Inverted Index

Jesus Arencibia David Garcia Pablo Guillo Alejandro Bolanos
Javier Franco Isabel Hernandez

October 8, 2023

Abstract

In this task, our goal was to create an inverted index using Python. More specifically, we wanted to create a function that would read a certain number of books and then create the index as a Datamart. Additionally, as we implemented two forms of creating the Datamart, we had to make a decision on which to choose. Also, we had to check that once created the reverse index, it worked as it was supposed to.

We developed our code in Python, creating auxiliary functions for eliminating the metadata of the books or normalizing the text, which helped in creating the main function. In the end, we had two ways of returning the inverted index: As a json or as a database. The results were positive given that the code was able to return us both the json and the database correctly.

Since we considered that a database is a better way to save loads of information, we reached the conclusion that it would be the structure of our Datamart. Another reason that helped us reach this conclusion was that it was easier to search specific data in a database than in a json.

1 Introduction

The concept of an inverted index has been a cornerstone in information retrieval systems for decades. It serves as a pivotal data structure that empowers efficient text search and retrieval, underpinning search engines, document databases, and various information management systems. An inverted index, at its core, reverses the mapping between terms and documents. Instead of associating documents with the terms they contain, it links terms to the documents in which they appear. This reversal facilitates rapid keyword-based searches and has revolutionized the way we access and retrieve textual information.

In this paper, we provide an exploration of the process of creating an inverted index, remarking steps such as metadata removal, text normalization, and index construction. We offer a comparative analysis of two representation formats, JSON and database, highlighting the advantages of the latter for managing substantial datasets. Furthermore, our contribution lies in presenting a robust algorithm for keyword-based searches within the inverted index. Our approach leverages SQL queries to efficiently locate documents containing specific keywords, streamlining the retrieval process.

2 Problem Statement

While the importance of inverted indexes in information retrieval cannot be overstated, the practical considerations and advancements in their implementation continue to evolve. This paper addresses critical questions concerning the optimization, scalability, and real-world applicability of inverted indexes. Furthermore, it explores the challenges associated with indexing and querying large-scale datasets, shedding light on potential solutions and best practices. By delving into the intricacies of inverted index construction and utilization, this research aims to provide valuable insights for researchers and practitioners in the field of information retrieval, offering a comprehensive view of the state-of-the-art techniques and strategies for efficient text search and retrieval in modern information systems.

Through this research, we navigated the complexities of handling diverse data formats, optimizing indexing processes for both speed and accuracy, and devising strategies to facilitate keyword-based searches across vast collections of documents. Our approach to leveraging SQL queries for efficient retrieval emerged as a direct response to the challenges posed by the ever-expanding volumes of digital content.

3 Method

Once we clearly defined our objective, we moved towards its resolution by implementing codes in Python. These codes take care of two fundamental tasks: the creation of inverted indexes from sets of texts and the efficient search for keywords in these sets of documents.

To create the inverted index, we start by removing the metadata from the text files, this ensures that the main content of the text is retained. Next, we normalize the text, converting all letters to lowercase and removing non-alphabetic characters. Next, we traverse each text file in the text set and split the content into unique words. We create a SQLite database and for each unique word, we add the words to the database. If it already exists, we add the name of the current file to the list of documents associated with that word. If it does not exist, we create a new entry in the database for the word and associate the name of the current file to this entry. Finally, we implement a function to search for keywords in the inverted index, which receives a keyword and uses an SQL query to search the database for the word and obtain the list of documents in which it appears. And then we will explain in more detail how we have done this project:

To begin with we have developed a function that performs Metadata Removal. This code aims to remove the metadata from a text file, preserving only the main content of the text. Metadata is the data that usually includes information such as the book title, author and other details that are not relevant for text analysis. The function `eliminate_metadata(input_file)` receives the location of a text file as input `(input_file)`. It then parses the contents of the file line by line and detects the first occurrence of the string "***" as the start signal for the metadata. From that point on, all subsequent lines are considered part of the text content and stored in a list called `content_without_metadata`. Finally, these lines are combined into a single string and returned as a result, thus achieving the elimination of this irrelevant data. Once we have this result we continue normalizing the text, which involves converting all letters to lowercase and removing non-alphabetic characters, such as punctuation marks and numbers. The normalizer(string) function receives a text string as input (string). First, it converts the entire string to lowercase using the lower() method. Then, it uses a regular expression to remove any character that is not a letter. The result is a normalized lowercase string with no non-letter characters.

Now that we have the data processed, we move on to the creation of the inverted index from the text sets. An inverted index is a data structure that associates each word in the texts with the list of documents in which it appears. The code goes through each text file and for each one divides the content into individual words, creating a set of unique words. For each word, it creates a new dictionary entry for the word and associates the current file name with that entry. The result is a dictionary that associates with each word a list of the books in which that word is found. In addition we have implemented a second variant storing the words and their respective books in a SQLite database.

And finally we implemented an algorithm for the Inverted Index Search, here we search for keywords in the previously created inverted index. It allows you to find documents containing specific keywords in an efficient way. The function `find_books_for_word(word, db_connection)` receives a keyword (Example: word) and uses a SQL query to search for the word in the database and obtain the list of documents in which it appears. Then, it returns this list as a result, thus finishing our project.

4 Conclusion and Future Work

In summary, our primary aim was to create an inverted index using Python, specifically focusing on building a function to process a set of books and generate a Datamart-style index. This project included the development of auxiliary functions for metadata removal and text normalization, crucial steps in creating the primary function. We provided users with two options for obtaining the inverted index: as a JSON or a database. Importantly, our code demonstrated successful functionality for both representation formats. Ultimately, we concluded that a database was the superior choice due to its efficiency in handling large volumes of data and its streamlined querying capabilities. This decision was further reinforced by the ease of searching specific data within a database compared to JSON. We also introduced a robust keyword-based search algorithm that allows us to efficiently search for and retrieve documents containing specific keywords within an inverted index.

As we look ahead, our project opens the door to future endeavors, such as algorithm optimization, experimentation with different database management systems, and scaling the system for larger datasets. Additionally, exploring alternative matrix multiplication methods and programming languages for performance comparisons holds promise. The incorporation of a web scraping module to download books from online sources and generate inverted indexes for these texts is another exciting avenue for expansion.

We have made the code and data associated with this research project available on GitHub. You can access the repository at the following link: `https://github.com/Javierfg2/Inverted_index`