Mahidol University

ITCS 414 – Information Retrieval and Storage

Project 2

MuGle: One Step Closer to Google

Deadline: Monday October 15, 2018 11:55PM

Update Logs

2018-09-17 Initial Version

Learning Objectives:

After the completion of this project, you are expected to:

1. Be able to implement a simple ranking based information retrieval system using Jaccard similarity.
2. Be able to implement a simple ranking based information retrieval system using Cosine similarity of TF-IDF weighted document vectors.
3. Be able to compare information retrieval systems using standard IR evaluation metrics such as precision, recall, and F1.
4. Be able to scientifically reason and analyze experimental results from comparing the two IR systems.

Overview:

In P01, you had hands-on experience implementing a simple search engine using Boolean conjunctive retrieval, where a document is a ***hit*** if it contains all the terms in the query, or a ***miss*** otherwise. Major drawbacks of such a system include:

1. [NO-RANKING] All the ***hit*** documents are the same. There is no mechanism to tell which document is more *relevant* to the search query than the others.
2. [NO-PARTIAL-MATCHING] If one term in the search query does not appear in a document, then the document is automatically a ***miss***, and will not appear in the search results. The ability to retrieve partially matched documents could prove to be crucial for effective search engines.

To mitigate such problems, ranking-based IR systems have been proposed, where a document is given a numeric relevance score that represents how well the document matches with the information need, represented by the search query. A trivial method to quantify the relevance score may implement the similarity between the document and the search query, using traditional similarity measures such as Jaccard and Cosine similarity. Assigning a numeric relevance score to each of the documents allows them to be ranked; then, the top ***k*** documents can be returned as search results.

In this project, you will have practical experience on not only implementing such ranking-based document systems, but also evaluating them, using scientifically proven IR evaluation protocols. Specifically, to evaluate the effectiveness of a search system, ground truth datasets and standard IR evaluation metrics, including precision, recall, and F1 will be used.

Starter Code

Download the Java starter code and dataset from MyCourses. You can use ant or Eclipse to build the code.

Corpora

You will be working with a dataset modified from LISA (Library and Information Science Abstracts) project[1]. The dataset is included as part of the package in directory "./data/lisa/". In the lisa directory, you will find the following files:

- **documents.txt**: This file contains all the documents in the corpus, one line per one document. Each line in the file has the following format: `<docID><TAB><text>`

  Where `<docID>` is the document ID, `<TAB>` is a tab character, and `<text>` is the raw textual content of the document.

- **queries.txt**: This file contains a set of test search queries, one line per one query. Each line has the following format: `<qID><TAB><text>`

  Where `<qID>` is the query ID, `<TAB>` is a tab character, and `<text>` is the raw textual content of the query.

- **relevance.txt**: This file contains the list of relevant documents (justified by human experts) corresponding to each query in query.txt. Each line has the following format:

  `<qID><TAB><docID List>`

  Where `<qID>` is the query ID, `<TAB>` is a tab character, and `<docID List>` is the list of relevant document IDs, separated by a space character.

Task 1 Implementing Jaccard and TFIDF Searchers

To facilitate your implementation, some codes have been pre-implemented for you to use.

`Searcher` (in Searcher.java) is an abstract class containing some default mechanisms such as:

`public static List<String> tokenize(String rawText)`: Takes a raw string as input, preprocesses, tokenizes, and outputs individual tokens as a list. **You must use this method to convert a raw string to tokens** (for consistency with the auto-grader's results).

`public static List<Document> parseDocumentFromFile(String filename)`: The input is a document (or query) filename. This method parses the file and outputs a list of Document objects.

`public static void displaySearchResults(List<SearchResult> results)`: Displays search results in a (more) beautiful format.

`abstract public List<SearchResult> search(String queryString, int k);` This is an ***abstract method*** for other extended subclasses to implement. It takes a raw string query, and ***k*** as input, and outputs the ranked list of ***k*** top search results.

[1] http://ir.dcs.gla.ac.uk/resources/test_collections/lisa/

Do not modify Searcher.java.

`Document` (in Document.java): This class serves as a containing object for a document (and a query). A document is a tuple of document ID, raw text, and preprocessed tokens. Please refer to the implementation for more detail. Do not modify Document.java.

`SearchResult` (in SearchResult.java): This class serves as a containing object for a search result. A search result is a tuple of Document object and its relevance score. Please refer to the implementation for more detail. Do not modify SearchResult.java.

Your tasks are simply to implement `JaccardSearcher` and `TFIDFSearcher` by extending `Searcher`, and the remaining part of `SearcherEvaluator`. Let $D = \{d_1, d_2, \dots\}$ be the set of documents in the corpus, $V = \{t_1, t_2, \dots\}$ be the vocabulary containing all the distinct terms in D. A document $d = \langle t_1, t_2, \dots \rangle \in D$ is an ordered sequence of tokens, where each $t_i \in V$. A search query $q = \langle t_1, t_2, \dots \rangle$ is an ordered sequence of terms, some of which are not in V. Let the operation *T(x)*, where *x* is either a document or a query, return a unique set of terms in *x*, namely $T(x) = \{t: t \in x\}$. The Jaccard and TFIDF based relevance scoring functions are defined as follows.

$$Score_{Jaccard}(q,d) = \begin{cases} \dfrac{|T(q) \cap T(d)|}{|T(q) \cup T(d)|} & , T(q) \neq \emptyset \; or \; T(d) \neq \emptyset \\ 0 & . \quad otherwise \end{cases}$$

$$Score_{TFIDF}(q,d,D) = Cosine(\vec{q}, \vec{d})$$

Where $Cosine(\vec{q}, \vec{d})$ is the Cosine similarity between the query vector $\vec{q}$ and document vector $\vec{d}$. A document/query vector is an ordered tuple of the weight of each term in *V* defined as:

$$\vec{q} = \langle w(t,q,D): t \in V \rangle; \; \vec{d} = \langle w(t,d,D): t \in V \rangle$$

Where *w(t,x,D)* is the TF-IDF weight of the term $t \in x$, defined as:

$$w(t,x,D) = tf(t,x) \cdot idf(t,x,D)$$

$$tf(t,x) = \begin{cases} 1 + \log_{10} Freq(t,x), & t \in x \\ 0 & , \quad otherwise \end{cases}$$

Where *Freq(t,x)* is the number of occurrences of the term *t* in *x*.

$$idf(t,D) = \log_{10}\left(1 + \frac{|D|}{df(t,D)}\right)$$

Where $|D|$ is the number of documents in the corpus, and *df(t,D)* is the number of documents in *D* that contain the term *t*. That is $df(t,D) = |\{d: t \in d, d \in D\}|$.

In the method `search(String queryString, int k)`, given a string query "`queryString`" and the number "k", it first tokenizes the query, calculating the relevance score (use $Score_{Jaccard}$ in `JaccardSearcher` and $Score_{TFIDF}$ in `TFIDFSearcher`) between the query and each document in the corpus, create a ranked list of `SearchResult` objects, and return the top `k` search results with the highest relevance score, breaking tie with document IDs (smaller IDs come first).

Executing `testJaccardSearcher()` in `StudentTester` should give the following output.

```
@@@ Testing Jaccard-based documents searcher on ./data/lisa
@@@ Finished loading 6004 documents from ./data/lisa/documents.txt
@@@ Results:
<1>[score=0.0][ID:1, THE INDIAN COUNCIL OF LIBRARY AND INFORMATION SERV...]
<2>[score=0.0][ID:2, THE LINGERING FRAGRANCE: PROCEEDINGS OF THE XXIV A...]
<3>[score=0.0][ID:3, XXV ALL INDIA LIBRARY CONFERENCE TRIVANDRUM 14-18 ...]
<4>[score=0.0][ID:4, MALAWI LIBRARY ASSOCIATION SECOND ANNUAL GENERAL M...]
<5>[score=0.0][ID:5, A PERSONAL VIEW OF THE ZAMBIA LIBRARY SERVICE. PER...]
<6>[score=0.0][ID:6, CLA 80: TURNING ON THE POWER. REPORT OF PROCEEDING...]
<7>[score=0.0][ID:7, UNCERTAIN BEGINNINGS. REPORT OF THE AMERICAN LIBRA...]
<8>[score=0.0][ID:8, MIDWINTER IN REAGAN'S WASHINGTON: AN ALA CONFERENC...]
<9>[score=0.0][ID:9, THE WHITE HOUSE CONFERENCE ON LIBRARY AND INFORMAT...]
<10>[score=0.0][ID:10, INFORMATION: BOOKS ARE JUST THE BEGINNING. THE MIC...]


@@@ Results: Information Retrieval
<1>[score=0.15384615384615385][ID:398, ONLINE INFORMATION RETRIEVAL BIBLIOGRAPHY. FOURTH ...]
<2>[score=0.13333333333333333][ID:2412, CONSIDERATIONS OF ON-LINE INFORMATION RETRIEVAL IN...]
<3>[score=0.125][ID:927, DATA BASE FOR SEPARATIONS CHEMISTRY. A DATA BASE I...]
<4>[score=0.125][ID:1789, A DATA SYSTEM ON MICROFICHE. PRESENTS AN INFORMATI...]
<5>[score=0.125][ID:1886, HOLOGRAPHIC INFORMATION MEDIA. DISCUSSES THE POSSI...]
<6>[score=0.125][ID:4353, THEORIES AND MODELS IN INFORMATION RETRIEVAL. TRAN...]
<7>[score=0.11764705882352941][ID:5096, MICROFORMS' THE TREND OF APPLICATIONS (IN CHINESE)...]
<8>[score=0.1111111111111111][ID:4275, CONSEQUENCES OF INFORMATION TECHNOLOGY. 5 CLEARLY ...]
<9>[score=0.1111111111111111][ID:5918, THE ABC'S OF THE AWLN (ADVANCED WISSWESSER LINE NO...]
<10>[score=0.10526315789473684][ID:3516, WILL LIBRARIANS BE NEEDED?. STARTING FROM F.W. LAN...]


@@@ Results: Machine Learning
<1>[score=0.06666666666666667][ID:489, ARTIFICIAL INTELLIGENCE: GENERAL. JANUARY, 1970-MA...]
<2>[score=0.06666666666666667][ID:5453, IT PAYS TO LEARN RUSSIAN. BRIEFLY DESCRIBES HOW SO...]
<3>[score=0.0625][ID:4277, AUTOMATION, MACHINE-READABLE RECORDS, AND ARCHIVAL...]
<4>[score=0.058823529411764705][ID:1789, A DATA SYSTEM ON MICROFICHE. PRESENTS AN INFORMATI...]
<5>[score=0.058823529411764705][ID:4543, GUIDELINES FOR TWO-YEAR COLLEGE LEARNING RESOURCES...]
<6>[score=0.058823529411764705][ID:5339, SERIALS CATALOGUING' UTLAS AND THE MACHINE ENVIRON...]
<7>[score=0.05555555555555555][ID:2872, THE FUTURE OF LIBRARY CATALOG' COMPUTER-SUPPORTED ...]
<8>[score=0.05555555555555555][ID:3948, A SURVEY OF MACHINE READABLE DATA BASES. 42 OF THE...]
<9>[score=0.05263157894736842][ID:1252, PROVISION OF ENERGY LITERATURE. DESCRIBES BOTH MAN...]
<10>[score=0.05][ID:106, RESOURCE SERVICES FOR CANADIAN SCHOOLS. HANDBOOK F...]


@@@ Results: Deep Learning
<1>[score=0.06666666666666667][ID:5453, IT PAYS TO LEARN RUSSIAN. BRIEFLY DESCRIBES HOW SO...]
<2>[score=0.058823529411764705][ID:4543, GUIDELINES FOR TWO-YEAR COLLEGE LEARNING RESOURCES...]
<3>[score=0.05][ID:106, RESOURCE SERVICES FOR CANADIAN SCHOOLS. HANDBOOK F...]
<4>[score=0.05][ID:2103, REMEDIATION AND REINFORCEMENT' BOOKS FOR CHILDREN ...]
<5>[score=0.05][ID:4544, GUIDELINES FOR TWO-YEAR COLLEGE LEARNING RESOURCES...]
<6>[score=0.045454545454545456][ID:3997, PSYCHOLINGUISTICS. 1964-NOVEMBER, 1980 (CITATIONS ...]
<7>[score=0.045454545454545456][ID:5969, PSYCHOLINGUISTICS. 1964-APRIL, 1982 (CITATIONS FRO...]
<8>[score=0.041666666666666664][ID:1077, TOWARDS INDIVIDUALITY: THE SCHOOL LIBRARY?. EXAMIN...]
<9>[score=0.041666666666666664][ID:3172, LIBRARY SYSTEMS ANALYSIS 2' PLANNING TECHNIQUES. T...]
<10>[score=0.041666666666666664][ID:4850, BOOKBINDING AND CONSERVATION BY HAND' A WORKING GU...]


@@@ Results: I AM INTERESTED IN INFORMATION ON THE PROVISION OF...
<1>[score=0.2222222222222222][ID:5801, SELECTIVE DISSEMINATION OF INFORMATION IN ACADEMIC...]
<2>[score=0.21428571428571427][ID:770, MANUAL SDI IN SPECIAL LIBRARIES. DESCRIBES THE NEE...]
<3>[score=0.15384615384615385][ID:3605, INFORMATION IN BUSINESS' THE ROLE OF THE LIBRARY. ...]
<4>[score=0.14705882352941177][ID:4852, COPYRIGHT AND ACADEMIC LIBRARY COPYING. THE ASSOCI...]
<5>[score=0.14583333333333334][ID:3815, CHEMICAL ABSTRACTS 1AS A SOURCE FOR NEWLY PUBLISHE...]
<6>[score=0.14285714285714285][ID:723, A HOUSING INFORMATION SERVICE. DESCRIBES THE DEVEL...]
<7>[score=0.13725490196078433][ID:2939, FRENCH LEGAL AND ECONOMIC DATA BASES AND DATA BANK...]
<8>[score=0.1351351351351514][ID:1220, USE OF PATENT INFORMATION AT A CZECHOSLOVAK RESEAR...]
<9>[score=0.13333333333333333][ID:1833, SELECTION OF SECONDARY INFORMATION SERVICES. THE C...]
<10>[score=0.1282051282051282][ID:1544, PROFILES OF TWELVE MUSIC LIBRARIES AT ACADEMIC INS...]


@@@ Results: THE WHITE HOUSE CONFERENCE ON LIBRARY AND INFORMAT...
<1>[score=1.0][ID:9, THE WHITE HOUSE CONFERENCE ON LIBRARY AND INFORMAT...]
<2>[score=0.34782608695652173][ID:887, THE MANAGEMENT OF ONLINE REFERENCE SEARCH SERVICES...]
<3>[score=0.30434782608695654][ID:1587, PART-TIME STUDENTS: THEIR USE OF A POLYTECHNIC LIB...]
<4>[score=0.2727272727272727][ID:3739, NATIONAL COMMISSION ON LIBRARIES AND INFORMATION S...]
<5>[score=0.25][ID:3914, ONLINE SERVICE IN PUBLIC LIBRARY-THE LANCASHIRE EX...]
<6>[score=0.21212121212121213][ID:156, DIRECTORY OF TEXAS LIBRARY NETWORKS AND INFORMATIO...]
<7>[score=0.20833333333333334][ID:1388, THE EFFECT OF CLOSED CATALOGS ON PUBLIC ACCESS. FO...]
<8>[score=0.20588235294117646][ID:54, THE 1980 DIRECTORY OF LIBRARY SYSTEMS IN NEW YORK ...]
<9>[score=0.2][ID:754, DATA BASE MANAGEMENT. 1970-MARCH, 1980 (CITATIONS ...]
<10>[score=0.2][ID:755, DATA BASE MANAGEMENT. 1979-JUNE, 1981 (CITATIONS F...]


@@@ Total time used: 2503 milliseconds.
```

Executing `testTFIDFSearcher()` in `StudentTester` should give the following output.

```
@@@ Testing TFIDF-based documents searcher on ./data/lisa
@@@ Finished loading 6004 documents from ./data/lisa/documents.txt
@@@ Results:
<1>[score=NaN][ID:1, THE INDIAN COUNCIL OF LIBRARY AND INFORMATION SERV...]
<2>[score=NaN][ID:2, THE LINGERING FRAGRANCE: PROCEEDINGS OF THE XXIV A...]
<3>[score=NaN][ID:3, XXV ALL INDIA LIBRARY CONFERENCE TRIVANDRUM 14-18 ...]
<4>[score=NaN][ID:4, MALAWI LIBRARY ASSOCIATION SECOND ANNUAL GENERAL M...]
<5>[score=NaN][ID:5, A PERSONAL VIEW OF THE ZAMBIA LIBRARY SERVICE. PER...]
<6>[score=NaN][ID:6, CLA 80: TURNING ON THE POWER. REPORT OF PROCEEDING...]
<7>[score=NaN][ID:7, UNCERTAIN BEGINNINGS. REPORT OF THE AMERICAN LIBRA...]
<8>[score=NaN][ID:8, MIDWINTER IN REAGAN'S WASHINGTON: AN ALA CONFERENC...]
<9>[score=NaN][ID:9, THE WHITE HOUSE CONFERENCE ON LIBRARY AND INFORMAT...]
<10>[score=NaN][ID:10, INFORMATION: BOOKS ARE JUST THE BEGINNING. THE MIC...]
```

```
@@@ Results: Information Retrieval
<1>[score=0.3125671339273122][ID:1789, A DATA SYSTEM ON MICROFICHE. PRESENTS AN INFORMATI...]
<2>[score=0.296016619772261961[ID:2882, MODELLING AND EVALUATING EFFECTIVENESS OF AN INFOR...]
<3>[score=0.2843652042769549][ID:2412, CONSIDERATIONS OF ON-LINE INFORMATION RETRIEVAL IN...]
<4>[score=0.281720903116363][ID:2790, INTELLIGENT AUTOMATIC INFORMATION RETRIEVAL. CONSI...]
<5>[score=0.2775826974010459][ID:738, A COMPARATIVE STUDY OF DOCUMENT RETRIEVAL SYSTEMS ...]
<6>[score=0.2734256039604928][ID:385, SYSTEMS APPROACH TO DESIGN AND RETRIEVAL OF INFORM...]
<7>[score=0.2653174077863827][ID:3346, INFORMATION RETRIEVAL THEORY AND DESIGN BASED ON A...]
<8>[score=0.2649726621360938][ID:2789, COMPUTERIZED DOCUMENT RETRIEVAL JANUARY, 1975-SEPT...]
<9>[score=0.25242698999388924][ID:3388, INFORMATION RETRIEVAL RESEARCH. PAPERS GIVEN AT TH...]
<10>[score=0.25078309960235634][ID:3906, ON-LINE INFORMATION RETRIEVAL SYSTEMS. 1977-DECEMB...]


@@@ Results: Machine Learning
<1>[score=0.24325831090190458][ID:106, RESOURCE SERVICES FOR CANADIAN SCHOOLS. HANDBOOK F...]
<2>[score=0.21477479068741892][ID:3948, A SURVEY OF MACHINE READABLE DATA BASES. 42 OF THE...]
<3>[score=0.21332587371816134][ID:489, ARTIFICIAL INTELLIGENCE: GENERAL. JANUARY, 1970-MA...]
<4>[score=0.21283087266682124][ID:4543, GUIDELINES FOR TWO-YEAR COLLEGE LEARNING RESOURCES...]
<5>[score=0.20993591305360565][ID:4277, AUTOMATION, MACHINE-READABLE RECORDS, AND ARCHIVAL...]
<6>[score=0.20591007618460314][ID:3498, LEARNED SOCIETIES AND JOURNAL PUBLISHING. REPORTS ...]
<7>[score=0.20452415229046675][ID:955, MACHINE TRANSLATION 1964-MAY 1981 (CITATIONS FROM ...]
<8>[score=0.2030711025081421][ID:1297, ACADEMIC LIBRARY INSTRUCTION' THE USE OF FILMS; TH...]
<9>[score=0.20128807156026798][ID:2872, THE FUTURE OF LIBRARY CATALOG' COMPUTER-SUPPORTED ...]
<10>[score=0.200047537196914][ID:4544, GUIDELINES FOR TWO-YEAR COLLEGE LEARNING RESOURCES...]


@@@ Results: Deep Learning
<1>[score=0.2716624996515419][ID:2534, MYSTERIES OF THE DEEP' MODELS OF THE UNIVERSE OF K...]
<2>[score=0.2273242332316567][ID:587, CRISIS-CENTRED, ISSUE-BASED' THE LONELINESS OF DIS...]
<3>[score=0.1995864837145998][ID:4030, STOIC INFLUENCES IN LIBRARIANSHIP: A CRITIQUE. PAP...]
<4>[score=0.15523739027834302][ID:106, RESOURCE SERVICES FOR CANADIAN SCHOOLS. HANDBOOK F...]
<5>[score=0.13581985799771074][ID:4543, GUIDELINES FOR TWO-YEAR COLLEGE LEARNING RESOURCES...]
<6>[score=0.13140329200017606][ID:3498, LEARNED SOCIETIES AND JOURNAL PUBLISHING. REPORTS ...]
<7>[score=0.1295915764498677][ID:1297, ACADEMIC LIBRARY INSTRUCTION' THE USE OF FILMS; TH...]
<8>[score=0.1276620621549154][ID:4544, GUIDELINES FOR TWO-YEAR COLLEGE LEARNING RESOURCES...]
<9>[score=0.10844579115719923][ID:3109, EDUCATIONAL MEDIA AND LEARNING THEORIES. EDUCATION...]
<10>[score=0.10636880160544172][ID:2560, TEACHING AND LEARNING MATERIALS FOR INFORMATION TR...]


@@@ Results: I AM INTERESTED IN INFORMATION ON THE PROVISION OF...
<1>[score=0.4189957005521096][ID:770, MANUAL SDI IN SPECIAL LIBRARIES. DESCRIBES THE NEE...]
<2>[score=0.30574607499488365][ID:4253, INFORMATION USES IN SOCIAL SERVICES DEPARTMENTS. A...]
<3>[score=0.2882705310889179][ID:3605, INFORMATION IN BUSINESS' THE ROLE OF THE LIBRARY. ...]
<4>[score=0.2867902654957982][ID:2407, PROBLEMS IN IMPROVING CLASS IN CURRENT AWARENESS B...]
<5>[score=0.22245893003184442][ID:1310, SOME CHARACTERISTICS OF SDI SYSTEMS (EXPERIENCE OF...]
<6>[score=0.22160401280197214][ID:5801, SELECTIVE DISSEMINATION OF INFORMATION IN ACADEMIC...]
<7>[score=0.2197274361215126][ID:2319, AN APPROACH TO THE CLIENTS AND USES OF SELECTIVE D...]
<8>[score=0.21353839214877973][ID:3815, CHEMICAL ABSTRACTS 1AS A SOURCE FOR NEWLY PUBLISHE...]
<9>[score=0.2086530079559519][ID:769, GUIDE FOR THE ESTABLISHMENT AND EVALUATION OF SERV...]
<10>[score=0.1981942755615691][ID:1264, STRUCTURE AND FUNCTIONS OF AN IN-HOUSE INFORMATION...]


@@@ Results: THE WHITE HOUSE CONFERENCE ON LIBRARY AND INFORMAT...
<1>[score=1.0][ID:9, THE WHITE HOUSE CONFERENCE ON LIBRARY AND INFORMAT...]
<2>[score=0.3404349750430772][ID:754, DATA BASE MANAGEMENT. 1970-MARCH, 1980 (CITATIONS ...]
<3>[score=0.2666823061058415][ID:3739, NATIONAL COMMISSION ON LIBRARIES AND INFORMATION S...]
<4>[score=0.261966098299881][ID:887, THE MANAGEMENT OF ONLINE REFERENCE SEARCH SERVICES...]
<5>[score=0.2532399909375966][ID:1587, PART-TIME STUDENTS: THEIR USE OF A POLYTECHNIC LIB...]
<6>[score=0.22683074593918603][ID:10, INFORMATION: BOOKS ARE JUST THE BEGINNING. THE MIC...]
<7>[score=0.21523573232777146][ID:755, DATA BASE MANAGEMENT. 1979-JUNE, 1981 (CITATIONS F...]
<8>[score=0.21270412451667417][ID:36, IMPLICATIONS OF THE WHITE HOUSE CONFERENCE ON LIBR...]
<9>[score=0.21076697554492863][ID:1388, THE EFFECT OF CLOSED CATALOGS ON PUBLIC ACCESS. FO...]
<10>[score=0.21009843649848184][ID:1007, NEBRASKA PRE-WHITE HOUSE CONFERENCE ON LIBRARIES A...]


@@@ Total time used: 5826 milliseconds.
```

Task 2 Implementing Evaluation Framework

The Jaccard-based searcher treats a document (and a query) as a set of terms, where the relevance score is quantified by Jaccard similarity index between the document and the query. Such a similarity measure represents the normalized set-overlapness between the document and the query. On the other hand, the TF-IDF searcher treats a document (and a query) as a vector of term weights, where the relevance score is quantified by the cosine similarity between the two vectors, representing how close the two vectors are located in the hyperspace, in terms of their angle gap.

While implemented with different approaches, both the Jaccard and TF-IDF searchers have the same objective—to retrieve relevant documents given a search query, and rank them based on their relevance scores. A natural question would be: Which of the search algorithms works better than the other? To scientifically and statistically answer this non-trivial question, a quantitative evaluation on both the search algorithms must be performed. The performance statistics will help to make sound justification when comparing the two search algorithms, and further applying them on possible real-world applications.

Let $D = \{d_1, d_2, \dots\}$ be the set of documents in the corpus, $Q = \{q_1, q_2, \dots\}$ be the set of test queries. For each test query q, let R_q^S be the set of relevant documents retrieved by the searcher *S*, and G_q be the set of ground-truth relevant documents (justified by human experts). The query-wise precision, recall, and F1 of the searcher *S* are defined as follows:

$$Precision_q^S = \frac{\left|R_q^S \cap G_q\right|}{\left|R_q^S\right|}$$

$$Recall_q^S = \frac{\left|R_q^S \cap G_q\right|}{\left|G_q\right|}$$

$$F1_q^S = \frac{2 \cdot Precision_q^S \cdot Recall_q^S}{Precision_q^S + Recall_q^S}$$

In order to compute the overall performance of a search engine, given a set of test query *Q*, you can simply take the average of the precision, recall, and F1 across $\forall q \in Q$. Mathematically,

$$AvgPrecision_Q^S = \frac{1}{|Q|}\sum_{q \in Q} Precision_q^S$$

$$AvgRecall_Q^S = \frac{1}{|Q|}\sum_{q \in Q} Recall_q^S$$

$$AvgF1_Q^S = \frac{1}{|Q|}\sum_{q \in Q} F1_q^S$$

In the package, you will find class `SearcherEvaluator` (in SearcherEvaluator.java). The constructor is already implemented for you, which loads in the queries and ground-truth documents into `queries` and `answers` class variables, respectively. You are only responsible for implementing the following methods:

`public double[] getQueryPRF(Document query, Searcher searcher, int k)`: Computes query-wise precision, recall, and F1 for `searcher` that retrieves the top `k` search results for the given `query`. Returns an array of exact three values, containing precision, recall, and F1, respectively.

`public double[] getAveragePRF(Searcher searcher, int k)`: Computes average precision, recall, and F1 for `searcher` that retrieves the top `k` search results for each query in `queries`. Returns an array of exact three values, containing precision, recall, and F1, respectively.

Executing `testCompareTwoSearchersOnSomeQueries()` in `StudentTester` should give the following output.

```
@@@ Comparing two searchers on some queries in ./data/lisa
@@@ Finished loading 35 documents from ./data/lisa/queries.txt
@@@ Finished loading 6004 documents from ./data/lisa/documents.txt
@@@ Finished loading 6004 documents from ./data/lisa/documents.txt
@@@ Query: [ID:1, I AM INTERESTED IN THE IDENTIFICATION AND EVALUATI...]
        Jaccard (P,R,F): [0.1, 0.5, 0.16666666666666669]
        TFIDF (P,R,F): [0.2, 1.0, 0.33333333333333337]
@@@ Query: [ID:18, I WOULD BE PLEASED TO RECEIVE ANY INFORMATION ON T...]
        Jaccard (P,R,F): [0.7, 0.1320754716981132, 0.2222222222222224]
        TFIDF (P,R,F): [0.9, 0.16981132075471697, 0.2857142857142857]
@@@ Query: [ID:35, I AM INTERESTED IN ALMOST ANYTHING TO DO WITH OCCU...]
        Jaccard (P,R,F): [0.0, 0.0, 0.0]
        TFIDF (P,R,F): [0.3, 0.42857142857142855, 0.3529411764705882]
@@@ Total time used: 4362 milliseconds.
```

Executing `testCompareTwoSearchersOnAllQueries()` in `StudentTester` should give the following output.

```
@@@ Comparing two searchers on all the queries in ./data/lisa
@@@ Finished loading 35 documents from ./data/lisa/queries.txt
@@@ Finished loading 6004 documents from ./data/lisa/documents.txt
@@@ Finished loading 6004 documents from ./data/lisa/documents.txt
@@@ Jaccard: [0.11714285714285716, 0.11296227751050206, 0.09837910465851286]
@@@ TFIDF: [0.24285714285714285, 0.33600655994655754, 0.23206026092658685]
@@@ Total time used: 25709 milliseconds.
```

General Instructions (and Warnings) for Implementation

1. While certain external libraries (commons-io and open-nlp) are provided, they are for facilitating file IO and text preprocessing (such as stemming). In your implementation parts, you should not have to rely on these external libraries. Furthermore, you are not allowed to include any other third-party libraries in the project. That is, you code should be able to compile and run without having to include external .jar files (that are not already provided in the package). The auto-grader will only copy required submission files into the grading system (so make sure the filenames match with the requirements).
2. Do not alter any existing method declarations, including changing method names, return type, parameters, exception throwing declaration, etc. However, ***you are allowed and encouraged to implemented additional helper methods and variables***, as long as they reside in the required submission files.
3. Make sure that your code compiles. The auto-grader will not fix compilation errors for you. Erroneous codes that do not compile may result in zero scores in corresponding test-cases.
4. Do not use packages. Put every Java classe in the "default package".
5. Write your names, IDs, and section(s) on top of every java file that you submit. You will be penalized for any submitted source code files whose authorship cannot be directly identified.
6. Comment your code thoroughly! You are working in a team. *Properly* commenting code will not only allow your teammates to understand your logic better, but will also prevent your project from being penalized for the lack of sufficient documentation.
7. [**Important**] Your code will be tested against different corpora. So make sure that you do not hard-code any corpus-specific information, and that your code is generalized for other corpora that follow the same assumptions (about format, filenames, etc.).
8. Your relevance scores may be slightly different from the sample outputs, but should not differ more than 10^{-7}.
9. Your implementation must be reasonably efficient. The run time (in terms of milliseconds) must not exceed twice the "Total time used" shown in each sample output. The sample outputs were generated on a low-end machine with Intel Core i5 650 @3.2GHz, 4GB of RAM, and a typically non-solid-state HDD.

Task 3: Report

Now that you have two functioning search systems and a means to evaluate them. Write a report to answer the following questions.

1. Which search algorithm (Jaccard vs. TFIDF) is a better search algorithm for the LISA corpus, in terms of relevance and time consumption? Quantitatively justify your reason scientifically and statistically (i.e. avoid using your gut feelings).
2. Currently, ***k*** is fixed at 10. Compute the average precision, recall, F1 for both the search systems for each ***k*** (i.e. precision@k, recall@k, and F1@k), where *k* ranges from 1…50. (You should write a script that automatically does this for you, instead of manually changing **k**.) Visualize your findings on beautiful and illustrative plots. What conclusions can you make?

3. From 2.), generate precision vs recall plots for each search system. Explain how you can use these plots to explain the performance of each search algorithm.

**Please write your own reports. We will submit your reports to TurnItIn and deduct points if high similarity index is detected.*

Grading

Here's the tentative breakdown of the criteria:

If you submit	5%
Follow the submission protocol	5% (Make sure you follow the submission instructions carefully.)
Jaccard Searcher	20%
TF-IDF Searcher	30%
Searcher Evaluation	20%
Report	20%

Additionally, you will be penalized by 50% if your program takes longer than 2x times shown in the example output to run. If a test case takes longer than 5 minutes to run, you will receive 0. The highest achievable score is 100%.

Deliverables

To sum up the items that you need to do. Please follow these steps.

1. Implement `JaccardSearcher.java`, `TFIDFSearcher.java`, and `SearcherEvaluator.java`. ***Write your team members' names, IDs, and sections in a comment section on top of each Java file.*** You will be penalized for any submitted code whose ownership cannot be determined. Do not write names or comments in any output files. You are not allowed to use any third-party libraries, except the ones provided. You must not modify `Document.java`, `Searcher.java`, `SearchResult.java`, and `StudentTester.java` (except the main method).
2. Analyze the results and write up the report.

Submission

It is important that you follow the submission instructions to enable your code to be properly graded by the auto grader. Failing to do so may result in a deduction and/or faulty evaluation of your project.

1. Create a new directory and name it `P2_<id1>_<id2>_<id3>`, e.g. `P2_5988111_5988222_5988333`. Let's call this the submission directory.
2. Put all the code files (i.e. .java files) that you have modified/created in the submission directory. **DO NOT SUBMIT** `Document.java`, `Searcher.java`, `SearchResult.java`, and `StudentTester.java`.
3. Copy your report as a single **PDF** file to the submission directory.
4. Use ***7zip*** to compress the submission directory and produce the ***submission package,*** e.g. `P2_5988111_5988222_5988333.7z`. Do not submit .zip or .rar files.
5. One of your team members then submits the submission package on MyCourses.

Late submission will suffer a penalty of 20% deduction of the actual scores for each late day. You can keep resubmitting your solutions, but the only latest version will be graded. *Even one second late is considered late.*

Extra credit (up to additional 10%):

ITCS414 has been known for tough exams. Getting good and legitimate scores on projects is an important key to excel in this course. Fortunately, you can earn up to 10% of extra credit for this project by implementing `MyCoolSearcher` (in MyCoolSearcher.jar) that extends `Searcher`. You can implement any search algorithm in `MyCoolSearcher` as long as it yields better performance in terms of average F1 on the LISA corpus than the best average F1 achieved by `TFIDFSearcher` (i.e. Avg F1 = 0.232, k = 10). Once you find a cool search algorithm that beats ours, then:

1. Implement `MyCoolSearcher` in MyCoolSearcher.java
2. Implement `public static void testYourSearcher(String corpus)` in StudentTester.java that produces the evaluation results of your searcher for the given `corpus`.
3. In the report, compare the results with those of Jaccard and TF-IDF based searchers, and explain how your cool searcher works, and why it performs better than our Jaccard and TFIDF based searchers.
4. Also submit MyCoolSearcher.java and StudentTester.java along with the submission package for us to verify.

Bug Report and Specs Clarification:

Though not likely, it is possible that our solutions may contain bugs. A bug in this context is not an insect, but error in the solution code that we implemented to generate the test cases. Hence, if you believe that your implementation is correct, yet yields different results, please contact us immediately so proper actions can be taken. Likewise, if the project specs contain instructions that are ambiguous, please notify us immediately.

Need help with the project?

If you have questions about the project, please first post them on the forum on MyCourses, so that other students with similar questions can benefit from the discussions. If you still have questions or concerns, please come see one of the instructors during the office hours, or make appointments in advance. TAs may also help with some trivial issues. We do not debug your code via email. If you need help with debugging your code, please come see us. Consulting ideas among friends is encouraged; however, the code must be written by members in your own team without looking at other teams' code. (See next section.)

Academic Integrity

Don't get bored about these warnings yet. But please, please do your own work. Though students are allowed and encouraged to discuss ideas with others, the actual solutions must be written by themselves without being dictated or looking at others' code. Inter-team collaboration in writing solutions is not allowed, as it would be unfair to other teams. It is better to submit a broken program that is a result of your own effort than taking somebody else's work for your own credit! Students who know how to obtain the solutions are encouraged to help others by guiding them and teaching them the core material needed

to complete the project, rather than giving away the solutions. **You can't keep helping your friends forever, so you would do them a favor by allowing them to be better problem solvers and life-long learners. ** Writing code is like writing an essay. If each of you writes your own code, then it is almost impossible that your codes will appear similar. Your code will be compared with current and previous students' submissions and online sources using state-of-the-art source-code similarity detection algorithms. If you get caught cheating, serious actions will be taken!