2017 / 2018

Spark Project : spam filter

AJERDI Yassir, SHAN TianTian and VASSAL Véronique
UNIVERSITE DE NICE-SOPHIA-ANTIPOLIS

I- Introduction:

This report will explain how our Spam filter works based on the ling-spam data. The version of Hortonwork Sandbox used is 2.5.

The goal of this code is to analyze a lot of emails and print the top words that can distinguish a spam from a ham. We will create two functions: The first function is used to compute the probability that a word is in a certain class of emails, and the second one to compute the Mutual Information factor. In the end, we will come up with the top 20 words that are used in Spam and Ham emails.

II- Project description:

Let us start by decompressing and uploading our data files into the HDFS directory

1. create an HDFS directory called *ling-spam* in the *tmp* folder:

```
hdfs dfs –mkdir /tmp/ling-spam
```

2. Proceed to upload our zip file to the directory we just created:

```
scp –P 2222 ~/Downloads/ling-spam.zip root@localhost:/tmp/
```

3. Transfer the zip file to the HDFS directory:

```
hdfs dfs -put /tmp/lingspam.zip /tmp/ling-spam
```

4. Decompress the file in the HDFS directory:

```
hdfs dfs -cat /tmp/ling-spam/ling-spam.zip | gzip -d | hdfs dfs -put- /tmp/ling-spam
```

Now that all our email files are stored in the HDFS directory *ling-spam*, let us take a look at the function *probaWordDir*. Before everything, we have to read the files with SparkContext's whole text files method, i.e. sc.wholeTextFiles. and compute the number of files. Afterwards, we split each file word by word without taking non-informative words into account, thanks to the map method and the 'filter' method. We extract unique words from our RDD using the 'distinct' method, then we use flatMap to change the structure of the RDD to have (word,1) in order to compute the number of files in which a word occurs using the method reduceByKey(_+_), this works by incrementing by 1 every time the same word is found. Finally we have to compute the proportion for each word and return the result as (ProbaWord, nbFiles) which is of the type (RDD[(String, Double)], Long).

Last but not least, we complete the function computeMutualInformationFactor. The parameters of the function correspond to P(occurs,class), P(occurs) and P(class) that appear in the Mutual Information formula. The value occurs can either be True or False and class can be Ham or Spam. The last parameter is a default value, when a word is in only one class, its probability P(occurs,class) becomes the default value. ProbaW is an RDD[(String, Double)] that contains a word and the probability that the word occurs (whatever the class). ProbaWC: RDD[(String, Double)] contains a word and probability that the word occurs in a certain class, so each word of ProbaWC is included in ProbaW. However, the opposite is false. For that, we create a new RDD using leftOuterJoin method on probaWC because we want words which are not in ProbaWC and we use getOrElse method to change with the default value chosen. At the end we have to compute the Mutual Information Factor for each word and return the result as (word, MI) with the structure (RDD[(String, Double)].

Finally, we compute for the directory Ham et Spam, thanks to the probaWordDir function the three RDDs for the Spam, the Ham and finally all the emails. Then we want the computation for each case (when occurs =True or False and the two class) so we use the map method, to get the values for when occurs take False, so we compute 1 less the proportion estimated before for the two class and all emails. We store them in the RDDs probaWordHamFalse, probaWordSpamFalse and probaWF. The default value is 0.2 divided by the total number of mails.

Then we compute the Mutual Information factor, depending on the class and the occurrence of a word. So, we obtain four RDDs, MIF1, MIF2, MIF3 and MIF4. Nevertheless, to have the final Mutual Information for each word, we must sum two by two (MIF1 and MIF4 then MIF2 and MIF3). The 'union' method is used to obtain an RDD that contains all that information with the structure RDD[(String, Double)](but the length of the RDD is equal to the sum of the two RDDs), then we can reduce the two RDDs by key to obtain the MI factor of every word.

The last thing that we have to do is to sort the two RDDs in a descending order, seeing that we aim to maximize the Mutual Information Factor. The 'TakeOrdered' method is used for this purpose. Finally, we store these top words in the file topWord.txt thanks to the saveAsTextFile method as well as coalesce (to have the results in the same file) and parallelize (to adapt the type of the file to use the method saveAsTextFile). Then we print the file using the following command: # hdfs dfs -cat /tmp/ajerdi_shan_vassal/topWord.txt/part-00000

III- Interpretation:

Foremost, we launch the algorithm, the top 20 words in the class Spam are:

(**!**,3.7550540403469075) (**free**,2.9445709049818025) (**our**,2.662674230537142) (**remove**,2.422575209383467) (**$**,2.3325427234062834) (**money**,2.0960265398872155) (**com**,2.068758898763603) (**day**,2.053278504343074) (**mail**,2.016820724676392) (**want**,1.9394911049004424) (**over**,1.9331305975541744) (**receive**,1.9133704874135988) (**here**,1.8330043077660858) (**internet**,1.8249400603010764) (**best**,1.8060261898932417) (**check**,1.8045741374231883) (**today**,1.7997103815290572) (**offer**,1.7852217182232142) (**please**,1.781629196795636)

Thanks to these top words, we can decide, if a certain email is a ham or spam, because if you find a big frequency of these words in the email, it can be considered as a spam. In fact, these words appear rarely in the ham and they are also very frequent in spam.

Then we do the same thing for the class ham: we want to compare the list we find with the previously one. So, the program has return the following 20-top word in the class Ham:

(**language**,4.050418198705712) (**university**,3.7219505757055646) (**english**,3.2080627471647647) (**linguistics**,3.133860973253806) (**papers**,3.0929698996135824) (**conference**,3.0371634094712543) (**edu**,3.0243930527570235) (**de**,2.9961365460975435) (**discussion**,2.9780229463495704) (**speaker**,2.9728009739000285) (**department**,2.939677895) (**study**,2.9353495885323024) (**science**,2.9237854659489413) (**structure**,2.922166984756693) (**topic**,2.901167931702477) (**analysis**,2.894687230256977) (**research**,2.8731164871183466) (**committee**,2.8709888853194863) (**author**,2.8624277445337114)

The emails in the ling-spam dataset were received by a mailing list related to linguistics, hence the nature of the top words used in ham emails. All these words are highly relevant to people studying or doing research on linguistics, so it stands to reason that they will appear very frequently in regular mails. That's why, we can see, in the topWord, more words coming from the current language whereas in the other, topWordHam, they are rather words used in the school language. indeed, this is what can be found in most spam that we receive. However, some emails may contain other more common words like when you write to your family and yet not be spam.

Finally, we think that the algorithm can be performed, for example using a larger database of the two class. In the second part, we can use dataset more diversified, because certain email can be classified in spam whereas it is a ham email.