

Projet Spark - Analyse de Sentiments sur Twitter

Othmane Aboussaad

Iyad Benmosbah

Kenza Mouharrar

Samuel Zerrouk

Novembre 2024

Table des matières

Introduction	3
1 Apache Spark : Concepts et Outils	4
1.1 Présentation de Spark	4
1.2 Installation de Spark et Scala	4
1.3 Premiers Pas avec Spark	4
2 RDD (Resilient Distributed Datasets)	5
2.1 Introduction	5
2.2 Concepts de Base	5
2.3 Création de RDD	5
2.4 Transformations	6
2.5 Actions	6
2.6 Partitionnement et Parallélisme	7
2.6.1 Définir le nombre de partitions	7
2.6.2 Repartitionner les données	7
2.7 Mise en Cache et Persistance	7
2.8 Exemple Complet avec RDD	8
2.9 Avantages et Cas d'Utilisation des RDDs	8
3 GraphX : Analyse de graphes avec Apache Spark	9
3.1 Introduction à GraphX	9
3.2 Représentation des Graphes dans GraphX	9
3.3 Exemple de Construction d'un Graph avec GraphX	10
3.4 Analyse des Graphes	11
3.5 Application : Graphe de Co-Occurrences de Mots dans les Tweets	11
3.6 Avantages de l'Approche par Graphes	11
3.7 Méthodologie	11
3.7.1 Collecte des Données	11
3.7.2 Prétraitement des Données	12
3.7.3 Analyse des Sentiments	12

3.8	Résultats et Analyse	13
3.8.1	Clusters de Mots	13
3.8.2	Analyse des Graphes	13
3.9	Conclusion et Perspectives	13
3.9.1	Conclusion	13

Introduction

L'objectif de ce projet est de se familiariser avec Apache Spark pour le traitement de données à grande échelle, tout en développant une application pratique d'analyse de sentiments en temps réel sur Twitter.

Ce rapport détaille les étapes suivies, les concepts étudiés et les résultats obtenus.

Chapitre 1

Apache Spark : Concepts et Outils

1.1 Présentation de Spark

Apache Spark est une plateforme de traitement distribué utilisée pour traiter de grandes quantités de données de manière rapide et fiable.

Avantages : traitement en mémoire, API unifiée, tolérance aux pannes.

1.2 Installation de Spark et Scala

- Télécharger Apache Spark depuis <https://spark.apache.org/downloads.html>.
- Installer Scala (<https://www.scala-lang.org/>).
- Configurer les variables d'environnement (SPARK_HOME, PATH).
- Vérifier l'installation avec la commande `spark-shell`.

1.3 Premiers Pas avec Spark

Lancer `spark-shell` et exécuter un exemple simple :

```
1 val data = List(1, 2, 3, 4, 5)
2 val rdd = sc.parallelize(data)
3 val squares = rdd.map(x => x * x)
4 squares.collect()
```

Chapitre 2

RDD (Resilient Distributed Datasets)

2.1 Introduction

Les RDDs sont le cœur d'Apache Spark, permettant la manipulation et le traitement distribué des données à grande échelle. Ils offrent une abstraction haut niveau pour travailler avec des données immuables et tolérantes aux pannes dans un environnement distribué. Cette section détaille leur fonctionnement, leur création, et les opérations possibles.

2.2 Concepts de Base

- **Immutabilité** : Une fois créé, un RDD ne peut pas être modifié. Les transformations appliquées sur un RDD génèrent un nouveau RDD, garantissant la reproductibilité et la sécurité des calculs.
- **Partitionnement** : Les données dans un RDD sont divisées en partitions, traitées parallèlement sur différents noeuds du cluster. Le nombre de partitions peut être défini manuellement.
- **Tolérance aux pannes** : Grâce à un mécanisme appelé *lineage*, Spark peut reconstruire les partitions perdues en cas de panne.
- **Évaluation différée (lazy evaluation)** : Les transformations sur un RDD sont différées jusqu'à ce qu'une action soit déclenchée.

2.3 Crédit de la RDD

Les RDDs peuvent être créés de plusieurs façons :

- À partir d'une collection locale :

```

1 val data = List(1, 2, 3, 4, 5)
2 val rdd = sc.parallelize(data)

```

- À partir de fichiers externes :

```

1 val textFile = sc.textFile("chemin/vers/fichier
    .txt")

```

- À partir de transformations :

```

1 val words = textFile.flatMap(line => line.split
    (" "))

```

- À partir de bases de données : Lire des données depuis des sources comme HDFS, Cassandra ou JDBC.

2.4 Transformations

Les transformations produisent de nouveaux RDDs sans modifier les données d'origine. Voici quelques exemples courants :

Transformation	Description	Exemple
map	Applique une fonction à chaque élément.	val squared = rdd.map(x => x * x)
filter	Garde les éléments qui remplissent une condition.	val even = rdd.filter(x => x % 2 == 0)
flatMap	Produit plusieurs résultats par élément.	val words = lines.flatMap(line => line.split(" "))
distinct	Supprime les doublons.	val unique = rdd.distinct()
union	Combine deux RDDs.	val combined = rdd1.union(rdd2)

TABLE 2.1 – Exemples de Transformations RDD

2.5 Actions

Les actions déclenchent les calculs et retournent les résultats. Voici quelques exemples courants :

Action	Description	Exemple
collect	Récupère tous les éléments dans le driver.	val result = rdd.collect()
count	Compte le nombre d'éléments.	val count = rdd.count()
first	Retourne le premier élément.	val first = rdd.first()
take	Récupère les N premiers éléments.	val top = rdd.take(5)
reduce	Combine les éléments en appliquant une fonction.	val sum = rdd.reduce((x, y) => x + y)

TABLE 2.2 – Exemples d’Actions RDD

2.6 Partitionnement et Parallélisme

2.6.1 Définir le nombre de partitions

Il est possible de spécifier le nombre de partitions lors de la création d’un RDD :

```
1 val rdd = sc.parallelize(1 to 100, 5) // 5
    partitions
```

2.6.2 Repartitionner les données

Il est possible d’ajuster le nombre de partitions à l’aide des méthodes `repartition` ou `coalesce` :

```
1 // Augmenter le nombre de partitions
2 val repartitioned = rdd.repartition(10)
3
4 // Reduire le nombre de partitions
5 val coalesced = rdd.coalesce(2)
```

2.7 Mise en Cache et Persistance

Pour améliorer les performances, Spark permet de conserver les données fréquemment utilisées en mémoire ou sur disque.

— **Cache** : Utilise la mémoire pour stocker les données.

```
1 val cached = rdd.cache()
```

- **Persistance** : Permet de choisir le niveau de stockage (mémoire, disque, ou les deux).

```
1 val persisted = rdd.persist(org.apache.spark.
    storage.StorageLevel.MEMORY_AND_DISK)
```

2.8 Exemple Complet avec RDD

Voici un exemple illustrant la création, la transformation, et l'action sur un RDD :

```
1 // Creation d'un RDD a partir d'une collection
2 val data = List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
3 val rdd = sc.parallelize(data, 4) // Divise en 4
    partitions
4
5 // Transformation : carre des nombres
6 val squaredRdd = rdd.map(x => x * x)
7
8 // Transformation : filtrer les nombres pairs
9 val evenSquares = squaredRdd.filter(x => x % 2 == 0)
10
11 // Action : collecter les resultats
12 val result = evenSquares.collect()
13
14 // Affichage des resultats
15 println("R sultats: " + result.mkString(", "))
```

2.9 Avantages et Cas d'Utilisation des RDDs

- **Flexibilité** : Les RDDs permettent un contrôle précis sur les transformations et actions.
- **Tolérance aux Pannes** : Grâce à *lineage*, les données peuvent être recréées en cas de panne.
- **Cas d'Utilisation** :
 - Manipulation de données non structurées ou semi-structurées.
 - Traitements nécessitant des transformations complexes.

Chapitre 3

GraphX : Analyse de graphes avec Apache Spark

3.1 Introduction à GraphX

GraphX est une composante d'Apache Spark qui fournit un framework puissant et flexible pour le traitement de graphes à grande échelle. Il combine la puissance de calcul distribué de Spark avec une API intuitive pour manipuler et analyser des graphes. GraphX représente les graphes comme une structure de données distribuée composée de **sommets** et d'**arêtes**, avec des propriétés associées.

- **Graph de propriétés** : Chaque sommet et chaque arête peuvent avoir des propriétés associées (par exemple, des identifiants ou des poids).
- **Partitionnement 2D** : GraphX utilise une technique de partitionnement 2D pour minimiser la communication entre machines en répartissant les sommets et arêtes sur différentes partitions.
- **RDDs** : GraphX s'appuie sur les RDDs pour représenter les sommets, les arêtes et la table de routage qui associe chaque sommet à sa partition.

3.2 Représentation des Graphes dans GraphX

Un graphe est défini par :

- **Table des sommets** : Chaque sommet est représenté par un RDD contenant son ID unique et ses propriétés.
- **Table des arêtes** : Chaque arête est représentée par un RDD contenant les IDs des sommets source et destination, ainsi que ses propriétés.

tés.

- **Table de routage** : Elle associe les sommets aux partitions où ils sont stockés, facilitant la distribution et la gestion des données.

3.3 Exemple de Construction d'un Graphe avec GraphX

Voici un exemple concret pour construire et analyser un graphe simple avec GraphX :

```
1 // Importer les bibliothèques nécessaires
2 import org.apache.spark._
3 import org.apache.spark.graphx._
4 import org.apache.spark.rdd.RDD
5
6 // Etape 1 : Créer un RDD de sommets
7 val vertices: RDD[(VertexId, String)] = sc.
8     parallelize(Array(
9         (1L, "Alice"), (2L, "Bob"), (3L, "Charlie")
10    ))
11
12 // Etape 2 : Créer un RDD d'arêtes
13 val edges: RDD[Edge[String]] = sc.parallelize(Array(
14     Edge(1L, 2L, "ami"), Edge(2L, 3L, "coll gue")
15    ))
16
17 // Etape 3 : Construire le graphe
18 val graph = Graph(vertices, edges)
19
20 // Etape 4 : Calculer les degrés des sommets
21 val degrees: VertexRDD[Int] = graph.degrees
22
23 // Etape 5 : Afficher les résultats
24 degrees.collect.foreach(println)
25 graph.triplets.collect.foreach(println)
```

3.4 Analyse des Graphes

Une fois le graphe construit, GraphX permet d'effectuer diverses analyses :

- **Calcul des degrés** : Identifie le nombre de connexions de chaque sommet.
- **Clusters de mots** : Détecte des groupes de mots fortement connectés dans un graphe construit à partir de co-occurrences.
- **Mots centraux** : Identifie les mots les plus connectés et leur rôle dans le réseau.
- **Évolution temporelle** : Analyse les changements dans le graphe sur différentes périodes.

3.5 Application : Graphe de Co-Occurrences de Mots dans les Tweets

Dans le contexte de l'analyse de sentiments, les graphes peuvent être utilisés pour étudier les co-occurrences de mots dans les tweets.

- **Sommets** : Chaque mot unique dans un ensemble de tweets représente un sommet.
- **Arêtes** : Une arête est tracée entre deux mots s'ils apparaissent ensemble dans un tweet.
- **Poids des arêtes** : Représente la fréquence de co-occurrence.

3.6 Avantages de l'Approche par Graphes

- **Contexte enrichi** : Les co-occurrences capturent le contexte d'utilisation des mots.
- **Relations cachées** : Les graphes révèlent des connexions inattendues entre les mots.
- **Analyse approfondie** : Permet d'identifier des clusters sémantiques ou des tendances temporelles.

3.7 Méthodologie

3.7.1 Collecte des Données

Nous avons collecté les données directement depuis Kaggle. La base de données porte sur des tweets à propos de la compagnie aérienne **Virgin**

America. Le fichier `Tweetstest.csv` contient initialement **10 000 tweets** répartis selon trois sentiments :

- Positive
 - Neutral
 - Negative

Cependant, afin d'améliorer le temps d'exécution, nous avons réduit la taille de la base à **100 tweets**.

3.7.2 Prétraitement des Données

Les étapes de prétraitement incluent :

- **Tokenisation** : séparation des phrases en mots individuels.
 - **Suppression des mots vides (stopwords)**.

Ce processus permet d'obtenir un jeu de données propre pour l'analyse.

FIGURE 3.1 – Exemple de données prétraitées.

3.7.3 Analyse des Sentiments

Pour l'analyse des sentiments, nous avons utilisé :

- **CountVectorizer** : transformation des mots en vecteurs numériques.
 - **StringIndexer** : assignation d'une étiquette numérique pour chaque catégorie de sentiment.
 - **Logistic Regression** : modèle de prédiction pour classifier les sentiments.

La mise en place du pipeline d'analyse comprend :

1. Transformation du texte en vecteurs.
2. Indexation des labels.
3. Classification des sentiments avec la régression logistique.

3.8 Résultats et Analyse

Les résultats obtenus montrent une classification réussie des tweets selon leur polarité (*positive, neutral, negative*).

	A	B	C	D	E	F	G	H	I	J	K	L
1	text,airline_sentiment,lrPrediction											
2	@VirginAmerica	- Let 2 scanned in passengers leave the plane than told someone to remove their bag from 1st class bin? #uncomfortable	,negative	,2.0								
3	@VirginAmerica	why is flight 345 redirected?	,neutral	,2.0								
4	@VirginAmerica	Random Q: what's the distribution of elevate avatars? I bet that kitty has a disproportionate share	http://t.co/APtZpuROp4	,neutral	,2.0							
5	@VirginAmerica	@ladygaga @carrieunderwood Julie Andrews all the way though @ladygaga was very impressive! NO to @Carrieunderwood	,positive	,1.0								
6	@VirginAmerica	DREAM http://t.co/oA2dRfAoQ2	http://t.co/IWWdAc2kHx	,neutral	,2.0							
7	@VirginAmerica	I'm #elevategold for a good reason: you rock!!	,positive	,2.0								
8	@VirginAmerica	are flights leaving Dallas for Seattle on time Feb 24?	,neutral	,1.0								
9	@VirginAmerica	I have an unused ticket but moved to a new city where you don't fly. How can I fly with you before it expires?	#travelhelp	,neutral	,2.0							
10	@VirginAmerica	when can I book my flight to Hawaii??	,neutral	,2.0								
11	Nice RT @VirginAmerica:	Vibe with the moonlight from takeoff to touchdown. #MoodlitMonday #ScienceBehindTheExperience	http://t.co/Y7O0uNxTQP	,neutral	,2.0							
12	@VirginAmerica	What happened 2 ur vegan food options?! At least say on ur site so i know I won't be able 2 eat anything for next 6 hrs	#fail	,negative	,0.0							
13	@VirginAmerica	I love the hipster innovation. You are a feel good brand.,positive	,2.0									
14	@VirginAmerica	did you know that suicide is the second leading cause of death among teens 10-24	,neutral	,2.0								
15	@VirginAmerica	I didn't today... Must mean I need to take another trip!	,neutral	,2.0								

FIGURE 3.2 – Résultats des prédictions.

3.8.1 Clusters de Mots

Nous avons utilisé des graphes pour analyser les relations entre les mots dans les tweets. Les clusters identifiés permettent de visualiser les groupes de mots les plus fréquents.

3.8.2 Analyse des Graphes

L'analyse des graphes a révélé les connexions principales entre les mots dans les tweets.

3.9 Conclusion et Perspectives

3.9.1 Conclusion

Ce projet a permis de :

- Se familiariser avec **Apache Spark** pour le traitement des données massives.

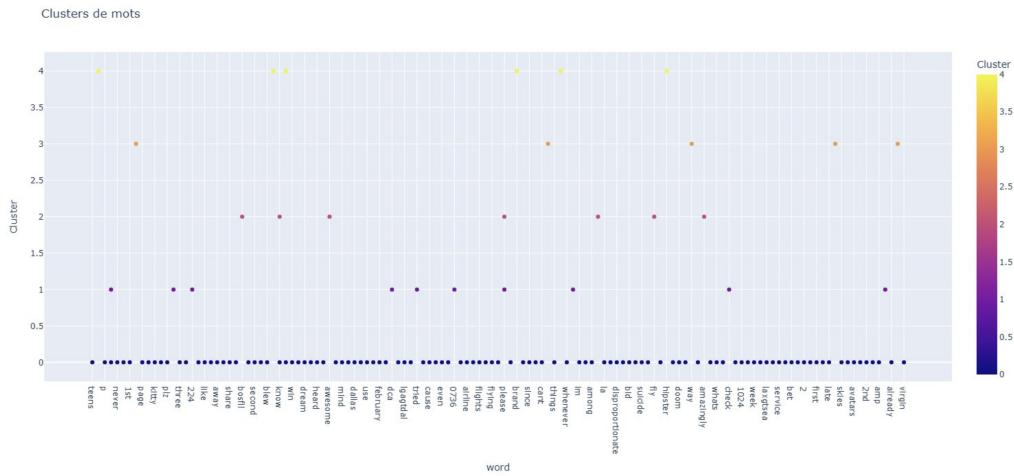


FIGURE 3.3 – Clusters de mots basés sur les co-occurrences.

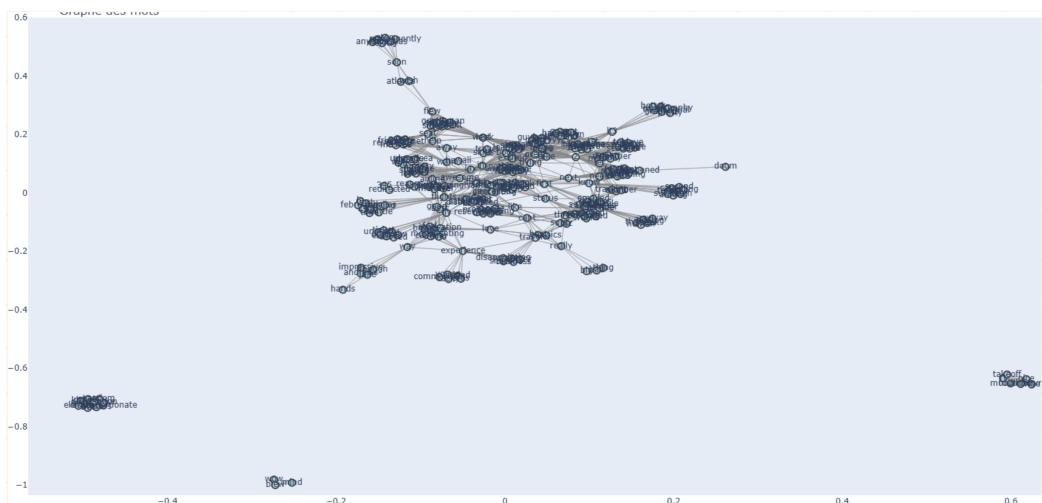


FIGURE 3.4 – Graphe des connexions de mots.

- Mettre en place une analyse des sentiments en utilisant des outils de machine learning.
 - Appliquer des techniques d'analyse de graphes pour l'étude des co-occurrences de mots.

Références

- Apache Spark Documentation : <https://spark.apache.org/>
 - Scala Documentation : <https://docs.scala-lang.org/>