**CSCI-GA 2580: Web Search Engines** Fall 2016

# Homework 1

*Due October 17, 2016, 5pm EST* Department of Computer Science, NYU

## 1 Overview

In this homework assignment, you will be implementing an infrastructure for the design and evaluation of core ranking methods. We will provide you with skeleton code which you will then augment with additional functionality.

As we mentioned in class, a search engine can be broken down into several components, depicted in Figure 1. In this assignment, you will be modifying our code for the front end, the ranker, and the evaluator. We will provide a very simple index which you will be modifying in Homework 2.

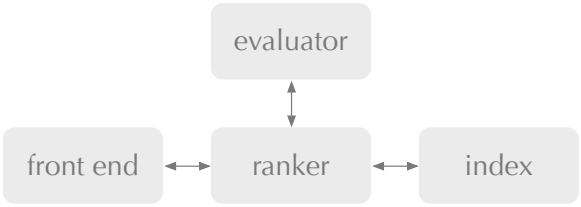


Figure 1: System architecture.

All code and data can be found here,

```
/home/congyu/cs2580/hw1/instructor
```

### 1.1 Front End

We will provide access to our search engine through a web server similar to the echo server you implemented in Homework 0. Our search engine is composed of two classes, `SearchEngine`, both the front end server and the driver for `Indexer`, and `QueryHandler`, the query processor, which invokes the `Ranker`. The front end server listens for connections from clients and, for each connection, the query handler parses the arguments in the URI for retrieval and ranking:

| attribute | value | description |
|---|---|---|
| query | string | user query |
| ranker | {fullscan, cosine, ql, phrase, numviews, linear} | ranking algorithm |
| format | {html, text} | output format |

That is, your request should have the form:

```
http://<HOST>:<PORT>/search?query=<QUERY>&ranker=<RANKER-TYPE>&format=<FORMAT-TYPE>
```

where any arguments should be URI-encoded. We described the ranker types in Lecture 2. The text output format should be of the form:

```
QUERY<TAB>DOCUMENTID-1<TAB>TITLE<TAB>SCORE
QUERY<TAB>DOCUMENTID-2<TAB>TITLE<TAB>SCORE
```

```
QUERY<TAB>DOCUMENTID-3<TAB>TITLE<TAB>SCORE
...
QUERY<TAB>DOCUMENTID-N<TAB>TITLE<TAB>SCORE
```

You should score *all* documents in the collection for each query. The documents should be sorted in *decreasing* order of relevance.

You are free to define your own HTML format.

In order to compile the search engine, switch to the parent directory of `src` and compile the code:

```
$ javac src/edu/nyu/cs/cs2580/*.java
```

In order to run the search engine, you need to index the corpus first, and then start the server (note that the path to your corpus and index are all specified inside `conf/engine.conf`):

```
$ java -cp src edu.nyu.cs.cs2580.SearchEngine --mode=index --options=conf/engine.conf
$ java -cp src -Xmx256m edu.nyu.cs.cs2580.SearchEngine --mode=serve --port=258XX \
    --options=conf/engine.conf
```

where `XX` is your group number.

## 1.2 Ranker

The ranker accepts queries from the front end and returns a ranked list of scored documents. In this assignment, we will be scoring the entire collection. Because our corpus is small, we can afford to do this. In Homework 2, we will implement methods for quickly scoring larger collections.

We will be iterating over each document id in the collection and score the document. Our example `RankerFullScan` code demonstrates how to iterate over our collection:

```
public Vector<ScoredDocument> runQuery(Query query, int numResults) {
  Vector<ScoredDocument> all = new Vector<ScoredDocument>();
  for (int i = 0; i < _indexer.numDocs(); ++i) {
    all.add(scoreDocument(query, i));
  }
  Collections.sort(all, Collections.reverseOrder());
  Vector<ScoredDocument> results = new Vector<ScoredDocument>();
  for (int i = 0; i < all.size() && i < numResults; ++i) {
    results.add(all.get(i));
  }
  return results;
}
```

You will be implementing the corresponding `runQuery` for your own Rankers, based on the information provided via either `Document` or `DocumentFull` classes:

- `Vector<String> getConvertedBodyTokens()`: returns a vector of tokens in the document body.
- `int getNumViews()`: returns the number of times the document was viewed in the last hour, a statistic number provided as part of the corpus, i.e., don't worry about how they are generated.

The returned `ScoredDocument` includes all of the relevant information about a search result:

- document identifier: a unique index for each document

- score: the retrieval score
- title: the title of the document

### 1.3 Index

Having a fast and reliable index is one of the prerequisites for a real web search engine. In this homework, we are not concerned with speed and will be scoring every document in the collection for every query. It should not be necessary to inspect the `Indexer` and you should be able to perform all scoring by inspecting the `Document` and using the following methods:

- `int numDocs()`: returns the number of documents in the corpus.
- `int corpusDocFrequencyByTerm (String term)`: returns the number of documents `s` occurs in.
- `int corpusTermFrequency(String term)`: returns the number of occurrences of `s` in the entire collection.
- `long totalTermFrequency()`: returns the total number of words occurrences in the collection (i.e. the sum of termFrequency(s) over all words in the vocabulary).

*You do not need to perform any string normalization in this assignment. You will address stemming and stopword removal in Homework 2.*

### 1.4 Evaluator

As we mentioned in Lecture 1, evaluation is a fundamental part of search engine design. Our evaluator will consume the system scoring from standard input using the `format=text` option. An evaluator also requires a set of judged labels for (query, document) pairs, which will be passed in as an argument to the execution.

In order to evaluate a system for a query, you would call the evaluator on the output of the retrieval:

```
$ curl "http://<HOST>:<PORT>/search?query=<QUERY>&ranker=<RANKER-TYPE>&format=text" | \
  java -cp src edu.nyu.cs.cs2580.Evaluator data/labels.tsv
```

Because we only evaluate performance on judged queries, `<QUERY>` *must* appear in the judgment file.

### 1.5 Submission

You should copy all code and data into a directory and submit assignments using the same procedure as Homework 0. Your submission directory should have three subdirectories:

- `conf`: Configuration for the search engine.
- `data`: Various data needed, which should be identical to the instructor data directory. The index should be automatically generated and should not be uploaded as part of your solution.
- `src`: Source code for your solution.
- `results`: including the results for questions 2.1, 2.2, and 2.3 (and 2.4 if you decide to do it). Your code **must** be able to generate the files in this directory.

# 2 Questions

## 2.1 Ranking Signals (40 points)

Compute the following document representations:

- $L_2$-normalized tf.idf vectors
- Language model probabilities with Jelinek-Mercer smoothing (7.3.1 in Croft *et al.*).

Implement the following bag of words ranking signals:

- cosine similarity (vector space model)
- query likelihood (language model)

In addition, implement the following ranking signals:

- a 'phrase' ranker defined as the number of query bigrams present in the document. You do not have to worry about weighting the bigrams, only counting the number of matches. If there is only one term in the query, this signal reverts back to counting the number of the (single) query term occurrences in the document.
- a 'numviews' ranker based on the number of views of the documents. This signal is query independent.

*Please use only the term vectors from the document body for all signals except numviews.*

Using the result `text` format from Section 1.1 and the `queries.tsv` in Appendix A.2, generate one results file for each of the following ranking methods:

- `hw1.1-vsm.tsv`: vector space model.
- `hw1.1-ql.tsv`: query likelihood with Jelinek-Mercer smoothing (you can set $\lambda = 0.50$).
- `hw1.1-phrase.tsv`: phrase-based ranker.
- `hw1.1-numviews.tsv`: numviews-based ranker.

Your code **must** be able to generate these files based on instructions you provide in a readme file.

## 2.2 Simple Linear Model (20 points)

Implement a simple linear ranking model,

$$y_{d,q} = \beta_{\cos}\cos(d,q) + \beta_{\text{LM}}p(q|\theta_d) + \beta_{\text{phrase}}\text{phrase}(d,q) + \beta_{\text{numviews}}\text{numviews}(d)$$

Using the result `text` format from Section 1.1 and the `queries.tsv` in Appendix A.2, generate one results file for the linear ranker and place it in this file:

- `hw1.2-linear.tsv`: linear ranker

Your code **must** be able to generate this file based on instructions you provide in a readme file.

### 2.3 Evaluation (40 points)

Using the `Evaluator.java` class as a skeleton, implement the following metrics,

- $Metric_0$: Precision at 1, 5, and 10
- $Metric_1$: Recall at 1, 5, and 10
- $Metric_2$: $F_{0.50}$ at 1, 5, and 10
- $Metric_3$: Precision at recall points $\{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$
- $Metric_4$: Average precision
- $Metric_5$: NDCG at 1, 5, and 10 (using the gain values presented in Lecture 2)
- $Metric_6$: Reciprocal rank

For binary evaluation metrics, you can assume that {Perfect, Excellent, Good} grades indicate relevance and {Fair, Bad} grades indicate non-relevance. You can also assume that any un-judged documents receive a grade of 'Bad'. The example code does **not** implement any rank-based metrics.

The output of the evaluator should have the following format,

```
<QUERY><TAB><Metric_0>...<TAB><Metric_n>
```

where `Metric_i` includes all of the metrics we are asking for. Please please them in the order above (i.e. Precision@1,Precision@5,Precision@10,Recall@1,...).

For each ranking method in questions 1 and 2, submit the evaluation metrics for each query in `queries.tsv` in Appendix A.2. You should use `labels.tsv` in Appendix A.2 for labeled data. Place the evaluation results in one file for each ranker:

- `hw1.3-vsm.tsv`
- `hw1.3-ql.tsv`
- `hw1.3-phrase.tsv`
- `hw1.3-numviews.tsv`
- `hw1.3-linear.tsv`

Your code **must** be able to generate these files.

# A Data

## A.1 Corpus

We will be using a pre-processed, sampled version of Wikipedia to feed our indexer. This file is located here:

```
/home/congyu/cs2580/hw1/instructor/data/corpus.tsv
```

You *should not* change the file at all in this homework assignment.

## A.2 Relevance Judgments

We will be providing a set of relevance judgements for a small set of queries. This file is located here:

```
/home/congyu/cs2580/hw1/instructor/data/labels.tsv
```

The format of the file is:

```
<QUERY-1><TAB><DOCUMENTID-1><TAB><RELEVANCE-GRADE>
<QUERY-1><TAB><DOCUMENTID-2><TAB><RELEVANCE-GRADE>
...
<QUERY-1><TAB><DOCUMENTID-N><TAB><RELEVANCE-GRADE>
<QUERY-2><TAB><DOCUMENTID-1><TAB><RELEVANCE-GRADE>
<QUERY-2><TAB><DOCUMENTID-2><TAB><RELEVANCE-GRADE>
...
<QUERY-2><TAB><DOCUMENTID-N><TAB><RELEVANCE-GRADE>
```

Relevance grades are on a five-point scale of {Perfect, Excellent, Good, Fair, Bad}. You can find the queries associated with these judgments here,

```
/home/congyu/cs2580/hw1/instructor/data/queries.tsv
```

You *should not* change the file at all in this homework assignment.