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)
```

Function: probaWordDir:

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

      //sc -> class org.apache.spark.SparkContext
```

```
5         //filesDir -> java.lang.String
6
7         val rdd = sc.wholeTextFiles(filesDir)
8         //rdd -> class org.apache.spark.rdd.MapPartitionsRDD
9         // The number of files is counted and stored in a variable
              ↪ nbFiles
10        val nbFiles = rdd.count()
11        //nbFiles -> long
12        // Non informative words must be removed from the set of
              ↪ unique words.
13        val stopWords = Set(".", ":", ",", " ", "/", "\\", "-", "'",
              ↪  "(", ")", "@")
14        //stopWords -> class scala.collection.immutable.HashSet\
              ↪ $HashTrieSet
15        // 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).
16        val wordBagRdd: RDD[(String, Set[String])] = rdd.map(
              ↪ textTuple =>
17              (textTuple._1, textTuple._2.trim().
18              split("\\s+").toSet.diff(stopWords)))
19        //wordBagRdd -> class org.apache.spark.rdd.MapPartitionsRDD
20        // Get the Number of occurrences amongst all files
21        val wordCountRdd: RDD[(String, Int)] = wordBagRdd.flatMap(x
              ↪ => x._2.map(y => (y, 1))).reduceByKey(_ +_)
22        //wordCountRdd -> class org.apache.spark.rdd.ShuffledRDD
23        val probaWord: RDD[(String, Double)] = wordCountRdd.map(x =>
              ↪  (x._1, x._2.toDouble / nbFiles))
24        //probaWord -> class org.apache.spark.rdd.MapPartitionsRDD
25        return (probaWord, nbFiles)
26
27
28    }
```

4 Task

We computed function: computeMutualInformationFactor with given formula:

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

```
1   def computeMutualInformationFactor(
2     probaWC: RDD[(String, Double)],//prob of just a class, some word
          ↪  could not be
3     probaW: RDD[(String, Double)],//all words prob, all word
4     probaC: Double, //prb of a class : class mails / all mails
5     probaDefault: Double // default value when a probability is
          ↪ missing
```

```
6  ): RDD[(String, Double)] = {
7            //p(occurs) =
8            val probWJoin: RDD[(String, (Double, Option[Double]))] =
                 ↪  probaW.leftOuterJoin(probaWC)// got all class
                 ↪ probs, if not -> default
9                                //p(accurs)  p(accurs,class)
10     //probWJoin -> class org.apache.spark.rdd.MapPartitionsRDD
11           val valueClassAndOcu: RDD[(String, (Double, Double))] =
                 ↪ probWJoin.map(x => (x._1, (x._2._1, x._2._2.
                 ↪ getOrElse(probaDefault))))
12
13           valueClassAndOcu.map(x => (x._1, x._2._2 * (math.log(x.
                 ↪ _2._2 / (x._2._1 * probaC)) / math.log(2.0))))
14       //valueClassAndOcu -> class org.apache.spark.rdd.
             ↪ MapPartitionsRDD
15
16 }
```

probaWC is a RDD with the map structure: word => probability the word occurs in an email of a given class.
probaW has the map structure: word => probability the word occurs (whatever the class).
probaC is the probability that an email belongs to the given class.
probaDefault is a probability when a word does not occur in both classes but only one with value given by formula:
$\frac{0.2}{totalNumberOfFiles}$
This function returns the factor of each words (so it returns a RDD) given a class value (spam or ham) and an occurrence value (true or false).

5 Task

a. We computed the couples (probaWordHam, nbFilesHam) for the directory 'ham' and (probaWordSpam, nbFilesSpam) for the directory 'spam'.

b. We computed the probability P(occurs, class) for each word. There are two values of class ('ham' and 'spam') and two values of occurs ('true' or 'true'). Hence, we obtained 4 RDDs, one RDD for each case: (true,ham), (true, spam), (false, ham) and (false, spam). Each RDD has the map structure: word => probability the word occurs (or not) in an email of a given class.

c. We computed the mutual information of each word as a RDD with the map structure: word => MI(word). With the usage of the function computeMutualInformationFactor. If a word occurs in only one class, its joint probability with the other class takes on the default value probaDefault defined earlier. The function computeMutualInformationFactor is called 4 times for each possible value of P(occurs, class): (true,ham), (true, spam), (false, ham) and (false, spam).

d. The main function prints on screen the 20 top words (maximizing the mutual information value) which can be used to distinguish a spam from an ham email by using the mutual information.

We have obtained this list of words:
number of files in hdfs:///tmp/ling-spam/ham/*.txt:2412:

- (Subject:,1.0)
- (language,0.6737147595356551)
- (university,0.6048922056384743)
- (linguistic,0.5149253731343284)
- (information,0.45480928689883915)
- (“,0.43905472636815923)
- (’s,0.4369817578772803)
- (1,0.4253731343283582)
- (one,0.41376451077943616)
- (include,0.38930348258706465)
- (please,0.38225538971807627)
- (fax,0.37603648424543945)
- (http,0.3756218905472637)
- (e,0.37354892205638474)
- (english,0.36069651741293535)
- (2,0.3561359867330017)
- (;,0.3511608623548922)
- (address,0.35074626865671643)
- (follow,0.3490878938640133)
- (send,0.33996683250414594)

number of files in hdfs:///tmp/ling-spam/spam/*.txt:481

- (Subject:,1.0)
- (!,0.8295218295218295)
- (our,0.604989604989605)
- (free,0.5738045738045738)
- ($,0.5384615384615384)
- (please,0.5322245322245323)
- (’s,0.525987525987526)

- (mail,0.5031185031185031)
- (?,0.4968814968814969)
- (",0.49272349272349275)
- (one,0.4802494802494803)
- (0,0.4677754677754678)
- (address,0.45322245322245325)
- (*,0.45114345114345117)
- (list,0.4490644490644491)
- (receive,0.44282744282744285)
- (com,0.44282744282744285)
- (information,0.4386694386694387)
- (http,0.4386694386694387)
- (send,0.43451143451143454)
- print 10 words:
- (Subject:,NaN)
- (frye,23.82071883488019)
- (troubleshoot,23.82071883488019)
- (954,23.82071883488019)
- (rakyat,23.82071883488019)
- (telephony,23.82071883488019)
- (slap,23.82071883488019)
- (richey,23.82071883488019)
- (wales,23.82071883488019)
- (cake,23.82071883488019)

e. These top words are also stored on HDFS in the file '/tmp/topWords.txt'

Main function:

```
1  def main(args: Array[String]) {
2
3        if(args.size > 0){
4                val conf = new SparkConf().setAppName("Spam Filter
                     ↪ Application").setMaster("local")
5            //conf -> class org.apache.spark.SparkConf
6                val sc = new SparkContext(conf)
7            //sc ->  class org.apache.spark.SparkContext
8                println("Got the path:"+args(0))
```

```
9               // args(0) should be something like "hdfs:///project
                    ↪ /, see readme
10
11              //process ham files
12              val (probaHW, nbHFiles) = probaWordDir(sc)(args(0)+"
                    ↪ ham/*.txt")
13              //probaHW -> class org.apache.spark.rdd.
                    ↪ MapPartitionsRDD
14          //nbHFiles -> long
15              //process spam files
16              val (probaSW, nbSFiles) = probaWordDir(sc)(args(0)+"
                    ↪ spam/*.txt")
17          //probaSW -> class org.apache.spark.rdd.MapPartitionsRDD
18          // nbSFiles -> long
19              print("number of files in "+ args(0)+"ham/*.txt" +":
                    ↪ ")
20              println(nbHFiles)
21              print("number of files in "+ args(0)+"spam/*.txt" +"
                    ↪ :")
22              println(nbSFiles)
23
24              val nbFiles = nbSFiles + nbHFiles
25          //nbFiles -> long
26
27              val probaWs = probaSW.map(x => (x._1,(x._2,1))).
                    ↪ union(probaHW.map(x => (x._1,(x._2,0))))
28              //probaWs -> class org.apache.spark.rdd.UnionRDD
29              val probaW = probaWs.reduceByKey((x,y) => if(y._2<1)
                    ↪  ((x._1*nbSFiles.toDouble+y._1*nbHFiles.
                    ↪ toDouble)/(nbFiles.toDouble),1) else ((y._1*
                    ↪ nbSFiles.toDouble+x._1*nbHFiles.toDouble)/(
                    ↪ nbFiles.toDouble) ,0)) .map(x => (x._1,x._2._1
                    ↪ ))
30          //probaW -> class org.apache.spark.rdd.MapPartitionsRDD
31
32
33
34              //Compute the probability P(occurs, class) for each
                    ↪ word.
35
36              val probaH = nbHFiles.toDouble / nbFiles.toDouble //
                    ↪  the probability that an email belongs to the
                    ↪ given class.
37              //probaH -> double
38              val probaS = nbSFiles.toDouble / nbFiles.toDouble
39              //probaS -> double
40              // Compute mutual information for each class and
                    ↪ occurs
41              val MITrueHam = computeMutualInformationFactor(
```

```
                    ↪ probaHW, probaW, probaH, 0.2 / nbFiles) // the
                    ↪  last is a default value
42                  //MITrueHam -> class org.apache.spark.rdd.
                    ↪ MapPartitionsRDD
43                  val MITrueSpam = computeMutualInformationFactor(
                    ↪ probaSW, probaW, probaS, 0.2 / nbFiles)
44                  //MITrueSpam -> class org.apache.spark.rdd.
                    ↪ MapPartitionsRDD
45                  val MIFalseHam = computeMutualInformationFactor(
                    ↪ probaHW.map(x => (x._1, 1 - x._2)), probaW,
                    ↪ probaH, 0.2 / nbFiles)
46                  //MIFalseHam -> class org.apache.spark.rdd.
                    ↪ MapPartitionsRDD
47                  val MIFalseSpam = computeMutualInformationFactor(
                    ↪ probaSW.map(x => (x._1, 1 - x._2)), probaW,
                    ↪ probaS, 0.2 / nbFiles)
48                  //MIFalseSpam -> class org.apache.spark.rdd.
                    ↪ MapPartitionsRDD
49
50                  //compute the mutual information of each word as a
                    ↪ RDD with the map structure: word => MI(word)
51                  //sum the prob for all words
52                  val MI :RDD[(String, Double)] = MITrueHam.union(
                    ↪ MITrueSpam).union(MIFalseHam).union(
                    ↪ MIFalseSpam).reduceByKey( (x, y) => x + y)
53                  //MI -> class org.apache.spark.rdd.ShuffledRDD
54                  // print on screen the 10 top words (maximizing the
                    ↪ mutual information value)
55                  //These words must be also stored on HDFS in the
                    ↪ file âĂIJ/tmp/topWords.txtâĂİ.
56                  val path: String = "/tmp/topWords.txt"
57                  //path -> class java.lang.String
58                  val topTenWords: Array[(String, Double)] = MI.top
                    ↪ (10)(Ordering[Double].on(x => x._2))
59                  //topTenWords -> class [Lscala.Tuple2;
60                  //coalesce to put the results in a single file
61                  sc.parallelize(topTenWords).keys.coalesce(1, true).
                    ↪ saveAsTextFile(path)
62          }
63          else
64                  println("Please write te directory where the ham and
                    ↪  spam")
65  }
```