Assignment Report

CSE 487/587 Assignment 2: Big Data Processing with Hadoop

Group:

Akash Kumar Roy	**ubit:** akashroy	**Person Number:** 50316991
Diksha Marathe	**ubit:** dikshaha	**Person Number:** 50290825
Rabiraj Bandyopadhyay	**ubit:** rabirajb	**Person Number:** 50318656

TASK 1:

- **Mapper Logic:**

word_count_mapper.py:

1.The mapper uses word_tokenize method of nltk library to for separating the line from sys.stdin.

2. Then the tokens are checked whether they contain only characters or not.

3. If the above condition is true then print(word,"\t", 1). This line actually prints the output in the sys.stdout(standard output), which is used by the reducer for calculating the final outputs.

- **Reducer Logic:**

word_count_reducer.py:

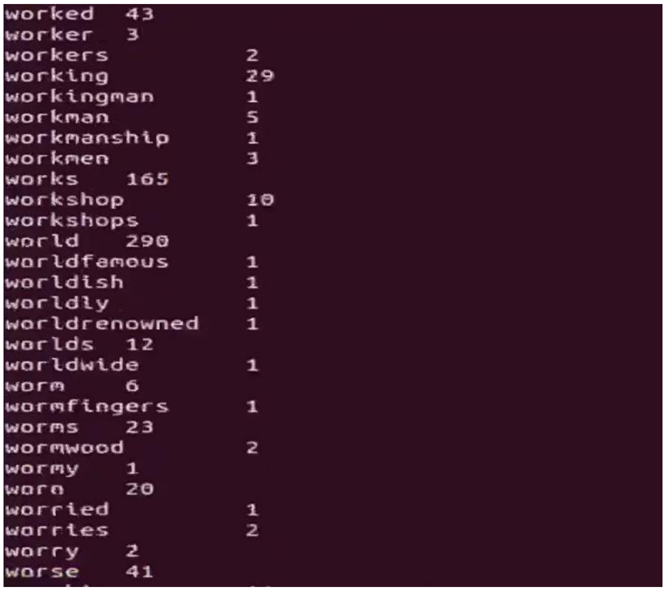
1.The reducer reads each line written by mapper and separates the word from the count.

2. For each line the reducer uses the groupby() method to group the words and the total counts and also sorts the words in ascending order.

3. The final output is then written in the directory specified in the –output parameter of the "mapred streaming" command as part-00000

4. The part-00000 file can be read by using the following command Hadoop fs –cat <output_directory of Hadoop>/ part-00000 which will give the count of each word encountered in the Gutenberg dataset which consists of 3 files james.txt, arthur.txt,leonardo.txt

OUTPUT SCREENSHOT: (This is just a part of the output. Could not take the screenshot of the whole output because its too large. The whole output is shown in the video)



```
worked  43
worker  3
workers         2
working         29
workingman      1
workman         5
workmanship     1
workmen         3
works   165
workshop        10
workshops       1
world   290
worldfamous     1
worldish        1
worldly         1
worldrenowned   1
worlds  12
worldwide       1
worm    6
wormfingers     1
worms   23
wormwood        2
wormy   1
worn    20
worried         1
worries         2
worry   2
worse   41
```

TASK 2:

- **Mapper Logic:**

trigram_mapper.py:

1. From Line 9 it uses sys.stdin to read the input from the terminal. It takes every line as input then remove all the stop words from that line using string.punctuation function. Then if there are any multiple space, paragraph break, tab break it will remove all those spaces and concatenate all the lines one after another with only one space between every word. “text” is the variable where all the lines are getting concatenated and creating a single paragraph.
2. Then we are splitting the text paragraph with the delimiter space (“ “). It will give us an array of all words in the input
3. Now the corresponding while loop is to create all the possible trigrams of the paragraph and the if conditions are provided to handle the edge cases (Like what happens if there’s only word left at the end of the paragraph). All the possible trigrams are stored in the output list.
4. Now in the outer for loop we store each trigram from the output list to the wordset variable. Since every trigram comprises of 3 words we created a nested for loop which will check whether the trigram has any of the key words (it is case insensitive for example science and SCIENCE both conditions have been checked). It will also check whether the trigram has the keyword as a substring (For example if the word is sciences then also it would count it as “$” but if the word is research and even though sea is a substring it will discard that word).

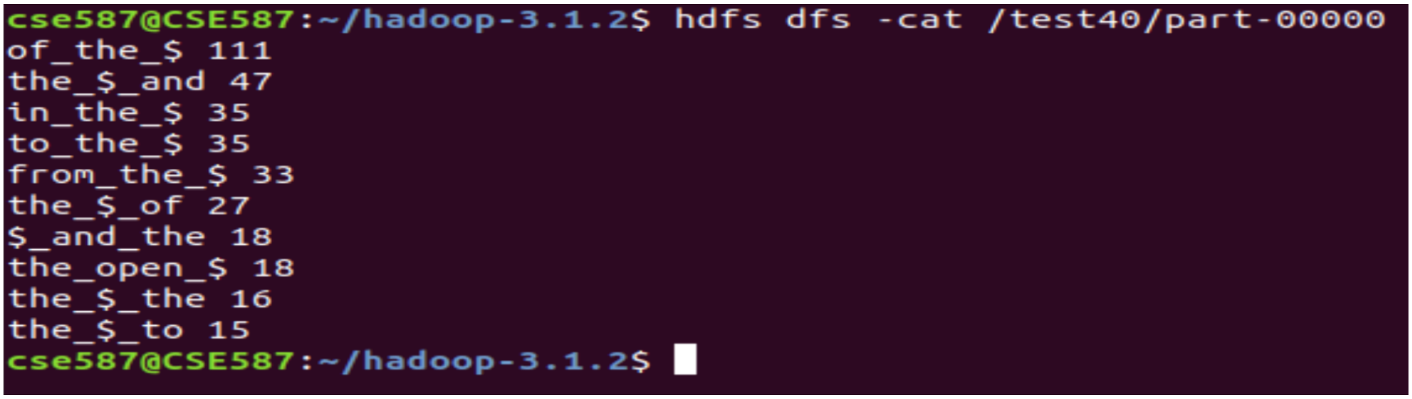
5. It will append all the substring in the map_list and the mapper will print the map_list as its output. It will also print 1 as a count number for every trigram.

- **Reducer Logic**

trigram_reducer.py

1. For Reducer output is dictionary to count the number of occurrences of a single trigram.
2. Since the mapper was printing all the trigrams and 1 as their count we will discard the 1 this time while reading the input from console since we will be joining the common trigrams and count their occurrences.
3. So, to add in the dictionary we just check whether the trigram is already in the dictionary or not if its not in the dictionary then it will to add to the dictionary and if it is already in the dictionary then it will only increase the count of that trigram.
4. Since we are asked for the top 10 trigrams we have sort the dictionary in descending order by value using a lambda function.
5. The last part of the reducer print top 10 trigrams along with their occurrences in the console.

OUTPUT SCREENSHOT:



```
cse587@CSE587:~/hadoop-3.1.2$ hdfs dfs -cat /test40/part-00000
of_the_$ 111
the_$_and 47
in_the_$ 35
to_the_$ 35
from_the_$ 33
the_$_of 27
$_and_the 18
the_open_$ 18
the_$_the 16
the_$_to 15
cse587@CSE587:~/hadoop-3.1.2$
```

TASK 3:

- **Mapper Logic:**

inverted_index_mapper.py

1. In python we can get the file path from in map reduce using os.getenv(“map_input_file”)”. So we are storing the whole path of the document in the file_name variable.
2. Since the filename comes with the whole path and we only care about the filename so we are splitting the string using “/” delimiter and keeping all the values in the file_details list.

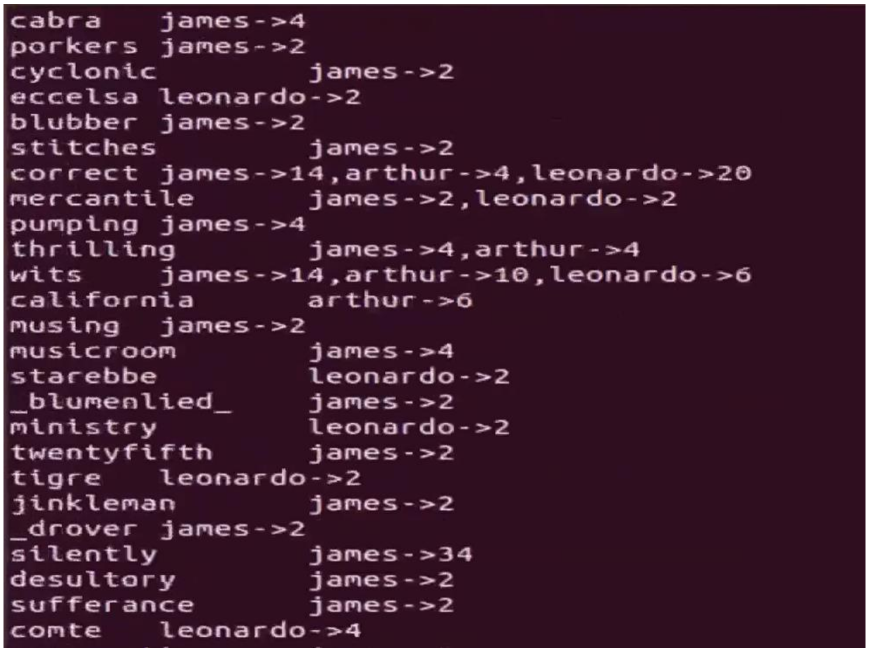
3. Then we are taking all the lines of the document as input and doing the preprocessing for each line like removing punctuations and extra spaces.
4. After preprocessing of each line, we are using space as a delimiter and extracting the words and storing all the words of a single line in the list named “words”
5. At the end the mapper will print all of these words of a line from the words list with their default count zero and with their doc id. (Since file_details list holds the delimited path so the original index will be stored at the last of list).

- **Reducer Logic:**

inverted_index_reducer.py

1. The reducer will read the output of the mapper which consists of the word with its default count and the doc id.
2. We separate all the variables using a delimiter “:”
3. We have used a default dictionary in python to count the number of occurrences of each word. The dictionary is named as “occurrence” in the code. The key of this dictionary is the word and value is another dictionary which stores doc id of the that key and the total occurrence of the key. First it checks whether the dictionary has the word or not if it has the word it increases it’s value by one otherwise if it is a new word then the dictionary set the word count as 1.
4. The last for loop iterates through the dictionary and print the word and its doc id and its number of occurrences from the nested dictionary.

OUTPUT SCREENSHOT: (This is just a part of the output. Could not take the screenshot of the whole output because it’s too large. The whole output is shown in the video)



```
cabra   james->4
porkers james->2
cyclonic        james->2
eccelsa leonardo->2
blubber james->2
stitches        james->2
correct james->14,arthur->4,leonardo->20
mercantile      james->2,leonardo->2
pumping james->4
thrilling       james->4,arthur->4
wits    james->14,arthur->10,leonardo->6
california      arthur->6
musing  james->2
musicroom       james->4
starebbe        leonardo->2
_blumenlied_    james->2
ministry        leonardo->2
twentyfifth     james->2
tigre   leonardo->2
jinkleman       james->2
_drover james->2
silently        james->34
desultory       james->2
sufferance      james->2
comte   leonardo->4
```

TASK 4:

- **Mapper Logic:**

join_mapper.py

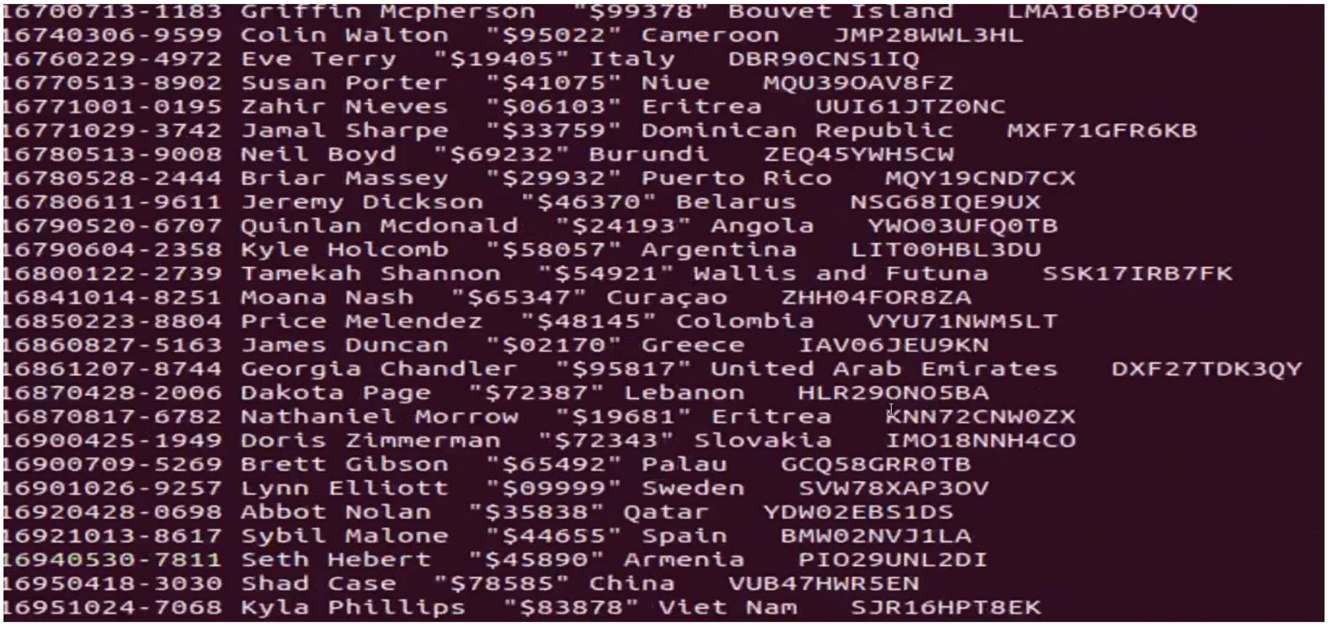
1. Following is the variable representation for the code:
    a. employee_id_first_table: To store the employee ids for the first dataset.
    b. employee_id_second_table: To store the employee ids for the first dataset.
    c. name_first_table: To store the employee names for the first dataset.
    d. salary: To store the salary column of the second dataset
    e. country: To store the country column of the second dataset
    f. passcode: To store the passcode column of the second dataset
2. We have added the headers as first element of each list.
3. Now while reading the data I have removed any "," with space and use space as delimiter to split the data.
4. From the given dataset one dataset has only two features (EmployeeID, Name) and another dataset has four features (EmployeeID, Salary, Country, Passcode) we have used the if condition to check whether the input is part of the first data set or part of the second dataset.
5. Once all the features are added to their corresponding list the mapper will print all the rows from both dataset one by one in the console.

- **Reducer Logic:**

join_reducer.py

1. Following is the variable representation for the code:
    a. employee_id_first_table: To store the employee ids for the first dataset.
    b. employee_id_second_table: To store the employee ids for the first dataset.
    c. name_first_table: To store the employee names for the first dataset.
    d. salary: To store the salary column of the second dataset
    e. country: To store the country column of the second dataset
    f. passcode: To store the passcode column of the second dataset
2. Same as previous this time the reducer will put all the data in their appropriate list from the mapper output.
3. Once all the lists are filled the for loop will run and it will check the employeeID's in the employee_id_first_table and then it will check whether the same employeeID is in the second list or not. If the same employeeID is in the second list then it will join all the respective fields of that employeeID (name,salary,country,passcode).
4. And at last the reducer will print the join table in the console

OUTPUT SCREENSHOT: (This is just a part of the output. Could not take the screenshot of the whole output because it’s too large. The whole output is shown in the video)



```
16700713-1183 Griffin Mcpherson  "$99378" Bouvet Island   LMA16BPO4VQ
16740306-9599 Colin Walton  "$95022" Cameroon   JMP28WWL3HL
16760229-4972 Eve Terry  "$19405" Italy   DBR90CNS1IQ
16770513-8902 Susan Porter  "$41075" Niue   MQU39OAV8FZ
16771001-0195 Zahir Nieves  "$06103" Eritrea   UUI61JTZ0NC
16771029-3742 Jamal Sharpe  "$33759" Dominican Republic   MXF71GFR6KB
16780513-9008 Neil Boyd  "$69232" Burundi   ZEQ45YWH5CW
16780528-2444 Briar Massey  "$29932" Puerto Rico   MQY19CND7CX
16780611-9611 Jeremy Dickson  "$46370" Belarus   NSG68IQE9UX
16790520-6707 Quinlan Mcdonald  "$24193" Angola   YWO03UFQ0TB
16790604-2358 Kyle Holcomb  "$58057" Argentina   LIT00HBL3DU
16800122-2739 Tamekah Shannon  "$54921" Wallis and Futuna   SSK17IRB7FK
16841014-8251 Moana Nash  "$65347" Curaçao   ZHH04FOR8ZA
16850223-8804 Price Melendez  "$48145" Colombia   VYU71NWM5LT
16860827-5163 James Duncan  "$02170" Greece   IAV06JEU9KN
16861207-8744 Georgia Chandler  "$95817" United Arab Emirates   DXF27TDK3QY
16870428-2006 Dakota Page  "$72387" Lebanon   HLR29ONO5BA
16870817-6782 Nathaniel Morrow  "$19681" Eritrea   KNN72CNW0ZX
16900425-1949 Doris Zimmerman  "$72343" Slovakia   IMO18NNH4CO
16900709-5269 Brett Gibson  "$65492" Palau   GCQ58GRR0TB
16901026-9257 Lynn Elliott  "$09999" Sweden   SVW78XAP3OV
16920428-0698 Abbot Nolan  "$35838" Qatar   YDW02EBS1DS
16921013-8617 Sybil Malone  "$44655" Spain   BMW02NVJ1LA
16940530-7811 Seth Hebert  "$45890" Armenia   PIO29UNL2DI
16950418-3030 Shad Case  "$78585" China   VUB47HWR5EN
16951024-7068 Kyla Phillips  "$83878" Viet Nam   SJR16HPT8EK
```

TASK 5:

Bonus Part:

There is a script called createDatasets.py which takes the Train.csv and Test.csv and creates the corresponding Normalized Datasets: Normalized_train.csv and Normalized_test.csv

It can be run externally to create the datasets, for effective runtime management the script has not been made a part of the map-reduce code.

Mapper Logic:

knn_mapper.py:

1) Read the training Data (Train.csv or Normalized_train.csv)
2) For each line in the test data do the following:
    a. Split the line into values and keep a value_index to keep track of the indices being accessed of the Test set.
    b. Create a list from the values and then convert it into numpy array.
    c. For each test data(vector) calculate the Euclidean distance between all the test data and the train data.
    d. Sort the distances calculated and take first k values.
    e. Output the data in the following format index+”\t”+”\t”+distance_value+”,”+label+”;”
    f. Increment value of value_index by 1

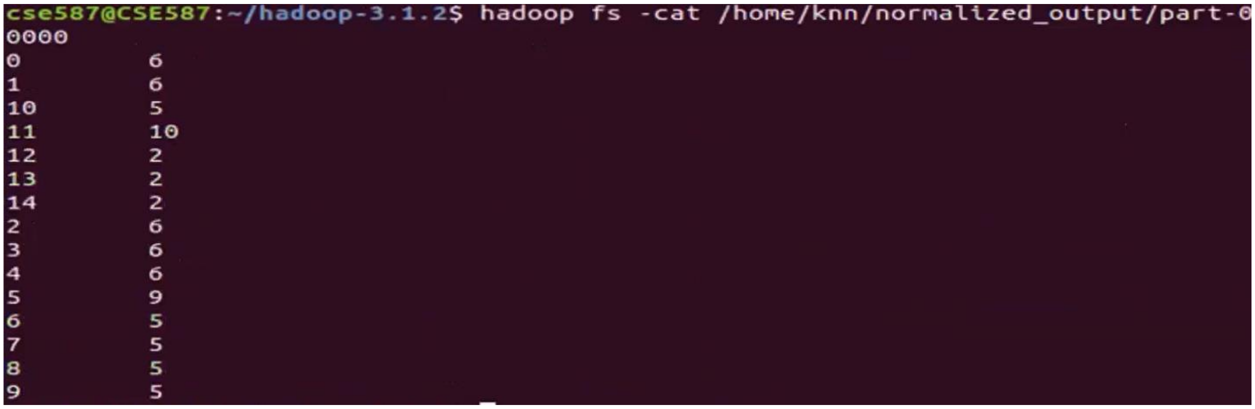
Reducer Logic:

knn_reducer.py

1. For each line in sys.stdin split the line by line.split() , this line will split the values into 2 parts the index and the distance+label separately
2. Then spit the distance+label using val.split(“;”) which results in creating list having the following data [“distance$_i$, label$_i$”] where i represents the index.
3. Then traverse through the values of the list by splitting the data again and getting the maximum of the labels by using np.bincount()
4. Then get the value from the np.bincount() and get the label occurring maximum times using np.argmax()
5. Then print the output to the output_directory in the following format index+”\t”+ label

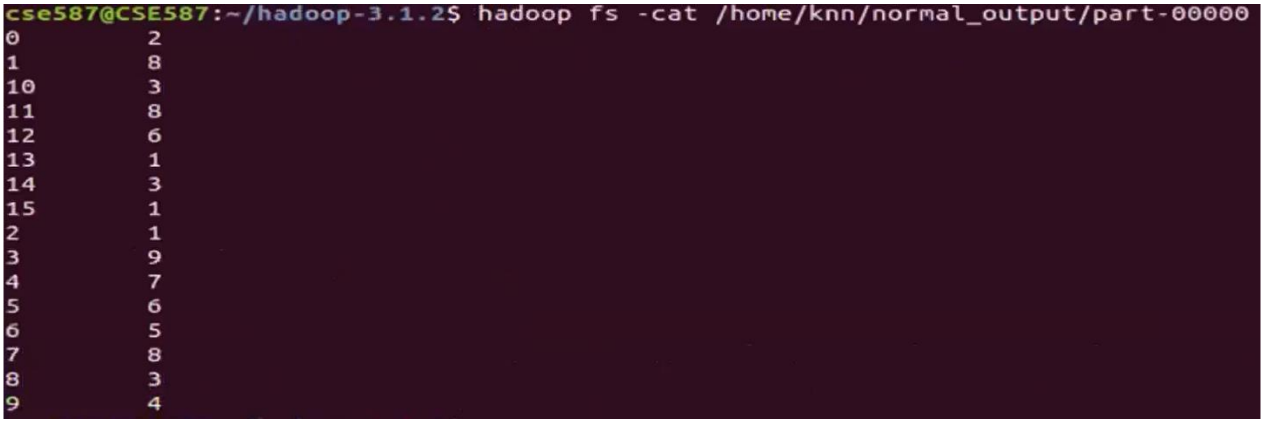
OUTPUT SCREENSHOTS:

1) With normalized data



```
cse587@CSE587:~/hadoop-3.1.2$ hadoop fs -cat /home/knn/normalized_output/part-0
0000
0        6
1        6
10       5
11       10
12       2
13       2
14       2
2        6
3        6
4        6
5        9
6        5
7        5
8        5
9        5
```

2) With given data



```
cse587@CSE587:~/hadoop-3.1.2$ hadoop fs -cat /home/knn/normal_output/part-00000
0        2
1        8
10       3
11       8
12       6
13       1
14       3
15       1
2        1
3        9
4        7
5        6
6        5
7        8
8        3
9        4
```