Lab Course: Distributed Data Analytics
Exercise Sheet 2

Syed Wasif Murtaza Jafri- 311226

Exercise 1: Data cleaning and text tokenization

In a text document you encounter words that are not helpful for your final model. For example, punctuations and numbers, meaningless words, common English stopwords etc. Your first task is to preprocess your data so you can remove these words from the documents. Your solution should be based on MPI framework. You have to develop (write code) a distributed algorithm that cleans the data by removing

the above mentioned words/numbers. You can take help of some python libraries i.e. NLTK. The developed program should take a set of documents in raw format as input, and outputs a set of documents that are cleaned and tokenized.
Please explain your solution and how you distribute work among workers.

1. Cleaning: remove all punctuations, numbers and The list of common English stopwords used in this exercise sheet can be found in the reference [4].
2. Tokenize: Tokenize your documents so it is easy to process in the next task i.e. Tokenize words and output as a comma separated document.

First I read the filepaths into a list and then splitted this list among workers. Each worker open files in its list and then clean and tokenize the file are return back to master. Master concatenate these list from workers and write in a file in which each line give tokens of one document.

In [ ]:
```python
import os
from nltk.corpus import stopwords
import nltk
import re
from mpi4py import MPI
import string

comm = MPI.COMM_WORLD # setting up the MPI communicator
total_worker = comm.Get_size() # getting number of workers
rank = comm.Get_rank() # storing rank of each worker
start_time = MPI.Wtime()  # time variable to find time at the end
master = 0  # master with id=0


def cleanTokenizeData(filesInChunk):
    '''Opening files in the Chunk and removing number ,english stopwords,punctuations and
    words smaller than 2 characters from text and then tokenizing the documents
    Each row of return list represent on document'''
    docList = []
    for file in filesInChunk:
        f = open(file, "r")
        docList.append(f.read())

    cleanData = [(lambda x: [word for word in
                             nltk.word_tokenize(re.sub(r'[0-9]', '', x).lower().strip(string.punctuation)) if
                             word.isalnum() and word not in stopwords.words('english') and len(word) > 2])(doc)
                 in docList]
    return cleanData


if rank == master:  # Condition of checking master to distribute data

    def getDocumentList(dir):
        '''Reading files absoulute path from dataset directory and returning it in a list '''
        file_paths = []
        for dir_path, _, filenames in os.walk(dir):
            if len(filenames) > 0:
                for file in filenames:
                    abs_path_train = os.path.abspath(os.path.join(dir_path, file))  # for complete refrence of
                    file_paths.append(abs_path_train)
        return file_paths


    fileList = getDocumentList('testDir')

    chunk_size = len(fileList) // total_worker

    # Splitting file path list between all workers
    for i in range(1, total_worker):  # for loop to send data
        filesInChunk = fileList[(i - 1) * chunk_size:i * chunk_size]
        comm.send(filesInChunk, dest=i, tag=1)

    # master calculation cleaning the chunk
    tokenDocList = cleanTokenizeData(fileList[i * chunk_size:])

    for i in range(1, total_worker):
        tokenData = comm.recv(source=i, tag=3)
        tokenDocList += tokenData

    print('Tokenized and clean Data List:', tokenDocList)
    print('Execution time :', MPI.Wtime() - start_time)

    '''Writing the tokenize data list to a file to be used in next exercise.
    Each Document is separated by new line'''
    with open('TokenizeData.txt', 'w') as f:
        for tokendoc in tokenDocList:
            f.write(str(tokendoc)[1: -1] + '\n') # removing brackets before writing


else:  # workers calculations
    filesInChunk = comm.recv(source=master, tag=1)
    comm.send(cleanTokenizeData(filesInChunk), dest=master, tag=3)
```

I ran the code on 8 workers. More time is taken because of IO operation of reading files.Thats why I could not experiment with less number of workers.

```
fessor', 'turkish', 'studies', 'university', 'michigan', 'list', 'goes', 'serdar', 'argic', 'closed',
'university', 'minnesota', 'tom', 'goodrich', 'professor', 'history', 'indiana', 'university', 'penns
ylvania', 'tibor', 'professor', 'emeritus', 'turkish', 'studies', 'columbia', 'university', 'justin',
 'mccarthy', 'professor', 'history', 'university', 'louisville', 'jon', 'mandaville', 'professor', 'h
istory', 'portland', 'state', 'university', 'oregon', 'robert', 'olson', 'professor', 'history', 'uni
versity', 'kentucky', 'madeline', 'zilfi', 'professor', 'history', 'university', 'maryland', 'james',
 'professor', 'turkish', 'studies', 'university', 'michigan', 'list', 'goes', 'serdar', 'argic', 'clo
sed', 'roads', 'mountain', 'passes', 'might', 'serve', 'ways', 'escape', 'turks', 'proceeded', 'work'
, 'extermination', 'ohanus', 'appressian', 'soviet', 'armenia', 'today', 'longer', 'exists', 'single'
, 'turkish', 'soul', 'sahak', 'melkonian'], ['path', 'newsgroups', 'subject', 'ajerk', 'date', 'apr',
 'organization', 'dimension', 'bbs', 'lines', 'sender', 'daemon', 'apr', 'good', 'case', 'rights', 'a
bortion', 'system', 'phone', 'cute', 'quote', 'computer', 'means', 'never', 'say', 'sorry'], ['path',
 'newsgroups', 'subject', 'argic', 'date', 'apr', 'organization', 'dimension', 'bbs', 'lines', 'sende
r', 'daemon', 'apr', 'aswer', 'one', 'question', 'get', 'retarded', 'system', 'phone', 'cute', 'quote
', 'computer', 'means', 'never', 'say', 'sorry']]
Execution time : 287.82005559999925
PS C:\Users\wasif\DDA Lab\Exercise02>
```

This list of tokenized and clean data is written in 'TokenizeData.txt' which I have placed it in code folder.

Exercise 2: Calculate Term Frequency (TF)

The Term Frequency (TF) is calculated by counting the number of times a token occurs in the document. This TF score is relative to a specific document, therefore you need to normalized it by dividing with the total number of tokens appearing in the document. A normalized TF score for a specific token t in a document d can be calculated as,

$$\mathrm{TF}(t,d) = \frac{n^d(t)}{\sum_{t' \in d} n^d(t')} , \quad (1)$$

where $n^d(t)$ is the number of times a token t appears in a document d and $|d|$ is the total number of tokens in the document d.
Develop an solution using MPI framework and write code. Please explain how you parallelize (or distribute) TF(t,d) calculation. Also explain your strategy from the data division and calculation division point of view as well. Perform a small experiment by varying number workers i.e. $P = \{2, 4, 8\}$. Also verify if you get the same result at the end.

Reading tokenize file from last exercise and storing in a list. Splitted this list into workers. Eeach worker calculates Term Frequency for each word in the document list by using counter and Normalize by dividing total number of terms in the document.At master gathered these chunks into dictionary where every key is a document Id and value is Term frequency of tokens in that documents

In [ ]:
```python
import json
from collections import Counter
from mpi4py import MPI

comm = MPI.COMM_WORLD  # setting up the MPI communicator
total_worker = comm.Get_size()  # getting number of workers
rank = comm.Get_rank()  # storing rank of each worker
start_time = MPI.Wtime()  # time variable to find time at the end
master = 0  # master with id=0


def calculate_Tf(tokenizeDocChunk, docIds):
    '''
    Calculating Term Frequency for each word in the document list by using counter
    Normalizing by dividing total number of terms in the document

    :param tokenizeDocChunk: Chunk of Tokenized Data list
    :param docIds: Chunk of document ids list
    :return: Term frequency of each word in the document
    '''
    tfDictChunk = {}
    i = 0 # for iterating document ids

    for lines in tokenizeDocChunk:
        words = dict(Counter(lines)) # counting term frequency for each word
        numOftokens = sum(words.values()) # total terms in a document
        for key in words:
            words.update({key: words[key] / numOftokens}) # updating normalize term frequency
        tfDictChunk.update({'document' + str(docIds[i]): words})
        i += 1

    return tfDictChunk


if rank == master:  # Condition of checking master to distribute data

    # reading tokenized data text file from last exercise
    with open('TokenizeData.txt', 'r') as f:
        # removing new line and commas and splitting each line with comma(,) into a list
        tokenizeDoc = [(line.rstrip('\n')).replace("'", "").replace(' ', '').split(",") for line in f]

    docIds = range(1, len(tokenizeDoc) + 1)  # for document ids stored as key in dictionary
    chunk_size = len(tokenizeDoc) // total_worker  # each chunk size processed by worker

    for i in range(1, total_worker):  # for loop to send data
        # sending the chunk of tokenized data list to workers
        comm.send(tokenizeDoc[(i - 1) * chunk_size:i * chunk_size], dest=i, tag=1)
        # sending the document id for keys in dictionary
        comm.send(docIds[(i - 1) * chunk_size:i * chunk_size], dest=i, tag=2)

    # master calculation of own chunk
    docFreqDict = calculate_Tf(tokenizeDoc[i * chunk_size:], docIds[i * chunk_size:])

    # receiving chunks of term frequencies from other workers and concatenating them
    for i in range(1, total_worker):
        docFreqDictChunk = comm.recv(source=i, tag=3)
        docFreqDict.update(docFreqDictChunk)

    # ordering the term frequency dictionary with key as document ids
    orderDocFreq = dict(sorted(docFreqDict.items()))

    # writing the TF dictionary to a file for later exercises
    json.dump(orderDocFreq, open("TF.txt", 'w'))


    print("Term Frequencies : document19997", orderDocFreq['document1321'])
    print("Execution Time :", MPI.Wtime() - start_time)  # total time takes to task complition

else:
    # receiving the chunk of tokenized data list from master
    tokenizeDocChunk = comm.recv(source=master, tag=1)

    # receiving the document ids for keys in dictionary
    docIds = comm.recv(source=master, tag=2)

    comm.send(calculate_Tf(tokenizeDocChunk, docIds), dest=master, tag=3)
```

Showing term frequencies for first and last document.

```
PS C:\Users\wasif\DDA Lab\Exercise02> mpiexec.exe -n 2 python .\exer2.py
Term Frequencies : document1 {'path': 0.037037037037037035, 'newsgroups': 0.037037037037037035, 'subject': 0.037037
037037037035, 'argic': 0.037037037037037035, 'date': 0.037037037037037035, 'apr': 0.07407407407407407, 'organization': 0.
037037037037037035, 'dimension': 0.037037037037037035, 'bbs': 0.037037037037037035, 'lines': 0.037037037037037035, 'se
nder': 0.037037037037037035, 'daemon': 0.037037037037037035, 'aswer': 0.037037037037037035, 'one': 0.03703703703703703
5, 'question': 0.037037037037037035, 'get': 0.037037037037037035, 'retarded': 0.037037037037037035, 'system': 0.037037
037037037035, 'phone': 0.037037037037037035, 'cute': 0.037037037037037035, 'quote': 0.037037037037037035, 'computer':
0.037037037037037035, 'means': 0.037037037037037035, 'never': 0.037037037037037035, 'say': 0.037037037037037035, 'sorr
y': 0.037037037037037035}
Execution Time : 1.7063034999955562
PS C:\Users\wasif\DDA Lab\Exercise02> mpiexec.exe -n 4 python .\exer2.py
Term Frequencies : document1 {'path': 0.037037037037037035, 'newsgroups': 0.037037037037037035, 'subject': 0.037037
037037037035, 'argic': 0.037037037037037035, 'date': 0.037037037037037035, 'apr': 0.07407407407407407, 'organization': 0.
037037037037037035, 'dimension': 0.037037037037037035, 'bbs': 0.037037037037037035, 'lines': 0.037037037037037035, 'se
nder': 0.037037037037037035, 'daemon': 0.037037037037037035, 'aswer': 0.037037037037037035, 'one': 0.03703703703703703
5, 'question': 0.037037037037037035, 'get': 0.037037037037037035, 'retarded': 0.037037037037037035, 'system': 0.037037
037037037035, 'phone': 0.037037037037037035, 'cute': 0.037037037037037035, 'quote': 0.037037037037037035, 'computer':
0.037037037037037035, 'means': 0.037037037037037035, 'never': 0.037037037037037035, 'say': 0.037037037037037035, 'sorr
y': 0.037037037037037035}
Execution Time : 1.3614088999966043
PS C:\Users\wasif\DDA Lab\Exercise02> mpiexec.exe -n 8 python .\exer2.py
Term Frequencies : document1 {'path': 0.037037037037037035, 'newsgroups': 0.037037037037037035, 'subject': 0.037037
037037037035, 'argic': 0.037037037037037035, 'date': 0.037037037037037035, 'apr': 0.07407407407407407, 'organization': 0.
037037037037037035, 'dimension': 0.037037037037037035, 'bbs': 0.037037037037037035, 'lines': 0.037037037037037035, 'se
nder': 0.037037037037037035, 'daemon': 0.037037037037037035, 'aswer': 0.037037037037037035, 'one': 0.03703703703703703
5, 'question': 0.037037037037037035, 'get': 0.037037037037037035, 'retarded': 0.037037037037037035, 'system': 0.037037
037037037035, 'phone': 0.037037037037037035, 'cute': 0.037037037037037035, 'quote': 0.037037037037037035, 'computer':
0.037037037037037035, 'means': 0.037037037037037035, 'never': 0.037037037037037035, 'say': 0.037037037037037035, 'sorr
y': 0.037037037037037035}
Execution Time : 1.4460126999983913
```

Results are same when experimenting with different number of workers.

Workers P	Execution Time
2	1.7063034999955562
4	1.3614088999966043
8	1.4460126999983913

Exercise 3: Calculate Inverse Document Frequency ((IDF)

The Inverse Document Frequency ((IDF) is counting the number of documents in the corpus and counting the number of documents that contain a token. While the TF is calculated on a per-document basis, the IDF is computed on the basis of the entire corpus. The final IDF score of a token t in the corpus C is obtained by taking logarithm of document count in the corpus divided by the number of documents in the corpus that contain a particular token. The IDF formula is given as

$$\mathrm{IDF}(t) = \log \frac{|C|}{\sum_{d \in C} \mathbb{1}(t,d)} , \quad (2)$$

where $\mathbb{1}(t,d)$ is an indicator function which returns 1 if a token t exists in document d. $|C|$ is the total number of documents in the corpus.

Develop an solution using MPI framework and write cod. Please explain how you parallelize (or distribute) IDF(t,d) calculation. Also explain your strategy from the data division and calculation division point of view as well. Perform a small experiment by varying number workers i.e. $P = \{2, 4, 8\}$. Also verify if you get the same result at the end.

Reading file containg tf from last exercise as dictionary and then slicing and sending this dictionary to other workers. Each worker Calculates sum of all term frequency for all term in a document and then master joins these dictionary and divide C with each term in this dictionary and taking log to give final Idf Dictionary

In [ ]:
```python
import json
import math
from collections import Counter
from mpi4py import MPI

comm = MPI.COMM_WORLD  # setting up the MPI communicator
total_worker = comm.Get_size()  # getting number of workers
rank = comm.Get_rank()  # storing rank of each worker
start_time = MPI.Wtime()  # time variable to find time at the end
master = 0  # master with id=0


def calculateSumTF(tfDictChunk):
    '''
    Calculating sum of all term frequency for all term in a document

    :param tfDictChunk: List of Term frequency dictionary for each document
    :return:  dictionary of every term frequency across all documents
    '''
    idfDictChunk = {}

    for key in tfDictChunk:
        docDic = tfDictChunk[key]
        for wordKey in docDic:
            if (idfDictChunk.get(wordKey) != None):  # if token already exist, add 1 in token
                idfDictChunk.update({wordKey: idfDictChunk.get(wordKey) + 1})
            else:  # if token does not exist put 1
                idfDictChunk.update({wordKey: 1})
    return idfDictChunk


if rank == master:  # Condition of checking master to distribute data
    # reading term frequency dictionary from last exercise
    tfDict = json.load(open("TF.txt"))

    C = len(tfDict)  # total number of documents

    chunk_size = len(tfDict) // total_worker  # each chunk size processed by worker

    for i in range(1, total_worker):  # for loop to send data

        # slicing dictionary for sending to workers
        tfDictChunk = {'document' + str(k): tfDict['document' + str(k)] for k in
                       range(((i - 1) * chunk_size) + 1, (i * chunk_size) + 1)
                       if 'document' + str(k) in tfDict}
        # sending to other workers
        comm.send(tfDictChunk, dest=i, tag=1)

    ## master calculation of own chunk
    tfDictChunk = {'document' + str(k): tfDict['document' + str(k)] for k in
                   range((i * chunk_size) + 1, len(tfDict) + 1) if
                   'document' + str(k) in tfDict}
    idfDict = Counter(calculateSumTF(tfDictChunk))

    '''receiving chunks from workers and concatinating them to make one dictionary
    in which each term tell in how many documents it appears'''
    for i in range(1, total_worker):
        idfDictChunk = comm.recv(source=i, tag=3)
        idfDict += Counter(idfDictChunk)

    # diving C with each term in this dictionary and taking log to give final Idf Dictionary
    finalIdfDict = {}
    for key in dict(idfDict):
        finalIdfDict.update({key: math.log(C / idfDict[key])})

    # writing the IDF dictionary to a file for later exercises
    json.dump(finalIdfDict, open("IDF.txt", 'w'))

    # printing term starting with abilit
    print("Idf for terms starts with abilit: ", {k:v for k,v in finalIdfDict.items() if k.startswith('abilit')}
    print("Execution Time :", MPI.Wtime() - start_time)  # total time takes to task completion
else:

    # receiving the chunk of term frequency dictionary list from master
    tfDictChunk = comm.recv(source=master, tag=1)

    comm.send(calculateSumTF(tfDictChunk), dest=master, tag=3)
```

Showing Idf for words starting with ast.Remaining Idf are stored in IDF.txt.

```
PS C:\Users\wasif\DDA Lab\Exercise02> mpiexec.exe -n 2 python .\Exer3.py
Idf for terms starts with abilit:  {'ability': 3.9925408972444756, 'abilities': 6.074696144795908}
Execution Time : 1.9553215999985696
PS C:\Users\wasif\DDA Lab\Exercise02> mpiexec.exe -n 4 python .\Exer3.py
Idf for terms starts with abilit:  {'ability': 3.9925408972444756, 'abilities': 6.074696144795908}
Execution Time : 1.8562213000041083
PS C:\Users\wasif\DDA Lab\Exercise02> mpiexec.exe -n 8 python .\Exer3.py
Idf for terms starts with abilit:  {'ability': 3.9925408972444756, 'abilities': 6.074696144795908}
Execution Time : 1.91034939999372
```

Same results when changing number of workers.

Workers P	Execution Time
2	1.9553215999985696
4	1.8562213000041083
8	1.91034939999372

Exercise 4: Calculate Term Frequency Inverse Document Frequency (TF-IDF) scores

In this exercise you will find the TF-IDF(t,d) for a given token t and a document d in the corpus C is the product of TF(t,d) and IDF(t), which is represented as,

$$\mathit{TF\text{-}IDF}(t,d) = \mathit{TF}(t,d) \times \mathit{IDF}(t) , \quad (3)$$

You have already calculated $TF(t,d)$ in Exercise 2 and $IDF(t)$ is Exercise 3.
In this exercise you have to think about how you can combine the complete pipeline in a single parallel/distributed program that can run worker $P = \{2, 4, 8\}$. Develop an solution using MPI framework and write code. Please explain how you parallelize (or distribute) the complete pipeline. Also explain your strategy from the data division and calculation division point of view as well. Perform a small experiment by varying number workers i.e. $P = \{2, 4, 8\}$. Also verify if you get the same result at the end.

Reading TF and IDF from files and storing in dictionaries.Splitting the TF dictionary among all workers while broadcasting IDf dictionary.Calculating tf-idf at each process and then merging the results at master process.

In [ ]:
```python
import json
from mpi4py import MPI

comm = MPI.COMM_WORLD  # setting up the MPI communicator
total_worker = comm.Get_size()  # getting number of workers
rank = comm.Get_rank()  # storing rank of each worker
start_time = MPI.Wtime()  # time variable to find time at the end
master = 0  # master with id=0


def calculateTfIdf(tfDictChunk, idfDic):
    ''' Calculating tf-idf by multiplying tf of each term in document with idf of that term'''
    tf_idfDictChunk = {}
    for key in tfDictChunk:
        docDic = tfDictChunk[key]
        tdfIdfDoc = {}
        for wordKey in docDic:
            if (idfDic.get(wordKey) != None):
                tdfIdfDoc.update({wordKey: docDic.get(wordKey) * idfDic.get(wordKey)})
        tf_idfDictChunk.update({key: tdfIdfDoc})
    return tf_idfDictChunk


if rank == master:  # Condition of checking master to distribute data#

    # reading term frequency dictionary from last exercise
    tfDict = json.load(open("TF.txt"))
    # reading term frequency dictionary from last exercise
    idfDic = json.load(open("IDF.txt"))

    C = len(tfDict)  # total number of documents

    chunk_size = len(tfDict) // total_worker  # each chunk size processed by worker

    for i in range(1, total_worker):  # for loop to send data
        # slicing tf dictionary for sending to workers
        tfDictChunk = {'document' + str(k): tfDict['document' + str(k)] for k in
                       range(((i - 1) * chunk_size) + 1, (i * chunk_size) + 1)
                       if 'document' + str(k) in tfDict}

        comm.send(tfDictChunk, dest=i, tag=1)

        # sending Idf dictionary to all workers
        comm.send(idfDic, dest=i, tag=2)

    ## master calculation of own chunk
    tfDictChunk = {'document' + str(k): tfDict['document' + str(k)] for k in
                   range((i * chunk_size) + 1, len(tfDict) + 1) if
                   'document' + str(k) in tfDict}
    tf_IdfDict = calculateTfIdf(tfDictChunk, idfDic)

    '''receiving chunks from workers and concatinating them to make one dictionary
    of documents where each value is also a dictionary of Terms idf in that document'''
    for i in range(1, total_worker):
        tf_idfDictChunk = comm.recv(source=i, tag=3)
        tf_IdfDict.update(tf_idfDictChunk)

    print("Tf-Idf : document19997:", tf_IdfDict['document19997'])
    print("length of Tf-Idf Dictionary : ", len(tf_IdfDict))

    print("Execution Time :", MPI.Wtime() - start_time)  # total time takes to task completion

    # writing the tf-IDF dictionary to a file
    json.dump(tf_IdfDict, open("tf_Idf.txt", 'w'))
else:  # receiving array parts to sum
    # receiving the chunk of term frequency dictionary list from master
    tfDictChunk = comm.recv(source=master, tag=1)
    # receiving whole IDF dictionary list from master
    idfDic = comm.recv(source=master, tag=2)

    comm.send(calculateTfIdf(tfDictChunk, idfDic), dest=master, tag=3)
```

```
PS C:\Users\wasif\DDA Lab\Exercise02> mpiexec.exe -n 2 python .\exer4.py
Tf-Idf : document19997: {'path': 0.0, 'newsgroups': 0.0, 'subject': 0.0, 'argic': 0.17836078540935624, 'date': 0.0, 'a
pr': 0.005880024274495046, 'organization': 0.0015179456336085145, 'dimension': 0.23110281093167986, 'bbs': 0.138597415
54036764, 'lines': 0.00010757961030013146, 'sender': 0.02251219099773859, 'daemon': 0.17905308663202857, 'aswer': 0.36
679027930685193, 'one': 0.03591382824385434, 'question': 0.08065059628507597, 'get': 0.04888129516026783, 'retarded':
0.27475669968507416, 'system': 0.06466544815178625, 'phone': 0.10796016145501523, 'cute': 0.22190053836506876, 'quote'
: 0.14847906322296858, 'computer': 0.0756437472743931, 'means': 0.10307011736638945, 'never': 0.08134351429313073, 'sa
y': 0.06747638893140558, 'sorry': 0.11653352414953516}
length of Tf-Idf Dictionary :  19997
Execution Time : 2.139510100001644
PS C:\Users\wasif\DDA Lab\Exercise02> mpiexec.exe -n 4 python .\exer4.py
Tf-Idf : document19997: {'path': 0.0, 'newsgroups': 0.0, 'subject': 0.0, 'argic': 0.17836078540935624, 'date': 0.0, 'a
pr': 0.005880024274495046, 'organization': 0.0015179456336085145, 'dimension': 0.23110281093167986, 'bbs': 0.138597415
54036764, 'lines': 0.00010757961030013146, 'sender': 0.02251219099773859, 'daemon': 0.17905308663202857, 'aswer': 0.36
679027930685193, 'one': 0.03591382824385434, 'question': 0.08065059628507597, 'get': 0.04888129516026783, 'retarded':
0.27475669968507416, 'system': 0.06466544815178625, 'phone': 0.10796016145501523, 'cute': 0.22190053836506876, 'quote'
: 0.14847906322296858, 'computer': 0.0756437472743931, 'means': 0.10307011736638945, 'never': 0.08134351429313073, 'sa
y': 0.06747638893140558, 'sorry': 0.11653352414953516}
length of Tf-Idf Dictionary :  19997
Execution Time : 2.0363853000017116
PS C:\Users\wasif\DDA Lab\Exercise02> mpiexec.exe -n 8 python .\exer4.py
Tf-Idf : document19997: {'path': 0.0, 'newsgroups': 0.0, 'subject': 0.0, 'argic': 0.17836078540935624, 'date': 0.0, 'a
pr': 0.005880024274495046, 'organization': 0.0015179456336085145, 'dimension': 0.23110281093167986, 'bbs': 0.138597415
54036764, 'lines': 0.00010757961030013146, 'sender': 0.02251219099773859, 'daemon': 0.17905308663202857, 'aswer': 0.36
679027930685193, 'one': 0.03591382824385434, 'question': 0.08065059628507597, 'get': 0.04888129516026783, 'retarded':
0.27475669968507416, 'system': 0.06466544815178625, 'phone': 0.10796016145501523, 'cute': 0.22190053836506876, 'quote'
: 0.14847906322296858, 'computer': 0.0756437472743931, 'means': 0.10307011736638945, 'never': 0.08134351429313073, 'sa
y': 0.06747638893140558, 'sorry': 0.11653352414953516}
length of Tf-Idf Dictionary :  19997
Execution Time : 2.084294799999043
```

Getting same results for document19997 for changing number of workers.

Workers P	Execution Time
2	2.139510100001644
4	2.0363853000017116
8	2.084294799999043

In [ ]: