LAB DISTRIBUTED DATA ANALYTICS I
TUTORIAL VI

Juan Fernando Espinosa Reinoso
303158
Group 1: Monday Tutorial

Exercise 1: Basic Map and Reduce (2 Points)

1.1 Dataset

First, for this exercise, the requirement is to use the **Gutenberg_ebook.txt** file. The goal is to get the **WordCount** in the ebook. First, a directory is created by aplying the following command: **hadoop fs -mkdir /dda/exercise1.** Next, putting the dataset in the folder is necessary.

Now it is time to dive in in the procedure.

1.2 Mapper

By taking advantage of **sys** library in python which allows to pass data between the mapper and reducer, initialization and extraction of each line of text is executed first. Splitting each word for the purpose and finally, by iterating through all the words print an output as follows: **(word, 1)**

```
#!/usr/bin/env python
import sys

for line in sys.stdin:
    line = line.split()

    for word in line:
        print('%s\t%s' % (word, 1))
```

The output of the mapper looks as follow (small sample):

```
Project 1
Gutenberg       1
Literary        1
Archive 1
Foundation,     1
how     1
to      1
help    1
produce 1
our     1
new     1
eBooks, 1
and     1
```

```
how     1
to      1
subscribe       1
to      1
our     1
email   1
newsletter      1
to      1
hear    1
about   1
```

1.3 Reducer

The Reducer receives as input the (word, 1) printed out by the mapper. Next, an iteration through all the words is applied: First, splitting the input data between words and value. A condition is applied for storing the counting words in a dictionary. If the word already exists as a **key** in the dictionary called **word_count** add +1 to the value, otherwise add the word as a key with value of 1.

```
#!/usr/bin/env python
import sys

word_count = {}
counting = 0
for line in sys.stdin:
    (words, count) = line.split('\t')
    if words in word_count:
        word_count[words] += 1
    else:
        word_count[words] = 1
print(word_count)
```

1.4 Execution

For the execution of the Map-Reduce process the HadoopStreaming comes to play. Check and copy the path of the streaming jar file saved within hadoop folders. It allows to run different codes without the default jar file. Taking the local paths from the mapper and reducer, as well as the path of the input from hadoop by applying the command: **hadoop fs -ls /path/to/file.** Once all the above-mentioned requirements are gotten it is time to execute.

We take the following information as execution command:

```
hadoop jar /usr/local/Cellar/hadoop/3.2.1_1/libexec/share/hadoop/tools/lib/hadoop-streaming-3.2.1.jar /
-mapper "python /Users/juanfer/Documents/Maestria/SemesterIII/Lab-DDA/tutorial-6/exercise-1/mapper.py" /
-reducer "python /Users/juanfer/Documents/Maestria/SemesterIII/Lab-DDA/tutorial-6/exercise-1/reducer.py" /
-input /dda/exercise1/gutenberg_ebook.txt /
-output /dda/exercise1/output
-file /Users/juanfer/Documents/Maestria/SemesterIII/Lab-DDA/tutorial-6/exercise-1/mapper.py /
-file /Users/juanfer/Documents/Maestria/SemesterIII/Lab-DDA/tutorial-6/exercise-1/reducer.py
```

he folaqefs We have to insert two times the path for the mapper and reducer: The first one points hadoop to the executables and the latter is for distributing the executables around all nodes. Finally, the output shown has the structure of (**Word, count)**
Note: Since there were no preprocessing steps, some punctuation signs are not identified.

The code works satisfactory.

Exercise 2: Data cleaning and text tokenization (6 points)

2.1 Dataset

The goal of the exercise is to overcome the issues experienced in the output of the first exercise. As a consequence, 5 text datasets were given. Similarly as the first exercise, a directory called **exercise2** were created and all the 5 files were stored in it. It is important to mention that all files have the same initial name (ie exercise2-1.txt, exercise2-2.txt, and on) because it is easier to execute all files using a trick that will be explained below.

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	juanfer	supergroup	546.97 KB	Jun 21 17:10	1	128 MB	exercise3-1.txt
-rw-r--r--	juanfer	supergroup	1.01 MB	Jun 21 17:10	1	128 MB	exercise3-2.txt
-rw-r--r--	juanfer	supergroup	228.6 KB	Jun 21 17:11	1	128 MB	exercise3-3.txt
-rw-r--r--	juanfer	supergroup	1.51 MB	Jun 21 17:11	1	128 MB	exercise3-4.txt
-rw-r--r--	juanfer	supergroup	905.71 KB	Jun 21 17:11	1	128 MB	exercise3-5.txt

2.2 Mapper

As from its name it is easy to infer, in this step the goal is to map which words are stop words and also if they have punctuation. Get the lines in all documents for next take every single word in it and make a binary classification: if the word is **stop-word = 0, and 1 otherwise.**

```
#!/usr/bin/env python
import numpy as np
import os
import sys

stop = ['a','able','about','across','after','all','almost','al
        'by','can','cannot','could','dear','did','do','does','
        'hers','him','his','how','however','i','if','in','into
        'my','neither','no','nor','not','of','off','often','or
        'so','some','than','that','the','their','them','then',
        'what','when','where','which','while','who','whom','wh

punctuations = []
for line in sys.stdin:
    line = line.split()
    for word in line:
        if word.isalpha() == False:
            continue
        if word in stop:
            print('%s\t%s' % (word, 0))
        else:
            print('%s\t%s' % (word, 1))
```

2.3 Reducer

Take the output from the **Mapper** and start iterating through each (word, classification). Next step is to create a dictionary with the classification binary value as key and all the words as values. This make it possible to drop all the misclassified values (i.e. 0) and return only the words that are not stop words nor have they punctuation.

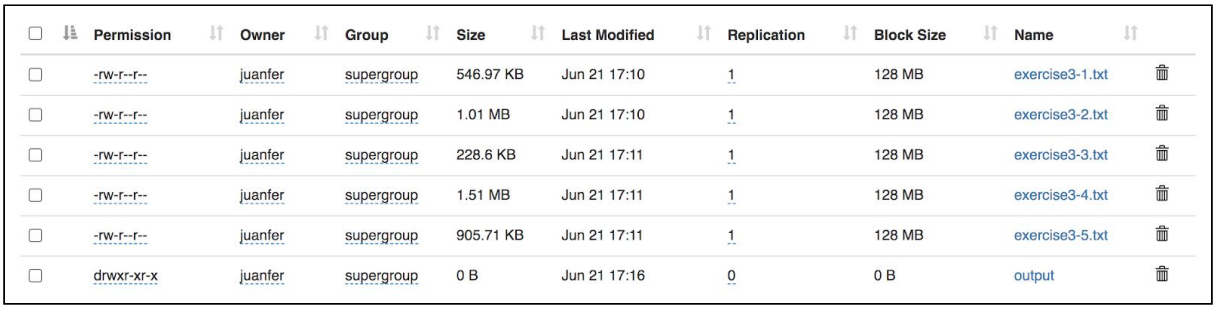
```python
#!/usr/bin/env python
import numpy as np
import os
import sys

words_cleaned = {}
for i, line in enumerate(sys.stdin):
    line = line.split()
    (word, count) = line
    if count not in words_cleaned:
        words_cleaned[count] = []
        words_cleaned[count].append(word)
    else:
        words_cleaned[count].append(word)
result = words_cleaned.pop('0')
print(words_cleaned)
```

2.4 Execution

The same requirements as explained in step 1.4 are gathered here. the execution command looks as follow:

```
hadoop jar /usr/local/Cellar/hadoop/3.2.1_1/libexec/share/hadoop/tools/lib/hadoop-streaming-3.2.1.jar /
-mapper "python /Users/juanfer/Documents/Maestria/SemesterIII/Lab-DDA/tutorial-6/exercise-2/ex2-mapper.py"
/
-reducer "python /Users/juanfer/Documents/Maestria/SemesterIII/Lab-DDA/tutorial-6/exercise-2/ex2-reducer.py"
 /
-input /dda/exercise2/exercise3*.txt  /
-output /dda/exercise2/output /
-file /Users/juanfer/Documents/Maestria/SemesterIII/Lab-DDA/tutorial-6/exercise-2/ex2-mapper.py /
-file /Users/juanfer/Documents/Maestria/SemesterIII/Lab-DDA/tutorial-6/exercise-2/ex2-reducer.py
```

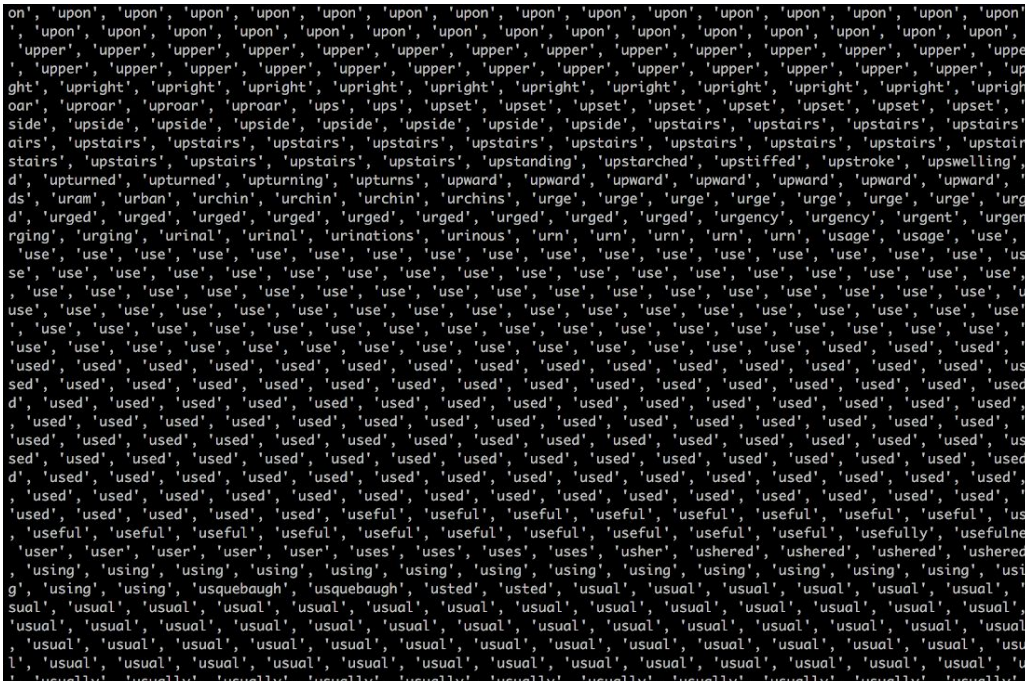
Here there are few keys that worth explain: To tell hadoop to run all the files at once and without writing all paths it is necessary to make all files have same initial name and then by adding an * the program will take all files that accomplish the condition. The output folder is created after running the code.



Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	juanfer	supergroup	546.97 KB	Jun 21 17:10	1	128 MB	exercise3-1.txt
-rw-r--r--	juanfer	supergroup	1.01 MB	Jun 21 17:10	1	128 MB	exercise3-2.txt
-rw-r--r--	juanfer	supergroup	228.6 KB	Jun 21 17:11	1	128 MB	exercise3-3.txt
-rw-r--r--	juanfer	supergroup	1.51 MB	Jun 21 17:11	1	128 MB	exercise3-4.txt
-rw-r--r--	juanfer	supergroup	905.71 KB	Jun 21 17:11	1	128 MB	exercise3-5.txt
drwxr-xr-x	juanfer	supergroup	0 B	Jun 21 17:16	0	0 B	output

Finally, to run and explore the results the following command is inserted: **hadoop fs -cat /dda/exercise2/output/filename.**

Note: Hadoop always sorts the inputs, hence it is possible to appreciate several words next to one another.



Exercise 3: Calculate TFIDF scores of words/tokens (12 points)

3.1 Dataset

For this exercise, we use the same dataset applied in exercise 2. Hence no need to create a directory.

3.2 Mapper

As this exercise requires, we need to calculate the TF and IDF in order to get the values. The mapper is doing the following process:

1. preprocessing the data: erasing all stop-words and punctuaction.
2. It is selecting the name of the files for the IDF calculation. We need to know how many documents are there.
3. As the TFIDF happens to be a unique value per document it is necessary to keep the order of words, therefore, at the end the name of the file and the words are merged together, and to the right the count number 1.

```python
#!/usr/bin/env python
import sys
import os
from string import punctuation

stop_words = ['a','able','about','across','after','all','almost','
        'by','can','cannot','could','dear','did','do','does','eith
        'hers','him','his','how','however','i','if','in','into','i
        'my','neither','no','nor','not','of','off','often','on','o
        'so','some','than','that','the','their','them','then','the
        'what','when','where','which','while','who','whom','why','

for line in sys.stdin:
    filename = os.environ["map_input_file"]
    line = line.translate(None, punctuation)
    line = line.split()
    for word in line:
        word = word.lower()
        if word.isalpha() == True and word not in stop_words:
            file_word = filename + "," + word
            print('%s\t%s' % (file_word, 1))
```

3.3 Reducer

The reducer task is to compute the TF and IDF simultaneously. The steps for doing so are stated next.

1. Call the outputs of the mapper and split it into the count and the (filename, word)
2. Next, split the filename and word as well by a comma-separated split.
3. Finally, merge into a list all the values for document and words
4. Iterate through the newly created list and create a dictionary with the **keys** as the filename and values as the words. This dictionary makes it pretty simple to extract the value of number of files as well as a dictionary of number of unique words with the respective number of appearances in a document.
5. As being said in the previous step a dictionary with the wordCount per document is created. Also by using Counter I got the number of unique words in the document. Then I proceed to calculate TF.
6. Followed it is necessary to get the number of documents each token appears on. Hence a small Counter iterating in the list of dictionaries created in TF makes it possible to extract the partial IDF.
7. Finally a dictionary applying the formula dof IDF is created with the words as keys and the values as outputs.
8. A final for loop is useful for iterating through all the words in each document and multiply the respective IDF with the same word. Hence, finally the output is a document specific TFIDF.

```python
import sys
import numpy as np
from collections import Counter

lista = []
TF = []
TFIDF = []
documents = {}
for line in sys.stdin:
    file_word, count = line.split('\t')
    document, words = file_word.split(',')
    lista.append((document,words))

for i,e in lista:
    if i in documents.keys():
        documents[i].append(e)
    else:
        documents[i] = []
        documents[i].append(e)

number_documents = int(len(documents.keys()))

for i in documents.values():
    frequency = Counter(i)
    unique_words = int(len(list(set(i))))
    partial_tf = {k: (float(v) / float(unique_words)) for k,v in frequency.items()}
    TF.append(partial_tf)
```

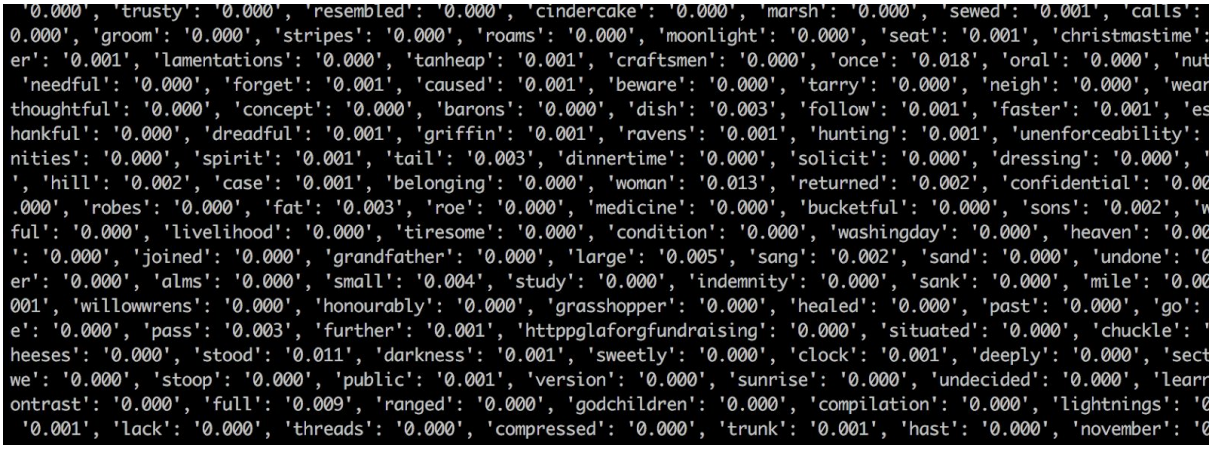
```python
documents_token = Counter(i for j in TF for i in j)
IDF = {k: (np.log(float(number_documents) / float(v))) for k,v in documents_token.items()}


for i in TF:
    result_dict = {}
    for key,value in i.items():
        for key1,value1 in IDF.items():
            result_dict["{}".format(key)] = '{:.3f}'.format(value * value1)
    TFIDF.append(result_dict)
print(TFIDF)
```

3.4 Execution

Similar as the previous execution, the command is as follows:

hadoop jar /usr/local/Cellar/hadoop/3.2.1_1/libexec/share/hadoop/tools/lib/hadoop-streaming-3.2.1.jar /
-mapper "python /Users/juanfer/Documents/Maestria/SemesterIII/Lab-DDA/tutorial-6/exercise-3/ex3-mapper.py" /
-reducer "python /Users/juanfer/Documents/Maestria/SemesterIII/Lab-DDA/tutorial-6/exercise-3/ex3-reducer.py" /
-input /dda/exercise3/exercise3*.txt /
-output /dda/exercise3/output /
-file /Users/juanfer/Documents/Maestria/SemesterIII/Lab-DDA/tutorial-6/exercise-3/ex3-mapper.py /
-file /Users/juanfer/Documents/Maestria/SemesterIII/Lab-DDA/tutorial-6/exercise-3/ex3-reducer.py

The output of the algorithm is a list which contains dictionaries per document with the resultant TFIDF.



As we can see, since the words are repeated seldomly and the number of unique words per document are really high and also the number of words repeated across files makes the output to be low with some values as zero because it requires more decimals.

The output were saved in the directory and the executable output command is: **hadoop fs -cat /dda/exercise3/output/part-00000**

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	juanfer	supergroup	0 B	Jun 22 02:08	1	128 MB	_SUCCESS
-rw-r--r--	juanfer	supergroup	343.39 KB	Jun 22 02:08	1	128 MB	part-00000

References:

[1] https://towardsdatascience.com/installing-hadoop-on-a-mac-ec01c67b003c

[2] https://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/

[3] https://www.roseindia.net/bigdata/hadoop-shell-commands.shtml

[4] http://rare-chiller-615.appspot.com/mr1.html
[5] https://maelfabien.github.io/bigdata/MRJobP/#hadoop-streaming