CSE 587 – Assignment 2 Report

<u>Team members:</u>

Srikanth Ammineni
UB ID: samminen
UB person#: 50317818

Goutham Krishna Reddy Sagam
UB ID: gsagam
UB person#: 50313948

Hemanth Inakollu
UB ID: hemanthi
UB person#: 50316838

1. Word count

Input: Gutenberg dataset (arthur.txt, james.txt, leonardo.txt)

Files were copied to HDFS which will serve as input for the MapReduce program.

Map reduce logic:

Mapper:

1) Reads input from HDFS using stdin
2) For each line:

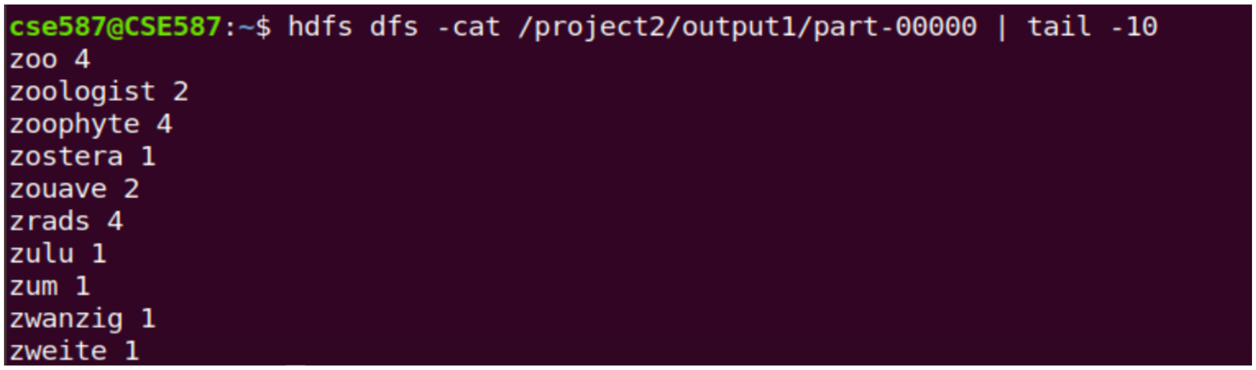
```
        tokenize the line using whitespace as delimiter
        emit (token,1)
```

Even if the same word appears again, it will be again emitted as <token, 1>

Reducer:

1) Reads the output from mapper using stdin
2) Since the reducer receives input in the sorted manner. Counts will be aggregated for each word
3) emit (token, aggregated count)

Ouput: Counts for each word present in corpus



```
cse587@CSE587:~$ hdfs dfs -cat /project2/output1/part-00000 | tail -10
zoo 4
zoologist 2
zoophyte 4
zostera 1
zouave 2
zrads 4
zulu 1
zum 1
zwanzig 1
zweite 1
```

2. N-grams

Input: Gutenberg dataset (arthur.txt, james.txt, leonardo.txt)

Files were copied to HDFS which will serve as input for the MapReduce program.

Map reduce logic:

There will be two maps reduce programs.

First map reduce program will find all the tri grams and it's count for the given key words and outputs local top 10 most occurring tri grams for each reducer.

Mapper:

1) Reads input from HDFS using stdin
2) For each line:
   Club last 1 or 2 words in previous line to the current line as they can also form trigrams
   Get tri-grams using nltk package
   Replace keywords in trigrams with "$".
   emit (tri-gram having "$", 1)

Even if the same tri-gram appears again, it will be again emitted as <tri-gram, 1>

Reducer:

1) Reads the output from mapper using stdin
2) Since the reducer receives input in the sorted manner. Counts will be aggregated for each trigram
3) emit local top 10 most occurring tri-grams

Second map reduce program will aggregate the output from first map reduce program and output the top 10 most occurring tri grams with their counts

Mapper:

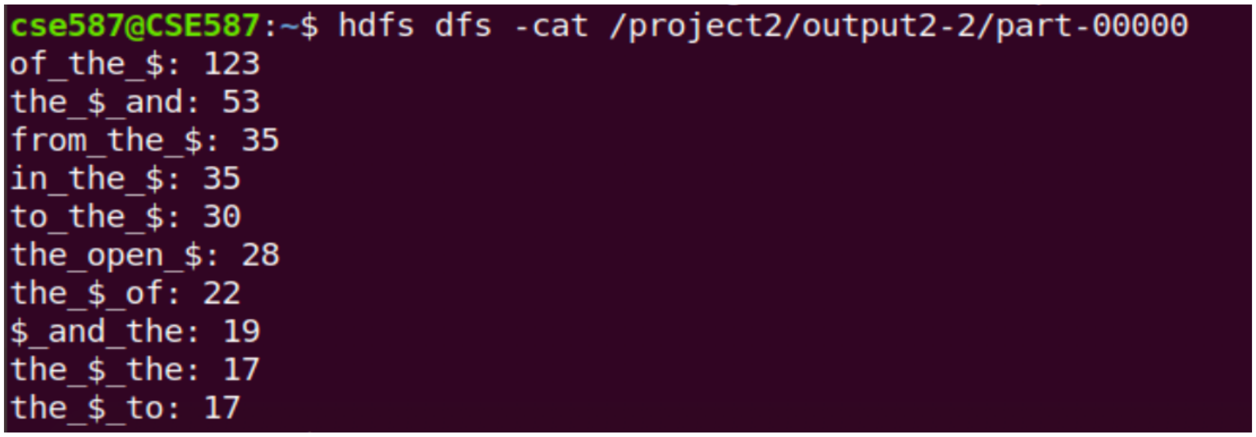
1) Gets output from first map reduce program
2) For each line:
   emit(line)

Output of the first map reduce program is passed on to reducer as is.

Reducer:

1) Reads the output from mapper using stdin
2) Aggregate the local top 10 most occurring trigrams
3) emit the global top 10 most occurring trigrams

Ouput: Counts for 10 most occurring trigrams in the corpus



```
cse587@CSE587:~$ hdfs dfs -cat /project2/output2-2/part-00000
of_the_$: 123
the_$_and: 53
from_the_$: 35
in_the_$: 35
to_the_$: 30
the_open_$: 28
the_$_of: 22
$_and_the: 19
the_$_the: 17
the_$_to: 17
```

3. Inverted Index

Input: Gutenberg dataset (arthur.txt, james.txt, leonardo.txt)

Files were copied to HDFS which will serve as input for the MapReduce program.

Map reduce logic:

Mapper:

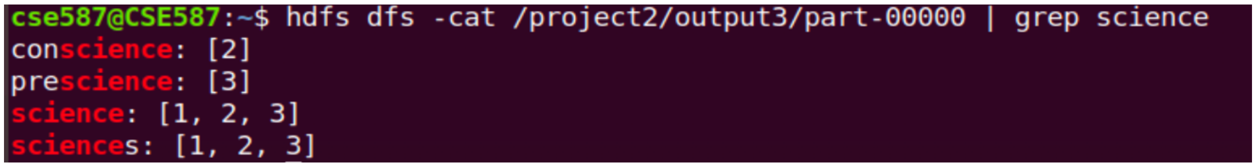
```
1) Reads input from HDFS using stdin
2) For each line:
        tokenize the line using whitespace as delimiter
        get file name that is currently being read
        Map file name to doc_id
        emit (token, doc_id)
```

if a same word appears again in another document, it will be again emitted as <token, corresponding doc id>

Reducer:

1) Reads the output from mapper using stdin
2) Since the reducer receives input in the sorted manner, All the doc ids will be aggregated as a list for each word.
3) emit (token, list of doc ids)

Output: Inverted index of the whole corpus



```
cse587@CSE587:~$ hdfs dfs -cat /project2/output3/part-00000 | grep science
conscience: [2]
prescience: [3]
science: [1, 2, 3]
sciences: [1, 2, 3]
```

4. Relational Join

Input: Join1.csv, Join2.csv

Actual inputs were given as excel files but they were converted to csv for ease of reading programmatically.

Files were copied to HDFS which will serve as input for the MapReduce program.

Map reduce logic:

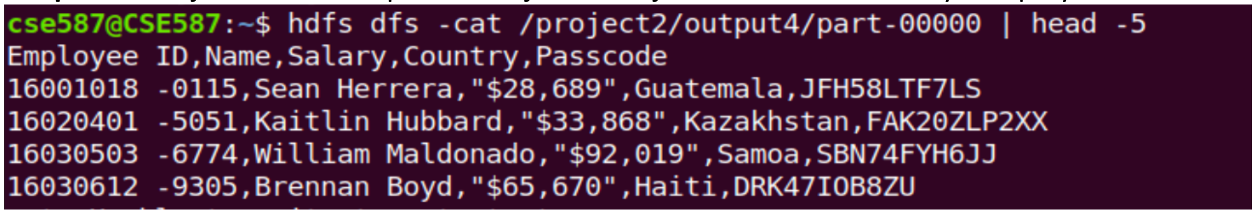
Mapper:

```
1) Reads input from HDFS using stdin
2) For each line:
      If record from Join1 file:
            Emit the record as is:  <Employee ID, Name>
      Else if record from Join2 file:
            Add a column <Name> with value as Null
            Emit the modified record as below
            <Employee ID, Name (Set as Null), Salary, Country, Passcode>
```

Reducer:

1) Reads the output from mapper using stdin
2) Since the reducer receives the records in the sorted manner, Update the join2 record with Employee name from Join1 record.
3) emit the updated record

Output: Inner join of records present in join1 and join2 files based on key “Employee ID”



```
cse587@CSE587:~$ hdfs dfs -cat /project2/output4/part-00000 | head -5
Employee ID,Name,Salary,Country,Passcode
16001018 -0115,Sean Herrera,"$28,689",Guatemala,JFH58LTF7LS
16020401 -5051,Kaitlin Hubbard,"$33,868",Kazakhstan,FAK20ZLP2XX
16030503 -6774,William Maldonado,"$92,019",Samoa,SBN74FYH6JJ
16030612 -9305,Brennan Boyd,"$65,670",Haiti,DRK47IOB8ZU
```

5. KNN

Input: Train.csv, Test.csv

Test.csv will be visible to every node. It will be passed on to each mapper using -file option

Files were copied to HDFS which will serve as input for the MapReduce program.

Map reduce logic:

Mapper:

1) Reads Training data input from HDFS using stdin
2) For each record in training data:
   Find Euclidean distance from all the records in test data
   Add distance and label to corresponding test records
   Emit test records with distance and label: <Test sample, distance, label>

Reducer:

1) Reads the output from mapper using stdin
2) Since the reducer receives the test records with distances and labels in the sorted manner, Get K smallest distances and their respective labels
3) Assign predicted value as most occurring label from the k labels
4) emit <Test sample, predicted value>

Output: corresponding predicted label for each test instance