EECS 767

Xi Chen

Yang Tian

Yuanwei Wu

**Information Retrieval Project Phase 1 Report**

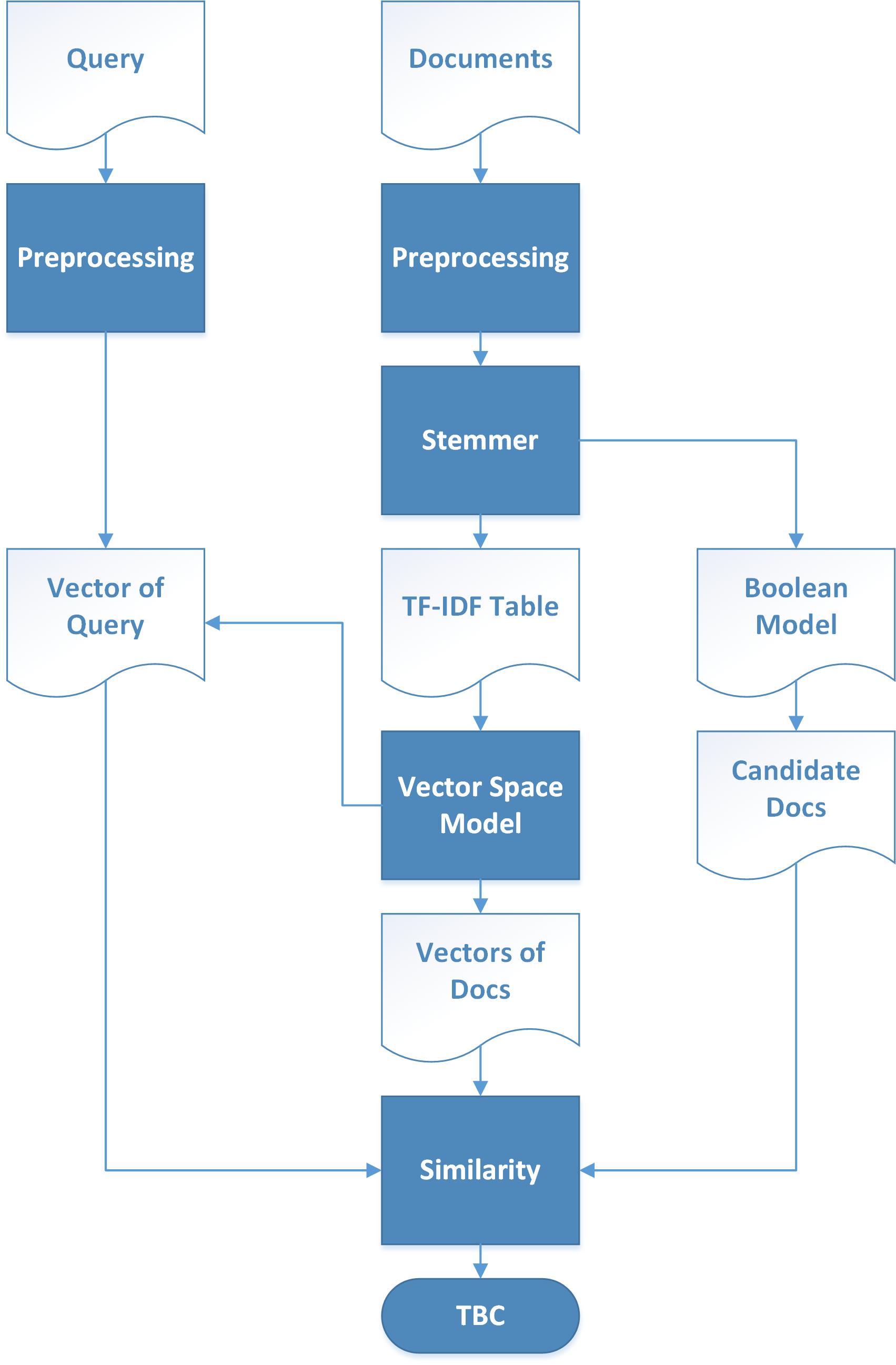
**Description**

In this project, we aimed to build an information retrieve system, which functioned as a search engine. The search engine should include the following components:

1. Document processing and indexing: This step includes removing HTML tags, converting all letters into lower case, removing stop list and stemming for pre-processing documents and query. And last, build an inverted index (including dictionary and posting lists) for documents.
2. Build a vector space model.
3. Niche crawler: The crawler is required to identify a domain of interest and should contain at least three components: a) a multi-threaded spider that fetches and parses webpages; b) the URL frontier which stores to-be-crawled URLs; and c) the URL repository that stores crawled URLs.
4. a) Feed the collected documents to the search engine; b) implement a Web-based interface to take user queries and return answers.
5. Add term proximity into the scoring mechanism.
6. Relevance feedback or Search personalization.

**Basic Architecture**

In phase 1, we already finished the implementation of document process, inverted index and vector space model (Figure 1).



Figure

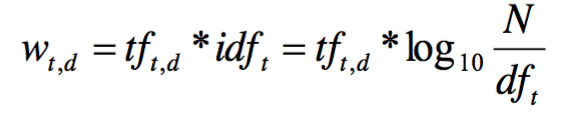
**Methods and Materials**

The documents we used for building our dictionary are 91 HTML documents provided on KU blackboard. And we picked Python as our programming language for building retrieval engines.

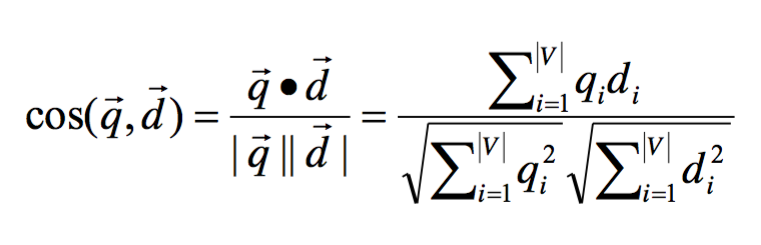
In pre-processing, we first removed all html tags and JavaScript parts from all html pages. Then we used NLTK package to remove stop list and do Porter stemming. After that, we built our inverted index dictionary with listing related DF and TF information. Meanwhile, we implemented another function to make sure that the user-inputted query will be pre-processed the same way as documents.

For building our vector space model, our implemented steps are as following:

1. Our model will calculate the inverse document frequency (idf) for each term in our dictionary.
2. By using the tf information in inverted index and idf got from the previous step, our model will calculate the tf–idf weight for each term in different documents (by the formula below) and provides the tf-idf documents vectors.



Our model will calculate the length for each vector. For query, once our users input their query, our model will calculate the vector length for query by the same method. And then our model will use a Boolean model algorithm to select the query related documents as candidate documents (the candidate should contain at least one term in the query). Last, our model will calculate the similarity between query and each candidate document and sort the documents by similarity as output. We used following formula for similarity calculating:



**Conclusion**

In phase 1, we have successfully built the base of our search engine. Our model already has a simple searching function. Afterward, we will add more functions as Niche crawler, term proximity, relevance feedback and etc. on our model.

For searching performance, we planned to test it after complete Niche crawler step.