Big Data and Artificial Intelligence

Lab Assignment (TP) 1: Big Textual Data Analytics

Vassilis Christophides

Getting started with Apache Spark

In this first lab assignment you will :

- Familiarize with the Google Colaboratory environment
- Learn how to create Spark RDDs from files or main memory and store them in text or structured files
- Acquainted with Spark APIs to process RDDs of objects or paired RDDs
- Understand how to extract, transform and aggregate complex information from different textual datasets

To work with Spark you will need either to *install Spark libraries in your PC* or *run Spark* on Google Collaboratory according to the instructions posted in the Moodle. You may read the quick introduction to Spark for *Scala* or *Python* programmers available at the following URL: `https://spark.apache.org/docs/latest/quick-start.html`

Exercise 0 (**2 points**): Download and Prepare your Document Corpus

Project Gutenberg was the first provider of free electronic books, or eBooks. Load (locally in your PC or in Google colab) and read in your Spark Instance the following three works by Shakespeare from the Project Gutenberg:

- `RomeoJuliet.txt https://www.gutenberg.org/files/1112/1112.txt`
- `Hamlet.txt https://www.gutenberg.org/files/1524/1524-0.txt`
- `RichardII.txt https://www.gutenberg.org/cache/epub/1776/pg1776.txt`

To exclude punctuation values and convert all words to lowercase, you need to write a cleaning function.

Hint: Though `regexp_replace()` is a powerful function, it can be difficult to use in many cases. That is why spark has provided some useful functions to deal with strings. Using `translate()` function we can replace one or more characters to another character without using regular expressions.

Exercise 1 (**4 points**): Count Words

In this exercise you should write the code for counting into your Spark Instance the occurrences of each unique word in your document corpus. The output on the `RomeoJuliet` document should be something like the following:

```
[('project', 90),
 ('gutenberg', 32),
 ('ebook', 13),
 ('of', 535),
 ('juliet', 65),
 ('shakespeare', 8),
 ('this', 279),
 ('was', 51),
 ('files', 3),
 ('produced', 2),
 ('at', 87),
 ('when', 55), …
]
```

Hints:

- Use `flatmap()` & `split()` functions to obtain the words in all lines separated by ‘ ‘
- Use `map()` & `reduceByKey()` functions to count how many times each word occurs
- Use `filter()` & `re.match()` functions to keep only words composed of letters and not of numbers

Exercise 2 (**6 points**): Finding Frequent Terms and Stop Words

You are asked to find the *stop words* (i.e., valid English words occurring very frequently) of your documents. You are asked to merge the stopwords of the three documents and report their total frequency in your corpus into a single csv file.

Hints:

- You can download the `nltk` library to get the list of English stopwords
- Use `filter()` function to obtain the stopwords in each document
- Use `union()` & `reduceByKey()` functions to combine the stopwords of all documents and fusion their frequences
- Use `sortByKey()` to sort in decreasing frequence order the stopwords of all documents.
- Use `map()` function to switch `(key,val)` pairs you obtained as `(val,key)` pairs
- Use `spark.createDataFrame()` to create a dataframe out of your rdd of combined stopwords and store it into a single csv file with `write.csv()` function

Exercise 3 (**5 points**): Simple Inverted Index

You are asked to implement an **inverted index** (`http://bit.ly/2wWhkKM`) for the corpus of three documents you worked in the previous exercises. An inverted index provides for each distinct word, the filenames in the corpus that contain this word, along with some other information (e.g., count/position within each document). For example, assume the following corpus, consisting of doc1.txt, doc2.txt and doc3.txt:

doc1.txt: “This is a very useful program. It is also quite easy.”
doc2.txt: “This is my first MapReduce program.”
doc3.txt: “Its result is an inverted index.”

An inverted index would contain the following data (in random order):

ID	**Term**	**Document**
1	This	doc1.txt, doc2.txt
2	Is	doc1.txt, doc2.txt, doc3.txt
3	A	doc1.txt
4	Program	doc1.txt, doc2.txt
…	…	…

Implement a simple inverted index for your document corpus as shown in the above Figure, after skipping all stopwords. How many distinct words exist in your corpus? Experiment with different possible aggregation implementations and select the most efficient one.

Hints:

- Reuse the code of the previous exercise to find the non-common words of a document, and then combine them for all documents of your corpus
- Use `map()` function to indicate the origin of each word in a document by its filename
- Use `distinct()` function to obtain unique words in your document corpus
- Use `groupByKey()` function to group the multiple occurrences of a unique word in the three documents of your corpus: experiment also with `reduceByKey()` and `aggregateByKey()`
- Use `zipWithIndex()` function to add an `id` for each element of your rdd

Exercise 4 (3 points): Extended Inverted Index

Extend the inverted index of the previous exercise, in order to also keep the frequency of each word in all documents of your corpus. The new output should be of the form where the word frequency should follow a single '#' character after the filename of the document that contains this word. Also, sort the document list based on the Frequency.

ID	**Term**	**Document #Frequency**
1	This	doc1.txt #1 , doc2.txt #1
2	Is	doc1.txt #2, doc2.txt #2, doc3.txt #1
3	A	doc1.txt #1
4	Program	doc1.txt #1 ,doc2.txt #1

…	…	…