MLLib Spark

Alexis Leloup

Stephan Locke

Simon Lecordier

Martin Thomas

Fiche DD

Unité de formation et de recherche en informatique
et électronique de l'université de Rennes 1
Université of Rennes 1

2 mars 2022

Table des matières

1 Introduction

Spark est un framework open source de calcul distribué développé à l'université de Californie. C'est un puissant moteur de traitement open-source, il permet des analyses d'importants volumes de données, avec des API en Java, Scala, Python, R et SQL, il est facile d'utilisation et largement utilisé. Spark exécute des programmes jusqu'à 100 fois plus vite que Hadoop [1] et MapReduce [2] en mémoire, et 10 fois plus vite sur disque.

Il peut être utilisé pour créer des applications de traitement de données en tant que librarie ou pour effectuer des analyses de données de manière interactive. Spark fourni de nombreuses bibliothèques dont SQL, DataFrames et Datasets, Mllib pour l'apprentissage automatique, GraphX pour le traitement des graphes et Spark Streaming.

De plus, Spark est multi plateforme et peut accéder à diverses sources de données, notamment HDFS, Apache Cassandra, Apache HBase et S3. [1]

Mllib est la librairie originale de Machine Learning pour Spark[2], et elle est construite sur les RDD [3] alors que Spark ML, une librairie plus récente est conçue sur les DataFrames. Dans ce sujet nous nous concentrerons sur la librairie MLlib de Spark. MLlib fonctionne particulièrement bien avec Python et R, elle interragit même avec NumPy. Cette API est majoritairement utilisée puisqu'elle est jusqu'à 110 fois plus rapide que Hadoop.

Elle est très complète puisqu'elle est capable de sous-traiter les principaux problèmes de Machine Leraning comme la classification, la régression, le clustering, la modélisation etc.[3] Tous les algorithmes de MLlib sont optimisés pour le calcul en parallèle sur un cluster.

Nous verrons quelques exemple expliqués pour comprendre au mieux ce framework. Nous avons utiliser MLlib via pyspark, qui est l'API de Spark pour Python. L'installation de hadoop est nécessaire pour utiliser cette librairie et permettre une gestion des fichiers.

1. Hadoop est un framework libre et open source écrit en Java destiné à faciliter la création d'applications distribuées et échelonnables permettant aux applications de travailler avec des milliers de nœuds et des pétaoctets de données.

2. MapReduce est un patron d'architecture inventé par Google dans lequel sont effectués des calculs parallèles et souvent distribués, de données potentiellement très volumineuses.

3. Resilient Distributed Dataset est une abstraction distribuée en mémoire qui permet aux développeurs d'effectuer des calculs parallèles en mémoire sur un cluster de façon complètement tolérante aux pannes.

2 Fonctionnement

Comme nous l'avons expliqué plus tôt, MLlib nous contraint d'utiliser des RDD. Pour mieux comprendre comment fonctionne MLlib il faudrait d'abord comprendre ce qu'est un RDD. Ce concept a été mis en place par les fondateurs de Spark, et représente des collections immutable, où chaque éléments est représenté en ligne. Par la suite on pourra faire deux type d'opérations sur ces fichiers, des transformations ou des actions comme un count.

Ce qui est intéressant avec cette librairie est le fait de stocker les jeux de données dans la RAM, ceci permet d'avoir des résultats très rapidement lorsque l'on va réaliser des calculs sur des volumétries importantes.

L'utilisation de ces algorithmes se fera en utilisant les trois types de données suivantes, les vecteurs, les labeledPoint et les Rating. Les vecteurs peuvent être représenté de deux façons, soit de manière dense en spécifiant directement les donnée soit en ne spécifiant uniquement les données non nulles, et leur place au sein du vecteur. Les labeledPoints sont utilisées dans l'apprentissage supervisée. Les rating sont des données utilisée pour le filtrage collaboratif, une technique visant à déterminer le degré d'interet d'un objet pour un individu en se absant sur l'interet d'autres individus. Nous nous concentrerons principalement sur l'utilisation des vectors. Il est important de préciser que ce type de donnée ne permet aucune opérations aritmétique.

L'entrainement d'un un modèle dépendra de l'algorithme utilisée ainsi que des parametres fournis.

Ce modèle entrainé permet pour prédire des valeurs grace à l'apprentissage sur le jeu de données.

3 Exemples

Le code associé à chaque exemple est disponible sur GitHub : `https://github.com/lealexio/Exemples_Fiche_DD`

3.1 Exemple 1

Le jeu de données utilisé pour cet article provient d'une campagnes de marketing téléphonique d'une institution bancaire portugaise. L'objectif de la classification est de prédire si le client va souscrire à un dépôt à terme. La première étape est d'importer les données :

```
spark = SparkSession.builder.appName("Example␣of␣classification").getOrCreate()
# Read a csv file into Spark DataFrame
# inferSchema = If True, type of column is automatically detected
# header = Read the first line of the CSV file as column names.
data_frame = spark.read.csv(r"bank_account.csv", inferSchema=True, header=True)
```

Listing 1 – Import du csv dans PySpark

Voici un extrait du dataframe obtenu :

```
DataFrame :
+---+-----------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+-------+
|age|        job| marital|education|default|balance|housing|loan|contact|day|month|duration|campaign|pdays|previous|poutcome|deposit|
+---+-----------+--------+---------+-------+-------+-------+----+-------+---+-----+--------+--------+-----+--------+--------+-------+
| 59|   director| married|secondary|     no|   2343|    yes|  no|unknown|  5|  may|    1042|       1|   -1|       0| unknown|    yes|
| 56|   director| married|secondary|     no|     45|     no|  no|unknown|  5|  may|    1467|       1|   -1|       0| unknown|    yes|
| 41| technician| married|secondary|     no|   1270|    yes|  no|unknown|  5|  may|    1389|       1|   -1|       0| unknown|    yes|
```

FIGURE 1 – Part of Dataframe

Ensuite on procède à un pre traitement de la donnée. Ce code est fourni par DataBrick, il indexe chaque colonne en utilisant StringIndexer, puis convertit les catégories indexées en variables uniques. C'est ici que l'on choisi la colonne cible, dans notre cas 'deposit'.

```
categoricalColumns = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'poutcome']
stages = []
for categoricalCol in categoricalColumns:
    stringIndexer = StringIndexer(inputCol=categoricalCol, outputCol=categoricalCol + 'Index')
    encoder = OneHotEncoder(inputCols=[stringIndexer.getOutputCol()], outputCols=[categoricalCol + "classVec"])
    stages += [stringIndexer, encoder]
label_stringIdx = StringIndexer(inputCol='deposit', outputCol='label')
stages += [label_stringIdx]
assemblerInputs = [c + "classVec" for c in categoricalColumns] + numericCols

# The VectorAssembler combines all the feature columns into a single vector column.
assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
stages += [assembler]
```

Listing 2 – Pre traitement des données

Ensuite on construit un pipeline d'apprentissage automatisé avec Spark. Un pipeline d'apprentissage automatique est un flux de travail complet combinant plusieurs algorithmes d'apprentissage automatique. Il peut y

avoir de nombreuses étapes nécessaires pour traiter et apprendre à partir des données, nécessitant une séquence d'algorithmes. Les pipelines définissent les phases et l'ordre d'un processus d'apprentissage automatique.

```
pipeline = Pipeline(stages=stages)
pipelineModel = pipeline.fit(data_frame)
df = pipelineModel.transform(data_frame)
```

Listing 3 – Création d'un pipeline

On divise aléatoirement le DataFrame avec les poids fournis avant de debuter des classifications.

```
# Randomly splits this DataFrame with the provided weights.
# seed=The seed for sampling
train, test = df.randomSplit([0.7, 0.3], seed=2018)

Dataset Count: 11162
Training Dataset Count: 7855
Test Dataset Count: 3307
```

Listing 4 – Création d'un pipeline

Enfin, on execute plusieurs classifieurs qui vont chacun retourner les probabilités d'adhésion des clients. Les colonnes suivantes sont ajoutées au dataset :

— rawPrediction est la sortie brute du classificateur par régression logistique (tableau de longueur égale au nombre de classes)

— probability est le résultat de l'application de la fonction logistique à rawPrediction (tableau de longueur égale à celle de rawPrediction)

— prediction est l'argument où la probabilité du tableau prend sa valeur maximale, et il donne l'étiquette la plus probable.

Puis on utilise un BinaryClassificationEvaluator pour evaluer la validitée des modèles :

```
Random-Forest Classifier Dataset :
      age          job  ...  prediction                                  probability
0      33   management  ...         0.0   [0.7460039012299077, 0.2539960987700923]
1      49   management  ...         0.0   [0.7460039012299077, 0.2539960987700923]
2      52   management  ...         1.0   [0.3199847168100235, 0.6800152831899765]
3      53   management  ...         0.0    [0.681788773854083, 0.3182112261459169]
4      58   management  ...         0.0  [0.7575318743006754, 0.24246812569932463]
...   ...          ...  ...         ...                                         ...
3306   41      unknown  ...         1.0   [0.1879373380260838, 0.8120626619739162]
[3307 rows x 6 columns]
Random-Forest Classifier score : 0.8755169298176253
```

Listing 5 – Random-Forest Classifier

```
DecisionTree Classifier Dataset :
      age          job  ...  prediction                                  probability
0      33   management  ...         0.0   [0.8385364216179926, 0.1614635783820074]
1      49   management  ...         0.0   [0.8385364216179926, 0.1614635783820074]
2      52   management  ...         1.0   [0.2344139650872818, 0.7655860349127181]
```

```
3       53   management   ...          0.0     [0.8385364216179926, 0.1614635783820074]
4       58   management   ...          0.0     [0.8385364216179926, 0.1614635783820074]
...    ...          ...   ...          ...                                          ...
3306    41      unknown   ...          1.0     [0.2153846153846154, 0.7846153846153846]
[3307 rows x 6 columns]
DecisionTree Classifier score : 0.7024457516569579
```

Listing 6 – DecisionTree Classifier

```
Gradient-Boosted Tree Classifier Dataset :
      age          job   ...  prediction                                   probability
0       33   management   ...         0.0     [0.966749327656039, 0.03325067234396095]
1       49   management   ...         0.0   [0.9907293490185795, 0.009270650981420547]
2       52   management   ...         1.0    [0.23886268163872085, 0.7611373183612792]
3       53   management   ...         0.0       [0.540802764318401, 0.459197235681599]
4       58   management   ...         0.0   [0.9693383429277272, 0.030661657072272752]
...    ...          ...   ...         ...                                          ...
3306    41      unknown   ...         1.0     [0.1997868075136707, 0.8002131924863293]
[3307 rows x 6 columns]
Gradient-Boosted Tree Classifier Score:  0.9005065692554148
```

Listing 7 – Gradient-Boosted Tree Classifier

Ici plusieurs classifieurs sont utilisés, on observe que Gradient Boosted Tree as le score le plus élevé, c'est donc théoriquement le modèle le plus fiable des trois présentés.

3.2 Exemple 2

Le deuxième exemple est beaucoup moin compliqué mais il utilisera un autre type de donnée dont nous avons parlé précedemment : les Ratings. L'objectif de cet exemple est de prédire la valeur que les utilisateurs choisirons pour des objets. Un exemple de Rating : 1,7,4.3 ici l'user 1 a rate l'item 7 à une valeur de 4.3.

On charge tout d'abord le dataset, dans notre cas généré grâce à un algorithme, nommé gen.py sur GitHub.

```
# Here we transform test.data into RDD
ratings = data.map(lambda l: l.split(',')) \
    .map(lambda l: Rating(int(l[0]), int(l[1]), float(l[2])))

# Construct model using Alternating Least Squares (ALS) and the data
rank = 10
numIterations = 10
model = ALS.train(ratings, rank, numIterations)
```

Listing 8 – Initiation des notations en fonction de l'ensemble des données et construire le modèle

Nous utilisons tous les RDD en excluant les ratings pour les prédire selon le modèle que nous avons généré précedemment.

```
# Predict thanks to the model
predictions = model.predictAll(testdata).map(lambda r: (r[0], r[1]))

# Evaluate model on training data
ratesAndPreds = ratings.map(lambda r: ((r[0], r[1]), r[2])).join(predictions)
MSE = ratesAndPreds.map(lambda r: (r[1][0] - r[1][1]) ** 2).mean()
print("Mean Squared Error = " + str(MSE))
```

Listing 9 – Predict using the training model

L'utilisation de ces données a pour but de prédire les ratings pour les users qui n'ont pas encore évalué certains items. Le modèle est entrainé sur les données des utilisateurs ayant choisi une valeur.Nous avons pu observé au cours de nos tests que les prédictions sont identique à celles de notre dataset si l'on évalue une seule fois un item. Si l'item à été évalué plusieurs fois alors la moyenne des valeurs est utilisée.

```
# Predict part
print(sameModel.predict(1, 4))
print(sameModel.predict(6, 1))

# the user 7 have rate the object 3 4 time
print(sameModel.predict(7, 3))
# compare if the algorithm is making an average on the user ratings item
print((7.1 + 7.8 + 5.7 + 4.2) / 4)
```

Listing 10 – Prédiction des évaluations utilisateurs depuis un modèle

3.3 Exemple 3

Dans le troisième exemple, on dispose d'un fichier csv regroupant différentes informations sur des véhicules automobiles. Le nom du véhivule, son nombre de cylindres, sa puissance, son poids ... etc.

L'objectif est d'utiliser MLLib afin de déterminer le couple d'un véhicule selon sa puissance, son poid et son nombre de cylindres. Évidemment il y a des manières plus simple de le faire, en faisant son rapport poids/puissance par exemple.

La première chose à faire et de récupérer les donnes et de les insérer dans un DataFrame. Cette tâche est effectuée grâce à un schéma, une sorte de mapping, qui va permettre d'interpréter les données de façon correcte. Cette opération cast les données String en Integer par exemple.

```
# Respect the csv file format
schema = StructType([car, mpg, cylinders, displacement, horsepower,
weight, acceleration, model, origin])
# Read the file following the scheme
df_pyspark = spark.read.option("delimiter", ";").option('header', 'true')
.option('inferSchema', 'true').schema(schema).csv('cars.csv')
df_pyspark.na.drop()
```

```
df_pyspark.printSchema()
```

Listing 11 – Load the csv file into DataFrame based on a specific scheme

Ensuite, on précise quelles données vont être interprétées pour réaliser la prédiction et sur quel champ la prédiction va être portée. On applique ensuite la règle de gestion linéaire, puis on lance le calcul.La donnée résultante est nommée "Independent Feature", elle est facilement comparable à la valeur réelle grâce à l'affichage du tableau.

```
+--------------------+------------+--------------------+
|Independent Feature|Acceleration|          prediction|
+--------------------+------------+--------------------+
| [3.0,100.0,2420.0]|        12.5|16.075300971637883|
| [3.0,110.0,2720.0]|        13.5|16.077972133950347|
|   [4.0,0.0,1835.0]|        17.3| 20.78304585288375|
|   [4.0,0.0,3035.0]|        20.5|23.426137138699712|
|  [4.0,46.0,1835.0]|        20.5| 17.75577822083274|
|  [4.0,48.0,2085.0]|        21.7|18.174801906882763|
|  [4.0,52.0,1649.0]|        16.5| 16.95123807604635|
|  [4.0,53.0,1795.0]|        17.5|17.207004016573144|
|  [4.0,60.0,2164.0]|        22.1|17.559083425562484|
|  [4.0,63.0,2051.0]|        17.0|17.112761831739025|
|  [4.0,63.0,2125.0]|        14.7|17.275752461031008|
|  [4.0,67.0,1850.0]|        13.8| 16.40680337770824|
|  [4.0,67.0,1963.0]|        15.5|16.655694473789243|
|  [4.0,67.0,1965.0]|        15.0|16.660099625932272|
|  [4.0,67.0,1995.0]|        16.2| 16.72617690807767|
|  [4.0,67.0,2000.0]|        16.0|16.737189788435238|
|  [4.0,70.0,2070.0]|        18.6| 16.69393961569871|
|  [4.0,70.0,2245.0]|        16.9|17.079390428213536|
|  [4.0,71.0,1925.0]|        14.0| 16.30875591941513|
|  [4.0,75.0,2108.0]|        15.5| 16.44858667684545|
+--------------------+------------+--------------------+
```

Listing 12 – Résultat du calcul de MLLib avec le jeu de données fourni

4 Bibliography

Références

[1] Denny Lee & Jules Damji, Apache Spark Key Terms, Explained, Jun. 22, 2016. Accessed on : Feb. 25, 2022. [Online]. Available : https://databricks.com/fr/blog/2016/06/22/apache-spark-key-terms-explained.html.

[2] Y. Benoit & A. Phelip, Les outils de la Data Science : Spark MLlib, théorie et concepts, 2015. Accessed on : Feb. 25, 2022. [Online]. Available : https://blog.engineering.publicissapient.fr/2015/05/11/les-outils-de-la-data-science-spark-mllib-theorie-et-concepts-12/.

[3] Microsoft, Documentation Azure HDInsight, Feb. 25 2022. Accessed on : Feb. 25, 2022. [Online]. Available : https://docs.microsoft.com/fr-fr/azure/hdinsight/spark/apache-spark-machine-learning-mllib-ipython.

[4] Juvenal JVC. Accessed on : Feb. 25, 2022. [Online]. Available : https://www.data-transitionnumerique.com/comprendre-rdd-spark/#:~:text=le%20RDD%2C%20Resilient%20Distributed%20Dataset,fa%C3%A7on%20compl%C3%A8tement%20tol%C3%A9rante%20aux%20pannes.