DATA INTENSIVE COMPUTING

ASSIGNMENT 2

Big Data Processing with Hadoop

**Prakhathi Murugesan- 5031 6846**

**Saranya Saravanan- 5031 4931**

**Vineeth Thayanithi- 5032 0197**

**PART-1: WORD COUNT:**

Word count is the number of times a word has occurred in the given document. In our solution, we would be finding the number of occurrences for each distinct word that is present in the given set of documents. By implementing this we might further use this output for other textual analyses. This is implemented using the concept of MapReduce where the execution contains two parts i.e Mapper and Reducer.

**Functionality executed by Mapper:**

* Import libraries for the regular expression
* Import the sys libraries
* For each line that is given as input to the mapper program from sys.stdin using the 3 files
  + Convert the line into lowercase.
  + Strip newlines from the line.
  + Replace non-alphanumeric characters from the line with blank space.
  + Split the line into individual words and store them in an array .
  + Print **word \t 1** for all words in the array. This will be the output from the mapper.
  + Clear the array and repeat for all lines.

**Functionality executed by Reducer:**

* Import sys and itemgetter from operator
* For each line from the output of the mapper that is read using the sys.stdin
  + Check if the length is greater than 2 to avoid getting inputs with only a newline
    - If it is remove the newline and split the line into two words, one is the word and the other is the count
    - Typecast the count to int
    - If the word is equal to the previous word add the count of the word and the previous count for that word
    - If it is not add the current word to a list and set word and the count as current words
* Finally, add the last word to the list
* Print all the words and their counts

**Files Used:**

* Arthur.txt
* James.txt
* Leonardo.txt
* Mapper.py
* Reducer.py

**PART-2: Top 10 Trigram:**

A trigram is defined as a group of 3 consecutive words. The goal of this task is given a set of keywords, we would want to generate all possible trigrams for that keyword. A given trigram is a trigram for a keyword if the keyword occurs at least once in the trigram. After finding all the trigrams for the keywords, we want to find the 10 most occurring trigrams. This is implemented using the concept of MapReduce where the execution contains two parts i.e Mapper and Reducer. We run multiple reducers for this task and hence we would require two mappers and two reducers.

**Functionality executed by Mapper1:**

* Import libraries for the regular expression
* Import the sys libraries
* For each line that is given as input to the mapper program from sys.stdin using the 3 files
  + Convert the line into lowercase.
  + Strip newlines from the line.
  + Replace non-alphanumeric characters from the line with blank space.
  + Split the line into individual words and store them in a data array.
  + Add all of these words into the final array.
  + Clear the data array and repeat for all lines.
* Store the keys in an array
  + For each of the keys
    - Find all the occurrences of the key in the final array.
    - For all of the occurrences of the keyword in the document
      * Construct 3 trigrams (1) Keyword replaced with $ as the last word of the trigram (2)keyword replaced with $ as the second word as the trigram (3) Keyword replaced with $ as the third trigram

**Functionality executed by Reducer1:**

* Import sys and itemgetter from operator
* For each line from the output of the mapper that is read using the sys.stdin
  + Check if the length is greater than 2 to avoid getting inputs with only a newline
    - If it is remove the newline and split the line into two words, one is the word and the other is the count
    - Typecast the count to int
    - If the word is equal to the previous word add the count of the word and the previous count for that word
    - If it is not, add the current word to a list and set word and the count as current words
* Finally, add the last word to the list.
* Sort the list with respect to the counts.
* Print the first 10 items from the list.

**Functionality executed by Mapper2:**

* Mapper2 gets input from sys.stdin
* For each line of the input
  + Print the line and this will be the input to the reducer2.

**Functionality executed by Reducer2:**

* Reducer2 performs the same function as reducer 1.

**Files Used**:

* Arthur.txt
* James.txt
* Leonardo.txt
* Mapper.py
* Reducer.py

**PART-3: INVERTED INDEX:**

Inverted index stores a mapping from the content. In our solution, we will be mapping each word present in any of the three files namely arthur.txt, james.txt and leonardo.txt to the location that is the name of the file/files in which the word is constituted.

This is implemented using the concept of MapReduce where the execution contains two parts i.e Mapper and Reducer. The mapper and reducer files executed here are mapnew.py and rednew.py

The output here is stored in three text files namely output\_part-00000.txt, output\_part-00001.txt, output\_part-00002.txt

**Functionality executed by Mapper:**

* Import libraries for the regular expression, stopwords and stemming
* Import the list of English stopwords from corpus file of nltk package
* The input from the text file is one line at a time and the code for the mapper is iterated through the set of lines.
  + The OS module from the standard utility library of python is used to access a function in it to interact with the operating system.
  + The “map\_input\_file” environment variable is used to retrieve the file on which the mapper is currently executing.
  + The “map\_input\_file” is passed as a key to the os.getenv() which returns the value of the environment variable key and is stored in a string File\_path.
  + The “File\_path” string is split to retrieve only the name of the text file from the entire path.
  + The line is converted to lowercase
  + The data is retrieved from the line using the regular expression to get only the alphanumeric words ([^A-Za-z\s]) and split and stored as a list.
  + Stemming is performed on the retrieved word using the PorterStemmer()
  + The stemmed word is checked for stop word by comparing it with a set of predefined stopwords. If the word is not a stopword then the below line is executed
    - Each term in the list is read and print as the output of the mapper in the following format
      * **Word <\t> Respective\_File\_name**

**Functionality executed by Reducer:**

* A dictionary is created.
* The reducer takes the output from the mapper as its input.
* Each input is a line consisting of a word and its filename separated by a tab. The reducer code is iterated through the lines of input.
  + The input is split by using the tab as the delimiter and stored in respective variables word and posting.
  + The word is stored as the key and the posting is stored as the value to the word i.e the word and posting are stored in the dictionary as a key-value pair.
  + The dictionary does not contain any duplicates.
* The dictionary now contains all the words/terms and the names of all the text files in which the word is present.
* Iterate through each word(key) in the dictionary
  + Iterate through each filename(value) of the key and append it to a list example: postings.
  + Join a comma ”,” in-between each index of the list “postings”.
  + Output or print the word and the list of postings in the following format
    - Word <\t> postings[]

**Files Used:**

* Arthur.txt
* James.txt
* Leonardo.txt
* mapnew.py
* rednew.py

**PART-4 RELATIONAL JOIN:**

For the given two tables, Relational Join is implemented by using MapReduce. The first table contains Employee Id and Name. The second table contains Employee Id, Salary, Country, and Passcode. Relational Join of these two tables would have Employee Id, Name, Salary, Country and Passcode. The Join of these two tables is done by using the primary key which is Employee Id.

This is implemented using MapReduce where the execution contains two parts i.e Mapper and Reducer.

**Functionality executed by Mapper:**

* Given data is converted to CSV format and both tables are passed as input to the mapper.
* In Hadoop, one line from the input is passed to the mapper at a time. In this case, one line from any of the two tables is passed at a time to the mapper.
* Each line is then pre-processed in mapper. An array is created by splitting the line based on comma(,).
* Since salary has a currency format, it has a comma in it. Therefore, salary has split into two parts. This case has been handled in mapper by concatenating these two parts and inserted into the array.
* The output from the mapper is a key-value pair. In this case, the key is the primary id. The first element in the array gives the employee id which is used as the key and the rest of the array is used as the value.
* The array is converted to a string format separated by comma and then passed as output. This speeds up the MapReduce computation significantly.
* To differentiate key and value, the tab is given in between them. By default, the first tab-spaced element is taken as key and rest is taken as value. When there is no tab specified in the output, the entire output is taken as key and value is taken as null.

**Functionality executed by Reducer:**

* The output from the mapper is passed as an input to the reducer.
* Every output from the mapper with the same key is passed to one reducer.
* Each input to the reducer is of string format so, each line is then split by tab.
* The first element indicates the key and rest are value.
* The value is again split by comma and is stored in an array.
* If the length of the array is more than 1, then the input is from the second table so, it is stored in details\_dict dictionary with employee\_id as key.
* Otherwise, the input is from the first table and it is stored in emp\_dict.
* The two dictionaries are iterated over the key and employee id, name, salary, country, and passcode are taken from the list and sent it as output.

Files Used:

* Join1.csv
* Join2.csv
* Mapper.py
* Reducer.py

**PART-5 KNN (K-Nearest Neighbour):**

KNN is one of the supervised machine learning algorithm used to solve both classification and regression problems. This is a non-parametric lazy learning algorithm which means that the algorithm doesn’t learn any pattern from the training data but it predicts the label based on feature similarity.

KNN- Algorithm:

* The algorithm works on the feature similarity. To do this, the euclidean distance between each test data and train data is calculated.
* The distances are sorted in ascending order and top K closest distances are selected.
* The corresponding labels of top k distances are picked.
* The most occurred label among the K labels is found and this is the predicted label for the given test data.
* K value is a hyperparameter and it is chosen by trial and error method.

KNN is implemented using MapReduce. Before passing the data to the Hadoop environment, the dataset is normalized. As the values in the dataset range differently, it is better to convert the values to a common scale without distorting the differences in the range of values. To normalize the data, values from each feature are taken and each value is subtracted from a minimum value of that feature and then divided by the range of the values of that corresponding feature. At the end of this step, all the values in each feature would range from 0-1. Given train and test data is passed to normalize function in the script and with the normalized data, new CSV file is created. The normalized train data is given as input to the mapper and normalized test data is read in mapper. Since train data is huge, it is wiser to pass it to the mapper. K value is initialized in the reducer.

**Functionality executed by Mapper:**

* Each row from train data is passed one at a time to the mapper.
* The input line is then split by comma and an array is created.
* The last element in the array is a label so it is stored as y and the rest of the elements form train data.
* In mapper, the test data is read and the distance between each test and train data is calculated by using np.linalg.norm.
* Finally, test data, distance, y is sent as output to the reducer.
* Here, test data is key and (distance, y) is the value.
* The test data is converted into string delimited by comma which speeds up the execution. To differentiate key and value, the tab is given in between them. By default, the first tab-spaced element is taken as key and rest is taken as value.

**Functionality executed by Reducer:**

* In reducer, the output from the mapper is read.
* Each line is read one at a time and the line is split by tab.
* The first element obtained is key which is the test data.
* Then the test data is split by comma and then it is converted into a tuple.
  + The tuple is then used as a key in a dictionary and the value for the dictionary is in the form of a list of lists.
  + A tuple is used as a key here because an array cannot be a key in a dictionary as it is mutable.
* For each row of test data, the distance between it and each training data is computed and the label of training data and the distance is passed to the reducer.
* In the dictionary, for each test data, the list of distance and label pairs is stored.
* Then for each test row, the list is sorted based on distances.
* For each test data row, top K labels are picked and stored in a list.
* The rest is then checked to pick the most occurred label.
* Finally, the test row and its predicted label are sent as output from the reducer.

Files Used:

* script.py - to normalize the dataset. This reads the dataset and produces two new normalized CSV files norm-train.csv and norm-test.csv.
* Train-data.csv
* Test-data.csv
* Mapper.py
* Reducer.py

These files should be kept inside ‘home/cse587/knn/’. The newly generated files are also present in this location.

Script to run KNN:

* Python3 /home/cse587/knn/script.py
* hdfs dfs -copyFromLocal **/home/cse587/knn/norm-test.csv** **/home/knn/input/**
* hadoop jar /home/cse587/hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar **-file /home/cse587/knn/norm-test.csv** -file /home/cse587/knn/mapper.py -mapper /home/cse587/knn/mapper.py -file /home/cse587/knn/reducer.py -reducer /home/cse587/knn/reducer.py **-input /home/proj2/knn/input** -output /home/proj2/output/out