[Year]

CSE 587: Data Intensive Computing
Assignment 2

Team Members:

Amit Upadhyay: 50336909 amitupad@buffalo.edu
Namrata Bakre: 50336849 nbakre@buffalo.edu
Nitin Kulkarni: 50337029 nitinvis@buffalo.edu

Aim:

- Setting up a virtual machine (OS- Linux) and Hadoop framework and getting familiar with various Hadoop framework process and commands
- A series of text documents are given and the frequency of each word in these documents(combined) is to be found using a MapReduce algorithm.

Steps Taken:

As SAS implies: Hadoop is an open-source software framework for storing data and running applications on clusters of commodity hardware. It provides massive storage for any kind of data, enormous processing power and the ability to handle virtually limitless concurrent tasks or jobs.

- A virtual machine called Oracle Virtual VM box was installed and the OS requested was Linux and version -Ubuntu 64bits.
- Hadoop 3.1.2 was installed.
- Using the command start-all.sh to start all Hadoop daemons, i.e., the namenode, datanodes, the jobtracker and tasktrackers.
- The input files were transferred to hdfs using copyFromLocal command.
- The file was executed using jar, mapper and reduce files at the hdfs.
- The output was brought back to local drive using copyToLocal command.

Implemented Logic:

We have used two files for each part of implementation:

1. Mapper.py (For Mapper)
2. Reducer.py (For Reducer)

PART 1 - Setup and Word Count:

Mapper:

- As the location of file and name can be anything, STDIN is being used.
- STDIN reads all the data from various input files and sends its in one by one order to mapper file as input.
- In mapper file, all the words from every file returns a key value pair with keys as the word and value as “1” (because every word, regardless of their frequency of repetition, is being considered)

Reducer:

- Again, using STDIN which inputs reducer the output generated from mapper file.
- Here, the frequency of each word is merged (regardless of their caps status (i.e., if they are lower case or smaller case or a combination of two). The output again is a key-value pair with key as every word which appears in all the files and its value being its total frequency. If the words are unique, i.e., they appear only ones, their value is 1 else, it's the number of times they appear.

PART 2 - N-grams:

Mapper:

For getting the tri-grams from the text files in the Gutenberg folder, I used the Ngram module from the nltk library.
Initially, I have created an empty dictionary in which I will be storing the trigram as the key and the value as 1. I am reading the files through sys.stdin.
I am then converting all the text to lowercase so that it's uniform and out keywords are correctly identified.
Then I am calling the ngrams module from nltk and getting the tri-grams. Then for each tri-gram I am doing basic text cleaning through re library. I am then searching for the key words in the tri-grams and only outputting those tri-grams which have the keywords in them. Finally, I am replacing the keywords in the tri-grams with a '$' symbol and then adding the tri-gram to the dictionary I have initially created as the key and setting the value to be 1.
Finally, I am printing the dictionary key and value which will be used as the input to the reducer function.

Reducer:

I am first creating a dictionary in which I will be storing the tri-gram as key and the count of that particular tri-gram as the value.
I am reading the output of the mapper through sys.stdin, then I am getting the trigrams and appending them as keys to the dictionary. If the tri-gram already exists in the dictionary I increase the value by 1 and if it doesn't then I will set the value to 1.
I then sort the dictionary and do basic text cleaning with the re library. Finally, I will display the 10 most common modified tri-grams and their count.

PART 3 - Inverted Index:

Mapper:

This file reads from stdin then splits the line with spaces and stores it as a list. For each word in the list, fetch the file name of current file using get_filename() function which is user defined. We are using os.environ['map_input_file'] which is set by Hadoop framework to get the filename. After getting the filename, process the word that is first lower the case and then keep only required characters as valid:
a-z, 0-9, ' –
Output of the Mapper is Word as key and Value as filename which is printed to stdout.

Reducer:

Created an empty dictionary at the beginning. This file reads from stdin and gets the key, value pair. If word does not exist in the dictionary, insert the word and append it with the filename. If word exists in the dictionary and filename doesn't exists as its value, then append the filename. If both word and filenames in which word is found exists in the dictionary then do nothing. At the end, print the dictionary which will have words and files in which the respective word is found.

PART 4 - Relational Join:

Mapper:

As the location of file and name can be anything, STDIN (sys.stdin) is being used.
Every row from both datasets was printed depending of their sizes. Each row was converted into a list where each element is j^{th} column in i^{th} row. Each row was printed.

Reducer:

The input to the reducer file is the output of the Mapper file which was taken in using sys.stdin. A dictionary was initialized. Iteration over each row in the STDIN is done and adding each row and each element into the dictionary with Employee ID as the primary key and finally appending the rest of the row to the value, making it a key value pair.

BONUS – K-Nearest Neighbor:

Mapper:

For implementing KNN I am first reading the dataset and doing basic preprocessing on the data such as splitting the data and labels and normalizing the data using the MinMaxScaler module from sklearn.
Then I have created a function named 'euclidean_distance' which takes the row number as the parameter. It first calculates the distance of a given row in the test dataset with all the rows in the training dataset and stores it in the dictionary whose key is the row number and the value is the Euclidean distance.
I then sort the distances from lowest to highest and returns the first 7 values (As we are considering 7 neighbors). Then I am doing basic text cleaning using the re library and finally using the row number (i.e., the key of the dictionary) then print the row and neighbor from the label of the training dataset. These printed values are then used as input by the reducer.

Reducer:

For the reducer, I take as input the values printed by the mapper using sys.stdin. I split, the input so I get the row number and the neighbors separately. I then append these in two separate lists. I then find out the unique values from the row numbers and sort them.
For prediction I created a list in which I will store the predicted labels. Then I append the value of the neighbor which is repeated the most into the list.
Finally, I print the row number and the predicted label.