Mansoor Nabawi, #309498

Tutorial 2, 05.13.2022

Exercise 1: Data cleaning and text tokenization (5 points)

Report 1

Flowchart of the processes

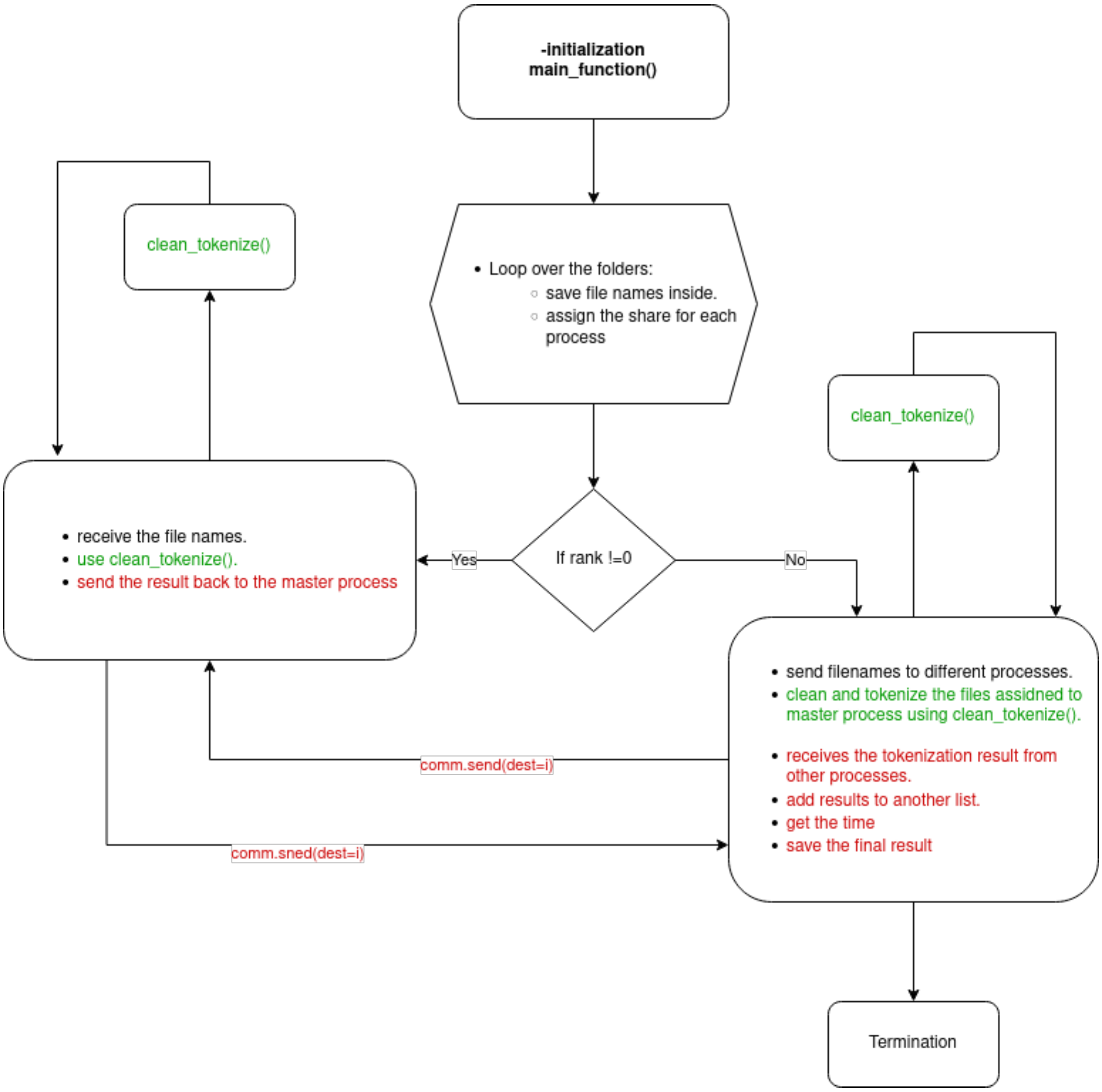

-initialization
main_function()
Loop over the folders:
save file names inside.
assign the share for each process
If rank !=0
Yes
No
clean_tokenize()
receive the file names.
use clean_tokenize().
send the result back to the master process
clean_tokenize()
send filenames to different processes.
clean and tokenize the files assidned to master process using clean_tokenize().
receives the tokenization result from other processes.
add results to another list.
get the time
save the final result
comm.send(dest=i)
comm.sned(dest=i)
Termination

Table of Times for 1

P=1	P=2	P=3	P=4	P=6	P=8	P=10
80.93	43.68	32.3	**25.73**	39.0	35.5	40

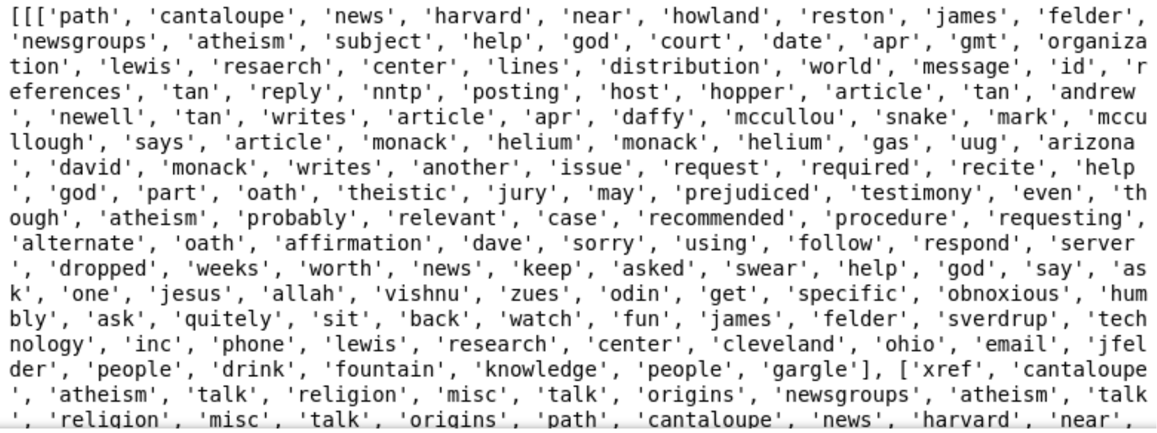
- The result of parallelization is very obvious in this exercise.
- As the number of workers increases, the computation time decreases until p=4.

- our best result is when we use 4 workers.

Strategy

- We loop over the files of each folder.
- we share the files in each folder to different workers.
- each worker will perform cleaning and tokenization on the subset of data it has and send the result back to rank0.
- rank0 is our master process and it also work as worker, but at the end it receives all the data and add them in one big list.

result of exercise 1



```
[[['path', 'cantaloupe', 'news', 'harvard', 'near', 'howland', 'reston', 'james', 'felder',
'newsgroups', 'atheism', 'subject', 'help', 'god', 'court', 'date', 'apr', 'gmt', 'organiza
tion', 'lewis', 'resaerch', 'center', 'lines', 'distribution', 'world', 'message', 'id', 'r
eferences', 'tan', 'reply', 'nntp', 'posting', 'host', 'hopper', 'article', 'tan', 'andrew
', 'newell', 'tan', 'writes', 'article', 'apr', 'daffy', 'mccullou', 'snake', 'mark', 'mccu
llough', 'says', 'article', 'monack', 'helium', 'monack', 'helium', 'gas', 'uug', 'arizona
', 'david', 'monack', 'writes', 'another', 'issue', 'request', 'required', 'recite', 'help
', 'god', 'part', 'oath', 'theistic', 'jury', 'may', 'prejudiced', 'testimony', 'even', 'th
ough', 'atheism', 'probably', 'relevant', 'case', 'recommended', 'procedure', 'requesting',
'alternate', 'oath', 'affirmation', 'dave', 'sorry', 'using', 'follow', 'respond', 'server
', 'dropped', 'weeks', 'worth', 'news', 'keep', 'asked', 'swear', 'help', 'god', 'say', 'as
k', 'one', 'jesus', 'allah', 'vishnu', 'zues', 'odin', 'get', 'specific', 'obnoxious', 'hum
bly', 'ask', 'quitely', 'sit', 'back', 'watch', 'fun', 'james', 'felder', 'sverdrup', 'tech
nology', 'inc', 'phone', 'lewis', 'research', 'center', 'cleveland', 'ohio', 'email', 'jfel
der', 'people', 'drink', 'fountain', 'knowledge', 'people', 'gargle'], ['xref', 'cantaloupe
', 'atheism', 'talk', 'religion', 'misc', 'talk', 'origins', 'newsgroups', 'atheism', 'talk
', 'religion', 'misc', 'talk', 'origins', 'path', 'cantaloupe', 'news', 'harvard', 'near',
```

In [11]:

```
%%writefile DDA02_ex1.py

#importing libraries
import os
import numpy as np
import re
import pickle
#import nltk
from mpi4py import MPI


#loading manual stopwords
from stop_words_man import stopw
stopword_manually = stopw() #created stopwords manually


#initialization
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()


def clean_tokenized(file_n):
    """
    This function receives a list of file names and goes through each file
    unnecessary words and symbols, it also checks every single word not to
    at the end tokenize them by tputting them in a list.
    args:
        - file_n: file names
    output:
        - tokenized words (list)
    """
    #empty lists will be used for final addtion and also filtering words.
    tokenized_words = []
    filtered_line = []
    for file in (file_n):
        with open(file, 'r', encoding = 'Latin-1') as f:
            text = f.read()
            #not word replaced by space
        text = [re.sub("[^a-zA-Z]", " ", f) for f in text.split()]
        text = " ".join(text)#to remove white spaces
        text = text.lower()
        text = [f for f in text.split() if len(f)>1] #single words deleted
        filtered_line = [w for w in text if not w in stopword_manually]
        tokenized_words += [filtered_line] #tokenization

    #with open("/home/mansoor/Desktop/DDA/ex02/saved"+f'/tokenized_rank_{ra
    #    pickle.dump(tokenized_words, fp)

    return tokenized_words




def main_function():
    """
    This function is the main function to run the parallel program on.
    """
    #time
    t0 = MPI.Wtime()

    path = "/home/mansoor/Desktop/DDA/ex02/20_newsgroups/"
    #folders in the path
    folders = os.listdir(path)
    #change the current working directory to path
    os.chdir(path)

    final_list = []
    #going through each folders and files inside of them.
    for i in range(len(folders)):
        line_inner = []
        filenames = []
        inner_path = (path+str(folders[i]))
        os.chdir(inner_path)
        filenames += [f for f in os.listdir(inner_path) if os.path.isfile(
        #determining the share for each process
        share = round(len(filenames)/size)


        #only slaves
        if rank != 0:

            #receiving names of the files
            filens = comm.recv(source=0, tag = 0)
            #assigning files to ditexterent workers except worker 0

            #cleaning and tokenization
            tokenized = clean_tokenized(filens)


            #sending to master node
            comm.send(tokenized, dest=0, tag=1)


        #master process
        else:

            #distributing the data
            for i in range(1, size):

                comm.send(filenames[(share*i):(share*(1+i))], dest = i, ta

            #master node's share of file
            filenames = filenames[(share*rank):(share*(1+rank))]
            #cleaning and tokenization
            tokenized = clean_tokenized(filenames)
            #saving the file
            #with open("/home/mansoor/Desktop/DDA/ex02/saved"+f'/tokenized_
            #    pickle.dump(tokenized, fp)
            #appending
            line_inner.append(tokenized)

            #receiving results
            for i in range(1, size):

                tknzd = comm.recv(source = i, tag = 1)

                line_inner.append(tknzd)



        #appending final result
        final_list.append(line_inner)
    t1 = MPI.Wtime() - t0
    print("Time: ", t1)
    #print(len(final_list))
    #save_csv(final_list, rank=10)
    with open("/home/mansoor/Desktop/DDA/ex02/saved"+f'/final_list_{size}.
        pickle.dump(final_list, fp)
    return final_list

main_func = main_function()
```

```
Overwriting DDA02_ex1.py
```

In [8]:

```
!mpiexec -n 4 python  DDA02_ex1.py
```

```
Time:  25.713739156723022
Time:  25.72397494316101
Time:  25.755807876586914
Time:  25.765300989151
```

Exercise02

Report 2

Flowchart of the processes

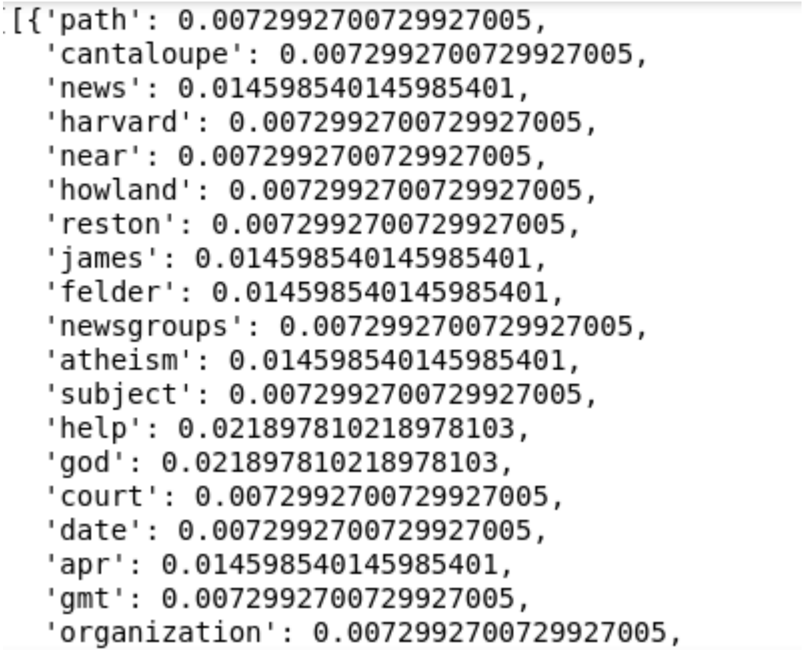
Table of Times for 2

P=2	P=4	P=6
2.9	**2.3**	6.02

- The result of parallelization visible in this exercise
- As the number of workers increases, the computation time decreases but only until 4.
- our best result is when we use 4 workers.

Strategy

- We loop over the files of each folder.
- we share the files in each folder to different workers.
- each worker will perform tf computation on the subset of data it has and send the result back to rank0.
- rank0 is our master process and it also work as worker, but at the end it receives all the data and add them in one big list.



```
[{'path': 0.0072992700729927005,
  'cantaloupe': 0.0072992700729927005,
  'news': 0.014598540145985401,
  'harvard': 0.0072992700729927005,
  'near': 0.0072992700729927005,
  'howland': 0.0072992700729927005,
  'reston': 0.0072992700729927005,
  'james': 0.014598540145985401,
  'felder': 0.014598540145985401,
  'newsgroups': 0.0072992700729927005,
  'atheism': 0.014598540145985401,
  'subject': 0.0072992700729927005,
  'help': 0.021897810218978103,
  'god': 0.021897810218978103,
  'court': 0.0072992700729927005,
  'date': 0.0072992700729927005,
  'apr': 0.014598540145985401,
  'gmt': 0.0072992700729927005,
  'organization': 0.0072992700729927005,
```

result of exercise 2

In [176…

```
%%writefile ex02_tf.py


from mpi4py import MPI
import numpy as np
import os
import pickle

#using saved data
with open ('saved/final_list_1.ob','rb') as fp:
    data = pickle.load(fp)

#initialization
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()
#name = MPI.Get_processor_name()

#batch = rank(len(data-1)/size)

#tf calculator function
def calculate_tf(tokenized_data):
    tf_list = []
    for document in tokenized_data:
        sentence_dict = dict()
        for word in document:
            sentence_dict[word] = sentence_dict.get(word,0)+1
        len_docu = len(document)

        for word in sentence_dict:
            sentence_dict[word] = sentence_dict[word]/len_docu
        tf_list.append(sentence_dict)
    return tf_list





t0 = MPI.Wtime()
def main_function():
    """
    This function is the main function to run the parallel program on.
    """
    #time


    final_list = []
    #going through each folders and files inside of them.
    for i in range(len(data)-1):
        line_inner = []

        #determining the share for each process
        share = round(len(data[i][0])/size)
        #the data is selected
        selected = data[i][0]


        #only slaves
        if rank != 0:

            #receiving names of the files
            filens = comm.recv(source=0, tag = 0)
            #assigning files to ditexterent workers except worker 0

            #calculating tf
            tf_res = calculate_tf(filens)

            #sending to master node
            comm.send(tf_res, dest=0, tag=1)


        #master process
        else:

            #distributing the data
            for i in range(1, size):

                comm.send(selected[(share*i):(share*(1+i))], dest = i, tag

            #master node's share of file
            filens = selected[(share*rank):(share*(1+rank))]
            #tf
            tf_res = calculate_tf(filens)
            line_inner.append(tf_res)

            #receiving results
            for i in range(1, size):

                tff = comm.recv(source = i, tag = 1)
                line_inner.append(tff)



        #appending final result
        final_list.append(line_inner)


    t1 = MPI.Wtime() - t0
    print("Time: ", t1)

    with open("/home/mansoor/Desktop/DDA/ex02/saved"+f'/tf_res_{size}.ob',
        pickle.dump(final_list, fp)
    return final_list

main_func = main_function()
```

```
Overwriting ex02_tf.py
```

In [174…

```
!mpiexec -n 4 python  ex02_tf.py
```

```
Time:  2.2453770637512207
Time:  2.2325971126556396
Time:  2.3546640872955322
Time:  2.2729651927948
```

In [121…

```
#just loading the data of tf
import pickle

#using saved data
with open ('saved/final_list_1.ob','rb') as fp:
    data = pickle.load(fp)
```

Exercise 3

Report 3

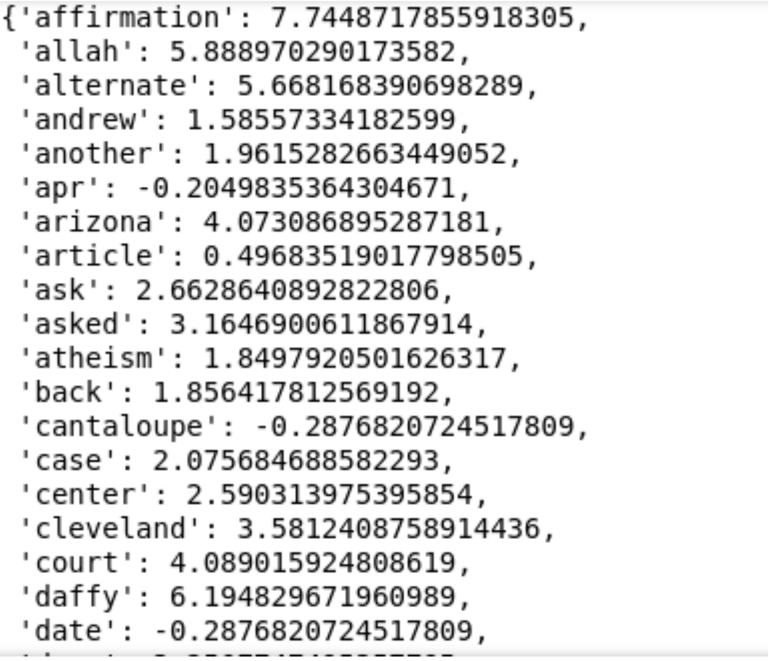
Table of Times for 3

P=2	P=4	P=6
7.9796	**5.9459**	6.0164

- The result of parallelization visible in this exercise
- As the number of workers increases, the computation time decreases.
- our best result is when we use 4 workers.

Strategy

- first we use the tokenized data from previous exercise.
- we append all of the documents in one list.
- word_frequency_in_doc() function goes through the data find the occurence and then ratio of word in the each batch.
- the result sent to the master process and it does the final frequency.
- save the data at the end



```
{'affirmation': 7.7448717855918305,
 'allah': 5.888970290173582,
 'alternate': 5.668168390698289,
 'andrew': 1.58557334182599,
 'another': 1.9615282663449052,
 'apr': -0.2049835364304671,
 'arizona': 4.073086895287181,
 'article': 0.49683519017798505,
 'ask': 2.6628640892822806,
 'asked': 3.1646900611867914,
 'atheism': 1.8497920501626317,
 'back': 1.856417812569192,
 'cantaloupe': -0.2876820724517809,
 'case': 2.075684688582293,
 'center': 2.590313975395854,
 'cleveland': 3.5812408758914436,
 'court': 4.089015924808619,
 'daffy': 6.194829671960989,
 'date': -0.2876820724517809,
```

result of exercise 3

In [161…

```
%%writefile ex03_idf_0.py

#loading libraries
from mpi4py import MPI
import numpy as np
import pandas as pd
import pickle
import os
from collections import defaultdict, Counter

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()


#using saved data
with open ('saved/final_list_1.ob','rb') as fp:
    data = pickle.load(fp)
    total_docs = []
    for i in range(len(data)-1):
        for j in range(len(data[i][0])):
            total_docs.append(data[i][0][j])


def word_frequency_in_doc(data):
    word_folder = dict()
    for i in range(len(data)):
        # Get unique words per document
        words = np.unique(data[i])
        # Counting the times a word has been mentioned in a document
        for word in words:
            if word in word_folder:
                word_folder[word] += 1
            else:
                word_folder[word] = 1
    batch = {k: (len(data) / v) for k, v in word_folder.items()}
    return batch


p_ranks = round(len(total_docs)/(size-1))


total = dict()

#master process
if rank == 0:
    t0 = MPI.Wtime()
    total_docs2 = total_docs[0:p_ranks]
    for i in range(1, size):
        total_docs1 = total_docs[(i*p_ranks):(p_ranks*(i+1))]
        comm.send(total_docs1, dest=i)
    output1 = word_frequency_in_doc(total_docs2)

    global_dict = None
    for i in range(1, size):
        idf = comm.recv()
        output1 = (Counter(output1) + Counter(idf))
    total = {k: np.log(v / (size)) for k, v in output1.items()}

    print('Time:',MPI.Wtime() - t0)
    with open("/home/mansoor/Desktop/DDA/ex02/saved"+f'/idf_res_{size}.ob'
        pickle.dump(total, fp)
```

```
#slaves
else:
    data = comm.recv()
    output = word_frequency_in_doc(data)
    comm.send(output, dest=0)
```

```
Overwriting ex03_idf_0.py
```

In [162…

```
!mpiexec -n 4 python ex03_idf_0.py
```

```
Time: 5.945959806442261
```

In [139…

```
#using saved data
with open ('saved/idf_res_4.ob','rb') as fp:
    idf_ = pickle.load(fp)
```

Exercise 4

Report 4

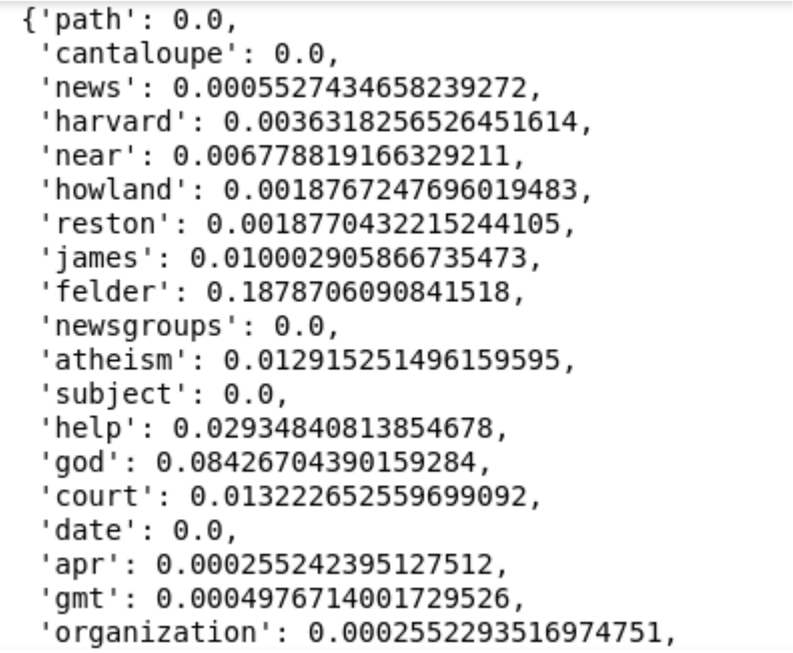
Table of Times for 4

P=2	P=4	P=6
15.98	**12.679**	17.332

- The result of parallelization visible in this exercise
- As the number of workers increases, the computation time decreases.
- our best result is when we use 4 workers.

Strategy

- first we use the tokenized data from previous exercise.
- we append all of the documents in one list.
- the master process sends batches of data to the remaining workers.
- each worker will accomplish 2 tasks: first get the TF for each document and secondly it will get the IDF for all the documents that each has received from the root.
- Master will merge the data from all the processes, measure the final IDF and multiply with each token TF.
- save the data at the end.



```
{'path': 0.0,
 'cantaloupe': 0.0,
 'news': 0.0005527434658239272,
 'harvard': 0.0036318256526451614,
 'near': 0.006778819166329211,
 'howland': 0.0018767247696019483,
 'reston': 0.0018770432215244105,
 'james': 0.010002905866735473,
 'felder': 0.1878706090841518,
 'newsgroups': 0.0,
 'atheism': 0.012915251496159595,
 'subject': 0.0,
 'help': 0.02934840813854678,
 'god': 0.08426704390159284,
 'court': 0.013222652559699092,
 'date': 0.0,
 'apr': 0.000255242395127512,
 'gmt': 0.0004976714001729526,
 'organization': 0.0002552293516974751,
```

result of exercise 4

In [187…

```
%%writefile ex04_tf_idf.py

#loading libraries
from mpi4py import MPI
import numpy as np
import pandas as pd
import pickle
import os
from collections import defaultdict, Counter

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()



#using saved data
with open ('saved/final_list_1.ob','rb') as fp:
    data = pickle.load(fp)
    total_docs = []
    for i in range(len(data)-1):
        for j in range(len(data[i][0])):
            total_docs.append(data[i][0][j])



def calculate_tfidf(data):
    chunk = []
    word_folder = dict()
    for i in range(len(data)):
        word = defaultdict(int)
        unique_tf = np.unique(data[i])
        counter = len(np.unique(unique_tf))
        for i in data[i]:
            word[i] +=1
        dictionary = {k: v / counter for k, v in word.items()}
        chunk.append(dictionary)
        for unique in unique_tf:
            if unique in word_folder:
                word_folder[unique] += 1
            else:
                word_folder[unique] = 1
    batch = {k: (len(data) / v) for k, v in word_folder.items()}

    return chunk, batch




p_ranks = round(len(total_docs)/(size-1))
final_idf = dict()
final = []
dictcalculate_tfidf = dict()
#master process
if rank == 0:
    t0 = MPI.Wtime()
    total_docs2 = total_docs[0:p_ranks]
    for i in range(1, size):
        total_docs1 = total_docs[(i*p_ranks):(p_ranks*(i+1))]
        comm.send(total_docs1, dest=i)
    TF1, IDF1 = calculate_tfidf(total_docs2)
```

```python
    global_dict = None
    for i in range(1, size):
        chunka = []
        TF2, IDF2 = comm.recv()
        TF1 = TF1 + TF2
        IDF1 = (Counter(IDF2) + Counter(IDF1))
    final = TF1 + final
    final_idf = {k: np.log(v / (size-1)) for k, v in IDF1.items()}
    for i in range(len(final)):
        n = final[i]
        for key, value in n.items():
            if key in final_idf:
                dictcalculate_tfidf[key] = (n[key]) * (final_idf[key])
            else:
                None

    print('Time:',MPI.Wtime() - t0)

else:
    data = comm.recv()
    TF, IDF = calculate_tfidf(data)
    output = (TF, IDF)
    comm.send(output, dest=0)


with open("/home/mansoor/Desktop/DDA/ex02/saved"+f'/tfidf_res_{size}.ob',
        pickle.dump(dictcalculate_tfidf, fp)
```

```
Overwriting ex04_tf_idf.py
```

In [188…

```
!mpiexec -n 4 python ex04_tf_idf.py
```

```
Time: 12.67906403541565
```

In [197…

```python
#using saved data
with open ('saved/tfidf_res_4.ob','rb') as fp:
    tfidf_ = pickle.load(fp)
```

In [ ]: