Boolean Model

A project for the *Information Retrieval* course

Angela Carraro

DSSC - UniTS

Aim of the project

Implementation of an Information Retrieval system using the Boolean Model.

Programming language: Python

Dataset: CMU Movie Summary Corpus
It's a collection of 42,306 movie plot summaries and metadata, available at the following link.
Problem: some movies are duplicated, I don't know why, so I had to use `set`s to check the queries results.

The project can be found on GitHub.

The IR system

It is able to answer boolean queries with AND, OR, and NOT. The system is also able to evaluate complex queries, even with many nested parentheses, like

"hello OR ((how AND (are OR you) OR I AND (am AND fine) OR I) AND am AND (sleepy OR hungry) AND cold)".

- Use the *and_query* function to connect all the words in your query text with ANDs.
- Use the *or_query* function to connect all the words in your query text with ORs.
- Use the *not_query* function to connect all the words in your query text with NOTs.
- Use the *query* function to answer a query with AND, OR and NOT without parentheses.
- Use the *query_with_pars* function to answer complex queries with AND, OR or NOT with parentheses, also nested.

The IR system

It can answer phrase queries using a positional index and also answer to queries like "$\texttt{term}_1 / k \ \ \texttt{term}_2$", with k an integer indicating the maximum number of words that can be between $\texttt{term}_1$ and $\texttt{term}_2$.

It can also answer wildcards queries using a permuterm index.

- Use the *trailing_wildcards* function to answer to trailing wildcards, like "term*".
- Use the *leading_wildcard* function to answer to leading wildcards, like "*term".
- Use the *general_wildcard* function to answer to general wildcards, like "C*T".
- Use the *multiple_wildcards* function to answer to multiple wildcards, like "*A*T".

The IR system

It performs normalization, removing all the punctuation and the symbols (except the ”end of word” symbol $) and putting everything to lowercase.

It can perform on demand spelling correction, using the edit distance (for time reasons keeping as correct the first character and searching only among the terms in the index that start with that character, but changing a parameter allows for a search in the entire index).

I evaluated the IR system on a set of test queries for each functionality, and in addition I checked that the results of the queries where correct using `assert`s.

Index

We have an index [`class Index`] that is a mixture of:

- An *inverted index* $\longrightarrow$ the basic index
- A *positional index* $\longrightarrow$ for phrase queries
- A *permuterm index* $\longrightarrow$ for wildcards queries

I implemented a way to save and load the entire index from disk, to avoid re-indexing when the program starts. To save the index I used ***`Pickle`***.

Saving the index makes us save a lot of time: processing the whole index takes around $220s$, while loading it from disk only around $45s$, which is almost five times less. Actually, after having implemented everything, processing it takes around $300s$ while loading it takes around $35s$.

Positional index

Used to answer to phrase queries.

To implement it, I added the list of the positions at which the term appears in the document `docID` into the `Posting` class.

To answer a phrase query we perform something like the intersection only that now we have to go inside and check if the two terms appear in adjacent positions. So we search if they are contained in the same document and if they are one after the other.

Permuterm index

Used to answer to general wildcard queries.

When I create the index I first create a *`Term`* with the term, the docID and the position in the document, then if the term is not present in my dictionary (trying to access it gives a *`KeyError`*) I create it, then I create all the rotations of the term combined with the ”end of word” character $. If the term is inside the dictionary I update its posting list merging the two terms. When I update the posting list of the ”normal” term all the posting lists of the rotated terms are updated, since they are a shallow copy of this posting list.

(✓ exploiting one of the annoying properties of Python)

Information Retrieval system

The Information Retrieval system [`class IRsystem`] contains both the corpus of documents and the index. It also implements the function `spelling_correction` to perform the spelling correction.

It also contains all the methods to answer to the various queries:

- `answer_and_query`
- `answer_or_query`
- `answer_not_query`
- `answer_query`
- `answer_phrase_query`
- `answer_phrase_query_ksteps`
- `answer_trailing_wildcard`
- `answer_leading_wildcard`
- `answer_multiple_wildcards`

Improvements

There are some features one could implement to extend the project:

- *Optimisation of boolean queries.*
  Evaluate the terms from the one with the shorter postings list to the largest.
- *Ranked retrieval.*
  Rank the documents according to how relevant they are.
- Use *standard test collections* to perform benchmarks.
  So to be able to compute precision and recall and display the precision-recall curve.

On to the code!

☺