Big Data Technologies

Lionel Fillatre – Polytech Nice Sophia – 2016/2017

Spark Project

Rules:

- **Deadline:** October, Friday 28th at 22h00. Penalty of -1 per hour late.
- **Oral defense:** October, Monday 31th from 10h15 until 12h15 during the practical works.
- Practical works must be made in team of **3 students**. Only one team with more or less than 3 students is allowed.
- The report will be uploaded on Jalon website in "Spark Project - Deposit Box".
- The report must be uploaded in "pdf" format.
- You must write a report **presenting** the code (**question per question**), **explaining** the code **and interpreting the results**.
- The report must be named "student1_student2_student3.pdf".
- The full code as a unique file named "student1_student2_student3.scala" must be joined to the report in a zip file.
- The zip file must be named "student1_student2_student3.zip".
- You must use HDP 3.2, Scala and Spark. You **must not use** explicit loop (for, while, etc.) but you must use "map" and/or "reduce" and/ou "join" functions (flatMap, reduceByKey, fullOuterJoin, etc.).
- The **legibility of the report** will be taken into account.
- Penalty of -1 per unsatisfied rule.
- The report will be graded on a 0-15 scale. You must defend in group your project during 10 minutes to obtain the 5 remaining points. The defense will be graded individually (no discussion between the team members). The final mark will be graded on a 0-20 scale.

Context:

We will explore the Ling-Spam email dataset. The archive "ling-spam.zip" from the Jalon website contains two directories, spam/ and ham/, containing the spam and legitimate emails, respectively. The dataset contains 2412 ham emails and 481 spam emails, all of which were received by a mailing list on linguistics. We want to extract the words that are most informative of whether an email is spam or ham. This extraction is called the spam filter.

The first steps in any natural language processing workflow are to remove stop words and lemmatization. Removing stop words involves filtering very common words such as the, this and so on. Lemmatization involves replacing different forms of the same word with a canonical form: both colors and color would be mapped to color, and organize, organizing and organizes would be mapped to organize. Removing stop words and lemmatization is very challenging, and beyond the scope of this project. The Ling-Spam e-mail dataset has been already cleaned and lemmatized.

When we do build the spam filter, we will use the presence of a particular word in an email as the feature for our model. We will use a bag-of-words approach: we consider which words

appear in an email, but not the word order. Intuitively, some words will be more important than others when deciding whether an email is spam. For instance, an email that contains language is likely to be ham, since the mailing list was for linguistics discussions, and language is a word unlikely to be used by spammers. Conversely, words which are common to both message types, for instance hello, are unlikely to be much use.
One way of quantifying the importance of a word in determining whether a message is spam is the Mutual Information (MI). The mutual information is the gain in information about whether a message is ham or spam if we know that it contains a particular word. For instance, the presence of language in a particular email is very informative as to whether that email is spam or ham. Similarly, the presence of the word dollar is informative since it appears often in spam messages and only infrequently in ham messages. By contrast, the presence of the word morning is uninformative, since it is approximately equally common in both spam and ham messages. Consider a particular word *w* in an email. The email can belong to two classes: *spam* or *ham*. The word *w* can occur in the email or not. The mutual information *MI* of the word *w* whether that email is spam or ham is then defined by:

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

Materials:

1. The zip file "*ling-spam.zip*".
2. The file "*build.sbt*" for using sbt.
3. The template "*template.scala*" file.

Useful commands:

Some Scala commands could be useful: *wholeTextFiles, map, flatMapValues, mapValues, filter, case, reduce, reduceByKey, Ordering.by, fullOuterJoin, join, leftOuterJoin, getOrElse, math.log, toDouble, toSet*

These commands are described on spark.apache.org/docs/latest/api/scala/index.html

Questions:

You must complete the "*template.scala*" file by performing the following tasks. For each task, you must **describe carefully the type** returned by each command (and also the type of each element within the command).

1. Download the Ling-Spam email data set *"ling-spam.zip"* from the Jalon website. The data set is described on http://www.csmining.org/index.php/ling-spam-datasets.html
   The data set must be uncompressed and stored in the HDFS directory /tmp/ling-spam

2. You must code a self-contained application. To compile the *".scala"* file, you must install and use sbt:

   ```
   wget http://dl.bintray.com/sbt/rpm/sbt-0.13.12.rpm
   sudo yum localinstall sbt-0.13.12.rpm
   sbt -update
   ```

To use sbt please refer to https://spark.apache.org/docs/1.4.1/quick-start.html
The self-contained application is described in the section "Self-Contained Applications".
The directory of the project will be named *SpamFilter*. It will respect the typical layout of a sbt directory.
The sbt compilation produces a jar file which must be run with "*spark-submit*".

3. You must complete the function *probaWordDir*. It must return the couple (*probaWord*, *nbFiles*).
    a. This function reads all the text files within a directory named *filesDir*

    b. The number of files is counted and stored in a variable *nbFiles*

    c. 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).

    d. Non informative words must be removed from the set of unique words. The list of non-informative words is ".", ":", ",", " ", "/", "\", "-", "'", "(", ")", "@"

    e. For each word, you must count the number of files in which it occurs. This value must be stored in a RDD, named *wordDirOccurency*, with the map structure: word => number of occurrences of this word.

    f. Finally, you must compute the probability of occurrence of a word. The probabilities of all the words must be stored in the RDD, named *probaWord*, with the map structure: word => probability of occurrences of the word. For each word, this probability is computed as the ratio:
    ratio = number of occurrences of the word / nbFiles

4. You must complete the function "*computeMutualInformationFactor*". A factor is a term

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

in the mutual information formula. The variable *probaWC* is a RDD with the map structure: word => probability the word occurs (or not) in an email of a given class. The variable *probaW* has the map structure: word => probability the word occurs (whatever the class). The variable *probaC* is the probability that an email belongs to the given class. When a word does not occur in both classes but only one, its probability $P(occurs, class)$ must take on the default value *probaDefault*. This function returns the factor for all the words (so it returns a RDD) given a class value (spam or ham) and an occurrence value (true or false).

5. You must complete the function "main".
    a. You must compute the couples (*probaWordHam*, *nbFilesHam*) for the directory "ham" and (*probaWordSpam*, *nbFilesSpam*) for the directory "spam".

    b. You must compute 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, you must obtain 4 RDDs, one RDD for each couple of cases: (true,ham), (true, spam), (false, ham) and (false, spam).

c. You must compute the mutual information of each word as a RDD with the map structure: word => MI(word). This computation must exploit the function *computeMutualInformationFactor*. Warning! If a word only occurs in a single class, its joint probability with the other class must take on the default value = 0.2/(total number of files, including spam and ham) and not the zero value.

d. The main function must finally print on screen the 10 top words (maximizing the mutual information value) which can be used to distinguish a spam from an ham email by using the mutual information.

e. These words must be also stored on HDFS in the file "/tmp/topWords.txt".

6. Explain briefly (without any code) how we can exploit automatically these top words to design a classifier which can classify an email as ham or spam.