Big Data Assignment2

Duplicate Detection - Pyspark

Lavanya.S.S[16161056]
NSM, Computer science

Problem Description

Duplicate Detection - The large growth in data volumes and the need to integrate data from multiple sources will bring the need to efficiently remove duplicates, there are multiple ways to represent same object by different names, different ways, but they all represent the same real-world object. to ensure high quality of data we should make sure there are no redundant data. in bigdata since we are dealing with huge amount of data its very essential to have more meaningful non-redundant data. The following problems are to be solved,
1a. Write a pyspark code to determine the 1,000 most popular words in the document collection provided. Please ensure that your code removes any special symbols, converts everything to lower case (and if possible: remove stop words, destemming, etc.)

Solution strategy -Part 1a:

The solution strategy for part 1a is
1.To write pyspark code to execute wordcount on the document collection provided

2.It is very important to preprocess the dataset before starting the wordcount, preprocessing the data includes

- Removing the capital letters by using to.lower() function
- We should also remove the stop words using standard list of stop words provided by nltk library, there are approximately 179 stop words in nltk library such as "the ,a ,and, in…."

3.A SparkConf object is created with SparkConf() which is used for
    a. Configuration of a Spark application
    b. To set various Spark parameters as key-value pairs.
4. A setter method in this SparkConf class - setAppName("WordCount") has been used to set the application name.
5. A SparkContext object is created with SparkContext() which is:
    a. Main entry point for Spark functionality as it represents the connection to a Spark cluster.
    b. It can be used to create RDD and broadcast variables on that cluster.
6. Also its very important to obtain the data in a linear mode, as the original dataset has text file name accompanied with the text data. Like this {key: <Filename>, value:<word, count>} pair for each file is put together with the help of RDD.union(other) method which returns the union of this RDD and another one. we must initially discard the text file name and only consider the text data.

7.Once we have the raw text data we need to split them based on spaces using split() ,to obtain words, then we should perform the wordcount operation, by iterating through each word in the document and incrementing the wordcount if the word is already in the dataset. Once we obtain the word and wordcount,

8.now we have to select the top 1000 most popular words in document collection , so we have to sort the word list based on its wordcount. And select only first 1000 words from this list. Also I have removed the digits from the word list using is.digit() , as we are considering only words in this problem

The output of the part1a is as follows,
['project', 'ebook', 'love', 'thy', 'may', 'thee', 'td', 'gutenberg', 'would', 'computer', 'dan', 'read', 'fair', 'like', 'states', 'thou', 'yet', 'youth', 'age', 'system', 'could', 'small', 'spa', 'books', 'copyright', 'donations', 'foundation', 'good', 'night', 'one', 'print', 'tr', 'us', '', 'baby', 'form', 'gutenbergtm', 'hart', 'money', 'quoth', 'reading', 'see', 'sweet', 'cant', 'course', 'domain', 'ebooks', 'even', 'lissa', 'loves', 'medium', 'michael', 'middleton', 'public', 'queen', 'software', 'university', 'using', 'adonis', 'angel', 'archive', 'away', 'central', 'copy', 'damages', 'day', 'distribute', 'doth', 'free', 'glass', 'god', 'hear', 'including', 'know', 'literary', 'monitoring', 'phoenix', 'received', 'soon', 'statement', 'students', 'table', 'thus', 'used', 'women', 'young', 'additional', 'also', 'among', 'breath', 'broken', 'debugging', 'dont', 'every', 'eye', 'eyes', 'fell', 'find', 'give', 'hath', 'heart', 'help', 'html', 'ill', 'illegal', 'legal', 'liability', 'licensing', 'live', 'made', 'men', 'might', 'mine', 'new', 'person', 'pity', 'please', 'pnbspp', 'poor', 'receive', 'refund', 'request', 'rest', 'right', 'school', 'shall', 'sing', 'sorrow', 'still', 'sun', 'things', 'tongue', 'trademark', 'turtle', 'vaded', 'well', 'whether', 'without', 'work', 'activities', 'actually', 'administration', 'ah', 'air', 'alas', 'anyone', 'bade', 'beauty', 'began', 'best', 'birds', 'brook', 'center', 'characters', 'check'…….]

2a) Problem Description

Write a pyspark code to create an inverted index for the 1,000 words determined in Part 1. The inverted index is supposed to be of the form

term1:doc1:weight1_1,doc2:weight2_1,doc3:weight3_1,…

term2: doc1:weight1_2,doc2:weight2_2,doc3:weight3_2,… …

Measure the execution time of the code for the large data set for 5, 10 and 15 executors

Solution strategy -Part 2a:

For creating the inverted index for 1000 words , we need to initially determine from which document the 1000 words are selected , map the words with the document and calculate each words weight, using the formula

weightx_y is:

no. of occurrences of termx in document y / total number of words in document y b.

we can calculate the no of occurrences of term x in document y by considering each document at a time and , total number of words in document y is calculated by sum of all the words in that particular document.

I created a matrix for storing the values of the weightx_y, initially appended it with 0's and gradually appended the weight, also we should make sure the values are type converted to float before appending otherwise python2.7 will consider it as 0 and exclude the decimal part.

The output format is as below

{'project': [[u'1ws4610.txt', 0.0029154518950437317], [u'2ws4510.txt', 0.0006309148264984228],[u'3ws4510.txt', 0.015451174289245983], [u'htmstep0.txt', 0.015451174289245983], [u'htmstep2.txt', 0.0033783783783783786], [u'htmstep31.txt', 0.0033783783783783786]]}

Word Document Name Weight

“project 1ws4610.txt 0.00072939460248”

I have used default dict datatype to store the inverted index .

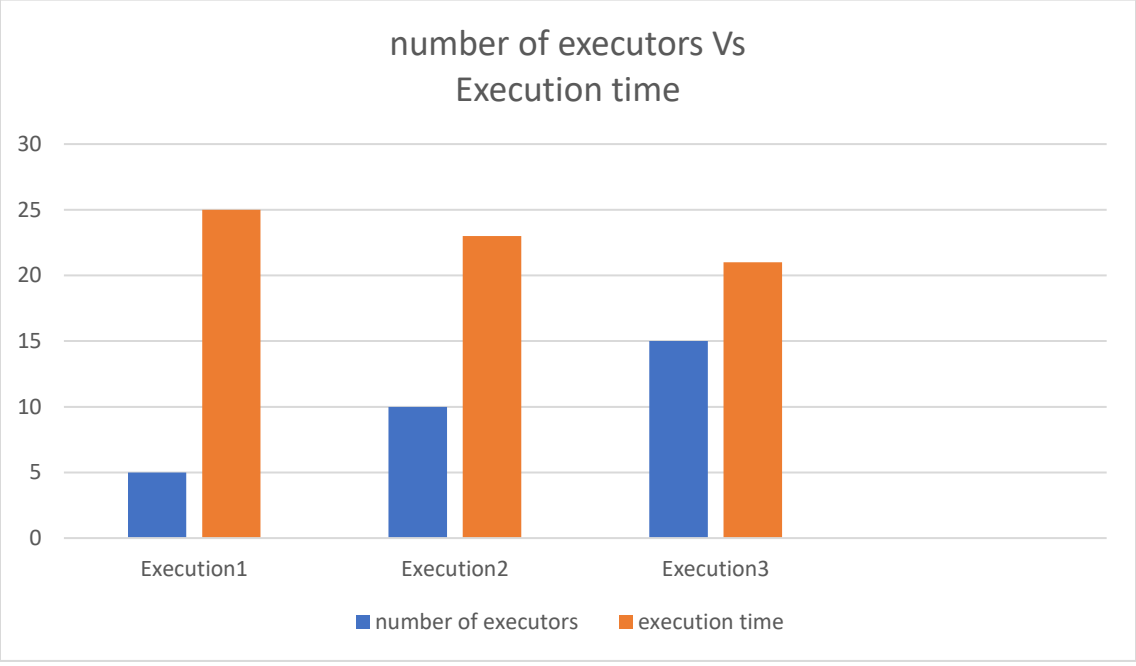
2b. Measure the execution time of the code for the large data set for 5, 10 and 15 executors

Since the whale cluster was congested with huge amount of user data ,I have used the medium dataset with 500 books, In this part we need to execute the code for medium data for 5,10 ,15 executors

The time taken for the files reduces as the number of executors increases

Sl no	Number of executors used	Execution Time taken
1	5	25
2	10	23
3	15	21

Graph



As we can observe as the number of executors decreases the time of execution reduces

3a. Write a pyspark code to calculate the similarity matrix S with each entry of S being S(docx, docy) = $t \epsilon V(weightt\_docx \times weightt\_docy)$ With V being the vocabulary (determined in part 1) and the weights having been determined in part 2

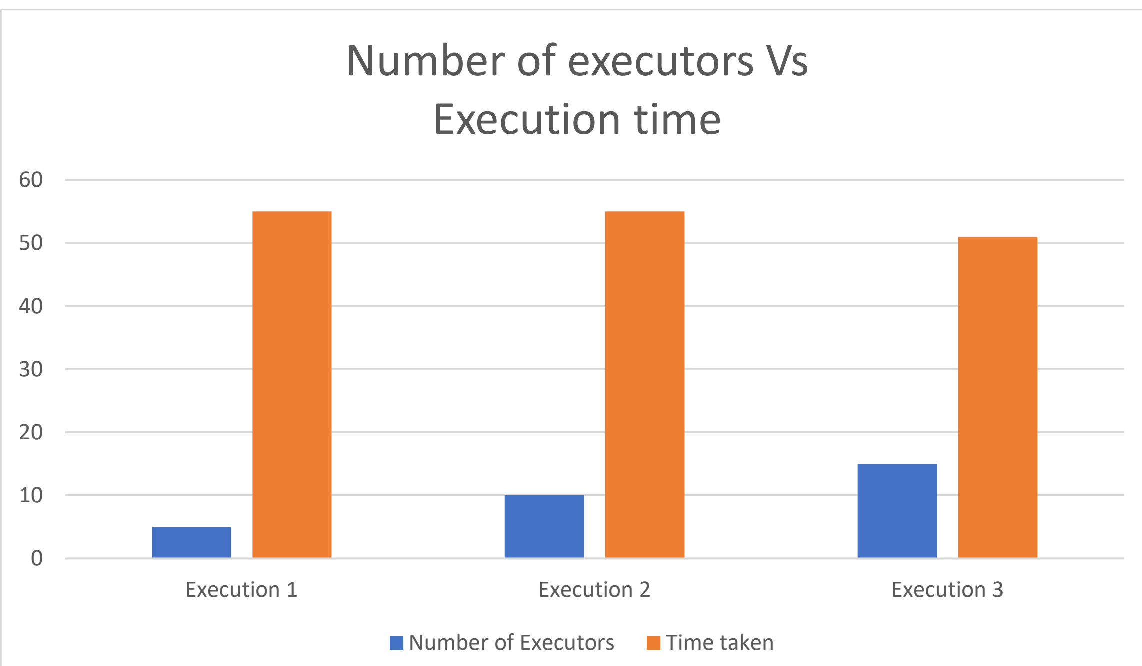
Solution strategy -Part 3a:

We have to create a matrix of the size of length of inverted index and have to multiply the weight of each word in document x and weight of same word in document y and add all these sum for each word, this value will be inserted in similarity matrix at the position sim[docx,docy] .

0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.503194199431054e-05, 2.503194199431054e-05, 2.6857682183414024e-05, 2.6857682183414024e-05, 2.7211218963363955e-05, 2.7211218963363955e-05, 2.7634060993378493e-05,

3b) inverted index was created by considering the medium dataset

Sl no	Number of executors used	Execution Time taken
1	5	55
2	10	55
3	15	51



As we can observe as the number of executors increases the time taken for execution decreases.

Part4: Provide a list of the 10 most similar (or identical) pair of documents from the large data set

Since the cluster was overloaded I was not able to execute the program on large data set

But from the medium dataset the list of 10 most similar pair of documents were ,

```
flatS=flatS[::-10]
for i in flatS:
    for j in range(0,len(S)):
        for k in range(0,j):
            if i == S[j][k]:
                print (j,k)
print(List_FileNames.collect()[j],List_FileNames.collect()[k])
```

this is the code snippet used for performing the operation , where I initially sorted the similarity matrix and selected top 10 values and identified their index , finally selected the text files using these indexes.

8gdct10.txt

1ws4610.txt

2ws4510.txt

7gdct10.txt

htmstep0.txt

2ws2710.txt

8gdct10.txt

7gdct10.txt

htmstep1.txt

htmstep3.txt

Steps to Execute the PySpark Code

1. Copy the following PySpark codes to your local directory, from where it needs to be executed.

    i. hw2bigdata.py

2. Place the input data into HDFS:

HDFS command: hdfs dfs –put <local file> /bigd27/<remotefilename>

Small Input (22 Short books) is already available in: /cosc6339_hw2/gutenberg-22

3. Delete already existing output folder from HDFS:

hdfs dfs -rm -r /big27/…

4.PySpark submit command for Part1-1st python code

spark-submit --master yarn --conf spark.ui.port=4060 --num-executors 15 hw2bigdata.py

While copying the files, and executing it, please make sure that there is read, write & execute permissions is granted for those files.

If not use -chmod to change the permissions.

Resources Used:

1. Whale Cluster:

   • The clusters consist of a login node (whale.cs.uh.edu) and several compute nodes. I

   • The cluster shares home directories , but are otherwise separate.

   • The only access method to whale from the outside world is by using ssh (Example: Putty).

   • The login nodes are to be used for editing, compiling and submitting jobs.

   • They are not to be used for running jobs such as parallel programs.

   • Program runs are submitted through Hadoop (framework that supports the distributed execution of large scale data processing) on this cluster.

   • Jobs are submitted from the login node and run on 1 or more compute nodes. Jobs then run until they terminate in some way, e.g. normal completion, timeout, abort.

2. PySpark:
   • The Spark Python API (PySpark) exposes the Spark programming model to Python.
   Key Differences in the Python API:
   There are a few key differences between the Python and Scala APIs:
   - Python is dynamically typed, so RDDs can hold objects of multiple types.
   - PySpark does not yet support a few API calls, such as lookup & non-text input files.

   • PySpark can also be used from standalone Python scripts by creating a SparkContext in your script and running the script using bin/pyspark
   • Functions can access objects in enclosing scopes

3.Python

- Python is an easy to learn, powerful programming language.
- It has efficient high-level data structures and a simple but effective approach to objecto riented programming.

- python’s elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.
- The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). • Python is also suitable as an extension language for customizable applications