# CSE5243 Assignment 1

## 1. Parsing the file

Since every file contains more than one piece of REUTERS news, every file isn't a standard XML file. When we read the file, we add the <REUTERSList> tag at the beginning of the file and </REUTERSList> tag at the end of file to make sure it is a standard XML file.

We use **Beautiful Soup(http://www.crummy.com/software/BeautifulSoup/)** to parse every file. For every router news, we get the NEWID, TOPICS, TITLE and BODY contents.

## 2. Determine the vocabulary of terms

**NLTK(**[**http://nltk.org/**](http://nltk.org/)**)** is used to determine the vocabulary of terms in the TITLE and BODY contents.

(1) Tokenization

We tokenize the text into sentences. For every sentence, we chopped it up into tokens at the same time.

(2) POS Tagging

We tag each token and only tokens which is an noun, verb, adjective and adverb are left. Since the pos tags in the treebank are different from the tags in the wordnet, we map the pos tags in the treebank to the tags in the wordnet when we perform the lemmatization.

(3) Normalization

We normalized the tokens into the lower cases and throw away the stop words, punctuations, numbers and tokens whose length is 1.

(4) Lemmatization

We lemmatize the tokens with WordNet.

(5) Merging synonyms

For the tokens which are synonyms, we replace them with the same token.

(6) Stemming

We have tried two popular stemming algorithms inside NLTK: Porter Stemmer and Snowball English Stemmer. However, the generated result is not as good as just using the WordNet Lemmatizer with pos-tag. For example, the word “company” is stemmed into “compani”, and the word “venture” is stemmed into “ventur”. We believe that a non-dictionary-based stemming algorithm does not provide sufficiently accurate result in general. As a result, we decide not to use any of those stemming algorithms at all.

## 3. Constructing the feature vectors

We construct two feature vectors: the term frequency vector and tf-idf vector.

We use the class labels as provided in the TOPICS tags of each article.

**The term frequency vector:**

For every token in the document, we calculate the number of times that term t occurs in the document.

**The tf-idf vector:**

tf–idf is the product of two statistics, term frequency and inverse document frequency. Considering that sometime, the term frequency is so large that it has a dominant impact on the final value, we use the augmented frequency, which is shown below.

A description...

f(t, d) : the frequency of term t in document d.

The inverse document frequency is obtained by dividing the total number of [documents](http://en.wikipedia.org/wiki/Documents) by the number of documents containing the term, and then taking the [logarithm](http://en.wikipedia.org/wiki/Logarithm) of that [quotient](http://en.wikipedia.org/wiki/Quotient).

A description...

*D*: the total number of documents in the corpus

## 4. Implementation Details

(1) Since the text in the TITLE is much more important than the text in the BODY, we assign a higher weight for the tokens which occure in the TITLE text. The weight is proportional to the length of all tokens in the document.

titleWeight = int(len(tokens) \* 0.05 + 1)

By ceiling the title weight, we can ensure the tokens exist even when the body text is empty while having little effect on other documents.

(2) For the news without topics, we assign 'None' to the class label.

(3) For the news with multiple topics, we assign multiple topics to the class label, each topic separated by a semicolon.

(4) In order to reduce the length of vector and avoid the unnecessary computation, we only include the terms with high frequency in the vector. The frequency threshold is 1.

5. Other issues not implemented yet

(1) We don't consider the spelling correction.

(2) We don't extract the name entities which maybe more important than regular tokens.

## 5. Future Works

Currently our pre-processor provides good(accurate) result, but it runs a little bit slow. It usually takes 1-5 seconds to process a single article, depending on its length. Obviously this is not

Performance

Multithreading