Lab Course: Distributed Data Analytics
Exercise Sheet 2
Group 2 - Monday

Submitted by: Sruthy Annie Santhosh, 312213

Topic:Complex Data Lab: Processing Text Data in a Distributed Setting

- My System

  Hardware and Software SetUp:

Processor	M1 chip
Operating System	MacOS Monterey
Number of Cores	8
RAM	8 GB
Python version	3.8.8

  The code editor used in this sheet : VSCode
  The output images attached are screenshots from the terminal.

In this exercise, due to the huge amount of dataset, the program was not running for when the workers were less than 8. Buffer limit error was being shown. Hence I have used 8, 12 and 16 workers in this program.

- Exercise 1: Data Cleaning and Text Tokenization

  First the dataset is read. The given dataset contains multiple folders which have multiple files. Hence first the file paths were generated and then the files were read and stored in an array. Each element of the array corresponded to a document.

The data cleaning and tokenization were done using point to point communication in MPI. I have used send() and recv() functions.

MPI framework:
The master worker splits the file data into different parts depending on the number of workers selected. Here due to splitting, different workers will get a subset of documents. The other workers will do the cleaning and tokenization process on their set of documents and then send it back to the Master. The master again combines the tokens produced to get an array of lists containing tokens of each document. The time taken for this operation is also calculated.

Cleaning and Tokenization:
Here I used the NLTK library to get the stopwords and to tokenize the data.
First the document data was tokenized, converted to lowercase, then only the tokens having alphabets were selected and then compared to the stop words. Thus the stop words are removed and tokens are found. This is done for each document and appended.

```
#Function to clean and tokenize data
def data_clean(data):
    print("cleaning data")
    text_data = []
    doc_data = []
    #Getting the list of stop words from nltk
    stopwords_ = set(stopwords.words('english'))

    #For iterating over each document
    for i,doc in enumerate(data):
        #Getting each word in a doc
        text = word_tokenize(doc)
        #Converting to lower case
        text2 = [t.lower() for t in text]
        #Getting only the words ( removing all other characters)
        text3= [word for word in text2 if word.isalpha()]
        #Removing stop words
        text4 = [word for word in text3 if word not in stopwords_]

        text_data.append(text4)

    return text_data
```

Master:

```
#Starting time for the Clean and Tokenization process
start_clean = MPI.Wtime()
#for loop to send data splits to other workers and to recieve the tokenized data back
for i in range(size-1):
    comm.send(data_split[i-1],dest = i+1, tag=1)
    #Recieving tokens for the documents send
    partial_token = comm.recv(source = MPI.ANY_SOURCE, tag =2)
    token.extend(partial_token) #appending the tokens
#End time for clean and tokenization process
end_clean = MPI.Wtime()
time_clean = end_clean - start_clean
print("\n Total time taken for clean and tokenization- ",round(time_clean,4))
#print("\n Number of documents- ",len(token))
```

Other workers:

```
#Performing the operations for Cleaning and Tokenization
data = comm.recv(source=0, tag=1) #Data is recieved
print("\nFile  has reached rank ",rank)
#The required function is called and then result is send back to master
comm.send(data_clean(data), dest = 0, tag =2)
```

Output: Only a small part is shown here as the data is huge

```
 Tokenised data-  [['rochester', 'udel', 'gatech', 'rpi', 'rutgers', 'chrstie',
d', 'david', 'harwood', 'writes', 'b', 'recent', 'criticism', 'listserv', 'b',
resented', 'essene', 'nt', 'eating', 'choices', 'info', 'came', 'biased', 'oppo
 'aatchoo', 'philosophy', 'principles', 'b', 'university', 'manitoba', 'science
eful'], ['rutgers', 'hudson', 'paul', 'hudson', 'jr', 'sabbath', 'admissions',
'sabbath', 'part', 'b', 'ceremonial', 'laws', 'post', 'text', 'reply', 'investi
mption', 'sabbath', 'b', 'ceremonial', 'law', 'see', 'refer', 'writes', 'collos
```

Time Taken:

Number of Workers	Time taken (s)
8	84.596
12	106.967
16	152.886

- Exercise 2: Calculate Term Frequency

Here we have the tokenized form of the dataset. The term frequency in each document is to be found: t- token and d- document , nd(t) - number of times a token appears in a doc

$$\mathrm{TF}(t,d) = \frac{n^d(t)}{\sum_{t' \in d} n^d(t')} \ ,$$

This is done using point to point communication in MPI. I have used send() and recv() functions.

MPI framework:
The master worker splits the tokenized data into different parts depending on the number of workers selected. Here due to splitting, different workers will get a subset of documents(sub arrays in the list). The other workers will do calculation of term frequency on their set of documents and then send it back to the Master. The master again combines the TF values for each document to get an array of lists containing TF of each token of each document. The time taken for this operation is also calculated.

Term Frequency Calculation:
First we find the count of each token in a document using Counter. This is stored as a dictionary. Then for each document, the total number of tokens are calculated. Then the tf of each token in doc is found by its count and dividing the total count.

```
#Function to calculate the Term Frequency
def TF(data):
    #Finding the count of each token in a document using Counter and python lists
    tf_token = [dict(Counter(token)) for token in data]

    #TF is found within each document
    for doc in tf_token:
        #Total number of tokens in the doc
        token_len = len(doc)
        for token in doc:
# TF = number of times token appears in the doc / total no of tokens in doc
            doc[token] = doc[token] / token_len

    return tf_token
```

Master:

```
#Splitting the tokens to send to different workers
token_split = np.array_split(token,size-1)

#Starting time for TF calculation
start_tf = MPI.Wtime()
#For loop to send the tokens and to recieve the TF values from other workers
for i in range(size-1):
    comm.send(token_split[i-1],dest = i+1, tag =3)
    tf_partial = comm.recv(source= MPI.ANY_SOURCE, tag=4)
    tf_list.extend(tf_partial) #appending the values to get all the documents


print("TF list-\n",tf_count)
#End time for TF process
end_tf = MPI.Wtime()
```

Other workers:

```
#Data recieved for TF calculation
token_part = comm.recv(source=0, tag =3)
#The required function is called and result is send back to master
comm.send(TF(token_part), dest = 0, tag = 4)
```

Output:

```
023, 'green': 0.010752688172043012, 'two': 0.053763440860215055, 'constit
 0.010752688172043012, 'way': 0.010752688172043012, 'telling': 0.01075268
688172043012, 'read': 0.010752688172043012, 'volts': 0.021505376344086023
0.010752688172043012, 'reversed': 0.010752688172043012, 'bad': 0.01075268
43012, 'elsewhere': 0.010752688172043012, 'drops': 0.010752688172043012,
 'develop': 0.010752688172043012, 'ac': 0.010752688172043012, 'hz': 0.010
172043012, 'bell': 0.010752688172043012, 'ringer': 0.010752688172043012,
': 0.01282051282051282, 'radio': 0.02564102564102564, 'freq': 0.012820512
4, 'user': 0.01282051282051282, 'thu': 0.01282051282051282, 'apr': 0.0128
, 'wondering': 0.01282051282051282, 'possible': 0.01282051282051282, 'use
 0.01282051282051282, 'able': 0.01282051282051282, 'signal': 0.0256410256
```

Time Taken:

Number of Workers	Time taken (s)
8	1.7334

12	2.382
16	3.181

- Exercise 3: Calculate Inverse Document Frequency

  Here we have the tokenized form of the dataset. The Inverse document frequency in the corpus is to be found: t- token and d- document , Id(t) - number of documents a token appears in the corpus, C - total number of documents in the corpus.

  $$\mathrm{IDF}(t) = \log \frac{|C|}{\sum_{d \in C} \mathbb{1}(t, d)} ,$$

  This is done using point to point communication in MPI. I have used send() and recv() functions.

  MPI framework:

  The master worker splits the tokenized data into different parts depending on the number of workers selected. Here due to splitting, different workers will get a subset of documents(sub arrays of list). Here we send two params: the token split, the document count of tokens. The other workers will do the calculation of IDF on their set of tokens and then send it back to the Master. The master again combines the IDF values for each token to get a dictionary for each token in the corpus The time taken for this operation is also calculated.

  Inverse Document Frequency Calculation:

  First we find the count of each token in a document using Counter. This is stored as a dictionary. Then we find the number of documents each token appears in. Then the idf of each token is found by taking the log of the total number of documents divided by the count.

```
#Function to count the occurance tokens in the whole corpus
#This data is needed for calculating IDF
def token_count(data):
    #Counter to calculate the count of each token in each doc
    tok_freq = {} #dictionary
    freq_doc = [dict(Counter(token)) for token in data]

    for count_token in freq_doc:#for each doc
        for token in count_token: # for each token
            if token in tok_freq.keys(): # checking if token already exists
                tok_freq[token] += 1 #increment count
            else:
                tok_freq[token] = 1 #add new token
    return tok_freq

#Function to calculate IDF
def idf(tok_list,token_count):
    idf_corpus = {} #dictionary
    C = 19998 #Total number of documents
    #for each token in the given data

    for token in tok_list:
        #idf = total number of docs / count of the documents having the token
        idf_corpus[token] = math.log(C/token_count[token])

    return idf_corpus
```

Master worker:

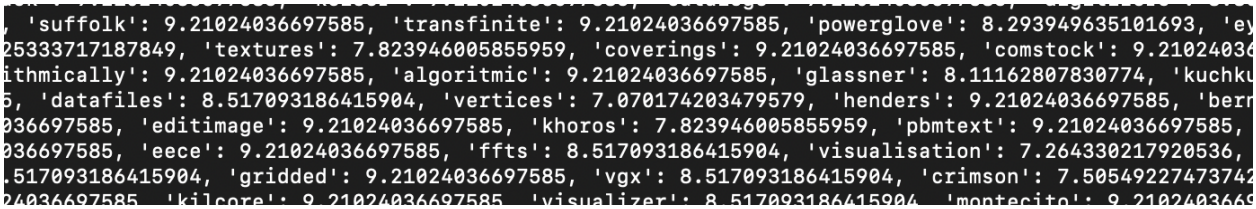
```
    #Start Time for IDF calculation
    start_idf = MPI.Wtime()
    #For loop for sending and recieving tokens and idf values
    for i in range(size-1):
        #sending the token split and the frequency of each token
        data = [idf_split[i-1],token_freq_doc]
        comm.send(data,dest = i+1, tag = 5)
        idf_val = comm.recv(source = MPI.ANY_SOURCE, tag = 6)
        idf_dict.update(idf_val) #updating in dictionary the idf value of each token

    #End time for IDF process
    end_idf = MPI.Wtime()
```

Other workers:

```
#Data recieved for IDF calcualation
frq_part, frq_list = comm.recv(source=0, tag = 5)
#The required function is called and result is send back to master
comm.send(idf(frq_part,frq_list), dest = 0, tag = 6)
```

Output: only a part of output is shown here



Time Taken:

Number of Workers	Time taken (s)
8	0.899
12	1.829
16	3.489

- Exercise 4: <u>Calculate Term Frequency Inverse Document Frequency scores.</u>

  Now we have the tf values for each token in each document and also the idf values for each token. The TF-IDF values for each token in each document in the corpus is to be found: t- token and d- document ,

  $$TF\text{-}IDF(t,d) = TF(t,d) \times IDF(t) \ ,$$

  This is done using point to point communication in MPI. I have used send() and recv() functions.

MPI framework:
The master worker splits the tf list into different parts depending on the number of workers selected. Here due to splitting, different workers will get a subset of documents. Here we send two params: the tf split, the idf values in dictionary. The other workers will do calculation of TF-IDF on their set of documents and then send it back to the Master. The master again combines the TF-IDF values for each split to get a list of tf-idf values for tokens in the documents in the corpus The time taken for this operation is also calculated.

Term frequency Inverse Document Frequency Calculation:
We iterate through each document in the data received and then we iterate through each token in the document. Then we multiply the tf value and idf value of the token to get the tf-idf value. This is done for all tokens in all documents.

```
#Function to calculate TF-IDF
def tf_idf(idf_dict,tf_list):
    print("TFIDF")
    #Iterating through each document in data
    for file in tf_list:
        #Iterating through each token
        for word in file:
            #TFIDF(token, doc) = tfvalue(token in the doc)* idfvalue(token)
            file[word] = file[word] * idf_dict[word]

    tf_idf = tf_list
    return tf_idf
```

Master Worker:

```
  #Start time for TF_IDF process
  start_tfidf = MPI.Wtime()
  #For loop to send the data to other workers for getting TF-IDF values
  for i in range(size-1):
      #sending the termfrequency list split and the idf dictionary
      data = [tf_split[i-1],idf_dict]
      comm.send(data,dest = i+1, tag = 8)
      tf_idf_part = comm.recv(source= MPI.ANY_SOURCE, tag = 10)
      tf_idf_list.extend(tf_idf_part) #appending the partial values

  print("TF-IDF-\n", tf_idf_list)
  #End time for TF_IDF process
```

Other workers:

```
#Data is recieved for TF-IDF calculation
tf_spl, idf_spl = comm.recv(source =0, tag = 8)
#The required function is called and result is send back to master
comm.send(tf_idf(idf_spl,tf_spl), dest = 0, tag = 10)
```

Output:

Time taken:

Number of Workers	Time taken (s)
8	10.4176
12	10.190
16	23.499

Total time taken for running the whole program:

Number of Workers	Time taken (s)
8	104.992
12	129.345

16	192.617

We observe that the time taken is increasing when we increase the workers. This can be due to the fact that the system only has 8 cores and when number of workers is greater than 8, more than one worker is running on the core, which can cause delay due to queue(not sure).

Also we see that the maximum time is taken for data cleaning and tokenization.

References:

- https://mpi4py.readthedocs.io/en/stable/reference/mpi4py.MPI.Comm.html#mpi4py.MPI.Comm.send
- https://numpy.org/doc/stable/reference/generated/numpy.array_split.html
- https://stackoverflow.com/questions/5797615/mpi-cores-or-processors
- https://stackoverflow.com/questions/64311489/siice-a-numpy-array-based-on-an-interval
- https://stackoverflow.com/questions/40336601/python-appending-array-to-an-array
- https://dylancastillo.co/nlp-snippets-clean-and-tokenize-text-with-python/
- https://stackoverflow.com/questions/20510768/count-frequency-of-words-in-a-list-and-sort-by-frequency
- https://www.jquery-az.com/3-ways-convert-python-list-string-join-map-str/
- https://stackoverflow.com/questions/42330201/assign-values-to-array-during-loop-python
- https://www.tutorialspoint.com/python3/os_walk.htm
- https://www.nltk.org/api/nltk.tokenize.html

- https://stackoverflow.com/questions/57009108/term-frequency-calculation-using-python
- https://analyticsindiamag.com/hands-on-implementation-of-tf-idf-from-scratch-in-python/
- https://www.askpython.com/python/examples/tf-idf-model-from-scratch