Project 1
Let's Build a Search Engine

GOAL	Put into practice the different IR strategies related to the search process
DEADLINE	Monday, October 18th (end of day)
TOTAL POINTS	175 points
SUBMISSION INFORMATION	This is a ***team assignment***. Therefore, it should be completed by a team of at least 2 members--at most your team can be comprised of 4 members. *Only 1 submission per team*. This time, in addition to the report you'll need to do a pass off. We can schedule that remotely, or we can schedule a visit to my office. Not all team members need to be present for passing off.
HINT	Take advantage of the different scripts you have developed to complete your homework assignments so far 😉 You will use most of what you implement for this project also in Project 2, so keep that in mind as you make set up/development choices in order to make the code extensible and modular.

As described in the 2013 book, Search Engine Society , people "*…growing up in the 21st century have only ever known a world in which search engines could be queried, and almost always provide some kind of an answer, even if it may not be the best one*"[1]. It is clear we know how to use search engines (**SE**) and how they work, but given everything we have covered in class, can we build one?

Learning Objective

Demonstrate your understanding of core IR concepts along with your ability to translate IR theory into practice. Develop and apply techniques required to handle large text collections.

Some Guidelines for Success

This is not a trivial project that can be completed in time if you start working on it a few days before it is due. To avoid panicking, below are some suggestions that can help:

- Meet with your team *early* to discuss foundational aspects: programming language and interface.
- Work on each of the SE functionalities outlined for the project *in order*, i.e., as we cover them in class. For example, we have discussed (and worked on) the practical aspects of text processing, which is why you can already start implementing and testing the corresponding functionality.
- For each SE functionality, start with a *simple baseline* to make sure it can be properly integrated into your SE. Once that is working, improve the initial implementation so that it responds to project requirements. For example, for ranking you can simply start with binary weights (i.e., whether a query term is on a document or not) and then build up the use of the TF-IDF weighting scheme.
- Write your report as you go along (at least the basic ideas), otherwise there is a chance you will forget the reasons why you made certain design decisions.

[1] Halavais, A. (2013). *Search engine society*. John Wiley & Sons.

Detailed Specifications

In groups of ***up to 4 students***, you will combine methodologies we have discussed in class and use them as a foundation to build your own general-purpose SE. In fulfilling each of the requirements below, you can use any IR framework and/or programming language, in addition to any available libraries. You will use a number of data sources in completing this Project. A brief description of each of these sources, as well as the URLs to download them can be found under the heading Resources (page 3).

User Interface. You will need to account for a user interface, so that it is possible for any user (in this case, the instructor) to interact with your SE. There are not restrictions for the user interface: from a simple command-line interface that runs on your own laptop to an elaborate web application. The choice is yours, as long as the interface displays query suggestions, ranked lists of results, and allows users to submit queries. Leverage what you have learned in other classes, and remember that you will be graded on functionality not how pretty it looks.

Text Processing, Stopwords, and Stemming/Lemmatizing. Using a document collection **DC**, you are required to

a. Implement and run a word tokenizer on DC. The tokenizer should remove at least capitalization and punctuation symbols.
b. Remove stopwords from the documents in DC. (Here you can use a traditional/default stopword list or you can augment a traditional one based on stopwords identified via for example Zipf's law analysis of DC.)
c. Normalize the non-stopwords in the documents in DC to their canonical form (i.e., stem or lemma your choice).
d. Create an **index** structure for DC, which should include for each stem/lemma *t* in DC (i) the documents (identified by their IDs) in which *t* appears and (ii) the frequency of occurrence of *t* in each document.

Suggesting queries. Taking advantage of a query log **QL**, generate suggestions that can guide query formulation. Suggestions should be triggered at a term level by a space, i.e., signal that a given query term is completed.

QL has the following format <ID, query, time-stamp>; one instance per line. To re-create the entire log, you must aggregate the 5 files you are given.

In generating suggestions:

a. Identify possible candidates: distinct queries in QL that include the term (or terms) triggering the suggestions. Note that given *t* words, the length of a candidate query suggestion **CQ** should be *t+1+n* (n≥0). For example, for the query "Sven" (*t*=1) a candidate query can be "Sven character Frozen" (*t*=3). However, for the query "Olaf and Sven from Frozen" (*t*=5) a candidate query such as "Frozen" (*t*=1) is not valid.
b. Rank each candidate query suggestion CQ given the following equation:

$$Score(CQ, q') = \frac{\#\ of\ sessions\ in\ which\ q'is\ modified\ to\ CQ}{\#\ of\ sessions\ in\ which\ q'appears}$$

where *q'* is the n-gram triggering the suggestions. The higher $Score(CQ, q')$ is, the more likely CQ is the best suggestion for q'.

Identifying candidate resources. Given a query *q*, create a set of documents **CR** comprised of all the documents in DC that contain each of the terms in *q*. If there are less than 50 documents, then you will need to consider resources in DC that contain n-1 terms in q (all the combinations).

Remember that you must perform stopword removal and stemming/lemmatization on *q* prior to processing it; i.e., the same steps applied to create index terms must be applied to each query.

Relevance Ranking. Compute for each resource (i.e., document) in CR its corresponding relevance score with respect to *q*. For that you will use the following strategy:

$$RelevanceScore(q, d) = \sum_{w \in q} TF\ (w, d) \times IDF(w)$$

$$TF(w, d) = \frac{freq(w, d)}{max_d}$$

$$IDF(w) = log_2 \frac{N}{n_w}$$

where *q* is the query, *d* is a document in CR, *w* is a non-stop, stemmed/lemmatized word in *q*, *freq(w, d)* is the number of times *w* appears in *d*, max_d is the number of times the most frequently-occurred term appears in *d* (which is *constant* for each document), *N* is the number of documents in DC, and n_w is the number of documents in DC in which *w* appears at least once.

Generating Snippets. For each selected result, create the corresponding snippet. The snippet of each (top-ranked) document d should include:

a. The title of *d*
b. The two sentences in *d* that have the highest *cosine similarity* with respect to *q*; with TF-IDF as the term weighting scheme.

Examining SE Performance. Run each of the queries in *TestSet,* using your SE. In response to each query you should identify the top-5 most relevant documents, generate the corresponding snippets, and display them in ranked order (see Table 1 for sample expected output).

Comparison. Run each of the queries in the *TestSet* directly on the Wikipedia search module. For each query, list the top-5 results retrieved along with their titles, which will serve as snippets (see Table 3 for sample expected output).

Resources

Document Collection DC

- Wikipedia Sample You will need to download a sample containing Wikipedia pages. To each processing, you are provided with a json file containing ID, title, and content for each of the pages. This is a large collection, and while you will not have to clean headers or other structural tags, you

do need to look into special characters, e.g., "\n" or "\r", and data structures that can support handling such amount of text.

In processing this collection, *consider the vocabulary size for your index*. Indeed, you can reduce (to a degree) your vocabulary via text processing and other strategies we have covered in class. Having said that, you will need to justify your design decisions in your project report, so simply choosing an arbitrary threshold for vocabulary size is not valid.

Query Log QL

- AOL query log. You will need to download the provided zip file and then re-create the log by aggregating the 5 individual text files. The query log includes a number of query sessions, each of which captures a period of sustained use activities on the AOL search engine. Each session differs in length and includes (i) user ID, (ii) query text, (iii) data and time of the search, and (iv) clicked resources (optional).

Queries *TestSet*

- Create a set of 5 queries of your choice, of varied length (at least a unigram, a bigram, and a trigram). This is the set of queries that you will use for evaluation purposes.

Others

- Some basic stop-word lists you can use:
  - https://gist.github.com/sebleier/554280
  - http://www.lemurproject.org/stopwords/stoplist.dft
  - https://github.com/stanfordnlp/CoreNLP/blob/master/data/edu/stanford/nlp/patterns/surface/stopwords.txt
  - https://www.ranks.nl/stopwords
  - Simple stopword list
- Simple stopword removal (java)
- Porter Stemmer (java)
- Remember the lecture notes discussing other resources you have at your disposal that you can leverage in the completion of this project.

Grading Criteria

- Project Set-Up (20 Points)
  - By **September 28th** your team needs to email to the instructor to let her know (i) who is part of the team, (ii) how does the team plan to work collaboratively, (iii) a link to project repository, and (iv) a link to URL from which final project can be accessed or pertinent details if the team decides to run the project from a local machine and therefore pass-off will need to be done by visiting the instructor by project deadline.

- Project Report (125 Points)
  - (20 Points). Details on implementation decisions pertaining to text processing (which tokenizer you used, any modifications make, stemmer/lemmatizer used, etc.).
  - (20 Points). Statistics on DC (number of documents, index size pre/post stemming/lemmatizing, stop word removal, etc.)

- (15 Points). Details on implementation decisions pertaining to query suggestions (which data structures did you used to ease the retrieval of suggestions, etc.).
- (20 Points). Details on implementation decisions pertaining to relevance ranking (data structures that eased the process, libraries used, etc.).
- (20 Points). Details on implementation decisions pertaining snippet generation (data structures that eased the process, libraries used, etc.).
- (30 Points). Discussion. You need to focus your discussion on two main areas *result analysis* and i*nsights/lessons learned*. For the discussion of the generated results, you should include your 5 test-queries, a sentence discussing the intent (i.e., expected information), outcomes of your searches using your SE (as specified on Tables 1 and 2) and Wikipedia search module (as specified on Table 3), your thoughts on the results retrieved by your SE, and an overall performance analysis based on the resources retrieved (or lack thereof) by Wikipedia with respect to your SE. For your overall discussion, you should describe insights on the project (challenges and solutions).

- Pass off:
  - (30 points) Upon reading the report, the instructor will run 5 test queries on-the-fly (either from the link provided in the pre-proposal or during pass-off)

Query:			
Ranking	***Document ID***	***Snippet***	***Ranking Score***
1			
2			
3			

Table 1. Expected information for the top-5 resources retrieved from DC in response to each query in *TestSet*.

Query:		
Ranking	***Candidate Suggestion***	***Score***
1		
2		
3		

Table 2. Expected information for the top-3 candidate suggestions that would be offered for each query in *TestSet* if the first query term were to be used to trigger the query suggestions.

Query:	
Ranking	***Document Title***
1	
2	
3	

Table 3. Expected information for the top-5 resources retrieved from Wikipedia in response to each query in *TestSet*.