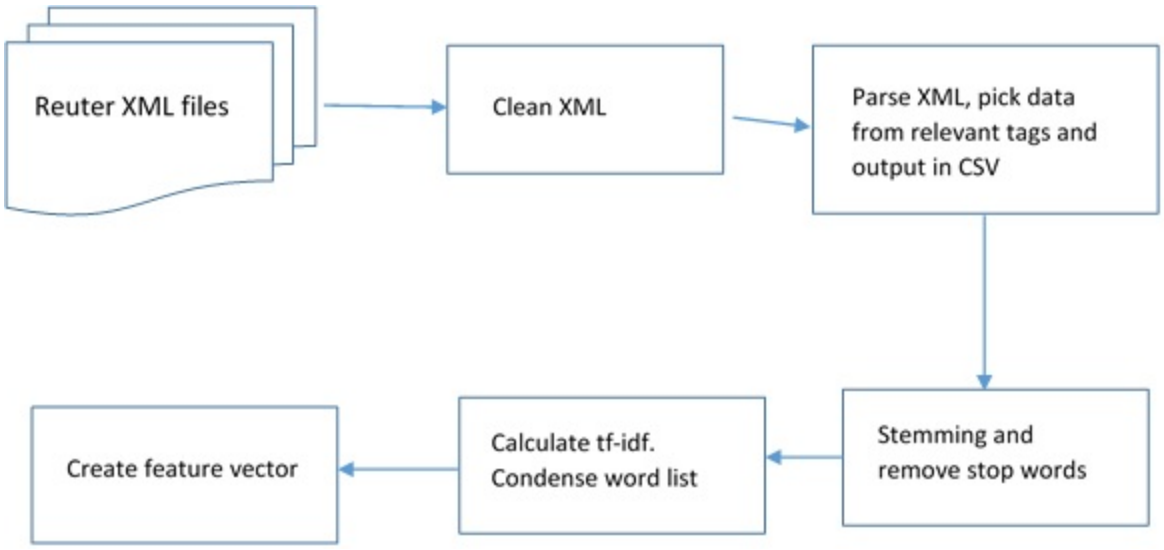
Shashank Agarwal
agarwal.202@osu.edu

Anurag Kalra
kalra.25@osu.edu

Report 1

The objective of this project is to pre-process the XML data files and prepare it for categorization, document similarity search and building document graphs (to be done in later modules). In this project part we calculate tf-idf for each word and create a feature vector. The output table has documents as row and words as column.

Steps:



Procedure:

We were given 22 '.sgm' files containing XML data. The data consists of Reuter's articles. There are around 21,500 article entries in the corpus.
We started with Java implementation and switched to Python later. In our implementation, we traverse each file individually. First we clean the XML data to remove special characters like &#5; etc. which are not useful from data mining perspective. We do other type of pre-processing also like inserting a root node to satisfy Java parser requirements. For each <REUTERS> element, we extract the relevant child elements. We get the content present in elements: <DATE>, <TOPICS>, <PLACES>, <TITLE> and <BODY> and ignore the structural tags like <D>.

Shashank Agarwal
agarwal.202@osu.edu

Anurag Kalra
kalra.25@osu.edu

The output of this cleaning code is a CSV file, where each row is an article containing multiple columns (like title, body, date). Then, we tokenize the body text. We use stemming and also remove the stop-words from the token list using a static stop word list.
After getting a clean list of words, we calculate the word frequency in given documents.
Two type of frequencies are calculated:

- Number of times each word appear in all documents
- Number of documents having the word

This frequency information is captured in 2 separate files: word_out_2.csv and word_out.csv

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).
IDF(t) = log_e(Total number of documents / Number of documents with term t in it).

tf-idf = tf(t) * idf(t)
We use these frequencies to calculate the term frequency-inverse document frequency (tf-idf) for each word. Calculating tf-idf allows us to further prune the word list and get a condensed set of words for each document.

After processing the training data, we were able to reduce the number of interesting words to: 2823

Finally we create the feature vector representing Document Vs Word. The word list form the dimension of the matrix and each row represents a document. If a word is contained in a document, we mark its value as 1 in matrix, if not, we mark it as 0.
The final output is in file: final_tdidf.csv

Modules Used:
We use nltk's **PortStemmer** for stemming and we used python's inbuilt hash tables (dict) for calculating word frequency distribution and for calculating tf-idf.

Problems:
The major problem with data pre-processing was invalid data. Eg. Java parser will throw exception whenever &#..; characters are encountered. Similarly, the parser will fail because it expects a XML to have root node. We fixed these problems by cleaning the files before trying to parse it.

Steps Involved:
1) First we extracted all files from the sgm files and saved it in a csv called "data.csv"

Shashank Agarwal
agarwal.202@osu.edu

Anurag Kalra
kalra.25@osu.edu

input= .sgm files file=cleanXML.jar output=files/pre_processing.csv

2) We then used this to remove all the stop words and saved the file to "out_file.csv"
input=data.csv file = read.py output=out_file.csv

3) Now we do stemming and create two files
a) Frequency of each word across all documents : word_out_2.csv
b) Number of documents where the word is present: word_out.csv
input=out_file.csv file=read_py.py output=word_out.csv & word_out_2.csv

4) We then calculated the tf-idf of each word and saved it to the file 'tdidf.csv'
input= word_out.csv & word_out_2.csv file=tdidf.py output = tdidf.csv

5) We use only the words with tf-idf of greate than 0.01, which results in 2823 words
input=tdidf.csv file=feature.py
output=final_tdidf.csv

6) We then create the feature vector using the list of words as one axis and the document body as the other. Final results are stored in 'final_tdidf.csv'
input = final_tdidf.csv, data.csv file=create_feature.py
output=feature_matrix.pytext

Team members:
Shashank Agarwal (agarwal.202@osu.edu)
Anurag Kalra (kalra.25@osu.edu)