<u>Team Member and contribution -</u>

1. Anu Yadav
2. Pravar Mahajan

We both worked equally on this project
Files are stored in /home/3/mahajan.89/CSE_5243. Please store all sgm files in data/input/.

<u>Problem description and assumptions -</u>

A set of 20 documents is given. Each file has around 10k reuters tag which contains tags like topic, places, people, body etc. Our main task is to create feature vectors using document frequency, tf-idf and bigrams.

<u>Description of Proposed Solution - Overview -</u>
We propose to solve the problem in 3 steps:

(1) Parsing XML documents - Since the documents are presented in XML format, the first step is to parse the relevant sections of the XML. For our purpose, we have used only four sections - body, title, places and topics. Other sections were discarded.
(2) Tokenizing and Stemming - This step involves conversion of raw string documents into tokens which can be converted into vector via the bag of words model. The tokens are stemmed to reduce the number of features as well as map different forms of same words to one representation.
(3) Conversion of tokens into vectors - In this step, the tokens are converted into word vectors. Three methods have been used for this conversion: Word Frequency, Bi-grams and Term Frequency - Inverse Document Frequency (TF-IDF)

<u>Processing Steps</u> -

- *Parsing* - This step involves extracting relevant tags from the given dataset which is in XML format. We replaced newline characters with whitespaces, since newlines get extracted as `\n` and add noise to our documents. BeautifulSoup python library is used.
- *Preprocessing* - This step involves :
  - *Tokenizing* - We have taken document as a string. This step involves splitting string by spaces, skipping non-ASCII characters, removing punctuations and numbers and converting all uppercase characters to lowercase. We also removed all the stopwords, like ‘the’, ‘is’, ‘a’, ‘an’ etc. The list of stopwords can be obtained as `nltk.corpus.stopwords.words('english')`

    - *Stemming* - Here we have converted words to their root word using inbuilt nltk library functions. Words are stemmed using Porter Stemmer algorithm, available as an NLTK package.
- *Vector creation* - In this step, we convert the documents, represented as collection of tokens, into a feature matrix. We have chosen three ways of representing the documents as feature matrix - Word Frequency, Bigrams and TF/IDF. To generate these matrices, we used `CountVectorizer` and `TfIdfTransformer` available in `sklearn` package.

Python Libraries used -

- BeautifulSoup
- Nltk
- Numpy
- Json
- Sklearn
- Scipy
- Os
- glob

Interpretation of code's output:

The code produces output files corresponding to various steps of processing. All these output files are saved in `data/output` directory. Here's a description of different output files produced:

(1) *parsed_documents.txt* - After parsing the XML data and extracting the relevant sections, the parsed documents are dumped into this file in JSON format. This file is a list of about 21,000 dictionary elements, each element corresponding to each document. Each element has four keys - body, title, topics and places.
(2) *unigram_features.dat* - This file contains all the unigram features which have been extracted from all the documents. The list of features is ordered, ie, the word in the n^{th} line of the file corresponds to $(n-1)^{th}$ column in the word frequency/tf-idf matrix.
(3) *bigram_features.dat* - This file contains all the bigram features which have been extracted from all the documents. The list of features is ordered, ie, the bigram in the n^{th} line of the file corresponds to the $(n-1)^{th}$ column in the bigram
(4) *word_freq_matrix.mtx* - This file contains the word frequency matrix. Since the matrix is very big and sparse, we chose the matrix market representation to store the contents of the word frequency matrix. The first row contains 3 space separated numbers, representing number of rows, number of columns and sum of all elements in the matrix. After that, each row describes a row coordinate, a

column coordinate and the element corresponding to that (row, column) in the matrix. The row coordinate corresponds to the document number[1] and the column coordinate corresponds to feature number. For example, a line which reads as “4 30579 3” signifies that in the 4th document, feature number 30579, ie, “weekday” (line 30580 of features.dat) occurs with a frequency of 3.

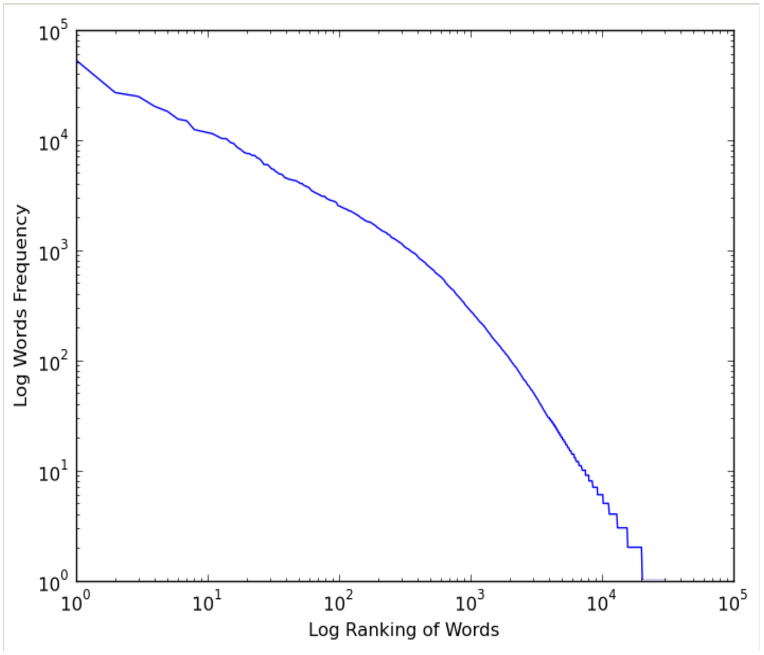
(5) *bigram_matrix.mtx* - This file contains the bigram frequency matrix. Similar to word_freq_matrix, we have used the matrix market representation to store the contents of the bigram matrix.

(6) *tf_idf_matrix.mtx* - This file contains the tf-idf matrix in the same matrix market format, as described in (4).

(7) *places_labels.dat and topics_labels.dat* - The places and topics labels are stored separately in place_labels.dat and topics_labels.dat respectively. We chose to store them separately because the number of topics/labels is not fixed in each document. Each line is a comma separated list of topics or places. Topic corresponding to document #n occurs in the nth line of topics_labels.dat and similarly places corresponding to document #n is present in nth line of the places_labels.dat.

<u>A test for correctness of our implementation:</u>

We perform a statistical test to verify the correctness of our implementation. We plot the frequency of unigram tokens obtained against the rank of the tokens with respect to their frequency of occurrence. The graph shows that the distribution of the unigrams tokens that we generated approximately follows the Zipf’s law:



[1] Document number - If we dump all reuters tag in one file then, this number corresponds to a particular reuter tag.