ASSIGNMENT 2

CSE 587-DATA INTENSIVE COMPUTING

Video Link - Drive
https://drive.google.com/file/d/1BLCiaMz6_6EeVflhNo6p9-t3SCdXp8T9/view?usp=sharing

You Tube https://youtu.be/h1XV85alZf0

Part 1 - Setup and WordCount

Goal: To produce the count of every word in the document by using the MapReduce algorithm.

Dataset: Gutenberg

Language: Python3

Data Pre-processing steps:

1. Libraries used:
    a. Regex - used to remove all special characters
    b. Nltk - used for removing stop words and stemming

Functionalities in 'mapper_1.py' and 'reducer_1.py':

1. **Mapper**
    a. The gutenberg dataset is loaded into the hdfs environment.
    b. Each line of the 3 text files present in the Gutenberg folder is read line by line and they are split into words.
    c. Special and other unwanted characters are removed from each word by using Regex and Nltk libraries.
    d. Finally the cleaned up words from all the 3 files along with the word count is sent to the reducer as the output.
2. **Reducer**
    a. The data from the mapper is read and a dictionary is created to store the words along with their counts.
    b. The words are arranged line by line so that corresponding counts are printed along with it.

c. So the program check list by list that is each list contains a single word.
d. After reading the list of words it gives the count as values to the dictionary that is created corresponding to the keys that are words.
e. Finally the dictionary is printed as the reducer output.

Output: The Gutenberg folder is copied into hadoop environment using the following command : hdfs dfs -copyFromLocal ./gutenberg/ /home/demopy26

After copying the required folder to hadoop environment the following command was used in order to run so that mapper and reducer code runs in hdfs environment
hadoop jar
/home/cse587/hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -file /home/cse587/mapper_1.py -mapper /home/cse587/mapper_1.py -file /home/cse587/reducer_1.py -reducer /home/cse587/reducer_1.py -input /home/demopy26 -output /home/demopy26/out1

The output with the words and their corresponding count of occurences is copied to a 'part_1_output.txt' file.

Part 2 - N-Grams

Goal: To implement a map reduce algorithm that will produce top 10 tri-grams around the keywords(science, sea, fire) from the given dataset after making the keyword with $.

Dataset: Gutenberg [3 novel's text]

Language: Python3

DataSet Pre Processing: Stemming and Stopword removal of the sentences in the novel. We used nltk stemming and stopwards library to achieve this.

MapReduce Jobs:

To achieve this, we use Hadoop Streaming and Map Reduce Jobs. We can run two map reduce jobs to complete this task. MapReduce Job 1 can use multiple reducers to create multiple local top 10 trigram lists from the dataset and Map Reduce Job 2 can aggregate all the local top 10 trigram lists to create a final global top 10 trigrams list.

<u>MapReduce Job 1:</u>

Mapper Job goes over the streaming data which comes in form a sentence. It pre-processes the data like removing non-alphanumeric characters and removing stop words before tokenizing the sentence. It extracts all possible trigrams from the tokenized sentence and sends those trigrams to reducers that have the specified keys (science, sea, fire) after masking the key with a $ character. While sending this data to the reducers, we send the sentence as the key so that trigrams are distributed evenly across available reducers.

Reducer Job keeps track of the incoming data from Mapper using a hashmap where it stores the trigrams as key and its number of occurrences as its value. Once all the data has streamed, it uses a min heap of size 10 to keep top 10 trigrams which is outputted as the result of the Reducer. Each reducer outputs its set of 10 top trigrams.

We used three reducers for this job which resulted in three local top 10 trigram lists.

Command:

```
cse587@CSE587:~$ hadoop jar /home/cse587/hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -Dmapred.reduce.tasks=3 -file /home/cs
e587/assignment2/trigram/mapper.py -mapper /home/cse587/assignment2/trigram/mapper.py -file /home/cse587/assignment2/trigram/reducer.py -reduc
er /home/cse587/assignment2/trigram/reducer.py -input home/assignment2/ -output home/assignment2/out
```

MapReduce Job 2:

Mapper Job is the identity mapper which takes input from the output of the first map reduce job (local top 10 trigram lists) and sends it to the reducers using a constant key so that every trigram goes to the single reducer even if this job applies multiple reducers. This is important to produce an aggregated global top 10 trigram list. Since the amount of data is a lot less (10 x number of reducers in MapReduceJob 1) than the amount of data streamed in MapReduce Job 1, Applying a single reducer for this job should not be a problem.

Reducer Job is the same job that runs in the first MapReduce Job.

Command:

```
cse587@CSE587:~$ hadoop jar /home/cse587/hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -Dmapred.reduce.tasks=3 -file /home/cs
e587/assignment2/trigram/identity_mapper.py -mapper /home/cse587/assignment2/trigram/identity_mapper.py -file /home/cse587/assignment2/trigram
/reducer.py -reducer /home/cse587/assignment2/trigram/reducer.py -input home/assignment2/out/ -output home/assignment2/out2
```

Output:

In the end of the two map reduce jobs, we get a global list of top 10 tri-grams around the keywords (science, fire, sea) in the given dataset. This list is copied to a 'part_2_output.txt' file.

Part 3 - Inverted Index

Goal: To implement an inverted index, a mapping from a word to its location within a file, using the Hadoop MapReduce algorithm for the given dataset.

Dataset : Gutenberg

Language: Python3

Data Pre-processing steps:

1. Libraries used:
    c. Regex - used to remove all special characters
    d. Nltk - used for removing stop words and stemming

Functionalities in 'mapper_3.py' and 'reducer_3.py':

1. **Mapper**
    a. The gutenberg dataset is copied to the hdfs.
    b. OS module is used to get the location of the input files inside the hdfs.
    c. Each line from the file is read and data preprocessing is done using the above 2 libraries.
    d. Each word from the file along with their filename(docid) is sent to the reducer as the output.
2. **Reducer**
    a. The data from the mapper is read and a dictionary is maintained to store the inverted index.
    b. The words and their corresponding docids are stored as key value pairs inside the dictionary.
    c. If the word is not present in the dictionary, it's been added to it with the docid.
    d. If the word is present and if the docid is not present, the docids are appended as values of the word as a list.
    e. Finally the dictionary is printed as the reducer output.

Output: The following command is used to run the mapper and reducer files

hadoop jar
/home/cse587/hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -file
/home/cse587/mapper_3.py -mapper "python3 mapper_3.py" -file
/home/cse587/reducer_3.py -reducer "python3 reducer_3.py" -input
/home/assignment_2/gutenberg -output /home/assignment_2/output

The output with the words and their corresponding inverted indexes is copied to a 'part_3_output.txt' file.

Part 4 - Relational Join

Goal: We need to join two tables that are present in different files(excel or csv) that are join1(excel/csv) and join 2(excel/csv) using employee as primary key.

Dataset: Join1(and Join2(Excel/csv format file)

Language: Python3

Functionalities in 'mapper_4.py' and 'reducer_4.py':

1. **Mapper**
    a. We are basically reading both the files and splitting it by comma separated.
    b. Firstly, we are assigning default values as -1 string so that whenever the data is read from one file the remaining columns that do not belong to that particular file are specified as -1.
    c. We are finding if the data from join1 file or join2 file based on the list that is returned(split) so if the split has 2 indexes then it is from join1 file else it is from join2.
    d. Finally, we print the values of both the tables separately and pass it to the reducer.
2. **Reducer**
    a. The data from the mapper is read and two dictionaries are created to store the data that is present in each file separately.
    b. Similar to the process that was done in mapper, strip and split the data and store it as single values.
    c. If the the table that is fetched as passcode as -1 (that is it does not have a value) then it is reading the join1 file table else if the condition

is false then it is reading join2 file table and we are storing these two tables as dictionaries based on employee id as key and remaining columns as values corresponding to it.

d. Finally we are iterating based on employee_id on both dictionaries and fetching the corresponding values.
e. Finally the dictionary is printed as the reducer output.

Output: The join folder is copied into hadoop environment using the following command: hdfs dfs -copyFromLocal ./join/ /home/demopy26

After copying the required folder to hadoop environment the following command was used in order to run so that mapper and reducer code runs in hdfs environment
hadoop jar /home/cse587/hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -file /home/cse587/mapper_4.py -mapper /home/cse587/mapper_4.py -file /home/cse587/reducer_4.py -reducer /home/cse587/reducer_4.py -input /home/demopy26 -output /home/demopy26/out1

The output with the relational join using employee_Id as primary key is copied to a 'part_4_output.txt' file.

Part 5 - K-Nearest Neighbour

Goal: To implement a K-Nearest Neighbor, a supervised machine learning algorithm, using the Hadoop MapReduce algorithm for the given dataset.

Dataset: Train.csv and Test.csv

Language: Python3

Data Pre-processing steps:

1. Data preprocessing is done outside the hdfs using a seperate script named 'Script.py'.
2. Libraries used:
    a. numpy- used for data preprocessing.
    b. pandas- used for reading data and processing the dataset

Functionalities in 'script.py ', 'mapper_5.py' and 'reducer_5.py':

1. **Script**

a. Train and test data are processed using pandas library
b. Minmax normalization is done on these data.
c. The normalised data is being stored as separate CSV files with the names ‘Train_norm.csv’ and ‘Test_norm.csv’ in the local system.

2. **Mapper**
   a. Normalized Test files are loaded using numpy genfromtxt module.
   b. The euclidean distances between the row in normalized train data with every row in test data are calculated.
   c. The euclidean distances along with their corresponding test rows and train labels are sent to the reducers as output.
3. **Reducer**
   a. The data from the mapper is read and a dictionary is maintained to store the results from the mapper.
   b. The test data and their corresponding euclidean distances between train data are stored as key value pairs inside the dictionary.
   c. The distances are sorted in the ascending order and the top k(k being 6) neighbours are selected.
   d. The test data along with the most common occurrences of the labels inside the k neighbours are stored as a key value pair in a new dictionary.
   e. Finally the dictionary is printed as the reducer output.

Output: The following command is used to run the mapper and reducer files

```
cse587@CSE587:~$ hadoop jar /home/cse587/hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -file /home/cse587/assignment2/knn/Tes
t_norm.csv -file /home/cse587/assignment2/knn/mapper.py -mapper /home/cse587/assignment2/knn/mapper.py -file /home/cse587/assignment2/knn/redu
cer.py -reducer /home/cse587/assignment2/knn/reducer.py -input home/knn/ -output home/knn/out
```

The output with the test rows along with their corresponding labels is copied to a ‘part_5_output.txt’ file.

Team Contribution

Team Members	UB IT Name	UB Email	Parts
Viswanathan Gopalakrishnan	vgopalak	vgopalak@buffalo.edu	1 and 4
Kundan Kumar	kkumar5	kkumar5@buffalo.edu	2 and KNN
Varsha Ravichandiran	varshara	varshara@buffalo.edu	3 and KNN