CSE 487/587 Assignment 2: Big Data Processing with Hadoop

Christopher Sam Roy
(50320374)
Department of Computer Science
State University of New York at Buffalo
Buffalo, NY 14260
croy2@buffalo.edu

Keshav Jethaliya
(50317073)
Department of Computer Science
State University of New York at Buffal
Buffalo, NY 14260
keshavje@buffalo.edu

Vishal Singh
(50317996)
Department of Computer Science
State University of New York at Buffalo
Buffalo, NY 14260
vsingh27@buffalo.edu

PART 1 - Setup and Word Count

Mapper

- For every line of input, first we trim the white spaces at the start and end of the sentence.
- Then remove all special characters, such that we are only left with either alphabets (Upper and lower case) or numbers.
- Split the document by space, so that we have a list of words.
- And finally print all the words with a count of 1.

Reducer

- Reducer will maintain a dictionary, with the word as key and its count as the value.
- For each line from input, which is a pair of words and count 1, reducer will create a new key for the dictionary.
- If the key is already present, it will add the count to its value.
- Finally, we print all the key-value pairs of the dictionary.

PART 2 - N-grams

Mapper

- For every line of input, first we process the document by trimming the white spaces at the start and end of the sentence, and converting everything to lowercase.
- Then remove all special characters, such that we are only left with either alphabets or numbers.
- Split the document by space, so that we have a list of words.
- Then we check for each word and choose the word if it is one of our keywords.
- We create and print the tri-grams of the words, and give a count 1 to each tri-gram.

Reducer

- Reducer will maintain a dictionary, with the tri-gram as key and its count as the value.
- It will count all the tri-grams, similar to the first part.
- But before printing, we sort the dictionary by its values in the decreasing order.
- Finally, we print the 10 most occurred modified tri-grams.

PART 3 – Inverted Index

Mapper

- For every line of input, first we trim the white spaces at the start and end of the sentence.
- Then remove all special characters, such that we are only left with either alphabets or numbers.
- Split the sentence by space, so that we have a list of words.
- Then we get the path of the file, the word is coming from, using os.environ[“map_input_file”].

- Extract the file name from the whole path.
- And print the words with their filenames.

Reducer

- Reducer maintains a dictionary, with key as the word and a list of filenames as its value.
- For each line of input, which is a word-filename pair, it creates a dictionary entry for the word as the key and a list containing filename as its value.
- If the key is already present in the dictionary, and the filename is not there in the value for that key, it will append the new file name in the list of filenames already present.
- Finally, it prints each word in the dictionary with a respective list of filenames.

PART 4 – Relational Join

Pre-processing: We have converted join1.xlsx and join2.xlsx to join1.txt and join2.txt respectively and then saved this file to hdfs. Now these two files contain tab separated column values and each row is end by “next-line” character (“\n”).

Mapper

- For every line of input, first we trim the white spaces at the start and end of the row.
- Split each row by tab (“\t”), so that we have a list of column values.
- As we have given two files, one containing two columns and other containing four columns, so we are distinguishing on the basis of split size and then inserting column respective value to global defined variable.
- At the end of the loop we will have either two column values if it reads from join1.txt or four column values if it reads from join2.txt. So, we will print all unique column values in a row with other empty column value as “-”.

Reducer

- Input to reducer is the output file from mapper which will have rows containing five column values with some value or with “-” value in it. And these rows are sorted by first column i.e. by employee_id.
- As all rows are sorted and our join column is employee_id, so all duplicated employee_id will be next to each other.
- Split each row by tab (“\t”), so that we have a list of column values.
- Save the previous column value and compare previous saved employee_id with current row employee_id.
- If we find two duplicate employee id then we will combine their value together and print in a row.

Bonus: K-Nearest Neighbor

Mapper:

- We have passed the test set using the “-file” option since it is mentioned as a small dataset. Hence it is accessed in the mapper using “numpy.genfromtxt”.
- Both the test and train set have been normalized (min-max) in the mapper before any calculations.
- Each row of the train set is accessed from the stdin and the Euclidean distance between the features of all the rows of the test set is calculated with respect to the accessed training row.
- The Euclidean distance between the training row and each row of test set is then printed out as an output of the mapper along with the label (target) of that training row.

- The output of mapper is a key-value pair separated with tab (“\t”) in which key is the test row index and value is the combination of target value in the training row and the Euclidean distance w.r.t the test row.

Reducer:

- The value of k in KNN is 5.
- The output from mapper is tab-separated, hence it is split in the reducer w.r.t tab. This gives the test row index along with the corresponding Euclidean distance with the training row followed by the target value.
- A dictionary is formed in the reducer that has the key as the test row index and value as the list of all the calculated distances between that particular test row and the entire training set.
- This dictionary allows the reducer to collect all the distances of a particular test row with all values of train set. Sorting this distance and then finding the k smallest distances and corresponding target values is the penultimate step of predicting the value of that test row.
- The final predicted value is calculated by finding the majority (most common) target value of the chosen k training rows.
- The output of reducer is the test row followed by the predicted target value separated with tab.

VIDEO LINK: (BOX)

https://buffalo.box.com/s/qlb921rbnnfqa2a60vep5meczccpn2ft

DRIVE:

https://drive.google.com/open?id=15dgxrCFjo1uQCFmI9B87s89bFML5_xFd