Florian Dargel : Framework de calcul distribué pyspark / algorithme de classification de sentiments.

Encadrant : Monsieur KOFFI Erwan

Le 18 mars 2022

Table des matières

1 Le sujet

À l'aide d'un Framework de calcul distribué de votre choix, entraîner un algorithme de classification de sentiments.

Trois ressources sont à votre disposition :

- Un fichier train.json contenant le dataset d'entraînement de l'algorithme.
- Un fichier test.json contenant le dataset de test de l'algorithme.
- Un fichier noclass.json sur lequel il faudra effectuer des prédictions.

Le rendu sera à envoyer à l'adresse erwan.koffi[at]gmail.com :

-Le code source de l'entraînement de l'algorithme et eventuellement la façon de l'exécuter.
-Le fichier noclass.json contenant la classification prédite par votre algorithme.
Une page décrivant votre méthode ansi qu'où intervient la distribution des calculs.

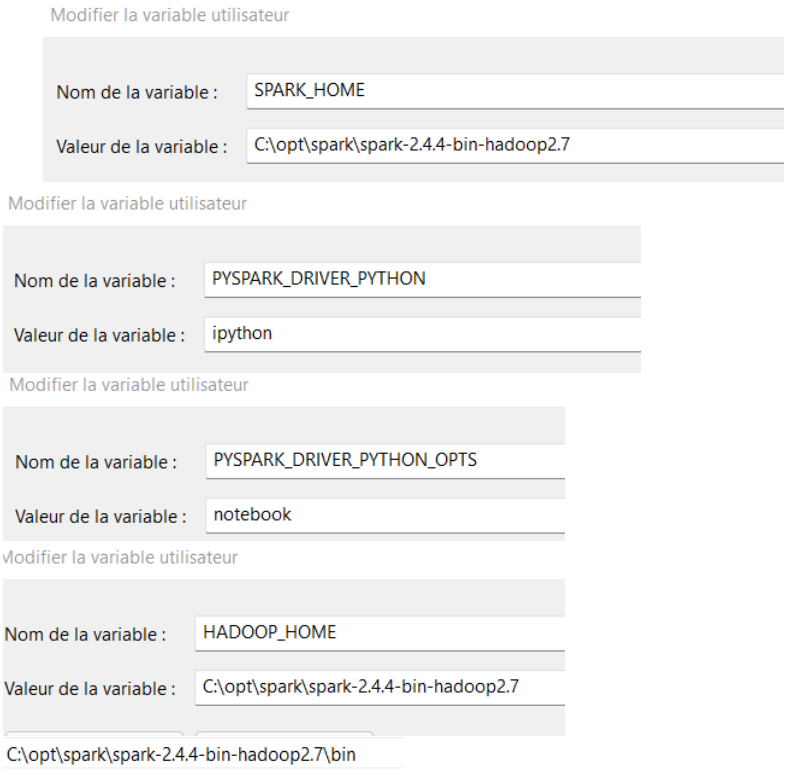
2 Le framerwork

2.1 La façon de l'executer

Dans mon terminal windows, j'ai installé Java, Scala, Anaconda et Spark.
Voici les étapes :
- Installer Anaconda
- Installer et vérifier l'installation de Java (utiliser la version 8 de Java)
- Installer et vérifier l'installation de Scala
- Télécharger et dézipper le fichier de Spark : spark-2.4.4-bin-hadoop2.7.tgz
- Ajouter au path les chemins vers l'environnement spark :
export PATH= $PATH :/usr/local/spark/bin$, si le fichier est dézippé dans /usr/local/spark.
- On peut lancer pyspark dans le terminal.

L'objectif étant d'utiliser les jupyter notebooks pour travailler sur l'environnement Spark. Pour cela, on modifie les options de lancement de pyspark de manière à lancer automatiquement un notebook lorsqu'on lance pyspark depuis le terminal.

2.2 Mes variables d'environnements



2.3 Les versions

J'utilise la version spark 2.4.4, Java 1.8.0_251 et scala 2.11.12

```
Entrée [1]: !pyspark --version
            executed in 1.76s, finished 16:50:24 2022-03-17
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.4
      /_/

Using Scala version 2.11.12, Java HotSpot(TM) Client VM, 1.8.0_251
Branch
Compiled by user  on 2019-08-27T21:21:38Z
Revision
Url
Type --help for more information.
```

J'utilise la version python 3.6.5

```
Entrée [1]: !python --version
            executed in 109ms, finished 22:04:13 2022-03-11
Python 3.6.5 :: Anaconda, Inc.
```

3 La méthode et l'intervention de la distribution des calculs.

3.1 L'importation de données

La distribution des calculs intervients dès l'importation de mon fichier train_json.

```
train_data = spark.read.json("C:/Users/na_to/OneDrive/Bureau/Insa/Mapromo/Calcul_distribue/Projet/train.json")
#4 partitions sur l'importation de données : voir http://localhost:4040 : données distribué RDD
```

Ce code génére 4 tasks correspondantes à l'importation du fichier, s'exécutant en parallèle et s'occupant chacune d'une partition.
On peut aperçevoir sur la Web UI de Spark. Il faut écrire sur l'onglet : **localhost :4040**

Aggregated Metrics by Executor

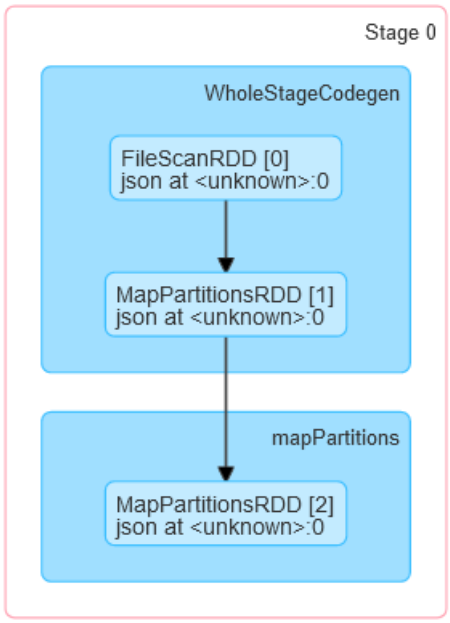
Executor ID	Address	Task Time	Total Tasks	Failed Tasks	Killed Tasks	Succeeded Tasks	Input Size / Records	Blacklisted
driver	MSI:51821	1 s	4	0	0	4	13.4 MB / 128401	false

Tasks (4)

Index	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	GC Time	Input Size / Records	Errors
0	0	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2022/03/11 22:04:25	0,3 s	76 ms	4.1 MB / 37255	
1	1	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2022/03/11 22:04:25	0,3 s	76 ms	4.1 MB / 39405	
2	2	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2022/03/11 22:04:25	0,3 s	76 ms	4.1 MB / 40078	
3	3	0	SUCCESS	PROCESS_LOCAL	driver	localhost	2022/03/11 22:04:25	0,3 s	65 ms	1273.7 KB / 11663	0

On peut notamment voir le temps d'exectution de chaque task. Dans un environnement idéal, chaque partition prendrait un temps similaire à être traitée. Or, dans la réalité, les traitements ressemblent le plus souvent à cela :

On peut notamment voir le schéma : Directed Acyclic Graph (DAG). On analyse les opérations en étapes. Les opérations de Stage0 sont :
1.FileScanRDD
2.MapPartitionsRDD



FileScan représente la lecture des données d'un fichier json.
MapPartitionsRDD est créé lorsqu'on utilise la transformation de partition de map

En comptant le nombre de classe de polarity (0 et 4 de notre variable cible), la distribution des calculs intervient aussi. On a 5 jobs supplémentaires qui se sont crées avec différents nombres de tasks.

```
train_data.groupby("polarity")\
      .count()\
      .show()
```

executed in 1.10s, finished 18:25:55 2022-03-17

```
+--------+-----+
|polarity|count|
+--------+-----+
|       0|61475|
|       4|66926|
+--------+-----+
```

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
5	showString at <unknown>:0 showString at <unknown>:0	2022/03/17 18:25:55	71 ms	1/1 (1 skipped)	75/75 (4 skipped)
4	showString at <unknown>:0 showString at <unknown>:0	2022/03/17 18:25:55	0,1 s	1/1 (1 skipped)	100/100 (4 skipped)
3	showString at <unknown>:0 showString at <unknown>:0	2022/03/17 18:25:55	38 ms	1/1 (1 skipped)	20/20 (4 skipped)
2	showString at <unknown>:0 showString at <unknown>:0	2022/03/17 18:25:55	12 ms	1/1 (1 skipped)	4/4 (4 skipped)
1	showString at <unknown>:0 showString at <unknown>:0	2022/03/17 18:25:55	0,3 s	2/2	5/5

On a qu'une seule partition lorsqu'on affiche les 20 première ligne de notre base de données avec **train_data.show()**

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
6	showString at <unknown>:0 showString at <unknown>:0	2022/03/17 18:36:31	16 ms	1/1	1/1

J'ai renommé la variable "polarity" par "label". J'ai ensuite mis cette variable en entier pour modéliser car auparavant il s'agissant d'une chaine de caractère "string".

3.2 Suppression des ponctuations

Il n'y a pas eu de distribution des calculs dans cette partie. Les tweet contienent beaucoup d'aléatoire qui nuit à l'estimation des modèles : les minuscules, les majuscules, les signes de ponctuation... On peut les garder mais plus de variabilité implique plus de données pour les apprendre. On préfère alors le nettoyer avant de le découper en mot (ou caractères ou syllabe). J'ai mis les tweet en minuscule et enlevé les ponctuations, par exemple : ",!?.

3.3 Ajout de variable

J'ai ajouté une nouvelle colonne qui consiste à compter le nombre de caractère par tweet, cette nouvelle colonne je l'ai nommé " length_of_message". Il sera présent lorsqu'on va modéliser notre modèle de classification.

3.4 Mots les plus fréquents

Il serait intéressant de regarder les mots qui interviennent le plus dans note base de données. La distribution des calculs intervients lorsqu'on souhaite regarder la fréquence des mots. On a 204 partitions.

```
]: topwords_train = wordCount(train_word).orderBy(['count'],ascending=False)
   topwords_train.show(50)
```

executed in 2.23s, finished 20:15:39 2022-03-17

```
+----------+-----+
|      word|count|
+----------+-----+
|         de|63026|
|         je|60897|
|         le|34041|
|         la|32126|
|          à|31028|
|        pas|28539|
```

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
9	showString at <unknown>:0 showString at <unknown>:0	2022/03/17 20:15:37	2 s	2/2	204/204

On peut enlever les mots les plus fréquents dont on suppose qu'il n'y aurait pas d'importance sur la classification de sentiments comme : de,je,la,à...
Ces mots seront supprimés par la suite avec la librairie "StopWordRemover".

Le calcul distributé intervient notamment lorsqu'on compte le nombre de mot unique dans notre base de données. On observe 235 partitions. On peut remarquer que 3 stages se sont crées avec succées contenant des nombres de tasks différents.

```
]: uniqueWordsCount = topwords_train.distinct().groupBy().count().head()[0]
   print(uniqueWordsCount)
```

executed in 5.12s, finished 20:39:38 2022-03-17

```
68107
```

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
11	head at <ipython-input-12-b674c6bad175>:1 head at <ipython-input-12-b674c6bad175>:1	2022/03/17 20:39:35	3 s	3/3 (1 skipped)	235/235 (4 skipped)

▾ Completed Stages (3)

Stage Id	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
21	head at <ipython-input-12-b674c6bad175>:1	+details	2022/03/17 20:39:38	18 ms	1/1			2003.0 B	
20	head at <ipython-input-12-b674c6bad175>:1	+details	2022/03/17 20:39:37	0,8 s	34/34			1374.5 KB	2003.0 B
19	head at <ipython-input-12-b674c6bad175>:1	+details	2022/03/17 20:39:35	2 s	200/200			2.0 MB	1374.5 KB

3.5 Transformation des tweet

Pour traiter les tweets, cela fonctionne en plusieurs étapes. On a tout d'abord la tokenisation. On utilise **Tokenizer**
La tokenisation est le processus consistant à prendre du texte (comme une phrase) et à le décomposer en termes individuels (généralement des mots).

On enlève ensuite les mots vides avec **StopWordsRemover**. Ce sont des mots qui doivent être exclus de l'entrée, généralement parce que les mots apparaissent fréquemment et n'ont pas autant de sens. Il est important de définir le language des tweets. Les tweets de la base de données sont majoritairement en français, on va donc supprimer les mots inutiles de la langue française.

On convertit ensuite les vecteurs de mots sous forme de valeur numérique pour la modélisation. On utilise **HashingTF**. HashingTF est un transformateur qui prend des ensembles de termes et convertit ces ensembles en vecteurs de caractéristiques de longueur fixe. Dans le traitement de texte, un "ensemble de termes" peut être un sac de mots. HashingTF utilise l'astuce de hachage. Une caractéristique brute est mappée dans un index (terme) en appliquant une fonction de hachage.

3.6 Preparation features

Les variables qu'on modélise sont la longueur des tweet et les vecteurs numériques des tweets qui on été converti. Ces variables sont appellé **lenght_of_message** et **transform_num**.

3.7 Modélisation

On fait intervenir la régression logistique qui est un algorithme de classification. On crée un Pipeline qui contient les variables à modéliser et le modèle de regression logistique.

Pour la modélisation il y a une centaine de job qui se sont créé en contenant chacun 4 partitions.

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
125	treeAggregate at RDDLossFunction.scala:61 treeAggregate at RDDLossFunction.scala:61	2022/03/17 22:10:36	5 s	1/1	4/4

On a la distribution des calculs qui interviennent notamment lorsqu'on sauvegarde notre modèle pour éviter de fit() une nouvelle fois.

```
: model_reg.save("C:/Users/na_to/OneDrive/Bureau/Insa/Mapromo/Calcul_distribue/Projet/logistic_model")
```

executed in 6.20s, finished 22:16:12 2022-03-17

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
129	parquet at LogisticRegression.scala:1241 parquet at LogisticRegression.scala:1241	2022/03/17 22:16:09	2 s	2/2	2/2
128	runJob at SparkHadoopWriter.scala:78 runJob at SparkHadoopWriter.scala:78	2022/03/17 22:16:08	0,3 s	1/1	1/1
127	runJob at SparkHadoopWriter.scala:78 runJob at SparkHadoopWriter.scala:78	2022/03/17 22:16:08	0,4 s	1/1	1/1
126	runJob at SparkHadoopWriter.scala:78 runJob at SparkHadoopWriter.scala:78	2022/03/17 22:16:06	1 s	1/1	1/1

On effectue les mêmes traitement du fichier train.json sur le fichier test.json.

Pour évaluer le modèle, on peut utiliser la librairie **BinaryClassificationEvaluator**, j'obtiens une valeur d'AUC de 0.54. La distribution des calculs intervients égalements avec plusieurs job et partitions.

La distribution des calculs intervient aussi pour le code suivant concernant le calcul de l'accuracy sur les données test.

```
|: correctpredictions = predictionsfinal.filter(predictionsfinal["prediction"] == predictionsfinal["label"]).count()
   totalData = predictionsfinal.count()
   print("correct prediction:", correctpredictions, ", total data:", totalData, ", accuracy:" ,correctpredictions/totalD
```

executed in 20.0s, finished 20:05:25 2022-03-18

```
correct prediction: 54259 , total data: 76759 , accuracy: 0.7068747638713375
```

3.8 Le fichier noclass.Json

Le modèle qui a été entrainé est appliqué sur le nouveaux jeux de données **noclass.Json**. Cette base de donnée ne contient pas la variable polarity mais uniquement les tweets. Le fichier **noclass_bis.Json** contient les tweets avec les valeurs qui ont été prédite par le modèle.