Projet de big data avec Apache Spark

Stanislas Morbieu

Décembre 2020

Objectifs

Le but de ce projet est de manipuler des APIs d'Apache Spark de plus haut niveau que celle vu en cours (qui utilise des RDDs). Plus particulièrement, nous utiliserons Spark SQL qui permet de travailler avec des DataFrames, ainsi que Spark MLlib qui permet d'utiliser des fonctionnalités de Machine Learning (ou "apprentissage automatique"). Le projet consiste à regrouper des documents textuels tel que les documents qui partagent le même thème se retrouvent dans le même groupe, et les documents qui portent sur des sujets très différents se trouvent dans des groupes différents.

Mise en place de l'environnement de travail

1. (a) Créer un notebook sur Google Colab
   (b) Installer Spark comme vu en cours
   (c) Définir la valeur de la variable d'environnement `PYSPARK_SUBMIT_ARGS` à `'packages org.apache.spark:spark-avro_2.12:3.0.1 pyspark-shell'`.
   Ceci permettra d'utiliser Avro.
   (d) Créer un objet SparkContext
   (e) Créer un objet de type SparkSession. Cet objet servira pour utiliser Spark SQL qui offre des fonctionnalités de plus haut niveau que les RDDs.

Données

2. (a) Télécharger des données textuelles à l'adresse suivante : `http://qwone.com/~jason/20Newsgroups/20news-19997.tar.gz`
   (b) Décompresser les données
   (c) Charger les données dans deux variables de type RDD tel que :
   — Les documents de la catégorie "alt.atheism" soient dans un RDD et ceux de "rec.sport.baseball" soient dans un autre RDD
   — Chaque RDD représente une liste de documents (un élément du RDD = un document)
   (d) Séparer le corps du message de l'entête.
   (e) Extraire quelques champs de l'entête, par exemple l'organisation et la catégorie (champ "Newsgroups").
   (f) Fusionner les deux RDD
   (g) Transformer le nouveau RDD obtenu pour que chaque élément soit de type pyspark.sql.Row
   (h) Créer un objet de type DataFrame à partir du RDD précédent
   (i) Sauvegarder la DataFrame au format Avro
   (j) Sauvegarder la DataFrame au format Parquet

Analyse descriptive

3. Faire une analyse descriptive de la DataFrame obtenue à l'étape précédente :
   (a) Vérifier qu'on a bien deux catégories différentes de documents
   (b) Donner le nombre d'organisations différentes
   (c) Suivant les champs extraits, donner d'autres statistiques descriptives

Transformation du texte et clustering

4. Consulter la page : `https://spark.apache.org/docs/latest/ml-features.html#feature-extractors`
   (a) Découper les documents en listes de mots à l'aide de Tokenizer
   (b) Créer une représentation vectorielle des documents à l'aide de HashingTF
5. Consulter la page : `https://spark.apache.org/docs/latest/ml-clustering.html` Utiliser l'algorithme KMeans avec un nombre du cluster égal à 2 (pour essayer de retrouver les 2 catégories qu'on a)
6. Analyser les résultats et la qualité de la partition obtenue. On pourra par exemple utiliser l'information mutuelle normalisée (NMI) et comparer avec ce qu'on aurait obtenu avec l'implémentation de k-means du package scikit-learn.

On cherche dans la suite à implémenter l'algorithme Spherical k-means.

Implémentation de K-means unidimensionnel

7. Donner une implémentation de l'algorithme des k-moyennes avec Apache Spark. On se limitera au cas unidimensionnel (chaque point est représenté par une seule dimension). Pour celà :
   (a) Définir la fonction `compute_centroids` qui prend en argument deux RDD :
   — `points` : chaque élément est la valeur du point pour sa seule dimension ;
   — `cluster_ids` : chaque élément est l'id du cluster auquel est associé le point.
   Cette fonction retourne un RDD de couples (cluster_id, moyenne). On utilisera par exemple les fonctions `reduceByKey`, `mapValues`, `sortByKey`, `zip` et `map` et on suivra les étapes suivantes :
      i. Construire le RDD `sum_by_cluster_id` où chaque élément est un couple constitué de l'id du cluster et de la somme des éléments contenus dans ce cluster.
      ii. Construire le RDD `count_by_cluster_id` où chaque élément est un couple constitué de l'id du cluster et du nombre d'éléments contenus dans ce cluster.
      iii. Construire le RDD de couples (cluster_id, moyenne).
   (b) Définir la fonction `assign_clusters` qui prend en argument deux RDDs :
   — `points` : comme défini précédemment ;
   — `centroids` : retourné par la fonction `compute_centroids` définie précédemment.
   Cette fonction retourne un RDD dont chaque élément est l'id du cluster auquel est associé le point. On décomposera ce code en trois étapes :
      i. Définir la fonction `squared_distances` qui prend deux arguments : la valeur pour le point et la liste Python des moyennes des différents clusters. Cette fonction doit retourner une liste des carrés des distances entre le point et les différentes moyennes des clusters.
      ii. Récupérer les moyennes issues du RDD `centroids` sous forme de liste Python. Quelle supposition a-t-on faite pour effectuer cette opération ?
      iii. Utiliser la fonction `numpy.argmin` pour retourner le RDD des assignations.
   (c) Implémenter l'étape d'initialisation et l'itération.

Spherical k-means et k-means multidimensionnel

8. Adapter l'implémentation précédente pour gérer le cas multidimensionnel
9. Adapter l'implémentation précédente pour implémenter Spherical k-means
10. Analyser les résultats. On pourra par exemple comparer à l'implémentation de Spherical k-means du package Coclust (`coclust.readthedocs.io/`) ou spherecluster (`https://github.com/jasonlaska/spherecluster`).

Autres classifications

11. Explorer les APIs de la bibliothèque MLlib de Spark pour faire de la classification non supervisée (`https://spark.apache.org/docs/latest/ml-clustering.html`), analyser et comparer les résultats.
12. Explorer les APIs de la bibliothèque MLlib de Spark pour faire de la classification supervisée (`https://spark.apache.org/docs/latest/ml-classification-regression.html`), analyser et comparer les résultats.