



Documentation of the project Extended Boolean Retrieval Information Model

created by:
Miftakhov Rustam,
Zegeryte Karolina

Description:

The Extended Boolean Information Retrieval Model is a model of information retrieval that extends the basic Boolean model to allow creating ranks for the information containers (in our project they are represented by documents and texts inside of them).

The main goal of this project is the search in texts with the given string, which contains terms and boolean operators such as "AND", "OR", "NOT" and brackets.

In this model, documents are represented as sets of terms, and queries are represented as Boolean expressions composed of terms and operators. Each term is assigned a weight that reflects its importance in the document. The weights are typically calculated using TF-IDF (term frequency-inverse document frequency) or other statistical measures.

When a user submits a query, the model retrieves all documents that match the query by evaluating the Boolean expression. The results of retrieval are based on each document's relevancy: the higher relevancy is, the more the document suits the request given in the input string, as determined by the weights assigned to the terms in the query and the documents.

Algorithm:

First of all, we proceed the documents we have and make calculations for each document and each term's weights in the documents to make the search faster. Then we look at the given request string. If it does not have terms and operators in order or contains unknown operators, we return a white page with the warning about rewriting the request string. After we receive the right form request string, we start to search and calculate the results.

Used algorithm is pretty simple and is divided into 2 groups:

1. Contains "AND" and "OR" operators only:

using given in lecture n. 2 formula, we calculate the results of the search. If we see brackets, we calculate the start a recursive call to calculate the results.

2. Contains operator "NOT" as well:

as we do not have proper information how should we treat such operator in this model , we use the following principle to calculate the results:

weight of term(s) before "NOT" will be $1 - \text{the weight}$.

This method may be wrong. If it is so, we will reimplement it once we receive the right algorithm for solution.

In the implementation the following formula to calculate the request or part of the request, which contains operators "AND" only:

$$relev(q, d_j) = 1 - \sqrt[t]{\frac{(1 - w_{1,j})^2 + (1 - w_{2,j})^2 \dots + (1 - w_{t,j})^2}{t}}$$

where q is the request / part of the request and d is the document number j, w is the weight of each term in the document, t – number of all weights.

For "OR" operation we use:

$$relev(q, d_j) = \sqrt[t]{\frac{w_{1,j}^2 + w_{2,j}^2 \dots + w_{t,j}^2}{t}}$$

Implementation details:

The code implements a boolean model search algorithm that processes user input and retrieves relevant documents from a collection of documents.

The **CTerms** class contains several methods for initializing the search system, calculating term weights, processing boolean queries, and retrieving relevant documents.

The **initialization()** method reads in a collection of documents and extracts the terms in each document. It then stores the terms and documents in various data structures, such as maps and sets. It also uses the function **normalizeTerm()**, which transform a term into its lowercase variant and erases all characters that should not be in the term.

The **termFrequency()** method takes two arguments: a term (a word) and a document (a string of words). It calculates the frequency of the term in the document, which is the number of times the term appears in the document, divided by the total number of terms in the document. The result is returned as a double-precision floating-point number.

The **inverseDocFrequency()** method calculates the inverse document frequency of a given term, which is a measure of how much information the term provides. It first checks if the term exists in the *termCountMap*, which means it appears in at least one document in the collection. If the term is not found in the map, it returns 0, which means the term does not provide any information in the collection. If the term is found in the map, it calculates the inverse document frequency using the above formula and returns the result.

The **maxInverseDocFrequency()** method takes two arguments: a string representing a term, and a double representing the inverse document frequency (IDF) of that term. It calculates the maximum possible IDF value for that term, given a predefined maximum document frequency threshold.

The **getWeightForTerms()** method calculates the weight of each term in each document using the term frequency-inverse document frequency (TF-IDF) algorithm. The resulting term weights are stored in a map.

The **ReadInput()** method processes user input and normalizes the terms by converting them to lowercase and removing any non-alphabetic characters. It also checks the grammar of the query to ensure that it is syntactically correct.

The **inputGrammarTest()** method is a helper function used to validate the grammar of the input query. It takes in several parameters:

was_term: a boolean reference that keeps track of whether the previous token was a term or not

brackets: an integer reference that keeps track of the number of open brackets that have not yet been closed

s: the input query string

was_not: a boolean reference that keeps track of whether the previous token was a "NOT" operator or not

Returns true in case there were no syntax errors.

The **processBooleanQuery()** method takes a boolean query as input and calculates the relevance of each document in the collection based on the query. The method implements the boolean model of information retrieval, which uses logical operators such as AND, OR, and NOT to combine query terms. The method calculates a relevance score for each document based on the presence or absence of the query terms in the document.

The **requestCalculation()** method retrieves the relevant documents for a given query and returns them sorted by relevance score.

The **main** function of the code initializes the **CTerms** object and creates an HTTP server for processing user requests. The server receives user input, processes the input using the **CTerms** object, and returns the relevant documents to the user in an HTML format.

Variable *output* represents the documents and their relevance towards the input string. If there were no suitable documents found, is empty and the warning to the user shows. Even in case the syntax if the request is correct, there may be a warning. It is caused by the amount of documents in our database: there should not be the words from the request.

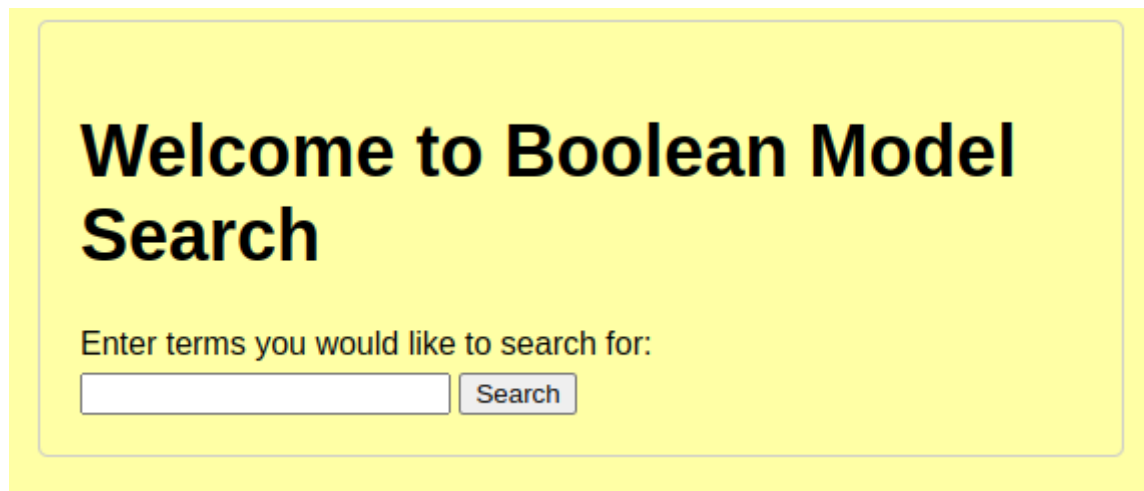
The whole project was created using C++ programming language and some HTML parts inside of it. Despite regular C++ libraries, we have also tried a httplib.h library to create a web page and make it work with our project, receive data from it and send our output information to display it to the user.

Input and output examples:

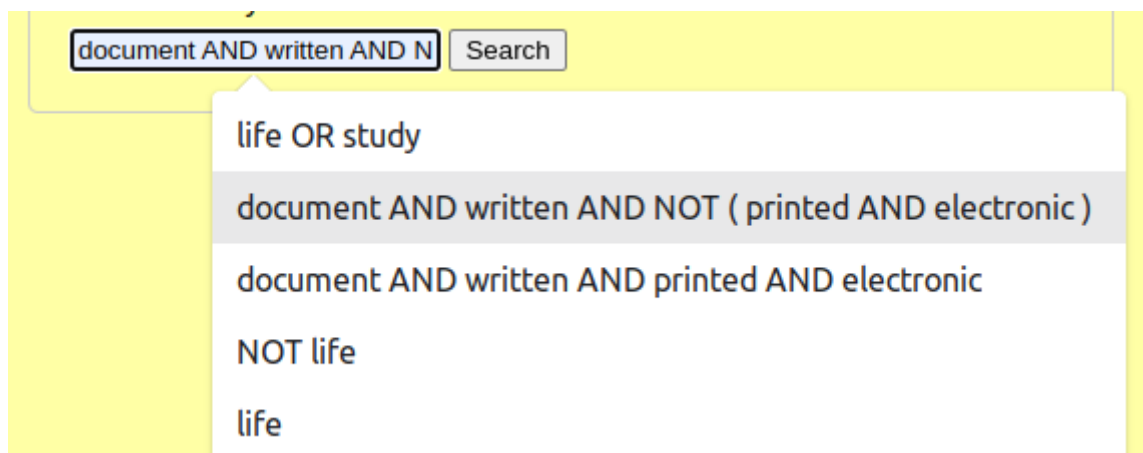
Here we will show you a couple of the possible input requests and their output from our “web page”.

On the main page you will see the greetings and a field you can enter your request in:

As we do not have designers in our team, the design of our pages is pretty simple. Hopefully, you consider it as “user-friendly” one. If you have some ideas, we are ready to hear you out.



When you click on the field move your cursor on it, some of the past requests will be given to you as an option what you can enter. It also shows you the acceptable syntax of a request.



After you submit you request, you will see a page with 10 or less the most relevant documents according to the input string. We will show you our output for the “beta version” of requests containing “NOT” operators:

This is a short version of the output. You see a couple of sentences describing the document and its relevance as well.

Top of relevant documents

Document documents/document.txt with relevancy 18.986045%

A few words about text you can see below:

A document is a written, printed, or electronic record that provides information or evidence on a particular subject or event. Documents come in many different forms, including reports, contracts, memos, letters, and more.

Document documents/document_28.txt with relevancy 18.579480%

A few words about text you can see below:

A license is a legal document that grants permission to use or possess something that belongs to someone else. Licenses are commonly used for a variety of purposes, such as driving a car, practicing a profession, or using software.

Document documents/vacation.txt with relevancy 18.350342%

A few words about text you can see below:

Vacation is a time to relax, unwind, and escape the stress of everyday life. Whether you prefer the beach, the mountains, or somewhere in between, there are countless destinations to choose from for your next vacation.

Document documents/universe.txt with relevancy 18.350342%

A few words about text you can see below:

The universe is a vast and mysterious place, stretching out to the farthest reaches of time and space. It is home to countless galaxies, stars, planets, and other celestial bodies, each with their own unique qualities and characteristics.

Document documents/tv.txt with relevancy 18.350342%

A few words about text you can see below:

Television, or TV, has been a staple of entertainment in households for many

Document documents/time.txt with relevancy 18.350342%

A few words about text you can see below:

Time is one of the most important concepts in our lives. It is a measurement

Here is the example of a wrong request:

Enter terms you would like to search for:

And a quite unpleasant output page:

Wrong input syntax. Please, write another one.

Yes, you are right, there is a white page with two sentences...

Experimental section:

As our project is not perfect and is represented by its demo version only, we can underline the following possible improvements:

1. Improve of the speed of preprocessing documents in our database. At the moment this is the slowest part of our project.
2. Improve the speed of results' calculating process.
We cannot complain about it on relatively small requests, cause the whole process takes less than a second for input strings of simple searches with max 6 terms. But the input with more complicated structures will perceptibly slow down the program. The speed of such long and complicated requests will decrease with almost linear time due to the implementation and depending on the length of the input and the amount of documents in the database. This decrease may be reduced using memorization techniques and other suitable improvements such as bitmask representation of the terms and documents' relations.
3. The design of our web page and its availabilities.
4. Increase of tests to find out probable mistakes and maybe even core dump situation when the program shuts down due to some inappropriate operations in it.
5. Fixing the "NOT" operand probable wrong implementation.

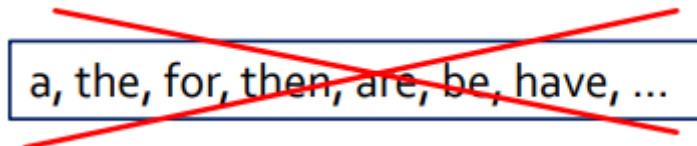
Other implementation improvements can be also acceptable. These 5 we see as the main problems of the project.

Discussion part:

Here we will discuss the information given in our lectures and our version of its implementation.

In the second lecture the normalization process was mentioned and we would like to present our version of solving such problem in the texts. As we have a Boolean Retrieval Information, we decided to make a simple normalization for every word in each document:

1. To get rid of any characters that are not included into the word, such as brackets, points, numbers, etc.
2. To get rid of “stop words” - very frequent ones, could be also numbers, names, etc. A good picture taken from the slides to describe



3. In other model of retrieval information should be used some techniques to get rid of different other forms of the same word (especially in case of irregular verbs). But our model should be as strict and precise as it can be, so we decided to leave other forms of the same words. In practice, it may be useful in law researches cases, for example.
4. Transform all the characters of a term into their lowercase representation. This one helps to find the terms faster and makes the implementation way easier.

In the same lecture we have found a couple of useful formulas to find out which documents suits the request the best. Their preciseness is negotiable, but at the moment this is the best technique we have found. Old experience solves new problems.

In the future, we would like to try more retrieval information techniques and find the fastest and the most reliable one, but we will always remember our first experience given by this project.

Conclusion part:

In this part we would like to underline the most important parts of this projects and our impression after working on this project.

We have an honor to be the one who has tried:

1. Implementation of Extended Boolean Information Retrieval Model.
2. Creation our own normalization function. We still have to learn much more about it, but as our first experience we consider that as essential part of our self-development process.
3. Implementation of processing the boolean type request. This part took as plenty hours and it's 100% worthy of each second spent on it. We had on mind to use a Context-free Grammar, but we decided to use a trusted technique which was given to us in the first year of our study – cycle and recursion mix. It would be much easier to track and debug using simple functions instead of more suitable and also complicated solutions.
4. Our first web page using C++. This one deserves its own point. We have tried to create a localhost web page with a simple design but still have tried some style techniques on it.
5. Creation of our own tests to check if the program works as it's supposed to. We admire people, who are quite patient employees at the QA-engineer positions. It is always difficult to accept your program may contain mistakes and weak parts, but creating your own tests and receive this disappointing feeling due to failures tempers junior programmers' minds.
6. Communication towards our project with our experienced and supportive teacher. The most important part of any projects created by the beginners. We are thankful for all the help our teacher gave us.

We will be happy to remake this project using more advanced techniques and to invite senior level programmers to take a part in it and share their knowledge and hints with us. In general, we consider work with data as a very pleasant and useful one, so we will not leave a period there, it's our very beginning on a very exciting path of improvements