Boolean Retrieval

August 8, 2025

Introduction

Information Retrieval

Information retrieval is a fundamental concept that plays a crucial role in the modern landscape of recommendation systems and Generative AI (GenAI). At a high level, it enables us to efficiently retrieve relevant data from vast collections of information.

Why Information Retrieval is Important?

- ✓ Search Engines: When you type a query, Google quickly scans through millions of available web pages online and presents you with the most relevant results.
- ✓ E-Commerce: Platforms like Amazon and Shopify utilize information retrieval technologies to help their customers find products they would like to purchase. Eg: If you type in "t-shirt for men," for example, you'll be presented with various men's t-shirts in a split second.
- ✓ Social Media: The combination of information retrieval and recommendation systems creates more personalized feeds on social media platforms.
- ✓ Internal Chatbots: Internal chatbots are becoming increasingly popular for customer assistance. These chatbots process requests and return relevant content or articles that might help troubleshoot customers' problems.

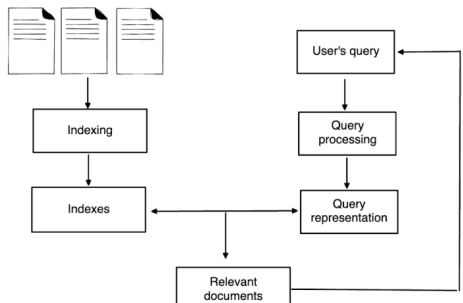
Working of Information Retrieval

Steps

- Document: Refers to any retrievable content for the user. This process includes videos on streaming services like YouTube or Netflix, web pages on search engines like Google or Bing, and data, images, and texts in information management systems such as company internal databases.
- Query: This represents the user's request, which can be expressed as a string of words or a complex set of criteria. A query represents specific information a user seeks, ranging from a simple keyword to a complex one requiring Boolean operators (e.g., "Find me a t-shirt AND for men").
- Indexing: As information retrieval needs to be fast and efficient in fetching relevant content, indexing plays a crucial role. It refers to creating a mapping between a document and its location in a database for faster retrieval.
- Retrieval: The process of returning relevant documents to the user. Several retrieval methods can be applied, including boolean retrieval and vector space models.

Information Retrieval (cont.)

- ✓ Document
- ✓ Query
- ✓ Indexing
- ✓ Retrieval



Boolean Retrieval System

Introduction

Definition

Boolean retrieval model retrieves relevant documents using a set of Boolean logic operators, such as AND, OR, and NOT.

- ✓ AND: Retrieves all documents that contain all terms in our query. If our query is "A AND B," then documents containing both A and B will be retrieved.
- ✓ OR: Retrieves all documents that contain any of the terms in our query. If our query is "A OR B," then documents containing either A or B, or both, will be retrieved.
- ✓ NOT: Excludes all documents that contain a specific term in our query. If our query is "A AND NOT B," documents containing A but not B will be retrieved.

Term-Document Incidence Matrix

In its most basic form, the Boolean retrieval model uses a concept called the Term-Document Incidence Matrix. Each cell in the matrix is expressed as a Boolean value with only two possible values: 0 or 1. It represents whether or not a term is present in a document. If a term is present, the value is 1; if not, the value is 0.

Example

Let's say we have three documents, and only three distinct terms exist in them. Document 1 contains terms 1 and 3 but not term 2; document 2 contains only term 1, and document 3 contains terms 1 and 2.

Find me documents with term 1 and term 2. To process this query, the Boolean retrieval method will perform the following operations:

- ✓ Take the row associated with term 1. In the above visualization, this value would be $[1, 1, 1]$.
- ✓ Take the row associated with term 2. In the above visualization, this value would be $[0, 0, 1]$.
- ✓ Perform a bitwise AND operation: $[1, 1, 1]$ and $[0, 0, 1] = [0, 0, 1]$.

Based on the result of the bitwise operation above, document 3 will be returned to us.

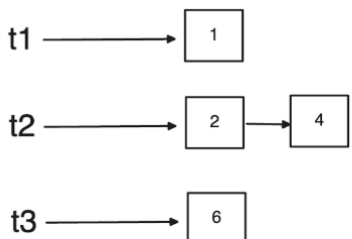
	document 1	document 2	document 3
term 1	1	1	1
term 2	0	0	1
term 3	1	0	0

Inverted Index in Boolean Retrieval Model

Problem with TDIM

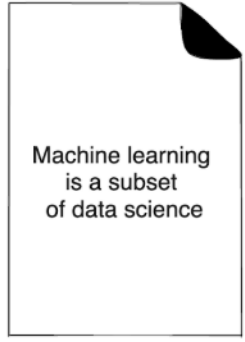
The problem with implementing a term-document matrix, as shown above, is that, in reality, the row associated with each term is extremely sparse.
Imagine we have 10,000 documents and 100,000 unique terms in total. If we implemented a term-document matrix as above, we'd end up with a 100,000 x 10,000 matrix.

Instead of creating the whole term-document matrix, we store only the IDs of the documents that contain the term x.



Steps in Inverted Index

- ✓ Preprocessing: Let's say our documents consist of texts. We begin by preprocessing and cleaning those texts, which includes methods like tokenization, normalization, lemmatization, and stopword removal.
- ✓ Start building Inverted index



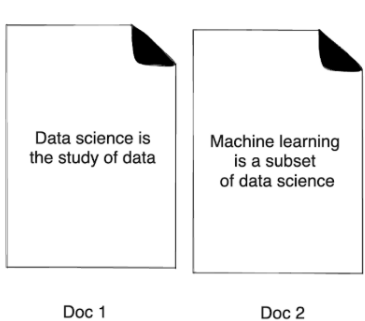
Steps in Inverted Index (cont.)

After cleaning the texts in the above two documents, we'll end up with the following:

Document 1: data science study data

Document 2: machine learning subset data science

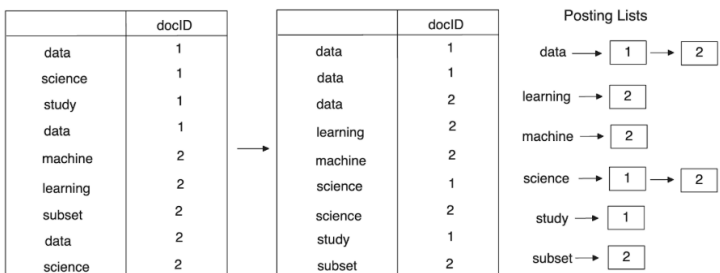
The inverted index method will then create a list of term-document IDs as follows:



	docID
data	1
science	1
study	1
data	1
machine	2
learning	2
subset	2
data	2
science	2

Steps in Inverted Index (cont.)

Next, we sort the terms alphabetically. If the same term appears in multiple documents, we sort them based on the document ID. Then, we create a posting list for each term that contains the document IDs in which the corresponding term is present.



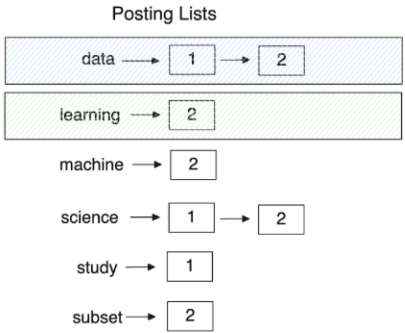
Querying

Now, we're ready to perform query processing. When retrieving based on a query:

- ✓ the first step is to parse the query to understand its structure and the operators that need to be checked (AND, OR, NOT).
- ✓ Then, operators are executed in order of precedence. Generally, NOT is processed first, followed by AND and OR to optimize the speed and efficiency of information retrieval.

Quering (cont.)

Let's say we have a query, "data AND learning." The query will be parsed first, identifying the AND operator. Then, it will retrieve the linked lists of terms "data" and "learning" from the inverted index. Since the operator is AND, it will find the intersection of both posting lists, which in our case is document 2. Therefore, document 2 will be returned to the user.



Experiment on Boolean Retrieval System

Step 1

Create three files:

- ✓ First file content: Apple juice is healthy and tasty.
- ✓ Second file content: Orange and apple are fruits.
- ✓ Third file content: Grape juice and watermelon are refreshing.

Code Listings

```
class BooleanRetrievalSystem:
    def __init__(self, folder_path):
        self.documents = []
        self.doc_names = []
        self.index = defaultdict(set)
        self.load_documents(folder_path)
        self.build_index()
```

Code (cont.)

Listing: Loading documents from folder

```
    def load_documents(self, folder_path):
        for filename in os.listdir(folder_path):
            if filename.endswith(".txt"):
                path = os.path.join(folder_path, filename)
                with open(path, "r", encoding="utf-8") as f:
                    self.documents.append(f.read())
                    self.doc_names.append(filename)
```

Code (cont.)

Listing: Preprocessing the text

```
    def preprocess(self, text):
        text = text.lower()
        text = re.sub(r"[^\w\s]", "", text)
        return text.split()
```

Code (cont.)

Listing: Building the inverted index

```
    def build_index(self):
        for doc_id, text in enumerate(self.documents):
            words = self.preprocess(text)
            for word in words:
                self.index[word].add(doc_id)
```

Code (cont.)

Listing: Parsing the Boolean query

```
1     def parse_query(self, query):
2         return query.upper().split()
```

Code (cont.)

Listing: Evaluating Boolean query

```
    def eval_query(self, query):
        tokens = self.parse_query(query)
        result_stack = []
        operator_stack = []

        def apply_operator(op, right, left=None):
            if op == "AND":
                return left & right
            elif op == "OR":
                return left | right
            elif op == "NOT":
                return all_docs - right

        all_docs = set(range(len(self.documents)))
```

Listing: Evaluating Boolean query

```
        i = 0
        while i < len(tokens):
            token = tokens[i]
            if token in {"AND", "OR"}:
                operator_stack.append(token)
            elif token == "NOT":
                i += 1
                term = tokens[i].lower()
                right = self.index.get(term, set())
                result_stack.append(apply_operator("NOT", right)
                    )
            else:
                term = token.lower()
                result_stack.append(self.index.get(term, set()))

            if len(result_stack) >= 2 and operator_stack:
                right = result_stack.pop()
```

```
                    left = result_stack.pop()
                    op = operator_stack.pop()
                    result_stack.append(apply_operator(op, right,
                        left))

                i += 1

        return result_stack[0] if result_stack else set()
```

Code (cont.)

Listing: Returning matching document names

```
    def get_doc_names(self, result_set):
        return [self.doc_names[doc_id] for doc_id in result_set]
```

Code (cont.)

Listing: Main execution block

```
if __name__ == "__main__":
    folder = "documents"
    brs = BooleanRetrievalSystem(folder)

    while True:
        query = input("\nEnter Boolean query (or type 'exit'): ")
        if query.lower() == 'exit':
            break
        result = brs.eval_query(query)
        if result:
            print("Matching documents:")
            for name in brs.get_doc_names(result):
                print("-", name)
```