Exercises with Spark

The goal of this lab is to make you write Spark programs. This lab is particularly long, with exercises of increasing difficulty. you are not expected to finish it!

1 Opening Spark

There are several ways to use Spark. Don’t spend too much time trying to install on your computer (when in doubt ask Louis Jachiet for an account on teach.jachiet.com)!

1.1 Using notebooks

Optional step, create a virtualenv (cleaner install):

```
virtualenv sparkjup
```

To activate the virtualenv (it needs to be done each time you want to use packages installed within the virtual environment):

```
source sparkjup/bin/activate
```

Install the required packages:

```
pip install pyspark notebook
```

Now to start the notebook:

```
jupyter-notebook
```

Using the notebook interface, create a Python notebook

```python
import pyspark
sc = pyspark.SparkContext('local[4]', appName="Spark Lab Session")
```

The string “local[4]” indicates that we want a local execution of Spark with 4 executors (you can increase or decrease this number in function of your computer speed, a good number is the number of CPU or the number of CPU minus one).

1.2 Using PySpark shell

You simply need to run the PySpark command and the shell will open. It is possible to open the shell in a notebook like interface see [https://www.sicara.ai/blog/2017-05-02-get-started-pyspark-jupyter-notebook-3-minutes](https://www.sicara.ai/blog/2017-05-02-get-started-pyspark-jupyter-notebook-3-minutes). The PySpark command can be launched on the Telecom machine lame11.enst.fr that can be accessed through ssh. Once logged in you need to use the following commands:

```
source /infres/ir510/hadoop/bin/hadoop_env
pyspark
```

2 First steps with Spark

2.1 First RDD

**DO:** Create a RDD containing the list $L$ of the integers ranging from 0 to 2999 (both included).

You can use the function `sc.parallelize(l)` to create an RDD containing the same elements as the list l.
2.2 Computing the sum of cubes

DO: Using the RDD computed at the previous step, compute a second RDD representing $C$, the list of cubes of integers in $L$. In other words $C = \{0^3, 1^3, \ldots, 2999^3\}$. What is the sum of elements in $C$?

2.3 Last digits of elements in $C$

DO: Compute an RDD containing a set of pairs $(0, v_0), \ldots, (9, v_9)$ where $v_i$ is the number of integers in $C$ having $i$ for last digit.

The last digit of an integer $i$ can be computed using $(i \% 10)$.

2.4 Digits of $C$

In Python, the list containing the digits of an integer can be computed using the following function:

```python
def digits(i):
    return [e for e in str(i)]
```

DO: Compute an RDD containing a set of pairs $(0, v_0), \ldots, (9, v_9)$ where $v_i$ is the number of times the digit $i$ appears in an integer of $C$.

3 Using the Movie Lens dataset

3.1 Getting the dataset

The Movie Lens is a dataset in which users rate movies. It is already on the school HDFS at the address /datasets/movie_small/ but if you work on your local machine you can find it here: https://grouplens.org/datasets/movielens/. For this lab session, I recommend to use the small dataset that can be downloaded here: http://files.grouplens.org/datasets/movielens/ml-latest-small.zip.

3.2 Getting the dataset into an RDD

To read the dataset use the following code snippet:

```python
import re
future_pattern = re.compile("""([^,]+|"[^"+]+")(?=,|$)""")

def parseCSV(line):
    return future_pattern.findall(line)

path_data = "/datasets/movie_small" # CHANGE ME
ratingsFile = sc.textFile(path_data+"/ratings.csv").map(parseCSV)
moviesFile = sc.textFile(path_data+"/movies.csv").map(parseCSV)
```

DO: Copy the code above and run it. Make sure that the data is loaded and explore the data using the function .take(2) on the RDDs.

3.3 Cleaning data

Both RDDs (ratingsFile and moviesFile) contain the data from the CSV and columns have been split, however the data is still raw: all fields are strings and the header of the dataset is still here.

DO: Using filter and map, create one RDD containing the movies and one containing ratings. In both RDD you should remove the header of the CSV file and you should cast the third column of ratingsFile to float.
3.4 10 best movies of all times

The RDD corresponding to ratings.csv contains 4 columns, the first corresponds to the id of the user rating the movie, the second is the id of the movie being rated, the third column is the actual rating while the fourth is the timestamp of the rating.

**DO:** Compute the 10 movieId that have the best average rating. Hint: use sortBy and take to select the best movies.

3.5 Ordered list of movies with names

The RDD corresponding to movies.csv contains 3 columns: the first is a movieId, the second is the title of the movie and the third describes the genre of the movie.

**DO:** Using the RDD containing the average rating and a join operation with the movie RDD, compute an RDD containing the ordered list movies with their name and average rating. What are the names of the 10 best movies?

*Note: Don’t forget to collect() the RDD to see the list!*

3.6 Better ordered list

The previous RDD was unsatisfactory because some movies made it to the top using only one 5-rating.

**DO:** Compute this list again by using the two following metrics:

\[
\text{grade}(m) = \frac{\sum_{r \in \text{ratings}(m)} r^2}{|\text{ratings}(m)| + 1} \\
\text{grade}(m) = \frac{\sum_{r \in \text{ratings}(m)} r}{|\text{ratings}(m)|} \times \log(1 + |\text{ratings}(m)|)
\]

where \(\text{ratings}(m)\) designates the multiset of ratings for movie \(m\) and \(\text{grade}(m)\) is the grade according which you should extract the top 10.

4 Re-implementing K-Means

The K-means algorithm is a very classical algorithm for data clustering. The algorithm is already implemented in Spark ML library (and we will use it next week) but the goal here is to “re-implement” to understand how it works behind the scene. It is important that you understand how K-Means works, therefore I invite you to read [https://en.wikipedia.org/wiki/K-means_clustering#Standard_algorithm_(naive_k-means)].

For the purpose of the exercise you can suppose that \(K\) is small and hence it is reasonable to have an RDD containing arrays of size \(K\).

4.1 Loading data

In this lab we will cluster the data found here [https://louis.jachiet.com/tmp/SZRd1P4Lb5S_data.tsv]. For that we can use the following piece of code:

```python
pts = []
ground_truth = dict()

with open("data.tsv","r") as f:
    for l in f.readlines():
        data=l.split("\t")
        pts.append(tuple(map(float,data[:-1])))
        ground_truth[pts[-1]] = int(data[-1].rstrip())

ptsRDD = sc.parallelize(pts)
```

Note that, with the file provided, the points are all 10-dimensional (those are not 2d-points).
4.2 Initialization

The first step of the K-Mean algorithm is to select the $K$ initial points. For that you can use the Forgy method that simply selects $K$ random points.

Here is an idea to select $K$ random points: map each point $p$ to $(p, s_p)$ where $s_p$ is a randomly generated floating point numbers (use `random.random()`) and then select the $K$ points $p_1, \ldots, p_K$ with the highest $s_{p_1}, \ldots, s_{p_K}$.

4.3 Step method

One iteration of the K-Means algorithm implies to find the closest center for each point in the dataset and then given the set of points $p_1, \ldots, p_m$ associated with center $c$ we find the new position of $c$ by computing the average of $p_1, \ldots, p_m$.

Note that we want our K-means to scale well, therefore we really want to avoid to materialize a list with all the points attached to any given center.

**Hint:** Create a function `assign` that takes a point $p$ and a list of points $c$ and returns the pair $(i, p)$ where $i$ is the index in $l$ of the point in $l$ which is the closest to $p$. Then compute the new centers using the ideas to compute averages that we have seen in class.

4.4 Getting results

The final result of your code should be a function `KMeansRDD(R, K, I)` where $R$ is an RDD containing tuples with the same dimension $D$. Then your code should find $K$ centers taken from $R$, then performs $I$ steps to optimize the centers and finally return an RDD containing pairs where the first element of the pair is a point $p$ in $R$ and the $i$ such that the $i$-th center after the $I$-th iteration is the closest center to point $p$.

4.5 Measuring the performance of your algorithm

The loading code given above provides a ground truth with $K = 7$ classes. After running `KMeansRDD(ptsRDD, 7, 100)`, compute the matrix $M$ of dimension $K \times K$ where $M_{i,j}$ is the number of points classified in the $i$-th class by your algorithm and in the $j$-th class in the ground truth. What can you say about the performance of your algorithm?