Lab 5

The objective of this laboratory is to start playing around with Apache Spark. We provide you a template in Java, from which you can write your first application. As a first task, you will repeat the exercise you had in Lab 2, which is reported below for your convenience. In Lab 2 you used MapReduce. Now you must use Spark and RDDs.

The workflow for building is the same as the one we used with Hadoop: once unzipped the template, import and open the project in your IDE (Eclipse), modify it, and then export a jar. Then, submit this job either locally (i.e., on the PC of LABINF):

```
spark-submit  --class it.polito.bigdata.spark.example.SparkDriver
--deploy-mode client --master local SparkProject.jar arguments
```

or on the cluster (you must execute spark-submit on bigdatalab.polito.it to submit your application on the cluster):

```
spark2-submit  --class it.polito.bigdata.spark.example.SparkDriver
--deploy-mode cluster --master yarn SparkProject.jar arguments
```

Notice how in the template we did not, even if possible, configure the Spark master and other settings explicitly in the main class of the driver. In this way, you can always submit the same jar on any configuration, either Spark standalone locally or a Yarn cluster. The main advantage of this method is for testing your application: first you can try your application on your machine, on a sample file, and then safely submit the same application for the real work.

1. Filter a file

If you completed Lab 1, you should now have (at least one) huge files with the word frequencies in the amazon food reviews, in the format `word\tnumber`, where number is an integer (a copy of the output of Lab 1 is available in the HDFS shared folder /data/students/bigdata-01QYD/Lab2/). You should also have realized that inspecting these results manually is not feasible.
Your task is to write a **Spark** application to filter these results, and analyze the filtered data.
The filter you should implement is the following:

- keep only the lines containing words that start with a prefix (a string) that is passed as a command-line parameter

1. Run you application locally on the **LABINF PC** by using the following command and select only the lines containing the words starting with “ho” from the local file SampleLocalFile.csv (open a linux shell on the LABINF PC and execute the following command in the folder containing the jar file associated with your application and SampleLocalFile.csv)

```
spark-submit  --class it.polito.bigdata.spark.example.SparkDriver
--deploy-mode client --master local SparkProject.jar
SampleLocalFile.csv LocalOutputFolder "ho"
```

Analyze the content of the local output folder.

2. Run you application on the cluster by using the following command and select only the lines containing the words starting with "ho" from the content of the HDFS folder /data/students/bigdata-01QYD/Lab2/ (Use ssh to open a shell on the gateway bigdatalab.polito.it by using the usual procedure and then execute the following command)

```
spark2-submit  --class it.polito.bigdata.spark.example.SparkDriver
--deploy-mode cluster --master yarn SparkProject.jar
/data/students/bigdata-01QYD/Lab2/ HDFSOutputFolder "ho"
```

Analyze the content of the HDFS output folder.

How to access logs by using the job history web page

The logs of your applications on the cluster is available at https://ma1-bigdata.polito.it:18489/
If there is any error and you need to analyze the content of the error log files follow these steps:

1. Open a new terminal on your local machine (do **not** use the previous terminal connected to bigdatalab.polito.it).
2. On the local terminal command line, type the command
   `kinit sXXXXXX`
   (then, you will be asked for your BigDataLab password)
   This command is used to obtain a Kerberos ticket (without it you cannot access the log web page)
3. Now you should have access to the spark job history and logs at https://ma1-bigdata.polito.it:18489/
4. Select your job and click on its App ID
   You can use the search box to select only your jobs
5. Click on the executor tab and analyze the stderr associated with you application.
   Pay attention that there is one log file for the driver and one log file for each executor.
   If you application raised an exception, analyze all the stderr files to identify the reason of the error (depending on the error type, the log containing the needed information can be the one associated with the driver or that of an executor)

How to access logs without the web page and the Kerberos ticket

If you are connecting from outside Polito, and hence 1) you cannot request a Kerberos ticket and 2) you cannot access the history web page, you can proceed as follows to retrieve the log files from the command line:

1. Open a shell on the gateway by using ssh
   ssh smatricola@bigdatalab.polito.it

2. Execute the following command in the remote shell:
   yarn logs -applicationId *application_1521819176307_2195*
   The last parameter is the application/job ID. You can retrieve the job ID of your application on the HUE interface: https://bigdatalab.polito.it:8080/hue/jobbrowser#!jobs

Bonus questions

Explore the web dashboard of your jobs[1]. The history and the dashboard of the jobs can be accessed at the following URL https://ma1-bigdata.polito.it:18489/
Note that a Kerberos ticket is needed to access that web page. Use the same step reported above to obtain a Kerberos ticket and access the interface available at https://ma1-bigdata.polito.it:18489/

Check out the DAG of your application (under Jobs tab click on the saveAsTextFile at SparkDriver.java link to access the DAG) and the statistics for the application, and then answer the following questions:

- How many executors have been allocated to your job? (The information is available in the Executors tab)
- How was the input data split among them?
- Was your application well balanced among the executors, or were there tasks much slower/heavier/bigger than the others? Why?

Keep in mind these questions for the next labs as well.

[1] For local applications, the web dashboard is killed as soon as your job finishes. On the cluster, it is moved on a history server, so you can access it also after the job has completed.

Lab 6

In this lab, we continue our work on the Amazon dataset using Apache Spark. Your task is similar to that of Lab 3: given the original Amazon food dataset (that you can find in the HDFS file system at /data/students/bigdata-01QYD/Lab3/Reviews.csv), find all the pairs of items frequently reviewed together.
In the following Ex. 1 you find the steps that you are required to perform on the dataset.

Ex. 1

Write a single Spark application that:

- Transposes the original Amazon food dataset, obtaining a PairRDD of the type:

  <user_id> → <list of the product_ids reviewed by user_id>
- Counts the frequencies of all the pairs of products reviewed together;
- Writes on the output folder all the pairs of products that appear more than once and their frequencies. The pairs of products must be *sorted by decreasing frequency*.

The input Amazon food dataset (available in the HFDS shared folder of the BigData@Polito cluster: /data/students/bigdata-01QYD/Lab3/Reviews.csv) lists all the reviews per-row (one review per line), and is comma-separated. In each line, two of the columns represent the user id and product id. The schema of Reviews.csv is the following:
Id,ProductId,UserId,ProfileName,HelpfulnessNumerator,HelpfulnessDenominator,Score,Time,Summary,Text

Inspect the output of your application to search for interesting facts, and analyze the job execution as usual (performances, number of executors, etc…).

Pay attention that the line starting with “Id,” is the header of the file and must not be considered.

On the web site you can download the file ReviewsSample.csv. It contains a sample of Reviews.csv. You can use it to perform some initial tests locally.

Bonus task

Extend the implemented application in order to write on the standard output the top 10, most frequent, pairs and their frequencies.

Note that Spark 1.6, or above, provides the following actions that can be applied on an RDD of type JavaRDD<T>:
1) List<T> top(int n, java.util.Comparator<T> comp)
2) List<T> takeOrdered (int n, java.util.Comparator<T> comp)

top returns the **n largest elements** of the RDD based on the specified Comparator

takeOrdered returns **the n smallest elements** of the RDD based on the specified Comparator

Note that the standard output of the driver is stored in the log files of your application. Use the following steps top access the log files.

How to access logs by using the job history web page

The logs of your applications on the cluster is available at https://ma1-bigdata.polito.it:18489/

If there is any error and you need to analyze the content of the error log files follow these steps:

1. Open a new terminal on your local machine (do **not** use the previous terminal connected to bigdatalab.polito.it).
2. On the local terminal command line, type the command
   `kinit sXXXXXX`
   (then, you will be asked for your BigDataLab password)
   This command is used to obtain a Kerberos ticket (without it you cannot access the log web page)
3. Now you should have access to the spark job history and logs at https://ma1-bigdata.polito.it:18489/
4. Select your job and click on its App ID
   You can use the search box to select only your jobs
5. Click on the executor tab and analyze the stderr associated with you application.
   Pay attention that there is one log file for the driver and one log file for each executor.
   If you application raised an exception, analyze all the stderr files to identify the reason of the error (depending on the error type, the log containing the needed information can be the one associated with the driver or that of an executor)

How to access logs without the web page and the Kerberos ticket

If you are connecting from outside Polito, and hence 1) you cannot request a Kerberos ticket and 2) you cannot access the history web page, you can proceed as follows to retrieve the log files from the command line:

1. Open a shell on the gateway by using ssh
   ssh smatricola@bigdatalab.polito.it
2. Execute the following command in the remote shell:
   yarn logs -applicationId *application_1521819176307_2195*
   The last parameter is the application/job ID. You can retrieve the job ID of your application on the HUE interface: https://bigdatalab.polito.it:8080/hue/jobbrowser#!jobs

Lab 7

In this lab, we analyze historical data about the stations of the bike sharing system of Barcelona. Your task consists in identifying the most "critical" timeslot (day of the week, hour) for each station and store the result in a KML file that can be visualized on a map.

The analysis is based on two files (available in the HFDS shared folder of the BigData@Polito cluster):

1. /data/students/bigdata-01QYD/Lab7/register.csv
2. /data/students/bigdata-01QYD/Lab7/stations.csv

1) register.csv contains the historical information about the number of used and free slots for ~3000 stations from May 2008 to September 2008. Each line of register.csv corresponds to one reading about the situation of one station at a specific timestamp. Each line has the following format:
   - *stationId***\t***timestamp***\t***usedslots***\t***freeslots*
     - where *timestamp* has the format *date time*

   For example, the line

   23 2008-05-15 19:01:00 5 13

   means that there were **5** used slots and **13** free slots at station **23** on **May 15, 2008** at **19:01:00**.

   Pay attention that the first line of register.csv contains the header of the file. Moreover, some of the lines of register.csv contain wrong data due to temporary problems of the monitoring system. Specifically, some lines are characterized by used slots = 0 and free slots = 0. Those lines must be filtered before performing the analysis.

2) stations.csv contains the description of the stations. Each line of registers.csv has the following format:
   - *stationId***\t***longitude***\t***latitude***\t***name*

   For example, the line

   1 2.180019 41.397978 Gran Via Corts Catalanes

   contains the information about station **1**. The coordinates of station 1 are 2.180019,41.397978 and its name is **Gran Via Corts Catalanes**.

Ex. 1

Write a single Spark application that identifies the most "critical" timeslot for each station. This analysis can support the planning of the rebalancing operations among stations. **Solve the problem by using RDDs**. **Do not use Datasets, DataFrames, and the other Spark SQL features in this practice** (you will use them to solve this problem during the next practice).

In this application, each pair "day of the week – hour" is a timeslot and is associated with all the readings associated with that pair, independently of the date. For instance, the timeslot "Wednesday - 15" corresponds to all the readings made on Wednesday from 15:00:00 to 15:59:59.

A station Si is in the critical state if the number of free slots is equal to 0 (i.e., the station if full).

The "criticality" of a station Si in the timeslot Tj is defined as

$$\frac{number\ of\ readings\ with\ num.of\ free\ slot\ equal\ to\ 0\ for\ the\ pair\ (Si,Tj)}{total\ number\ of\ readings\ for\ the\ pair\ (Si,Tj)}$$

Write an application that:

- Computes the criticality value for each pair (Si,Tj).
- Selects only the pairs having a criticality value greater than a minimum criticality threshold. The minimum criticality threshold is an argument of the application.
- Selects the most critical timeslot for each station (consider only timeslots with a criticality greater than the minimum criticality threshold). If there are two or more timeslots characterized by the highest criticality value for a station, select only one of those timeslots. Specifically, select the one associated with the earliest hour. If also the hour is the same, consider the lexicographical order of the name of the week day.
- Stores in one single KML file the information about the most critical timeslot for each station. Specifically, the KML file must contain one marker of type Placemark for each pair (Si, most critical timeslot for Si) characterized by the following features:
    - StationId
    - Day of the week and hour of the critical timeslot
    - Criticality value
    - Coordinates of the station (longitude, latitude)

  Do not include in the KML file the stations for which there are no timeslots satisfying the minimum criticality threshold.

The output KML file has the following format:

```
<kml xmlns="http://www.opengis.net/kml/2.2"><Document>
        <Placemark><name>44</name><ExtendedData><Data
        name="DayWeek"><value>Mon</value></Data><Data
        name="Hour"><value>3</value></Data><Data
        name="Criticality"><value>0.5440729483282675</value></Data></ExtendedData><
        Point><coordinates>2.189700,41.379047</coordinates></Point></Placemark>
        <Placemark><name>9</name><ExtendedData><Data
        name="DayWeek"><value>Sat</value></Data><Data
        name="Hour"><value>10</value></Data><Data
        name="Criticality"><value>0.5215827338129496</value></Data></ExtendedData><
        Point><coordinates>2.185294,41.385006</coordinates></Point></Placemark>
</Document></kml>
```

The KML file is composed of the three sections:

1. Header
2. One line for each marker of type Placemark
3. Footer

The template of the project available for this lab includes:

- The class DateTool characterized by the static String DayOfTheWeek(String timestamp) method. The provided method returns the day of the week of the timestamp provides as parameter. For instance the instruction DateTool.DayOfTheWeek(" 2017-05-12") returns the string "Fri".
- The code that is needed to write a HDFS file (at the end of the main method of SparkDriver.java). You cannot use the standard approach based on saveAsTextFile() because the content of the output file is obtained concatenating three different parts in the proposed order:
    1) The fix header of the output KML file.
    2) One line for each marker. The content of this central part is associated with the content of the JavaRDD<String> containing the result of your elaboration.
    3) The fix footer of the output KML file.

The content of the KML file can be used to show on a Google map the stations characterized by a high criticality value and the associated information. You can visualize the result of you analysis on a map by copy and paste the content of the generated KML file in the form of the following web page http://display-kml.appspot.com/ or you can use one of the many other KML viewers available online.

Lab 8

In this lab, we analyze historical data about the stations of the bike sharing system of Barcelona. The data are the same you already analyzed during the previous practice, as well as the goal of your task: computing the “criticality” for the pairs (station, timeslot) and select the most critical ones. However, in this practice you are requested to exploit different BigData approaches.

The analysis is based on 2 files available in the HFDS shared folder of the BigData@Polito cluster:

1. /data/students/bigdata-01QYD/Lab7/**register.csv**
2. /data/students/bigdata-01QYD/Lab7/**stations.csv**

1) **register.csv** contains the historical information about the number of used and free slots for ~3000 stations from May 2008 to September 2008. Each line of register.csv corresponds to one reading about the situation of one station at a specific timestamp. Each line has the following format:
   - *station***\t***timestamp***\t***used_slots***\t***free_slots*

   For example, the line

   23 2008-05-15 19:01:00 5 13

   means that there were **5** used slots and **13** free slots at station **23** on **May 15, 2008** at **19:01:00**.

   The first line of register.csv contains the header of the file.

   Pay attention that some of the lines of register.csv contain wrong data due to temporary problems of the monitoring system. Specifically, some lines are characterized by used_slots = 0 and free_slots = 0. Those lines must be filtered before performing the analysis.

2) **stations.csv** contains the description of the stations. Each line of registers.csv has the following format:
   - *id***\t***longitude***\t***latitude***\t***name*

   For example, the line

   1 2.180019 41.397978 Gran Via Corts Catalanes

   contains the information about station **1**. The coordinates of station 1 are 2.180019,41.397978 and its name is **Gran Via Corts Catalanes**.

Ex. 1

Write a single Spark application that selects the pairs (station, timeslot) that are characterized by a high “criticality” value. The first part of this practice is similar to the one that you already solved during the previous practice. However, in this case you are requested to **solve the problem by using two different sets of Spark SQL APIs.**

(i) Implement **a first version** of the application by using **typed Datasets** whenever possible, and the associated type-safe transformations, i.e., map, filter, etc.

(ii) Implement **a second version** of the application based on the use of **SQL queries** in the Spark application**,** i.e., **SparkSession.sql(“SELECT …”).**

In this application, each pair “day of the week – hour” is a timeslot and is associated with all the readings associated with that pair, independently of the date. For instance, the timeslot “Wednesday - 15” corresponds to all the readings made on Wednesday from 15:00:00 to 15:59:59.

A station Si is in the critical state if the number of free slots is equal to 0 (i.e., the station if full).
The “criticality” of a station Si in the timeslot Tj is defined as

$$\frac{number\ of\ readings\ with\ num.of\ free\ slot\ equal\ to\ 0\ for\ the\ pair\ (Si,Tj)}{total\ number\ of\ readings\ for\ the\ pair\ (Si,Tj)}$$

Write two versions (based on typed Datasets and SQL queries, respectively) of an application that:

- Computes the **criticality value** for each pair (Si,Tj).
- Selects only the pairs having a criticality value greater than a minimum criticality threshold. The **minimum criticality threshold** is an argument of the application.
- **Join** the content of the previous selected records with the content of stations.csv to retrieve **the coordinates of the stations**.
- Store in the output folder the selected records, by using **csv files (with header)**. Store only the following attributes:
    - station
    - day of week
    - hour
    - criticality
    - station longitude
    - station latitude
- Store the results **by decreasing criticality**. If there are two or more records characterized by the same criticality value, consider the station (in ascending order). If also the station is the same, consider the day of the week (ascending) and finally the hour (ascending).

Note that:

- The provided template includes the class DateTool characterized by the following static methods:
  - **`String DayOfTheWeek(String timestamp)`**
    The provided method returns the day of the week of the String containing a timestamp provides as parameter.
    For instance, the instruction DateTool.DayOfTheWeek("2017-05-12") returns the string "Fri".
  - **`String DayOfTheWeek(java.sql.Timestamp timestamp)`**
    This method returns the same information returned by the other method but the parameter of this version of the method is a java.sql.Timestamp object.
  - **`int hour(Timestamp timestamp)`**
    This method returns the hour information associated with the input timestamp.

- The SQL-like language available in Spark SQL is characterized by a predefined function called **hour(timestamp)** that can be used in the SQL queries, or in the **selectExpr** transformation, to select the “hour part” of a given timestamp.
  The **DateTool.hour(Timestamp timestamp)** is needed only in the map transformations associated with typed Datasets.

- In order to specify that the separator of the input CSV files is “tab”, set the delimiter option to **\\t**, i.e., invoke **.option("delimiter", "\\t")** during the reading of the input data.

Lab 9

In this laboratory, we will apply some classification algorithms to the Amazon fine-foods dataset we have been exploring so far, exploiting SparkSQL to create the input DataFrame of the machine learning algorithms.

Your goal is to produce a classification model able to label reviews as “useful” or “useless” (i.e., you will produce a binary classification model). For each review on Amazon.com, users are able to vote the helpfulness of such review, with a thumb up/down. Our dataset provides this information through two columns: the number of users that have voted the content of a review (helpfulness denominator) and the number of users that have thumbed up (helpfulness numerator). The helpfulness index of a review is given by the ratio of the two, and it can be computed only for the reviews that have been voted at least one time. For this task, a review belongs to the “useful” class if its helpfulness index is above 90% (0.9). Otherwise, the review is labeled as “useless”. We are interested in predicting the class label (useful or useless) of the reviews that have never been voted in order to automatically predict if they are useful or useless.

There are different ways to evaluate the quality of the built classification model. Here, we choose one of the simplest: we divide the labeled input dataset (i.e., the reviews for which the label is known) in two splits, we train the dataset on the first one and then we test it on the second part, computing the average precision of the result.

For your ease, you are provided of a template (Lab9_Template.zip) to fill, which covers already the evaluation part (split of the dataset and testing phase, print of the results).
Your task is to:

1. Read and preprocess the dataset an store it into a DataFrame;
2. Create a Pipeline that builds a classification model that can predict the labels of the (unlabeled) reviews (i.e., it predicts if a review is helpful or useless).

Ex. 1 Preprocessing

The first step of this task consists of generating a DataFrame characterized by two fields: label (it is the target attribute of the classification problem) and features (it is a vector of doubles containing the values of the predictive attributes that are used to infer the value of the target attribute). Each record of the data frame is a labeled record and the DataFrame can be used as input of a classification algorithm to infer a classification model.

Specifically, in the preprocessing step, you have to read the Amazon fine-foods dataset (/data/students/bigdata-01QYD/Lab3/Reviews.csv) and create a JavaRDD<LabeledPoint> RDD based on the input dataset. Then, store the content of the JavaRDD<LabeledPoint> RDD into a SparkSQL DataFrame. Remind that each object of type LabeledPoint is characterized by two fields: label (it is a double) and features (it is a vector of doubles). In this task, we set the value of label to 1.0 for representing the class label “helpful” and to 0.0

for representing the class label “useless”. For the (predictive) features, we initially choose as only feature the length of the field “Text” (i.e., the vector features contains one single double value for each record that is set to the length of “Text”).

Since not all the reviews have been rated, you must filter out the reviews that have never been rated (i.e., those having 0 as helpfulness denominator) before creating the JavaRDD<LabeledPoint> RDD. Remove also the header of the input file.

Ex.2 Creating a Pipeline

Now create a Pipeline for generating the classification model. Remember that one of the steps of the pipeline will be a classification algorithm (e.g., a logistic regression algorithm or a Decision Tree), and that you will probably need other pre-processing and post-processing steps to treat the labels (if you are using the Decision Tree algorithm).

When you have created the pipeline, run and take note of the final precision/quality of the built model displayed on the stdout by the evaluation code provided at the end of the SparkDriver.java file.

Create two versions of your application. One version based on the Decision tree algorithm and another one based on the Logistic regression algorithm. Finally, compare the quality of the two generated models.

Ex.3 Adding features

Now it is your turn to improve the classifier you have implemented so far. Try to generate 3-4 other features (to be included in the features vector) based on the original review features, and re-run the pipeline. Generate the features that you think could be useful for predicting the value of the class label (useful/useless). What is the quality of your classification model?

Ex.4 Use the content of the review

Try to create a classification model based on the content of the field “Text” (i.e., on the words appearing in Text). Use the code titled “Textual data classification example code” (available on the web site of the course) as reference code.
Is the quality of the generated model better than the previous ones?