CSE587 Assignment 2

Rishabh Joshi (rjoshi4@buffalo.edu, 50316839)

Shivaprakash Rajshekar Hiremath (hiremath@buffalo.edu, 50313811)

Deepanshu Yadav (dyadav@buffalo.edu, 50321285)

Generic Instructions:

➢ Video:

0:00:00 **intro**

0:01:09 **task1**

0:03:44 **task2 mapper1**

0:08:10 **task2 mapper2**

0:10:00 **task3**

0:12:28 **task4**

0:14:12 **knn**

➢ Other:

The comments have been written in the code for explaining step by step

Kindly install below modules:

pip3 install nltk

command: Enter **python3** and press Enter

In python interpreter type below commands and run to download and install:

import nltk

nltk.download('stopwords')

nltk.download('punkt')

nltk.download('wordnet')

Note while running if it asks for any explicit packages other than the mentioned kindly install and re run by renaming or deleting the contents of output folder. (We have tried to remember all the packages installed in case something is missed out kindly install them.)

For Task 5:

pip3 install pandas

This is required to read test.csv

We had run all the tasks with 5 mappers and 4 reducers to ensure our code works fine with multiple mappers and reducers. (In the video)

Commands used in the video are present in commands.docx.

Video Link: https://buffalo.box.com/s/l2pj4bk2gvkddu6aje8yopu58yu8bsn9

The results of executed tests in the video are present in the folder(results.zip).

Task1:

Note to grader:

- We haven't removed specific special characters that are not identified by the libraries.
- We have kept the numerals and not removed them in our processing.

Code Explanation:

mapper_task1.py

1) We read the data from the 3 input files process each of them.
   - In Processing we are performing below steps:
   - Complete line is converted to lower case.
   - We are tokenizing each line by word_tokenize of nltk library.
   - We then remove stop words if any.
   - We then check if any punctuations exist in words.
   - We only take the root word.
   - We finally check if it is not an empty string after the above processing. If not we add it to the list of processed words.
2) We then print the words and count as 1 with tab separator.

reducer_task1.py

1) We separate each input by tab separator and group the data based on the key i.e., word
2) We then count the total count of same words and print the results.

Task2:

Note to grader:

- We haven't removed specific special characters that are not identified by the libraries.
- We have kept the numerals and not removed them in our processing.
- **We haven't remove stop words here.**

Code Explanation:

mapper_task2.py

1) We read the data from the 3 input files process each of them.
   - In Processing we are performing below steps:
   - Complete line is converted to lower case.
   - We are tokenizing each line by word_tokenize of nltk library.
   - We then check if any punctuations exist in words.
   - We finally check if it is not an empty string after the above processing. If not we add it to the list of processed words.
   - From these processed words we check and replace the keywords with $
   - We create trigrams using nltk library.
2) We then print the processed trigrams and count as 1 with tab separator.

reducer_task2.py

1. We separate each input by tab separator and group the data based on the key i.e., word
2. We then count the total count of trigrams of same kind.
3. We sort the trigrams based on values and print the top 10.

mapper_task2.py

1. We separate each input by tab separator.
2. We add a dummy key called **'key'** and print trigram and count so that all the same key values go to a single reducer.

reducer1_task2.py

1. We separate each input by tab separator and group the data based on the key i.e., dummy key word
2. We then count the total count of trigrams of same kind.
3. We sort the trigrams based on values and print the top global 10.

4. If no values are received by the reducer then that receiver will not print anything.
   So, in this case only one reducer will provide the results and remaining reducers will not print anything due to a dummy key used.

Task3:

Note to grader:

- We haven't removed specific special characters that are not identified by the libraries.
- We have kept the numerals and not removed them in our processing.

Code Explanation:

mapper_task3.py

1) We read the data from the 3 input files process each of them.
   - In Processing we are performing below steps:
   - Complete line is converted to lower case.
   - We are tokenizing each line by word_tokenize of nltk library.
   - We then remove stop words if any.
   - We then check if any punctuations exist in words.
   - We only take the root word
   - We finally check if it is not an empty string after the above processing. If not we add it to the list of processed words.
2) We extract the file name by import os and keep only filename using ntpath.basename function.
3) We then print the words and their filename with tab separator.

reducer_task3.py

1) We separate each input by tab separator and group the data based on the key i.e., word
2) We then append the filenames in the list and perform set operation on the list and print the results.

Task4:

Note to grader:

- xlsx files have been converted to csv to pass as input to the mapper.

Code Explanation:

mapper_task4.py

1. We read the data from the 2 input files (Join1.csv and Join2.csv)
2. The data is split using ',' as separator
3. In Join2 table, the salary has comma in between the value and also, some countries have comma between the names. These have been taken care of by rejoining the respective values after splitting as mentioned in code comments
4. Output of mapper is all values of both the tables with -1 used for values which are missing in either table

reducer_task4.py

1. We separate each input by '~' separator and put the data in two dictionaries based on which table it is coming from.
2. All the keys from one dictionary are picked and respective values of employeeName, salary, Country and PassCode are assigned from respective dictionaries and printed to output file.
3. If any key was present in other dictionary and not in the one which was iterated upon, it is also checked using another for loop and values corresponding to it are also printed.

Task5: KNN

Note to grader:

- We have normalized the test and train data before giving it as input to MapReduce
- In the output file, first column is Row Number of Test Data and second column is Predicted Label
- We have initialized k=3 in the mapper_knn.py and reducer_knn.py

Code Explanation: (Working with multiple reducers also)

Normalization

1) The train and test data are combined in one file with all 47 attributes and are normalized using sklearns min-max scaler for better accuracy. Then, the train and test data are separated into 2 files TrainNN.csv and TestNN.csv. It can be found in the P5Preprocessing.ipynb file.

mapper_knn.py

1) We read the data from the train input file as stdin and test data is loaded in each mapper using pandas read_csv().
2) We create a dictionary in which key is the test row index and values are the list of pairs of distance of the test data with train data.
3) We read the train data line by line and since our complete test data is loaded in memory we calculate for every test row the square of the distance of the test data row with that train data row and append the pair which is the distance and the label of the train row with which the distance is being calculated to the list corresponding to the test row.
4) Once the test dictionary is populated, we iterate it and sort every list corresponding to the test row index based on the distance
5) We print the first K elements with key as the row index and two values separated with tab which include the distance and the label.
6) Having key as test row number ensures us that for every mapper in which test data is loaded will send all the distances to the same reducer, hence it will work on multiple reducers.

reducer_knn.py

1) We separate each input by tab separator and group the data based on the key i.e., the test row index
2) In every group which has (number of mappers*k) values we select the top K smallest distance values and then take the label which occurs with majority in those K elements and then print the test row index and it's predicted label.