Big Data Technologies

Polytech Nice Sophia – 2017/2018

Spark Project

Rules:

- **Deadline:** October, Wednesday 25th at 22h00. Penalty of -1 per hour late.
- **Exam (30 minutes):** October, Monday 30th from 10h15 until 10h45 during the practical works.
- The project must be made in team of **3 students**. Only one team with more or less than 3 students is allowed per tutorial.
- The report with the full code must be uploaded as a unique zip file named "student1_student2_student3.zip".
- The zip file will be uploaded on the Jalon website in "Spark Project - Deposit Box".
- The report must be written 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".
- You must use HDP (at least version 2.3.2), Scala and Spark. You **must not use** explicit loop (for, while, etc.) but you must use "map" and/or "reduce" and/or "join" functions (flatMap, reduceByKey, fullOuterJoin, etc.).
- The **legibility of the report** will be considered.
- Penalty of -1 per unsatisfied rule.
- The exam will be graded on a 0-15 scale. The exam will contain precise questions on the Scala/Spark code necessary to make the project. You should use carefully the commands suggested in the following.
- The report together with the code will be graded on a 0-5 scale. Hence, 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. These words are called the top words.

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 are looking for the top words, 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. Consider a particular word *w*. The email can belong to two classes (*spam* or *ham*) and 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:

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

The goal of this project is to find the most informative words (the top words) which can be exploited to distinguish spam and ham emails.

Materials:

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

Useful commands:

Some Scala commands should be **useful** for the the **project** and the **exam**: *wholeTextFiles, map, flatMapValues, mapValues, filter, case, reduce, reduceByKey, Ordering.by, fullOuterJoin, join, leftOuterJoin, getOrElse, math.log, toDouble, toSet, takeOrdered, foreach, swap.*

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

Questions:

You must complete the "*spamTopWord.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). You must not create any new files, except the file containing the top words.

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 in the two subdirectories /tmp/ling-spam/ham and /tmp/ling-spam/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
   ```

   Warning: with certain HDP sandbox, you need to do the following steps before installing sbt:
   - Edit the following file

     ```
     vi /etc/yum.repos.d/sandbox.repo
     ```

     and be sure that "enabled=0" in this file
   - Then

     ```
     yum clean all
     yum 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 *spamTopWords*. 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. 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

   g. Warning, the probaWordDir will be called twice: the first time for the spam emails and the second time for the ham emails.

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 of each 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 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. 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 occurs in only one 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. The function *computeMutualInformationFactor* will be called 4 times for each possible value of *(occurs, class)*: (true,ham), (true, spam), (false, ham) and (false, spam).

   d. The main function must finally print 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.

   e. These top words must be also stored on HDFS in the file "/tmp/YOURNAME/topWords.txt" where YOURNAME is your family name.