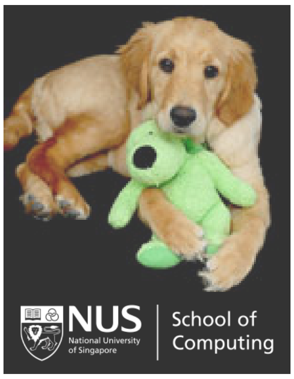

Information Retrieval

NUS SoC, AY 23/24, Semester II, Fridays 12:00-14:00

Last updated: 5 Apr 2024 - Information updated for AY23/24 Semester 2.

Deadline for submission: **25 Apr (Thu), 2pm SGT**

Homework #4 » Legal Case Retrieval Mini Project

In our final Homework 4, we will hold an information retrieval contest with real-world documents and queries: the problem of legal case retrieval. As described in lecture, legal retrieval is a case where structured documents are prevalent, so serve as a good testbed for a variety of different retrieval approaches.

Competition framework / Leaderboard.

Commonalities with Homeworks #2 and #3

The indexing and query commands will use an (almost) identical input format to Homeworks #2 and #3, so that you need not modify any of your code to deal with command line processing. To recap:

Indexing: `$ python index.py -i dataset-file -d dictionary-file -p postings-file`

Searching: `$ python search.py -d dictionary-file -p postings-file -q query-file -o output-file-of-results`

The differences from Homeworks #2 and #3 are that 1) `dataset-file` is a csv file containing all the documents to be indexed, and 2) `query-file` specifies a single query instead of a list of queries.

However, significantly different from the previous homework, we will be using a legal corpus, provided by Intelllex, a company with origins partially from NUS.

Problem Statement: Given 1) a legal corpus (to be posted in Canvas Files) as the candidate document collection to retrieve from, and 2) a set of queries (each query in its own file), return the list of the IDs of the relevant documents for each query, in sorted order of relevance. Your search engine should return the entire set of relevant documents (don't threshold to the top *K* relevant documents).

Your system should return the results for the query `query-file` on a single line. Separate the IDs of different documents by a single space ' '. Return an empty line if no cases are relevant.

For this assignment, you may use any type of preprocessing, post-processing, indexing and querying process you wish. You may wish to incorporate or use other python libraries or external resources. However, please take note of the following:
1) **AI / Machine learning-related approaches are generally not allowed** unless it is already part of NLTK or implemented by you.
2) Your submission is **not allowed to access the internet** (e.g., to use some online APIs).
3) You'll have to **include all necessary Python libraries with your submission properly** -- We will not install new libraries to grade your submissions.

Intelllex, the company we are working with for this contest, is particularly interested in good IR systems for this problem and thus is cooperating with us for this homework assignments. They have provided the corpus (the documents are in the public domain, as is most government information, but the additional structuring the Intelllex team has done is their own work) and relevance judgments for a small number of queries.

More detail on the inputs: Queries and Cases

The legal cases and the information needs have a particular structure in this task. Let's start with the information needs.

Queries:

In Intelllex's own system, searchers (lawyers or paralegals) use the familiar search bar to issue **free text** or **Boolean queries**, such as in the training query `q1.txt` : `quiet phone call`. and `q2.txt` : `"fertility treatment" AND damages`. The keywords enclosed in double quotes are meant to be searched as a phrase. and **a phrase can only appear as part of Boolean queries and the two types of queries can't be mixed.**. Therefore, queries such as `"fertility treament" damage` and `fertility treament AND damage` are simply invalid. In addition, they are are 2 or 3 words long, max; As such, if you want to support phrasal queries, you use n-word indices or with positional indices. To keep things sipmle, there are no ORs, NOTs or parentheses in the queries issued by us so you can simplify your query parsing code if you choose.

However, do also keep in mind that the results are simply a ranked list of documents. The Boolean queries (possibly with phrases) do not have to be processed strictly as defined. For example, a simple baseline system can just ignore all the Boolean operators and double quotes in the queries.

Lastly, the evaluation metric is Mean Average F2 (MAF2). Basically, for each query, we will compute the F2 values at the positions where a relevant document is returned. The average of these F2 values is the average F2 for the query. We then further compute the MAF2 by taking the mean of the average F2 over all the queries. You are to think about how get a high score in this metric and design your system accordingly.

Query Relevance Assessments:

The query is the first line of the query file. The file also comes with (very few) relevance judgments, as subsequent lines. Each line marks a positive (relevant) legal case identified within the corpus. You should ideally have your system rank documents from the positive list before any other documents. As relevance judgments are expensive (lawyers assigned the judgments made available to you), the bulk of the Intelllex corpus was not assessed for their relevance. That is, there may be additional documents that are relevant in the corpus that are not listed. However, your system will be evaluated only on those documents that have been assessed as relevant. We show the example for the above `q1.txt` .

```
    quiet phone call
    6807771
    3992148
    4001247
```

The above indicates that there are 3 documents, with `document_id` s 6807771, 4001247 and 3992148, that are relevant to the query.

Cases:

The legal cases are given in a csv file. Each case consists of 5 fields in the following format: **"document_id","title","content","date_posted","court"**.

Below are snippets of a document, ID 6807771, a case relevant to the above example query:

```
"6807771","Burstow R v. Ireland, R v. [1997] UKHL
34","JISCBAILII_CASES_CRIME

JISCBAILII_CASES_ENGLISH_LEGAL_SYSTEM


Burstow R v. Ireland, R v. [1997] UKHL 34 (24th July, 1997)



HOUSE OF LORDS





��Lord Goff of Chieveley ��Lord Slynn of Hadley
��Lord Steyn
��Lord Hope of Craighead ��Lord
Hutton


...

I would therefore answer the certified question in
the affirmative and dismiss this appeal also.","1997-07-24 00:00:00","UK
House of Lords"
```

You may choose to index or omit `title`, `court`, `date_posted` depending on whether you think they are useful to assessing a case's relevance to the query. More importantly, the `content` has much structure itself. You may decide to try to treat such work using preprocessing in your indexing if you think you can capitalize on it. Note that different jurisdictions may have differences in formatting, or even a different court's format compared to others.

Zones and Fields

As introduced in Week 8, **Zones** are free text areas usually within a document that holds some special significance. **Fields** are more akin to database columns (in a database, we would actually make them columns), in that they take on a specific value from some (possibly infinite) enumerated set of values.

Along with the standard notion of a document as an ordered set of words, handling either / both zones and fields is important for certain aspects of case retrieval.

Query Refinement

It is compulsory to implement at least one query refinement technique, such as (pseudo) Relevant Feedback and Query Expansion in your system. You might choose to switch it off in the end if you don't find it effective. However, it must be clear from your submission / README file that you have implemented one such technique and evaluated its effectiveness.

If you choose to implement two or more such techniques, you are eligible for up to 20% bonus marks. Please refer to the sections below for more information.

You might notice that many of the terms used in the text of the legal cases themselves do not overlap with the query terms used. This is known as the *anomalous state of knowledge (ASK) problem* or *vocabulary mismatch*, in which the searcher may use terminology that doesn't fit the documents' expression of the same semantics. A simple way that you can deal with the problem is to utilize **(pseudo) Relevant Feedback and / or Query Expansion**.

For example, we can perform a preliminary round of retrieval on the query terms. We can then assume that the top few documents are relevant and expand the query by 1) using the Rocchio formula, or 2) extracting important terms from these documents and adding them to the query. This is basically pseudo relevance feedback.

As another example, we can use manually created ontology (e.g., WordNet) or automatically generated thesauri (e.g., Co-occurrence Thesaurus) to identify related query terms.

Constraint on Index / Submission Size

Although you are free to explore different techniques in your system, you should keep your index at a reasonable size (e.g., by choosing space-efficient techniques and / or applying indexing compression techniques). Otherwise, your approach is not really scalable.

Since the corpus size is around 700MB, your submission (i.e., code + documentation + index + other miscellaneous files) should be no more than 800MB before zipping (and 500MB after zipping due the limit in Canvas). Submissions that fail to comply with this constraint will be rejected.

What to turn in?

You are required to submit `README.txt` , `index.py` , `search.py` , `dictionary.txt` , and `postings.txt` . Please do not include the legal case corpus.

In addition, to be eligible for the bonus marks, you are required to submit `BONUS.docx` .

Submission Formatting

You are allowed to do this assignment individually or as a team of up to 4 students. There will be no difference in grading criteria if you do the assignment as a large team or individually. For the submission information below, simply replace any mention of a student number with the student numbers concatenated with a separating dash (e.g., A000000X-A000001Y-A000002Z). Please ensure you use the same identifier (student numbers in the same order) in all places that require a student number.

For us to grade this assignment in a timely manner, we need you to adhere strictly to the following submission guidelines. They will help us grade the assignment in an appropriate manner. You will be penalized if you do not follow these instructions. Your student number in all of the following statements should not have any spaces and any letters should be in CAPITALS. You are to turn in the following files:

- A plain text documentation file `README.txt` : this is a text only file that describes any information you want me to know about your submission. You should not include any identifiable information about your assignment (your name, phone number, etc.) except your student number and email (we need the email to contact you about your grade, please use your e*******@u.nus.edu address, not your email alias). This is to help you get an objective grade in your assignment, as we won't associate student numbers with student names.
- All source code. Minimally, you should submit your `index.py` and `search.py` . However, please feel free to include additional files as needed. **We will be reading your code, so please do us a favor and format it nicely.** If you're using external libraries, make sure to include some nicely so they play well with our default ir_env environment (and acknowledge your external libraries as a source of help in your submission).
- All files related to your index. Minimally, you should submit your `dictionary.txt` and `postings.txt` . However, please feel free to include additional files as needed.
- **(For bonus marks only)** A Word document `BONUS.docx` : this is a Word document that describes the information related to the query expansion techniques you have implemented. You may include tables / diagrams in this document.

These files will need to be suitably zipped in a single file called `<student number>.zip` . Please use a zip archive and not tar.gz, bzip, rar or cab files. Make sure when the archive unzips that all of the necessary files are found in a directory called `<student number>` . Upload the resulting zip file to Canvas Files by the due date: 25 Apr (Thu), 2pm SGT

Grading Guidelines

The grading criteria for the assignment is below.

- 35% Documentation. This component is graded with a higher weightage in this assignment than in previous ones. This component breaks down into the following two subcomponents:
    - 5% For following the submission instructions and formatting your documentation accordingly.
    - 5% For code level documentation.
    - 10% For the originality of your ideas. Submissions attempting to do something unusual or interesting will be given higher marks. Sometimes attempting an interesting algorithm may have negative consequences on the performance of your system. In such cases, you can comment out the code that does not work as well. You should still document this code, so we can give you an appropriate documentation score. However, We will then assess your code's performance based on what actually runs from your submission.
    - 15% For your high level documentation, in your README document. This component comprises of an overview of your system, your system architecture, the techniques

used to improve the retrieval performance, and the allocation of work to each of the individual members of the project team. In particular, describe the techniques you have implemented / experimented with. Discuss about the effects of those techniques on the retrieval performance with reference to some experimental results and analysis. *If you have implemented two or more query expansion techniques for bonus marks, you should put all the information related to those techniques in BONUS.docx.*

- 65% Performance of your code. This component breaks down into several subcomponents:
  - 25% We will compare it against a baseline TF×IDF ranked retrieval implementation, in which the entire document is treated without zones (i.e., all zone/field information is removed). If your system works at least as good as the standard baseline, you will receive all 25% for this component.
  - 35% We will use a competition framework to assign credit to teams and to show the leaderboard. The framework uses Python 3.10.12 with NLTK installed (i.e., same as the Compute Cluster). The framework will attempt to auto-run your code, with the training queries and a few other development queries (hidden from you, but run by the framework), with a delay (e.g., of 3 hours) to prevent phishing for relevant documents. **Please do not log the queries.** In addition, if in a team, **please do not permute your student numbers to allow yourselves additional runs;** this is unfair to smaller teams. **Any team violating these guidelines will get 0 marks for this component.**
  - 5% We will measure the time efficiency of your system to answer queries. Your system should be able to answer a query within one minute. This requirement is mostly so that we can grade assignments in a timely manner.

- 20% (Bonus marks) Exploration on query refinement. Describe the query refinement techniques you have implemented / experimented with. Discuss about the effects of those techniques on the retrieval performance with reference to some experimental results and analysis. The bonus marks will be awarded based on the number / correctness / complexity of the techniques implemented, as well as the amount / quality of the discussion in the document.

Hints

- Working in a group can be helpful but can be counter-productive too. We have observed that group work tends to make homework submissions grades slightly higher than single submissions but also more average (it's harder to have an outstanding grade with a group). If you think you work better in a larger group, please do. Make sure to clearly partition the work and demarcate the API between your modules.
- While you don't need to print scores out for the official runs that your system will be graded on, you may find it useful to include such information in your debugging output when developing your solution.
- Similar to homework assignments #2 and #3, we will only be giving you a few queries to work with and (incomplete) query relevance judgments. We can only give you a few query relevance judgments, as the legal case relevance process also takes time to do for our human expert at Intelllex to assemble. However, we suggest you use your peers to pose

some queries yourself and assess whether they are relevant (this may be hard -- Intelllex tells us that legal background is required to understand many documents). Documentation and participation marks will be given to student teams who do this. *You might also try to search the Web for legal texts and landmark cases that are tagged (usually those tagged by Intelllex are considered such landmark cases anyways) to construct other queries.*

- Bulletproof your input and output. Note that now the input is only a singe file instead of a directory. Check that your output is in the correct format (docIDs separated by single spaces, no quotations, no tabs).
- If you're fishing for ideas about how to do legal case retrieval, you might find past iterations of the Legal Retrieval TREC task interesting. This was a yearly contest, that featured legal case retrieval from 2006-2012. You are encouraged, but not obliged, to use ideas from this research community. *Note that the "e-discovery" task that are the retrieval tasks in some years deals with evidence for legal cases, and not the legal cases themselves; so you may want to disregard some of the strategies involved there, but overall, we believe some of the general methods in these research papers may be helpful.*
