UNIVERSITY OF NICE, SOPHIA ANTIPOLIS

Project in Spark 2017

Adrianna Janik

Ion Mosnoi

Lei Guo

October 24, 2017

1 TASK

Firstly we uncompressed the data stored in ling-spam.zip folder with *Extract all* command. Secondly we open Virtual Box machine with Hortonworks, we signed in with maria_dev username and maria_dev password on Ambari available under 127.0.0.1:8080 ip address. We have selected *Files view*, than navigated to */tmp* folder and created directories *tmp/ling-spam/ham* and *ling-spam/spam*. Following that we logged in with ssh credentials to Hortonworks machine

```
$ssh root@127.0.0.1 -p 2222
```

In the meantime upload to the virtual machine ling-spam.zip with:

```
$sudo scp -P 2222 ../ling-spam.zip  root@127.0.0.1:/tmp/
```

We unzipped ling-spam.zip with:

```
$unzip ling-spam.zip -d /tmp/ling-spam
```

We putted files into /tmp/ling-spam/ folder in hdfs with:

```
$hdfs dfs -put ./ling-spam/ham /tmp/ling-spam/ham
$hdfs dfs -put ./ling-spam/spam /tmp/ling-spam/spam
```

2 Task

Installation of sbt:

```
$wget http://dl.bintray.com/sbt/rpm/sbt-0.13.12.rpm
```

Edit file /etc/yum.repos.d/sandbox.repo:

```
~[sandbox]
~name=Sandbox repository (tutorials)
~gpgcheck=0
~enabled=0
~baseurl=http://dev2.hortonworks.com.s3.amazonaws.com/repo/dev/
   ↪ master/utils/
```

```
$yum clean all
$yum update
$sudo yum localinstall sbt-0.13.12.rpm
$sbt -update
$sudo scp -P 2222 -r ../spamTopWords/*  root@127.0.0.1:/tmp/
   ↪ spamTopWords/
$sbt package
```

3 Task

Firstly we created Spark Context with:

```
val conf = new SparkConf(.setAppName(‘‘Spam Filter Application’’).
   ↪ setMaster(‘‘local’’)
val sc = new SparkContext(conf))
```

Than we called function *probaWordDir* with defined spark context as well as folder name for which we want to count words.

```
val (probaHW, nbHFiles) = probaWordDir(sc)(args(0)+"ham/*.txt")
print("number of files in "+ args(0)+"ham/*.txt" +":")
println(nbHFiles)


//process spam files
val (probaSW, nbSFiles) = probaWordDir(sc)(args(0)+"spam/*.txt")
print("number of files in "+ args(0)+"spam/*.txt" +":")
println(nbSFiles)
```

```
val rdd = sc.wholeTextFiles(filesDir)
// The number of files is counted and stored in a variable nbFiles
val nbFiles = rdd.count()
// Non informative words must be removed from the set of unique
   ↪ words.
```

```
val stopWords = Set(".", ":", ",", " ", "/", "\\", "-", "’", "(",
    ↪ ")", "@")
// Each text file must be splitted into a set of unique words
//(if a word occurs several times, it is saved only one time in
    ↪ the set).
val wordBagRdd: RDD[(String, Set[String])] = rdd.map(textTuple =>
        (textTuple._1, textTuple._2.trim().
        split("\\s+").toSet.diff(stopWords)))
// Get the Number of occurrences amongst all files
val wordDirOccurency: RDD[(String, Int)] = wordBagRdd.flatMap(
x => x._2.map(y => (y, 1))).reduceByKey(_ + _)
val probaWord: RDD[(String, Double)] = wordDirOccurency.map(
x => (x._1, x._2.toDouble / nbFiles))
return (probaWord, nbFiles)
```

4 ,5 Tasks

Function: probaWordDir:

```
def probaWordDir(sc:SparkContext)(filesDir:String)
:(RDD[(String, Double)], Long) = {


      val rdd = sc.wholeTextFiles(filesDir)
      // The number of files is counted and stored in a variable
         ↪ nbFiles
      val nbFiles = rdd.count()
      // Non informative words must be removed from the set of
         ↪ unique words.
      val stopWords = Set(".", ":", ",", " ", "/", "\\", "-", "’",
         ↪  "(", ")", "@")
      // Each text file must be splitted into a set of unique
         ↪ words (if a word occurs several times, it is saved
         ↪ only one time in the set).
      val wordBagRdd: RDD[(String, Set[String])] = rdd.map(
         ↪ textTuple =>
              (textTuple._1, textTuple._2.trim().
              split("\\s+").toSet.diff(stopWords)))
      // Get the Number of occurrences amongst all files
      val wordCountRdd: RDD[(String, Int)] = wordBagRdd.flatMap(x
         ↪ => x._2.map(y => (y, 1))).reduceByKey(_ +_)
      val probaWord: RDD[(String, Double)] = wordCountRdd.map(x =>
         ↪  (x._1, x._2.toDouble / nbFiles))
      return (probaWord, nbFiles)


}
```

Main function:

```scala
def main(args: Array[String]) {

    if(args.size > 0){
        val conf = new SparkConf().setAppName("Spam Filter Application").setMaster("local")
        val sc = new SparkContext(conf)
        println("Got the path:"+args(0))
        // args(0) should be something like "hdfs:///project/, see readme

        //process ham files
        val (probaHW, nbHFiles) = probaWordDir(sc)(args(0)+"ham/*.txt")

        //process spam files
        val (probaSW, nbSFiles) = probaWordDir(sc)(args(0)+"spam/*.txt")
        print("number of files in "+ args(0)+"ham/*.txt" +":")
        println(nbHFiles)
        print("number of files in "+ args(0)+"spam/*.txt" +":")
        println(nbSFiles)

        val nbFiles = nbSFiles + nbHFiles
        val probaW = probaSW.union(probaHW).reduceByKey((x,y) => (x*nbSFiles.toDouble+y*nbSFiles.toDouble)/(nbFiles.toDouble)) //not sure

        //Compute the probability P(occurs, class) for each word.

        val probaH = nbHFiles.toDouble / nbFiles.toDouble // the probability that an email belongs to the given class.
        val probaS = nbSFiles.toDouble / nbFiles.toDouble
        // Compute mutual information for each class and occurs
        val MITrueHam = computeMutualInformationFactor(probaHW, probaW, probaH, 0.2 / nbFiles) // the last is a default value
        val MITrueSpam = computeMutualInformationFactor(probaSW, probaW, probaS, 0.2 / nbFiles)
        val MIFalseHam = computeMutualInformationFactor(probaHW.map(x => (x._1, 1 - x._2)), probaW, probaH, 0.2 / nbFiles)
        val MIFalseSpam = computeMutualInformationFactor(probaSW.map(x => (x._1, 1 - x._2)), probaW,
                   ↪ probaS, 0.2 / nbFiles)

                //compute the mutual information of each word as a
                   ↪ RDD with the map structure: word => MI(word)
                //sum the prob for all words
                val MI :RDD[(String, Double)] = MITrueHam.union(
                   ↪ MITrueSpam).union(MIFalseHam).union(
                   ↪ MIFalseSpam).reduceByKey( (x, y) => x + y)

                // print on screen the 10 top words (maximizing the
                   ↪ mutual information value)
                //These words must be also stored on HDFS in the
                   ↪ file âĂIJ/tmp/topWords.txtâĂİ.
                val path: String = "/tmp/topWords.txt"
                val topTenWords: Array[(String, Double)] = MI.top
                   ↪ (10)(Ordering[Double].on(x => x._2))
                //coalesce to put the results in a single file
                sc.parallelize(topTenWords).keys.coalesce(1, true).
                   ↪ saveAsTextFile(path)
        }
        else
                println("Please write te directory where the ham and
                   ↪  spam")
}
```

Function: computeMutualInformationFactor

```
def computeMutualInformationFactor(
  probaWC: RDD[(String, Double)],//prob of just a class, some word
     ↪  could not be
  probaW: RDD[(String, Double)],//all words prob, all word
  probaC: Double, //prb of a class : class mails / all mails
  probaDefault: Double // default value when a probability is
     ↪ missing
): RDD[(String, Double)] = {
          //p(occurs) =
          val probWJoin: RDD[(String, (Double, Option[Double]))] =
             ↪  probaW.leftOuterJoin(probaWC)// got all class
             ↪ probs, if not -> default
                             //p(accurs)  p(accurs,class)
          val valueClassAndOcu: RDD[(String, (Double, Double))] =
             ↪ probWJoin.map(x => (x._1, (x._2._1, x._2._2.
             ↪ getOrElse(probaDefault))))
          //We have to change ln to log2 (by using ln(x)/ln(2)=
             ↪ log2(x)
          valueClassAndOcu.map(x => (x._1, x._2._2 * (math.log(x.
             ↪ _2._2 / (x._2._1 * probaC)) / math.log(2.0))))

}
```