**COM 4115/6115 Text Processing (2016/17)**

Assignment: Document Retrieval

Pan Chengdong 2016.11.13

**Introduction**

This system is a basic document retrieval system, based on the vector space model. Vector Space Model is an algebraic model for representing text documents. In this system, I am using a classic vector space model: TF-IDF weights. What's more, this system also supports query only using Standard Boolean Model. The vector space model has the following advantages over the Standard Boolean model: Simple model based on linear algebra, Term weights not binary, Allows ranking documents according to their possible relevance and so on. In the implementation part, I will give details of how I implement these two models to build my document retrieval system.

**Implementation**

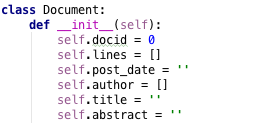
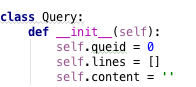
I implemented my document retrieval system based on the ‘Subtasks and a possible breakdown of work’ in assignment description file. In this part I will illustrate how I complete each subtask and reach the final goal.

1. **Command line options:**

I implemented this subtask according to the CommandLine code in ‘eval\_ir.py’. For details of options you can read the README.md file. It has to be noted that my system is running in python 2.X environment, there may be some compatibility errors if in other python version.

1. **Load documents and quires:**

In this subtask, I modify the origin code in ‘read\_documents.py’ to store more details about each document or query. The mainly modification is added more properties for Document class and added Query class. Details illustrate in below picture:

The post\_date, author, title, abstract properties are used to store specific information of each document, which can be useful if more precise query needed in future.

1. **Tokenization and preprocessing:**

I am using the word tokenization method provided by nltk library to tokenize documents and queries, which will split sentences by single space and return a list of word contained in the sentences:

../../../../Desktop/Screen%20Shot%202016-11-13%20at%2011.58.48%2

The preprocessing step contains two sub-steps: remove punctuations from tokenized list and remove stop words from the list if needed. This sub task is implemented in removed\_stop\_list method, in which stop word and punctuations are stored into a python set for better performance.

1. **Stemming:**

I implemented this sub task according to the hint in assignment description file:

../../../../Desktop/Screen%20Shot%202016-11-14%20at%2012.02.54%2

1. **Build two-level dictionary index:**

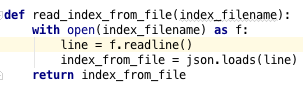
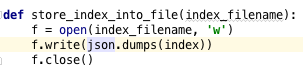
After loading documents and a list of preprocessing operations, I started to build the core index structure into a two-level python dict data structure. The structure is illustrated:



The first level of key, value pairs are: key is each term in the documents (after stemming) and value is a dictionary, which is the second level of key, value pairs: the key is document id and value is the count this term appears in this document. The reason I am using dictionary to store index is that it support O(1) time complexity to locate a key.

1. **Storing and loading index:**

I used Json format to store index into files because Json format can preserve this two-level index structure and would not waste space. The store index code and load index code are illustrated below:



It has to be noted that if index file exist, my system would not build index again but just load existing index into memory for query.

1. **Boolean retrieval:**

Both Boolean Retrieval Model and VSM are based on the similarity between documents and the query sentence. For BRM, the similarity is higher if more term in query sentence appears in a document, in which the count of term is not important. However, in VSM, the count is taken into account when calculate similarity.

Based on the index, I implement my BRM in boolean\_query method in ‘task.py’ file.

1. **Computing TF-IDF:**

In my code, the TF of a term is composed by two part: TF in a specific document and TF in this query sentence. The IDF of a term is based on how many documents contains this term.

So, in my system, for a specific document and a specific query,

TF = Counts in document \* Counts in query;

IDF = log10(Total documents number / Number of documents contain this term)

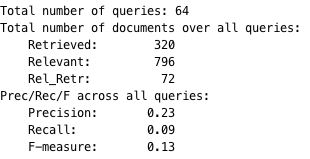
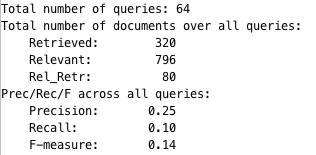
1. **Vector space model:**

The result of a query is based on the similarity between this query and the documents. In VSM, each document and query is regarded as a vector in a multi-dimension space, the similarity is calculating the cosine between two vectors. In my system, I am using TF-IDF as the criterion of similarity. The VSM is implemented in tfidf\_query method in ‘task.py’ file.

1. **Ranking and evaluation:**

My system will return the Top N result based on BRM or VSM, in which the N can be configured using command line when run the code.

The evaluation is based on the ‘eval\_ir.py’ file. And the evaluation results of the TOP 5 results generated from my code are shown below:

BSM evaluation result VSM evaluation result

We can see from the evaluation results that the VSM is much better than naïve BRM.