COMP90042 Web Search and Text Analysis, Semester 1 2015

Project 1: Building and evaluating an information retrieval engine

Due:	5pm, Wed 15th April 2015 (see below for late submission policy)
Submission method:	To be described on LMS
Submission materials:	Python source code files in plaintext and README file; Report in PDF format.
Marks:	25% of subject mark

Overview

This project requires you to develop a complete information retrieval engine in Python, for efficient search in a large collection of (we)blog documents. Your objective is to develop indexing and querying algorithms to support efficient ad-hoc document retrieval. A key focus of this project is critical analysis and experimental evaluation, for which you will need to report on the relative merits of several information retrieval methods. In the PDF report, you will need to describe the insights you have developed through this work, as well as your approach, motivations and empirical findings. Note that the report is the most important part of your submission, while the software is only secondary; you will not be assessed on the quality of the software programming.

Resources

Given a large corpus of blog documents and a query, your task is to return a ranked list of relevant documents. We will be using a subset of the data from one of the early TREC evaluations, the large-scale annual evaluation exercise for IR research. You can find the materials in the directory `/home/subjects/comp90042/2015-sm1/project1/` which can be accessed from the new CIS servers, e.g., `nutmeg.eng.unimelb.edu.au` or `dimefox.eng.unimelb.edu.au`. The blog documents are in the subdirectory `blogs`, where each document is stored in plaintext in a separate file. The filenames serve as the document identifiers. These files already been *parsed* using Apache Tika to remove HTML and other formatting, leaving the text component of each post.[1] The corpus is small for IR, but still large enough to need careful treatment — 175Mb uncompressed, and 59Mb in the zipfile. You will have to take care with file storage to ensure that you don't exceed your hard disk capacity or filesystem quota (and time and memory limitations!).

As well as the document collection, you'll also find the query files `06.topics.851-900.txt` and `qrels.february`. The former file is a list of 50 queries, including for each query a title and a detailed elaboration of the information need. The latter file has relevance judgements (*qrels*) for the queries, listing for each query number (field 1) the document identifier (field 3) and human judged relevance (field 4). Note that although the TREC queries call for opinion-oriented results (and matches containing opinions are assigned qrels > 1), in this project we ignore this aspect by considering binary relevance only, i.e., all qrels ≥ 1 are deemed relevant.

[1]There are inevitably a few processing errors, which you can safely ignore.

Terms of Use

As part of the terms of use of the TREC '06 Blog data, in using the data you must agree to several usage conditions, as outlined in the file `README.licence`. These conditions concern the copyright of the crawled data and limitations on the use of the data. Note that you will should not distribute copies of the dataset to others, and you will need to delete any copies you have made of the data at the end of this semester. The corpus comprises unfiltered crawled text from the web, and may include offensive or objectionable material. This reflects the general composition of the web and the general challenges present in web based information retrieval. The University of Melbourne takes no responsibility for opinions expressed in the corpus, nor takes any responsibility for offence caused by these documents.

Task 1: Development of a Retrieval System

Your code should perform the following:

1. Preprocess the corpus to obtain bag-of-words document representations. You may want to consider tokenisation, filtering, stopping, case folding, and/or stemming.
2. Create a inverted index of the corpus, based on a vector space model with TF*IDF weighting.
3. Implement query processing using the cosine distance for document ranking, to sequentially process a list of queries.
4. Return a set of *k* (e.g., *k*=10 or 100) matching documents for each query, sorted in decreasing order of cosine distance.
5. Evaluate the quality of your ranked result lists.
6. Make use of suitable efficient data structures and persistent storage to avoid redundant processing. Note that we will be using an in-memory index, so you don't need to worry about efficient disk formats or file compression.
7. Implement an extension (see details below).

You will probably want to begin working with a small subset of the corpus and/or queries to allow for faster development. For your final results in the report, you should attempt to use the whole dataset. If you find this too challenging, then please justify your choice of subset of the corpus or selection of queries in your report.

Methods

You will need to implement a VSM information retrieval system and an extension of your choice. The first part requires you to construct an inverted index over your corpus, to provide an efficient means of looking up documents containing a query term and the TF*IDF for each term, and means of computing cosine similarity between the query and the documents. You will need to think carefully about the most appropriate data structures for storing the index and processing the query, taking into account the importance of fast query execution times as well as being conservative with computer memory.

The second development task involves extending your IR system by implementing one or more extensions of your choice from the list below:

1. a query expansion mechanism using a dictionary (e.g., WordNet, which can be accessed through nltk) or an automatically learned thesaurus
2. develop an efficient implementation of the index, in terms of the memory footprint, through the use of compressed encoding of the dictionary and/or posting lists. For this you will probably want to use Cython or similar to allow the use of C/C++ structures in Python.

3. compare several different methods for term weighting and query-document comparison besides TF*IDF and cosine similarity. For instance, you might use variants of TF*IDF, pivoted document length normalisation, the BIM and/or the BM25 model.
4. index not just single words, but also multiword units to support phrase queries. You might want to use a biword dictionary or position based index, or some combination.
5. relevance feedback for query adaptation (using e.g., the Rocchio algorithm), in which you can compare explicit and pseudo relevance feedback
6. another idea that takes your fancy, of similar effort and complexity to the ideas listed above

You will need to evaluate the effectiveness of your methods, in terms of the quality of the retrieval outputs as well as memory use and runtime (particularly for 2, above). To evaluate you should run several queries by hand, using either TREC 'topics' and detailed information needs (you will need to consider how to frame these as a query string), or coming up with some queries of your own. You should look at the results and judge how relevant they are to your information need, and select a few examples to demonstrate your system's performance for your report. You should also evaluate your system automatically for the TREC topics using the *qrels* provided and present the results in your report.

Task 2: Report

The report should consist of 1200–1800 words and should discuss:

1. A basic description of the task and the methods that you have explored, including the choice and justification of your choice of processing techniques, models and algorithms;
2. Whether the methods worked well, using example queries to demonstrate the effectiveness of the methods you tried. What worked and what didn't?
3. Measure empirical performance using the queries provided and the qrels, e.g., using MAP, Prec@10 and precision-recall curves. You should consider using this data to tune your system (being careful to holdout data for testing.) Does automatic evaluation concur with your judgements? How do you express the information need as a query string?
4. Considerations of efficiency of your implementation, in terms of space and time. Motivate your choice of data-structures and algorithms, and measure its usage. Is your system appropriate for this dataset, and how might it scale to larger datasets and longer queries?
5. A discussion of the challenges you faced in solving the task, and the suitability of the methods chosen for the task. What lessons have you learned that you would use were you to start afresh?

Please aim to be concise in your report, overlong or vague reports will be penalised. You will not have space to discuss the many technical aspects of your software implementation, instead you must focus on the scientific questions and the knowledge you have gained from your experiments. Do not include copious numbers of results tables or graphs, please use concise means of displaying your results, such as overlaying several curves on one graph or reporting averages. Remember that the report is the primary unit of assessment, so please devote sufficient time and effort to best answer the above requirements.

Submission

Submission will entail two parts:
1. A zip or tar archive containing your python code and a README file that explains how to run your submission on the CIS servers, and the format of your system output.
2. Your written report, as a single Portable Document Format (PDF) file.

Details of how to submit will be given on the LMS closer to the deadline.

Your software should be written in Python and can use any of the core libraries, including scientific computing libraries numpy, scipy and matplotlib. You may also use nltk. If you wish to use other libraries outside of the core libraries please ask for permission first.

Late submissions will be docked marks at a rate of 10% of the project mark per business day, and no submissions will be accepted more than 5 business days (i.e., one week) late.

Submissions will be assessed based on the following criteria:

- *Critical analysis:* clear presentation of approach, experimental setup used, display of experimental findings (e.g., in tables and/or charts) and interpretation of findings
- *Efficiency:* clever use of data structures and algorithms, and descriptive analysis of these choices
- *Soundness:* sensible decisions in selection and tuning of methods; overall retrieval accuracy
- *Report quality:* well argued and logical structured presentation of project
- *Creativity:* evidence of careful thought, understanding and innovation

Changes and updates to this document

Please check the LMS for any updates or clarifications to this document, which will supersede this version.

Academic Misconduct

For most people, collaboration will form a natural part of the undertaking of this project. However, it is still an individual task, and so reuse of code or excessive influence in algorithm choice and development will be considered cheating. We will be checking submissions for originality and will invoke the University's Academic Misconduct policy (http://academichonesty.unimelb.edu.au/policy.html) where inappropriate levels of collusion or plagiarism are deemed to have taken place.