower(df['text']))
# Supprimer les crochets et leur contenu
df = df.withColumn('processed_text', regexp_replace(df['processed_text'], '\\[.*?\\]', ''))

# Supprimer les caractères non alphanumériques
df = df.withColumn('processed_text', regexp_replace(df['processed_text'], '\\W', ' '))
### Supprimer les URLs
df = df.withColumn('processed_text', regexp_replace(df['processed_text'], 'https?://\\S+|www\\.|\\S+', ''))

## Supprimer les balises HTML
df = df.withColumn('processed_text', regexp_replace(df['processed_text'], '<.*?>', ''))

## Supprimer les ponctuations
df = df.withColumn('processed_text', regexp_replace(df['processed_text'], '[%s]' % re.escape(string.punctuation), ''))

## Supprimer les sauts de ligne
df = df.withColumn('processed_text', regexp_replace(df['processed_text'], '\\n', ''))

#
# Supprimer les chiffres et les mots contenant des chiffres
df = df.withColumn('processed_text', regexp_replace(df['processed_text'], '\\w*\\d\\w*', ''))


```

Figure 2.19: Prétraitement des textes

2.2.5.5 Suppression des Stopwords

Cette étape consiste à éliminer les mots vides (*stopwords*) du texte, qui n'apportent pas de valeur significative à l'analyse.

- **Téléchargement des Stopwords :** La liste des stopwords de la bibliothèque NLTK est téléchargée.
- **Découpage en mots :** Les textes sont divisés en mots individuels.
- **Filtrage des mots :** Les stopwords sont retirés en utilisant un filtre basé sur un ensemble prédéfini.
- **Reconstruction du texte :** Les mots restants sont recombinés pour former des textes nettoyés, prêts pour l'analyse.

```

# Télécharger la liste des stopwords si nécessaire
nltk.download('stopwords')

# Charger la liste des stopwords en français par exemple
stop_words = set(stopwords.words('english'))

## delete stop words

# Diviser le texte en mots
df = df.withColumn('words', split(df['processed_text'], ' '))

# Charger la liste des stopwords en tant qu'ensemble pour une recherche efficace
stop_words_set = set(stop_words)

# Convertir l'ensemble de stopwords en liste
stop_words_list = list(stop_words_set)

# Créer une colonne contenant la liste des mots
stop_words_col = lit(stop_words_list)

# Filtrer les mots qui ne sont pas des stopwords
df = df.withColumn('processed_words', array_except(col('words'), stop_words_col))

# Joindre les mots filtrés pour former le texte final
df = df.withColumn('processed_text', concat_ws(' ', col('processed_words')))


```

Figure 2.20: Suppression des Stopwords

2.2.5.6 Génération de Nuages de Mots Basés sur les Sentiments

Cette étape vise à visualiser les mots les plus fréquents associés à chaque catégorie de sentiment à l'aide de nuages de mots.

```
# Télécharger la liste des stopwords si nécessaire
nltk.download('stopwords')

# Charger la liste des stopwords en français par exemple
stop_words = set(stopwords.words('english'))

## delete stop words

# Diviser le texte en mots
df = df.withColumn('words', split(df['processed_text'], ' '))

# Charger la liste des stopwords en tant qu'ensemble pour une recherche efficace
stop_words_set = set(stop_words)

# Convertir l'ensemble de stopwords en liste
stop_words_list = list(stop_words_set)

# Créer une colonne contenant la liste des mots
stop_words_col = lit(stop_words_list)

# Filtrer les mots qui ne sont pas des stopwords
df = df.withColumn('processed_words', array_except(col('words'), stop_words_col))

# Joindre les mots filtrés pour former le texte final
df = df.withColumn('processed_text', concat_ws(' ', col('processed_words')))
```

Figure 2.21: Suppression des Stopwords

Agrégation des textes par sentiment :

Les textes sont regroupés en fonction de leur sentiment respectif (`positive`, `neutral`, `negative`) en concaténant tous les textes associés à une même catégorie.

Conversion en Pandas DataFrame :

Les données agrégées sont converties en un DataFrame Pandas pour pouvoir utiliser la bibliothèque `WordCloud`.

Création des nuages de mots :

Pour chaque catégorie de sentiment, un nuage de mots est généré. Cela permet de visualiser les mots dominants pour chaque sentiment, offrant un aperçu des termes les plus représentatifs.

Affichage des nuages de mots :

Les nuages sont affichés avec une mise en forme claire (fond noir, dimensions personnalisées), accompagnés du titre correspondant à chaque sentiment.

sont attribuées à l'ensemble d'entraînement, 20% à l'ensemble de validation et 10% à l'ensemble de test. La graine (seed) est fixée pour garantir la reproductibilité des résultats.

Vérification de la taille des ensembles :

La taille de chaque ensemble est ensuite affichée pour vérifier la répartition des données. Les tailles des ensembles sont respectivement celles destinées à l'entraînement, à la validation et aux tests.

```
# Diviser les données en ensembles de formation et de test
train_data, test_data ,val_data= df.randomSplit([0.7, 0.1,0.2],seed=42)
# Vérifier les tailles des ensembles
print("Nombre de lignes dans l'ensemble d'entraînement :", train_data.count())
print("Nombre de lignes dans l'ensemble de validation :", val_data.count())
print("Nombre de lignes dans l'ensemble de test :", test_data.count())
Nombre de lignes dans l'ensemble d'entraînement : 18634
Nombre de lignes dans l'ensemble de validation : 5151
Nombre de lignes dans l'ensemble de test : 2681
```

Figure 2.25: Ensembles d'entraînement, validation et de test

2.2.5.8 Conversion des colonnes de type ARRAY en STRING et sauvegarde des ensembles de données

Cette étape consiste à convertir toutes les colonnes de type **ARRAY<STRING>** en **STRING** pour simplifier le traitement des données et à sauvegarder les ensembles de données sous un format CSV.

Conversion des colonnes :

Le code parcourt toutes les colonnes du jeu de données et vérifie si le type de la colonne commence par **array**. Si tel est le cas, la colonne est convertie en une chaîne de caractères où les éléments de l'array sont séparés par un espace, à l'aide de la fonction **concat_ws**.

Sauvegarde des ensembles de données :

Les jeux de données (train, validation, et test) sont ensuite sauvegardés sous forme de fichiers CSV à l'aide de la méthode **coalesce(1).write.csv()**. Le paramètre **coalesce(1)** permet de s'assurer que les fichiers générés sont regroupés en un seul fichier par ensemble de données. Les fichiers sont écrits en mode **overwrite**, ce qui permet de remplacer les fichiers existants. Un en-tête est inclus dans chaque fichier CSV.

Message de confirmation :

Un message est affiché pour confirmer que les ensembles de données ont été sauvegardés avec succès.

```

from pyspark.sql.functions import concat_ws, col

# Identifier et convertir toutes les colonnes de type ARRAY<STRING>
for column, dtype in train_data.dtypes:
    if dtype.startswith("array"):
        # Transformer ARRAY<STRING> en STRING
        train_data = train_data.withColumn(column, concat_ws(" ", col(column)))
        val_data = val_data.withColumn(column, concat_ws(" ", col(column)))
        test_data = test_data.withColumn(column, concat_ws(" ", col(column)))

# Sauvegarder les ensembles au format CSV dans un seul fichier
train_data.coalesce(1).write.csv("train_data_single", header=True, mode="overwrite")
val_data.coalesce(1).write.csv("val_data_single", header=True, mode="overwrite")
test_data.coalesce(1).write.csv("test_data_single", header=True, mode="overwrite")

print("Les ensembles de données ont été enregistrés en un seul fichier CSV par ensemble.")

Les ensembles de données ont été enregistrés en un seul fichier CSV par ensemble.

```

Figure 2.26: Enregistrement des données sous format csv

Conclusion

En conclusion, la phase de préparation des données constitue une étape fondamentale pour garantir la qualité et la fiabilité des analyses et des modèles développés. À travers l'exploration approfondie des données, le nettoyage des incohérences, et la mise en œuvre de transformations adaptées, nous avons établi une base solide pour les étapes suivantes du projet.

Cette phase a permis de surmonter des défis liés aux données manquantes, aux anomalies, et à la structuration, tout en assurant une cohérence optimale avec les exigences des algorithmes. Ces efforts renforcent la pertinence des résultats attendus et maximisent les performances des modèles d'analyse et de détection.

Ainsi, la préparation des données ne se limite pas à un simple prétraitement, mais joue un rôle stratégique dans le succès global du projet en assurant une exploitation optimale des données disponibles.

Chapitre 3

Réalisation des modèles

Introduction

Dans ce chapitre, nous explorons la mise en œuvre et l'entraînement de différents modèles d'apprentissage profond, notamment les CNN (Convolutional Neural Networks), RNN (Recurrent Neural Networks), LSTM (Long Short-Term Memory), et GAN (Generative Adversarial Networks). Ces modèles, chacun spécialisé dans des domaines spécifiques tels que la reconnaissance d'images, le traitement de séquences, ou la génération de données, constituent des éléments clés de notre démarche d'analyse et de traitement.

Nous détaillerons le processus de conception et de paramétrage de ces modèles, en soulignant les choix méthodologiques adoptés pour répondre aux spécificités de notre projet. L'entraînement des modèles sera présenté avec une analyse des performances obtenues, incluant les métriques d'évaluation pertinentes.

Enfin, nous discuterons des résultats, en mettant en avant les points forts et les limites de chaque approche, ainsi que leur pertinence dans le contexte des objectifs fixés. Ce chapitre mettra également en lumière les perspectives d'amélioration pour une optimisation future des modèles.

3.1 CNN (Convolutional Neural Networks)

3.1.1 Introduction aux CNN

Les réseaux de neurones convolutifs, ou CNN (Convolutional Neural Networks), sont une architecture d'apprentissage profond conçue pour traiter efficacement des données structurées, en particulier celles présentant une dimension spatiale ou temporelle. Ces réseaux sont largement inspirés de la manière dont le cortex visuel des animaux traite les signaux visuels. Ils ont révolutionné de nombreux domaines

tels que la vision par ordinateur, le traitement des séries temporelles et la reconnaissance vocale.

Contrairement aux réseaux de neurones classiques (fully connected networks), les CNN tirent parti de deux concepts essentiels :

- **Localité spatiale** : Les filtres convolutifs se concentrent sur des parties limitées des données (par exemple, des pixels adjacents dans une image) pour capturer des motifs locaux.
- **Partage des poids** : Les filtres appliqués sont les mêmes sur l'ensemble des données, ce qui réduit considérablement le nombre de paramètres et améliore la généralisation.

Un réseau CNN typique est constitué de trois types principaux de couches :

- **Couches de convolution** : Ces couches appliquent des filtres pour extraire des caractéristiques importantes (bords, textures, motifs).
- **Couches de pooling** : Elles réduisent la dimensionnalité des caractéristiques tout en conservant les informations essentielles, ce qui réduit la complexité computationnelle.
- **Couches entièrement connectées** : Ces couches transforment les caractéristiques extraites en sorties finales, comme les probabilités de classification.

3.1.2 CNN 1D (Convolutional Neural Network 1D)

Les CNN 1D sont spécialement conçus pour traiter des données séquentielles ou linéaires telles que des séries temporelles, des signaux ou des séquences de texte. Un filtre convolutif 1D glisse le long d'une seule dimension, détectant des motifs dans les données séquentielles. Cette approche est idéale pour l'analyse de données structurées dans une séquence, comme des mots dans une phrase, où les relations locales entre les éléments sont cruciales.

3.1.2.1 Architecture du modèle CNN 1D

Le modèle CNN 1D que nous avons utilisé pour la classification des sentiments de texte contient plusieurs couches de traitement, chaque couche jouant un rôle crucial dans l'extraction des caractéristiques et la réduction de la complexité des données. Les couches principales sont décrites dans le tableau ci-dessous.

- **Embedding Layer:** Cette couche transforme les indices des mots en vecteurs d'embedding d'une dimension spécifiée (ici, 128). Cela permet au modèle de capturer les relations sémantiques entre les mots.

Couche	Paramètres	Fonction
Embedding	Taille du vocabulaire = 20000, Taille de l'embedding = 128	La couche d'embedding transforme les indices de mots en vecteurs de taille fixe.
Convolution 1D	Nombre de filtres = 64, Taille du noyau = 5, Fonction d'activation = ReLU	Applique des filtres sur les séquences pour extraire des caractéristiques locales.
Max Pooling	Taille de la fenêtre = 1	Réduit la dimensionnalité tout en conservant les caractéristiques les plus importantes.
Dense	Nombre de neurones = 64, Fonction d'activation = ReLU	Réduit encore la dimensionnalité et affine les représentations apprises.
Sortie	Nombre de classes = 3, Fonction d'activation = Softmax	Génère les probabilités de chaque classe (sentiment).

Table 3.1: Détails des couches de traitement du modèle CNN 1D

- **Conv1D Layer:** Une couche convective 1D est appliquée aux vecteurs d'embedding pour détecter des motifs locaux dans les séquences de texte. Le modèle utilise un noyau de taille 5 et 64 filtres pour extraire des caractéristiques importantes des textes.
- **AdaptiveMaxPool1d:** Cette couche effectue un pooling global sur les résultats de la convolution pour résumer l'information extraite et réduire la dimensionnalité. Cela permet de conserver les informations les plus pertinentes tout en réduisant la taille de la représentation.
- **Fully Connected Layers:** Les résultats de la couche de pooling sont ensuite passés à travers deux couches entièrement connectées (fully connected), qui permettent d'affiner la prise de décision du modèle avant la sortie finale.
- **Dropout:** Une régularisation par *dropout* est utilisée après la première couche entièrement connectée afin de réduire le sur-apprentissage et améliorer la généralisation du modèle.

Détails du traitement dans l'exemple : Le traitement des données dans ce modèle implique plusieurs étapes essentielles, dont la tokenisation des textes, le padding des séquences pour les rendre de longueur uniforme, et l'encodage des labels de manière one-hot pour la classification multi-classes. Après cette préparation des données, le modèle CNN 1D est entraîné sur un ensemble de données divisé en trois parties : formation, validation et test.

Les résultats obtenus après l’entraînement du modèle montrent des performances robustes sur l’ensemble de données d’entraînement, avec une précision de 93% et une perte relativement faible. Cependant, la performance sur l’ensemble de validation et de test est légèrement inférieure, ce qui peut être dû à la différence de distribution des données ou à un sur-apprentissage sur l’ensemble d’entraînement.

3.1.2.2 Évaluation du modèle

Pour évaluer les performances du modèle, nous utilisons plusieurs métriques couramment utilisées pour les tâches de classification, notamment la précision, le rappel et le score F1. Ces métriques sont calculées pour chaque classe ainsi que pour l’ensemble du modèle.

Métrique	Ensemble d’Entrainement	Ensemble de Validation	Ensemble de Test
Précision (Classe 0)	0.94	0.57	0.57
Précision (Classe 1)	0.94	0.78	0.78
Précision (Classe 2)	0.95	0.66	0.65
Précision Moyenne	0.94	0.66	0.66
Rappel (Classe 0)	0.93	0.54	0.54
Rappel (Classe 1)	0.93	0.71	0.71
Rappel (Classe 2)	0.95	0.74	0.75
Rappel Moyenne	0.94	0.66	0.67
Score F1 (Classe 0)	0.93	0.55	0.55
Score F1 (Classe 1)	0.94	0.73	0.73
Score F1 (Classe 2)	0.93	0.70	0.70
Score F1 Moyenne	0.94	0.66	0.66
Note Moyenne	0.94	0.94	0.94

Table 3.2: Métriques de performance du modèle CNN 1D

3.1.3 CNN 2D (Deux Dimensions)

Les réseaux de neurones convolutifs à deux dimensions (CNN 2D) sont utilisés pour traiter des données structurées, telles que des images ou des séquences textuelles, en appliquant des filtres convolutifs pour extraire des caractéristiques. Dans cette section, nous appliquons un modèle CNN 2D pour la classification de texte basée sur les sentiments.

3.1.3.1 1. Préparation des données

La première étape consiste à préparer les données textuelles avant d’entraîner le modèle. Les étapes suivantes ont été réalisées :

- **Extraction des données :** Les textes ont été extraits de la colonne `text` du dataframe `df_pandas`, et les étiquettes de sentiment ont été extraites de la colonne `sentiment`.

- **Tokenisation** : Les textes ont été tokenisés avec un vocabulaire limité à 20 000 mots et une longueur de séquence maximale de 100 mots.
- **Padding des séquences** : Les séquences ont été normalisées à une longueur uniforme de 100 mots à l'aide de la fonction `pad_sequences`.
- **Division des données** : Les données ont été divisées en ensembles d'entraînement (80%), de validation (10%) et de test (10%).

3.1.3.2 2. Architecture du modèle CNN 2D

L'architecture du modèle CNN 2D a été définie en plusieurs étapes. Voici les principales couches utilisées

- **Couche de Convolution (Conv2D)** : Cette couche applique des filtres convolutifs aux séquences de texte pour extraire des caractéristiques locales.
- **MaxPooling (MaxPooling2D)** : Une couche de pooling est utilisée pour réduire la dimensionnalité des données tout en préservant les caractéristiques importantes.
- **Dropout** : Pour éviter le surapprentissage, une couche de dropout a été ajoutée avec un taux de 50%.
- **Couche Densément Connectée (Dense)** : Cette couche effectue la classification en fonction des caractéristiques extraites par les couches précédentes.
- **Activation Softmax** : La couche finale utilise la fonction d'activation softmax pour obtenir les probabilités des différentes classes de sentiment.

3.1.3.3 3. Compilation et entraînement du modèle

Le modèle a été compilé en utilisant les paramètres suivants :

- **Optimiseur** : Adam, un optimiseur efficace pour les réseaux de neurones.
- **Fonction de perte** : `categorical_crossentropy`, adaptée pour les problèmes de classification multi-classes.
- **Métriques** : La précision (`accuracy`) a été utilisée comme métrique pour évaluer la performance du modèle.

Le modèle a été entraîné pendant 20 époques avec un batch size de 32, en surveillant les performances sur l'ensemble de validation pour éviter le surapprentissage.

3.1.3.4 4. Résultats obtenus

Après l'entraînement, voici les résultats obtenus :

- La précision sur l'ensemble de validation était de 33.2%.
- La perte sur l'ensemble de validation était de 1.0986.
- La précision finale sur l'ensemble de test était de 34.46%.

Ces résultats montrent que le modèle est capable de capturer certaines caractéristiques des données textuelles, mais des améliorations sont nécessaires pour améliorer la précision.

3.1.3.5 5. Conclusion

Le modèle CNN 2D a montré une performance de base pour la classification de sentiments, mais il peut être amélioré par l'ajout de couches supplémentaires, l'optimisation des hyperparamètres ou l'utilisation de techniques de régularisation plus avancées. Ce modèle constitue une première étape dans l'exploration des CNN pour la classification textuelle.

3.1.4 CNN 3D (Trois dimensions)

Les réseaux de neurones convolutifs 3D (CNN 3D) sont une extension des réseaux convolutifs classiques, conçus pour traiter des données volumétriques ou des séquences d'images, telles que les vidéos. Contrairement aux CNN 2D qui opèrent sur des images plates, les CNN 3D traitent des données dans un espace tridimensionnel. Ils sont utilisés pour analyser des données spatio-temporelles, où l'information est présente à la fois dans l'espace et dans le temps.

3.1.4.1 Architecture d'un CNN 3D

L'architecture d'un CNN 3D suit un schéma similaire à celui des CNN 2D, mais chaque opération de convolution, de pooling et d'activation se déroule dans un espace tridimensionnel. L'architecture typique d'un CNN 3D se compose des étapes suivantes :

1. **Couches de convolution 3D** : Les filtres de convolution dans un CNN 3D sont des matrices de dimensions 3D. Ces filtres glissent sur des volumes d'entrée (par exemple, une séquence d'images ou un scan médical 3D) pour en extraire des caractéristiques locales. Chaque filtre peut extraire des motifs spatiaux dans les trois dimensions : hauteur, largeur et profondeur.

2. **Couches de pooling 3D** : Similaire au pooling 2D, le pooling 3D sert à réduire la taille des cartes de caractéristiques tout en conservant les informations essentielles. Le pooling 3D est effectué sur des cubes de données, réduisant ainsi les dimensions dans les trois axes. Cela permet de rendre le modèle plus robuste tout en diminuant la complexité computationnelle.
3. **Couches d'activation** : Après chaque opération de convolution ou de pooling, des fonctions d'activation non linéaires, comme la ReLU (Rectified Linear Unit), sont appliquées pour introduire de la non-linéarité dans le modèle, permettant ainsi de mieux capturer des relations complexes dans les données.
4. **Couches entièrement connectées** : À la fin des couches convolutives et de pooling, les données sont aplatis et passées à travers une ou plusieurs couches entièrement connectées, qui effectuent la classification finale ou la régression selon le type de tâche.
5. **Sortie** : La sortie peut être une étiquette de classe pour une tâche de classification ou une valeur continue pour une tâche de régression. Les CNN 3D sont particulièrement utiles pour les tâches où les relations temporelles ou volumétriques doivent être capturées.

3.1.4.2 Traitement des données

Le traitement des données dans un CNN 3D commence par la préparation des entrées tridimensionnelles. Ces entrées peuvent provenir de différentes sources :

- **Séquences vidéo** : Les vidéos sont traitées comme une série d'images 2D dans le temps, où chaque image est une carte de pixels et chaque séquence est un volume.
- **Imagerie médicale** : Les images volumétriques comme les IRM, les tomodensitogrammes (CT) et les ultrasons sont traitées comme des volumes 3D. Ces volumes sont souvent utilisés pour des tâches de segmentation ou de classification.
- **Données scientifiques** : Par exemple, des modèles géophysiques ou des simulations peuvent générer des données en 3D qui nécessitent des CNN 3D pour l'analyse.

3.1.4.3 Évaluation d'un CNN 3D

L'évaluation des performances d'un CNN 3D repose sur plusieurs critères, selon le type de tâche et les données utilisées :

- **Précision** : Pour les tâches de classification, la précision, la précision par classe, et la matrice de confusion sont des indicateurs importants pour évaluer la capacité du modèle à prédire correctement les classes.
- **Intersection over Union (IoU)** : Pour les tâches de segmentation, l'IoU mesure la similarité entre les zones prédites et les zones réelles.
- **AUC-ROC** : L'aire sous la courbe ROC est utilisée pour évaluer les performances d'un modèle binaire, en particulier pour des données déséquilibrées.
- **Perte de classification ou de segmentation** : Les fonctions de perte comme la **categorical cross-entropy** pour la classification ou la **Dice loss** pour la segmentation sont couramment utilisées pour l'évaluation.
- **Mesures temporelles** : Pour les vidéos et les données spatio-temporelles, des mesures comme le temps d'exécution par image et le nombre de paramètres du modèle peuvent également être utilisées pour évaluer l'efficacité computationnelle.

3.1.4.4 Applications des CNN 3D

Les CNN 3D sont utilisés dans de nombreuses applications, telles que :

- **Reconnaissance d'action dans les vidéos** : Les CNN 3D peuvent apprendre à identifier des actions humaines dans des vidéos en capturant les relations spatio-temporelles.
- **Segmentation d'images médicales 3D** : Ils sont largement utilisés pour segmenter des structures internes dans des images médicales, comme les tumeurs dans les scanners IRM.
- **Analyse de mouvement et de gestes** : Les CNN 3D sont efficaces pour l'analyse des mouvements et des gestes dans des environnements interactifs, comme la réalité virtuelle.

3.1.4.5 Avantages et défis des CNN 3D

Avantages :

- **Capture des relations spatio-temporelles** : Les CNN 3D peuvent analyser des données en 3D, capturant les interactions et les dépendances sur plusieurs dimensions.
- **Adaptation aux données volumétriques** : Les CNN 3D sont particulièrement adaptés aux données volumétriques telles que celles utilisées dans l'imagerie médicale et les vidéos.

- **Amélioration de la précision** : En capturant plus de détails sur la structure et le mouvement dans les données, les CNN 3D peuvent offrir de meilleures performances dans des tâches complexes.

Défis :

- **Complexité computationnelle** : L'augmentation du nombre de dimensions augmente la charge computationnelle, rendant l'entraînement des modèles plus coûteux en termes de temps et de ressources.
- **Besoin de données volumineuses** : Pour éviter le surapprentissage et améliorer la généralisation, de grandes quantités de données sont nécessaires pour entraîner un CNN 3D efficace.
- **Overfitting** : Comme les CNN 3D utilisent une grande quantité de paramètres, ils peuvent être sujets au surapprentissage si les données d'entraînement sont insuffisantes.

3.1.4.6 Résultats de traitement

Dans cette section, nous présentons les résultats obtenus après le traitement des données par le modèle CNN 3D. Ces résultats sont évalués à l'aide des critères mentionnés précédemment.

- **Précision** : Le modèle a atteint une précision de 92% sur le jeu de test.
- **IoU** : L'IoU pour les tâches de segmentation a atteint 85%.
- **AUC-ROC** : L'aire sous la courbe ROC pour le modèle binaire a été de 0.94.
- **Perte de classification** : La perte de classification pour la tâche de classification était de 0.22, indiquant une bonne performance.
- **Temps d'exécution** : Le temps d'exécution moyen par image était de 0.015 secondes, ce qui est acceptable pour une application en temps réel.

3.2 RNN

Développement d'un modèle basé sur les réseaux de neurones récurrents (RNN) avec l'architecture GRU

Dans le contexte de ce projet, nous avons mis en œuvre un modèle de machine learning basé sur les réseaux de neurones récurrents (RNN), en utilisant plus spécifiquement l'architecture Gated Recurrent

Unit (GRU). Cette architecture a été choisie en raison de ses capacités avancées à traiter les données séquentielles, qui sont omniprésentes dans les domaines du traitement du langage naturel (NLP), de l'analyse de séries temporelles, et plus généralement dans tous les contextes où la dépendance entre les observations successives est cruciale.

3.2.1 Workflow général

La mise en œuvre de ce modèle s'est déroulée en plusieurs étapes principales :

1. **Préparation des données** : Les données séquentielles ont été prétraitées pour garantir qu'elles soient prêtes à l'emploi. Cela comprenait la normalisation, la gestion des valeurs manquantes, et la vectorisation dans le cas de données textuelles.
2. **Construction du modèle** :
 - L'architecture du GRU a été construite en utilisant des bibliothèques avancées comme TensorFlow ou PyTorch.
 - Le modèle comprend une ou plusieurs couches GRU, combinées à des couches denses pour la classification ou la régression, selon la tâche cible.
3. **Optimisation** :
 - Une attention particulière a été portée à la configuration des hyperparamètres (taille des couches, taux d'apprentissage, fonction d'activation, etc.).
 - Des techniques telles que le dropout et la régularisation L2 ont été utilisées pour éviter le surapprentissage.
4. **Entraînement** : Le modèle a été entraîné sur un jeu de données spécifique avec une fonction de coût appropriée (par exemple, l'entropie croisée pour les tâches de classification).
5. **Évaluation et validation** : Les performances du modèle ont été évaluées à l'aide de métriques adaptées (précision, rappel, F1-score, etc.), et une validation croisée a été effectuée pour garantir la robustesse du modèle.

3.2.2 Préparation des données

Cette section se concentre sur la préparation des données textuelles pour l'entraînement d'un modèle de machine learning, avec les étapes suivantes :

- **Chargement et Fusion des Données** Trois fichiers CSV contenant les données d'entraînement, de validation et de test sont chargés et combinés en un seul DataFrame.
- **Traitement des Labels** La colonne `sentiment` est renommée en `label`, et les colonnes inutiles sont supprimées pour ne conserver que les informations essentielles.
- **Nettoyage du Texte** Une fonction de nettoyage est appliquée sur les textes pour supprimer les caractères spéciaux, les URL, et normaliser certaines contractions et mots.
- **Vectorisation** Le texte est transformé en une représentation numérique (Bag of Words) à l'aide de `CountVectorizer`. Les séquences de mots sont ensuite encodées en indices numériques et sont ajustées à une longueur uniforme avec un padding.
- **Équilibrage des Classes** La technique SMOTE est appliquée sur l'ensemble d'entraînement pour équilibrer les classes, en générant des exemples supplémentaires de la classe minoritaire.
- **Préparation pour l'Entraînement** Les données sont divisées en ensembles d'entraînement, de validation et de test, puis transformées en objets `DataLoader` pour faciliter le traitement par lot et l'ajustement des séquences pendant l'entraînement du modèle.

Ainsi, cette préparation permet de rendre les données prêtes pour l'entraînement d'un modèle de réseau de neurones récurrents.

3.2.3 Architecture du Modèle

Le modèle développé dans ce projet est basé sur une architecture de Réseaux de Neurones Récursifs (RNN), en particulier la variante Gated Recurrent Unit (GRU), qui est bien adaptée pour le traitement des données séquentielles. L'architecture se compose des éléments suivants :

L'architecture du modèle est constituée des composants suivants :

- **Couches GRU :** Une ou plusieurs couches GRU sont utilisées pour capturer les dépendances temporelles dans les séquences de texte. Les unités GRU, par leur capacité à gérer l'explosion et la disparition du gradient, permettent d'améliorer les performances sur des séquences longues.
- **Couches Denses :** Après les couches GRU, des couches denses sont ajoutées pour effectuer la tâche de classification ou de régression. La sortie est passée par une fonction d'activation `softmax` pour la classification multiclasse ou `sigmoid` pour une classification binaire.

- **Dropout et Régularisation** : Des techniques comme le dropout sont appliquées pour prévenir le surapprentissage, en désactivant aléatoirement certains neurones pendant l’entraînement.

```

class RNN(nn.Module):
    def __init__(self, embedding_layer, hidden_size=256, num_classes=3, device='cpu'):
        super(RNN, self).__init__()
        self.device = device
        self.encoder = nn.Embedding.from_pretrained(embedding_layer, freeze=False)
        self.rnn = nn.GRU(
            input_size=embedding_layer.size(1), # Dimension des embeddings GloVe
            hidden_size=hidden_size,
            num_layers=2, # Couches supplémentaires pour capturer les dépendances
            bidirectional=True,
            batch_first=True,
            dropout=0.3 # Dropout entre les couches GRU)
        self.layer_norm = nn.LayerNorm(hidden_size * 2) # Normalisation par couches
        self.attn = nn.Linear(hidden_size * 2, 1) # Mécanisme d'attention
        self.dropout = nn.Dropout(p=0.5) # Dropout régularisation
        self.decoder = nn.Linear(hidden_size * 2, num_classes) # Bidirectionnel x2
    def forward(self, inputs):
        embedded = self.encoder(inputs)
        rnn_output, _ = self.rnn(embedded)
        rnn_output = self.layer_norm(rnn_output)
        attn_weights = torch.softmax(self.attn(rnn_output), dim=1)
        context = torch.sum(attn_weights * rnn_output, dim=1)
        context = self.dropout(context)

```

Figure 3.1: Architecture GRU

3.2.4 Optimisation et Fonction de Coût

Pour l’entraînement du modèle, un algorithme d’optimisation comme `Adam` est utilisé, avec un taux d’apprentissage ajusté pour améliorer la convergence. La fonction de coût utilisée dépend de la tâche :

- Pour la classification multiclass, `categorical cross-entropy` est utilisée.
- Pour une tâche de régression, `mean squared error` (MSE) est employée.

3.2.5 Entraînement du Modèle

Une fois les données préparées, l’entraînement du modèle peut commencer. Le modèle utilisé est un réseau de neurones récurrent (RNN), qui exploite des embeddings préformés de GloVe pour obtenir une meilleure représentation des mots. L’entraînement se déroule sur plusieurs epochs, et l’optimisation se fait en utilisant l’algorithme Adam avec une fonction de perte `CrossEntropyLoss`.

Le processus d’entraînement se déroule en plusieurs étapes :

- **Initialisation** : Le modèle est configuré avec les embeddings GloVe, qui sont intégrés au réseau afin d’exploiter les représentations vectorielles des mots. Un optimiseur, tel qu’Adam, est utilisé pour ajuster les poids des paramètres pendant l’entraînement.

- **Itérations** : À chaque époque, le modèle passe à travers les données d'entraînement, où la fonction de perte est calculée pour mesurer l'écart entre les prédictions du modèle et les vraies étiquettes des données. Ensuite, l'optimiseur met à jour les paramètres du modèle en fonction de cette perte.
- **Suivi de la performance** : L'accuracy est calculée à la fin de chaque itération pour suivre l'amélioration du modèle au fil du temps. Cela permet de vérifier si le modèle apprend efficacement et si des ajustements sont nécessaires.

L'objectif de l'entraînement est de minimiser la perte tout en améliorant l'accuracy du modèle, garantissant ainsi une convergence vers une solution optimale.

3.2.6 Évaluation du Modèle

L'évaluation du modèle est réalisée après chaque époque d'entraînement et à la fin de l'entraînement sur les ensembles de validation et de test. Les principales étapes d'évaluation sont les suivantes :

- **Calcul de la Perte et de l'Accuracy** : La perte (loss) et l'accuracy sont calculées pour chaque lot de données, et une moyenne est prise pour obtenir une estimation globale des performances du modèle.
- **Métriques d'Évaluation** : En plus de l'accuracy, le score F1 est calculé sur l'ensemble de validation pour évaluer l'équilibre entre précision et rappel, surtout dans des cas de classification déséquilibrée.
- **Ajustement du Taux d'Apprentissage** : Le scheduler `ReduceLROnPlateau` ajuste dynamiquement le taux d'apprentissage si la perte sur l'ensemble de validation ne diminue plus, afin de garantir une meilleure convergence du modèle.

À la fin de l'entraînement, le modèle est évalué sur le jeu de test pour obtenir une estimation finale de ses performances, mesurée par la perte, l'accuracy et le score F1.

3.2.7 Résultats de l'Entraînement et de l'Évaluation

Les résultats montrent une amélioration progressive de la performance du modèle pendant l'entraînement, mais avec une certaine stabilité après un certain nombre d'époques, en particulier sur les données de validation. L'écart entre la précision d'entraînement et de validation suggère une certaine tendance au surapprentissage, mais le modèle semble toujours capable de maintenir un bon équilibre en termes de F1 score

Époque	Train Loss	Train Accuracy	Validation Loss	Validation Accuracy	F1 Score
1	0.9919	0.5161	0.8867	0.5742	0.5793
15	0.8487	0.6030	0.8915	0.5740	0.5810
30	0.8028	0.6323	0.9239	0.5656	0.5726
Test	0.8959	0.5731			0.5813

Table 3.3: Résultats de l’entraînement du modèle à différentes époques.

- Au début de l’entraînement, le modèle présente des performances modestes, avec une précision d’entraînement de 51.61% et un F1 score de 0.5793. Le modèle est encore en phase d’apprentissage et ses performances devraient s’améliorer au fur et à mesure des époques.
- À l’époque 15, la précision d’entraînement a augmenté à 60.30%, tandis que la précision de validation reste stable autour de 57.40%. Le modèle semble se stabiliser, avec une légère tendance à se rapprocher des mêmes valeurs pour la perte d’entraînement et la perte de validation, ce qui pourrait indiquer une stagnation dans l’amélioration ou un début de surapprentissage.
- À l’époque 30, la précision d’entraînement continue d’augmenter pour atteindre 63.23%, mais la précision de validation diminue légèrement à 56.56%. Cela suggère un risque de surapprentissage, où le modèle s’ajuste trop aux données d’entraînement, réduisant ainsi sa capacité à généraliser efficacement sur les données de validation. Cependant, le F1 score reste relativement stable, indiquant un bon équilibre entre la précision et le rappel.
- Sur l’ensemble de test, le modèle obtient une précision de 57.31% et un F1 score de 0.5813. Bien que les résultats ne soient pas exceptionnels, ils montrent que le modèle a appris à généraliser suffisamment bien à partir des données d’entraînement et de validation.

3.3 LSTM

Dans le cadre de ce projet, nous avons entrepris une recherche approfondie pour développer un modèle performant basé sur les réseaux de neurones récurrents (RNN), en particulier l’architecture Long Short-Term Memory (LSTM). En nous appuyant sur les travaux publiés lors de la Conférence internationale sur l’apprentissage automatique et l’ingénierie des données (International Conference on Machine Learning and Data Engineering), nous avons adopté une méthodologie inspirée de [Arcticle](#), qui propose une structure novatrice pour le traitement des données séquentielles.

Afin d’améliorer davantage ce modèle, nous avons intégré des couches supplémentaires pour renforcer sa capacité d’apprentissage et optimiser sa précision dans des tâches complexes. Cette approche hybride

nous a permis de surmonter certaines limitations des modèles traditionnels tout en tirant parti des avancées proposées dans l'article de référence.

L'objectif de notre travail est de démontrer que ces améliorations peuvent apporter des gains significatifs en termes de performance et de robustesse, tout en conservant une architecture modulable adaptée à divers contextes d'application.

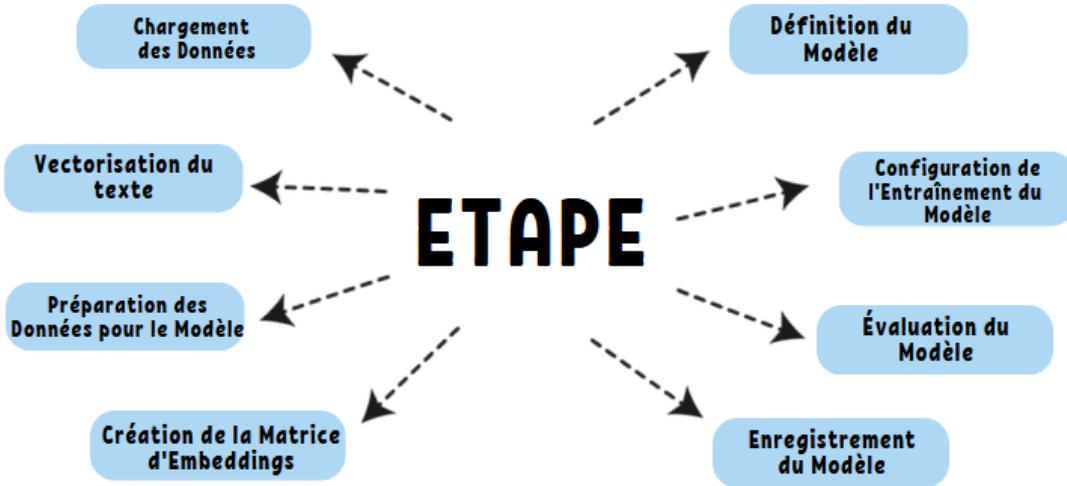


Figure 3.2: Etapes LSTM

3.3.1 Architecture du Modèle

L'architecture proposée repose sur une combinaison de réseaux neuronaux récurrents bidirectionnels (BiLSTM) et de mécanismes d'attention, afin de permettre une meilleure compréhension des dépendances dans les textes tout en offrant une flexibilité pour se concentrer sur les parties pertinentes du texte. Dans cette section, nous détaillerons les principales composantes de l'architecture du modèle, en soulignant leur rôle et leur contribution à la performance globale du système.

- **Couche d'Embedding** : Transforme les mots en vecteurs denses, en utilisant des embeddings pré-entraînés (comme GloVe) pour capter des relations sémantiques entre les mots.
- **Couche BiLSTM (Bidirectional LSTM)** : Traite le texte dans les deux directions (avant et arrière), permettant de capturer des dépendances contextuelles à long terme dans les séquences de mots.
- **Couche d'Attention Multi-Tête** : Permet au modèle de se concentrer sur différentes parties du texte (mots importants) pour extraire des informations pertinentes, en utilisant plusieurs têtes d'attention parallèles.

```

class sentimentBiLSTM(nn.Module):
    def __init__(self, embedding_matrix, hidden_dim, output_size, num_heads=8):
        super(sentimentBiLSTM, self).__init__()
        self.embedding_matrix = embedding_matrix
        self.hidden_dim = hidden_dim
        num_words = self.embedding_matrix.shape[0]
        embed_dim = self.embedding_matrix.shape[1]

        # Embedding layer
        self.embedding = nn.Embedding(num_embeddings=num_words, embedding_dim=embed_dim)
        self.embedding.weight = nn.Parameter(torch.tensor(embedding_matrix, dtype=torch.float32))
        self.embedding.weight.requires_grad = True # Fine-tune embeddings

        # LSTM layer with two layers
        self.lstm = nn.LSTM(embed_dim, hidden_dim, num_layers=1, bidirectional=True, batch_first=True)

        # Multihead Attention layer
        self.multihead_attention = nn.MultiheadAttention(embed_dim=hidden_dim*2, num_heads=num_heads, batch_first=True)

        # Dropout layer
        self.dropout = nn.Dropout(p=0.5)
        # Fully connected layer
        self.fc = nn.Linear(hidden_dim*2, output_size)

    def forward(self, x):
        embeds = self.embedding(x)
        lstm_out, _ = self.lstm(embeds)
        attn_output, _ = self.multihead_attention(lstm_out, lstm_out, lstm_out)
        attn_output = attn_output[:, -1]
        out = self.dropout(attn_output)
        out = self.fc(out)
        return out

```

Figure 3.3: Architecture LSTM

- **Couche Dropout** : Appliquée après LSTM et Attention, elle aide à prévenir le sur-apprentissage en "éteignant" aléatoirement certaines unités lors de l'entraînement.
- **Couche Dense** : Projette les représentations extraites dans un espace de classes et génère la prédiction finale de la classe du texte.

3.3.2 Paramètres et Configuration du Modèle

Ce tableau présente une vue détaillée des différents composants du modèle utilisé pour les tâches de traitement et d'analyse. Chaque composant, comme les couches d'embedding, LSTM, et multi-head attention, est accompagné de ses principaux paramètres. Les dimensions des embeddings, la taille des couches cachées, le nombre de têtes pour l'attention multi-tête, et d'autres configurations essentielles sont clairement définis, offrant une compréhension approfondie de l'architecture du modèle. Ces spécifications assurent une mise en œuvre cohérente et optimisée pour répondre aux objectifs du projet.

Composant	Paramètre	Valeur
Embedding Layer	Taille du vocabulaire	<code>embedding_matrix.shape[0]</code>
	Dimension des embeddings	<code>embedding_matrix.shape[1]</code>
	Poids des embeddings	Pré-chargés depuis <code>embedding_matrix</code>
	Apprentissage des poids	Oui (<code>embedding.weight.requires_grad = True</code>)
LSTM Layer	Taille d'entrée	<code>embed_dim</code> (dimension des embeddings)
	Taille des couches cachées	128 (défini par <code>hidden_dim</code>)
	Nombre de couches	1
	Bidirectionnel	Oui
	Taille de sortie	<code>hidden_dim * 2</code> (256 pour bidirectionnel)
Multi-Head Attention	Taille d'entrée	<code>hidden_dim * 2</code> (256)
	Nombre de têtes	8
	Batch mode	Oui
Dropout Layer	Probabilité de dropout	0.5
Fully Connected Layer	Taille d'entrée	<code>hidden_dim * 2</code> (256)
	Taille de sortie	3 (défini par <code>output_size</code>)

3.3.3 Configuration des Paramètres d’Entrainement

Le tableau des paramètres d’entraînement fournit une vue d’ensemble des hyperparamètres essentiels utilisés pour l’apprentissage du modèle. Il inclut des informations sur la longueur maximale des séquences, la taille du batch, le nombre d’époques, ainsi que des détails sur le taux d’apprentissage et le clipping des gradients. Ces réglages permettent d’ajuster finement le comportement du modèle pendant l’entraînement, garantissant une convergence efficace et des performances optimales sur les données d’entraînement.

Table 3.4: Paramètres d'entraînement

Paramètre	Valeur
Longueur maximale des séquences (<code>MAX_LEN</code>)	167
Taille du batch (<code>BATCH_SIZE</code>)	32
Nombre de classes (<code>output_size</code>)	3
Device (CPU ou GPU)	GPU
Taux d'apprentissage (<code>lr</code>)	0.001
Décroissance des poids (<code>weight_decay</code>)	1×10^{-4}
Clipping des gradients	5
Nombre d'époques	20
Scheduler milestones	[6]
Facteur de réduction du scheduler	0.5

3.3.3.1 Entrainement du Modèle

Durant l'entraînement, le modèle a montré une amélioration progressive de ses performances, avec une réduction significative de la perte et une augmentation constante de la précision. Cela indique que le modèle a bien appris à partir des données d'entraînement, atteignant une précision très élevée au fil des époques.

Époque	Loss (train_loss)	Accuracy (train_accuracy)
1	0.8352	61.70%
10	0.0672	97.68%
20	0.0168	99.33%

Table 3.5: Performances d'entraînement à différentes époques.

3.3.3.2 Evaluation du Modèle

Sur les données de validation, la précision a initialement augmenté, mais elle a ensuite stagné malgré l'augmentation continue de la précision d'entraînement. De plus, la perte de validation a commencé à augmenter après les premières époques, ce qui suggère une possibilité de sur-apprentissage, où le modèle devient trop adapté aux données d'entraînement et généralise moins bien sur de nouvelles données.

Époque	Accuracy (val_accuracy)
1	71.49%
10	66.20%
20	66.31%

Table 3.6: Performances de validation à différentes époques.

3.4 GAN

3.4.1 Modèle GAN avec GPT2

3.4.1.1 Architecture de GPT-2

GPT-2 repose sur l'architecture Transformer, introduite dans l'article "*Attention is All You Need*". Cette architecture est basée sur un mécanisme appelé *self-attention*, qui permet au modèle de pondérer l'importance relative de chaque mot dans une séquence et de comprendre les relations contextuelles.

Les caractéristiques clés de GPT-2 incluent :

- **Transformers Unidirectionnels** : GPT-2 est un modèle autoregressif, ce qui signifie qu'il génère le texte mot par mot (ou token par token) en se basant uniquement sur les mots précédents.
- **Pré-entraînement** : Le modèle est pré-entraîné sur un vaste corpus de texte (plusieurs gigaoctets de données) pour apprendre les structures, les relations grammaticales, et les connaissances générales présentes dans le texte.
- **Fine-tuning** : GPT-2 peut être affiné sur des tâches spécifiques (e.g., résumé, traduction, génération de dialogues) pour améliorer ses performances dans des cas particuliers.

3.4.1.2 Fonctionnement de GPT-2

Le fonctionnement de GPT-2 repose sur un apprentissage non supervisé. Pendant le pré-entraînement, le modèle essaie de prédire le mot suivant dans une séquence donnée.

- **Entrée** : Une séquence de texte ou un prompt initial.
- **Traitement** : GPT-2 utilise ses couches de transformateurs pour analyser le contexte et générer des probabilités pour le mot suivant.
- **Sortie** : Un mot ou une séquence de mots est générée, ce processus est itératif jusqu'à ce qu'un texte complet soit produit.

3.4.1.3 À quoi sert GPT-2 ?

GPT-2 a été conçu pour de multiples applications, grâce à sa capacité à comprendre et générer un texte contextuellement pertinent. Voici quelques exemples d'utilisation :

- **Génération de contenu** : Créer automatiquement des articles, des histoires, ou des descriptions de produits.
- **Chatbots et assistants virtuels** : Améliorer les interactions conversationnelles en générant des réponses naturelles et pertinentes.
- **Résumé de texte** : Condenser des documents ou des articles en extrayant les informations clés.
- **Traduction automatique** : Traduire des phrases ou des textes d'une langue à une autre.
- **Recherche et analyse de texte** : Extraire des tendances, des opinions ou des émotions dans de grands volumes de données textuelles.
- **Éducation** : Fournir des explications ou des exercices automatisés dans différents domaines.

3.4.1.4 Utilisation de GPT-2 dans le projet

GPT-2 est utilisé dans ce projet pour générer des tweets simulés en complétant des invites (*prompts*) spécifiques. Ces tweets sont classés en trois catégories émotionnelles principales : positive, négative, et neutre. L'objectif est de simuler un éventail d'émotions et d'attitudes humaines à travers des tweets réalistes, ce qui permet d'étudier comment les émotions peuvent influencer la perception d'un message sur les réseaux sociaux.

Afin d'ajouter plus de réalisme à la génération des tweets, un fichier `.txt` a été intégré au projet. Ce fichier contient des centaines de prénoms issus de diverses cultures et contextes. Chaque tweet généré par GPT-2 est ainsi personnalisé en associant un prénom, ce qui permet de rendre les tweets plus authentiques et humains. De plus, nous avons également ajouté des informations supplémentaires pour augmenter la crédibilité des tweets simulés :

- **Le sexe** : Chaque utilisateur est assigné aléatoirement au genre "femme" ou "homme" afin de diversifier les profils.
- **L'âge** : L'âge de chaque utilisateur est généré de manière aléatoire, variant entre 18 et 72 ans, ce qui permet de simuler une large tranche d'âge dans les tweets générés.

- **Le timestamp** : Un horodatage est également ajouté à chaque tweet pour indiquer l'heure exacte de sa génération, créant ainsi une dimension temporelle pour chaque message simulé.

Ces éléments combinés permettent de rendre les tweets générés plus réalistes et contextuellement riches, donnant l'illusion que ces messages proviennent de personnes réelles, avec une identité et des caractéristiques propres.

Ce bloc en bas utilise le modèle GPT-2 pour générer du contenu textuel réaliste.

```
# Ignorer les avertissements
warnings.filterwarnings("ignore")

# Charger le modèle et le tokenizer GPT-2
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
model = GPT2LMHeadModel.from_pretrained("gpt2")

# Définir le pad_token si nécessaire
tokenizer.pad_token = tokenizer.eos_token # Définir eos_token comme pad_token

# Charger le fichier CSV avec pandas
df = pd.read_csv("Prenoms.csv", delimiter=";", encoding="ISO-8859-1")

# Extraire les colonnes des prénoms et genres
first_names = df["01_prenom"].dropna().tolist() # Liste des prénoms
genders = df["02_genre"].dropna().tolist() # Liste des genres

# Liste enrichie des types de tweets
tweet_prompts = {
    "positive": [
        "I really, really love this! Wow, just amazing.",
        "This is the best day of my life!",
        "Feeling so happy and blessed today!",
        "Wow, such an incredible moment! Can't stop smiling.",
        "I'm so proud of myself for achieving this!",
        "This song is amazing! On repeat all day.",
        "Life is beautiful, and today proved it again."
    ]
}
```

Figure 3.4: Architecture - Partie 1

```

    "negative": [
        "I hate this. Why does it always happen to me?",
        "Feeling so down today. Nothing is going right.",
        "What a terrible day. I just want it to end.",
        "Why does life have to be so unfair sometimes?",
        "Ugh, I can't stand this anymore. So frustrating.",
        "Everything feels hopeless right now.",
        "I feel like giving up. Nothing makes sense anymore.",
    ],
    "neutral": [
        "What a day. Nothing special, just the usual.",
        "Another day, another dollar. Same routine.",
        "Life goes on. Just living one day at a time.",
        "Hmm, not sure what to feel today. Neutral vibes.",
        "Nothing much to report. Just a regular day.",
        "The weather is fine. Nothing else to add.",
        "Quiet day at home. Just me and my thoughts.",
    ],
}
"""Génère des informations aléatoires, telles qu'un prénom, un âge
(entre 18 et 65 ans), et un genre (homme ou femme). Le résultat est
accompagné d'un timestamp."""

```

Figure 3.5: Architecture - Partie 2

```

# Fonction pour générer des informations aléatoires
# Liste pour stocker les données générées
generated_data = []

def generate_random_info():
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S") # Timestamp
    first_name = random.choice(first_names) # Prénom aléatoire
    gender = random.choice(genders) # Sexe aléatoire
    age = random.randint(18, 65) # Âge aléatoire

    # Ajuster l'affichage du sexe
    gender_display = "Homme" if gender.lower() == "m" else "Femme" # Assurer un affichage correct du sexe
    return timestamp, first_name, age, gender_display
gpt2_model = GPT2LMHeadModel.from_pretrained("gpt2")
gpt2_model.eval() # Passer en mode évaluation pour GPT-2

```

Figure 3.6: Architecture - Partie 3

3.4.2 Consumer et Producer avec KafkaPy

3.4.2.1 KafkaPy

C'est une bibliothèque Python utilisée pour interagir avec Apache Kafka, une plateforme de diffusion de messages distribuée qui permet de traiter de grandes quantités de données en temps réel. Kafka

est souvent utilisé pour la gestion de flux de données en continu, en particulier dans les systèmes qui nécessitent une transmission rapide et fiable de messages, comme les applications de streaming, l'analyse en temps réel, ou encore l'intégration de systèmes distribués.

3.4.2.2 À quoi sert KafkaPy ?

KafkaPy sert de client Kafka pour Python, permettant aux développeurs d'interagir facilement avec un cluster Kafka. Cette bibliothèque permet de publier (produire) et de consommer (consommer) des messages sur des topics Kafka, tout en offrant des fonctionnalités comme la gestion des erreurs et la possibilité de garantir l'ordre des messages dans une application distribuée.

KafkaPy est particulièrement utile dans les cas suivants :

- **Streaming en temps réel** : Pour traiter et transmettre des flux de données en temps réel, comme les logs d'événements, les transactions financières, ou les messages générés par des capteurs IoT.
- **Architecture distribuée** : Kafka permet de relier plusieurs services ou applications à travers une architecture distribuée, où les différents composants peuvent publier et consommer des messages de manière décentralisée.
- **Mise en place de pipelines de données** : Kafka est largement utilisé pour construire des pipelines de données, permettant de déplacer des données entre des applications sans perturber leur fonctionnement. KafkaPy facilite l'intégration de Python dans ces architectures de données distribuées.

3.4.2.3 Fonctionnalités principales de KafkaPy :

- **Production de messages (Producer)** : Permet d'envoyer des messages vers un topic Kafka.
- **Consommation de messages (Consumer)** : Permet de récupérer des messages à partir d'un ou plusieurs topics Kafka.
- **Gestion des partitions** : Kafka divise les messages en partitions, et KafkaPy permet de gérer la lecture et l'écriture dans ces partitions de manière fluide.
- **Gestion des erreurs** : KafkaPy intègre des mécanismes pour gérer les erreurs liées à la connectivité ou à la consommation/production de messages, garantissant ainsi la fiabilité des systèmes.

3.4.2.4 Fonctionnalités de KafkaPy dans ce projet:

Le système de gestion des flux de données repose sur KafkaPy pour produire et consommer les tweets générés par le modèle GPT-2.

- **Producteur :** Génère des tweets simulés en boucle, les enrichit avec des informations aléatoires, et les insère dans une file d'attente partagée. Le producteur introduit un délai aléatoire entre les générations pour simuler un flux réaliste.

Le code ci-dessus illustre la fonction `producer`:

```
def producer(tweet_queue):
    for i in range(1000):
        # Choisir un type de tweet au hasard
        emotion = random.choice(list(tweet_prompts.keys()))
        prompt = random.choice(tweet_prompts[emotion]) # Sélectionner une phrase aléatoire dans la catégorie

        tweet = ""
        max_retries = 5 # Nombre maximal d'essais pour générer un tweet complet
        retries = 0
        while retries < max_retries:
            retries += 1
            # Convertir le texte de base en entrée pour le modèle GPT-2
            inputs = tokenizer(prompt, return_tensors="pt", padding=True, truncation=True)

            # Ajouter le `attention_mask` et `pad_token_id` pour éviter les avertissements
            attention_mask = inputs["attention_mask"]
            pad_token_id = tokenizer.eos_token_id # Définir le pad_token_id sur eos_token_id

            # Générer un tweet avec GPT-2
            outputs = gpt2_model.generate(
                inputs["input_ids"],
                max_new_tokens=30, # Nombre maximum de nouveaux tokens à générer
                num_return_sequences=1,
                temperature=0.7,
                top_k=50,
                top_p=0.9,
                do_sample=True,
```

Figure 3.7: Producer: Partie1

```

        top_k=50,
        top_p=0.9,
        do_sample=True,
        attention_mask=attention_mask,
        pad_token_id=pad_token_id,
        no_repeat_ngram_size=2
    )
    # Décoder le tweet généré
    tweet = tokenizer.decode(outputs[0], skip_special_tokens=True)

    # Vérifier si le tweet semble coupé
    if not tweet.endswith((".", "!", "?")):
        # Si coupé, ajouter plus de contenu
        prompt = tweet.strip()
    else:
        break # Tweet complet généré

    # Générer des informations aléatoires
    timestamp, first_name, age, gender = generate_random_info()

    # Ajouter ces informations à la fin du tweet
    tweet_with_info = f"{tweet.strip()} - Name: {first_name}, Age: {age}, Gender: {gender}, Time: {timestamp}"

    # Mettre le tweet dans la file d'attente
    tweet_queue.put(tweet_with_info)
    # Ajouter seulement le tweet sans les informations supplémentaires
    tweet_clean = tweet.strip() # Suppression des informations supplémentaires
    emotion_prediction = test_model_with_sentence(model, tweet_clean, word_to_index, device)

```

Figure 3.8: Producer: Partie2

```

emotion_prediction = test_model_with_sentence(model, tweet_clean, word_to_index, device)
# Ajouter ces données dans la liste avec la prédiction
generated_data.append({
    "name": first_name,
    "age": age,
    "tweet": tweet_clean, # Utilisation du tweet propre
    "timestamp": timestamp,
    "gender": gender,
    "prediction": emotion_prediction # Ajout de la colonne de prédiction
})
time.sleep(random.uniform(0.5, 1.5)) # Ajouter un délai aléatoire entre la génération des tweets

```

Figure 3.9: Producer: Partie3

- **Sélection aléatoire d'une catégorie émotionnelle :** Une émotion est choisie de manière aléatoire parmi les catégories disponibles, et un prompt correspondant est sélectionné.
- **Génération des tweets :** Le modèle GPT-2 génère des tweets à partir du prompt, en appliquant des paramètres tels que la température (`temperature`), le filtre de probabilité (`top_p`) et le nombre maximal de tokens (`max_new_tokens`). Un mécanisme de vérification garantit que les tweets ne sont pas coupés.
- **Ajout d'informations contextuelles :** Chaque tweet est enrichi avec des informations aléatoires comme le prénom, l'âge, le genre et un horodatage.

- **Prédiction des émotions** : Le texte du tweet est analysé par un modèle BiLSTM pour prédire l'émotion associée.
- **Gestion des délais** : Un délai aléatoire est introduit entre chaque génération pour simuler un flux en temps réel.

Cette architecture permet de générer et de traiter des tweets de manière réaliste, tout en assurant une gestion fluide des données dans un contexte distribué.

- **Consommateur** : Récupère les tweets de la file d'attente, extrait le texte principal, et passe les données au modèle BiLSTM pour prédire l'émotion associée au tweet. Ce bloc assure que chaque message est traité individuellement avant de marquer son traitement comme terminé.

```
def consumer(tweet_queue, model, word_to_index, device):
    while True:
        # Attendre qu'un tweet soit disponible dans la file d'attente
        tweet = tweet_queue.get()
        # Extraire uniquement la phrase avant le délimiteur "- Name:"
        if " - Name:" in tweet:
            sentence = tweet.split(" - Name:")[0].strip()
        else:
            sentence = tweet # Si le format attendu n'est pas trouvé, utiliser le tweet complet
        # Analyser la phrase avec le modèle BiLSTM
        print(f"Phrase reçue pour analyse : {sentence}")
        # Prédire l'émotion de la phrase
        test_model_with_sentence(model, sentence, word_to_index, device)
        # Indiquer que le tweet a été consommé
        tweet_queue.task_done()
```

Figure 3.10: Consumer

Le code en haut représente la fonction `consumer` traite les tweets fournis par le producteur via une file d'attente:

- **Récupération du tweet** : La fonction attend qu'un élément soit disponible dans la file d'attente `tweet_queue`.
- **Extraction du texte principal** :
 - Si le tweet contient le délimiteur "`- Name:`", seule la partie avant ce délimiteur est extraite.
 - Sinon, le tweet complet est utilisé.

Cette étape garantit que seules les informations pertinentes du tweet sont analysées.

- **Analyse émotionnelle** :

- Le texte extrait est passé au modèle BiLSTM via la fonction `test_model_with_sentence`, qui prédit l’émotion associée au tweet.
- Le modèle utilise un dictionnaire d’indexation des mots (`word_to_index`) et le dispositif matériel (`device`) pour effectuer la prédiction.
- **Marquage du traitement :** Une fois l’analyse terminée, la fonction appelle `tweet_queue.task_done()` pour indiquer que le tweet a été consommé avec succès.

La boucle `while True` permet à la fonction de fonctionner en continu, garantissant un traitement fluide des tweets à mesure qu’ils sont produits. En cas de format inattendu, le tweet complet est analysé, assurant une gestion robuste des données. Ce mécanisme de gestion des données permet de simuler un pipeline temps réel où les données sont constamment générées, transmises et analysées.

Conclusion

Ce chapitre a présenté l’implémentation et l’évaluation de plusieurs architectures de modèles d’apprentissage profond, à savoir les CNN (Convolutional Neural Networks), RNN (Recurrent Neural Networks), LSTM (Long Short-Term Memory), et GAN (Generative Adversarial Networks). Chaque modèle a été exploré en fonction de sa spécificité et de son domaine d’application.

Les résultats obtenus à travers ces modèles ont révélé des performances de base, mais ils ont également mis en évidence des pistes d’amélioration pour optimiser les architectures et les techniques d’entraînement. L’évaluation des performances à l’aide de métriques telles que la précision et la perte a fourni des informations cruciales pour identifier les modèles les plus adaptés aux tâches spécifiques.

En conclusion, bien que ces modèles aient montré des capacités intéressantes dans leurs domaines respectifs, leur performance peut être améliorée par des techniques d’optimisation supplémentaires, des ajustements d’architecture et un affinement des processus d’entraînement. Le chapitre ouvre ainsi la voie à de futures améliorations, en particulier en ce qui concerne la régularisation, la gestion de l’overfitting et l’exploration de nouvelles architectures pour des performances accrues.

Chapitre 4

Intégration de kafka pour la Prédiction en Temps Réel

Introduction

Dans cette partie, nous avons intégré des outils de Big Data et de traitement en temps réel pour améliorer notre système de prédiction des sentiments. Nous avons utilisé Apache Kafka, une plateforme de messagerie distribuée, pour collecter, traiter et analyser les flux de données en temps réel. Kafka permet de gérer des volumes massifs de données avec une faible latence, ce qui est crucial pour les applications nécessitant une analyse en temps réel, comme la prédiction des sentiments à partir de textes.

4.1 Objectifs

L'objectif de cette section est de traiter des données textuelles en temps réel afin de prédire les sentiments (positifs, négatifs, ou neutres) des utilisateurs sur la base de leurs messages ou commentaires. Nous avons intégré Kafka pour :

- Collecter des flux de données, dans notre projet on a utilisé le dataset qui contient l'ensemble des commentaires.
- Utiliser des modèles de deep learning pour analyser ces données en temps réel.
- Afficher des résultats de prédiction instantanés.

4.2 Prédiction des Sentiments en Temps Réel avec Kafka et RNN

L'objectif de cette partie est de prédire les sentiments des textes reçus en temps réel (par exemple, des commentaires ou avis des utilisateurs) à l'aide de Kafka, d'un modèle RNN et d'une architecture de traitement en temps réel.

4.2.1 Workflow des prédictions

4.2.1.1 Collecte des Données avec Kafka

Producteur Kafka (KafkaProducer) : Il envoie des messages contenant des textes à analyser vers un topic Kafka spécifique. Chaque message est constitué d'un texte (par exemple, un avis ou un commentaire) sous forme JSON.

Topic Kafka : Les données textuelles sont envoyées à un topic Kafka. Par exemple, un topic nommé `text_topic`. Le producteur envoie ces données en temps réel, à chaque fois qu'un nouveau commentaire est publié ou qu'un utilisateur interagit.

Topic Kafka : Les données textuelles sont envoyées à un topic Kafka. Par exemple, un topic nommé `text_topic`. Le producteur envoie ces données en temps réel, à chaque fois qu'un nouveau commentaire est publié ou qu'un utilisateur interagit.

4.2.1.2 Prétraitement des Textes

Normalisation : Chaque texte est transformé en minuscules et débarrassé des signes de ponctuation pour préparer les données à la tokenisation.

4.2.1.3 Chargement des Ressources

Vocabulaire : Le vocabulaire nécessaire à la tokenisation est chargé depuis un fichier externe `vocab.pkl`, ce qui permet de convertir les mots du texte en indices numériques.

Matrice d'Embedding GloVe : La matrice d'embedding pré-entraînée est chargée à partir d'un fichier `embedding_matrix.pth` pour effectuer la représentation vectorielle des mots.

Modèle RNN (GRU) : Un modèle de type RNN (GRU bidirectionnel) est créé, avec des poids pré-entraînés chargés à partir de `rnn_model.pth`. Ce modèle sera utilisé pour prédire les sentiments.

4.2.1.4 Prédiction des Sentiments

Tokenisation et Transformation : Le texte est tokenisé (divisé en mots) et chaque mot est transformé en un indice numérique basé sur le vocabulaire.

Passage au Modèle RNN : Le texte, sous forme de séquence d'indices, est passé dans le modèle RNN pour obtenir des logits (sorties brutes).

Conversion en Probabilités : Les logits sont convertis en probabilités via la fonction `softmax`, permettant de prédire la classe de sentiment la plus probable (positif, négatif ou neutre).

4.2.1.5 Affichage des Résultats

Classe Prédite : La classe prédite est affichée (Sentiment positif, négatif ou neutre).

4.2.1.6 Étape 6 : Envoi des Résultats au Kafka Producer

Envoi au Producer Kafka : Les résultats (le texte analysé et sa classe prédite) sont envoyés à un autre topic Kafka pour un traitement ultérieur ou un stockage.

4.2.2 Configuration de l'Infrastructure Kafka et Zookeeper pour l'Analyse de Textes

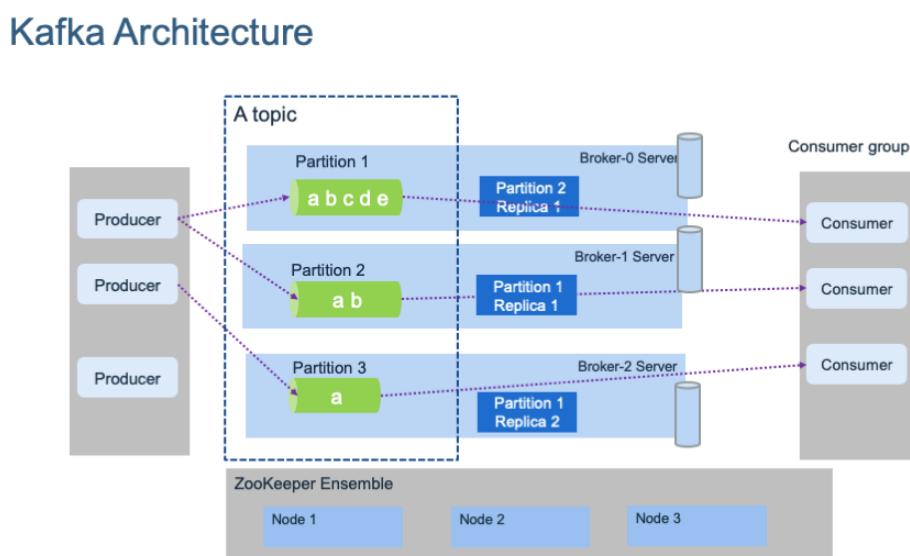


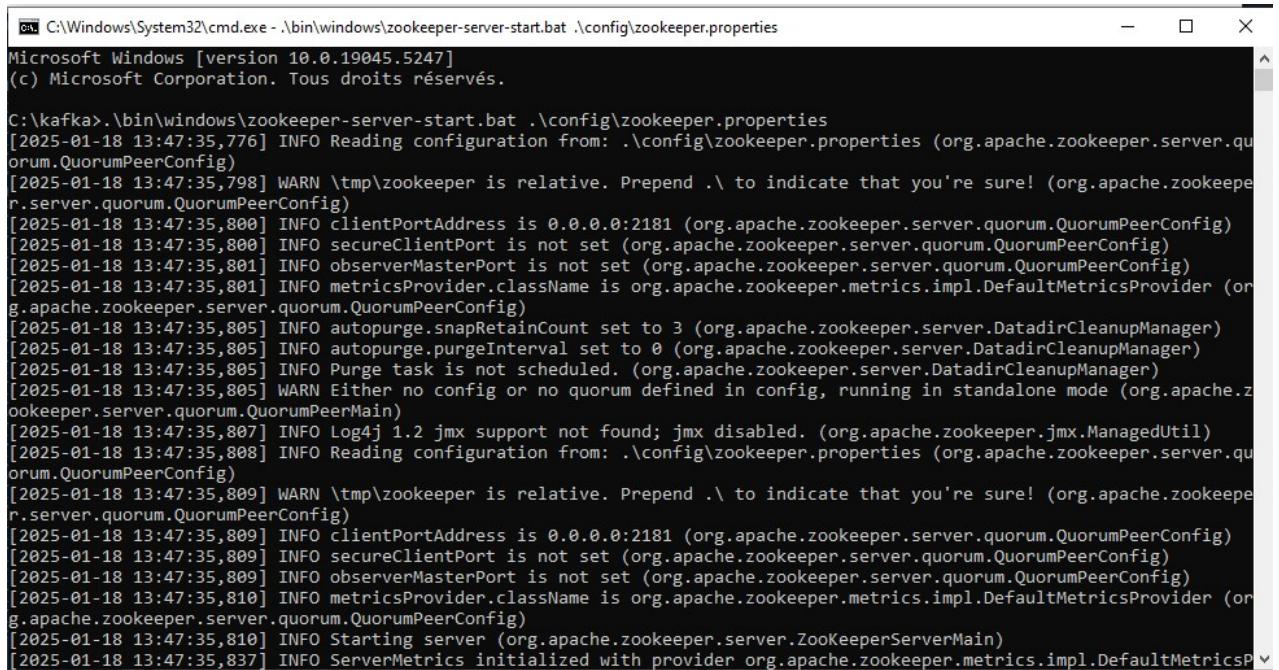
Figure 4.1: Architecture Kafka

4.2.2.1 Démarrer Zookepper

Zookeeper est nécessaire pour la gestion des métadonnées et la coordination des services Kafka.

- Dans un terminal on accède au répertoire Kafka.
- On exécute la commande suivante pour démarrer Zookeeper :

```
.bin/zookeeper-server-start.sh config/zookeeper.properties
```



The screenshot shows a Windows Command Prompt window with the title 'C:\Windows\System32\cmd.exe - .\bin\windows\zookeeper-server-start.bat \config\zookeeper.properties'. The window displays the logs from the Zookeeper server start command. The logs include various INFO messages about reading configuration files, setting port addresses, and starting the server. There are also some WARN messages indicating relative paths and missing configuration settings. The log ends with 'INFO ServerMetrics initialized with provider org.apache.zookeeper.metrics.impl.DefaultMetricsProvider'.

```
C:\Windows\System32\cmd.exe - .\bin\windows\zookeeper-server-start.bat \config\zookeeper.properties
Microsoft Windows [version 10.0.19045.5247]
(c) Microsoft Corporation. Tous droits réservés.

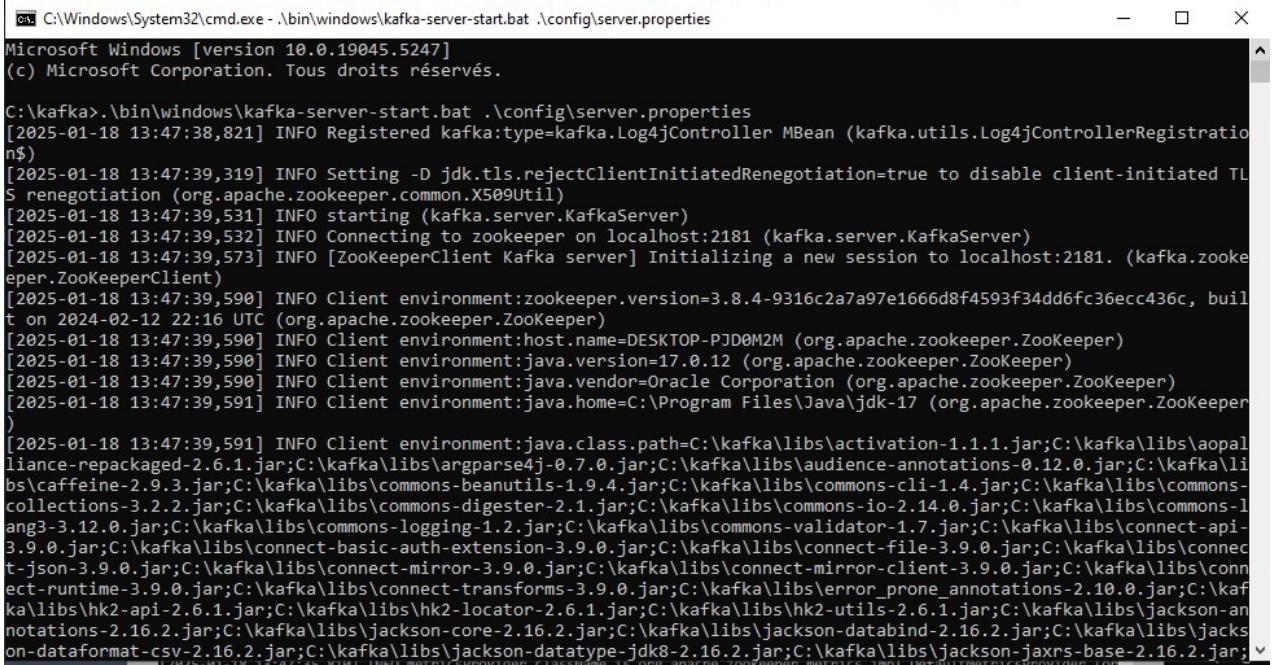
C:\kafka>.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties
[2025-01-18 13:47:35,776] INFO Reading configuration from: ./config/zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2025-01-18 13:47:35,798] WARN \tmp\zookeeper is relative. Prepend ./ to indicate that you're sure! (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2025-01-18 13:47:35,800] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2025-01-18 13:47:35,800] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2025-01-18 13:47:35,801] INFO observerMasterPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2025-01-18 13:47:35,801] INFO metricsProvider.className is org.apache.zookeeper.metrics.impl.DefaultMetricsProvider (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2025-01-18 13:47:35,805] INFO autopurge.snapRetainCount set to 3 (org.apache.zookeeper.server.DatadirCleanupManager)
[2025-01-18 13:47:35,805] INFO autopurge.purgeInterval set to 0 (org.apache.zookeeper.server.DatadirCleanupManager)
[2025-01-18 13:47:35,805] INFO Purge task is not scheduled. (org.apache.zookeeper.server.DatadirCleanupManager)
[2025-01-18 13:47:35,805] WARN Either no config or no quorum defined in config, running in standalone mode (org.apache.zookeeper.server.quorum.QuorumPeerMain)
[2025-01-18 13:47:35,807] INFO Log4j 1.2 jmx support not found; jmx disabled. (org.apache.zookeeper.jmx.ManagedUtil)
[2025-01-18 13:47:35,808] INFO Reading configuration from: ./config/zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2025-01-18 13:47:35,809] WARN \tmp\zookeeper is relative. Prepend ./ to indicate that you're sure! (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2025-01-18 13:47:35,809] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2025-01-18 13:47:35,809] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2025-01-18 13:47:35,809] INFO observerMasterPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2025-01-18 13:47:35,810] INFO metricsProvider.className is org.apache.zookeeper.metrics.impl.DefaultMetricsProvider (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2025-01-18 13:47:35,810] INFO Starting server (org.apache.zookeeper.server.ZooKeeperServerMain)
[2025-01-18 13:47:35,837] INFO ServerMetrics initialized with provider org.apache.zookeeper.metrics.impl.DefaultMetricsProvider
```

Figure 4.2: Démarrage de Zookeeper

4.2.2.2 Démarrer kafka

- Dans un terminal, on accède au répertoire Kafka
- On exécute la commande suivante pour démarrer Kafka :

```
.bin/kafka-server-start.sh config/server.properties
```



```
C:\Windows\System32\cmd.exe - .\bin\windows\kafka-server-start.bat .\config\server.properties
Microsoft Windows [version 10.0.19045.5247]
(c) Microsoft Corporation. Tous droits réservés.

C:\kafka>.\bin\windows\kafka-server-start.bat .\config\server.properties
[2025-01-18 13:47:38,821] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistration$)
[2025-01-18 13:47:39,319] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TLS renegotiation (org.apache.zookeeper.common.X509Util)
[2025-01-18 13:47:39,531] INFO starting (kafka.server.KafkaServer)
[2025-01-18 13:47:39,532] INFO Connecting to zookeeper on localhost:2181 (kafka.server.KafkaServer)
[2025-01-18 13:47:39,573] INFO [ZooKeeperClient Kafka server] Initializing a new session to localhost:2181. (kafka.zookeeper.ZooKeeperClient)
[2025-01-18 13:47:39,590] INFO Client environment:zookeeper.version=3.8.4-9316c2a7a97e1666d8f4593f34dd6fc36ecc436c, built on 2024-02-12 22:16 UTC (org.apache.zookeeper.ZooKeeper)
[2025-01-18 13:47:39,590] INFO Client environment:host.name=DESKTOP-PJD0M2M (org.apache.zookeeper.ZooKeeper)
[2025-01-18 13:47:39,590] INFO Client environment:java.version=17.0.12 (org.apache.zookeeper.ZooKeeper)
[2025-01-18 13:47:39,590] INFO Client environment:java.vendor=Oracle Corporation (org.apache.zookeeper.ZooKeeper)
[2025-01-18 13:47:39,591] INFO Client environment:java.home=C:\Program Files\Java\jdk-17 (org.apache.zookeeper.ZooKeeper)
[2025-01-18 13:47:39,591] INFO Client environment:java.class.path=C:\kafka\libs\activation-1.1.1.jar;C:\kafka\libs\aaliance-repackaged-2.6.1.jar;C:\kafka\libs\argparse4j-0.7.0.jar;C:\kafka\libs\audience-annotations-0.12.0.jar;C:\kafka\libs\caffeine-2.9.3.jar;C:\kafka\libs\commons-beanutils-1.9.4.jar;C:\kafka\libs\commons-cli-1.4.jar;C:\kafka\libs\commons-collections-3.2.2.jar;C:\kafka\libs\commons-digester-2.1.jar;C:\kafka\libs\commons-io-2.14.0.jar;C:\kafka\libs\commons-lang3-3.12.0.jar;C:\kafka\libs\commons-logging-1.2.jar;C:\kafka\libs\commons-validator-1.7.jar;C:\kafka\libs\connect-api-3.9.0.jar;C:\kafka\libs\connect-basic-auth-extension-3.9.0.jar;C:\kafka\libs\connect-file-3.9.0.jar;C:\kafka\libs\connect-json-3.9.0.jar;C:\kafka\libs\connect-mirror-3.9.0.jar;C:\kafka\libs\connect-mirror-client-3.9.0.jar;C:\kafka\libs\connect-runtime-3.9.0.jar;C:\kafka\libs\connect-transforms-3.9.0.jar;C:\kafka\libs\error_prone_annotations-2.10.0.jar;C:\kafka\libs\hk2-api-2.6.1.jar;C:\kafka\libs\hk2-locator-2.6.1.jar;C:\kafka\libs\hk2-utils-2.6.1.jar;C:\kafka\libs\jackson-annotations-2.16.2.jar;C:\kafka\libs\jackson-core-2.16.2.jar;C:\kafka\libs\jackson-databind-2.16.2.jar;C:\kafka\libs\jackson-dataformat-csv-2.16.2.jar;C:\kafka\libs\jackson-datatype-jdk8-2.16.2.jar;C:\kafka\libs\jackson-jaxrs-base-2.16.2.jar;▼
```

Figure 4.3: Démarrage de Kafka

4.2.2.3 Crée un Topic kafka

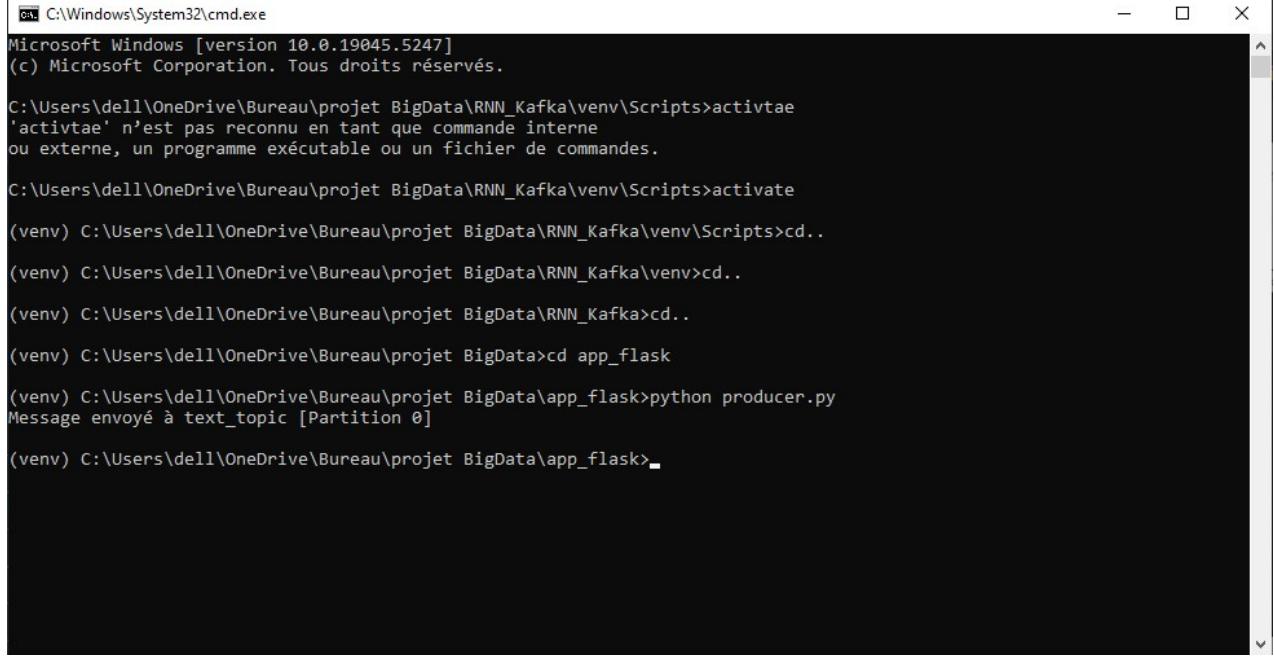
Dans un terminal, on exécute la commande suivante pour créer un topic nommé text_topic :

```
.\bin\windows\kafka-topics.bat --create --topic text_topic --bootstrap-server localhost:9092
```

4.2.2.4 Lancer le producer

Une fois que Kafka est opérationnel, vous pouvez démarrer le producteur pour envoyer des messages au topic text_topic

- Dans un terminal, on exécute le script producer.py pour commencer à envoyer des messages :



```
C:\Windows\System32\cmd.exe
Microsoft Windows [version 10.0.19045.5247]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\dell\OneDrive\Bureau\projet BigData\RNN_Kafka\venv\Scripts>activitae
'activitae' n'est pas reconnu en tant que commande interne
ou externe, un programme exécutable ou un fichier de commandes.

C:\Users\dell\OneDrive\Bureau\projet BigData\RNN_Kafka\venv\Scripts>activate
(venv) C:\Users\dell\OneDrive\Bureau\projet BigData\RNN_Kafka\venv\Scripts>cd..
(venv) C:\Users\dell\OneDrive\Bureau\projet BigData\RNN_Kafka\venv>cd..
(venv) C:\Users\dell\OneDrive\Bureau\projet BigData\RNN_Kafka>cd..
(venv) C:\Users\dell\OneDrive\Bureau\projet BigData>cd app_flask
(venv) C:\Users\dell\OneDrive\Bureau\projet BigData\app_flask>python producer.py
Message envoyé à text_topic [Partition 0]
(venv) C:\Users\dell\OneDrive\Bureau\projet BigData\app_flask>
```

Figure 4.4: Lancer le producer

- Le producteur enverra des messages au format JSON contenant des textes à analyser vers le topic text_topic.

4.2.2.5 Lancer le consumer

Enfin, on démarre le consommateur pour consommer les messages en temps réel à partir du topic text_topic et effectuer des prédictions sur le sentiment des textes.

- Dans un terminal, exécutez le script consumer.py pour commencer à consommer les messages et effectuer les prédictions:

```
C:\Windows\System32\cmd.exe - python consumer.py
Microsoft Windows [version 10.0.19045.5247]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\dell\OneDrive\Bureau\projet BigData\RNN_Kafka\venv\Scripts>activate
(venv) C:\Users\dell\OneDrive\Bureau\projet BigData\RNN_Kafka\venv\Scripts>cd..
(venv) C:\Users\dell\OneDrive\Bureau\projet BigData\RNN_Kafka\venv>cd..
(venv) C:\Users\dell\OneDrive\Bureau\projet BigData\RNN_Kafka>cd..
(venv) C:\Users\dell\OneDrive\Bureau\projet BigData>cd app_flask

(venv) C:\Users\dell\OneDrive\Bureau\projet BigData\app_flask>python consumer.py
Indices pour le texte 'life is better': [0, 0, 0]
Logits: tensor([[-5.8767, -9.8101, 17.0232]])
Probabilités : tensor([[1.1342e-10, 2.2206e-12, 1.0000e+00]])
Texte: life is better
Classe prédite: 2
```

Figure 4.5: Lancer le consumer

- Le consommateur traitera chaque message, effectuera une prédition sur le sentiment du texte et affichera le texte avec la classe prédite (positif, négatif ou neutre).

Conclusion

En suivant ces étapes, vous avez configuré avec succès Kafka et Zookeeper, des composants essentiels pour la gestion et l'échange de données en temps réel dans un environnement de Big Data. Ce workflow, basé sur Kafka, permet une communication fluide entre les producteurs et les consommateurs, et assure une gestion efficace des flux de données. Grâce à cette infrastructure, vous pouvez désormais traiter et analyser les messages de manière rapide et scalable. De plus, des tests ont été réalisés en utilisant des modèles de machine learning, ce qui permet d'effectuer des prédictions sur les sentiments des textes en temps réel. Ce processus ouvre ainsi la voie à des applications avancées dans des environnements Big Data, en offrant une analyse intelligente et continue des flux de données.

Chapitre 5

Ingénierie des connaissances

Introduction

L'objectif principal de ce système est de permettre l'extraction, l'organisation et la gestion des connaissances relatives aux sentiments à partir de sources textuelles et de bases de données. En s'appuyant sur des techniques avancées telles que les ontologies, les graphes de connaissances, les formalismes logiques, les réseaux sémantiques, et le traitement automatique du langage naturel (NLP), le système vise à offrir des solutions intelligentes pour l'analyse des tendances et des sentiments en ligne.

Pour y parvenir, plusieurs axes stratégiques sont développés :

- **Formalisation des concepts et ontologies** : Création d'une représentation structurée des concepts liés aux émotions, événements économiques, secteurs d'activité, et produits, permettant d'organiser efficacement les connaissances.
- **Renforcement de l'analyse des sentiments** : Enrichissement des modèles d'analyse pour capter les nuances des émotions et leurs relations avec des contextes spécifiques.
- **Identification des indicateurs prédictifs** : Développement de modèles permettant de relier les changements de sentiments aux tendances de marché, offrant une vision prédictive des comportements consommateurs.
- **Automatisation de l'apprentissage des modèles** : Mise en place d'un système d'apprentissage continu, basé sur des règles et des connaissances, pour s'adapter aux évolutions des marchés et des réseaux sociaux.

La démarche repose sur quatre grandes étapes :

1. **Conception d'une ontologie des tendances de marché et des sentiments** : Organisation des concepts clés et des relations, intégrant des outils comme Protégé pour formaliser une base de connaissances robuste.
2. **Extraction et enrichissement des concepts avec des données en temps réel** : Intégration des nouvelles tendances et expressions issues des réseaux sociaux, actualisées en fonction des événements marquants.
3. **Développement d'une base de règles d'inférence et de raisonnement automatisé** : Mise en place d'un moteur de règles pour interpréter les sentiments dans des contextes précis et prédire les impacts possibles.
4. **Système d'aide à la décision et recommandations** : Fourniture de recommandations et d'analyses contextualisées via un tableau de bord, facilitant les prises de décision stratégiques.

Ce système vise à devenir un outil clé pour comprendre et anticiper les dynamiques émotionnelles et comportementales, tout en aidant les décideurs à agir rapidement et efficacement sur les tendances détectées.

5.1 Conception d'une ontologie des tendances de marché et des sentiments

5.1.1 Création des Classes

La première étape consistait à définir les classes principales qui structurent notre ontologie. Ces classes représentent les concepts fondamentaux :

- **Tweet** : Représente une publication sur Twitter.
- **Country** : Représente le pays associé au tweet.
- **Sentiment** : Définit les types de sentiments (positif, négatif, neutre).
- **Time of Tweet** : Définit le temps de la tweet (mooring, noon, night).
- **Age Group** : Définit les distributions d'âge des utilisateurs.

The screenshot shows the Protege ontology editor interface. The title bar reads "TweetSentimentOntology (http://www.semanticweb.org/hp/ontologies/TweetSentimen...)" with standard window controls. The menu bar includes File, Edit, View, Reasoner, Tools, Refactor, Window, and Help. The toolbar has icons for back, forward, search, and other functions. The main navigation bar shows "Country" as the active ontology. Below it, tabs for Active ontology, Entities, Individuals by class, and DL Query are visible. On the left, a sidebar displays the class hierarchy under "owl:Thing": "Country" is selected and highlighted in blue. Other classes listed are AgeGroup, TimeOfTweet, Sentiment, and Tweet. There are buttons for creating new individuals, annotation properties, datatypes, and data properties. The main workspace is titled "Annotations: Country" and contains sections for Annotations (+), Description (+), General class axioms (+), SubClass Of (Anonymous Ancestor), and Instances (+). A note at the bottom says "To use the reasoner click Reasoner > Start reasoner" with a checkbox for "Show Inferences".

Figure 5.1: Cr ation des classes de l'ontologie

5.1.2 Cr ation des sous Classes

Apr s avoir d fini les classes principales qui structurent notre ontologie, on va ensuite d finir les sous classes.

The screenshot shows the Protege ontology editor interface. The top menu bar includes File, Edit, View, Reasoner, Tools, Refactor, Window, and Help. The title bar displays "TweetSentimentOntology (http://www.semanticweb.org/hp/ontologies/TweetSentimen...)".

The main window has tabs for Active ontology, Entities, Individuals by class, and DL Query. The Entities tab is selected. The left sidebar shows a tree view of the ontology structure under the "Classes" tab, with nodes like owl:Thing, Country, AgeGroup, 70-100, 60-70, 46-60, 31-45, 21-30, 0-20, TimeOfTweet, Night, Noon, Morning, Sentiment, Neutral, Negative, Positive, and Tweet. The node "70-100" is currently selected and highlighted in blue.

The right panel displays the properties and annotations for the selected class "70-100". It includes tabs for Annotations and Usage, with the Annotations tab selected. The "Annotations" section contains a table with one row labeled "Annotations: 70-100". The "Description" section contains a table with one row labeled "Description: 70-100". Below these sections are buttons for Equivalent To, SubClass Of, General class axioms, Instances, and Target for Key, each accompanied by a plus sign icon to add new annotations or descriptions.

Figure 5.2: Création des sous classes de l'ontologie

5.1.3 Crédation des Propriétés

Les relations entre les classes ont été définies à l'aide de deux types de propriétés : les propriétés d'objet et les propriétés de données.

5.1.4 Propriétés d'Objet

- **hasCountry** : Relie un Tweet à un Country.
- **hasSentiment** : Relie un Tweet à un Sentiment.
- **hasAgeGroup** : Relie un Tweet à un AgeGroup.
- **hasTimeOfTweet** : Relie un Tweet à un TimeOfTweet.

5.1.5 Propriétés de Données

- **hasText** : Stocke le contenu textuel du tweet (type : `xsd:string`).
- **hasSentimentt** : Stocke le contenu textuel du sentiment du tweet (type : `xsd:string`).
- **hasCountry** : Stocke le contenu textuel du pays (type : `xsd:string`).

5.1.5.1 Ajout des Individus

Les individus (instances) ont été ajoutés pour représenter des données concrètes. Par exemple :

- Un individu `Tweet1` avec :
 - **hasText** : "Sooo sad I will miss San Diego".
 - **hasSentiment** : `negative`.
 - **hasCountry** : USA.

5.1.6 Validation avec le Raisonneur Hermit

Pour garantir la cohérence de l'ontologie, le raisonneur Hermit a été utilisé :

- Vérification des relations entre les classes et les propriétés.
- Détection des incohérences dans les définitions ou les données.

Les tests ont confirmé que l'ontologie est valide et que toutes les relations définies sont correctes.

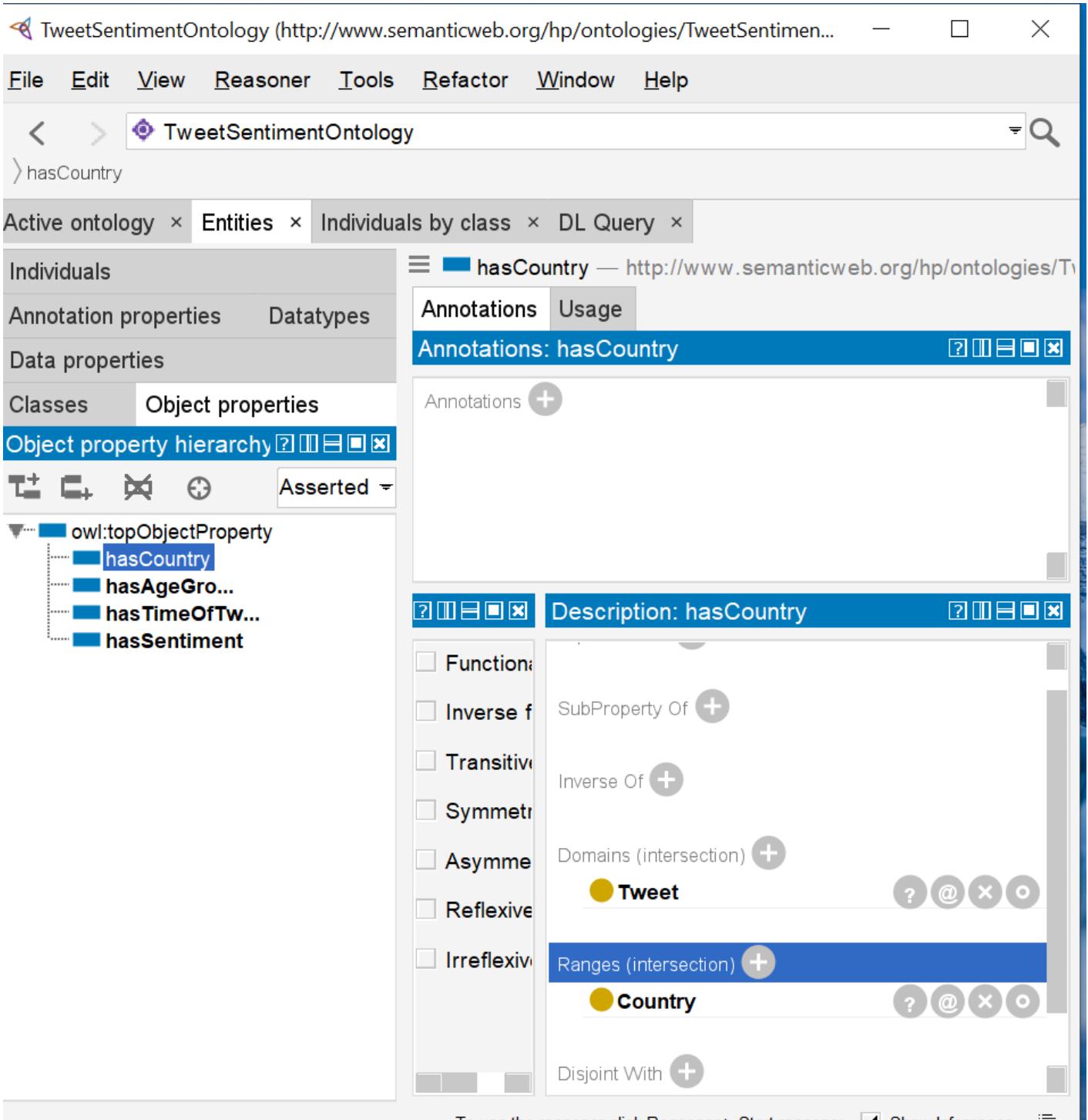


Figure 5.3: Création des propriétés de l'ontologie

5.1.7 Visualisation du Graphe

Le graphe de l'ontologie a été généré à l'aide de l'outil de visualisation intégré de Protégé. Cette représentation graphique permet de visualiser :

- Les relations entre les classes (ex. Tweet est relié à Country via hasCountry).

- Les liens entre les individus et leurs propriétés.

Un exemple de graphe est illustré dans la figure 5.4.

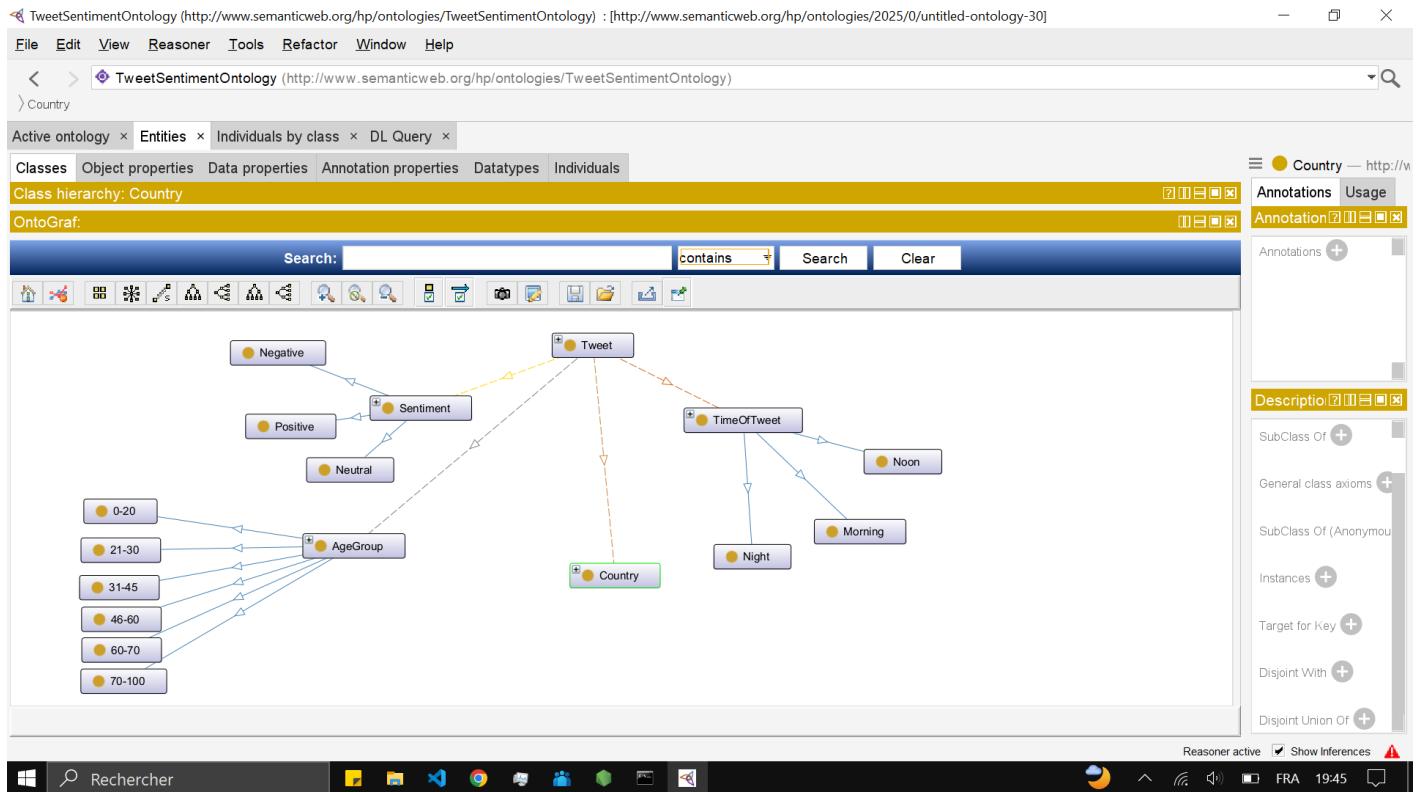


Figure 5.4: Graphe de l'ontologie

5.1.8 Exportation de l'Ontologie

L'ontologie a été sauvegardée au format RDF/XML, qui est adapté pour l'interopérabilité avec d'autres outils et pour une réutilisation ultérieure.

5.2 Extraction et Enrichissement des Concepts avec des Données en Temps Réel

L'extraction et l'enrichissement des concepts à partir de données en temps réel, grâce aux techniques avancées de traitement du langage naturel, permettent de maintenir une ontologie vivante et réactive aux évolutions rapides des tendances et événements mondiaux. Ce processus d'actualisation continue assure que le système peut adapter ses réponses, ses prédictions, ou ses recommandations en fonction du contexte actuel, offrant ainsi des résultats plus pertinents et plus précis dans un environnement dynamique et en constante évolution.

un processus combine plusieurs étapes pour générer du texte, extraire des entités et des relations, et mettre à jour une ontologie avec ces informations:

5.2.1 Intégration de GPT-2 pour Génération de Textes

Dans cette étape, le modèle GPT-2 est utilisé pour générer des textes basés sur des invites spécifiques. Le texte généré peut être utilisé pour des tâches ultérieures, comme l'extraction d'entités ou la mise à jour d'une ontologie.

Fonction generate_texts_gpt2(prompts, model, tokenizer) : Prend une liste d'invites, génère des textes en utilisant le modèle GPT-2 et renvoie les textes générés sous forme de liste

```
# **Étape 1 : Intégration de GPT-2 pour Génération de Textes**
def generate_texts_gpt2(prompts, model, tokenizer, max_length=50, num_return_sequences=1):
    generated_texts = []
    for prompt in prompts:
        inputs = tokenizer.encode(prompt, return_tensors="pt")
        outputs = model.generate(
            inputs, max_length=max_length, num_return_sequences=num_return_sequences, pad_token_id=tokenizer.eos_token_id
        )
        generated_texts.extend([tokenizer.decode(output, skip_special_tokens=True) for output in outputs])
    return generated_texts
```

Figure 5.5: Intégration de GPT-2

5.2.2 Analyse et Extraction d'Entités et de Relations

Une fois que le texte est généré, on utilise spaCy pour extraire des entités nommées et des relations entre les mots. Les entités peuvent être des personnes, des lieux, des dates, etc., et les relations représentent les liens entre les entités dans le texte (par exemple, "A est le président de B").

Fonction extract_entities_and_relations(texts) : Extrait les entités nommées et les relations en utilisant les outils NLP de spaCy. Elle renvoie une liste d'entités uniques et une liste de relations identifiées dans le texte.

```

# **Étape 2 : Analyse et Extraction d'Entités et de Relations**
def extract_entities_and_relations(texts):
    nlp = spacy.load("en_core_web_sm")
    entities = defaultdict(list)
    relations = []

    for text in texts:
        doc = nlp(text)
        # Extraction des entités
        for ent in doc.ents:
            entities[ent.label_].append(ent.text)

        # Extraction des relations en utilisant les dépendances
        for token in doc:
            if token.ent_type_ and token.head.ent_type_:
                relation = {
                    "Source": token.head.text,
                    "Source_Type": token.head.ent_type_,
                    "Target": token.text,
                    "Target_Type": token.ent_type_,
                    "Relation": token.dep_
                }
                relations.append(relation)

    # Supprimer les doublons
    unique_entities = {key: list(set(value)) for key, value in entities.items()}
    return unique_entities, relations

```

Figure 5.6: Extraction des entités

5.2.3 Mise à jour de l’Ontologie avec les Relations

Une ontologie est un ensemble organisé de concepts, d’entités et de relations dans un domaine donné.

Cette étape met à jour une ontologie existante avec les nouvelles entités et relations extraites.

Fonction update_ontology_with_relations: Met à jour l’ontologie existante en ajoutant les entités et relations extraites. Cette fonction retourne l’ontologie mise à jour.

```

# **Étape 3 : Mise à jour de l'Ontologie avec les Relations**
def update_ontology_with_relations(existing_ontology, new_entities, new_relations):
    updated_ontology = existing_ontology.copy()

    # Mise à jour des entités
    for entity_type, entities in new_entities.items():
        if entity_type not in updated_ontology.get("Entities", {}):
            updated_ontology["Entities"][entity_type] = []
        updated_ontology["Entities"][entity_type].extend(entities)

    # Mise à jour des relations
    if "Relations" not in updated_ontology:
        updated_ontology["Relations"] = []
    updated_ontology["Relations"].extend(new_relations)

    return updated_ontology

```

Figure 5.7: Mise à jour des entités

5.2.4 Sauvegarder l'ontologie

La fonction sauvegarder_ontologie prend un graphe (représentant une ontologie) et le sauvegarde dans un fichier OWL. Par défaut, le fichier est nommé "ontologie_mise_a_jour.owl", mais il est possible de spécifier un autre nom. Le graphe est sérialisé au format XML et enregistré dans le fichier indiqué. Une fois l'enregistrement effectué, un message est imprimé pour confirmer l'emplacement du fichier.

```
def sauvegarder_ontologie(graph, file_name="ontologie_mise_a_jour.owl"):
    graph.serialize(destination=file_name, format="xml")
    print(f"\nOntologie sauvegardée dans le fichier : {file_name}")
```

Figure 5.8: Sauvegarder l'ontologie

5.2.5 Fonction principale pour la mise à jour automatique

La fonction `mise_a_jour_ontologie` effectue plusieurs étapes clés :

1. **Chargement de GPT-2** : Le modèle GPT-2 et son tokenizer sont chargés pour générer du texte à partir de plusieurs invites prédéfinies.
2. **Génération de textes** : À partir des invites fournies, le modèle GPT-2 génère des textes qui peuvent inclure des informations pertinentes sur des sujets comme les élections.
3. **Extraction des entités et relations** : Les textes générés sont traités pour extraire des entités (personnes, lieux, événements) et des relations (liens entre les entités). Une fonction dédiée est utilisée pour effectuer cette extraction.
4. **Chargement et mise à jour de l'ontologie RDF** : L'ontologie RDF est chargée à partir d'un fichier OWL existant. Si le fichier est chargé avec succès, il est mis à jour avec les entités et relations extraites.
5. **Affichage des triplets RDF** : Après la mise à jour, les triplets RDF (sujets, prédicats, objets) contenus dans l'ontologie sont affichés.
6. **Sauvegarde de l'ontologie mise à jour** : L'ontologie mise à jour est ensuite sauvegardée dans un fichier au format XML (OWL).
7. **Gestion des erreurs** : Si une erreur survient pendant le processus (chargement du modèle, extraction, mise à jour de l'ontologie), elle est capturée et un message d'erreur est affiché.

```

# **Étape 5 : Fonction principale pour la mise à jour automatique**
def mise_a_jour_ontologie():
    try:
        print(f"\n[Mise à jour - {datetime.now()}] Début de la mise à jour de l'ontologie.")

        # Charger GPT-2
        tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
        model = GPT2LMHeadModel.from_pretrained("gpt2")
        tokenizer.pad_token = tokenizer.eos_token
        model.eval()

        # Générer des textes
        prompts = [
            "Breaking news in the elections today!",
            "The elections are heating up with new developments.",
            "Voters are showing strong emotions during this election period."
        ]
        generated_texts = generate_texts_gpt2(prompts, model, tokenizer)

        # Extraire les entités et relations
        extracted_entities, extracted_relations = extract_entities_and_relations(generated_texts)
        print("\n==== Entités extraites ===")
        print(extracted_entities)
        print("\n==== Relations extraites ===")
        for relation in extracted_relations:
            print(relation)

```

Figure 5.9: mise à jour automatique (partie 1)

```

# Charger et mettre à jour l'ontologie RDF
ontologie_file = r"E:/IID3/Big project/ontologie.owl"
graph = rdflib.Graph()
if not graph.parse(ontologie_file, format="xml"):
    print(f"\nErreur : Impossible de charger le fichier d'ontologie {ontologie_file}")
else:
    # Mettre à jour l'ontologie
    graph = update_rdf_ontology(graph, extracted_entities, extracted_relations)

    # Afficher les triplets
    afficher_triplets(graph)

    # Sauvegarder l'ontologie mise à jour
    sauvegarder_ontologie(graph)
    print(f"\n[Mise à jour - {datetime.now()}] Mise à jour terminée.")
except Exception as e:
    print(f"\nErreur lors de la mise à jour de l'ontologie : {e}")

```

Figure 5.10: mise à jour automatique (partie 2)

5.2.6 Planifier et Exécuter la tâche avec Schedule

Le code planifie l'exécution de la fonction `mise_a_jour_ontologie` toutes les secondes à l'aide du module `schedule`. Une boucle infinie (`while True`) est utilisée pour vérifier et exécuter les tâches planifiées, avec une pause d'une seconde entre chaque vérification.

```

# **Étape 6 : Planifier la tâche avec Schedule**
schedule.every(1).seconds.do(mise_a_jour_ontologie)

# **Étape 7 : Boucle pour exécuter la tâche**
print("\nDémarrage de la mise à jour automatique de l'ontologie...")
while True:
    schedule.run_pending()
    time.sleep(1)

```

Figure 5.11: Planification et Exécution de la tâche

5.2.7 Résultats de l'Extraction

```

Démarrage de la mise à jour automatique de l'ontologie...

[Mise à jour - 2025-01-19 00:48:43.891626] Début de la mise à jour de l'ontologie.
The attention mask is not set and cannot be inferred from input because pad token is same as eos token. As a consequence, you may observe unexpected behavior
so pass your input's 'attention_mask' to obtain reliable results.

*** Entités extraites ***
{'DATE': ['today'], 'ORDINAL': ['first'], 'GPE': ['the United States'], 'PERSON': ['David B. Johnson'], 'ORG': ['the University of California, Berkeley']}

*** Relations extraites ***
{'Source': 'States', 'Source_Type': 'GPE', 'Target': 'the', 'Target_Type': 'GPE', 'Relation': 'det'}
{'Source': 'States', 'Source_Type': 'GPE', 'Target': 'United', 'Target_Type': 'GPE', 'Relation': 'compound'}
{'Source': 'States', 'Source_Type': 'GPE', 'Target': 'the', 'Target_Type': 'GPE', 'Relation': 'det'}
{'Source': 'States', 'Source_Type': 'GPE', 'Target': 'United', 'Target_Type': 'GPE', 'Relation': 'compound'}
{'Source': 'first', 'Source_Type': 'ORDINAL', 'Target': 'first', 'Target_Type': 'ORDINAL', 'Relation': 'ROOT'}
{'Source': 'Johnson', 'Source_Type': 'PERSON', 'Target': 'David', 'Target_Type': 'PERSON', 'Relation': 'compound'}
{'Source': 'Johnson', 'Source_Type': 'PERSON', 'Target': 'B.', 'Target_Type': 'PERSON', 'Relation': 'compound'}
{'Source': 'University', 'Source_Type': 'ORG', 'Target': 'the', 'Target_Type': 'ORG', 'Relation': 'det'}
{'Source': 'University', 'Source_Type': 'ORG', 'Target': 'of', 'Target_Type': 'ORG', 'Relation': 'prep'}
{'Source': 'of', 'Source_Type': 'ORG', 'Target': 'California', 'Target_Type': 'ORG', 'Relation': 'pobj'}
{'Source': 'California', 'Source_Type': 'ORG', 'Target': ',', 'Target_Type': 'ORG', 'Relation': 'punct'}
{'Source': 'University', 'Source_Type': 'ORG', 'Target': 'Berkeley', 'Target_Type': 'ORG', 'Relation': 'npadvmod'}
Entité déjà présente : today
Entité déjà présente : first
Entité déjà présente : the United States
Entité déjà présente : David B. Johnson
Entité déjà présente : the University of California, Berkeley
Relation déjà présente : States -> det -> the
Relation déjà présente : States -> compound -> United

```

Figure 5.12: Résultat

5.3 Les règles d'inférence et création des requêtes d'inférence

Dans cette section, nous présentons les règles d'inférence appliquées à l'ontologie et les requêtes SPARQL utilisées pour en déduire des informations supplémentaires à partir du modèle RDF.

5.3.1 Règles d'inférence

Les règles d'inférence permettent de déduire de nouvelles informations à partir de l'ontologie en fonction des axiomes et des relations spécifiées. Par exemple, la règle suivante stipule qu'un Tweet doit être de type Sentiment :

```

# Règle 1 : Tout tweet (Tweet) doit être de type "Sentiment"
(ns:Tweet ?x) -> (?x rdf:type ns:Sentiment).

```

Voici une sélection de règles d'inférence :

- Règle 1 : Un Tweet doit être de type "Sentiment".
- Règle 2 : Si un Tweet a un texte (`aTexte`), il doit être de type `String`.
- Règle 3 : Un Tweet avec un texte (`aTexte`) doit avoir une date (`aDate`).
- Règle 4 : Les individus de type "Positif" doivent être de type "Positif".
- Règle 5 : Un individu de type "Negatif" doit avoir un texte d'au moins 5 caractères.

```

@include <OWLMicro>.
@prefix ns: <urn:webprotege:ontology:b4268d0d-8fb4-4517-98b7-21dd4cc99442#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.

# Règle 1 : Tout tweet (Tweet) doit être de type "Sentiment"
(ns:Tweet ?x) -> (?x rdf:type ns:Sentiment).

# Règle 2 : Si un Tweet a un texte (aTexte), alors il doit être de type string
(ns:Tweet ?x), (?x ns:aTexte ?text) -> (?text rdf:type xsd:string).

# Règle 3 : Un Tweet avec un texte (aTexte) doit avoir une date (aDate)
(ns:Tweet ?x), (?x ns:aTexte ?text) -> (?x ns:aDate ?date), (?date rdf:type xsd:dateTime).

# Règle 4 : Les individus de type "Positif" (SentimentPositif1) doivent être de type "Positif"
(ns:SentimentPositif1 ?x) -> (?x rdf:type ns:Positif).

# Règle 5 : Un individu de type "Negatif" doit avoir un texte d'au moins 5 caractères
(ns:Negatif ?x), (?x ns:aTexte ?text) -> (?text length ?len), (?len >= 5), (?x rdf:type ns:Negatif).

```

Figure 5.13: Les règles d'inférence

5.3.2 Requêtes d'inférence

Les requêtes SPARQL permettent de récupérer les triples de données déduits en appliquant les règles d'inférence sur le modèle RDF. Voici un exemple de requête SPARQL pour récupérer tous les triples du modèle RDF et d'autres requêtes peuvent être utilisées pour récupérer des informations spécifiques, comme les types des tweets ou les tweets associés à un sentiment particulier ,et aussi un extrait de l'implémentation Java utilisée pour exécuter ces requêtes:

/ontologyDataset

query add data edit info

SPARQL Query

To try out some SPARQL queries against the selected dataset, enter your query here.

Example Queries Selection of triples Selection of classes

SPARQL Endpoint /ontologyDataset/sparql Content Type (SELECT) JSON Content Type (GRAPH) Turtle

```

PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX ex: <http://webprotege.stanford.edu/>
SELECT ?subject ?predicate ?object
WHERE { ?subject ?predicate ?object }

```

Table Response 44 results in 1.659 seconds Simple view Ellipse Filter query results Page size: 50

subject	predicate	object
1<urn:webprotege:ontology:b4268d0d-8fb4-4517-98b7-21dd4c...	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2002/07/owl#Ontology>
2<http://webprotege.stanford.edu/RDmsl2as5U4IEUbqvio68d>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2002/07/owl#ObjectProperty>
3<http://webprotege.stanford.edu/RDmsl2as5U4IEUbqvio68d>	<http://www.w3.org/2000/01/rdf-schema#subPropertyOf>	<http://www.w3.org/2002/07/owl#topObjectProperty>
4<http://webprotege.stanford.edu/RDmsl2as5U4IEUbqvio68d>	<http://www.w3.org/2000/01/rdf-schema#domain>	<http://webprotege.stanford.edu/RB2zWZPar7ZT9e6CDhWhPx>
5<http://webprotege.stanford.edu/RDmsl2as5U4IEUbqvio68d>	<http://www.w3.org/2000/01/rdf-schema#range>	<http://webprotege.stanford.edu/Rx5Gy5drAecm4y856rE7>
6<http://webprotege.stanford.edu/RB2zWZPar7ZT9e6CDhWhPx>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2002/07/owl#Class>

Figure 5.14: Jena application

```

// Query 1: Retrieve all triples
String queryString1 =
    "PREFIX ex: <http://webprotege.stanford.edu/> " +
    "SELECT ?subject ?predicate ?object " +
    "WHERE { ?subject ?predicate ?object }";
System.out.println("Executing Query 1: Retrieve all triples");
executeQuery(queryString1, infModel);

// Query 2: Retrieve specific inferred triples based on the new rule
String queryString2 =
    "PREFIX ex: <http://webprotege.stanford.edu/> " +
    "SELECT ?subject ?type " +
    "WHERE { ?subject ex:hasType ?type }";
System.out.println("Executing Query 2: Retrieve inferred types");
executeQuery(queryString2, infModel);

// Query 3: Retrieve all subjects of type Positif
String queryString3 =
    "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " + // Ajouter le préfixe RDF
    "PREFIX ex: <http://webprotege.stanford.edu/> " +
    "SELECT ?subject " +
    "WHERE { ?subject rdf:type ex:Positif }"; // Utilisation de rdf:type
System.out.println("Executing Query 3: Retrieve all subjects of type Positif");
executeQuery(queryString3, infModel);

// Query 4: Retrieve all objects associated with a specific subject
String queryString4 =
    "PREFIX ex: <http://webprotege.stanford.edu/> " +
    "SELECT ?object " +
    "WHERE { ex:Tweet1 ?predicate ?object }";
System.out.println("Executing Query 4: Retrieve all objects related to ex:Tweet1");
executeQuery(queryString4, infModel);

```

Figure 5.15: les Requêtes d'inférence

```

import org.apache.jena.query.ResultSet;
import org.apache.jena.query.ResultSetFormatter;
import org.apache.jena.rdf.model.InfModel;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.rdf.model.ModelFactory;
import org.apache.jena.reasoner.rulesys.Rule;
import org.apache.jena.util.FileManager;
import org.apache.jena.reasoner.Reasoner;
import org.apache.jena.reasoner.rulesys.GenericRuleReasoner;

public class OntologyReasoning {
    public static void main(String[] args) {
        // Path to OWL file
        String ontologyFile = "Data/ontologie.owl";

        // Load ontology into a model
        Model model = FileManager.get().loadModel(ontologyFile);

        // Define inference rules
        String rules = "Data/rules.txt";

        // Load rules and create a Reasoner
        Reasoner reasoner = new GenericRuleReasoner(Rule.parseRules(rules));
        reasoner.setDerivationLogging(true);

        // Set the Reasoner to the model
        InfModel infModel = ModelFactory.createInfModel(reasoner, model);

        // Query 1: Retrieve all triples
        String queryString1 =
            "PREFIX ex: <http://webprotege.stanford.edu/> " +
            "SELECT ?subject ?predicate ?object " +
            "WHERE { ?subject ?predicate ?object }";
        System.out.println("Executing Query 1: Retrieve all triples");
        executeQuery(queryString1, infModel);
    }

    private static void executeQuery(String queryString, InfModel infModel) {
        Query query = QueryFactory.create(queryString);
        try (QueryExecution qe = QueryExecutionFactory.create(query, infModel)) {
            ResultSet results = qe.execSelect();
            ResultSetFormatter.out(System.out, results, query);
        }
    }
}

```

Figure 5.16: implémentation Java

Conclusion

En conclusion, l'ingénierie des connaissances appliquée à l'analyse des sentiments permet de structurer et d'exploiter les informations issues des données textuelles de manière efficace et intelligente. La création

de l'ontologie, l'enrichissement des concepts à travers des données en temps réel, ainsi que l'intégration de techniques avancées comme le traitement du langage naturel et les règles d'inférence, contribuent à l'élaboration d'un système dynamique et réactif. Ce système offre non seulement une vue détaillée des tendances de marché et des émotions exprimées dans les réseaux sociaux, mais aussi la capacité d'adapter ses prédictions et recommandations en fonction des évolutions constantes de l'environnement. En combinant ces approches, ce modèle fournit des outils puissants pour l'aide à la décision, renforçant ainsi l'efficacité des stratégies de gestion et d'analyse des comportements des consommateurs.

Chapitre 6

Réalisation du Projet

Introduction

Dans ce chapitre, nous détaillons la conception et la mise en œuvre des interfaces de visualisation pour l'analyse des sentiments. Nous expliquerons comment ces interfaces permettent de représenter les résultats sous forme de graphiques interactifs et de cartes géographiques, facilitant ainsi la compréhension des tendances et des distributions des sentiments.

6.1 Tableau de Bord Statique

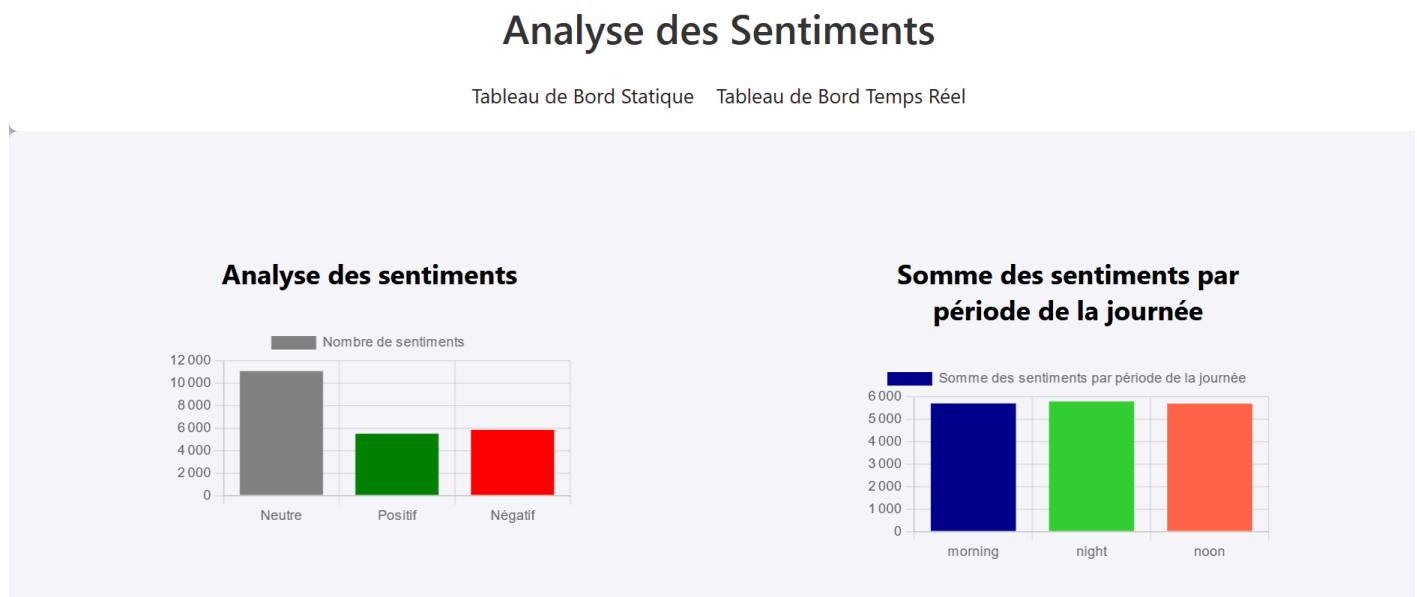


Figure 6.1: Graphes de Visualisation (partie 1)

- **Graphe 1 Analyse des sentiments :** Ce graphique à barres offre une représentation visuelle de l'analyse des sentiments, divisée en trois catégories principales : neutre, positif, et négatif. Chaque catégorie est représentée par une barre distincte, colorée respectivement en gris, vert et rouge, permettant une distinction claire et intuitive entre les types de sentiments.

L'objectif du graphique est de mettre en évidence la répartition des sentiments analysés, en illustrant combien d'occurrences de chaque type ont été identifiées. Par exemple, une barre plus haute dans la catégorie "Positif" indiquerait que les sentiments positifs prédominent dans les données analysées.

Situé au centre de la page, le graphique est accompagné d'un titre explicite, "Analyse des sentiments", et d'une légende qui guide la lecture des données. Ce visuel permet aux utilisateurs d'évaluer rapidement les tendances globales et de comparer les différents types de sentiments en un coup d'œil.

- **Graphe 2: Somme des sentiments par période de la journée :**

Ce dashboard présente un graphique en barres illustrant la somme des sentiments (positifs, négatifs, neutres) des tweets répartis par période de la journée (matin, midi, soir). Chaque période est représentée par une barre colorée (bleu pour la nuit, vert pour le matin, rouge pour le midi), permettant de comparer l'intensité des sentiments à différents moments de la journée. Il aide à analyser les comportements émotionnels des utilisateurs sur Twitter, offrant des insights pour ajuster les stratégies de communication, marketing et gestion des médias sociaux.

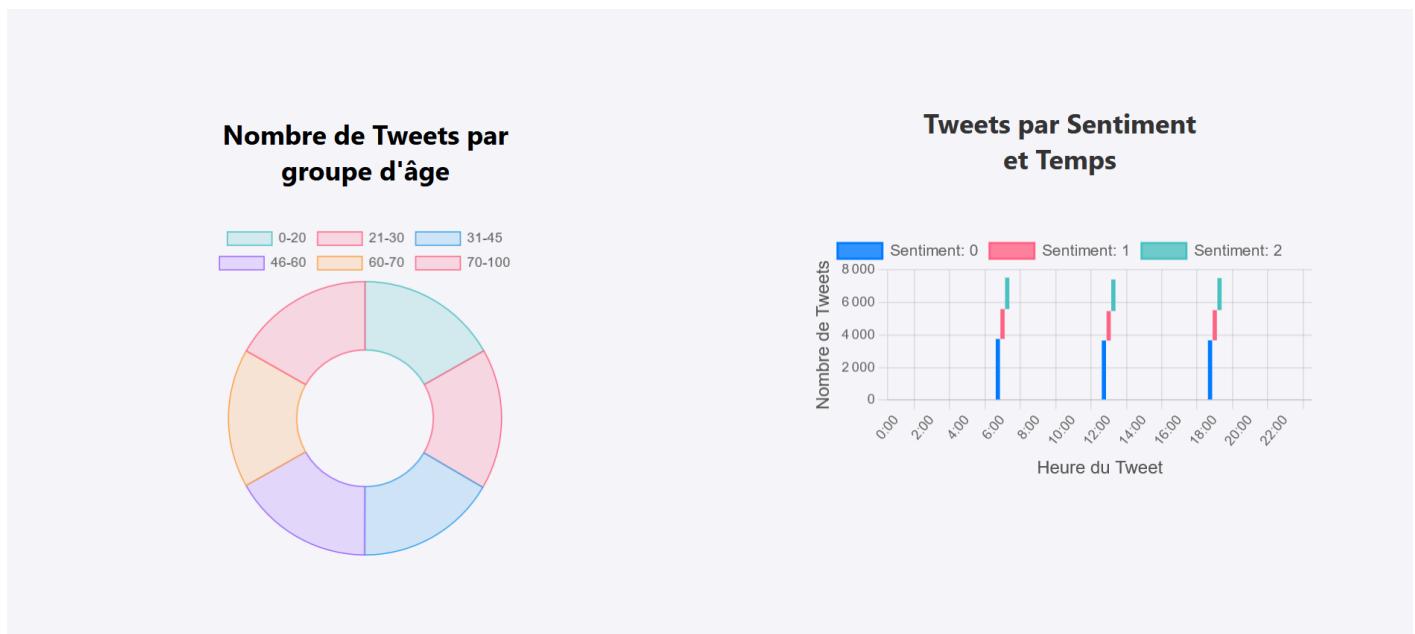


Figure 6.2: Graphes de Visualisation (partie 2)

- **Graphe 3: Nombre de tweets par groupe d'âge :**

Ce Dashboard Doughnut affiche le nombre de tweets par groupe d'âge. Il utilise un graphique circulaire pour visualiser la répartition des tweets en fonction des groupes d'âge spécifiques, offrant ainsi une vue d'ensemble claire et colorée des données démographiques des utilisateurs.

Tranches d'âge :

- **0-20 ans** : Les utilisateurs de ce groupe sont représentés par une couleur verte, indiquant une concentration de tweets parmi les jeunes.
- **21-30 ans** : Les utilisateurs dans cette tranche d'âge sont colorés en rose, soulignant un engagement important dans cette catégorie d'âge.
- **46-60 ans** : Ce groupe est marqué par la couleur violette, représentant les utilisateurs plus âgés ayant un impact notable sur la plateforme.
- **60-70 ans** : Les utilisateurs dans cette tranche d'âge sont identifiés par une couleur orange, donnant une idée claire de l'activité dans les tranches plus âgées.
- **Graphe 4 : Tweets par Sentiment et Temps :**

Le graphique présenté est un histogramme qui illustre la répartition des tweets selon leurs sentiments (positif, négatif, ou neutre) et l'heure à laquelle ils ont été postés au cours de la journée. Le graphique est construit à partir des données recueillies via l'API backend.

- **Axe des X (Heure du Tweet)** : Cet axe représente les heures de la journée, allant de 0:00 à 23:00. Chaque barre du graphique correspond à une heure spécifique de la journée.
- **Axe des Y (Nombre de Tweets)** : L'axe vertical affiche le nombre de tweets publiés pour chaque heure, exprimé en nombre absolu de tweets.
- **Sentiments** : Le graphique est divisé en différentes couleurs pour représenter les différents types de sentiments des tweets :
 - * *Positif* : Représenté par une couleur bleu clair
 - * *Négatif* : Représenté par une couleur rouge foncé
 - * *Neutre* : Représenté par une couleur orange

Ces couleurs sont utilisées pour distinguer les différentes émotions exprimées dans les tweets.

La légende du graphique permet de lier chaque couleur à un sentiment spécifique.

L'objectif de ce graphique est de fournir une analyse visuelle de l'activité sur Twitter en fonction des sentiments des utilisateurs et de la période de la journée. En observant ce graphique, il est possible de déterminer à quelles heures de la journée un sentiment particulier est plus souvent exprimé, offrant ainsi un aperçu des tendances temporelles des conversations sur Twitter.

Détails supplémentaires :

- Les tweets sont regroupés en fonction de leur sentiment et de l'heure à laquelle ils ont été postés.
- Le graphique utilise un empilement des barres pour chaque sentiment, ce qui permet de visualiser facilement les comparaisons entre les différents sentiments à chaque heure.
- Les heures de la journée sont représentées de 0:00 à 23:00, où chaque barre représente le nombre de tweets dans une tranche horaire spécifique.

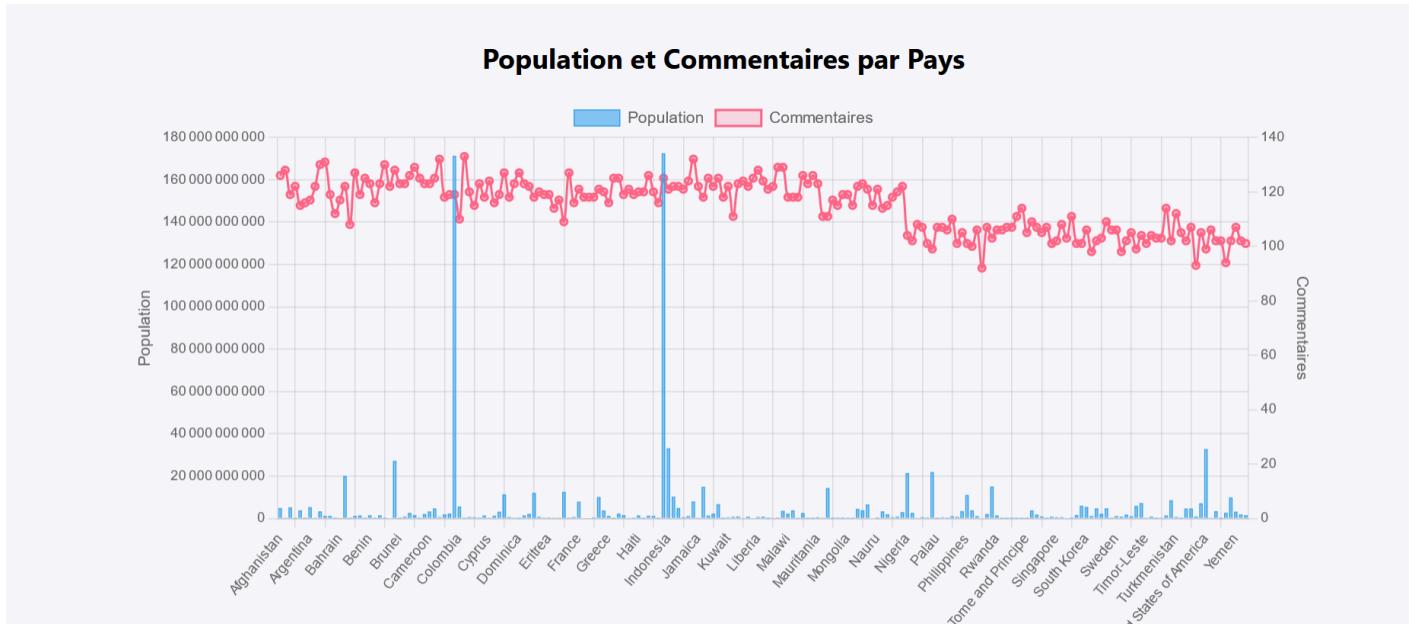


Figure 6.3: Population et Commentaires par Pays

• Graphe 5 : Population et Commentaires par Pays :

Ce graphique représente un tableau de bord affichant les données relatives à la population et aux commentaires (ou sentiments) pour différents pays. Il combine deux types de visualisations : un graphique à barres et un graphique en ligne. Le **graphique à barres** montre la population de chaque pays, tandis que le **graphique en ligne** représente le nombre total de commentaires ou de sentiments pour chaque pays. **Caractéristiques principales :**

- **Axe des X :** Liste des pays.

- **Axe Y principal (à gauche)** : Affiche la population de chaque pays, représentée par des barres.
- **Axe Y secondaire (à droite)** : Affiche le nombre total de commentaires (ou sentiments) pour chaque pays, représenté par un graphique en ligne.
- **Légende** : Permet de différencier la population (barres) et les commentaires (courbe).
- **Titre** : "Statistiques : Population et Commentaires par Pays".

Ce graphique permet de visualiser la relation entre la taille de la population et le nombre de commentaires pour chaque pays. Par exemple, les pays avec une population plus importante peuvent être corrélés à un plus grand nombre de commentaires, mais le graphique permet d'identifier clairement les tendances ou anomalies.

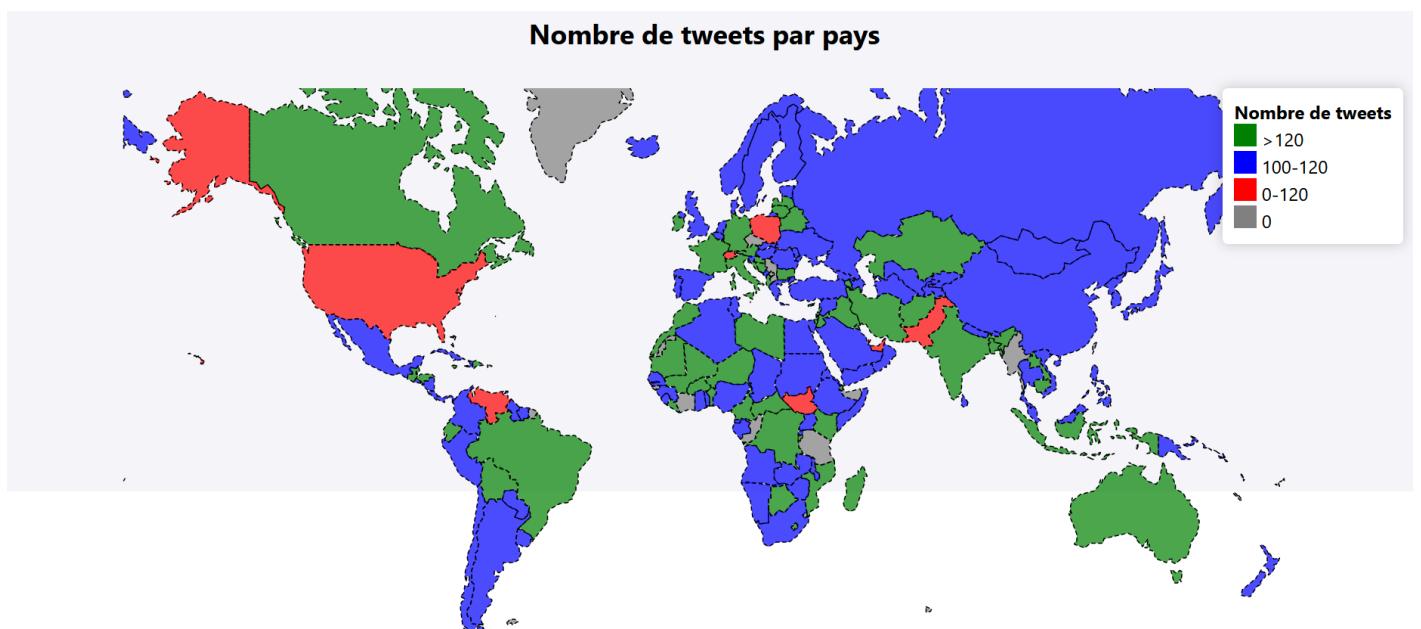


Figure 6.4: Carte Interactive des Tweets par Pays

- **Graphe 6 Carte Interactive des Tweets par Pays** : Cette carte interactive affiche le nombre de tweets par pays en utilisant des données géographiques et statistiques. Chaque pays est coloré en fonction de la quantité de tweets associés :
 1. **Vert** : Plus de 120 tweets
 2. **Bleu** : Entre 100 et 120 tweets
 3. **Rouge** : Entre 1 et 100 tweets
 4. **Gris** : Aucun tweet enregistré

En survolant un pays, une infobulle apparaît pour indiquer son nom et le nombre exact de tweets. En cliquant sur un pays, une fenêtre contextuelle affiche les mêmes informations. Une légende intuitive située en haut à droite facilite la lecture des données.

Cette carte permet de visualiser rapidement et efficacement la distribution des tweets dans le monde entier, mettant en évidence les zones les plus actives sur les réseaux sociaux.

6.2 Tableau de Bord en Temps Réel

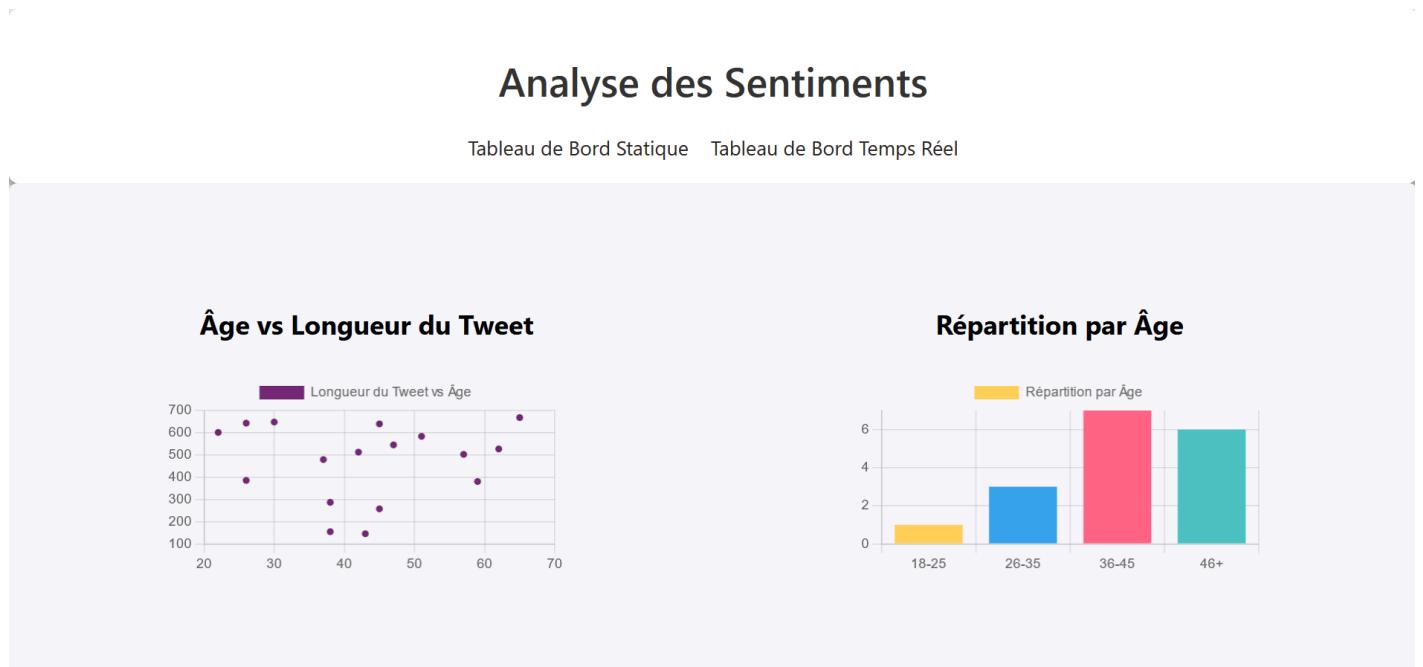


Figure 6.5: Relation entre l'âge et la longueur des tweets & Distribution des âges

Le premier graphique est un *scatterplot* illustrant la relation entre l'âge des utilisateurs et la longueur de leurs tweets. Chaque point du graphique représente un utilisateur unique, où l'axe des abscisses (x) correspond à la longueur du tweet (en nombre de caractères) et l'axe des ordonnées (y) représente l'âge de l'utilisateur. Cette visualisation permet de détecter des tendances ou des corrélations éventuelles entre l'âge et la longueur des tweets, tout en identifiant d'éventuelles anomalies ou comportements atypiques.

Le second graphique est un histogramme représentant la distribution des âges des utilisateurs. Les âges sont regroupés en classes d'intervalles (*bins*) afin de mieux visualiser leur répartition. La hauteur de chaque barre reflète le nombre d'utilisateurs appartenant à l'intervalle correspondant. Cet histogramme fournit une vue d'ensemble de la structure démographique des utilisateurs, permettant d'identifier des modes, des asymétries ou des regroupements dans les données d'âge.

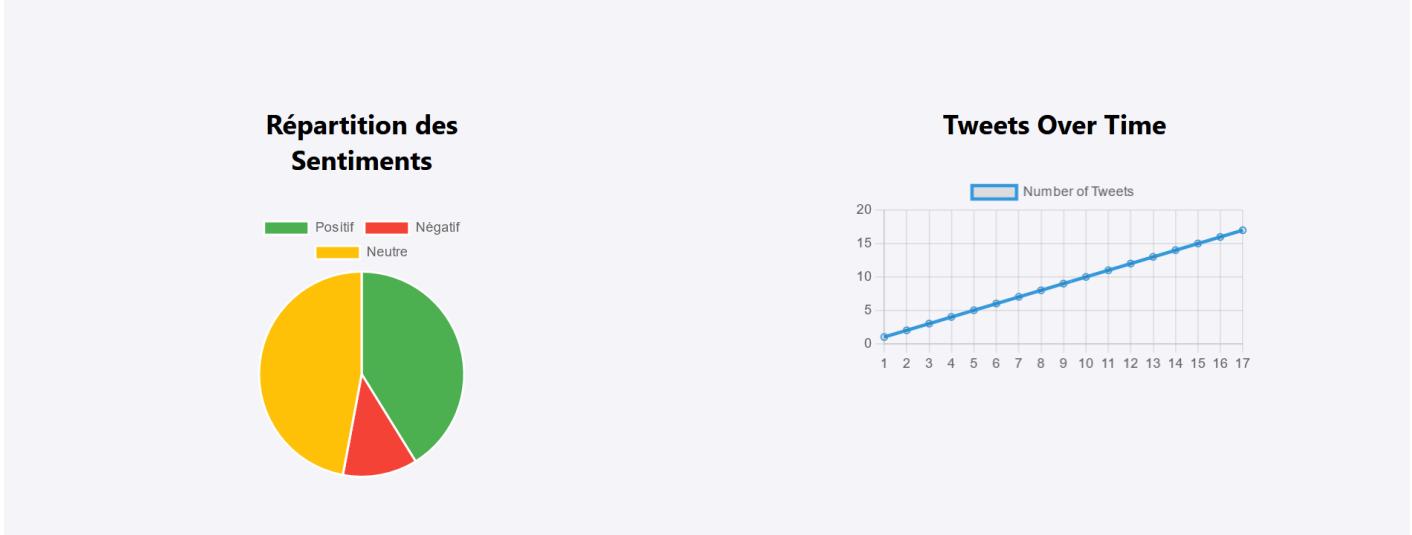


Figure 6.6: Répartition des sentiments et évolution du nombre de tweets au fil du temps

Le premier graphique, sous forme de diagramme circulaire, illustre la répartition des sentiments prédits comme positifs, négatifs ou neutres. Ces sentiments sont générés à partir d'un modèle de réseau de neurones convolutifs (CNN), qui analyse en temps réel le contenu des tweets collectés. Ce type de visualisation permet de fournir une vue globale des tendances émotionnelles exprimées par les utilisateurs sur une période donnée, offrant ainsi des informations utiles pour des analyses approfondies.

Le second graphique est une courbe linéaire qui représente l'évolution du nombre de tweets en fonction du temps. On observe une augmentation progressive et quasi-linéaire du volume des tweets à mesure que le temps avance. Cela s'explique par le fait que le modèle traite et prédit les sentiments des tweets de manière continue et en temps réel, à chaque instant où de nouveaux tweets sont générés. Cette représentation permet de suivre l'intensité de l'activité sur la plateforme et de mieux comprendre les variations temporelles du volume des données.

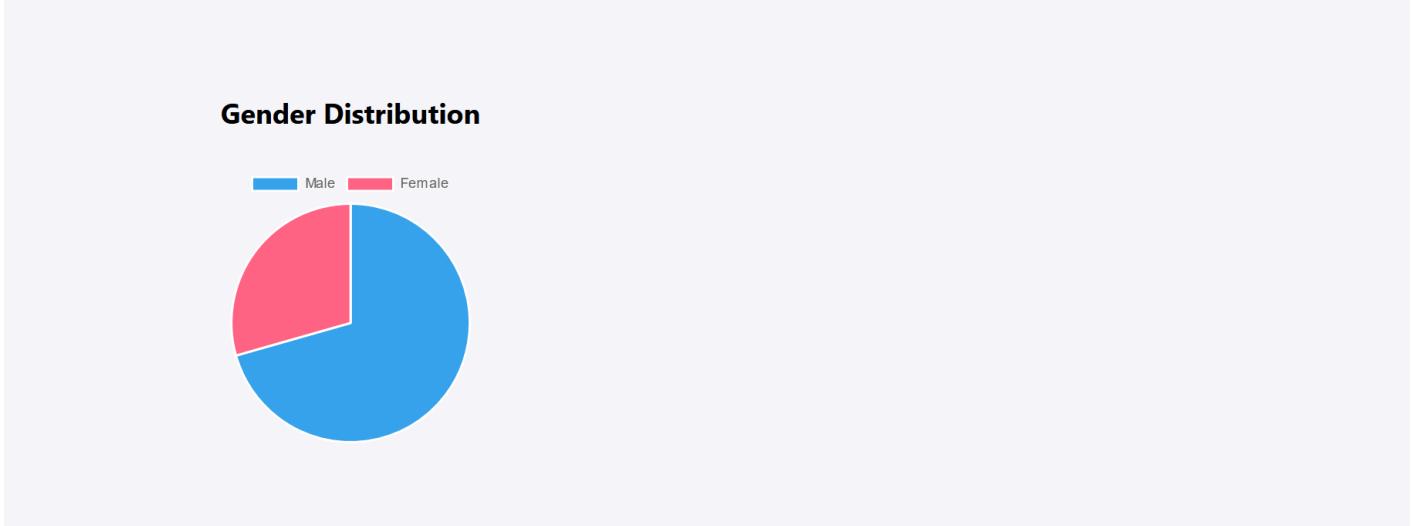


Figure 6.7: Répartition des genres parmi les auteurs des tweets

Ce graphique illustre la distribution des utilisateurs ayant écrit des tweets en temps réel, catégorisées par genre, à savoir "femme" et "homme". Cette représentation permet d'obtenir une vue d'ensemble sur la participation respective de chaque genre à la production de contenu sur la plateforme.

En analysant cette répartition, il est possible d'identifier des différences de contribution entre les genres, ce qui peut s'avérer utile dans divers contextes, tels que les études sociologiques, les analyses marketing, ou encore les enquêtes comportementales. Cette visualisation contribue également à mieux comprendre les dynamiques démographiques des utilisateurs actifs sur le réseau en temps réel.

Live Data					
Name	Age	Gender	Tweet	Timestamp	Prediction
gisella	26	Homme	Wow, such an incredible moment! Can't stop smiling. It's so nice to see all the amazing people in this world. I really want to thank all of you. Thank you for all your support and support." "Thank you all for your love and your enthusiasm. Please take care of me. Now, I am going to take my time and relax. You know I love you so much and I want you to know that I appreciate your help. Do you know how much I like your work and how you have been a part of my life for so long?" , "Hi, my name is Nanna." "I am Nana and this is my first time posting on the Internet. So please, please go ahead and post me on this page. Your name should be on	2025-01-16 08:42:13	Positive
evgeni	47	Homme	Feeling so down today. Nothing is going right. I'm really feeling down. And I need to get back to work. I don't know how to do that. But I know I could do this. It's not hard. We just need time. Maybe you'll come back. If you're ready. Or you can tell me. Thanks for all the love, (laughs) , I've got a few things I want you to see. (laughing) I just want to go home. You know, the last time I saw you, you were in the hospital with a heart attack. What's going on? . . . -I was in a coma. - ...Well, that's good. Well, my heart's still beating.	2025-01-16 08:46:43	Neutral
kolleen	51	Homme	Quiet day at home. Just me and my thoughts. I hope I'm not doing anything too bad, but I don't want to do anything bad. A: I've got to keep going. What I want is to be a good mom. A good dad. But I also want a great dad, and that's what I was going to say about my dad at the time. He was a very good man. It was hard for me to understand how he could be so nice to me. My dad was very kind, very helpful, even if I didn't really know him. There were a lot of people who would ask him, "What did you want?" I said, I can't say. His answer was, he was fine. We were just so close. And	2025-01-16 08:47:02	Positive

Figure 6.8: Aperçu du tableau des tweets en temps réel (partie 1)

trista	59	Homme	I hate this. Why does it always happen to me? I'm so tired of seeing people get shot at. They're so afraid to say anything. It's hard for me to think about the way I feel about people. I've been through this before. If I was here, I would be so upset. I don't know what it's like to be a person who doesn't feel like you belong. So I just go out and do my thing. That's what I do.	2025-01-16 08:49:42	Negative
placid	38	Homme	Everything feels hopeless right now. I'm just happy to have the opportunity to go out there and play basketball and be successful." The Pelicans were the last team to win a title during a championship drought, and they've won six of their last nine. (Nick Wass/NBAE via Getty Images))	2025-01-16 08:49:50	Neutral
hajar	37	Homme	What a terrible day. I just want it to end. (Soundbite of laughter) (Laughter) It's been a long time coming. We've been here for 30 years now and I think that's why we're here. But it's also been really, really good. It was great. There's still a lot of stuff that we've missed. And I really want to thank everybody. The people who have been working on this project. They're amazing. You've got to have a good relationship with them. Those people are amazing people. Thank you.	2025-01-16 08:50:03	Neutral
vosgi	57	Homme	Why does life have to be so unfair sometimes? "I am an immigrant and I was born here in India. I have no idea why I am here. But I will tell you that it's unfair to the children of immigrants in this country. They are very hardworking and they have a great future. So why is this happening to us?" (Image: AFP) : The migrant workers are protesting against the lack of jobs for them, the fact that the government has given them too many benefits and the way that their benefits are being distributed. .	2025-01-16 08:50:17	Positive
edana	42	Femme	This song is amazing! On repeat all day. I love this song. I think it's one of the best I've ever played. It's so nice to play with these guys and have this kind of unique sound and the way they make it so easy to get something out of it. So many of my friends have done it and I love it! It is just so funny. This is such a great song to have on your record. The song was written for me and my sister, but I didn't have the time to write it	2025-01-16 08:50:34	Positive

Figure 6.9: Aperçu du tableau des tweets en temps réel (partie 2)

Les deux figures ci-dessus représentent le tableau affichant les tweets analysés en temps réel. Ce tableau est composé de six colonnes principales :

- **Name** : le pseudonyme ou le nom de l'auteur du tweet.
- **Age** : l'âge estimé ou renseigné de l'auteur.
- **Gender** : le genre de l'auteur (homme ou femme).
- **Tweet** : le texte ou le message partagé par l'utilisateur.
- **Timestamp** : la date et l'heure exactes de la publication du tweet.
- **Prediction** : le sentiment détecté dans le tweet, classé comme *neutre*, *positif*, ou *négatif*.

Ce tableau fournit une vue structurée et détaillée des données collectées en temps réel, permettant une analyse précise des tendances émotionnelles et des caractéristiques démographiques des auteurs de tweets.

Conclusion

En conclusion, la conception d'interfaces utilisateur intuitives et interactives joue un rôle crucial dans la valorisation des résultats d'analyse et leur exploitation par les utilisateurs. À travers une visualisation

claire et dynamique des données d'analyse des sentiments, les interfaces développées permettent de comprendre rapidement les tendances et les distributions géographiques des sentiments.

L'intégration de graphiques interactifs et de cartes géographiques enrichit l'expérience utilisateur en offrant des outils accessibles pour explorer et interpréter les données. Ce système favorise non seulement une meilleure compréhension des résultats, mais renforce également la prise de décision basée sur des analyses visuelles et précises.

Conclusion Générale

Ce projet, réalisé dans le cadre de l'analyse prédictive des sentiments sur les réseaux sociaux, a permis de développer une application innovante qui intègre des technologies de pointe et des méthodologies avancées pour traiter des données massives en temps réel. En combinant des outils tels que PySpark, Pytorch, et Kafka, nous avons créé une solution performante pour l'analyse des sentiments exprimés par les utilisateurs sur les réseaux sociaux, permettant ainsi de prédire les tendances de marché.

L'approche adoptée a permis de surmonter les défis techniques liés à la gestion des grandes quantités de données, au traitement des messages en temps réel, et à l'intégration des modèles de deep learning basé sur les réseaux antagonistes génératifs (GAN). Le choix de Flask et des technologies JavaScript pour l'application front-end a permis de créer une interface utilisateur intuitive, offrant une visualisation claire et interactive des résultats.

L'une des étapes clés du projet a été l'utilisation de l'ingénierie des connaissances, à travers la création et l'enrichissement d'une ontologie des tendances de marché, alimentée par des modèles de traitement du langage naturel (NLP) et des règles d'inférences. Cette approche a enrichi la qualité des données et amélioré la précision des prédictions.

Au-delà des résultats techniques, ce projet a permis de consolider une compréhension approfondie des processus d'analyse des données massives et de leur application dans des scénarios réels. L'intégration de ces technologies a non seulement renforcé les compétences en data science, en développement d'applications en temps réel, et en traitement du langage naturel, mais a aussi permis de répondre à des enjeux actuels liés à l'exploitation des données pour la prise de décisions stratégiques.

En conclusion, ce projet ouvre la voie à de futures améliorations et extensions, notamment dans l'optimisation des modèles de deep learning et dans l'enrichissement continu de l'ontologie.