ENSEA

Beyond Engineering

AI & BIG DATA

Big Data

Lab report : Big Textual Data Analytics

2nd year student

Erwan BROUDIN, Faez MEZDARI

15 février 2023

Exercise 0 : Download and Prepare your Document Corpus

In this part we have to download the three texts Romeo and Juliet, Hamlet and Richard III.
To do this, we use the addfile (present in spark) and textFile functions to create an RDD that we can use later. Here is how we import the different texts :

```
##### Download all the text files
import time
from pyspark import SparkFiles

RJ_url = "https://www.gutenberg.org/files/1112/1112.txt"
spark.sparkContext.addFile(RJ_url)
RomeoJuliet_rdd=sc.textFile(SparkFiles.get("1112.txt"))

Hamlet_url = "https://www.gutenberg.org/files/1524/1524-0.txt"
spark.sparkContext.addFile(Hamlet_url)
Hamlet_rdd=sc.textFile(SparkFiles.get("1524-0.txt"))

Richard_url = "https://www.gutenberg.org/cache/epub/1776/pg1776.txt"
spark.sparkContext.addFile(Richard_url)
Richard_rdd=sc.textFile(SparkFiles.get("pg1776.txt"))
```

Code 1 – Importation of the three texts

In the text file on Project Gutenberg, there are many lines at the beginning talking about the project. We decided not to delete them to avoid wasting time on a point that would not have given us anything more in the following, or we would not want to analyze the data but simply show that we know how to handle them.

Afterward, we need to clean up the text : there are some extracts that do not interest us. For example, words consisting of numbers, or even line breaks that only appear as [”]. We will first make the text "cleaner" with the function we call "cleaningFunc" which will remove all the function from a string, as well as put all the characters of a word in lower case. Here is how we do it :

```
##### Create the cleaning function

import string

def cleaningFunc(text):
  text = text.lower()
  text = text.translate(text.maketrans("","",string.punctuation))
  return text
```

Code 2 – cleaningFunc

Finally, we want to be able to apply it to the three previously imported texts, so we need to make a flatMap that will allow us to define how to separate lines, as well as to apply the cleaningFunc function to the whole text. As we want to measure the execution time, we also set two timers that will allow us to calculate the execution time.

```
##### Clean all the texts
start = time.time()

RJ_clean = RomeoJuliet_rdd.flatMap(lambda line : line.split(" ")).map(cleaningFunc)

Hamlet_clean = Hamlet_rdd.flatMap(lambda line : line.split(" ")).map(cleaningFunc)

Richard_clean = Richard_rdd.flatMap(lambda line : line.split(" ")).map(cleaningFunc)

end = time.time()

print(end-start)

Richard_clean.take(20)
```

Code 3 – Cleaning the three texts

The execution time of this part is 0.01 seconds.
We can check that the code has run by using the take function and seeing what happens then :

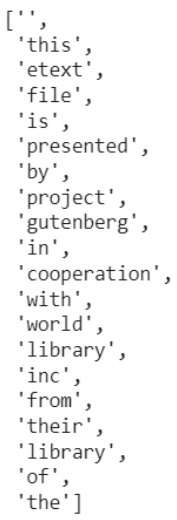

IMAGE 1 – Richard III text cleaned

All the words have been separated, miniscule and the punctuation has been removed.

Exercise 1 : Count Words

In this exercise, we have two main objectives :

- to remove words that are not words ("2012" for example)
- to count the words so as not to have repetition

To do this, we will use the filter function which allows us to filter the words in the RDD according to a certain criterion : in this case, the fact that the value is only composed of a string of letters. We also do some mapping : each word is weighted by one, and when the program encounters a duplicate with "reduceByKey", we will add the two values together to see the value at the end of the process for each word.
Here is the code that we make, in order to count the words in each of the three texts :

```
##### Count the (filtered) words of each documents

import re
import string
from pyspark import SparkContext

start = time.time()

RJ_CountWords = RJ_clean.filter(lambda x: re.match('[a-z]+', x)).map(lambda word: (word, 1)).reduceByKey(lambda x, y : x + y)
Hamlet_CountWords = Hamlet_clean.filter(lambda x: re.match('[a-z]+', x)).map(lambda word: (word, 1)).reduceByKey(lambda x, y : x + y)
Richard_CountWords = Richard_clean.filter(lambda x: re.match('[a-z]+', x)).map(lambda word: (word, 1)).reduceByKey(lambda x, y : x + y)

end = time.time()
print(end-start)

Richard_CountWords.take(10)
```

Code 4 – Counting each words

Once executed, we then display the execution time and we can also observe that each word has actually been counted to indicate the number of occurrences :

```
0.11334037780761719
[('this', 204),
 ('is', 296),
 ('presented', 1),
 ('project', 26),
 ('gutenberg', 21),
 ('in', 306),
 ('cooperation', 1),
 ('world', 31),
 ('library', 16),
 ('inc', 11)]
```

IMAGE 2 – Richard III text, words counted

The result is as expected, and we can even notice that the execution time is 0.12 seconds.

Exercise 2 : Finding Frequent Terms and Stop Words

In this part, we need to analyze stopwords only. Some words such as "this", "a", "for" are very common words, and it is these that we will be dealing with first. In order to keep only these words, we need to import the list of stopwords into a list object, using the nltk library. Here is how we did the import :

```
###### Import the list of stopwords

import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stopwords = stopwords.words('english')
```

Code 5 – Importation of the stopwords

Next, we map the word - as in the previous exercise - but add a criterion for filtering : the word is only kept if it is in the list of stopwords previously imported. Here is how we code this part :

```
##### Keep only the stopwords from the text files


RJ_StopWords = RJ_clean.filter(lambda x : x in stopwords).filter(lambda x: re.match('[a-z]+', x)).map(lambda word: (word, 1)).reduceByKey(lambda x, y : x + y)
Hamlet_StopWords = Hamlet_clean.filter(lambda x : x in stopwords).filter(lambda x: re.match('[a-z]+', x)).map(lambda word: (word, 1)).reduceByKey(lambda x, y : x + y)
Richard_StopWords = Richard_clean.filter(lambda x : x in stopwords).filter(lambda x: re.match('[a-z]+', x)).map(lambda word: (word, 1)).reduceByKey(lambda x, y : x + y)
```

Code 6 – Texts without words that are not stopwords.

For the next point, we need to combine the three texts into one RDD using the "union" function. To use it, we simply call it as a method, specifying as an argument the text on which we are doing the union.
Then, we do a "reduceByKey" in order to keep only one time each word iteration (thus combining the number of occurrences of "this" in Romero and Juliet, Hamlet and Richard III), and we sort by decreasing number of occurrences of the stopwords.
Here is the code we write to achieve this :

```
###### Create a rdd with all the stopwords of the 3 texts

all_StopWords_V1 = RJ_StopWords.union(Hamlet_StopWords).union(Richard_StopWords).reduceByKey(lambda x, y : x + y).sortBy(lambda row : row[1], ascending=False)

all_StopWords_V1.take(30)
```

Code 7 – Every stopwords in a single RDD

The result displayed on the screen is the sum of the stopwords occurrences in the different texts :

```
[('the', 3004),
 ('and', 2636),
 ('to', 2049),
 ('of', 1961),
 ('i', 1588),
 ('a', 1440),
 ('my', 1342),
 ('in', 1191),
```

IMAGE 3 – Occurence of the most used stopwords

For the next question, we are only asked to reverse the columns : the occurrences must appear before the words. We do a different mapping as follows :

```
all_StopWords = all_StopWords_V1.map(lambda x : (x[1],x[0]))
```

Code 8 – Every stopwords in a single RDD with occurence first

---

Finally, we create a DataFrame that contains all the data we were processing, and we will convert it as a CSV file. To do this, we simply call the "createDataFrame" function which will create it from our RDD, and then use the write.csv function which will save this DataFrame in a CSV format that we can export.
This is how it is done :

```
##### Create the dataframe

start = time.time()

columns = ["count","stopword"]
allStopWordsDF = spark.createDataFrame(all_StopWords, columns)

allStopWordsDF.show()

##### Create the csv file

allStopWordsFormatCSV = allStopWordsDF.write.csv("StopWord")

end = time.time()

print(end-start)
```

Code 9 – Create a DataFrame and a CSV File

This is the DataFrame we obtained :

```
+-----+--------+
|count|stopword|
+-----+--------+
| 3004|     the|
| 2636|     and|
| 2049|      to|
| 1961|      of|
| 1588|       i|
| 1440|       a|
| 1342|      my|
| 1191|      in|
| 1180|     you|
| 1067|    that|
| 1032|      is|
|  877|    with|
|  848|      it|
|  841|     not|
|  824|    this|
|  764|     for|
|  675|     his|
|  672|      me|
|  662|      be|
|  616|     but|
+-----+--------+
```

IMAGE 4 – DataFrame

We can also see the CSV file being saved in the default directory :

3004	the
2636	and
2049	to
1961	of
1588	i
1440	a
1342	my
1191	in
1180	you
1067	that
1032	is

IMAGE 5 – CSV File

The execution time of this question is about two seconds.

Exercise 3 : Simple Inverted Index

In this question, we will work on words that are not stopwords. We also do not want to know the number of occurrences of each word, but only in which text each word is represented. We then kept only the stopwords before joining the three texts :

```
##### Delete the stopwords from the text files

start = time.time()

RJ_OtherWords = RJ_clean.filter(lambda x : x not in stopwords).filter(lambda x: re.match(
    '[a-z]+', x)).map(lambda word: (word, "RJ.txt")).reduceByKey(lambda x, y : x )
Hamlet_OtherWords = Hamlet_clean.filter(lambda x : x not in stopwords).filter(lambda x:
    re.match('[a-z]+', x)).map(lambda word: (word, "Hamlet.txt")).reduceByKey(lambda x, y
     : x)
Richard_OtherWords = Richard_clean.filter(lambda x : x not in stopwords).filter(lambda x:
     re.match('[a-z]+', x)).map(lambda word: (word, "Richard.txt")).reduceByKey(lambda x,
     y : x)

###### Create a rdd with all the other words of the 3 texts

all_OtherWords = RJ_OtherWords.union(Hamlet_OtherWords).union(Richard_OtherWords).
    reduceByKey(lambda x, y : x + ", " + y).sortByKey().zipWithIndex().map(lambda x : (x
    [1],x[0]))

end = time.time()

print(end-start)

all_OtherWords.take(10)
```

Code 10 – Sorting and indexing "Uncommon" words

This execution allows us to see that we have an identifier, followed by the list of common words arranged in alphabetical order and the names of the texts in which they are used.

```
[(0, ('abate', 'RJ.txt, Hamlet.txt')),
 (1, ('abatements', 'Hamlet.txt')),
 (2, ('abbey', 'RJ.txt')),
 (3, ('abbot', 'Richard.txt')),
 (4, ('abed', 'RJ.txt')),
 (5, ('abels', 'Richard.txt')),
 (6, ('abet', 'Richard.txt')),
 (7, ('abhorred', 'RJ.txt, Hamlet.txt')),
 (8, ('abhors', 'RJ.txt')),
 (9, ('abide', 'RJ.txt, Hamlet.txt, Richard.txt'))]
```

IMAGE 6 – RDD : Common words

Exercise 4 : Extended Inverted Index

For this last exercise, we need to extend the interest of the previous exercise. We want to see not only in which text the different words appear, but also how often they are used. To do this, we use a counter which we will increment. This counter is a little different from the ones we are used to seeing because it is put into a string in each iteration. Here is how we do this :

```
##### Delete the stopwords from the text files

start = time.time()

RJ_OtherWords_Extended = RJ_clean.filter(lambda x : x not in stopwords).filter(lambda x:
    re.match('[a-z]+', x)).map(lambda word: (word, "RJ.txt #1")).reduceByKey(lambda x, y
    : "RJ.txt #"+str(int(x[x.find("#")+1:])+int(y[y.find("#")+1:])) )
Hamlet_OtherWords_Extended = Hamlet_clean.filter(lambda x : x not in stopwords).filter(
    lambda x: re.match('[a-z]+', x)).map(lambda word: (word, "Hamlet.txt #1")).
    reduceByKey(lambda x, y : "Hamlet.txt #"+str(int(x[x.find("#")+1:])+int(y[y.find("#")
    +1:])) )
Richard_OtherWords_Extended = Richard_clean.filter(lambda x : x not in stopwords).filter(
    lambda x: re.match('[a-z]+', x)).map(lambda word: (word, "Richard.txt #1")).
    reduceByKey(lambda x, y : "Richard.txt #"+str(int(x[x.find("#")+1:])+int(y[y.find("#"
    )+1:])) )

end = time.time()
print(end-start)

RJ_OtherWords_Extended.take(10)
```

Code 11 – Counting words in texts

We can see with the text of Romeo and Juliet that this code works well as shown by the display of the first 10 elements of the RDD :

```
[('project', 'RJ.txt #90'),
 ('gutenberg', 'RJ.txt #32'),
 ('ebook', 'RJ.txt #13'),
 ('juliet', 'RJ.txt #65'),
 ('shakespeare', 'RJ.txt #8'),
 ('files', 'RJ.txt #3'),
 ('produced', 'RJ.txt #2'),
 ('proofing', 'RJ.txt #1'),
 ('tools', 'RJ.txt #1'),
 ('developed', 'RJ.txt #1')]
```

Image 7 – Romeo and Juliet : words counted

Then we simply need to combine these three texts into a single RDD and index them. Before doing this, we have arranged them in alphabetical order for easier reading.

```
###### Create a rdd with all the other words of the 3 texts

start = time.time()

all_OtherWords_Extended = RJ_OtherWords_Extended.union(Hamlet_OtherWords_Extended).union(
    Richard_OtherWords_Extended).reduceByKey(lambda x, y : x + ", " + y).sortByKey().
    zipWithIndex().map(lambda x : (x[1],x[0]))

end = time.time()

print(end-start)

all_OtherWords_Extended.take(10)
```

Code 12 – Indexing sorting and make the union of all "Uncommon words"

We can see from the following display that we have an indexed rdd, sorted alphabetically by word, which allows us to retrieve both the text from which it is taken, as well as the number of occurrences in each.

```
[(0, ('abate', 'RJ.txt #1, Hamlet.txt #1')),
 (1, ('abatements', 'Hamlet.txt #1')),
 (2, ('abbey', 'RJ.txt #1')),
 (3, ('abbot', 'Richard.txt #7')),
 (4, ('abed', 'RJ.txt #1')),
 (5, ('abels', 'Richard.txt #1')),
 (6, ('abet', 'Richard.txt #1')),
 (7, ('abhorred', 'RJ.txt #1, Hamlet.txt #1')),
 (8, ('abhors', 'RJ.txt #1')),
 (9, ('abide', 'RJ.txt #1, Hamlet.txt #1, Richard.txt #1'))]
```

IMAGE 8 – RDD : Indexed, sorted and counted common words

This exercise took 1.28 second to complete.