Faire du ML avec des données massives : Introduction à Apache Spark

Table of Contents

Faire du ML avec des données massives : Introduction à Apache Spark 1

Préambule 1

Partie I : Installation de Spark et intégration de Jupyter 2

Installer la JVM 2
Installer Spark 3
Intégrer Jupyter Notebook 5

Partie II : Exploration de quelques méthodes de base PySpark 6

Le jeu de données 6
La fonction map() 7
La fonction reduceByKey() 8
La fonction filter() 9

Préambule

Les données massives (big data) ont fait l'objet d'une grande attention ces dernières années pour de bonnes raisons. Des entreprises comme Google ou Yahoo! ont vu leur base d'utilisateurs croître de manière significative et collectent de plus en plus d'informations sur la manière dont les gens interagissent avec leurs produits.

Si les éditeurs de logiciels ont amélioré leur capacité à collecter des quantités massives de données, leur capacité à les analyser et à leur donner un sens ne s'est pas toujours améliorée. Les grandes entreprises ont dû mettre au point de nouveaux outils et de nouvelles approches capables de pallier ce problème.

Le paradigme **MapReduce** permet de distribuer efficacement le traitement des données massives sur des centaines voire des milliers d'ordinateurs. **Hadoop** est un projet open source qui est rapidement devenu la boîte à outils de traitement dominante pour les données volumineuses.

Hadoop se compose d'un système de fichiers (Hadoop Distributed File System, ou **HDFS**) et de sa propre implémentation du paradigme MapReduce. MapReduce convertit les calculs en étapes **Map** et **Reduce** que Hadoop peut facilement distribuer sur de nombreuses machines.

Hadoop a ainsi permis d'analyser de grands ensembles de données, mais il s'appuie fortement sur le stockage sur disque pour les calculs. Bien qu'il soit peu coûteux de stocker de grandes quantités de données de cette manière, l'accès en lecture et en écriture sur le disque dur est beaucoup plus lent. Pour les calculs nécessitant plusieurs passages sur les mêmes données ou de nombreuses étapes intermédiaires, cela pose problème. Hadoop en vient à ne pas être une bonne solution en raison de la nécessité d'écrire et de lire sur le disque entre chaque étape.

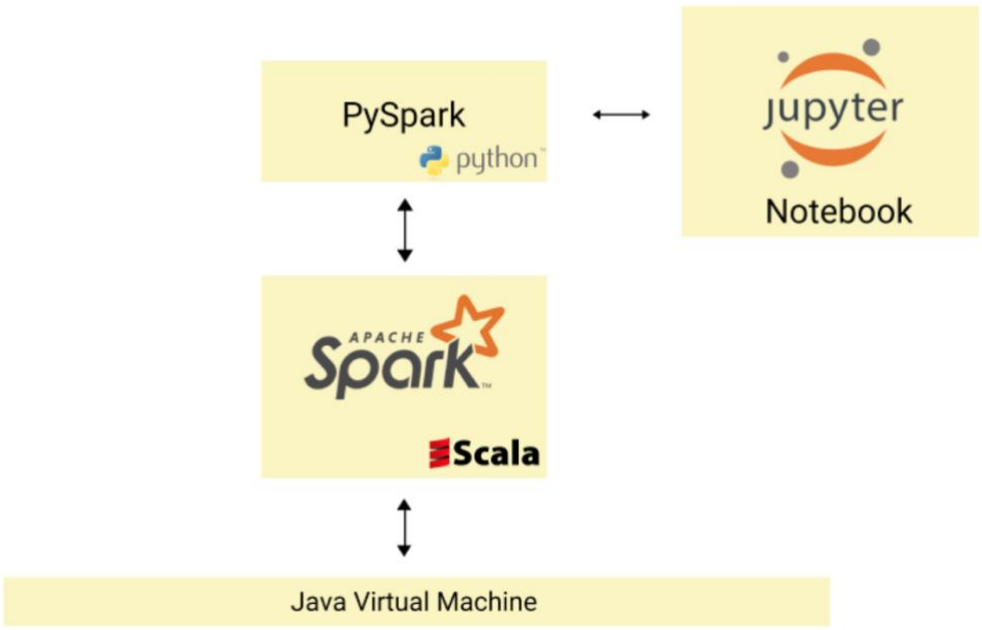
Profitant de la baisse significative des coût de la mémoire vive (RAM), **Spark** s'est rapidement imposée comme solution en remplacement de Hadoop. Spark propose des structures de données distribuées stockées en mémoire et permettant d'accélérer de manière significative le traitement des données.

Partie I : Installation de Spark et intégration de Jupyter

Dans cette section, nous allons installer Spark sur Windows et intégrer PySpark à Jupyter Notebook. Spark peut être utilisé de deux manières :

- **Mode local :** l'application Spark s'exécute sur une seule machine. Ce mode sert à construire et tester vos prototypes d'applications Spark sur votre propre machine.
- **Mode cluster :** l'application Spark s'exécute sur plusieurs machines. Ce mode cluster est utile lorsque vous voudrez exécuter votre application Spark sur plusieurs machines dans un environnement cloud comme Amazon Web Services ou Microsoft Azure.

Dans ce TP, nous allons couvrir l'installation de Spark en mode local sur Windows.
Le schéma ci-après présente les composantes haut niveau de l'installation que nous ferons aujourd'hui.



Spark fonctionne sur la machine virtuelle Java (JVM), qui est fournie avec le kit de développement Java SE (JDK). Nous allons installer la version JDK 8u202.

Installer la JVM

1. Télécharger la version de Java SE Development Kit qui nous intéresse.
    a. Ouvrir ce lien et télécharger le jdk correspondant à la version d'architecture de votre Windows (32 ou 64bits)
    b. Installer jdk en double cliquant sur l'exécutable téléchargé et suivre les instructions. Lorsqu'on vous demande dans quel répertoire souhaitez-vous

installer le jdk, je vous recommande de l'installer dans un répertoire qui n'a pas d'espace dans le nom (par exemple créer un nouveau répertoire c:\jdk , éviter 'Program Files')

c. Vérifier l'installation en ouvrant l'invité de commande et en tapant la commande 'java -version'. Vous devriez obtenir un message indiquant la version de java actuellement utilisée.
d. Si vous obtenez une erreur ou une version différente de celle que vous venez d'installer, votre installation Java JDK n'a probablement pas été ajoutée correctement à votre PATH. Vous pouvez régler cela en suivant les étapes de la question 2

2. Ajouter JDK au PATH.
    a. Tout d'abord, déterminez l'emplacement du fichier bin du JDK. En général, il se trouve dans C:\Program Files\Java\répertoire-de-votre-version-de-JDK\bin mais il peut se trouver ailleurs si vous l'avez installé dans un autre répertoire. Se placer dans le répertoire et copier le chemin.
    b. Une fois que vous avez le chemin d'installation du JDK, cliquez avec le bouton droit de la souris sur Poste de travail et sélectionnez Propriétés dans le menu contextuel.

Remarque : l'accès à la boîte des propriétés du système varie légèrement d'une version de Windows à l'autre. Quelle que soit la version utilisée, cliquez avec le bouton droit de la souris sur l'icône "Ordinateur" ou "Poste de travail" dans l'Explorateur, puis cliquez sur Protection du système pour ouvrir la boîte Propriétés du système.

    c. Dans la boîte Propriétés du système, sélectionnez l'onglet Avancé. Dans l'onglet Avancé, cliquez sur le bouton Variables d'environnement. Localisez la ligne PATH sous Variables système et cliquez sur Modifier. Dans la zone Modifier la variable système, en veillant à ce qu'un point-virgule préserve le dernier chemin d'accès, ajoutez le chemin d'installation du fichier bin du kit de développement Java. Par exemple, C:\Program Files\Java\jdk1.8.0_10\bin. Fermez avec un point-virgule. Cliquez sur OK.
    d. Une fois de retour dans les Propriétés du système, onglet Avancé, cliquez sur Appliquer les modifications.

Installer Spark

Nous allons télécharger et travailler avec une version pré-compilée de Spark.

1. Naviguer sur la page de téléchargement des archives de Spark et sélectionner le fichier **spark-2.4.6-bin-hadoop2.7.tgz**
    a. Le fichier téléchargé est au format .tgz, si vous n'avez pas encore installé de logiciels capables de décompresser ce format de fichier, vous pouvez télécharger le logiciel **WinRar**. Une version gratuite est disponible sur le lien rarlabs.com, dans l'onglet 'Downloads', télécharger la version de WinRar correspondant à votre architecture de Windows (32bits ou 64bits).
    b. Utiliser WinRar pour décompresser le fichier spark téléchargé.
    c. Une fois le fichier décompressé, on obtient un dossier associé contenant spark. Au lieu de le laisser dans le répertoire des téléchargements, on va le placer dans un endroit où en s'en souviendra plus facilement.

d. Aller sur le lecteur C et créer un nouveau dossier que vous nommerez spark. L'emplacement sera donc C:\spark. On ouvre ce nouveau dossier et on y copie le contenu entier du répertoire spark se trouvant dans le dossier des téléchargements.
e. Ensuite, dans c:\spark ouvrir le dossier conf et renommer le fichier log4j.properties.template en log4j.properties
f. Ouvrir le fichier des propriétés log4j.properties avec un éditeur de texte tel que WordPad
g. Rechercher la ligne log4j.rootCategory=INFO et changer la valeur INFO en ERROR. On obtient log4j.rootCategory=ERROR. Sauvegarder et fermer votre éditeur.

2. L'étape suivante est d'installer un utilitaire qui va simuler la présence de Hadoop sur notre machine de telle sorte que nous n'ayons pas à installer Hadoop pour de vrai. Cela est suffisant pour explorer spark dans le cadre d'une installation en mode local sous windows et nous permet d'éviter une installation supplémentaire. *(Note : Cette étape n'est pas nécessaire pour les systèmes d'exploitation MacOS ou Linux)*.
   a. Allez sur ce lien. Lorsque vous cliquez dessus, vous obtenez un fichier exécutable que vous allez copier hors du dossier de téléchargements.
   b. Allez sur le lecteur C et créer un nouveau dossier appelé winutils. A l'intérieur de celui-ci créer un sous-répertoire nommé bin. Dans c:\winutils\bin, coller l'exécutable winutils.exe que nous venons de télécharger.
   c. Ouvrir ensuite l'invite de commande et taper cd c:\winutils\bin\ puis faire entrée. Taper winutils.exe chmod 777 \tmp\hive. Cette commande permet de s'assurer que toutes les autorisations de fichiers dont vous avez besoin pour exécuter Spark avec succès sont en place sans aucune erreur. Vous pouvez fermer l'invite de commande.
3. On va maintenant configurer quelques variables d'environnement.
   a. Fermer toutes les fenêtre ouvertes. Aller dans le panneau de configuration de Windows, cliquer sur Système et sécurité puis sur système et enfin sur paramètres système avancés. De là, cliquer sur Variables d'environnement.
   b. Cliquer sur 'nouveau' dans les variables d'environnement de l'utilisateur et créer une nouvelle variable SPARK_HOME ayant pour valeur le répertoire dans lequel spark a été installé, par exemple c:\spark.
   c. Créer une autre variable JAVA_HOME ayant pour valeur le répertoire où a été installé le jdk, par exemple c:\jdk.
   d. Enfin, créer une dernière variable HADOOP_HOME ayant pour valeur le répertoire dans lequel vous avez placé l'utilitaire winutils.exe, ce sera c:\winutils.
   e. Mettre à jour le PATH. Pour cela, dans les variables d'environnement système, rechercher la variable PATH. Sélectionner puis cliquer sur Editer et ajouter un nouveau chemin : %SPARK_HOME%\bin. Normalement, le jdk a déjà été rajouté au PATH, si ce n'est pas encore le cas, ajouter un autre chemin : %JAVA_HOME%\bin

4. Lancer PySpark
   a. Ouvrir l'invite de commande et lancer la commande c:\spark\bin\pyspark. Vous devriez obtenir quelque chose ressemblant à l'image ci-dessous :

Intégrer Jupyter Notebook

Il est possible de rendre l'application Jupyter Notebook compatible avec Spark de différentes manières. L'une d'entre elles consiste à créer un fichier de configuration et à lancer Jupyter Notebook avec cette configuration. Une autre consiste à importer PySpark dans le notebook au moment de l'exécution des cellules. Nous allons nous concentrer sur cette dernière approche afin de ne pas avoir à redémarrer Jupyter Notebook à chaque fois que nous voulons utiliser Spark.

1. Installer la bibliothèque Python findspark
    a. findspark est un package python permettant de rechercher l'emplacement de PySpark en utilisant la variable d'environnement SPARK_HOME que nous avons défini précédemment.
    b. Utiliser pip pour installer la bibliothèque findspark, lancer pip install findspark dans l'invite de commande
2. Tester que tout fonctionne correctement :
    a. Pour cela, télécharger le jeu de données recent-grade.csv dans votre répertoire de travail. Vous pouvez télécharger le fichier à partir de l'invite de commande en utilisant la commande curl lien-de-telechargement *(l'outil cURL doit au préalable avoir été installé sur la machine, si ne n'est pas le cas, vous pouvez le télécharger à l'adresse suivante)*
    b. Ouvrir un notebook jupyter dans ce même répertoire de travail et exécuter le code ci-après :

```
# retrouver le chemin vers PySpark.
import findspark
findspark.init()

# Importer PySpark et initialiser l'objet SparkContext.
import pyspark
sc = pyspark.SparkContext()

# Lire `recent-grads.csv` dans un RDD.
```

```
f = sc.textFile('recent-grads.csv')
data = f.map(lambda line: line.split('\n'))
data.take(10)
```

Si vous n'obtenez pas d'erreur et que vous pouvez voir les 10 premières lignes de 'recent-grads.csv', c'est que tout est correctement installé.
Si vous obtenez une erreur, vous pouvez vous aider de Google ou de StackOverflow pour trouver une solution en fonction du message d'erreur obtenu.

Partie II : Exploration de quelques méthodes de base PySpark

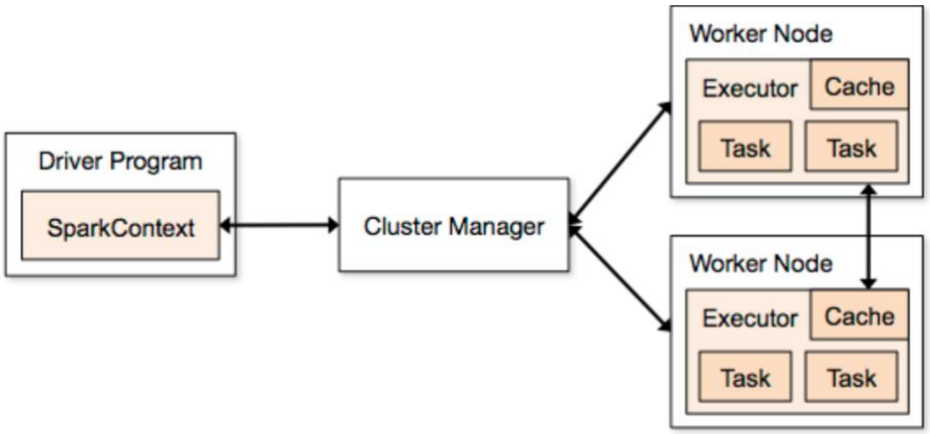
La structure de données centrale de Spark est RDD (resilient distributed dataset). Un RDD est la représentation par Spark d'un ensemble de données distribué dans la RAM d'un cluster de plusieurs machines. Un objet RDD est essentiellement une collection d'éléments que nous pouvons utiliser pour contenir des listes de tuples, des dictionnaires, des listes, etc.
À l'instar d'un DataFrame pandas, nous pouvons charger un jeu de données dans un RDD, puis exécuter n'importe laquelle des méthodes accessibles à cet objet.
Les RDD sont à la base des objets Java, la bibliothèque PySpark permet d'interfacer ces objets avec le langage Python.
Dans cette section, nous allons explorer les méthodes de base et quelques transformations disponibles dans Spark.

Le jeu de données

Nous allons expérimenter les différentes méthodes avec le fichier daily_show_guest.csv qui contient les noms de tous les invités ayant déjà participés à l'émission américaine The Daily Show.
Télécharger le jeu de données à partir du lien suivant.

1. Lire un fichier et l'afficher dans un RDD
    a. Lire le fichier daily_show_guest.csv dans un RDD nommé raw_data en utilisant la méthode textFile() de SparkContext.

Dans Spark, le SparkContext est un objet qui gère la connexion aux clusters et coordonne les processus sur ces clusters. Plus précisément, il se connecte aux gestionnaires de cluster. Ces derniers contrôlent les exécuteurs qui effectuent les calculs. Voici un diagramme tiré de la documentation Spark qui vous aidera à visualiser l'architecture :



Après avoir importé PySpark dans notre cellule de travail, nous instancions un objet sc avec la ligne de code :

```
sc = pyspark.SparkContext()
```

b. Afficher les cinq premiers éléments du RDD avec la méthode take().

2. Quelques méthodes de base

La principale chose à savoir lorsque l'on travaille avec Spark est le pipelining de données. Chaque opération ou calcul dans Spark est essentiellement une série d'étapes que nous pouvons enchaîner et exécuter successivement pour former un pipeline. Chaque étape du pipeline renvoie soit une valeur Python (comme un entier), soit une structure de données Python (comme un dictionnaire), soit un objet RDD.

Il y a 2 types de méthodes dans Spark

- Les transformations : map(), filter(), reduceByKey()
- Les actions : take(), saveAsTextFile(), count(), collect()

Les transformations sont des opérations paresseuses qui renvoient toujours une référence à un objet RDD.
Les actions sont les calculs effectués sur un objet RDD.
Spark n'exécute pas réellement les transformations, il attend jusqu'à ce qu'une action ait besoin d'utiliser le RDD résultant d'une transformation. Toute fonction qui renvoie un RDD est une transformation, et toute fonction qui renvoie une valeur est une action.

La fonction map()

La fonction map(f) applique une fonction f à chaque élément du RDD. Comme les RDD sont des objets itérables (comme la plupart des objets Python), Spark exécute la fonction f à chaque itération et renvoie un nouveau RDD.
Si vous regardez attentivement, vous verrez que raw_data est dans un format un peu difficile à travailler : on a une suite de chaines caractères (string) séparées par un délimiteur qui est une virgule. Bien que tous les éléments soient des chaînes de caractères, nous aimerions convertir chacun d'entre eux en une liste afin de rendre les données plus faciles à gérer. Pour cela, une façon classique de faire est de procéder de la manière suivante :

- Utilisez une boucle for pour parcourir la collection.
- Divisez chaque string sur le délimiteur.
- Stocker le résultat dans une liste.

Voici comment on fait la même chose avec la méthode map() :

- Appeler la fonction RDD map() pour spécifier que nous voulons appliquer la fonction entre parenthèses à chaque ligne de notre ensemble de données.
- Ecrire une fonction lambda qui divise chaque ligne en utilisant le délimiteur ‘,’ et affecter le RDD résultant à la variable daily_show.
- Appeler la fonction RDD take() sur daily_show pour afficher les cinq premiers éléments (ou lignes) du RDD résultant.

a. Écrire une fonction qui prend en entrée une ligne du RDD et qui divise chaque ligne sur le délimiteur ‘,’. Pour le faire vous pouvez utiliser une fonction lambda de la façon suivante : `lambda ligne : ligne.split(‘,’))`. La méthode split() renvoie une liste de string.
b. Appliquer la méthode map() au RDD raw_data et affecter le résultat à une variable daily_show. Dans les parenthèses de map(), appliquer la fonction précédemment écrite

La fonction reduceByKey()

Nous voulons compter le nombre d'invités qui ont participé au Daily Show au cours de chaque année. Si daily_show était une liste de listes, nous pourrions écrire le code Python suivant pour obtenir ce résultat :

```
tally = dict()
for line in daily_show:
 year = line[0]
 if year in tally.keys():
  tally[year] = tally[year] + 1
 else:
  tally[year] = 1
```

Les clés du dictionnaire tally sont les années et les valeurs sont les totaux du nombre de lignes associées à chaque année.

Pour obtenir le même résultat avec Spark, nous devrons utiliser une étape map(), puis une étape reduceByKey().

a. Dans une variable tally, appliquer au RDD daily_show une fonction map() qui prend une fonction lambda `lambda x : (x[0], 1)`
b. Chaîner avec la fonction reduceByKey() qui prend une fonction lambda `lambda x, y : x+y`.

Pendant l’étape du map(), nous utilisons une lambda fonction pour créer un tuple dont :

- les clés sont x[0], la première valeur de la liste
- les valeurs sont le chiffre 1

Après le map, Spark retiendra en mémoire une liste de tuples ressemblant à :

('YEAR', 1)
('1991', 1)
('1991', 1)
('1991', 1)
('1991', 1)
...

Ensuite, nous voulons réduire cela à :

('YEAR', 1)
('1991', 4)
...

La méthode reduceByKey(f) va combiner les tuples ayant des clés identiques en utilisant la fonction spécifiée entre parenthèses.

c. Afficher la variable tally avec la fonction python print(). Que remarquez vous ?

Pour voir les résultats de ces deux étapes, nous utiliserons la commande take(), qui force le code paresseux à s'exécuter immédiatement. Comme tally est un RDD, nous ne pouvons pas utiliser la fonction len() de Python pour connaître le nombre d'éléments de la collection. Au lieu de cela, nous devons utiliser la fonction RDD count().

d. Afficher le contenu de tally avec la ligne suivante : tally.take(tally.count())

La fonction filter()

Contrairement aux pandas, Spark ne connaît pas les en-têtes de colonnes et ne les a pas mis de côté. Nous avons besoin d'un moyen pour supprimer l'élément ('YEAR', 1) de notre collection. Nous aurons besoin d'une solution de contournement, car les objets RDD sont immutables une fois que nous les avons créés.

Les objets RDD sont immutables, ce qui signifie que nous ne pouvons pas modifier leurs valeurs une fois que nous les avons créés. En Python, les objets de type liste et dictionnaire sont mutables (nous pouvons modifier leurs valeurs), tandis que les objets de type tuple sont immuables. La seule façon de modifier un objet tuple en Python est de créer un nouvel objet tuple avec les mises à jour nécessaires. Spark utilise l'immutabilité des RDD pour améliorer la vitesse de calcul.

Spark dispose d'une fonction filter(f) qui crée un nouvel objet RDD en filtrant un objet RDD existant selon des critères spécifiques. Si nous spécifions une fonction f qui renvoie une valeur binaire, True ou False, le RDD résultant sera composé d'éléments pour lesquels la fonction a été évaluée à True. Pour en savoir plus sur la fonction filter(), consultez la documentation Spark.

a) Écrivez une fonction nommée filter_year que nous pouvons utiliser pour filtrer l'élément qui commence par le texte YEAR, au lieu d'une année réelle.
b) Appliquer cette fonction au RDD daily_show grâce à la méthode filter() et affecter son contenu à une variable filtered_daily_show.
c) Afficher les 5 premiers éléments du RDD avec la méthode take()
d) Dans la cellule de code suivante, nous allons filtrer les professions pour lesquelles l'occupation est vide, mettre en minuscules chaque profession, générer un histogramme des professions et sortir les cinq premiers tuples de l'histogramme. Pour cela, chainer dans l'ordre les fonctions filter(), map(), reduceByKey() et enfin take() sur l'objet filtered_daily_show.

Pour filtrer les professions, vous pouvez utiliser la lambda suivante : `lambda line: line[1] != ''`
Pour réduire la casse, vous pouvez utiliser la lambda suivante : `lambda line: (line[1].lower(), 1)`
Pour générer l'histogramme, vous pouvez utiliser la lambda suivante : `lambda x,y: x+y`

Cette question permet de démontrer les capacités de Spark. Le chainage de fonctions permet d'enchaîner une série de transformations de données dans un pipeline en laissant Spark gérer le tout en arrière-plan. Les développeurs qui ont écrit Spark ont spécifiquement conçu cette fonctionnalité et l'ont optimisée pour l'exécution de tâches successives.

Avant l'arrivée de Spark, l'exécution de nombreuses tâches successives dans Hadoop prenait énormément de temps. Hadoop devait écrire les résultats intermédiaires sur le disque et n'avait pas connaissance de l'ensemble du pipeline. Grâce à son approche agressive de l'utilisation de la mémoire et à son noyau bien conçu, Spark améliore considérablement le temps d'exécution d'Hadoop. (Si vous êtes curieux, vous pouvez en savoir plus sur ce sujet dans ce fil de discussion Quora).