Project 2 Report

Team Members:
Yash Patange (yashsanj) #50319943
Kunal Reddy (kunalred) #50320285
Nihit Natu (nihitume) #50321035

Part 1: Word Count

The given task is to identify the number of times each word has occurred in the gutenberg dataset.

Mapper: The mapper traverses all the documents line by line given by standard input and splits them into individual words. We then print out the words and their count of occurences (which is 1) to standard output which serves as input for the reducer program.

Reducer: The reducer takes the mapper's output as input through standard input. As the mapper gives the words in **sorted** order, we take the first occurence of word and count the number of times the rest of the word occurrences happen until a new word is encountered. The word and the total count is printed and we move on to the next word and repeat the above steps.

Part 2: N-grams

The task was to find the top ten trigrams in the gutenberg dataset.

Mapper: The mapper traverses through every line in the input files and performs three preprocessing steps removing punctuations using re.sub method, converting into lower case and lemmatize all the words in the line. The entire word corpus is appended to a list and then we search for a for the given keywords in that list. If we find a given keyword, we try to generate 3 possible trigrams for each occurence of a keyword. The second mapper code for handling multiple reducer cases, just combines the output provided by the intermediate reducers prints it on the command line.

Reducer: The reducer takes the output of the mapper which contains a list of trigrams each having a count of 1 separated by a tab delimiter. We created a dictionary to store the count of the trigrams where the trigrams were the keys and the value was the count. The code traversed through each line and added a new key in the dictionary if a new trigram was found. If an existing trigram was found the current value in the dictionary was incremented by 1. After traversing through the output completely, the dictionary was sorted in descending order using operator module and then the top ten entries in the dictionary were printed to get the desired result. The second reducer code is similar to this but instead of count equal to one it would be equal to the trigram count given by the previous reducers and the sorted dictionary would be printed to get the final result.

Part 3: Inverted Index

Mapper: The Mapper program takes the gutenberg dataset as input through standard input. We are retrieving the filepath through os.environ and file name using ntpath library. We assigned ids to the filenames. arthur.txt as 1, james.txt as 2,leonardo.txt as 3 respectively.
We are also tokenizing the documents into words and doc-ids. This is printed as standard output and is given as input to the reducer program.

Reducer: In the reducer program, we created a dictionary which has a list of values. In this case, keys are the words and doc-ids are lists of values. When a new word is encountered, we check if it's already there in the dictionary. If yes, we just append the doc-id to the value. Else, we put the key value pair into the dictionary. At the end, we are printing the contents of the dictionary.

Part 4: Relational Join

The given task was to perform a join on 2 tables using a common key(Employee ID) similar to SQL joins. The approach used was a reduce-side join.

Mapper: In the mapper program, both of the tables were read from the standard input streams, line by line.The files provided as input are text files. The converter we used a comma as a delimiter too, which we realized and we concatenated the values that had a comma in it. No matter whichever file is first read the mapper will handle the cases well and just emit the files in a proper order at the end of the mapper phase.

Reducer: In the reducer phase, the program will again read the tables from the standard stream. The program then distinguishes between the rows of the table by checking how many columns the line has. Once this is done, the reducer simply combines the 2 tables based on the key(Employee ID) and emits the 2 new joined tables.

Part 5: KNN

The given task was to perform K-Nearest Neighbours classification task on the given training and test data.

Mapper: In the mapper program, the test data is taken directly from the Hadoop DFS and Train data is read line by line from sys.stdin. We again convert this Train data to a Dataframe. We normalize the data for further use. The mapper program then calculates the K Nearest neighbours for each data point in the Test data. Once we get the K data points from the Train set we append these K labels to a list and print this list. So the output of the mapper is a list of K labels for each data point in the test data

Reducer: The reducer program reads all the lists generated by the mapper code line by line using sys.stdin and stores each of this list into another list. We then traverse each list to calculate the predicted label for a data point. The prediction is done using a Voting method

that calculates the number of votes for each label in the K labels generated. The label with the highest votes is the predicted label. The output of the reducer is the Data Entry and the predicted label for all the rows in the Test set.