TP Spark

My color codes:
`find . -name '*solution*'` : shell instruction
`df.show(4)` : Python or PySpark instruction.
`variable` : file name or variable

Obvious remarks :

- even if some correction may be available online (whether from us or another source), refrain from checking the correction of a question before you have seriously tried writing your own solution.
- this lab is not graded so please avoid sending us your solution. However, whether you use the spark shell or notebook, it's probably a good idea to copy your successful instructions in some external file (.py or .ipynb) outside the container, for your own record.

I exhort you to peruse the Spark's official doc or the more specific PySpark doc. Nonetheless, I also provide links toward the book co-authored by Spark's designer: Spark - The Definitive Guide: Big Data Processing.

This lab is made up of 3 parts:

1. An introduction to spark (legacy RDD API), in which we shall write a traditional wordcount application on a tiny dataset: a medieval poem from the 12^{th} century.
2. Then we will analyze some data using the Spark SQL = Dataframe API.

   We shall work on a dataset retrieved from Chicago city's data portal. The dataset records informations about food establishment inspections carried by the city's public health services. In particular, for each inspection one records the result of that inspection as well as a list of violations.
3. Then we will use Spark MLlib to predict the results of the food inspections from the list of violations, using Logistic regression. *Disclaimer: this lab was adapted from a Microsoft Azure tutorial.*

Environment

Docker is available on university's computer. If you are working from home, you may (i) connect through ssh on the university's computer. https://www.dep-informatique.u-psud.fr/node/350. Basically,

```
ssh first.last@ssh1.pgip.universite-paris-saclay.fr
```

In that case, you probably won't have efficient access to the GUI, but almost every part of this lab can be performed on a shell, so you don't miss much. However, all students connecting to ssh1.pgip... work within the same docker machine. Therefore, to avoid conflicts, you must (a) rename the container name to some other name instead of sparklab, and (b) don't publish any port (remove all the -p xxxx:yyyy instructions) since they would also conflict and you won't use these graphical interfaces anyway on ssh. (ii) or you may install Docker on your own computer (but we won't help with that).

Launch a jupyter pyspark notebook in docker. The docker instruction is[1]:

```
docker run \
--security-opt seccomp=unconfined \
--name sparklab -it \
-p 8888:8888 \
-p 4040:4040 -p 4041:4041 \
jupyter/pyspark-notebook
```

You may find some information about the image at https://jupyter-docker-stacks.readthedocs.io/en/latest/using/specifics.html#apache-spark

[1] Actually, if we do not care about ipython and about jupyter, we could use official pyspark image; `docker run -it apache/spark-py /opt/spark/bin/pyspark`, which is only 1GB instead of 4GB: https://hub.docker.com/r/apache/spark-py

Do not stop the container (we could have launched in *detached* mode, but this way you can observe some log messages). If you stop it's no big trouble, but you will have to restart the container before you can proceed. Execute all subsequent operations in another shell.

To access the Jupyter Lab GUI (this is not really needed until we start using a notebook in 3): you should observe some message like the following. Open the corresponding url in a web browser, using the localhost url (127.0.0.1) as an url, as illustrated below, because the alternative url suggested (here, *2e52bd448641*) is only valid from within the container, hence basically useless.

```
To access the server, open this file in a browser:
file:///home/jovyan/.local/share/jupyter/runtime/jpserver-8-open.html
Or copy and paste one of these URLs:
http://2e52bd448641:8888/lab?token=707d897fc17daf012b2f0327beb94ea66946a47e21c14029
 or http://127.0.0.1:8888/lab?token=707d897fc17daf012b2f0327beb94ea66946a47e21c14029
```

I you stopped and restarted the container (or if you launched in detached mode) you won't get this log message. In that case, type `docker exec -it sparklab jupyter server list` which returns the token and use the localhost (127.0.0.1) for the url, as illustrated above.

1 First steps with the REPL, Spark queries on RDDs

For this part, we shall rely on the `lai-eliduc.txt` dataset, which you can find in the same archive as those instructions.

1. Discovering Spark through the Python interpreter for spark ; pyspark.

   (a) Launch a shell on the spark interpreter : `docker exec -it sparklab bash`, or using the GUI.

   (b) Put the `lai-eliduc.txt` file in the container : `docker cp lai-eliduc.txt sparklab:/`, or using the GUI upload . If you receive an error message while trying to copy the file (telling you the cp is aborted, probably due to uid rights issue), you can work around the issue by downloading the lab files from within the container instead of copying the file into the container. You may use `wget`, then unzip to extract the file ; both commands are part of the docker image we are using.

   (c) Check where `pyspark` executable lies (`which pyspark`) and identify the Spark executables that are available in the same directory. The directory is probably `/usr/local/spark/bin`. You could also try `printenv | grep spark` (not much else).

   (d) You will probably prefer the `ipython` interpreter to the plain `python` one, so before running pyspark execute the following instruction. IPython (which is used by Jupyter notebooks as a kernel - Python backend) provides you with syntax highlighting, some magic commands, history navigation, etc.

   ```
   export PYSPARK_DRIVER_PYTHON=/opt/conda/bin/ipython
   ```

   (e) Run `pyspark`.

   You will observe something like:

   ```
   Welcome to
         ____              __
        / __/__  ___ _____/ /__
       _\ \/ _ \/ _ `/ __/  '_/
      /__ / .__/\_,_/_/ /_/\_\   version 3.2.1
         /_/

   Using Python version 3.9.7 (default, Sep 29 2021 19:20:46)
   Spark context Web UI available at http://2e52bd448641:4040
   Spark context available as 'sc' (master = local[*], app id = local-1646050844700).
   SparkSession available as 'spark'.
   ```

   → A `spark.sparkContext` has been defined, called : `sc`.

   → (Use localhost) URL of SparkUI where you can observe job completion, DAG, etc.

2. Your first Spark instructions (you may of course adopt other `names`, but then adapt the instructions) :

   (a) Create a spark rdd named `myrdd` from a Python list with

   ```
   sc.parallelize(data)
   ```

   Then print the first two items in `myrdd`.

   (b) From the spark interpreter, load the file `lai-eliduc.txt` in an RDD called `lignes`.

(c) Count the number of lines in this RDD.

(d) Count the number of partitions in the RDD.

3. Another way to execute a Spark application is to create a Python application (script)

   (a) in the GUI, launch a Python notebook.

   (b) to run a Spark application from Python, import the Spark modules:

```
from pyspark.sql import SparkSession

# Spark session & context
spark = SparkSession.builder.master('local').getOrCreate()
sc = spark.sparkContext
```

> You may check using `sc.getConf().getAll()` that while the driver's host remains the same and while the notebook still relies on some pyspark shell, the application id and start time are not the same anymore, our new spark application is independent from the previous one.
> The Spark UI port has probably been incremented (spark takes the first available starting with 4040), which you can check with `sc.uiWebUrl`.

   (c) Then perform some spark computations as you did from the interpreter.
   Here we run our application from a notebook in local mode. But in general we would have executed the Python application with spark-submit, which allows to specify cluster options.

4. Using either the pyspark shell or the python notebook, write a wordcount program in Spark:

   - you may consider that words are split around spaces or better, use a regular expression as a delimiter in `re.split(pattern,string)`. Ex: a delimiter could be any sequence of one or more non-word caracter (use PCRE class!).
   - write the result of your wordcount in a directory called `nbmots`

   If you do not find the solution (and only after trying), you may get inspiration from online doc or the container's example `/usr/local/spark/examples/src/main/python/wordcount.py`.

   Why is the solution from the doc using a `flatMap` on the RDD `lignes` and not a `map`?

> You will observe that the result is written into a file `part00000` (or if using pyspark shell, 2 files `part00000` and `part00001`). You may display the whole result with `cat *` but the inherent distribution of RDDs prevents us from specifying a filename into which the result would be written. Here are nevertheless some solutions to write a RDD in a single file (that should be avoided in general but here the result is small enough that it fits in memory):
>
> - transformation `coalesce(k)` merges the RDD partitions into a single partition when $k = 1$.
> - one may convert the RDD to a scala collection with `collect` then write this collection into a file.
> - one may use hdfs: `hdfs dfs -getmerge <src> <localdst>` (but irrelevant for this lab since we do not use hdfs)

5. Add a transformationto the above "program" . We wish to drop items that do appear less than 8 times. Do not write the result into a file but display the first 10 RDD elements in the REPL.

2 Spark Dataframes.

1. Retrieve the datafile:

```
wget -O food.csv https://data.cityofchicago.org/api/views/4ijn-s7e5/rows.csv
```

2. Let us first load all the libraries needed for this lab:

```
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer
from pyspark.sql import Row
from pyspark.sql.functions import udf, desc, col
from pyspark.sql.types import *
```

3. Load a Dataframe `df1` from file `food.csv`. Then, display the Dataframe schema using `.show()` and `.printSchema()` You may find the spark doc useful.

4. Create a dataframe `inspections` by projecting on the columns that we shall use:

   `'Inspection ID', 'DBA Name','results', 'Violations'`

   and then removing all lines containing a "NULL". Make sure you drop the nulls after projecting, otherwise you will end up with an empty dataframe.

   Would you save or waste time if you loaded the `inspections` dataframe from file `food.csv` instead of `df1`? Explain.

   `.dropna()` p110

5. Compute from `inspections` the list `nbre` of possible inspection results, and the number of inspections yielding such results, ordered by decreasing number of inspections. The resulting dataframe should have the following schema, so you will probably want to rename columns :

```
root
 |-- results: string (nullable = true)
 |-- nb: long (nullable = false)
```

> For the list of available methods on a Dataframe: the doc.

`.sort()...` p36

6. Then:

   - Display the first 4 lines of the result: what difference between `.take(4)` and `.show(4)`?
   - Display the execution plan `nbre` with `explain(true)`.
   - check in the web interface the DAG of tasks.
   - Display stats about `nbre` using `.describe()`

7. Visualize the number of inspections for each possible result in a pie chart.

> The easiest way is probably to convert (on the driver) the Spark dataframe to a (non-distributed) Pandas dataframe , which you can plot as in https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.plot.pie.html and then visualize.
>
> `toPandas()`

```
#Some useful functions :
"""
spark_df.toPandas()
pandas_df.set_index(...)
pandas_df.plot.pie(y=...)
plot.figure.savefig('pie.pdf')
"""
```

3 Predictions with spark.

We next turn on to the MLlib library of Spark in order to perform predictions from our data.

3.1 Labelling data: defining categories

1. Observe a few lines of the "results" and "violations" columns.

   Logistic regression is a binary "classification" method, so we will group results into 2 categories using the following function:

```
def labelForResults(s):
    if s == 'Fail':
        return 0.0
    elif s == 'Pass w/ Conditions' or s == 'Pass':
        return 1.0
    else:
        return -1.0
```

2. Register the above function as a udf, and use it to transform the data: we want to obtain a 2-column DataFrame `labeledData` : the columns are `label` and `violations`, where category `label` takes values 0.0 or 1.0.. Values less than 0 are filtered out. `udf()` p121

   ⚠ `label` must be of numeric data type, so we may use

   `myudf = udf(myfunction,DoubleType())`

3. We naturally distinguish two phases; a training phase and a validation phase. To keep things simple, we split our data in 2: (i) a Dataframe `training` containing 25% of `labeledData` records will be used to train the model, whereas (ii) Dataframe `validationDf` containing the remaining records will be used for validation. Partition `labeledData` records through random sampling . Use the Spark function `randomSplit` , taking 105 as a (🇬🇧 seed) to initialize the random generator. `.randomSplit()` p409

3.2 Defining the model: specifying predictive variables and tuning parameters

1. We now have data labeled with their category. We still have to extract from the text field `violations` the variables from which our model will buid predictions. For this, we convert that field to a vector of numbers. The logistic regression will then be applied to those vectors.

   We define a 3-steps pipeline:

   - the first step splits `violations` into a sequence of words (Tokenizer)
   - the second step converts the sequences to a frequency vector (each word gets assigned an index, then we map each list to its corresponding frequency vector simply by counting the occurrences of each word)
   - the last step applies the linear regression (use 10 iterations, with regularisation parameter equal 0.01)

   The book is not the best source of information on those aspects, better check the spark doc, which uses logistic regression as an example for ML pipelines.

2. Train your pipeline on the training data, then validate the pipeline on test data.

4 Executing a spark application.

1. Instead of interacting with spark through the REPL, we next execute an independant application. Start with a simple application: you may adapt SimpleApp from spark doc, or reuse your wordcount code from above to write an application that takes as parameters the name of input and output files.

```
spark/bin/spark-submit \
  --master local[1] \
  yourApp.py \
```

   We will not bother to remove warnings as long as there are no error messages.

2. Read and understand some of the examples :

```
/usr/local/spark/examples/src/main/python/ml/kmeans_example.py
/usr/local/spark/examples/src/main/python/transitive_closure.py
/usr/local/spark/examples/src/main/python/pagerank.py
```

   Identify where `cache()` is used and why.