Assignment 2

Pooja Sharma (poojaomp@buffalo.edu)
Aditya Bansal (adityaba@buffalo.edu)
Deepesh Yadav (deepeshv@buffalo.edu)

The link to video for all parts is:
Assignment2_video.mp4

Part 1:

The logic for Mapper:

- We have stripped white spaces in the file for each line.
- Then, we have split the line by whitespace characters into words.
- We then use regex to remove the punctuation marks
- The mapper emits{prints} “word 1” for all the words it sees.
- Before emitting, we check if after removing the punctuation characters the length of the word is at least 1. For example if “.” was the only character left then removing the punctuation mark will make it an empty string.

The logic for Reducer:

- The reducer splits the input it receives based on the “\t” character into word, count.
- The reducer maintains a dictionary to track all the words it sees {word : count }.
- It checks if the current word is already a key of the dictionary (i.e The reducer has already seen that word).
- If it already exists in the dictionary, then it increases the value of the key (count) by the count value it receives.
- Otherwise, it adds the entry to the dictionary and initializes the count to 1.

Part 1 Output:

```
waterholes      1
waterjar        1
waterjugjar     1
watermeasurers  1
watermelon      1
wateroceans     1
waterouzel      1
waterpower      1
waterrings      1
waters  192
watersheds      1
watershrew      1
waterspider     2
waterspray      1
watervapour     3
waterway        2
waterweed       1
watery  8
wattles 4
wavelengths     6
wavemotions     2
waves   176
wavesLightWhat  1
waving  8
```

Part 2: N-grams

The logic for Mapper:

- We have stripped white spaces in the file for each line.
- Then, we have applied regex to remove all the special characters and numbers from the file. We have also replaced all the uppercase letters to lowercase letters.
- Then, we are splitting the line into words and storing them in words[] array.
- We are iterating over each word in the array as:
    - We do lemmatization of each word so that we don't miss on similar words.
    - We then check if the word is equal to the keywords we are searching, i.e. science, sea, fire.
    - If the word is equal, we are checking if this word has 2 words before it. If yes, we add the previous two words separated by '_' and add a $ at the end. Then we print this word in the file with count 1.
    - Similarly, we are checking if this word has 1 word before and 1 word after it. If yes, we add the previous word, add a $ and then add the next word at the end. We print this word in the file with count 1.
    - Then, we are checking if this word has 2 words after it. If yes, we add a $ and then add the next two words at the end. We print this word in the file with count 1.
- This way, we get all the trigrams with count 1 in the output file.

The logic for Reducer:

- We have initialized a 2D array to store the trigrams and their final count.
- We take the output from the mapper and strip the whitespaces in line.
- We then split each line on '\t' and store the word and count.
- Since reducer receives the words sorted by the key, we check for each line, if the new word is equal to the previous/current word.
- If yes, we are adding the count of this word to the previous count.
- If no, it means that there is a new trigram, and thus we append the previous word and its respective count to the 2D array and then, initialize the current word and count with new values.
- Then, we are sorting the 2D array in descending order of the count using the 'sorted' function.
- Then, we scan the array and print the top 10 words with their count in the output file.

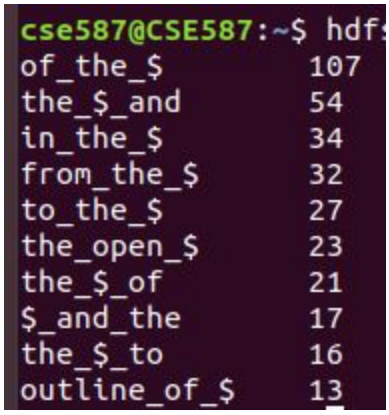
Logic for Mapper2:

- We are taking the output from each reducer, which will be top 10 words with their count.
- We then strip each line and simply print the value and their count which will be an input to the reducer 2.

Logic for Reducer2:

- Same as Reducer 1.

Part 2 Output:



Part 3:

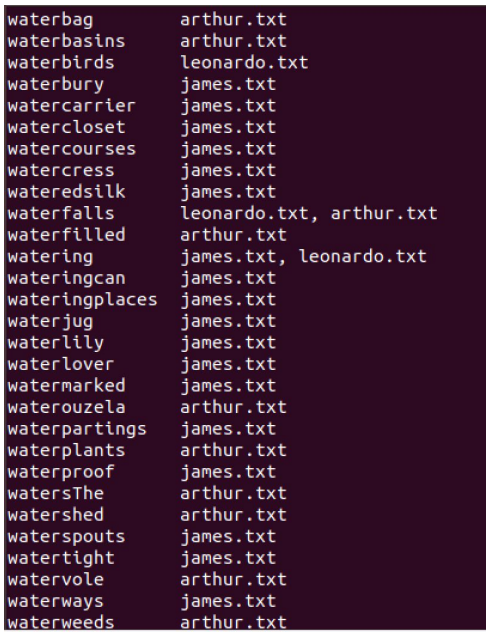
The logic for Mapper:

- We have stripped white spaces in the file for each line.
- Then, we have split the line by whitespace characters into words.
- We then use regex to remove the punctuation marks
- We use the os library to get the file path and extract the filename from it.
- The mapper emits{prints} “word filename” for all the words it sees.
- Before emitting, we check if after removing the punctuation characters the length of the word is at least 1. For example if “.” was the only character left then removing the punctuation mark will make it an empty string.

The logic for Reducer:

- The reducer splits the input it receives based on the “\t” character into word, filename.
- The reducer maintains a dictionary to track all the words it sees
  {word : file names separated by comma }.
- It checks if the current word is already a key of the dictionary (i.e The reducer has already seen that word).
- If it already exists in the dictionary, then it checks if the filename is already the part of the string representing the list of filenames. It only appends the filename it it isn’t already in the string.
- If the word is not in the dictionary, it adds the entry into the dictionary and creates a string containing the filename as the value associated with the key “word”.

Part 3 Output:



```
waterbag        arthur.txt
waterbasins     arthur.txt
waterbirds      leonardo.txt
waterbury       james.txt
watercarrier    james.txt
watercloset     james.txt
watercourses    james.txt
watercress      james.txt
wateredsilk     james.txt
waterfalls      leonardo.txt, arthur.txt
waterfilled     arthur.txt
watering        james.txt, leonardo.txt
wateringcan     james.txt
wateringplaces  james.txt
waterjug        james.txt
waterlily       james.txt
waterlover      james.txt
watermarked     james.txt
waterouzela     arthur.txt
waterpartings   james.txt
waterplants     arthur.txt
waterproof      james.txt
watersThe       arthur.txt
watershed       arthur.txt
waterspouts     james.txt
watertight      james.txt
watervole       arthur.txt
waterways       james.txt
waterweeds      arthur.txt
```

Part 4:

The logic of Mapper:

- Given employeeId,Salary,Name,Country,Passcode as -1 values to signify if any value isn't given yet as no column can have negative value
- We have stripped white space in the file for each line.
- After that we are splitting our columns using "," .
- No we are taking if-else loop to differentiate tables in "Join2.csv" there are 4 columns and in "Join1.csv" there are 2 columns or 5 columns in case line break by salary also or 6 words if line break by salary and country both.
- So if length of words is 4 then we will save values for columns of "Join2.csv" and value of "Name" will remain -1 and if there are two words then we will save for "Join1.csv" and values of "Name", "Country" and "Passcode" will remain -1.
- Now we are printing each row with all 5 columns and giving it as input to our reducer.

The logic of Reducer:

- In reducer first we are initializing variables to store the values for the previous row since we have to join two rows so we need values for both of them.
  foundKey - This will save the key that contains the value of the previous row.
  isFirst -It will check if the row is first or not.
  isCurrentSalaryMapping - This is boolean operator that will tell if line is from which table.If true then line if from "Join2.csv"
  currentPasscode=This will save the current value of Passcode.
  currentSalary -This will save current value of salary.

currentCountry-This will save the current value of the country.
currentEmployeeID-This will contain the current employee id.

- Now for each line of input first we are stripping whitespace from both sides of the line.
- Now we will put an if else loop to identify which row is from which table.
- If the name is equal to -1 it means the row is from the second table and we will save all it's column values in our current variables. And mark isCurrentSalaryMapping as true.
- Else row is from the first table and we will mark isCurrentSalaryMapping as false.
- Now if isCurrentSalaryMapping is false then we will combine the rows because the row of table1 having the same EmployeeId is coming after the row of Table1.
- Now we have initialized the current key in which we are saving all column values from current variables and the name.
- Now we are going to print our current key as string and save it in our output file.

Part 4 Output:

```
16001018 -0115,Sean Herrera,"$28,689",Guatemala,JFH58LTF7LS
16020401 -5051,Kaitlin Hubbard,"$33,868",Kazakhstan,FAK20ZLP2XX
16030503 -6774,William Maldonado,"$92,019",Samoa,SBN74FYH6JJ
16030612 -9305,Brennan Boyd,"$65,670",Haiti,DRK47IOB8ZU
16031211 -8540,Xanthus Roman,"$72,771",Comoros,PAT27JJA6XB
16040110 -9038,Brielle Weiss,"$19,785",Bolivia,JEQ68XYP3MN
16040828 -9221,Dalton Mccall,"$89,983","Virgin Islands, United States",KXY23CHX2OC
16050728 -1673,Lunea Clarke,"$72,063",Chad,RLJ79WSK7XO
16051003 -3665,Colette Kirby,"$41,630",Dominican Republic,WRN14VLN3VD
16060113 -5817,Ferdinand Stewart,"$92,932",Monaco,WOV44LRT2ZD
16060415 -7529,Fredericka Davidson,"$71,823",Rwanda,AMK92WUZ6MP
16070128 -6445,Cleo Alvarez,"$42,257",Panama,QPY55RSZ3WO
16070214 -4304,Deanna Cardenas,"$29,284",Saint Barthélemy,LLX37SJF7HT
16080512 -1522,Nehru Rivers,"$17,611",Bhutan,MMY82OHH0QW
16100501 -2636,Basia Ballard,"$01,702",Mexico,ITO91QFT9AO
16101124 -9891,Kaseem Dickerson,"$75,105",Turks and Caicos Islands,TWI40PVM4LC
16120428 -6379,Donna Haney,"$82,026",Guinea-Bissau,IVR42SIN5ZY
16140530 -1837,Clark Wall,"$68,573",Montenegro,XGS23DFI6BM
16160230 -4113,Cecilia Cox,"$26,126","Virgin Islands, United States",CYF43NLU7YB
16180923 -8320,Chantale Morrow,"$77,352",Czech Republic,IAY65QLB0DW
```

Column1-EmployeeID,Column2-Name,Column3-Salary,Column4-Country,Column5-Passcode

Part5:

The logic of Mapper:

- In Mapper first we are loading test data using pandas as the test is going to be with every mapper.
- Now we are reading train data as input since it will be distributed in chunks in every mapper.
- Now for each line of input we are stripping it of whitespace and splitting it by ",".
- After this we are splitting the features and labels from the train row and appending them to list_features and list_labels.
- Now we are converting our list_features and list_labels to a numpy array with datatype as 'float64'.

- After this we are normalizing our list_features(train features) and test_sample using sklearn min-max scaler to improve the accuracy of our results.
- Now for each row in features_test (test row) we are finding it's euclidian distance from all rows of train data.
- distance=np.sqrt(np.sum(np.square((np.subtract(np.repeat(row,np.size(list_features,0),0),features_train))),axis=1)) -This is the formula we are using to find euclidean distance.
- Now we are concatenating test row no, euclidean distance,label of train row into a numpy matrix.
- Now we are printing each row of this matrix separately and giving it as input for our reducer.

The logic of Reducer:

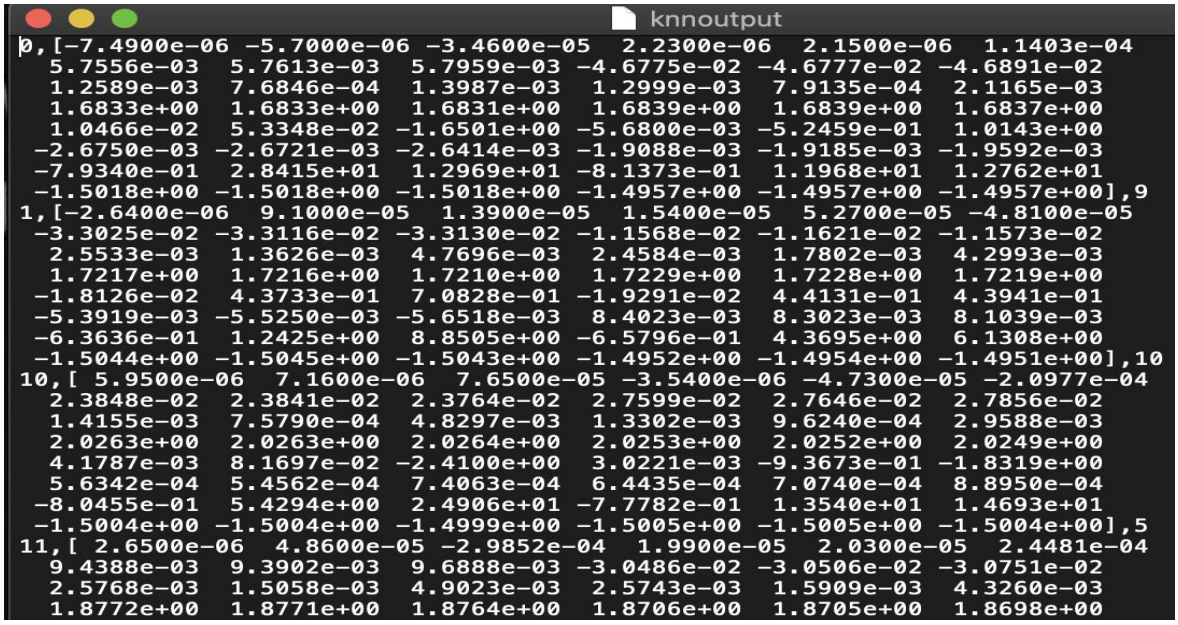
- In our reducer we are again importing our test data so as to print each row with it's label.
- We have given k-value=50 which means we are going to consider first 50 rows whose distance is least for each test row.
- Now we will read each line of our input from mapper and strip it from white space and split it using "," and after that we are appending it to a list
- After reading all rows and appending them we will transform our list of inputs to numpy matrix with datatype of 'float64'.
- *matrix=np.array((np.split(distance_sort, np.where(np.diff(distance_sort[:,0]))[0]+1)))*
- Distance_sort[:,0]-first we are fetching column values of 1st column.

  Numpy.diff- now to check where the value in the column actually changes.

  np.diff(arr[:,1]) -Any thing non-zero means that the item next to it was different, we can use *numpy.where* to find the indices of non-zero items and then add 1 to it because the actual index of such item is one more than the returned index

  - using this formula we are splitting our matrix into matrices with different test row no.
- Now for each matrix which is for one test row containing it's row number it'd distance from each train row in second column and label of train row in third column we are going to find the result.
- First we will sort this matrix using the second column which is it's distance from each training row.
- Now we will extract the first Knn-value rows which we will consider to find label for that particular row.
- Now from the third row we will find the label with maximum frequency.
- Using this we have got our result now we are printing test row number, test row and the label which we just found and giving it as our output.

Part 5 Output:



Column1-Test Row Index ,Column2-Test Row, Column3-Label which we found using knn