Big Data Processing with Hadoop

Shanmukha Haripriya Kota	Ravikiran Sunnam	Uneeth Akula
Department of Computer Science	Department of Computer Science	Department of Computer Science
University at Buffalo, NY	University at Buffalo, NY	University at Buffalo, NY
skota@buffalo.edu	rsunnam@buffalo.edu	uneethak@buffalo.edu
UBIT: skota	UBIT: rsunnam	UBIT: uneethak

Abstract

The objective of this assignment is to get started with big data processing with Hadoop. The goals of the assignment are to implement basic text processing tasks from scratch on the Hadoop framework. This project is implemented in python and Hadoop streaming jar is used to run map reduce in Hadoop environment.

Setup and Initialization

- Necessary files are moved into HDFS using: Hadoop fs –copyFromLocal /source /destination

1 Setup and Word Count

Objective: To implement Map Reduce Algorithm to produce count of every word in the document.

Map Logic:

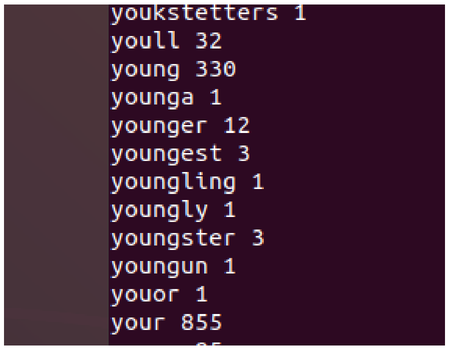
- Mapper takes input from stdin. Path to input files is given in streaming command.
- Hadoop converts the input from the files into line by line format and provides the input to mapper.
- Mapper reads the input line by line, splits the input line on space to get necessary words.
- Wordnet Lemmatizer from NLTK is used to lemmatize words.
- Map output is a tab separated string of (word, 1). Example: **word 1**
- Map output is then flushed on stdout.

Reduce Logic:

- Reducer reads input from stdin.
- Input is a tab separated, (word 1).
- A dictionary is maintained to check if the word is seen already, where key is the word, value is a list of counts. Example: **<Word: [1,1,1,1]>**
- For the line in input, line is split on \t (tab), to get word and count=1
- If the word is present in the dictionary, its count (1) is appended to the list.
- If the word isn’t present in the dictionary, it is added to it and count is appended to the list.
- The Reducer output is then calculated by calculating sum of all the elements in the list for a key and is flushed to stdout.

- The output file thus generated can be retrieved from the output path mentioned in the streaming command.

Output:



```
youkstetters 1
youll 32
young 330
younga 1
younger 12
youngest 3
youngling 1
youngly 1
youngster 3
youngun 1
youor 1
your 855
```

Figure 1.0: Word Count Sample Output

2 N-grams

Objective: To implement a Map Reduce algorithm that will produce modified tri-grams around the key words, after replacing the key word with '$' and to find 10 global most frequently occurring trigrams in the dataset. Keywords in the dataset are: 'science', 'fire', 'sea'

Map Logic:

- Mapper takes input from stdin. Path to input files is given in streaming command.
- Hadoop converts the input from the files into line by line format and provides the input to mapper.
- Mapper reads the input line by line, splits the input line on space to get necessary words.
- Punctuations are removed.
- We import stopwords from nltk.corpus to remove stopwords.
- Wordnet Lemmatizer from nltk.stem is used to lemmatize words.
- In order to overcome an edge case where there are more than two keywords in a trigram, we changed keywords: science to $1, fire to $2, sea to $3.
- We maintain a dictionary to map the key words back to the trigram, so that each trigram only contains one '$'.
- We have handled the case where there are two '$_'s.
- The logic in mapper divides a single trigram with two '$_'s' into two trigrams, each with on '$_'
- We then create trigrams from all the words after processing them.
- We replace $1, $2, $3 with $ and flush the map output.
- Map output is of the form (trigram, 1)

Reduce Logic:

- Reducer reads input from stdin.
- Input is a tab separated, (trigram	1).
- A dictionary is maintained to check if the trigram is seen already, where key is the trigram, value is a list of counts. Example: **<trigram: [1,1,1,1]>**
- For the line in input, line is split on \t (tab), to get trigram and count=1
- If the trigram is present in the dictionary, its count (1) is appended to the list.
- If the trigram isn't present in the dictionary, it is added to it and count is appended to the list.
- Total count of each is then calculated by calculating sum of all the elements in the list for a key.
- Then the reducer output is calculated by using a priority queue. We insert all elements into a heapq and obtain 10 most frequently occurring trigrams

Note: This is local to every reducer.

Map Logic:

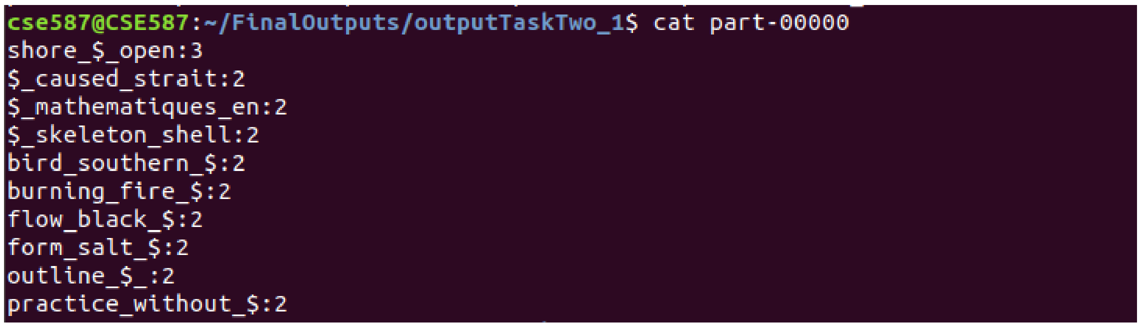
- Mapper takes input from stdin. Path to input files is given in streaming command. Here input path contains the files generated by each reducer.
- Hadoop converts the input from the files into line by line format and provides the input to mapper.
- Mapper reads the input line by line, splits the input line on space to get necessary trigrams and count from the input.
- Map output is of the form (trigram, count)

Reduce Logic:

- Reducer reads input from stdin.
- Input is a tab separated, (trigram count).
- A dictionary is maintained to check if the trigram is seen already, where key is the trigram, value is a list of counts. Example: **<trigram: [count]>**
- For the line in input, line is split on \t (tab), to get trigram and count=count
- If the trigram is present in the dictionary, its count is appended to the list.
- If the trigram isn't present in the dictionary, it is added to it and count is appended to the list.
- Then the reducer output is calculated by using a priority queue. We insert all elements into a heapq and obtain 10 most frequently occurring trigrams

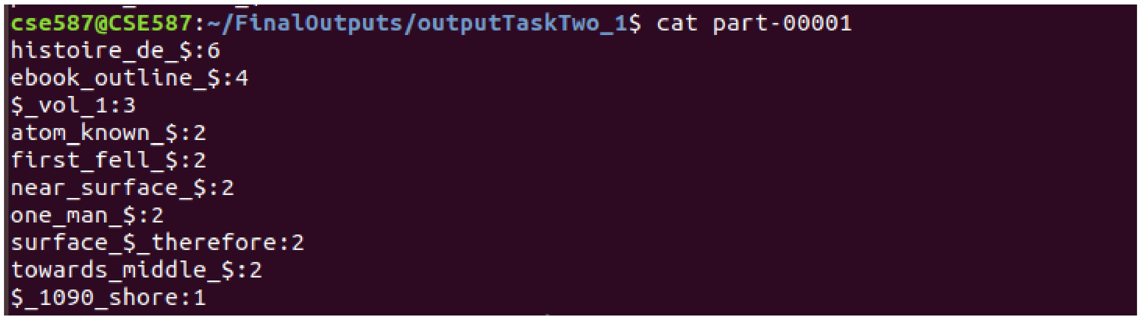
Note: This is global top 10 most frequently occurring trigrams in the dataset.

Output:



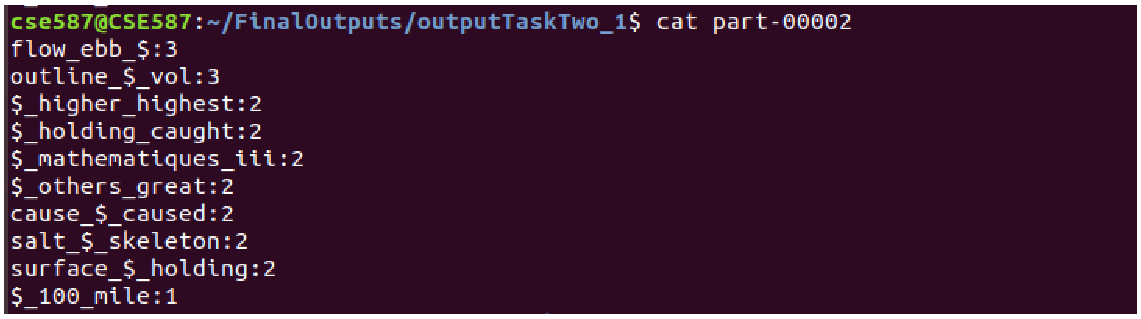
```
cse587@CSE587:~/FinalOutputs/outputTaskTwo_1$ cat part-00000
shore_$_open:3
$_caused_strait:2
$_mathematiques_en:2
$_skeleton_shell:2
bird_southern_$:2
burning_fire_$:2
flow_black_$:2
form_salt_$:2
outline_$_:2
practice_without_$:2
```

Figure 2.0: N-grams Sample Output: Reducer 1



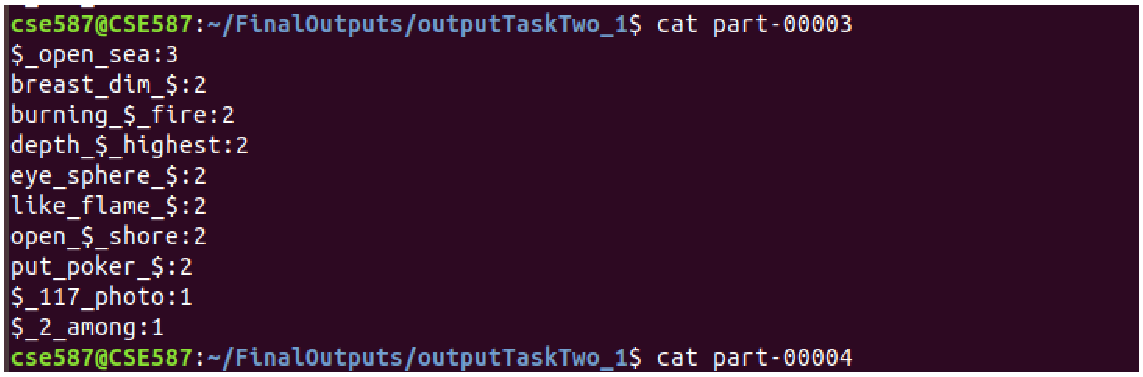
```
cse587@CSE587:~/FinalOutputs/outputTaskTwo_1$ cat part-00001
histoire_de_$:6
ebook_outline_$:4
$_vol_1:3
atom_known_$:2
first_fell_$:2
near_surface_$:2
one_man_$:2
surface_$_therefore:2
towards_middle_$:2
$_1090_shore:1
```

Figure 2.1: N-grams Sample Output: Reducer 2



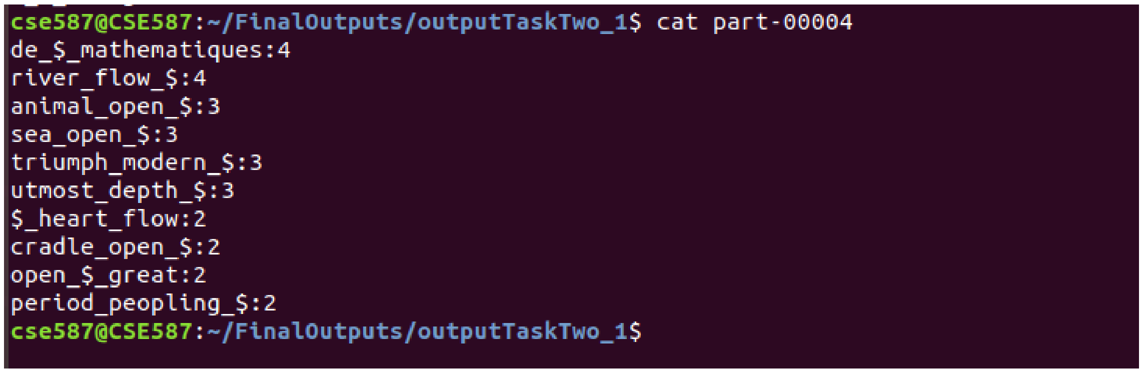
```
cse587@CSE587:~/FinalOutputs/outputTaskTwo_1$ cat part-00002
flow_ebb_$:3
outline_$_vol:3
$_higher_highest:2
$_holding_caught:2
$_mathematiques_iii:2
$_others_great:2
cause_$_caused:2
salt_$_skeleton:2
surface_$_holding:2
$_100_mile:1
```

Figure 2.2: N-grams Sample Output: Reducer 3



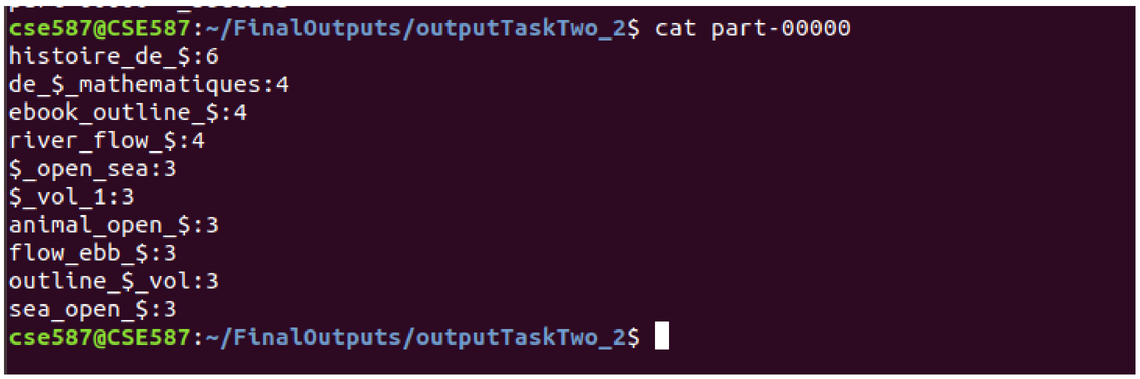
```
cse587@CSE587:~/FinalOutputs/outputTaskTwo_1$ cat part-00003
$_open_sea:3
breast_dim_$:2
burning_$_fire:2
depth_$_highest:2
eye_sphere_$:2
like_flame_$:2
open_$_shore:2
put_poker_$:2
$_117_photo:1
$_2_among:1
cse587@CSE587:~/FinalOutputs/outputTaskTwo_1$ cat part-00004
```

Figure 2.3: N-grams Sample Output: Reducer 4



```
cse587@CSE587:~/FinalOutputs/outputTaskTwo_1$ cat part-00004
de_$_mathematiques:4
river_flow_$:4
animal_open_$:3
sea_open_$:3
triumph_modern_$:3
utmost_depth_$:3
$_heart_flow:2
cradle_open_$:2
open_$_great:2
period_peopling_$:2
cse587@CSE587:~/FinalOutputs/outputTaskTwo_1$
```

Figure 2.4: N-grams Sample Output: Reducer 5



```
cse587@CSE587:~/FinalOutputs/outputTaskTwo_2$ cat part-00000
histoire_de_$:6
de_$_mathematiques:4
ebook_outline_$:4
river_flow_$:4
$_open_sea:3
$_vol_1:3
animal_open_$:3
flow_ebb_$:3
outline_$_vol:3
sea_open_$:3
cse587@CSE587:~/FinalOutputs/outputTaskTwo_2$
```

Figure 2.5: N-grams Sample Output: Final Output

3 Inverted Index

Objective: To implement a Map Reduce algorithm to produce inverted index for the whole dataset. Dataset used: Gutenberg dataset.

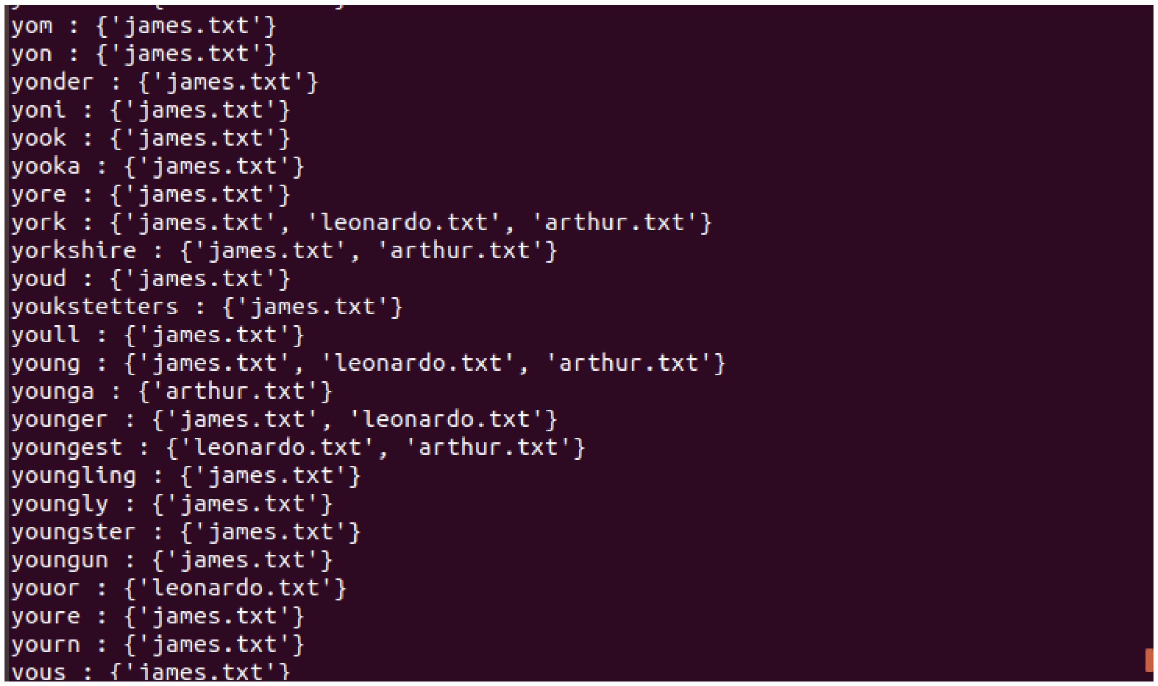
Map Logic:

- Mapper takes input from stdin. Path to input files is given in streaming command.
- Hadoop converts the input from the files into line by line format and provides the input to mapper.
- Mapper reads the input line by line, splits the input line on space to get necessary words.
- Punctuations are removed.
- We import stopwords from nltk.corpus to remove stopwords.
- Wordnet Lemmatizer from nltk.stem is used to lemmatize words.
- Document Name is then derived from the environ variable "map_file_input"
- Map output in format of (word, doc_name) is then flushed onto stdout.

Reduce Logic:

- Reducer reads input from stdin.
- Input is a tab separated, (word	doc_name).
- A dictionary is maintained to check if the word is seen already, where key is the word, value is a list of doc_names. Example: **<Word: [doc_name, doc_name, doc_name…doc_name]>**
- For the line in input, line is split on \t (tab), to get word and doc_name
- If the word is present in the dictionary, its doc_name is appended to the list.
- If the word isn't present in the dictionary, it is added to it and doc_name is appended to the list.
- The Reducer output is then calculated by taking set(doc_names) in the list for all keys and is flushed to stdout.

Output:



```
yom : {'james.txt'}
yon : {'james.txt'}
yonder : {'james.txt'}
yoni : {'james.txt'}
yook : {'james.txt'}
yooka : {'james.txt'}
yore : {'james.txt'}
york : {'james.txt', 'leonardo.txt', 'arthur.txt'}
yorkshire : {'james.txt', 'arthur.txt'}
youd : {'james.txt'}
youkstetters : {'james.txt'}
youll : {'james.txt'}
young : {'james.txt', 'leonardo.txt', 'arthur.txt'}
younga : {'arthur.txt'}
younger : {'james.txt', 'leonardo.txt'}
youngest : {'leonardo.txt', 'arthur.txt'}
youngling : {'james.txt'}
youngly : {'james.txt'}
youngster : {'james.txt'}
youngun : {'james.txt'}
youor : {'leonardo.txt'}
youre : {'james.txt'}
yourn : {'james.txt'}
yous : {'james.txt'}
```

Figure 3.0: Inverted Index Sample Output

4 Relational Join

Objective: To implement a Map Reduce algorithm to join two datasets using a primary key. Assumed Primary Key is ‘Employee ID’

Map Logic:

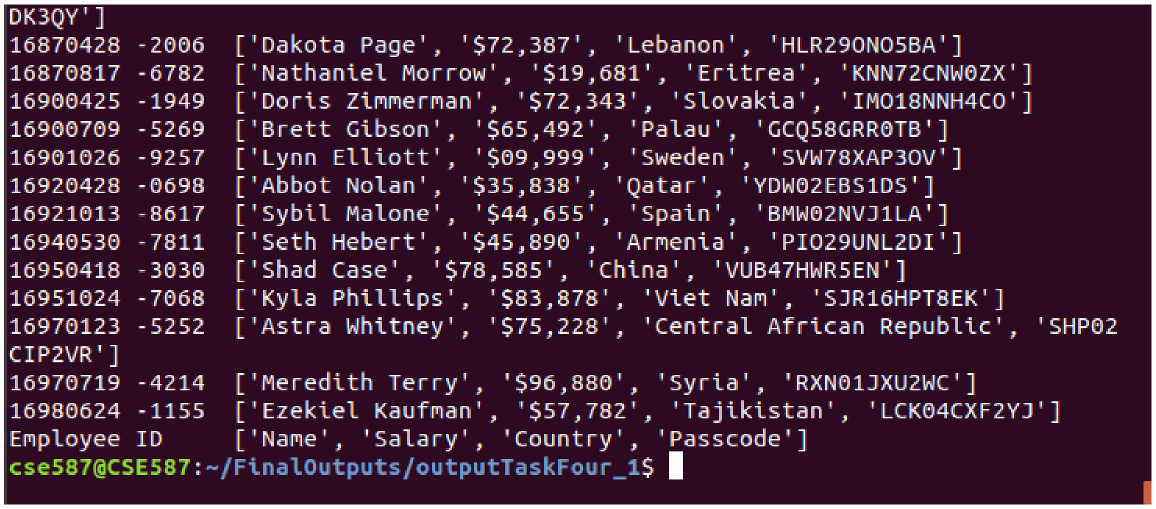
- The input files given are first converted to .csv from .xlsx
- Mapper takes input from stdin. Path to input files is given in streaming command is a csv.
- Hadoop converts the input from the files into line by line format and provides the input to mapper.
- Mapper reads the input line by line splits the line into different columns(value) and primary key as“key”
- Mapper output is modified to preserve the order of the columns in the output.
  <Primary Key(EmployeeID), (Number_of_columns+Column_number+Column_values)>

  Example:
  Sample Input Line - Employee ID; salary; country; passcode of join2.csv
  Sample Output - <Employee ID, salary>
  <Employee ID, country>
  <Employee ID, passcode>
  Sample Input Line - Employee ID; name of join1.csv
  Sample Output - <Employee ID, name>

Reduce Logic:

- Reducer reads input from stdin.
- Input is a tab separated, (key value).
- A dictionary is maintained to check if the word is seen already.
- Reducer reads the input from the mapper and joins the values to form a complete table.
- The order is preserved based on the data sent from the files. example data from file1 will be added first and data from file2 will be added later to the table. This is achieved based on the column number and file. Mapper output is modified to preserve this order.

Output:



```
DK3QY']
16870428 -2006  ['Dakota Page', '$72,387', 'Lebanon', 'HLR29ONO5BA']
16870817 -6782  ['Nathaniel Morrow', '$19,681', 'Eritrea', 'KNN72CNW0ZX']
16900425 -1949  ['Doris Zimmerman', '$72,343', 'Slovakia', 'IMO18NNH4CO']
16900709 -5269  ['Brett Gibson', '$65,492', 'Palau', 'GCQ58GRR0TB']
16901026 -9257  ['Lynn Elliott', '$09,999', 'Sweden', 'SVW78XAP3OV']
16920428 -0698  ['Abbot Nolan', '$35,838', 'Qatar', 'YDW02EBS1DS']
16921013 -8617  ['Sybil Malone', '$44,655', 'Spain', 'BMW02NVJ1LA']
16940530 -7811  ['Seth Hebert', '$45,890', 'Armenia', 'PIO29UNL2DI']
16950418 -3030  ['Shad Case', '$78,585', 'China', 'VUB47HWR5EN']
16951024 -7068  ['Kyla Phillips', '$83,878', 'Viet Nam', 'SJR16HPT8EK']
16970123 -5252  ['Astra Whitney', '$75,228', 'Central African Republic', 'SHP02
CIP2VR']
16970719 -4214  ['Meredith Terry', '$96,880', 'Syria', 'RXN01JXU2WC']
16980624 -1155  ['Ezekiel Kaufman', '$57,782', 'Tajikistan', 'LCK04CXF2YJ']
Employee ID     ['Name', 'Salary', 'Country', 'Passcode']
cse587@CSE587:~/FinalOutputs/outputTaskFour_1$
```

Figure 4: Relational Join Sample Output

5 K-Nearest Neighbors

Objective: To implement KNN using Map Reduce Paradigm. The algorithm should return the corresponding predicted label for each test instance.

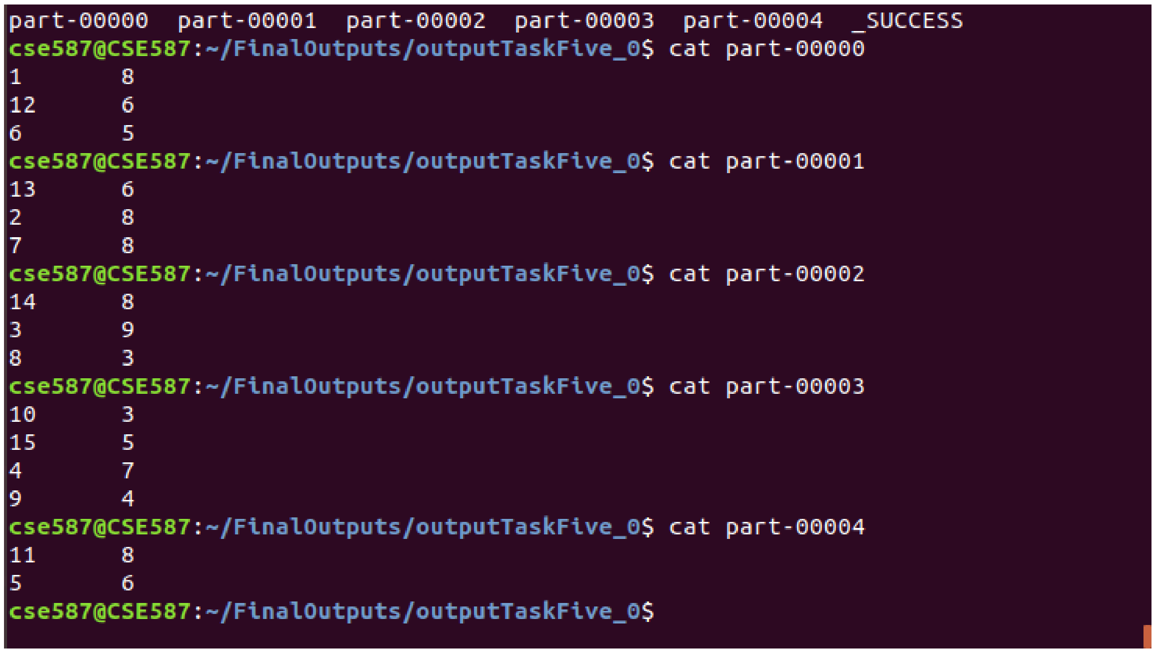
Map Logic:

- Mapper takes input from stdin. Path to input files is given in streaming command that is Train.csv. Test data is passed as a file along with streaming command.
- Hadoop converts the input from the files into line by line format and provides the input to mapper.
- Mapper opens the Test.csv file and calculates the distance for all test data for a given data point in Train.csv (stdin).
  Sample Output:
  <testcase1, "distance category">
  …..
  <testcase14, "distance category">
- We have used multiple mappers in this case (5), so that Train.csv is divided across the mappers.

Reduce Logic:

- Reducer reads input from stdin.
- Input is a tab separated, (key value).
- A dictionary is maintained to check if the word is seen already.
- A Min Heap is maintained in the reduces of size 5 and the category is predicted based on the most occurrence category in the Min Heap maintained. (K=5)
- We have used multiple reducers to split the mapper output based on key hashed.

Output:



```
part-00000  part-00001  part-00002  part-00003  part-00004  _SUCCESS
cse587@CSE587:~/FinalOutputs/outputTaskFive_0$ cat part-00000
1       8
12      6
6       5
cse587@CSE587:~/FinalOutputs/outputTaskFive_0$ cat part-00001
13      6
2       8
7       8
cse587@CSE587:~/FinalOutputs/outputTaskFive_0$ cat part-00002
14      8
3       9
8       3
cse587@CSE587:~/FinalOutputs/outputTaskFive_0$ cat part-00003
10      3
15      5
4       7
9       4
cse587@CSE587:~/FinalOutputs/outputTaskFive_0$ cat part-00004
11      8
5       6
cse587@CSE587:~/FinalOutputs/outputTaskFive_0$
```

Figure 5: KNN Sample Output

6 Streaming Commands

Task1:

```
hadoop jar /home/cse587/hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -file /home/cse587/mapper.py -mapper /home/cse587/mapper.py -file /home/cse587/reducer.py -reducer /home/cse587/reducer.py -input /home/pa2/gutenberg -output /home/pa2/videoOutputs/taskOne
```

```
hadoop fs -copyToLocal /home/pa2/videoOutputs/taskOne/ /home/cse587/videoOutputs/
```

Task2:

```
hadoop jar /home/cse587/hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -file /home/cse587/mapperTask2.py -mapper /home/cse587/mapperTask2.py -file /home/cse587/reducerTask2.py -reducer /home/cse587/reducerTask2.py -input /home/pa2/gutenberg -output /home/pa2/videoOutputs/taskTwo1 -jobconf mapred.reduce.tasks=5
```

```
hadoop jar /home/cse587/hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -file /home/cse587/mapperTask2_1.py -mapper /home/cse587/mapperTask2_1.py -file /home/cse587/reducerTask2.py -reducer /home/cse587/reducerTask2.py -input /home/pa2/videoOutputs/taskTwo1/ -output /home/pa2/videoOutputs/taskTwo2
```

```
hadoop fs -copyToLocal /home/pa2/videoOutputs/taskTwo1/ /home/cse587/videoOutputs/
```

```
hadoop fs -copyToLocal /home/pa2/videoOutputs/taskTwo2/ /home/cse587/videoOutputs/
```

Task3:

```
hadoop jar /home/cse587/hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -file /home/cse587/mapperTask3.py -mapper /home/cse587/mapperTask3.py -file /home/cse587/reducerTask3.py -reducer /home/cse587/reducerTask3.py -input /home/pa2/gutenberg -output /home/pa2/videoOutputs/taskThree
```

```
hadoop fs -copyToLocal /home/pa2/videoOutputs/taskThree/ /home/cse587/videoOutputs/
```

Task4:

```
hadoop jar /home/cse587/hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -file /home/cse587/mapperDataSet.py -mapper /home/cse587/mapperDataSet.py -file /home/cse587/reducerDataSet.py -reducer /home/cse587/reducerDataSet.py -input /home/pa2/taskFour -output /home/pa2/videoOutputs/taskFour
```

```
hadoop fs -copyToLocal /home/pa2/videoOutputs/taskFour/ /home/cse587/videoOutputs/
```

Task5:

```
hadoop jar /home/cse587/hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -D mapred.map.tasks=5 -file /home/cse587/mapperKNN.py -mapper /home/cse587/mapperKNN.py -file /home/cse587/reducerKNN.py -reducer /home/cse587/reducerKNN.py -input /home/pa2/project2/KNN -output /home/pa2/videoOutputs/taskFive -file ./Test.csv
```

```
hadoop fs -copyToLocal /home/pa2/videoOutputs/taskFive/ /home/cse587/videoOutputs/
```