What's in the Vector

A Search Engine for Text Retrieval
(Based on Vector Space Model)

Design Document

Aditya Agarwal 2013A7PS062P
Gyanendra Mishra 2013A7PS126P
Prabhjyot Singh Sodhi 2013A7PS151P
Rishabh Agarwal 2013A7PS148P
Varun Wachaspati J 2013A7PS166P

Part -1 : Tokenization and Normalization

All files from the corpus were read and tokenized using Python's NLTK library which basically removed stop words and Python's in-built string module translate method to remove punctuations. These tokens were then lemmatized and stemmed using the NLTK's implementation of Porter Stemmer algorithm. These tokens are now written into different python files named after the original file name(as it was in the corpus for faster retrieval purposes) in a different directory called tokens as a python **List** data structure(as dynamic appending is possible and operations such as finding length of list is O(1)). Also reading from file stream has a larger overhead than importing from python file.

Part-2 : Building Inverted Index

The data structure used for defining the index is Python **dictionary** (Hashmap) which maps the individual tokens generated in the previous step to a Python **List** of 2-Dimensional **Tuples** which stores the document name and term frequency of the token in that particular document in the tuple. But parsing a dictionary of this size of such huge size takes up a lot of time hence we make a different python file for each and import them as done in the previous part.

Part-3 : Building Document Vectors

As we our corpus is static in nature, building the vectors dynamically at run time would redundant overhead hence we compute these document vectors beforehand. We are implementing Logarithmic Normalized tf-idf scoring to compute these vectors.

$$w_{t,d} = \log(1 + \mathrm{tf}_{t,d}) \times \log_{10}(N / \mathrm{df}_t)$$

Term frequency was computed in the previous part for individual document- token pair. Document frequency is the length of the **List** generated in the previous part.

Individual vectors are represented by Python **List** (owing to its dynamic append feature) and stored in different python files (as importing them is faster than reading from a single large file) in the doc_vector directory.

Part-4 : Search Query Retrieval

The query is taken as an input, converted into tokens, lemmatized and normalized. Further from this list of tokens a vector is generated following the same weighting function as above. The angle between the vector generated and all the document vectors containing the tokens is found and stored in a python **List**, sorted in ascending order using the in built sort function and the list thus generated is our final output. Top 10 results are shown per page.