Rapport

[Dans chaque image, les commandaires en-desous des commands ne sont pas les frais résultats de chaque ligne de command. C'est just pour montrer la forme du résultat de chaque ligne de command.]

3

```
// spamFilter.scala

import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf
import org.apache.spark.rdd.RDD


  def probaWordDir(sc:SparkContext)(filesDir:String)
  :(RDD[(String, Double)], Long) = {
  val files = sc.wholeTextFiles(filesDir + "/*.txt").collect.toList
  val nbFiles = files.length
  val wordset = files.map(x => x._2.split("\\W+").distinct.toSet)
  // res25: List[scala.collection.immutable.Setobject spamFilter {
  // [String]] = List(Set(green, orange, yellow, red), Set(banana, orange, green, apple, kiwi, pear, red), Set(salad, potato, carrot, gr
  val wordDirOccurency = sc.parallelize(wordset.flatMap(x => x.map(word => (word,1)))).reduceByKey((x,y) => (x+y))
  // res26: Array[(String, Int)] = Array((kiwi,1), (yellow,1), (banana,1), (salad,1), (red,2), (pear,1), (orange,2), (apple,1), (green,3
  val probaWord = wordDirOccurency.map(x => (x._1, x._2.toDouble/nbFiles))
  // res31: Array[(String, Double)] = Array((kiwi,0.3333333333333333), (yellow,0.3333333333333333), (banana,0.3333333333333333), (salad,
  return (probaWord, nbFiles)
  }
```

a val files = sc.wholeTextFiles(filesDir + "/*.txt").collect.toList

on a déjà mis tous les fichiers de notre projet sous le répertoire "FilesDir"

Après la lecture de tous ces fichier à travers la commandes sc. wholeTextFiles

```
scala> val files = sc.wholeTextFiles("/tmp/ling-spam/ham/*.txt").collect.toList
16/10/28 19:44:43 INFO MemoryStore: ensureFreeSpace(216664) called with curMem=0, maxMem=278302556
16/10/28 19:44:43 INFO MemoryStore: Block broadcast_0 stored as values in memory (estimated size 211.6 KB, free 265.2 MB)
16/10/28 19:44:46 INFO MemoryStore: ensureFreeSpace(26021) called with curMem=216664, maxMem=278302556
16/10/28 19:44:46 INFO MemoryStore: Block broadcast_0_piece0 stored as bytes in memory (estimated size 25.4 KB, free 265.2 MB)
16/10/28 19:44:46 INFO BlockManagerInfo: Added broadcast_0_piece0 in memory on localhost:39165 (size: 25.4 KB, free: 265.4 MB)
16/10/28 19:44:47 INFO SparkContext: Created broadcast 0 from wholeTextFiles at <console>:21
16/10/28 19:45:02 INFO FileInputFormat: Total input paths to process : 2412
16/10/28 19:45:02 INFO FileInputFormat: Total input paths to process : 2412
```

```
p/ling-spam/ham/9-939msg1.txt:0+5628,/tmp/ling-spam/ham/9-940msg1.txt:0+2711,/tmp/ling-spam/ham/9-941msg1.txt:0+5525,/tmp/ling-
spam/ham/9-942msg1.txt:0+1055,/tmp/ling-spam/ham/9-944msg1.txt:0+5160,/tmp/ling-spam/ham/9-947msg1.txt:0+2123,/tmp/ling-spam/ha
m/9-948msg1.txt:0+2588,/tmp/ling-spam/ham/9-94msg1.txt:0+2713,/tmp/ling-spam/ham/9-94msg2.txt:0+6187,/tmp/ling-spam/ham/9-952ms
g1.txt:0+1855,/tmp/ling-spam/ham/9-955msg1.txt:0+3684,/tmp/ling-spam/ham/9-956msg1.txt:0+2562,/tmp/ling-spam/ham/9-957msg1.txt:
0+4463,/tmp/ling-spam/ham/9-958msg1.txt:0+6747,/tmp/ling-spam/ham/9-959msg1.txt:0+4045,/tmp/ling-spam/ham/9-960msg1.txt:0+1819,
/tmp/ling-spam/ham/9-961msg1.txt:0+5072,/tmp/ling-spam/ham/9-966msg1.txt:0+285,/tmp/ling-spam/ham/9-967msg1.txt:0+12496,/tmp/li
ng-spam/ham/9-968msg1.txt:0+7615,/tmp/ling-spam/ham/9-969msg1.txt:0+1112,/tmp/ling-spam/ham/9-969msg2.txt:0+5199,/tmp/ling-spam
/ham/9-970msg1.txt:0+893,/tmp/ling-spam/ham/9-971msg1.txt:0+1819,/tmp/ling-spam/ham/9-972msg1.txt:0+1715,/tmp/ling-spam/ham/9-9
73msg1.txt:0+3638,/tmp/ling-spam/ham/9-974msg1.txt:0+3257,/tmp/ling-spam/ham/9-975msg1.txt:0+1924,/tmp/ling-spam/ham/9-976msg1.
txt:0+2129,/tmp/ling-spam/ham/9-977msg1.txt:0+3200,/tmp/ling-spam/ham/9-981msg1.txt:0+7277,/tmp/ling-spam/ham/9-983msg1.txt:0+8
69,/tmp/ling-spam/ham/9-983msg2.txt:0+4457,/tmp/ling-spam/ham/9-984msg1.txt:0+4752,/tmp/ling-spam/ham/9-984msg2.txt:0+4579,/tmp
/ling-spam/ham/9-986msg1.txt:0+1068,/tmp/ling-spam/ham/9-987msg1.txt:0+3143,/tmp/ling-spam/ham/9-989msg1.txt:0+5238,/tmp/ling-s
pam/ham/9-98msg1.txt:0+2656,/tmp/ling-spam/ham/9-992msg1.txt:0+2524,/tmp/ling-spam/ham/9-995msg1.txt:0+2249,/tmp/ling-spam/ham/
9-996msg1.txt:0+673,/tmp/ling-spam/ham/9-997msg1.txt:0+436
```

b- val nbFiles = files.length

Le variable nbfiles contient le nombre des fichiers contenus dans "FilesDir"

c- val wordset = files.map(x => sc.parallelize(x._2.split("\\W+"))).distinct())

wordset est un rdd qui contient la liste des mots séparé pour chaque fichier à travers la commande split("\\W+"). on a ajouté ".distinct() " pour que chaque mot ne soit pas répéter.

d- val wordDirOccurency = sc.parallelize(a.flatMap(x => x.map(word => (word,1)))).reduceByKey((x,y) => (x+y))

wordDirOccurency est un rdd qui retourne pour chaque mot son nombre d'occurrence pour tous les fichiers.

e- val probaWord = wordDirOccurency.map(x => (x._1,(x._2/nbFiles).toFloat))

probaWord est un rdd qui contient la probabilité d'occurrence d'un mot.

```
def probaWordDir2Dirs(sc:SparkContext)(filesDir1:String, filesDir2:String)
:(RDD[(String, Double)], Long) = {

val files = sc.wholeTextFiles(filesDir1 + "/*.txt").collect.toList ::: sc.wholeTextFiles(filesDir2 + "/*.txt").collect.toList
val nbFiles = files.length
val wordset = files.map(x => x._2.split("\\W+").distinct.toSet)
// res25: List[scala.collection.immutable.Set[String]] = List(Set(green, orange, yellow, red), Set(banana, orange, green, apple, kiwi,
val wordDirOccurency = sc.parallelize(wordset.flatMap(x => x.map(word => (word,1)))).reduceByKey((x,y) => (x+y))
// res26: Array[(String, Int)] = Array((kiwi,1), (yellow,1), (banana,1), (salad,1), (red,2), (pear,1), (orange,2), (apple,1), (green,3
val probaWord = wordDirOccurency.map(x => (x._1, x._2.toDouble/nbFiles))
// res31: Array[(String, Double)] = Array((kiwi,0.3333333333333333), (yellow,0.3333333333333333), (banana,0.3333333333333333), (salad,
return (probaWord, nbFiles)
}
```

4

```
def computeMutualInformationFactor(
  probaWC:RDD[(String, Double)],  // word => probability the word occurs (or not) in an email of a given class
  probaW:RDD[(String, Double)],  // word => probability the word occurs (whatever the class)
  probaC: Double,  // the probability that an email belongs to the given class
  probaDefault: Double // default value when a probability is missing
  // When a word does not occur in both classes but only one, its probability P(occurs, class) must take on the default value probaDefault
):RDD[(String, Double)] = {

   probaW.leftOuterJoin(probaWC).mapValues {
    _._2 match {
      case Some(_) => probaWC * math.log(probaWC/(probaW*probaC))
      case None => probaDefault * math.log(probaDefault/(probaW*probaC))
    }
  }
}
```

cette fonction permet de définir la formule suivante :

$$P(occurs, class)\log_2\left(\frac{P(occurs, class)}{P(occurs)P(class)}\right)$$

5

```
  def main(args: Array[String]) {
    val (probaWordHam, nbFilesHam) : RDD[(String, Double)], Long) = probaWordDir(sc)("/tmp/ling-spam/ham")
    val (probaWordSpam, nbFilesSpam) : RDD[(String, Double)], Long) = probaWordDir(sc)("/tmp/ling-spam/spam")
    val (probaWord, nbFiles) : RDD[(String, Double)], Long) = probaWordDir(sc)("/tmp/ling-spam/ham", "/tmp/ling-spam/Spam")
    // formule
    val probaAbsentWordSpam = probaWordSpam.mapValues { 1.0 - _ }
    val probaAbsentWordHam = probaWordHam.mapValues { 1.0 - _ }
    // les 4 P(occurs, class)
    val probaPresentAndSpam = probaWordSpam.mapValues {_ * probaSpam}
    val probaPresentAndHam = probaWordHam.mapValues { _ * probaHam }
    val probaAbsentAndSpam = probaAbsentWordSpam.mapValues { _ * probaSpam }
    val probaAbsentAndHam = probaAbsentWordHam.mapValues { _ * probaHam }
    // P(occurs)
    val probaWordAbsent = probaWord.mapValues { 1.0 - _ }
    // P(class)
    val probaSpam =  nbFilesSpam.toDouble / nbFiles
    val probaHam = 1.0 - probaSpam
    // probaDefault
    val probaDefault = 0.2/nbFiles
    // word => MI(word)
    val MI = List(
      computeMutualInformationFactor(probaPresentAndSpam, probaWord, probaSpam, probaDefault),
      computeMutualInformationFactor(probaPresentAndHam, probaWord, probaHam, probaDefault),
      computeMutualInformationFactor(probaAbsentAndSpam, probaWordAbsent, probaSpam, probaDefault),
      computeMutualInformationFactor(probaAbsentAndHam, probaWordAbsent, probaHam, probaDefault)
    ).reduce {
      (term1, term2) => term1.join(term2).mapValues {
        case (l, r) => l + r
      }
    }
    // the 10 top words (maximizing the mutual information value)
    val resultat = MI.takeOrdered(10)(Ordering.by { _ => _._2 })
        .foreach { println }
    }
    resultat.saveAsTextFile("hdfs://tmp/ling-Spam/resultat.txt")
  }

} // end of spamFilter
```

a

val (probaWordHam, nbFilesHam) : RDD[(String, Double)], Long) = probaWordDir(sc)("/tmp/ling-spam/ham")

val (probaWordSpam, nbFilesSpam) : RDD[(String, Double)], Long) = probaWordDir(sc)("/tmp/ling-spam/spam")

Pour calculer les couples (probaWordHam, nbFilesHam) et (probaWordSpam, nbFilesSpam), on a appliqué la fonction probaWordDir de la question 3/ sur le répertoire contenant les fichiers ham "/tmp/ling-spam/ham" et le répertoire contenant les fichiers spam.

b-val probaAbsentWordSpam = probaWordSpam.mapValues { 1.0 - _ }

val probaAbsentWordHam = probaWordHam.mapValues { 1.0 - _ }

Dans les 2 premières lignes on a calculé la probabilité qu'un mot soit absent dans spam et dans ham.

val probaSpam = nbFilesSpam.toDouble / nbFiles

val probaHam = 1.0 - probaSpam

Puis on a calculé la probabilité qu'un mail soit spam "probaSpam" (qui est la division de nombre de fichier de spam par nombre de fichier total) et aussi la probabilité qu'un mail soit ham "probaHam"

```
val probaPresentAndSpam = probaWordSpam.mapValues {_ * probaSpam}

val probaPresentAndHam = probaWordHam.mapValues { _ * probaHam }

val probaAbsentAndSpam = probaAbsentWordSpam.mapValues { _ * probaSpam }

val probaAbsentAndHam = probaAbsentWordHam.mapValues { _ * probaHam }
```

Après on a calculé P(occur, class) pour les 4 combinaisons possible (true,ham), (true, spam), (false, ham) and (false, spam).Le résultat va être donc 4 Rdd.

Par exemple la probabilité qu'un mot soit présent (true) dans un mail sachant qu'il est spam est la multiplication de la valeur probaWordSpam trouvé auparavant avec la valeur probaSpam. C'est pour cela on a appliqué mapvalues avec dedans la fonction de multiplication. c'est le même principe pour les 3 autres probaPresentAndHam (true, ham), probaAbsentAndSpam(false,spam), probaAbsentAndHam(false,ham)

5-c

```
val MI = List(

    computeMutualInformationFactor(probaPresentAndSpam, probaWord, probaSpam, probaDefault),

    computeMutualInformationFactor(probaPresentAndHam, probaWord, probaHam, probaDefault),

    computeMutualInformationFactor(probaAbsentAndSpam, probaWordAbsent, probaSpam, probaDefault),

    computeMutualInformationFactor(probaAbsentAndHam, probaWordAbsent, probaHam, probaDefault)

  ).reduce {

    (term1, term2) => term1.join(term2).mapValues {

      case (a, b) => a + b

}
```

$$MI(w) = \sum_{\substack{occurs \in \{true,false\} \\ class \in \{spam,ham\}}} P(occurs, class) \log_2 \left(\frac{P(occurs, class)}{P(occurs)P(class)} \right)$$

5-d

```
MI.takeOrdered(10)(Ordering.by { _ => _._2 })
    .foreach { println }
```

cette commande affiche à l'écran les 10 mots qui ont le maximum de MI

5-e `resultat.saveAsTextFile("hdfs://tmp/ling-Spam/resultat.txt")`

Cette commande sert à sauvegarder les mots obtenus dans le fichier /tmp/resultat.txt