Goodreads' Books and Reviews

Diogo Almeida
up201806630@edu.fe.up.pt

Pedro Queirós
up201806329@edu.fe.up.pt

ABSTRACT

Over time, the amount of data available online has grown in an unimaginable rate, reinforcing the need to have mechanisms to connect and gather all the available information. This document concerns the development process of one of those mechanisms: an information system on Goodreads' books and its reviews. To obtain a dataset with relevant and suitable information, data refinement and enrichment were performed. Furthermore, the dataset was analyzed for a better understanding of the available data, with some statistics being made for that same purpose. After using *Solr* in order to index the previously processed documents, the evaluation of several system configurations was made, demonstrating that the retrieval system's capacity is highly dependent on the defined indexing schema and the use of weighting filters. Moreover, with the objective of enhancing performance, several search system improvements were implemented. This improved system was then compared against another system configuration, concluding that the integrated improvements were having a positive impact in the system's overall query performance.

CCS CONCEPTS

• **Information systems → Evaluation of retrieval results**.

KEYWORDS

Dataset , Data Extraction, Data Preparation, Data Cleaning, Book, Review, Domain Conceptual Model, Pipeline, Python, Pandas, Solr, Schema, Query, Evaluation

1 INTRODUCTION

Books have been playing an important role in society since the early times of humanity. Nowadays, books can be read in many different formats and have numerous purposes: to transmit knowledge about a specific subject, to tell a story, to serve as a record for future generations, among others.

The current panorama of book search systems is vast in regards to the information that it is able to retrieve, letting users mainly search for titles, authors, keywords or genres (e.g., Amazon Book Search[1] or WorldCat Search[24]). The main goal of this project is to complement these types of search systems with a search engine that allows users to also search for books based on e.g., reviews, descriptions and awards, in order to provide an easier and better experience when trying to find a book to read that fits their preferences.

This article describes the first and second development phases of this search system and is divided into two major sections that characterize the several steps taken: the Data Collection section, which details the data preparation process, starts with **Data Extraction and Enrichment**, which describes the process of data gathering and enrichment, as well as detailing the source of the information. This is an essential step to guarantee that relevant data for the problem is used to create the first version of the dataset. The next section is **Data Preparation**, that details the procedure of cleaning and refining the collected data in order to have a consistent and practical dataset that is easier to handle. The **Domain Conceptual Model** section details how data can be conceptually described in the domain. It is followed by the **Possible Search Tasks** section, in which some of the possible queries to do in the system are listed. Finally, the **Dataset Characterization** section details how tools like plots and graphs are essential to better explore and understand all the collected information.

The Information Retrieval section describes the implemented Information Retrieval system and evaluates the results of possible retrieval tasks. The **Information Retrieval Tool Selection** contains a brief comparison between the two main search engines considered for the project. In Section 7, we have an **Information Retrieval Introduction**, where we describe the goals of an Information Retrieval system and detail the three different systems we'll use along with the methods to compare the global performance of these systems. Afterwards, the system's collections and documents are described in the **Collections and Documents** section. The following section, **Index Processing**, explains the documents' index processing, along with the description and characterization of the filters applied to a newly created field type, used in the most relevant fields. In the **Retrieval Process** section, the system is evaluated by comparing the performance of the three different system configurations on various information needs. Lastly, for the **Information Retrieval Tool Evaluation**, we present some remarks regarding the chosen search tool and explain why it ultimately was the best choice for the project.

Subsequently, in the **Search System Improvements** section we explore the different approaches taken in order to try and improve our search system's results. An analysis on the impact of these improvements on the system's search performance using Section 11's results from other system configurations is also performed.

In the **Revisions Introduced** section, we detail the various rectifications applied to the some of the sections of this report.

Finally, for the **Conclusions and Future Work**, we shortly recap all the sections in this paper and explain our main findings. We also present some of our plans for the future development phases of the project.

2 DATA EXTRACTION AND ENRICHMENT

In order to extract the best information for the project, various open data platforms were consulted. After some research, Kaggle[14] proved to be the best solution. Kaggle is a platform for data scientists and machine learning practitioners to publish datasets and explore and build models. The dataset from Kaggle, **'Goodreads Books - 31 Features'**[17], which has around 150 MB, contains 52,199 books with 31 features each. These features include several relevant aspects about a book, like its title, author, description, publisher, the series it belongs to, various details about its rating (1-5 stars), and more. There are also, for each book, the *recommended_books* feature,

that contains books that are recommended by Goodreads and the *books_in_series* feature that contains the books that belong to that book's series, it is important to note that all the books mentioned (by id) in these 2 features are also present in the dataset.

After an initial review of the Kaggle dataset, it was decided to enrich the project's data further with more text-rich fields. Therefore, another dataset was created using some reviews from each book in the Kaggle dataset. These reviews were scraped from the Goodreads website[13] and include the review's author, text and publication date. The information was scraped and exported to a .csv file using a Python script with the *Requests*[18] and *BeautifulSoup*[19] libraries. This .csv file ended up with a size of around 372 MB, containing around 8 reviews for each book.

3 DATA PREPARATION

Before proceeding with its preparation, the data was first carefully analysed to ensure that only the relevant information for the project was selected based on the possible search tasks that could be done.

Firstly, we removed the columns that don't have relevant information or that have a significant amount of NaN (Not a Number) values, for instance, the "asin" (Amazon Standard Identification Number) column.

Secondly, books that didn't have an original title got this column filled with their title.

Thirdly, the books without all the relevant information, for instance, title, publisher or author, were removed from the dataset, resulting in 35,880 books.

Next, due to bad encoding in the original dataset, there were some strange characters appearing in books that had column values that weren't in English. This was fixed so all characters from all languages appear correctly.

After that, books with descriptions containing less than 25 characters were also removed for not having enough information, leaving 35,655 books.

Books that weren't in the reviews dataset also ended up being removed, resulting in the final 35,347 books. Although the reviews dataset was scraped using the books from the original dataset, it wasn't possible to scrape all of them due to errors, so they had to be removed from the dataset as well.

Finally, the referenced books in the *recommended_books* or *books_in_series* columns that weren't in the dataset (because they were deleted by the previous operations) were also removed. Some columns (awards, genres and their votes, and characters) were extracted to a new .csv file to make it easier to analyse the final data that will be used. These columns were also removed from the original dataset in order not to have duplicated information. All of these operations were made using *Pandas*[15], a Python library. The full data pipeline can be observed in Figure 10.

4 DOMAIN CONCEPTUAL MODEL

A domain conceptual model was made to represent the main entities of the domain. The book class is the center of the conceptual model and it has the following attributes: Goodreads id, the title, the original title, the links for the book's cover, Goodreads page, Amazon and WorldCat, number of ratings, average rating, number of pages, isbn, isbn13 and the book's description. Every book was written by at least one author and published by one publisher. A book can belong to a series (collection), can have settings (which are the locations where the story unfolds) and can have characters. Each book has several genres which have several votes from the readers. Furthermore, they also have the number of votes for the ratings from 1 to 5 and some reviews made by some Goodreads users. Finally, books still have awards with the respective year when they have won it and recommended books based on their genre or rating. The UML for the Conceptual model is present in Figure 1.

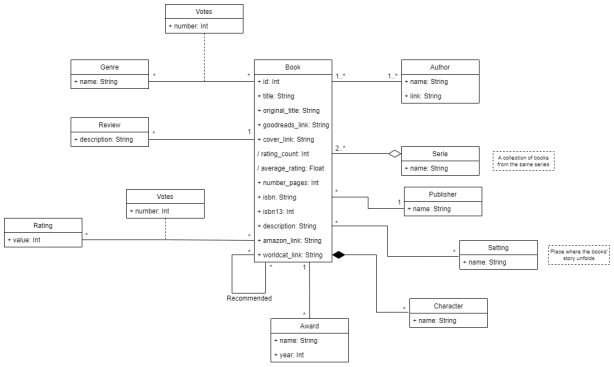

Figure 1: Conceptual Model

5 DATA CHARACTERIZATION

With the aim of better understanding and classifying all the collected data, charts were developed regarding some of the dataset's most interesting and striking features. These features and respective charts are analyzed next.

A. Awards and Books

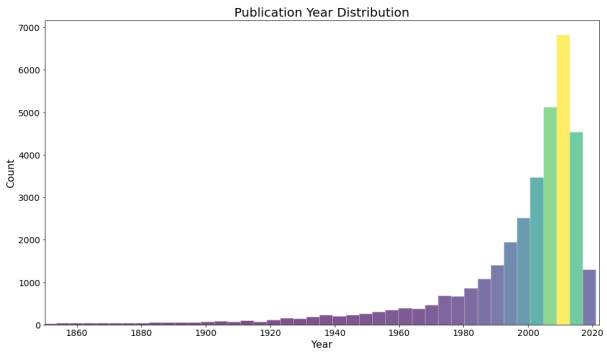

Figure 2: Publication Year Distribution

Regarding the year a book was published, it can be observed in **Figure 2** that the dataset mainly focuses on modern releases, containing many books published after the 2000's. Another interesting note is that there seems to be an increase in included books published from the 1800's onwards.

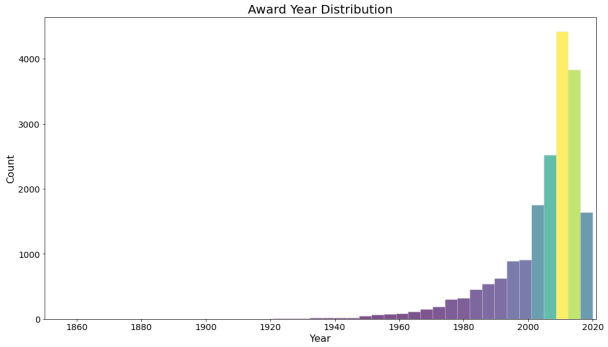

Figure 3: Award Year Distribution

As observed in **Figure 3**, the year an award was given also follows the same trend as the book publication year distribution seen previously, with most awards being handed out after the 2000's. This makes sense, since most books present in the dataset were published around this time.

About 8,948 of the total of 35,347 books (25.3%) were given an award. Since the dataset contains 19,186 awards, a lot of these awarded books have been distinguished multiple times: 4,018 to be exact (44.9% of the awarded books). The fact that a considerable chunk of books got an award is a good indicator of positive book ratings, a feature that will be analyzed later.

B. Book Genres

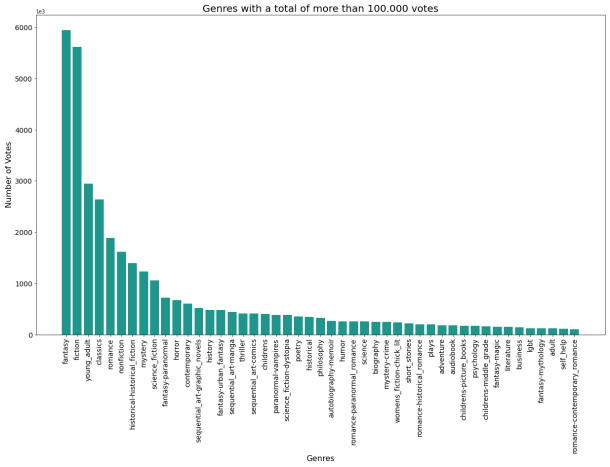

Figure 4: Genres with a total of more than 100000 votes

In the Goodreads website, the source of the books dataset, users can vote in the genres they think the book fits into. Upon analyzing **Figure 4**, the conclusion is that the "fantasy" and "fiction" genres are the most voted, with almost 6,000,000 votes each, 3,000,000 more than the third most voted genre "young_adult".

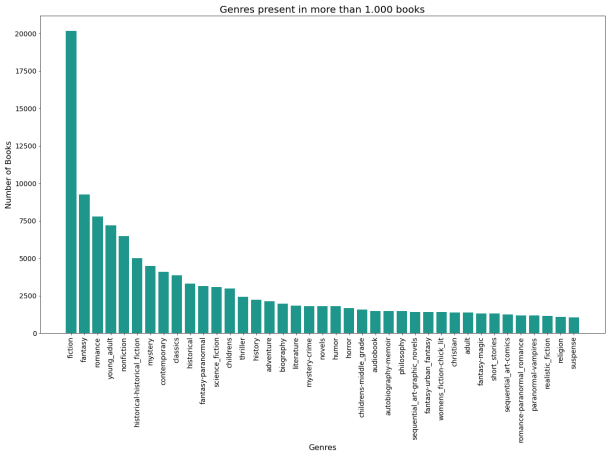

Figure 5: Genres present in more than 1000 books

The bar plot present in **Figure 5** is similar to **Figure 4's**, however, it now counts the genres that are present (have at least one vote) in more than 1,000 books. In this chart, we can observe that the "fiction" genre is present in a lot more books than the "fantasy" genre, although both genres were very close in votes, as analyzed in the previous chart (with "fantasy" actually having more votes than "fiction"). From this we can deduce that, in our dataset, "fantasy" genre books are fewer, but more popular (have more votes and therefore more users reading it) than "fiction" genre books, that need presence in more books to match the amount of votes that the "fantasy" genre has. In fact, the votes to books ratio in the "fantasy" genre is superior to every other genre.

C. Book Ratings

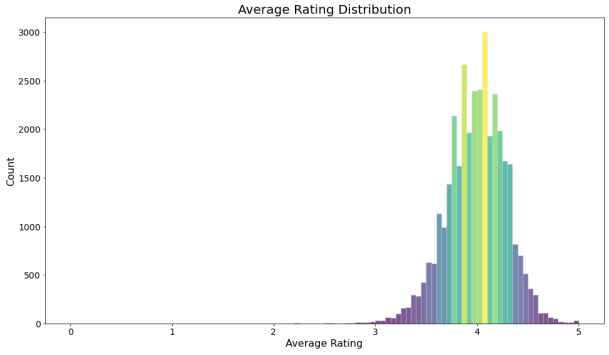

Figure 6: Average Book Rating Distribution

By observing **Figure 6** we can see that, in a classification of 0-5 stars, most books in the dataset have a rating of around 4 stars. This large amount of positive ratings was expected, since in the book awards analysis done before, we saw that there was a good amount of awarded books, which generally means that those books are acclaimed by both critics and readers/users.

D. Authors

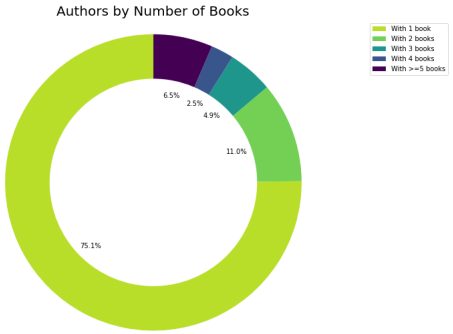

Figure 7: Authors by Number of Books

From observing **Figure 7** we conclude that the majority of authors present in the dataset (75.1%) only published 1 book. Since the dataset contains 35,347 books and 19,171 authors, this means 14,405 books were written by authors that only wrote 1 book. Therefore, 20,942 books, which is more than half of the books dataset (59.2%), were written by only 4,781 authors, which averages at about 4/5 books per author, despite that, according to the pie chart, only 6.5%+2.5%=9% (1716) of authors have written more than 3 books. This might happen because of the large percentage (46.2%) of books that belong to a series and are, consequently, written by the same author.

E. Book Descriptions and Reviews

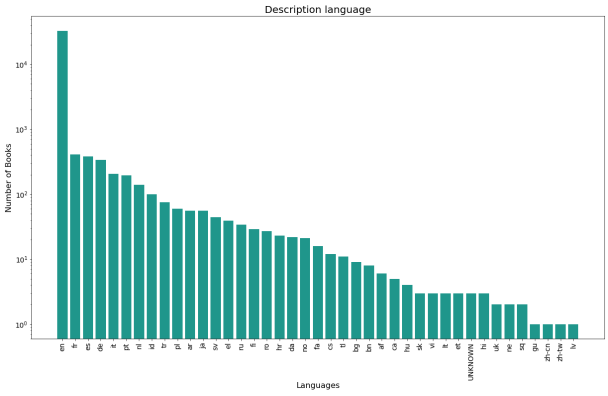

Figure 8: Book Descriptions Language

In **Figure 8**, we explore the different languages the descriptions were written in. The language for the descriptions was detected using the Python library *spacy-langdetect*[4]. From the chart, which uses a logarithmic scale, we conclude that the vast majority of descriptions are in English, with French, Spanish, German and other languages following after with less descriptions. Analyzing the book description language was an attempt of obtaining a rough approximation of each book's original idiom, although some of the descriptions end up being translated to English in the website, especially for popular books.

Figure 9: Book Reviews Word Cloud

For the book reviews, we decided to generate a word cloud plot (**Figure 9**) to try and see their general tone. Naturally, there are a lot of words associated with the elements of books such as "read", "character", "story" or even the word "book" itself. What is interesting is the large amount of positive words, such as "love", "good", "great" or "well", which might be related to the high rating distribution the books in the dataset have, as analysed before.

6 POSSIBLE SEARCH TASKS

Following an extensive data analysis of the main features that made the group acknowledge the dataset's relevant information, we can now consider some possible search tasks to be used:

- Search for the biography of an historical figure;
- Search for comic books featuring a specific character as the protagonist;
- Search for non-fictional books about the research and work of one of history's greatest minds;
- Search for fantasy children's book that are easy to read.

7 INFORMATION RETRIEVAL TOOL SELECTION

After an initial examination, the two main tools that were considered for Information Retrieval were *Apache Solr*[23] and *Elasticsearch*[10]. Both of these tools are open-source, are built on top of *Lucene*[2] and offer a great variety of features, for instance, distributed full-text search, near real-time indexing, high availability and support for *NoSQL* data.

Despite being the older tool, *Solr* ended up being our final choice. This is because *Solr* is best suited for search applications that use significant amounts of static data, which perfectly fits the problem at hand, as described in the previous sections.

While Solr has a lot of advantages and useful features, such as support for rich-text documents and complex search queries, it also has a significant disadvantage: its poor documentation and lack of examples/tutorials. This was especially troublesome because most of the already lacking examples and documentation were all directed towards schemas in *XML*, making it harder for the group to implement the schema using *JSON*.

8 INFORMATION RETRIEVAL INTRODUCTION

The main goal of an Information Retrieval system is to find documents within a collection that are relevant in order to satisfy a user

information need, while retrieving as few non-pertinent documents as possible. This research work specifically focuses on the most common retrieval task: *ad hoc* search, where the information need is provided through a user-initiated query that is performed on all documents of the collection.

The chosen tool for this project, *Solr*, offers various features which enhance the retrieval process. The platform allows the definition of custom filters for different fields in order to create dynamic searches, while also providing the possibility to grant weights to specific fields to boost their relevance. Having said that, to study and evaluate the results of possible search tasks, the following three systems will be utilized:

- A baseline system, which only performs basic matching on key fields;
- A system where the main textual fields are indexed with the usage of filters;
- A system that uses weights in order to boost relevant fields in the retrieval process.

Moreover, a set of information needs, defined in Section 6, will be translated to textual queries and tested in each system. Each of these queries will then be evaluated based on their Precision @ 10 (*P@10*)[8][6] and Average Precision (*AvP*)[8][6] values. *P@10* measures the precision, translating into the fraction of relevant retrieved documents, and *AvP* is the average of the precision values obtained for the set of the top K documents existing after each relevant document is retrieved.

Finally, to compare the global performance of all three systems we will use the Mean Average Precision (*MAP*)[8][6], which is obtained by averaging the *AvP* values off all the queries in a system.

9 COLLECTIONS AND DOCUMENTS

The relevant data for the information system is mainly split in two .csv files, regarding the books and their reviews. There were two approaches we considered in order to insert this data into *Solr*: the first one consisted in creating two types of documents, books and reviews, in the same core, creating the necessity of using several schemas or a single complex schema in order to be compatible with the different types of data. For this approach, we were also forced to use the *join* operation in queries, which wasn't ideal; the second one involved having only one type of document containing the books and their reviews in one of the fields, allowing the use of a single flexible schema compatible with the different types of data. However, merging the reviews into one field of the books dataset meant that we could only use the text content field from the reviews, having to discard the author and the date, which wasn't an issue since they weren't relevant for searching. Since the first approach made querying much harder, we decided to opt for the second approach. With this in mind, the two .csv files were merged and converted to a *JSON* file, so they could be put into *Solr* along with the flexible schema created, making the information ready to be queried on.

10 INDEXING PROCESS

One of the most crucial stages in Information Retrieval is Indexing, which minimizes the documents to the relevant terms contained in them. Before beginning this process, we analyzed which fields from each document were appropriate for indexing. For a field to be suitable it has to be useful for searching books, fulfilling the previously defined information needs, by serving as a query parameter. The documents were indexed using *Solr's* Post tool and the final schema can be consulted in **Table 14**.

Considering the project's context, it is important to modify *Solr's* indexing pipeline for the fields with unstructured data. The fields with native values were defined using the default *Solr* field types, such as *string*, *pfloat* and *pint*. Furthermore, all of the important text fields e.g., title, series, author, description and reviews were defined using an analyzer pipeline, which contained both tokenizers and filters for the enhanced processing of each individual token.

Having said that, a new field type for this textual content, ***standard_text***, was created using the following filters:

- *ASCIIFoldingFilterFactory*, which converts alphabetic, numeric, and symbolic *Unicode* characters which are not in the Basic Latin *Unicode* block to their *ASCII* equivalents, if one exists.
- *LowerCaseFilterFactory*, which converts any uppercase letters in a token to the equivalent lowercase token.
- *SynonymGraphFilterFactory*, which does single- or multi-token synonym mapping, producing a fully correct graph output. Each token is looked up in the list of synonyms and if a match is found, then the synonym is emitted in place of the token.

This custom field type uses these filters for both indexing and querying, while also making use of *Solr's Standart Tokenizer*, which splits the text field into tokens, treating whitespace and punctuation as delimiters.

11 RETRIEVAL PROCESS

In order to evaluate the performance of the different systems detailed in Section 7, as well as the impact of filters and weights/boosts, we defined four different information needs. Each information need contains a basic description, a user story to explain its context, its relevance judgement to settle if a document is relevant or not and the query that represents it. For the query section, the meaning of the abbreviations used for the *eDismax* parameters is the following:

- **Query (q)**;
- **Filter Query (fq)**;
- **Query Field with optional boost (qf)**: The fields where the search is made;
- **Phrase boosted Field (pf)**: Assigns a boost depending on the proximity of the searched words;
- **Phrase boost Slope (ps)**: Maximum amount of tokens wanted between between the searched words.

With the goal of evaluating each system's performance, for each information need we analyzed the top 10 query results and their relevance, allowing us to calculate the *P@10* and *AvP*, and draw conclusions based on these values afterwards.

One important detail to note is that the query fields vary with each information need. These fields and their respective weights (for System 3), which were determined by following an *Ad Hoc* approach, are presented before every information need.

A. Bibliographies of Adolf Hitler

Table 1: Weights Distribution (System 3)

Field (qf)	Weight
title	3
genre	2
description	1
author	3

Information Need: Biographies about Adolf Hitler.
User Story: "As a History student studying about World War II, I want to find books and biographies that portray the life of Adolf Hitler".
Relevance Judgement: In this information need we intend to search for books that talk about Adolf Hitler, more specifically, biographies about the dictator. Since the book is a biography, the book has to have "Biography" as one of its genres or have the word "biography" in the title or description. It is also important for "Hitler" to be mentioned in the book's title or be the book's author. Mentions of this word may also appear in the description, despite having less importance. With this in mind we distributed the weights as seen in **Table 1**.
Query:

- **q**: hitler and biography
- **qf**: title, genre_and_votes, description, author

Table 2: Bibliographies of Adolf Hitler Information Need Results

Rank	System 1	System 2	System 3
1	R	R	R
2	N	N	R
3	R	R	R
4	R	R	R
5	N	R	R
6	R	R	R
7	R	R	R
8	N	N	N
9	N	N	R
10	R	R	N
AvP	0.732937	0.801020	0.986111
P@10	0.60	0.70	0.80

Result Analysis: The second system has better performance than the first system, with the third system having the best results as we can see in **Table 2** and **Figure 15**. This last system has a better average precision than the other two, which means that relevant results tend to appear sooner. Since the word "Hitler" appears in many descriptions of books about, e.g., the Second World War or the Holocaust, Systems 1 and 2, that give a book's description the same importance as the title or the author will be more likely to show these irrelevant results. On the other hand, System 3, that weighs the fields in favor of the book's title, genre and author, will certainly find biographies written by or about Hitler more easily.

B. Comic books whose protagonist is Spider-Man

Table 3: Weights Distribution (System 3)

Field (qf)	Weight
title	3
genre	2
description	1
characters	2

Information Need: Comic books featuring Spider-Man as the main character.
User Story: "As a Spider-Man fan, I want to find comic books whose protagonist is Spider-Man".
Relevance Judgement: For this information need we intend to retrieve comic books in which Spider-Man appears as the main character. Therefore, the words "Spider-Man" (or its synonyms) or "Peter Parker" should most importantly appear in the title, which most likely means the character is the protagonist. The superhero's name may also appear in the description or in the "characters" field, although with less importance than the title. Finally, since we're searching for a comic book, one of the book's genres must be "Comics". The weights for each query field are presented in **Table 3**.
Query:

- **q**: comics AND ("spider-man" OR "Peter Parker")
- **qf**: title, genre_and_votes, description, characters

Table 4: Comic books whose protagonist is Spider-Man Information Need Results

Rank	System 1	System 2	System 3
1	R	R	R
2	R	R	R
3	R	R	R
4	R	R	R
5	R	N	R
6	R	R	R
7	R	N	R
8	R	R	R
9	N	R	R
10	R	N	R
AvP	0.99	0.91	1.00
P@10	0.90	0.70	1.00

Result Analysis: The three systems have similar average performances, with the second system having the lowest score, as it can be seen in **Table 4** and **Figure 16**. Although we were expecting very high results, since we were using the book's title as a search field, which usually features the main character's name, some books where Spider-Man appears as a side character still appeared, not being so relevant for the goal of the query. The third system has a better performance since it assigns a higher weight to the title and characters, along with the use of the synonyms file. System 2 has a slightly lower performance than System 1 because the use of synonyms alone, especially in the book's description, can be misleading.

C. Non-fictional books about the life's work of Albert Einstein

Table 5: Weights Distribution (System 3)

Field (qf)	Weight
title	3
genre	2
description	1
author	3
characters	3

Information Need: Non-fictional books about the research and studies of Albert Einstein.
User Story: "As a person investigating Albert Einstein's work, I want to find non-fictional books about his research, both Scientific and Philosophical".
Relevance Judgement: With this information need we want to find books that detail Albert Einstein's career work, be it Scientific or Philosophical. First of all, because we want books that explain and explore Einstein's work, the scientist should either be the book's author or have his name feature in either the title or the book's characters (or both). The name can also appear in the description although it is less relevant. Additionally, since we are looking for non-fictional books, this genre must be part of the book's genres, along with "Science" or "Philosophy". Having said that, the weights for each query field are detailed in **Table 5**.
Query:

- **q**: einstein AND -fiction AND (philosophy OR science)
- **qf**: title, genre_and_votes, description, characters

Table 6: Non-fictional books about the life's work of Albert Einstein Information Need Results

Rank	System 1	System 2	System 3
1	N	N	R
2	R	R	R
3	R	R	R
4	N	N	N
5	R	N	R
6	N	R	N
7	R	R	N
8	N	N	N
9	N	N	N
10	R	R	R
AvP	0.567619	0.547619	0.86
P@10	0.50	0.50	0.50

Result Analysis: The third system has a better average precision than the other two systems, with the second system having a slightly lower but similar score to the first system, as we can observe in **Table 6** and **Figure 17**. Einstein's name comes up frequently in books, being often referred to as an inspiration for modern scientists or even used to describe an intelligent person. A system that focuses more on the books' titles and their authors will show relevant results sooner, which is what System 3 does. The first and the second systems give the same importance to the books' title, genre, description and author, frequently "catching" books that mention "Einstein" outside of the information need's context, resulting in worse performance.

D. Fantasy children's books set in a medieval era that are easy to read

Table 7: Weights Distribution (System 3)

Field	Weight
reviews (qf)	2
reviews (pf)	5
reviews (ps)	5

Information Need: Easy to read children-friendly books set in the medieval era.
User Story: "As a parent, I want to find children-friendly books set in the medieval era that are easy to read".
Relevance Judgement: For this information need, relevant books portrait easy-to-read fantasy children's books, set in a medieval era. First of all, the relevant books must feature "Fantasy" and "Childrens" as two of their genres. Since the books are set in a medieval era, the description must mention the word "kingdom". Finally, pertinent books must also include the words "easy" and "read" with at most five words of distance (possible adjectives or connectors in between) in at least one of their reviews. The weights for each query field are detailed in **Table 7**.
Query:

- **q**: easy read
- **fq**:
  - **genre_and_votes**: fantasy
  - **genre_and_votes**: childrens
  - **description**: kingdom
- **qf**: reviews
- **pf**: reviews

Table 8: Fantasy children's books set in a medieval era that are easy to read Information Need Results

Rank	System 1	System 2	System 3
1	N	N	N
2	N	N	R
3	N	N	R
4	N	N	N
5	N	N	N
6	R	R	R
7	N	N	N
8	N	R	N
9	R	N	N
10	R	R	N
AvP	0.229630	0.238889	0.555556
P@10	0.30	0.30	0.30

Result Analysis: Once again, the third system has the best performance compared to the other two, as we can see in **Table 8** and

Figure 18. In order to find books that are "easy to read" we are required to search beyond the book's title, description or genre, since these don't usually offer an opinion on the readability or quality of a book. Therefore, to obtain this information, we have to search in the users' reviews. The first two systems cannot match the words "easy" and "read" with adjectives or connectors between, e.g., "easy to read" or "easy and fast read". However, System 3 can match strings with up to five words between "easy" and "read", assigning different weights based on the number of words between, making it better for the goal of this query. It is also important to note the outstanding drop in *AvP* and *P@10* scores compared to the other information needs, in all three systems. This happens because, although the word "kingdom" is often used in medieval-themed books, it also covers a wide range of other contexts (e.g., kingdom of magical beings or kingdoms that existed before or after the medieval era) which, in this case, is misleading and leads to worse results.

E. General Conclusions

Taking into account all the results from the multiple information needs across the different systems, we can now calculate the *MAP* (Mean Average Precision) of these three systems.

Table 9: *MAP* values for the three Information Systems

System 1	System 2	System 3
0.630047	0.624382	0.850417

Table 9 contains the *MAP* values for each system, and will allow us to compare the systems on a more general scope in order to get a better overview of their performance.

By observing the results, we can conclude that System 3 is an improvement of both Systems 1 and 2, which was expected because of the schema and different weights assigned to each field for each query, allowing a more accurate and precise search depending on the context of the information need. We can also see that the results from System 1, which is schema-less, are slightly better than the results from System 2. Although there isn't a huge difference, we can conclude that a system with this particular schema, while being better at finding and matching keywords, is more vulnerable to ambiguity, frequently matching keywords in fields that aren't that relevant for that particular information need (since there is no weight), which is precisely what we can observe in the information needs present in Subsections B and C, where the queries matched more keywords in the description than in the title, thus being less relevant.

12 INFORMATION RETRIEVAL TOOL EVALUATION

Following the Information Retrieval phase, we have some comments on *Solr*, the tool used for this process:

- As mentioned before, *Solr's* documentation is very limited, making it difficult to learn how to perform certain tasks since there are very few practical examples and tutorials, especially for the chosen schema format: *JSON*
- The initial setup and configuration of the tool was not user-friendly, with the lack of any clear documentation making it more difficult.
- However, once we got the platform setup correctly, *Solr* offers many useful features, allowing the definition of complex queries while having a very fast query response time.

Overall, although *Solr* does take some time to learn and configure, it still allows for the implementation of a good Information Retrieval system, as the obtained results show.

13 SEARCH SYSTEM IMPROVEMENTS

There are many different approaches to improving a search engine, thus, we had to consider which enhancements made sense in the context of this project. Some of the initially considered improvements included: adding new ways to search and query the dataset (e.g., search by ranges), diversifying and expanding the collected documents, and developing a graphical user interface that integrates the search engine. After a thorough analysis, the developed improvements for the search system, that are explained and detailed throughout this section, are the following:

- Enhanced Synonyms;
- OpenNLP Toolkit;
  - Tokenizers;
  - Filters;
  - Named Entity Recognition;
- Learning to Rank/Page Rank
- Multiple Core Search;
- Graphical User Interface.

In order to analyze these improvements, we need to use a metric that can serve as a guide for the development process and that can be used to draw new conclusions. Therefore, we used the same information needs and evaluation metrics present in Section 11 in order to create a valid before-after comparison scenario.

Essentially, we will first describe how each improvement was implemented and how they function in the search system. From there on we will perform queries on the improved search system and evaluate its performance by comparing the obtained results to System 3's results, also present in Section 11.

13.1 Enhanced Synonyms

In the Retrieval Process detailed in Section 11, we applied a simple synonym filter to the information need present in Subsection B. This synonym filter simply matched "Spider-Man" to other variants of the word, and while it showed good potential by allowing better token matching, it didn't have much use outside of this specific information need, making this synonym matching system not very useful for other distinct information needs.

With this in mind, we decided to expand on this system, allowing our search system to retrieve useful and relevant documents that would normally be deemed as irrelevant by Solr, due to it only being capable of matching the exact word searched. In order to build a useful synonym filter for a great range of information needs, we had to maximize the number of synonyms that were provided to this filter. To achieve that, we used the *NLTK* (*Natural Language Toolkit*)[5] library in Python, which provides several text processing tools with a good amount of test datasets.

Initially, all the nouns present in the dataset's textual content (e.g., book's title and description) were extracted in order to build a set from where we would generate synonyms to be incorporated into the search system. After storing all these nouns in a list, we used *NLTK* and iterated through this set, generating synonyms for each unique noun present in it.

The final result was a list with synonyms for 21,561 unique nouns, to be used in the books' fields. This list was stored in a text file *synonyms.txt*, to later be integrated with the already mentioned *SynonymGraphFilterFactory* filter in the schema file.

13.2 *OpenNLP* Implementation

The *Apache OpenNLP*[3] library is a machine learning based toolkit for the processing of natural language text. It supports the most common NLP tasks, although the ones we are focusing on are functions such as tokenization, sentence detection/segmentation, part-of-speech tagging, named entity extraction and chunking.

We decided to use this library mainly because it is included in the standard *Solr* distribution, even though, by default, it is disabled. In order to enable this library, originally located in *Solr's contrib* module, all we need to do is include it in the search platform's configuration file. After concluding this process we now have access to *OpenNLP's* tokenizers, filters and processors.

It is important to note that all the machine learning models used with the *OpenNLP* modules are trained for and only work with English text. Although, as was observed in the Data Characterization Section, our dataset contains some books with non-English descriptions, the vast majority of the dataset's books have descriptions in English. Therefore, there is no need to add extra complexity to the system by adding more *OpenNLP* models to support additional languages.

Lastly, all the pre-trained machine learning models used in the *OpenNLP*-related enhancements were downloaded from the *SourceForge OpenNLP*[20] website.

A. Tokenizers

Importing the *OpenNLP* library allows us to modify the schema in order to perform a more complex tokenization process in fields with substantial text content (e.g., book's description and reviews). This procedure is done by the *OpenNLPTokenizer* class, that does tokenization and sentence detection by using the machine learning models *en-token.bin* and *en-sent.bin*, respectively.

B. Filters

In addition to the tokenizers, the *OpenNLP* library also provides us with many useful filters. In this context, we decided to use filters that allowed part-of-speech tagging and chunking.

The concept of part-of-speech tagging consists in marking tokens with their corresponding word type based on the token itself and the context of the token. The Tagger that performs this operation is implemented by the *OpenNLPPOSFilterFactory* class and uses a probability model to predict the correct POS (part-of-speech) tag out of the tag set. Therefore, we integrated this filter into the system's schema and used it with the pre-trained machine learning model *en-pos-maxent.bin*.

The resulting tags from part-of-speech tagging can then be used to perform chunking, which consists of dividing a text in syntactically correlated parts of words, such as noun groups or verb groups. The Chunker Tool which executes this task is integrated by the *OpenNLPChunkerFilterFactory* class and, similar to the POS Tagger, uses a machine learning model in order to correctly identify chunks in text. Thus, we implemented this filter in our schema and used it along side the machine learning model *en-chunker.bin*.

C. Named Entity Recognition

Another *OpenNLP* tool we used was NER (Named Entity Recognition), which consists in detecting and categorizing entities in a text field through the use of a machine learning model. This model is dependent on the language and entity type it was trained for. In this book-focused context, we decided that it made sense to add the three following models:

- **Person name finder model**: In order to find a book through its characters more easily and complementing the already present "characters" text field;
- **Location name finder model**: With the goal of facilitating a book search by its locations/settings and complementing the already present "settings" text field;
- **Date name finder model**: To make a book search by a certain period in time (e.g., a book whose story unfolds in the 90s) find matches more easily.

There were other interesting models we considered (e.g., Organization name finder model), however, adding more models meant increasing the system's complexity and could lead to an increase in the number of ambiguous terms being matched.

After choosing the relevant entities and their models, we implemented NER by adding an *OpenNLP* processor class to the default update request processor chain, that handles entity extracting from text. Then, comparably to what was done for the *OpenNLP* Tokenizers and Filters, we integrate the needed pre-trained machine learning models into the system, in this case: *en-ner-person.bin*, *en-ner-location.bin* and *en-ner-date.bin*.

In conclusion, these three models enrich the dataset by creating new text fields containing the entities they identified, providing better matching when searching for these specific entities.

13.3 Learning to Rank/Page Rank

In Information Retrieval systems, Learning to Rank (LTR) focuses on the concept of Query Re-Ranking, which allows us to run a simple query for matching documents and then re-rank the top N documents using trained machine learning models and the scores from more complex queries. The hope is that such sophisticated models can make more nuanced ranking decisions than standard ranking functions like TF-IDF[22] or BM25[16], used by *Solr*. This re-ranking process is done by using a ranking model, with *Solr* supporting three different models by default: the *Linear Model*, the *Multiple Additive Trees Model* and the *Neural Network Model*. The simplest model, and the one we will use for this project, is the *Linear Model*, which simply computes scores by using the dot product between the selected features and their respective weights.

After selecting our ranking model, we can now move on to feature selection and model training, which take place offline and

outside *Solr*. For the feature engineering, we decided that the selected features and the consequent query re-rank should should focus on a book's secondary aspects (e.g., its number of ratings which translates to the book's popularity as analyzed in Section 5) rather than its primary characteristics (e.g., the book's title), which already have a considerable weight assigned to them in the main query. With this in mind, we selected the following features:

- ***maximize_rating***: Which gives a higher score to books with a higher number of ratings (*rating_count*);
- ***maximize_average_rating***: That prioritizes books with a higher average rating (*average_rating*);
- ***maximize_recommendation_number***: Which grants books with a higher number of recommendations a higher score (*times_recommended*);
- ***description_bm25***: This feature gives books that match the searched term more frequently in the description a higher score;
- ***original_score***: That keeps the main query's original score.

All the mentioned features were then integrated into *Solr* by using the *org.apache.solr.ltr.feature.SolrFeature* class, with the exception of the *original_score* feature, which was integrated by using the *org.apache.solr.ltr.feature.OriginalScoreFeature* class.

It should be noted that the ***maximize_recommendation_number*** feature uses a newly added field: *times_recommended*. This field contains the number of times a book was "recommended" by other books, that is, the number of times the id of that book appeared in the *recommended_books* field of other books. This is similar to how Page Rank prioritizes certain websites by counting the amount of links pointing to them, which means that integrating this feature with LTR allows us to have a basic Page Rank implementation in our system.

Following feature extraction, we are now able to train the machine learning model. Although similar linear models are usually trained using a pointwise approach (i.e., the model tries to identify, in the context of a query, if a document is relevant or not), we will instead be using a pairwise approach (i.e., the model, given a pair of documents, tries to find the best ordering for that pair), which works better in practice than pointwise approaches because predicting relative order is closer to the process of ranking than the process of predicting an exact class label or relevance score.

Therefore, since we will be using a pairwise approach for the training phase, we opted to use SVMRank[21], a variant of the Support Vector Machine[11] that transforms this ranking problem into a binary classification problem. To implement SVMRank, we used the scikit-learn[7] Python library, which allows us to find an optimal weights vector for the selected features that will consequently result in the best possible ordering for the top N relevant documents.

For the data used in the model training, we decided to use the information needs from Section 11 and selecting the relevant documents for each query by analyzing all the retrieved results. However, the high class imbalance (with more irrelevant documents than relevant documents) and the short amount of examples results in a model that might be subject to both underfit and overfit. This is an issue that can only be mitigated by giving the model considerably more data, which isn't possible in this context.

Finally, with the model trained and ready, LTR can now be implemented in Solr by integrating the resulting weights vector into the Linear Model in *Solr*.

13.4 Impact of these Improvements in Performance

With all the performance-impacting improvements explained, we can now evaluate whether or not these enhancements improved our search system's performance. For this evaluation, we decided to use System 3's results from Section 11, and compare them to the Improved System's results for the same information needs. These information needs will be the same as the ones in Section 11, with some minor modifications on the query syntax for the Improved System, to allow it to utilize some of its new features (e.g., NER fields).

With that said, since Section 11's information needs were already addressed, we withdrew some of their content from this section. Thus, we will only include their titles and respective queries.

Finally, similarly to the procedures taken in Section 11, we will perform an analysis of the top 10 query results and their relevance for each information need. These results will then allow us to calculate the *P@10* and *AvP* and take conclusions based on these values.

A. Bibliographies of Adolf Hitler
Query:

- **q**: hitler and biography
- **qf**: title, genre_and_votes, description, author

Changes: Added rerank query (rq) **{!ltr model=my_efi_model efi.text="hitler biography"}** to allow LTR.

Table 10: Bibliographies of Adolf Hitler Information Need Results

Rank	1	2	3	4	5	6	7	8	9	10
System 3	R	R	R	R	R	R	R	N	R	N
Improved System	R	R	R	R	N	N	R	R	R	R

Score	**AvP**	**P@10**
System 3	0.986111	0.80
Improved System	0.880258	0.80

Result Analysis: System 3 and the Improved System have the same *P@10*, as we can observe in **Table 10** and **Figure 19**, which means both systems matched the same number of books in the top 10 results. In this same table we can also observe that the average precision is higher in System 3 compared to the Improved System. In our dataset, there are a lot of World War II biographies (e.g., from Allies' soldiers that fought the Nazis or from the Jews that were trying to survive the holocaust) that reference Adolf Hitler due to his role in the war. These types of biographies aren't really relevant for this specific information need since we're looking for biographies that are specifically about Hitler. Therefore, since the Improved System uses improved *OpenNLP* tokenization/chunking and synonyms, which associate the word "Hitler" to other words

(e.g., "Fuhrer") the query will more often match these irrelevant biographies, resulting in a worse overall performance than System 3.

B. Non-fictional books about the life's work of Albert Einstein
Query:

- **q**: einstein AND -fiction AND (philosophy OR science)
- **qf**: title, genre_and_votes, description, characters

Changes: Addition of the query field (qf) ***ner_person_field*** to complement the already present *characters* field. Added rerank query (rq) **{!ltr model=my_efi_model efi.text="einstein science philosophy"}** to allow LTR.

Table 11: Non-fictional books about the life's work of Albert Einstein Information Need Results

Rank	1	2	3	4	5	6	7	8	9	10
System 3	R	R	R	N	R	N	N	N	N	R
Improved System	R	R	N	N	N	N	R	N	N	N

Score	**AvP**	**P@10**
System 3	0.86	0.50
Improved System	0.809524	0.30

Result Analysis: By observing **Table 11** and **Figure 20** we can once again see that System 3 has a better performance than the Improved System. Another important aspect to note is that System 3 also matched more relevant books in its top 10 results than the Improved System, since its *P@10* is higher. In the analysis done in **Section 11 Subsection C**, where we explored this information need across the three different systems, we concluded that there was some ambiguity occurring with the word "Einstein", since it was often mentioned and associated with the concept of intelligence. This same problem also happens with the synonyms list used by the Improved System, since it includes words such as "genius", "brain" and "mastermind" as synonyms to the word "Einstein". This allows non-relevant books containing these words in the title or in the description to appear. Thus the Improved System's performance is harmed not only by being more sensible to the "Einstein" word ambiguity in text, since it has a more complex tokenizer, but also by having "Einstein" associated with words unrelated to Albert Einstein in its synonyms list. It is important to note however that the use of the extra field generated by NER in this query (*ner_person_field*) did help mitigate this problem by reducing the number of books containing the mentions of the word "Einstein" outside of this information need's context, however, it is impossible for NER to detect all these cases perfectly.

C. Fantasy children's books set in a medieval era that are easy to read
Query:

- **q**: easy read
- **fq**:
  - **genre_and_votes**: fantasy
  - **genre_and_votes**: childrens
  - **description**: kingdom
- **qf**: reviews
- **pf**: reviews

Changes: Added rerank query (rq) **{!ltr model=my_efi_model efi.text="kingdom"}** to allow LTR.

Table 12: Fantasy children's books set in a medieval era that are easy to read Information Need Results

Rank	1	2	3	4	5	6	7	8	9	10
System 3	N	R	R	N	N	R	N	N	N	N
Improved System	R	R	N	R	N	R	N	R	N	N

Score	**AvP**	**P@10**
System 3	0.555556	0.30
Improved System	0.808333	0.50

Result Analysis: In this information need, the Improved System has a better performance than System 3, as we can see in **Table 12** and **Figure 21**. During **Section 11 Subsection D**, in which we analyzed this information need for the three different systems, we concluded that all systems had a drop in score due to the word "kingdom" being vague relatively to the context of the information need; however, the Improved System doesn't seem to be affected by this, since it achieved a high score. This mainly happens because of the enhanced synonyms implementation on the Improved System, which, in addition to the word "kingdom", matches synonyms that allow more medieval books to be shown in the results (e.g., realm, empire and duchy). Therefore, this wider range of results, paired with the improved *OpenNLP* tokenization and LTR, that narrows down these results, is what causes this increase in performance.

E. General Conclusions
Considering all the results from the multiple information needs, we can now calculate the *MAP* (Mean Average Precision) of these two systems and compare their overall performance.

Table 13: *MAP* values for the two Information Systems

System 3	**Improved System**
0.800556	0.832705

By observing **Table 13's** *MAP* values for each system, we can conclude that the Improved System is an enhancement over System 3. Although this was the expected outcome, since the Improved System contains a more advanced tokenizer and performs sentence detection, chunking and part-of-speech tagging, we can conclude by observing Subsections A and B that this system doesn't perform as well as System 3 for information needs containing ambiguous or specific terms. This mainly occurs due to the usage of synonyms, since the list containing them has synonyms for a considerable amount of words, therefore making the system more vulnerable to matching terms that don't have the same meaning as the original search. Nevertheless, the use of synonyms does boost performance in more "general" queries with vague terms, allowing a greater number of keywords to be matched and a consequent increase in

number of results, which are then rearranged and filtered by LTR, as was observed in Subsection C.

To conclude, the enhancements performed on this Improved Search System are working as intended, making the system perform better on unclear and faint queries and allowing the user to search for entities that weren't originally mentioned in the dataset's *characters* and *location* fields through the use of NER-generated fields.

13.5 Multiple Core Search

Although a multiple core search implementation makes querying more complex than a single core search implementation, it improves user experience in the developed search system by allowing a wider search. With that being said, we decided to create two cores, each with their own schema: the first one for the books and the second one for the reviews.

It is important to note that, even with two distinct cores, the main focus of the search engine is still the book search. To guarantee this, queries join the results from both the books' core and the reviews' core, showing the books that match the query fields from both cores as result, allowing the user to not only search by the book's fields but also search by the review's parameters, e.g., the review's text or the review's author.

13.6 Graphical User Interface

Since using the *Solr GUI* was the only method of interacting with the system, this made querying a difficult process for users that are not familiar with query syntax or HTTP requests.

Therefore, although it doesn't affect query performance, another important addition to the project was the creation of a Web Front-end, which greatly enriches the overall user experience and makes searching for a book an easier and clearer process.

In order to implement this interface we used *Node.js*[9] along with the solr-node[12] package, this provided us with an easy and flexible integration of the search queries on the Web Front-end. For the search user experience, the created design offers a layout that allows users to search for a book through its main features, either individually or simultaneously: the book's Title, Author, Genre, Series, Description, Characters and Settings, along with the possibility of also searching by the textual content of the book's Reviews and the Author of these reviews. This Web Front-end has 2 main pages in total, with both containing the previously mentioned search interface:

- **Main/Results Page**: Displays the results after a book search;
- **Book Page**, accessed by clicking a book in the results page - contains all the book information and its reviews.

An important thing to note is that the Book Page also allows users to instantly search for books with the same author and genres as the book featuring in that page, along with the possibility of immediately accessing that book's recommended and in-series books by clicking on them. Additional screenshots of the developed interface can be observed in the **Figures 22-26**

14 REVISIONS INTRODUCED

The goal of this section is to present the several modifications and corrections applied to the previous modules of this report.

We started by making some minor changes such as correcting the phrasing in some sections of the report and by adding missing references, thus granting better understanding and clarity of the presented concepts. Another important change was increasing the size of a few figures and graphs for better readability.

Nevertheless, we also made some important changes, particularly in the Search System Evaluation section, where we, in addition to the already present *AvP* and *P@10* values, included the relevancy classification for each result of the top 10 results of the query.

Finally, we remade the Precision-Recall curve calculations, which resulted in a change to their respective graphs. This implied a recalculation of the *P@10* and *AvP* values for all the queries, which meant that some of the previous conclusions had to be adjusted.

15 CONCLUSIONS AND FUTURE WORK

In this paper we presented and detailed the procedures done during the development of a book search system that aims to provide a richer search experience, letting its users search for books based on e.g., reviews, descriptions or awards.

In the initial sections of the paper we detailed the data characterization process that ultimately led to the creation of our dataset, explaining the operations done to both integrate different sources of information and to clean data. The result was a much better structured dataset that could be easily integrated into a search system.

The next stage focused on the Information Retrieval process. We began by researching about different search tools and comparing them to see which one fitted our problem better, with the final choice being *Solr*. Following that, we detailed our indexing process, where we explained the schema, filters and tokenizers we used in order to optimize the system and achieve better results. We then explored the actual retrieval process, where we selected four different information needs and compared their performance through the precision scores from the first ten documents across three different systems, revealing the performance boost the added schema and weights caused. We concluded that System 3 was having better results due to the performed optimizations.

We also explored the several implemented search system improvements, which were integrated with the goal of increasing the performance of the search engine. These improvements focused on various concepts, such as: the use of *OpenNLP* tools like filters, tokenizers and NER, the implementation of a LTR model, and lastly the use of a machine-learning-generated synonym list. The Improved System was also analyzed and compared to Section 11's System 3, having better overall performance in general queries with vague terms but worse performance when searching for ambiguous or specific terms, a trade-off mainly caused by the use of a synonym list with a great number of terms. Finally, with user experience in mind, we also implemented multiple core search and a web front-end for the search engine.

Regarding future work, we plan to further improve our search system, mainly by developing a more robust Page Rank implementation that takes into account the popularity of books in the book recommendations and also by improving the current ranking model used in LTR, in order to achieve a more sophisticated query re-ranking.

REFERENCES

[1] *Amazon Book Search.* URL: https://www.amazon.com/advanced-search/books (visited on 11/16/2021).

[2] *Apache Lucene - Welcome to Apache Lucene.* URL: https://lucene.apache.org/ (visited on 12/14/2021).

[3] *Apache OpenNLP - Welcome to Apache OpenNLP.* URL: https://opennlp.apache.org/ (visited on 01/20/2022).

[4] Abhijit Balaji. *Spacy Python Library.* URL: https://spacy.io/universe/project/spacy-langdetect (visited on 11/15/2021).

[5] Steven Bird et al. *NLTK: Natural Language Toolkit.* URL: https://www.nltk.org/ (visited on 01/20/2022).

[6] W. Bruce Croft et al. *Search engines : information retrieval in practice.* 2010. URL: https://catalogo.up.pt/F/?func=direct&doc_number=000567357 (visited on 01/20/2022).

[7] David Cournapeau. *scikit-learn: machine learning in Python — scikit-learn 1.0.2 documentation.* URL: https://scikit-learn.org/stable/ (visited on 01/20/2022).

[8] Christopher D. Manning et al. *Introduction to Information Retrieval.* 2008. URL: https://catalogo.up.pt/F/?func=direct&doc_number=000537893 (visited on 01/20/2022).

[9] Ryan Dahl. *Node.js.* URL: https://nodejs.org/en/ (visited on 01/20/2022).

[10] *Elasticsearch: The Official Distributed Search Analytics Engine.* URL: https://www.elastic.co/elasticsearch/ (visited on 12/01/2021).

[11] Rohith Gandhi. *Support Vector Machine — Introduction to Machine Learning Algorithms.* June 2018. URL: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47 (visited on 01/20/2022).

[12] Donghyun Go. *solr-node.* URL: https://www.npmjs.com/package/solr-node (visited on 01/20/2022).

[13] Goodreads.com. *Goodreads | Meet your next favorite book.* URL: https://www.goodreads.com/ (visited on 11/03/2021).

[14] Kaggle. *Your Machine Learning and Data Science Community.* URL: https://www.kaggle.com/ (visited on 10/27/2021).

[15] Wes McKinney et al. *Pandas Python Library.* URL: https://pandas.pydata.org/ (visited on 11/02/2021).

[16] *Okapi BM25 - Wikipedia.* URL: https://en.wikipedia.org/wiki/Okapi_BM25 (visited on 01/20/2022).

[17] Austin Reese. *Goodreads Books - 31 Features.* July 2020. URL: https://www.kaggle.com/austinreese/goodreads-books (visited on 11/02/2021).

[18] Kenneth Reitz et al. *Requests Python Library.* URL: https://docs.python-requests.org/en/latest/ (visited on 11/07/2021).

[19] Leonard Richardson et al. *BeautifulSoup Python Library.* URL: https://beautiful-soup-4.readthedocs.io/en/latest/ (visited on 11/07/2021).

[20] *SourceForge OpenNLP Tools Models.* URL: http://opennlp.sourceforge.net/models-1.5/ (visited on 01/20/2022).

[21] *SVM-rank: Support Vector Machine for Ranking.* URL: https://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html (visited on 01/20/2022).

[22] *tf–idf - Wikipedia.* URL: https://en.wikipedia.org/wiki/Tf%E2%80%93idf (visited on 01/20/2022).

[23] *Welcome to Apache Lucene.* URL: https://lucene.apache.org/ (visited on 12/14/2021).

[24] *WorldCat Book Search.* URL: https://www.worldcat.org/ (visited on 11/16/2021).

ATTACHMENTS

The following pages contain the attachments for all the mentioned figures in this paper, it also contains additional plots and tables used during research.

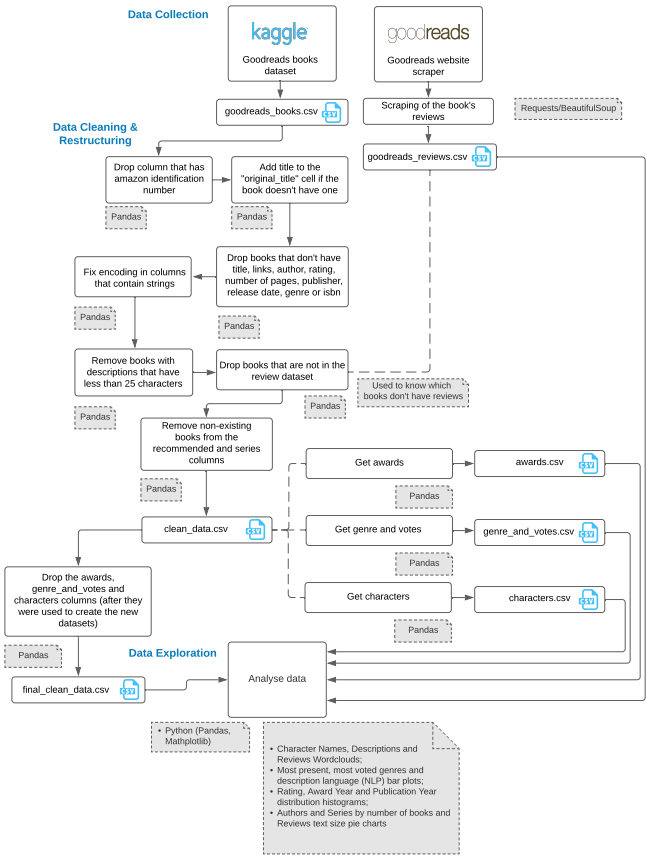

Figure 10: Data Pipeline

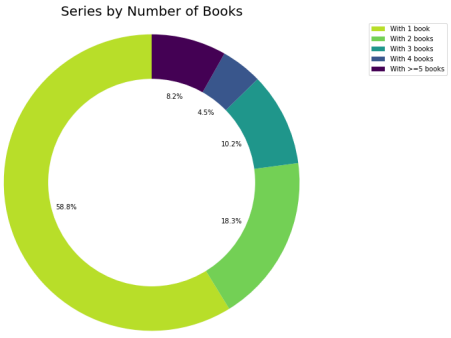

Figure 11: Series by Number of Books

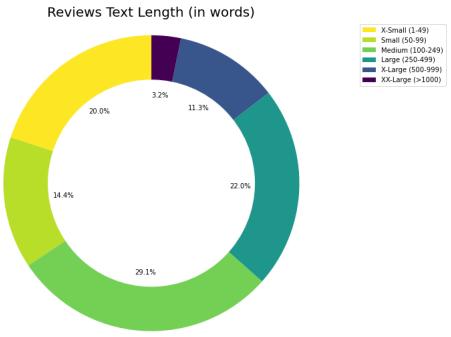

Figure 12: Reviews Text Length

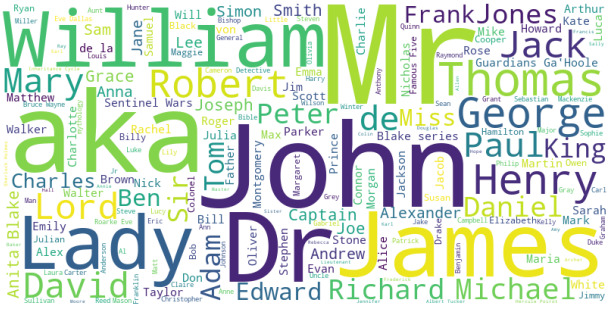

Figure 13: Book Characters Word Cloud

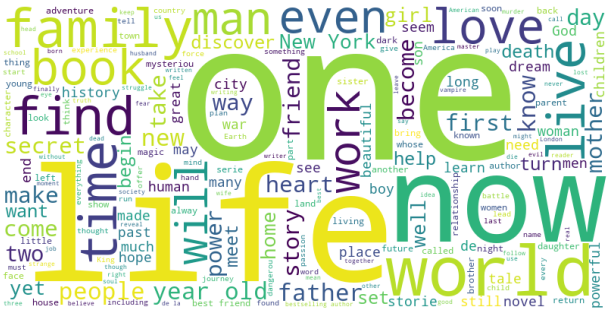

Figure 14: Book Descriptions Word Cloud

Table 14: Schema Field Types

Field	Type	Indexed
title	standard_text	Yes
link	string	No
series	standard_text	Yes
cover_link	string	No
author	standard_text	Yes
author_link	string	No
rating_count	pint	No
average_rating	pfloat	No
five_star_ratings	pint	No
four_star_ratings	pint	No
three_star_ratings	pint	No
two_star_ratings	pint	No
one_star_ratings	pint	No
number_of_pages	pint	Yes
date_published	standard_text	Yes
publisher	standard_text	Yes
original_title	standard_text	Yes
genre_and_votes	standard_text	Yes
isbn	string	Yes
isbn13	string	Yes
settings	standard_text	Yes
characters	standard_text	Yes
awards	standard_text	Yes
amazon_redirect_link	string	No
worldcat_redirect_link	string	No
recommended_books	standard_text	Yes
books_in_series	standard_text	Yes
description	standard_text	Yes
reviews	standard_text	Yes

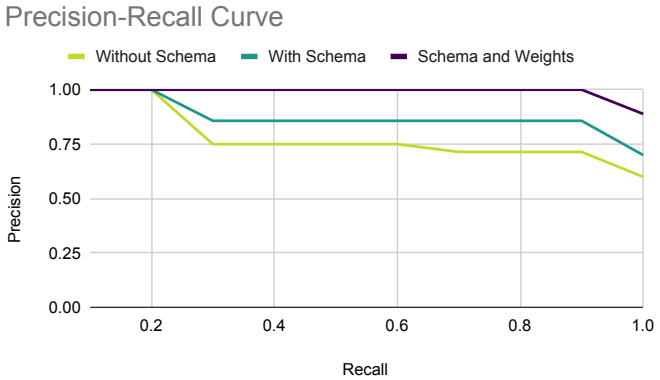

Figure 15: Bibliographies of Adolf Hitler - Evaluation on 3 Systems

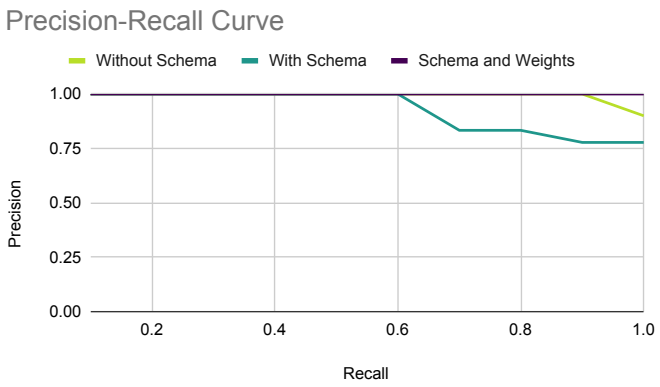

Figure 16: Comic books whose protagonist is Spider-Man - Evaluation on 3 Systems

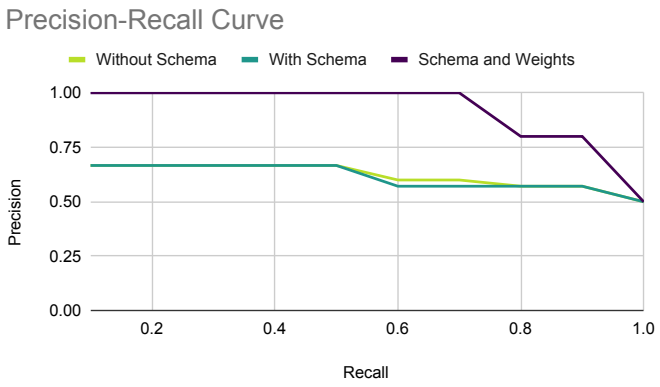

Figure 17: Non-fictional books about the life's work of Albert Einstein - Evaluation on 3 Systems

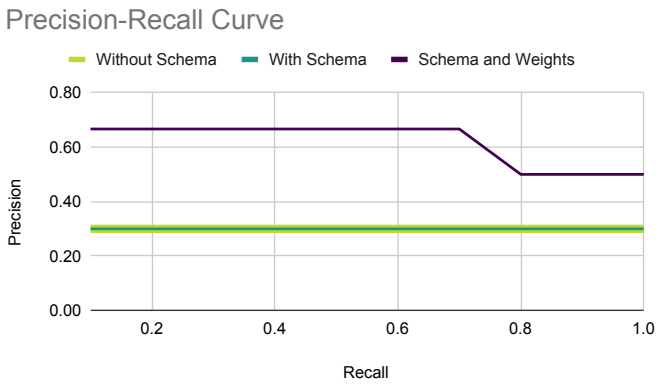

Figure 18: Fantasy children's books set in a medieval era that are easy to read - Evaluation on 3 Systems

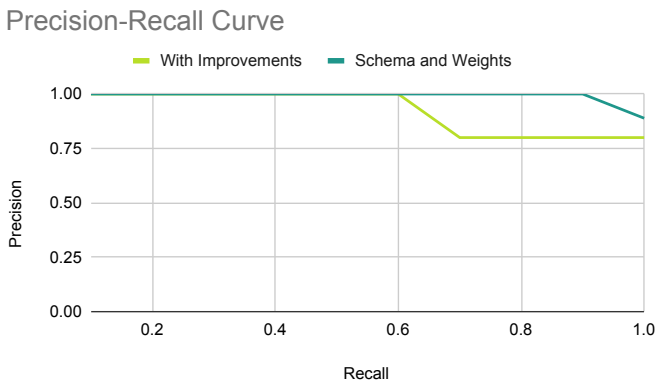

Figure 19: Bibliographies of Adolf Hitler - Evaluation on 2 Systems

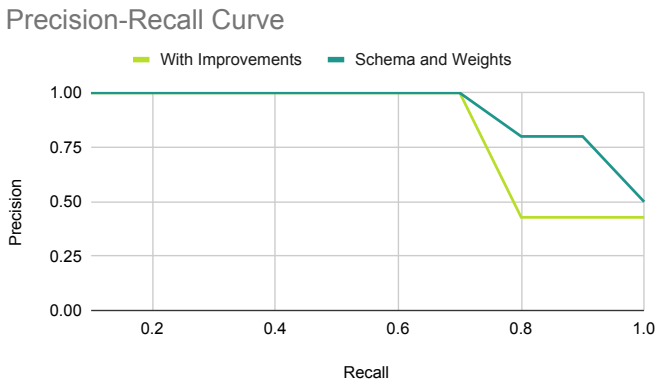

Figure 20: Non-fictional books about the life's work of Albert Einstein - Evaluation on 2 Systems

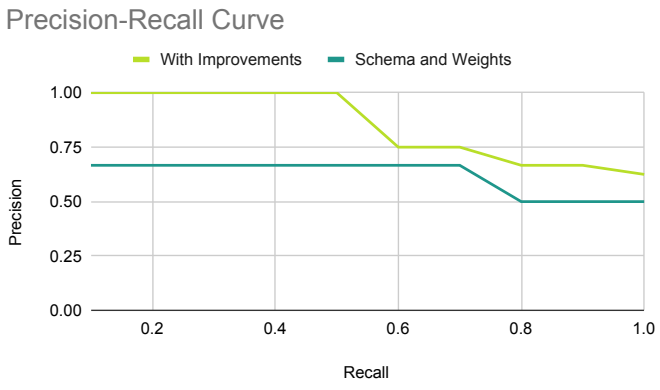

Figure 21: Fantasy children's books set in a medieval era that are easy to read - Evaluation on 2 Systems

Figure 22: Main Search Page

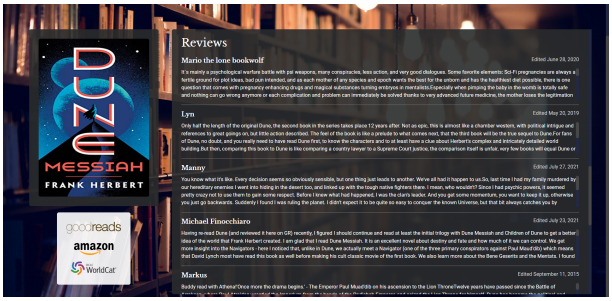

Figure 26: Book Page - Book Reviews

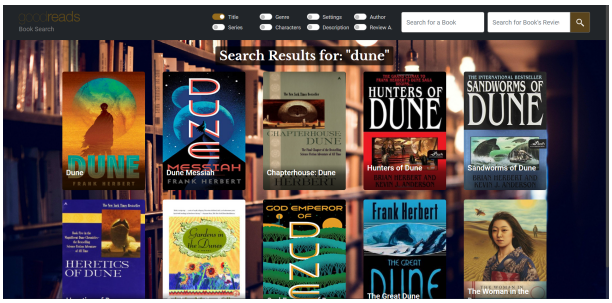

Figure 23: Main Search Page with Results

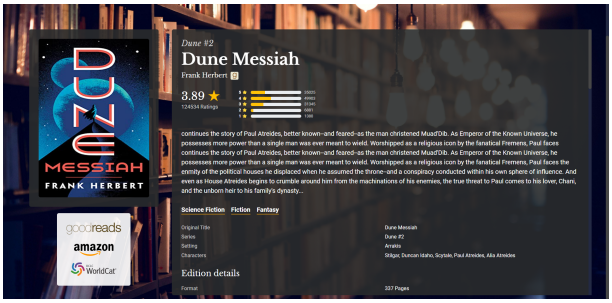

Figure 24: Book Page - Book Information

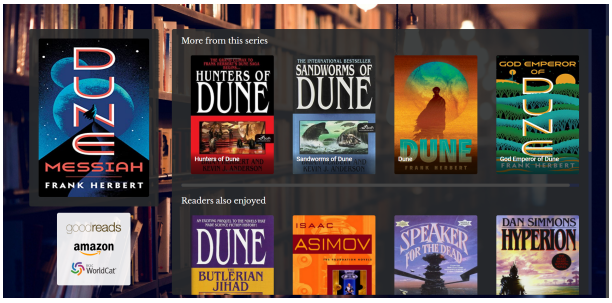

Figure 25: Book Page - In-series and Recommended Books