Projet de big data avec Apache Spark

Stanislas Morbieu

Mai 2021

Consignes générales

L'objectif de ce projet est d'utiliser Apache Spark pour implémenter un algorithme de partitionnement tel que k-means. Le projet est à réaliser en binome.

Le rendu attendu est double : un **rapport** et le **code associé**. Une attention toute particulière devra être portée à :

- l'**explication du fonctionnement** de Spark (ex : "Comment se fait la parallélisation des données pour telle partie de code ?", "Quelles sont les limites ?", ...) ;
- l'**interprétation** des résultats (ex : "On obtient une valeur de xx pour le score yy, ce qui signifie que les classes sont bien séparées.") ;
- l'**illustration** par quelques exemples (ex : "Voici la visualisation des résultats du clustering pour un jeu de données généré de la manière suivante... ").

Les questions suivantes doivent vous guider pour l'implémentation, tout élément complémentaire répondant aux points précédents sera apprécié.

Mise en place de l'environnement de travail

1. (a) Créer un notebook sur Google Colab
   (b) Installer scikit-learn et une bibliothèque de visualisation (par exemple matplotlib) ou tout autre package qui pourra servir dans la suite
   (c) Installer Spark
   (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) Générer un jeu de données constitué de points avec deux dimensions (pour faciliter la visualisation). On pourra par exemple utiliser : `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html`
   (b) Créer un RDD pour représenter le jeu de données où chaque élément est un point caractérisé par ses coordonnées et son étiquette (classe).
   (c) Transformer le RDD pour que chaque élément soit de type pyspark.sql.Row
   (d) Créer un objet de type DataFrame à partir du RDD précédent.
   (e) Constituer d'autres jeux de données pouvant illustrer le résultat de classifications. On pourra par exemple faire varier le nombre de points et de dimensions.

Analyse descriptive

3. Faire une analyse descriptive (simple) de la DataFrame obtenue à l'étape précédente. On pourra par exemple :

(a) Vérifier qu'on a bien le nombre de classes générées

(b) Vérifier le nombre de points

(c) Voir la variance des classes

Clustering avec des implémentations disponibles

4. Partionner les points avec l'algorithme k-means, en spécifiant le nombre de clusters comme étant égal au nombre de classes du jeu de données (l'objectif ici n'est pas de trouver le nombre de clusters idéal) :
   (a) en utilisant scikit-learn
   (b) en utilisant une implémentation disponible dans Spark (voir la bibliothèque MLlib incluse dans Spark). Deux versions sont disponibles : une avec l'API DataFrame et une autre avec l'API RDD.
5. Analyser les résultats et la qualité de la partition obtenue. On pourra par exemple utiliser l'information mutuelle normalisée (NMI) dont l'implémentation est disponible dans scikit-learn.
6. Mesurer le temps d'exécution des différentes méthodes. Discuter des avantages potentiels des deux méthodes (scikit-learn et Spark).

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

Implémentation de K-means

7. Donner une implémentation de l'algorithme k-means avec Apache Spark. On pourra se limiter dans un premier temps 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.
8. Adapter l'implémentation précédente si nécessaire pour gérer le cas multidimensionnel
9. Analyser les résultats et les comparer aux méthodes précédentes.

Bonus

10. Implémenter une variante de k-means. On pourra par exemple implémenter k-médoïdes ou Spherical k-means.
11. Utiliser l'implémentation précédente sur des exemples pertinents.