

Project in Spark 2017

Adrianna Janik

Ion Mosnoi

Lei Guo

October 25, 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 user-name 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

```
1 $ssh root@127.0.0.1 -p 2222
```

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

```
1 $sudo scp -P 2222 ../ling-spam.zip root@127.0.0.1:/tmp/
```

We unzipped ling-spam.zip with:

```
1 $unzip ling-spam.zip -d /tmp/ling-spam
```

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

```
1 $hdfs dfs -put ./ling-spam/ham /tmp/ling-spam/ham
2 $hdfs dfs -put ./ling-spam/spam /tmp/ling-spam/spam
```

2 TASK

Installation of sbt:

```
1 $wget http://dl.bintray.com/sbt/rpm/sbt-0.13.12.rpm
```

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

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

```
1 $yum clean all
2 $yum update
3 $sudo yum localinstall sbt-0.13.12.rpm
4 $sbt -update
5 $sudo scp -P 2222 -r ../spamTopWords/* root@127.0.0.1:/tmp/
   ↪ spamTopWords/
6 $sbt package
```

3 TASK

Firstly we created Spark Context with:

```
1 val conf = new SparkConf().setAppName('Spam Filter Application').
   ↪ setMaster('local')
2 val sc = new SparkContext(conf)
```

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

```
1 val (probaHW, nbHFiles) = probaWordDir(sc)(args(0)+"ham/*.txt")
2 print("number of files in "+ args(0)+"ham/*.txt" +":")
3 println(nbHFiles)
4
5
6 //process spam files
7 val (probaSW, nbSFiles) = probaWordDir(sc)(args(0)+"spam/*.txt")
8 print("number of files in "+ args(0)+"spam/*.txt" +":")
9 println(nbSFiles)
```

Function: *probaWordDir*:

```
1 def probaWordDir(sc:SparkContext)(filesDir:String)
2 :(RDD[(String, Double)], Long) = {
3
4     //sc -> class org.apache.spark.SparkContext
```

```

5      //filesDir -> java.lang.String
6      //read the files
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(".", ":", ",", " ", "/", "\\\"", "-", "'",
      ↳ "(", ")", "@", "Subject:")
14     //stopWords -> class scala.collection.immutable.HashSet\
      ↳ $HashTrieSet
15     // get the words in an email, delete the duplicate in one
      ↳ email, delete the stop words
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     // count the words in all emails
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     //calculate the probability
24     val probaWord: RDD[(String, Double)] = wordCountRdd.map(x =>
      ↳ (x._1, x._2.toDouble / nbFiles))
25     //probaWord -> class org.apache.spark.rdd.MapPartitionsRDD
26     return (probaWord, nbFiles)
27
28
29 }

```

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     //got (word,(prob for both classes, prob for class)),
    ↪ if the prob f    or class does not exist set the
    ↪ default
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    //calculate the formula for mutual information
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 'false'). 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 top 20 words:

- (bio,23.820986127869574)
- (touch-tone,23.820986127869574)
- (woodland,23.820986127869574)
- (8080,23.820986127869574)
- (ibi,23.820986127869574)
- (wales,23.820986127869574)
- (pearce,23.820986127869574)
- (slap,23.820986127869574)
- (commissioner,23.820986127869574)
- (n,23.820986127869574)
- (2442,23.820986127869574)
- (cake,23.820986127869574)
- (dawson,23.820986127869574)
- (detailed,23.820986127869574)
- (trilogy,23.820986127869574)
- (miranda,23.820986127869574)
- (piggy,23.820986127869574)
- (marke,23.820986127869574)
- (lightn,23.820986127869574)
- (pristine,23.820986127869574)

- 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
5             ↪ Application").setMaster("local")
6         //conf -> class org.apache.spark.SparkConf
7         //initiate spark context
8         val sc = new SparkContext(conf)
```

```

8      //sc -> class org.apache.spark.SparkContext
9      println("Got the path:"+args(0))
10     // args(0) should be something like "hdfs:///project
11     ↪ /, see readme
12
13     //process ham files
14     val (probaHW, nbHFiles) = probaWordDir(sc)(args(0)+"
15     ↪ ham/*.txt")
16     //probaHW -> class org.apache.spark.rdd.
17     ↪ MapPartitionsRDD
18 //nbHFiles -> long
19 //process spam files
20 val (probaSW, nbSFiles) = probaWordDir(sc)(args(0)+"
21 ↪ spam/*.txt")
22 //probaSW -> class org.apache.spark.rdd.MapPartitionsRDD
23 // nbSFiles -> long
24 print("number of files in "+ args(0)+"ham/*.txt" +":
25 ↪ ")
26 println(nbHFiles)
27 print("number of files in "+ args(0)+"spam/*.txt" +":
28 ↪ ")
29 println(nbSFiles)
30
31 val nbFiles = nbSFiles + nbHFiles
32 //nbFiles -> long
33
34 val probaWs = probaSW.map(x => (x._1,(x._2,1))).
35 ↪ union(probaHW.map(x => (x._1,(x._2,0))))
36 //probaWs -> class org.apache.spark.rdd.UnionRDD
37 val probaW = probaWs.reduceByKey((x,y) => if(y._2<1)
38 ↪ ((x._1*nbSFiles.toDouble+y._1*nbHFiles.
39 ↪ toDouble)/(nbFiles.toDouble),1) else ((y._1*
40 ↪ nbSFiles.toDouble+x._1*nbHFiles.toDouble)/(
41 ↪ nbFiles.toDouble) ,0)) .map(x => (x._1,x._2._1
42 ↪ ))
43 //probaW -> class org.apache.spark.rdd.MapPartitionsRDD
44
45
46 //Compute the probability P(occurs, class) for each
47 ↪ word.
48
49 val probaH = nbHFiles.toDouble / nbFiles.toDouble //
50 ↪ the probability that an email belongs to the
51 ↪ given class.
52 //probaH -> double
53 val probaS = nbSFiles.toDouble / nbFiles.toDouble
54 //probaS -> double
55 // Compute mutual information for each class and

```

```

    ↪ occurs
42 val MITrueHam = computeMutualInformationFactor(
    ↪ probaHW, probaW, probaH, 0.2 / nbFiles) // the
    ↪ last is a default value
43 //MITrueHam -> class org.apache.spark.rdd.
    ↪ MapPartitionsRDD
44 val MITrueSpam = computeMutualInformationFactor(
    ↪ probaSW, probaW, probaS, 0.2 / nbFiles)
45 //MITrueSpam -> class org.apache.spark.rdd.
    ↪ MapPartitionsRDD
46 val MIFalseHam = computeMutualInformationFactor(
    ↪ probaHW.map(x => (x._1, 1.00001 - x._2)),
    ↪ probaW, probaH, 0.2 / nbFiles)
47 //MIFalseHam -> class org.apache.spark.rdd.
    ↪ MapPartitionsRDD
48 val MIFalseSpam = computeMutualInformationFactor(
    ↪ probaSW.map(x => (x._1, 1.00001 - x._2)),
    ↪ probaW, probaS, 0.2 / nbFiles)
49 //MIFalseSpam -> class org.apache.spark.rdd.
    ↪ MapPartitionsRDD
50
51
52 println("print top MIFalseSpam prob:")
53 MIFalseSpam.top(10)(Ordering[Double].on(x => x._2)).
    ↪ foreach{ println }
54 println("print top MIFalseHam prob:")
55 MIFalseHam.top(10)(Ordering[Double].on(x => x._2)).
    ↪ foreach{ println }
56 println("print top MITrueSpam prob:")
57 MITrueSpam.top(10)(Ordering[Double].on(x => x._2)).
    ↪ foreach{ println }
58 println("print top MITrueHam prob:")
59 MITrueHam.top(10)(Ordering[Double].on(x => x._2)).
    ↪ foreach{ println }
60
61 //sum the mutual information
62 val MI :RDD[(String, Double)] = MITrueHam.union(
    ↪ MITrueSpam).union(MIFalseHam).union(
    ↪ MIFalseSpam).reduceByKey( (x, y) => x + y)
63 //MI -> class org.apache.spark.rdd.ShuffledRDD
64 //These words must be also stored on HDFS in the
    ↪ file "tmp/tmp/topWords.txt".
65 val path: String = "/tmp/topWords.txt"
66 //path -> class java.lang.String
67 val topTenWords: Array[(String, Double)] = MI.top
    ↪ (20)(Ordering[Double].on(x => x._2))
68 //topTenWords -> class [Lscala.Tuple2;
69 //save the top 20 words
70 sc.parallelize(topTenWords).keys.coalesce(1, true).

```

```
71         ↪ saveAsTextFile(path)
72     }
73     else
74         println("Please write te directory where the ham and
75         ↪ spam")
76 }
```
