# Data cleaning and text tokenization:

The dataset used for this exercise is Twenty News Group Corpus which has 20 categories where each category has 1000 individual files in it. So, to perform data cleaning and tokenization for this large corpus using Parallel proccessing MPI. I have planned on working with 20 parallel processes for this task where my each worker would be working on each category of the dataset. To complete the pre-processing, i have taken help of nltk(natural language programming toolkit). Using the below code: Firstly, i am going to read the contents inside each file, then the contents are tokenized to words using "word_tokenize" from nltk.and then each such tokens, I have converted them to lower case to maintain the similarity(for better comparison). Then punctuation within each word is removed by using "str.maketrans()" , which returns a translation table that maps each character in the intabstring into the character at the same position in the outtab string.Next, I have removed the charaters which are not alphabetic("like numbers") from the data.There are certain stop words(like I, Me, this, those etc.. for "english") which need to be removed as they carry less special meaning than the keywords. After successful running of this code you would see a clean text which is divided to a set of tokens. I.e in my parallel distribution my each worker would take raw text files as input and would preprocess the data and generate the respective list of tokens specific to those files. I have planned of writting my preprocessed output into a file, and then later i planned it to do after performing my next exercises. My next exercise deals with the same task but additionally to calculate the tf, idf and tf.idf of each token inside the document.

In [ ]:

```
"""
Created on Sun Apr 22 20:20:07 2018

@author: saikiran
"""

from mpi4py import MPI
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string
import os
comm = MPI.COMM_WORLD
rank = comm.rank
print("my rank is:", rank)
start_time = MPI.Wtime()
print("start time is:",start_time)
n=20
if rank == 0:
    #list the files inside this particular path
    dir_list = os.listdir('path of your dataset directory')
    print(dir_list)
    scat_variable = [(dir_list[i])for i in range(n)]
else:
    scat_variable = None
receive = comm.scatter(scat_variable, root=0)
print("process={},document shared is={}".format(rank,receive))
arr = os.listdir('Path of your dataset directory' + receive)
for each_file in arr:
    filename = 'Path of your dataset directory' + receive +'\\'+each_file
    file = open(filename, 'rt')
    text = file.read()
    file.close()
    #splitting the text to words
    tokens = word_tokenize(text)
    # convert to words to lower case
    tokens = [w.lower() for w in tokens]
    # remove punctuation from each word
    table = str.maketrans('', '', string.punctuation)
    stripped = [w.translate(table) for w in tokens]
    # remove remaining tokens that are not alphabetic
    words = [word for word in stripped if word.isalpha()]
    # filtering out stop words
    stop_words = set(stopwords.words('english'))
    words = [w for w in words if not w in stop_words]
    print(words)
    end_time = MPI.Wtime()
    print("end time is:",end_time)
    print("total execution time is :",end_time-start_time)
    '''with open(filename,'w') as f:
        f.write(words)'''
```

## Calculate Term Frequency (TF)

In this task, I have preprocessed my data using 4 workers, which means that my each worker would take 5 categories and work on preprocessing the raw text.Preprocessing is done in the same way as explained in the previous exercise. But in addition to preprocessing, i have calculated the Term-Frequency score of each token with respect to each document. The term frequency of a token can be calculated using the formula: TF(t,d)= nd(t)/|d|. where nd(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.It is calculated for each document level. After succesful generation of list of tokens from a document. I have used Counter from collections which would provide the counter object as output with the count of number of occurences of each token from the given list. I prefered counter object for my rest of my exercise as its easy to make calculation between the counter objects without any need of further iterations. From the Counter object, I calculated the TF_score of each token using the given formula. After calculating the successful TF_scores of tokens for all documents, I have written each one of them into a separate files using "json.dumps"(converts my counter object to json format, as we can't write counter objects into a file directly.Note: this can also be done using pickle.dump but since the content written into the file using pickle is in unreadable format, i have used json approach) which would be helpful for me to calculate the tf_idf score in my next exercise.(Note: I have created a separate directory folder path to save all my tf,idf and tf_idf results)

In [ ]:

```
"""
Created on Mon Apr 23 23:38:19 2018

@author: saikiran
"""


from mpi4py import MPI
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import json
import string
import os
from collections import Counter


comm = MPI.COMM_WORLD
rank = comm.rank
print("my rank is:", rank)
start_time = MPI.Wtime()
print("start time is:",start_time)



if rank == 0:
    dir_list = os.listdir('Path of your dataset directory')
    chunks = [dir_list[x:x+5] for x in range(0, len(dir_list), 5)]
    print(chunks)
    scat_variable = [(chunks[i])for i in range(len(chunks))]
else:
    scat_variable = None
receive = comm.scatter(scat_variable, root=0)
print("process={},document shared is={}".format(rank,receive))
for each_document in receive:
    arr = os.listdir('Path of your dataset directory' + each_document)
    for each_file in arr:
        filename = 'path of your dataset directory' + each_document +'\\'+each_file
        file = open(filename, 'rt')
        text = file.read()
        file.close()
        #splitting text to words
        tokens = word_tokenize(text)
        # convert words to lower case
        tokens = [w.lower() for w in tokens]
        # remove punctuation from each word
        table = str.maketrans('', '', string.punctuation)
        stripped = [w.translate(table) for w in tokens]
        # remove remaining tokens that are not alphabetic
        words = [word for word in stripped if word.isalpha()]
        # filtering out stop words
        stop_words = set(stopwords.words('english'))
        words = [w for w in words if not w in stop_words]
        #print(words)
        end_time = MPI.Wtime()
        #print("end time is:",end_time)
        #print("total execution time is :",end_time-start_time)
        tf_score = Counter(words)
        total_score = sum(tf_score.values())
        for x in tf_score:
            tf_score[x] = tf_score[x]/total_score
        #print(tf_score)

        newpath = r'Path of the file where you need to print your results' + each_document
        if not os.path.exists(newpath):
            os.makedirs(newpath)
        with open(newpath+each_file+".txt", 'wt') as f:
            f.write(json.dumps(tf_score))
            f.close()
```

## Calculate Inverse Document Frequency (IDF)

In this task, I have preprocessed my data using 4 workers, which means that my each worker would take 5 categories and work on preprocessing the raw text.Preprocessing is done in the same way as explained in the previous exercise. But in addition to preprocessing, i have calculated the Inverse Document Frequency score of each token present in all documents with respect to the corpus. The Idf score of each token can be calculated using the formula : IDF(t) = log (|C|/sum(I(t, d))), where |C| is the total number of documents present in the corpus and sum(I(t, d)) is the sum of total number of documents in which a particular token is present.(Note: no matter how many times a token is appearing in the document. If the token is present then its value with respect to the document is considered as 1.). The logarithm which is used to calculated IDF_score is log to the base 10. After succesful generation of list of tokens from a document. I have used Counter from collections which would provide the counter object as output with the count of number of occurences of each token from the given list. I prefered counter object for my rest of my exercise as its easy to make calculation between the counter objects without any need of further iterations. Now for calculating the idf_score, no matter how many times a token appears in the document, i initialised each token present in each document value to 1 if the token is present in it. Now i have added all my counter objects and added the count of total number of documents. I used MPI.Reduce to send all my counter_objects from each worker to the master or root node using "MPI.SUM" as operation. Now my Master node will have a single counter object with a huge list of tokens along with their number of occurances in each document and also the total number of documents present in the corpus. Then i have calculated the IDF_score of each token using the given formula.After calculating the successful IDF_scores of all tokens in the corpus, I have written them into a separate file using "json.dumps"(converts my counter object to json format, as we can't write counter objects into a file directly.Note: this can also be done using pickle.dump but since the content written into the file using pickle is in unreadable format, i have used json approach) which would be helpful for me to calculate the tf_idf score in my next exercise.(Note: I have created a separate directory folder path to save all my tf,idf and tf_idf results)

In [ ]:

```
"""
Created on Mon Apr 23 23:38:19 2018

@author: saikiran
"""
import math
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import json
from mpi4py import MPI
import string
import os
from collections import Counter

comm = MPI.COMM_WORLD
rank = comm.rank
print("my rank is:", rank)
start_time = MPI.Wtime()
print("start time is:",start_time)
file_count = 0
#final_idf_data_list=[]
#final_idf_data_list = Counter(final_idf_data_list)
file_total ={}
file_total = Counter(file_total)
if rank == 0:
    dir_list = os.listdir('Path of your dataset directory')
    #print(dir_list)
    chunks = [dir_list[x:x+5] for x in range(0, len(dir_list), 5)]
    #print(chunks)
    scat_variable = [(chunks[i])for i in range(len(chunks))]
else:
    scat_variable = None
receive = comm.scatter(scat_variable, root=0)
print("process={},document shared is={}".format(rank,receive))
for each_document in receive:
    arr = os.listdir('path of your dataset directory' + each_document)
    for each_file in arr:

        filename = 'path of your dataset directory' + each_document +'\\'+each_file
        file = open(filename, 'rt')
        text = file.read()
        file.close()
        #splitting to words
        tokens = word_tokenize(text)
        # convert to lower case
        tokens = [w.lower() for w in tokens]
        # remove punctuation from each word
        table = str.maketrans('', '', string.punctuation)
        stripped = [w.translate(table) for w in tokens]
        # remove remaining tokens that are not alphabetic
        words = [word for word in stripped if word.isalpha()]
        # filter out stop words
        stop_words = set(stopwords.words('english'))
        words = [w for w in words if not w in stop_words]
        #print(words)
        end_time = MPI.Wtime()
        #print("end time is:",end_time)
        #print("total execution time is :",end_time-start_time)
        idf_score = Counter(words)
        file_count += 1
        #total_score = sum(tf_score.values())
        for x in idf_score:
            if idf_score[x]>0:
                idf_score[x] = 1
            else:
                idf_score[x]= 0
        file_total = file_total+idf_score

root=0
document_count = comm.reduce(file_count,root=root,op=MPI.SUM)
#print(document_count)
data = comm.reduce(file_total,root=root,op=MPI.SUM)
if rank==root:
    total_number = document_count
    final_idf_data_list = data
    print("before",final_idf_data_list)
    for x in final_idf_data_list:
        final_idf_data_list[x] = math.log10((total_number)/final_idf_data_list[x])
    #print("after:",final_idf_data_list)
    newpath = r'path of the file where you print your idf results' +'\\idf_results\\'
    if not os.path.exists(newpath):
        os.makedirs(newpath)
    with open(newpath+"idf_result.txt","wt") as f:
        f.write(json.dumps(final_idf_data_list))
        f.close()
```

## Calculate Term Frequency Inverse Document Frequency(TF-IDF) scores

In this Exercise, I have used 4 workers,which means that my each worker would take 5 categories and work on calculating the tf_idf scores of each token.I have used the results which were obtained from previous exercises i.e tf and idf respectively.Firstly, i have loaded my idf results and tf results using "json.loads" which would load the json format of the file and is of dict type. I scattered my Tf results among 4 workers equally and Now i have broadcasted the idf_score to all the workers so that each worker would have a copy of idf result which they utilise to calculate for tf_idf scores of each token assigned to them. The tf_idf score for each token is calculated using the formula : TF-IDF(t, d) = TF(t, d) × IDF(t).Now for each tf_score result file present in the each worker would be multiplied with the idf_result. Multiplication is done in such a way that only if the token present in each tf matches with the token in idf then their values are multiplied else the token's value is considered as zero. I have stored the multiplication results into another dictionary of counter object type. I have written each one of the tf_idf result into a separate files using "json.dumps"(converts my counter object to json format, as we can't write counter objects into a file directly.Note: this can also be done using pickle.dump but since the content written into the file using pickle is in unreadable format, i have used json approach) Note: I have created a separate directory folder path to save all my tf,idf and tf_idf results.

In [ ]:

```
"""
Created on Tue Apr 24 23:14:50 2018

@author: saikiran
"""
from mpi4py import MPI
import json
import os
from collections import Counter

comm = MPI.COMM_WORLD
rank = comm.rank
print("my rank is:", rank)
start_time = MPI.Wtime()
print("start time is:",start_time)
if rank == 0:
    idf_result= open('path of the idf result file','rt').read()
    idf =  json.loads(idf_result)
    #print(type(idf))
    bcast_variable = idf
    #print(type(idf))
    dir_list = os.listdir('path of the tf result file')
    #print(dir_list)
    chunks = [dir_list[x:x+5] for x in range(0, len(dir_list), 5)]
    #print(chunks)
    scat_variable = [(chunks[i])for i in range(len(chunks))]
else:
    bcast_variable = None
    scat_variable = None
receive1 = comm.scatter(scat_variable, root=0)
receive2 = comm.bcast(bcast_variable,root=0)
#print("process={},document shared is={}".format(rank,receive1))
#print("process={},document shared is={}".format(rank,receive2))
for each_document in receive1:
    arr = os.listdir('path of tf file results' + each_document)
    for each_file in arr:
        filename = 'path of tf file results' + each_document +'\\'+each_file
        tf_results = open(filename, 'rt').read()
        tf = json.loads(tf_results)
        tf_idf = Counter(dict((k, v * receive2[k]) for k, v in tf.items() if k in receive2)
        #print(tf_idf)
        newpath = r'path of the file where you print your tf-idf results' + each_document +
        if not os.path.exists(newpath):
            os.makedirs(newpath)
        with open(newpath+each_file, 'wt') as f:
            f.write(json.dumps(tf_idf))
            f.close()
```

## My results for tf, idf, tf_idf would be same for any number of workers. As my workers are dividing the

## tasks on news-category basis. and the results on each document inside the category any how remains the same.

```
In [ ]:
```