Book Characterization using Project Gutenberg's Open Library with Goodreads Reviews

João Baltazar
up201905616@up.pt
Faculty of Engineering of the University of Porto
Porto, Portugal

Nuno Costa
up201906272@up.pt
Faculty of Engineering of the University of Porto
Porto, Portugal

Pedro Gonçalo Correia
up201905348@up.pt
Faculty of Engineering of the University of Porto
Porto, Portugal

Abstract

Catalogs of books and other texts are more available now than ever, as online libraries aggregate public domain literature. As such, rich unstructured data is plentiful but mostly unexplored. In this paper, we seek to find and showcase that underlying information by characterizing books based on their theme, word count, time period, and popularity. By further extracting book corpora and characteristics from Project Gutenberg's Open Library and joining them with Goodreads' reviews, we aim to incorporate this data in the Apache Solr's search engine to fulfill search tasks such as finding specific excerpts or understanding the context for a given book quote. We compare multiple approaches with different characteristics by evaluating them on specific information needs to conclude which is preferable, and then expand on them with specialized, user-oriented features and search optimization through a backend and advanced Solr tools.

CCS Concepts: • **Information systems → Data mining**; **Information integration**.

Keywords: information theory, information retrieval, information processing, data characterization

1 Introduction

Data has become more and more accessible in recent years. Massive bodies of text, such as books, have become readily available in large online e-book libraries like Project Gutenberg's [12]. However, processing that same data and retrieving relevant information, such as patterns and trends, is neither automated nor easy. Finding context for a given book quote, for example, is a significant information need with a lackluster response, as misquotes are still frequently spread around. Open-source libraries do exist, but are not user-friendly, namely, with a dire lack of useful indexing and complementary information. This way, compiling and characterizing books with Project Gutenberg's Open Library and further joining them with Goodreads [10] Reviews and integrating them on the Apache Solr search engine [16], all behind a powerful user-friendly website, appeals to both the archivists and casual readers alike: a big collection of plain text literature, processed and fully searchable, and a hub of recommendations and free books, tagged with genres, themes, time periods and authors.

2 Dataset

The data retrieval pipeline is shown in Appendix A.

The produced dataset has 8000 book entries, making up around 3.2 GB of memory space, and 8300 review entries (10 reviews each for 830 books), making up around 18 MB of memory space. Each book entry includes the book transcription, as well as its bibliographic information. Of the 8000 book entries, around 2700 of them have the overall rating.

2.1 Data Collection and Processing

The first step to collect data was to scrape the catalog of the most popular books in Project Gutenberg, retrieving the identifiers of each book, in fetch_book_ids.py. For each id retrieved, the book information was scraped from the respective webpage. This information underwent several filtering and processing stages in different scripts. Duplicate books that appear as different versions in Project Gutenberg were removed, keeping the version marked as "improved" if such information exists, or the first version processed otherwise. Books that did not have English as their language was removed. To ensure that the scraped content is free to use, books that did not have "Public domain in the USA" as copyright status was also removed.

After filtering and collecting each book's information into *books_info.jsonl*, the transcription of the books was also fetched from Project Gutenberg. The transcription file was processed, removing its header and footer, as well as single new lines, since they do not correspond to real paragraphs. Double new lines, which correspond to a single paragraph, were converted into single new lines. Each processed transcription was stored in a different *json* file, along with the respective book information.

For each book title fetched from Project Gutenberg's open library, a search query was made on Goodreads. To associate a book with the proper reviews, the python script *fetch_goodreads_ids.py* chose the top search result with a matching author name (when applicable).

The fetchers for reviews and ratings, however, proved to be a harder task. As reviews were loaded through *JavaScript*, pure HTML fetching was not possible. Even when using a time delay for the request's response, the list of reviews would still not be ready. As such, a different package, called **selenium** was used.

With **selenium** [14], an automated browser window would be created for each book. Whenever the previously mentioned *JavaScript* event was executed, the tool would recognize the changes on the HTML code and allow the script to properly scrape the review corpus.

This method, however, proved to be significantly slower when compared to the book scraper, which used simple HTTP requests. As such, the volume of review data that was possible to actually obtain was smaller. As such, ten reviews were fetched per book review page, ordered by relevance.

The overall rating information retrieved was stored inside the file with the respective book transcription, while the list of reviews of each book was stored in a different *json* file.

2.2 Data Source and Quality

The data sources were selected because of their large availability of free-to-use and reliable information. Project Gutenberg is an online library of over 60000 free eBooks that provide textual transcriptions for each book along with its bibliographic record. Goodreads is a website focused on book recommendations, from which it is possible to gather book ratings and reviews.

In terms of data completeness, the aforementioned data retrieval pipeline was carefully assembled in order to preserve as much information as possible from the sources. Uncertain dates of birth and/or death were accepted as they reflect real gaps in historical records. Furthermore, there are some null, *Anonymous* or *Various* writers, which also are broadly attributed to either faulty historical records, or works which, due to their nature, have no authors, like fables or religious/institutional texts; or multiple authors, like scientific journals. The data is timely, as both the books and reviews are up-to-date, and extracted directly from the source for this purpose. Regarding data correctness, there is occasionally some trace of Project Gutenberg's footer on the end of texts due to its non-standard nature, although these are rare occurrences. All data is consistent, in plain English, and adequately formatted to *json* files in a standard structure according to the conceptual model.

2.3 Conceptual Model

The produced dataset can be modelled according to Figure 1.

A book is defined by its title, the release date on which it was published on Project Gutenberg's platform, its Goodreads average rating, number of ratings, and number of reviews as well as the book corpus.

Each book can have zero to multiple subjects associated with it, which are defined by a name. They can also have zero to multiple authors (some books have no known author,

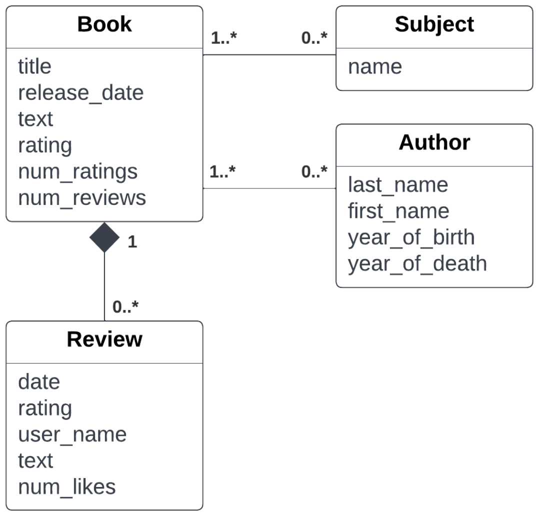


Figure 1. Dataset Conceptual Model.

such as the Bible), which are defined by their name and year of birth and death.

Books can also have Goodreads' user reviews associated with them. Each of these reviews is defined by the review date, the name of the reviewer, the number of likes the community gave to that review as well as the rating given to the book, and the review comment (text).

2.4 Data Characterization

There's a lot of data in this dataset, especially when it concerns the text fields such as the book text and the review text. Tables 1 through 5 and Figures 2 to 5 try to showcase some of its patterns.

Table 1. Top 10 most used words in the books.

word	count	frequency
the	35999199	0.065195
of	19749394	0.035766
and	17721901	0.032095
to	14476953	0.026218
a	13785780	0.024966
in	10074152	0.018244
that	6193965	0.011217
was	5648257	0.010229
he	5636281	0.010207
I	5369173	0.009724

As Table 1 showcases, the books roughly follow Zipf's law [17], supporting the reliability of the textual data. Disregarding *stop words* [1], the most common words in our books dataset are presented in Table 2.

Table 2. Top 10 most used words in the books — excluding *stop words*.

word	count	frequency
time	859451	0.003968
man	859450	0.003968
great	686541	0.003170
good	591981	0.002733
day	575192	0.002656
men	558979	0.002581
life	492075	0.002272
long	488925	0.002258
place	396396	0.001830
people	386386	0.001784

As for the book categories, there's a total of 5182 and the top 10 are shown on Table 3.

Table 3. Top 10 most common book categories.

category	count	frequency
short stories	485	0.024288
science fiction	407	0.020382
united states	348	0.017427
great britain	236	0.011818
england	233	0.011668
world war	232	0.011618
adventure stories	193	0.009665
fiction	185	0.009264
conduct of life	152	0.007612
detective and mystery stories	148	0.007411

The most frequent categories are consistent with major literature genres, such as fiction, science fiction, or adventure stories, the language of the content — United States, Great Britain, England — and the historic time period with the most representation on the dataset (World War).

As for the authors, the plots on figures 2 and 3 showcase their distribution in terms of birth and death years.

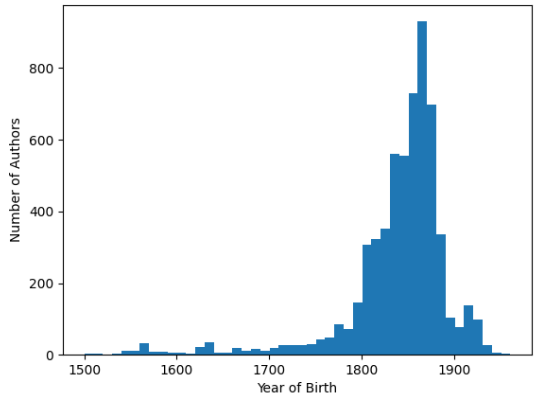


Figure 2. Authors' year of birth distribution.

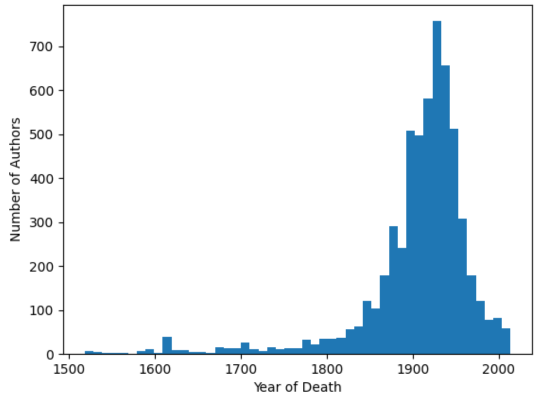


Figure 3. Authors' year of death distribution.

As such, we can conclude that most books in the dataset were written between the early 19^{th} and mid 20^{th} century.

The authors with the most books in the dataset are shown on Table 4.

Table 4. Top 10 most common authors.

author	count
Mark Twain	36
Charles Dickens	27
Edward Bulwer-Lytton	25
Horatio Alger	23
William Henry Giles Kingston	23
William Shakespeare	21
Georg Ebers	21
Honor de Balzac	19
George Manville Fenn	18
Arthur Conan Doyle	18

As for book review ratings, their statistics are evidenced on Table 5 and Figure 4.

Table 5. Rating value distribution.

mean	3.707
standard deviation	0.581
minimum	1.000
25% percentile	3.420
50% percentile	3.770
75% percentile	4.010
maximum	5.000

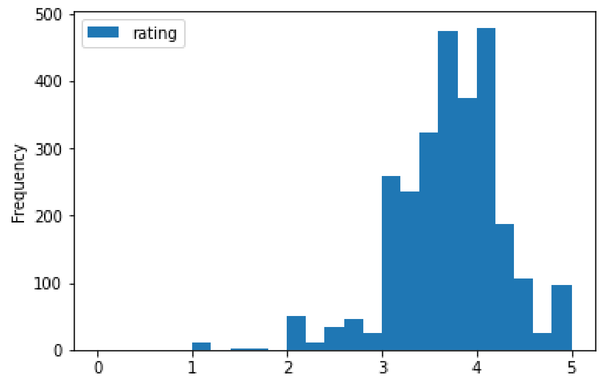


Figure 4. Distribution of the book rating values.

Most of the books' average rating fall between 3 and 4 points (out of 5).

When plotting the mean number of words in a review according to the rating (whole number from 1 to 5) given in that same review, an interesting relationship can be noticed on Figure 5.

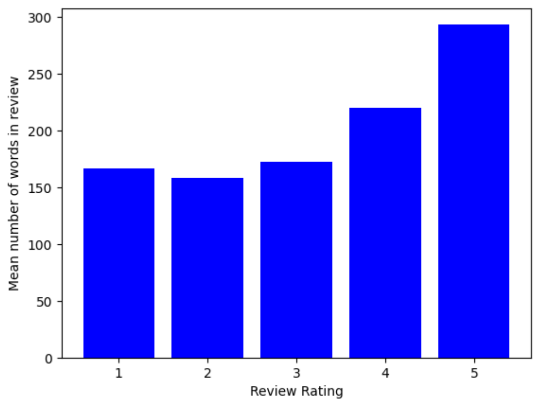


Figure 5. Mean number of words in reviews per rating.

As such, we can observe that users tend to write significantly more when they give the book a positive rating, taking the time and effort to justify the score with more depth.

It is important to assess whether the review samples are representative of the overall public sentiment towards the books. Figure 6 compares the average score from the review samples for a given book with their overall Goodreads score, and shows a clear alignment between the two.

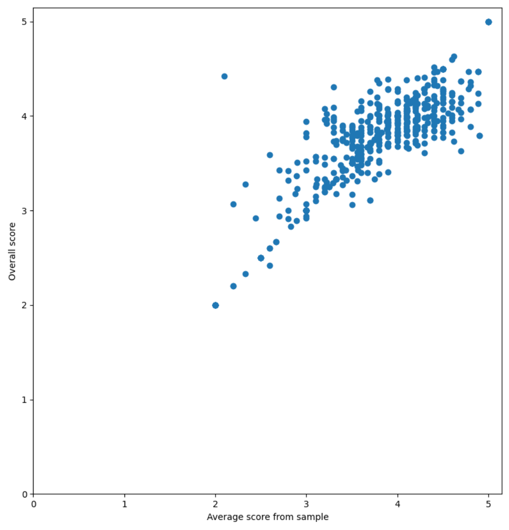


Figure 6. Representativeness of book score samples.

Additionally, it is evident that the distribution of scores for books with review samples is representative of the overall score distribution found in Figure 4.

3 Search Scenarios

The resulting information search system will set out to fulfill search scenarios with added value, with a focus on responding to information needs that are not well attended to already, namely:

- I want to find excerpts about a specific concept or event.
- I want to find context on a book quote.
- I want to read a book from a certain time period.
- I want to understand what other people think of a book.
- I want to browse an extensive library of copyright-free literature.

4 Collection and Indexing

For the information retrieval task, it was important to choose a search engine that would suit the project's needs and constraints. To this end, the best choice proved to be Apache Solr, a free, open-source search engine based on the Apache Lucene library. This tool can work with large amounts of semi-structured data, such as the one generated, with full-text search capabilities, simple RESTful XML/HTTP and JSON APIs, and a stable docker image. [9].

4.1 Document Definition

From the dataset described in Section 2, We created a single core in Solr with three types of documents. All documents have a field *content_type* to identify its type, which can have the values "BOOK", "AUTHOR", or "REVIEW". The "AUTHOR" and "REVIEW" documents are defined as nested documents of "BOOK" documents, making queries on parent documents using information from child documents more convenient and efficient.

Besides the *content_type* field and the globally unique *id* field required by Solr, each document type contains the respective fields shown in Figure 1. The subjects are included as an array contained in the "BOOK" documents.

4.2 Indexing Process

The indexing process plays a significant role in the information system since it influences the efficiency and quality of the search results. In addition to deciding which fields should be indexed, it is also important to define the transformation pipelines that should be executed to preprocess the text fields that will be indexed. This preprocessing makes the system more resilient to searches that aren't exact matches.

4.2.1 Indexed fields. The fields chosen to be indexed are shown in the *indexed* column of Table 6. These indices are enough to adequately solve the search scenarios, providing all useful information without wasting unnecessary space.

The *content_type* index is essential to distinguish document types, which are then queried on the other indexed fields.

Regarding books, it is important to search by *title*, *subjects*, *rating*, and *text*, as well as sort by *release_date*, so they should be indexed. The number of ratings and reviews is useful, albeit only as a stored field, to complement the search result, and not as a search parameter. Following this analysis, these fields were not indexed.

Authors may be searched by their *last_name* and *first_name*, and by their time period, through the fields *year_of_birth* and *year_of_death*. Therefore, these fields were indexed.

Reviews only need to be searched by their text and rating, as the other pieces of information — *date*, *user_name* and *num_likes* — are not very interesting as search parameters and thus were not indexed.

4.2.2 Index analyzers. To make our information system more resilient to minor differences between the field content and the query, such as case sensitivity or accents, it was important to define analyzers to process the indexed text fields and the queries. The *title*, *text* and *subjects* fields contain unstructured text or text with very little structure. In these fields, searches are expected to be lax, since there is redundancy and abundance, so a text field type named *GeneralTextField* was created for them. On the other hand, the *first_name* and *last_name* fields should accept some imprecision, but they refer to people's names rather than general text, thereby demanding a stricter search, being assigned a text field type named *NameTextField* that we created for this purpose. Finally, the *user_name* field pertains to usernames, which will not be indexed, and searching by this field is not a focus in this information system. Even if it was, the matches are expected to be exact or at least very strict, so the field was handled as a regular string.

We tried two different approaches for defining the index analyzers for each new text field type. In the first approach, we treated both types equally, employing the *Standard Tokenizer*. This tokenizer splits the text field into tokens, treating whitespace and punctuation as delimiters and discarding the delimiters. We applied the *ASCII Folding Filter* to convert Unicode characters, such as characters with accents, into their ASCII counterparts, while preserving the original tokens, so that searches match either possibility. We also applied the *Lower Case Filter*, which converts all tokens to lowercase, so that the search becomes case insensitive.

In the second approach, we added four more filters to the *GeneralTextField* type. The first filter, *Synonym Graph Filter* allows matching synonyms instead of the exact words. The second filter, *Flatten Graph Filter*, is required by the previous filter during indexing, although it was not used as a query analyzer. The third filter, *English Possessive Filter*, removes the English singular possessives from the tokens. The fourth filter, *English Minimal Stem Filter* stems plural English words to their singular form so that both singular and plural words match.

4.3 Schema

We defined a different schema for each approach we described in Section 4.2.2. Both schemata include four custom field types. The first type, named *ContentType*, is an enumeration of the three valid types of documents described in Section 4.1. The second one, *DateRangeFieldType*, is a type of class *DateRangeField* to represent ranges of dates, in particular whole years. The other two field types are *GeneralTextField* and *NameTextField*, described in more detail in Section 4.2. These fields define the tokenizers and filters for the respective approach in each schema, with the index analyzers being the same as the query analyzers, except *Flatten Graph Filter*, which is only defined for the index analyzer.

The fields defined in both schemata can be seen in Table 6. The *subjects* field is the only *multivalued* field.

Table 6. Fields defined in the schemata.

field name	field type	indexed
content_type	ContentType	yes
title	GeneralTextField	yes
release_date	pdate	yes
subjects	GeneralTextField	yes
rating	pfloat	yes
num_ratings	plong	no
num_reviews	plong	no
text	GeneralTextField	yes
last_name	NameTextField	yes
first_name	NameTextField	yes
year_of_birth	DateRangeFieldType	yes
year_of_death	DateRangeFieldType	yes
date	pdate	no
user_name	string	no
num_likes	plong	no

5 Retrieval

After indexing, filtering, and tokenizing the dataset, it is feasible to start the information retrieval process. We attempted to find, for each search scenario presented in 3, an adequate query that answers an information need. The following subsections showcase those queries.

As for query parsers, we utilized *Lucene* [4] for most of the queries, and used *eDisMax* [2] for its Independent Boosting capabilities. For each information need, we specify its search scenario, the query parser used, a base query, and an improved query. We also present, for each query, the field parameters used by the query parser. Both query parsers require a query parameter, *q*, that defines the query using standard query syntax [5]. They also allow overriding the default operator for query expressions, "AND" or "OR", with the *q.op* parameter. Finally, *eDisMax* optionally allows the specification of one or multiple boost query parameters, *bq*, which are query clauses that will be added to the main query to influence the score.

5.1 Information Need 1

Search Scenario: I want to find excerpts about a specific concept or event

Information Need: Books explaining home economics

Query Parser: *Lucene*

Base Query: Looks for books which include *home* and *economics* somewhere in their subjects, title or body.

- **q:** `(subjects:economics OR text:economics OR title:economics) (subjects:home OR text:home OR title:home)`
- **q.op:** `AND`

Improved Query: Uses Field Boosting on matches for the subjects, giving more importance to them (factor of 1.5), wildcard on *economics* to accept any words with the radical *econ*, and fuzzing on *home* to look for close matches, such as *homes* or *homely*.

- **q:** `((subjects:econ*) ^ 1.5 OR text:econ* OR title:econ*) ((subjects:home~) ^ 1.5 OR text:home~ OR title:home~)`
- **q.op:** `AND`

5.2 Information Need 2

Search Scenario: I want to read a book from a certain time period

Information Need: Recent romantic novels released preferably around 2018

Query Parser: *eDisMax*

Base Query: Looks for books with the *fiction* subject that have *love* as a subject or as a word in their text.

- **q:** `subjects:fiction (subjects:love OR text:love)`
- **q.op:** `AND`

Improved Query: Uses Term Boosting for *love* matches on the subject (factor of 2), wildcard on *love* to accept any words with the radical *lov*, and Independent Boosting on the *release_date* field, favouring recently released books.

- **q:** `(subjects:lov* ^ 2 OR text:lov*)`
- **q.op:** `AND`
- **bq:** `release_date:[NOW/DAY-6YEAR TO NOW/DAY-2YEAR]`

5.3 Information Need 3

Search Scenario: I want to find context on a book quote.

Information Need: Books containing the quote "waste not, want not", even if the user doesn't quite know the quote (inputting "want not, waste not")

Query Parser: *Lucene*

Base Query: Searches for books which include the literal passage *"want not, waste not"*.

- **q:** `text:"want not, waste not"`
- **q.op:** `AND`

Improved Query: Uses Proximity Search to search for books which include the segments of the quote within 5 words of each other.

- **q:** `text:"want not, waste not"~5`
- **q.op:** `AND`

5.4 Information Need 4

Search Scenario: I want to find context on a book quote.

Information Need: Books containing the quote "waste not, want not", even if the user doesn't quite know the quote and has typos (inputting "want nto, waste not")

Query Parser: *Lucene*

Base Query: Searches for books which include the literal passage *"want nto, waste not"*.

- **q:** `text:"want nto, waste not"`
- **q.op:** `AND`

Improved Query: Uses Proximity Search to search for books that include the segments of the mistyped quote within 5 words of each other.

- **q:** `text:"want nto, waste not"~5`
- **q.op:** `AND`

5.5 Other Search Scenarios

Two search scenarios were not tackled in the previously shown queries. This is because its results were not subject to evaluation — that is, its results are either binary or too broad.

I want to understand what other people think of a book relates to retrieving the reviews of a book that the user is looking for. As such, querying that book has one and only one relevant result — so the evaluation is binary. One could try to do other kinds of queries — such as attempting to retrieve books where the user reviews express some sentiment, such as being 'easy to read' —, but that also proven to be too arduous, since defining what are the correct relevant documents of such queries was not possible. In order to fulfill this search need, we may search for a book with the query parameter *fl* containing the *[child]* flag in order to show the reviews nested in the book documents.

I want to browse an extensive library of copyright-free literature relates to many different queries, as it is tackled through designing a user-friendly interface for querying — which is not the focus of this section.

Following these current limitations, the remaining search scenarios will be explored with the implementation of a frontend for our search system.

Although none of the search scenarios used for evaluation include information about the author, that is also possible in our search system. For example, in order to find the books authored by "Le Fanu" we may simply search with the query:

- **q:** `{!parent which="content_type:BOOK"} last_name:"Le Fanu"`
- **q.op:** `AND`

6 Evaluation

To evaluate how well the system performs on each of the queries relative to the information needs, we needed different systems and different metrics to compare.

We adopted the following metrics:

- Precision at 10 (P@10), which is the proportion of the top 10 documents that are relevant, emulating the common user experience of looking at the top results.
- Average Precision (AP), which is the average of the P@10 value obtained for the set of top 10 documents existing after each relevant document is retrieved.
- Mean Average Precision (mAP), for each system, which is the mean over the AP values for the system.

As for the different systems to test the queries on, we defined two different schemata according to the specifications in Section 4.2.2 and assigned them to separate cores.

Given that we have 2 different cores with their own schemata, and 2 different queries for each information need (one with basic field querying and another with boosting, fuzzing, and proximity matching), we get 4 different systems:

- (**sys1**) Base schema with basic field querying
- (**sys1_syn**) Synonym schema with basic field querying
- (**sys2**) Base schema with complex querying
- (**sys2_syn**) Synonym schema with complex querying

In the following sections, we showcase, for each query, the results for each of the metrics, an explanation of how we determined the relevant query results used as a basis for the metrics, and the mean average precision for each system. The Precision–Recall curves for each system–query pair are present in Appendix D.

To calculate the efficiency metrics, the documents retrieved by the query were compared with a subset of expected relevant documents. To this end, each retrieved document i is classified as one of the following:

1. relevant (R), if the i^{th} document matches one of the relevant documents
2. irrelevant (I), if the i^{th} document does not match any of the relevant documents
3. non-applicable (N/A), if there's no i^{th} document – which means that the query did not return more than i documents

6.1 Information Need 1

Table 7 showcases the obtained metrics for Information Need 1.

Table 7. Information Need 1 evaluation results.

Rank	sys1	sys2	sys1_syn	sys2_syn
1	I	R	I	R
2	R	R	I	I
3	R	I	R	R
4	I	R	R	R
5	R	R	R	R
6	R	R	R	R
7	R	R	R	R
8	I	R	R	I
9	I	R	I	I
10	R	R	I	I
P@10	0.6	0.9	0.6	0.6
AP	0.542817	0.857103	0.483095	0.742381

To define the relevant documents to this information need, we did the process in a backward fashion. Firstly, we iterated all document categories and listed which books fit in which categories. After that, we verified that the category 'Home Economics' has 10 and only 10 relevant documents – which is perfect for our metrics (P@10). After having the relevant documents, we defined the information need to find books on this topic.

The results showcase what was to be expected - the boosted query bumps the performance on both the *base* and *synonym* systems. This performance bump, however, is not as noticeable on the*synonym* system, which can be explained due to the noise caused by the synonym filter around the searched keywords.

6.2 Information Need 2

Table 8 showcases the obtained metrics for Information Need 2.

Table 8. Information Need 2 evaluation results.

Rank	sys1	sys2	sys1_syn	sys2_syn
1	I	R	I	R
2	I	I	I	I
3	I	R	I	I
4	R	I	I	R
5	R	R	I	I
6	R	R	I	I
7	I	I	I	I
8	I	I	I	R
9	I	R	I	R
10	I	I	I	R
P@10	0.3	0.5	0.0	0.5
AP	0.258690	0.606032	0.0	0.467183

For this information need, we manually looked up all fiction love books that were released around 2018. We selected the 10 books whose release dates were the closest to mid-2018.

Like in the previous information need, the boosted query bumps the metrics and the *synonym* system still performs worse when compared to the *base* system.

6.3 Information Need 3

Table 9 showcases the obtained metrics for Information Need 3.

Table 9. Information Need 3 evaluation results.

Rank	sys1	sys2	sys1_syn	sys2_syn
1	N/A	R	I	I
2	N/A	R	N/A	I
3	N/A	R	N/A	R
4	N/A	I	N/A	R
5	N/A	I	N/A	I
6	N/A	R	N/A	I
7	N/A	I	N/A	I
8	N/A	R	N/A	I
9	N/A	R	N/A	I
10	N/A	R	N/A	I
P@10	0.0	0.7	0.0	0.2
AP	0.0	0.757976	0.0	0.252460

For this information need, the relevant documents were also retrieved programmatically. To do so, we iterated through all of the documents searching for some of the most common English expressions, retrieved from [8]. After calculating their frequency per book, we selected the expression which appeared in 10 books. As such, those 10 books were selected as the 10 most relevant.

Since we're dealing with phrase matching for an incorrect expression and only the boosted query has phrase proximity operations, the boosted query systems outperform the simple ones, as expected. Note that the basic querying systems do not return any results — neither relevant nor irrelevant.

6.4 Information Need 4

Table 10 showcases the obtained metrics for Information Need 4.

Table 10. Information Need 4 evaluation results.

Rank	sys1	sys2	sys1_syn	sys2_syn
1	N/A	N/A	I	I
2	N/A	N/A	N/A	I
3	N/A	N/A	N/A	R
4	N/A	N/A	N/A	R
5	N/A	N/A	N/A	I
6	N/A	N/A	N/A	I
7	N/A	N/A	N/A	I
8	N/A	N/A	N/A	I
9	N/A	N/A	N/A	I
10	N/A	N/A	N/A	I
P@10	0.0	0.0	0.0	0.2
AP	0.0	0.0	0.0	0.252460

The relevant documents for this information need are the same as the ones for Information Need 3.

This information need illustrates where the *synonym* system outperforms the *base* one. When faced with a spelling mistake in a phrase to match — given the query parser's limitations —, only the *synonym* system can output results as it can associate `"nto"` to `"not"`. As the typo is fixed, the proximity operator can do his work to salvage some relevant results on this query.

6.5 Mean Average Precision per System

Table 11 showcases the Mean Average Precision for each of the systems on which the evaluation was made on.

Table 11. Mean Average Precision (*mAP*) for each of the Systems.

System	mAP
sys1	0.200377
sys1_syn	0.120774
sys2	0.555278
sys2_syn	0.428621

The obtained results also match what was to be expected: the systems that adopt boosting, fuzzing, wildcards, and proximity search mechanisms outperform the ones that don't. On the other hand, the difference in *mAP* between the *base* and *synonym* systems is negligible and thus we can't draw any conclusions on which system is best. However, it is possible to infer that the *synonym* system is more resilient to query imprecisions and errors, while the *base* system can be more precise in the right conditions. This information may be useful when deciding which system to use in the future.

7 Search System Improvements

The improvements that were carried out aim to fulfill the user-oriented information needs — *I want to understand what other people think of a book* and *I want to browse an extensive library of copyright-free literature* — and increase the robustness of the search system. To that end, the book and review set was significantly increased; a website was designed for better user experience; various advanced search tools were added for search efficacy; and OpenNLP was integrated to permit named entity recognition.

7.1 Expanded Dataset

We have collected additional books and reviews, significantly extending our dataset to 14581 book entries and 19420 review entries (an average of 5 reviews each for 3884 books), making up to around 5.4 GB of memory space. With nearly double the size and much more reviews, this dataset is sure to be useful to more and more users.

7.2 Named Entity Recognition

Solr offers a multitude of filters, factories, and tokenizers. Some of the most interesting ones are from the OpenNLP module. This module, based on Apache OpenNLP [6], provides *a machine learning-based toolkit for the processing of natural language text.* Such processing may be extremely useful, not only by processing fields and extracting useful information but also by interpreting the meaning of the user queries.

Even though the referred second case is deemed to be possible in Solr, it was not possible to replicate that behavior. Still, NER was applied to the book's corpora.

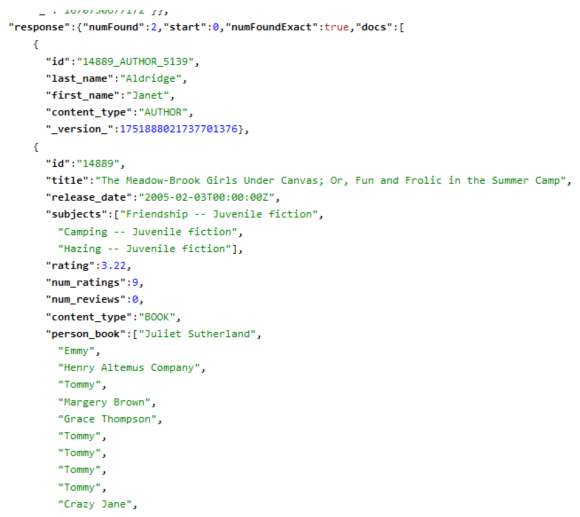
Named Entity Recognition (NER), *is a process where a sentence or a chunk of text is parsed through to find entities that can be put under categories like names, organizations, locations, (...), etc.* [3]. Retrieving such information from the books would allow the user to search more easily for main characters or books taking place in a specific location or era, which would in turn help meet some of the search scenarios previously defined in 3.

In order to allow Solr to extract Named Entities, one needs to update *solrconfig.xml*'s Processor Chain by adding the *OpenNLPExtractNamedEntitiesUpdateProcessorFactory*. This factory accepts a pre-trained ML model, an analyzer type and the source and destination fields - that is, the source field from which the factory will extract the information, and the destination field which the factory will store that information on. The only restriction is that the chosen analyzer type, which will correspond to the type on which the source field will be interpreted, have the *OpenNLPTokenizerFactory* as a tokenizer, which can be easily achieved by updating the core's schema.

Most filters, factories, and tokenizers need an associated pre-trained model. All models are readily available at the OpenNLP website for 1.5-compatible models [11].

We ended up choosing three different types of entities that would be relevant to the Information System: people, locations, and dates. Each of which has its field, which stores all occurrences of a named entity of that type on the *text* field of every book and review.

Figure 7 shows some of the found named entities for the book *The Meadow-Brook Girls Under Canvas* by Janet Aldridge.



```
"response":{"numFound":2,"start":0,"numFoundExact":true,"docs":[
    {
      "id":"14889_AUTHOR_5139",
      "last_name":"Aldridge",
      "first_name":"Janet",
      "content_type":"AUTHOR",
      "_version_":1751888021737701376},
    {
      "id":"14889",
      "title":"The Meadow-Brook Girls Under Canvas; Or, Fun and Frolic in the Summer Camp",
      "release_date":"2005-02-03T00:00:00Z",
      "subjects":["Friendship -- Juvenile fiction",
        "Camping -- Juvenile fiction",
        "Hazing -- Juvenile fiction"],
      "rating":3.22,
      "num_ratings":9,
      "num_reviews":0,
      "content_type":"BOOK",
      "person_book":["Juliet Sutherland",
        "Emmy",
        "Henry Altemus Company",
        "Tommy",
        "Margery Brown",
        "Grace Thompson",
        "Tommy",
        "Tommy",
        "Tommy",
        "Tommy",
        "Crazy Jane",
```

Figure 7. Named Entity Recognition of people in *The Meadow-Brook Girls Under Canvas* in field *person_book*.

Although the results are satisfactory, running the OpenNLP schema on the whole dataset proved unfeasible in terms of computational resources. As such, the main UI application - which we will discuss in the following section - does not take advantage of these newly generated features as it is using a core running the base schema.

7.3 Accessibility and Browsing

Focusing on accessibility and convenience, we have implemented two layers of abstraction on top of our Solr system. The first one is a REST API implemented in Django [15], which translates higher-level queries into Solr queries with the appropriate boosts and other parameters. The second is a React [13] frontend, which presents an appealing interface through which the user can browse the dataset of books with many search capabilities.

7.3.1 Simple Search. The user may resort to the search bar from the front page in order to make a simple query. This query will have a structure similar to the improved query from Section 5.1, using the fields *text*, *title*, and *subjects*, with a field boost of 1.5 on the latter two.

7.3.2 Advanced Search. For more sophisticated searches with more fields, the user may open the advanced search menu, as shown on Figure 14. This menu provides additional fields so that the user may look for a specific title, category, author name, number of ratings, rating range, release date range, and the range of time the author was alive. The implementation of the author fields required the special syntax of querying nested documents explained in Section 5.5.

7.3.3 Faceting. When filling out the advanced search form, the user may enter a category for the book. To make this selection easier and more convenient, the possible categories are suggested to the user as they fill this field. This was implemented with help of Solr's Faceting feature.

In each schema, we added a non-stored string field named *subjects_facet* which copies the value from *subjects*. This field is used as a facet field to retrieve the complete category values, as opposed to individual tokens. This way, we can query the system for the categories that exist in its documents.

7.3.4 Quote Search. When the "*Quote Search*" checkbox is ticked, simple and advanced searches start searching for book quotes, using a proximity search of 5 words. Moreover, this mode takes advantage of Solr's highlighting feature to show the context of the searched quote that appears in the matched documents, as seen in Figure 15. By simply activating the highlight flag (hl), Solr documents will have an associated 'highlight' document, which contains the snippet of the book corpus where the search quote was found in.

7.3.5 Mistype Mitigation. To complement Solr's search capabilities with robustness to mistyping, TextBlob's spelling correction was incorporated into the system. TextBlob [7] is a python library that helps to process textual data. However, given that this spell check may have false positives, it is still possible to force the system to do a search that does not use TextBlob's capabilities.

7.3.6 Recommendation of Related Content. After encountering a search result we are looking for, it is natural to have the desire to see results that are similar to this one. As such, we make use of Solr's *MoreLikeThis* component to search for results similar to a given book. To configure this feature, we loaded the *MoreLikeThis* component and defined the target field as *text* – which means that the similarity score between two given books will be given by how similar their *text* fields are. Easily enough, using the now available Solr route */mlt*, we can find similar books to the one with the given ID.

7.3.7 Book details. When browsing the list of search results, it is possible to click on a result to see the details of the respective book on a different page. This page presents the book transcription as well as the list of its reviews, as can be seen in Figure 16.

7.4 Evaluation of the Improved System

Having enriched our search system with the described improvements, we were not only able to make a more accessible and user-friendly system but also managed to improve its results on the information needs described in Section 5. This was mainly due to our detection of spelling mistakes. Table 12 shows the evaluation of the improved system for each information need (*IN*).

Table 12. Information Needs evaluation results for the latest system.

Rank	IN1	IN2	IN3	IN4
1	R	R	R	R
2	R	I	R	R
3	R	R	R	R
4	R	I	I	I
5	R	R	I	I
6	R	R	R	R
7	I	I	I	I
8	R	I	R	R
9	R	R	R	R
10	R	I	R	R
P@10	0.9	0.5	0.7	0.7
AP	0.952103	0.606032	0.757976	0.757976

The mean average precision *mAP* is 0.768522. Precision over recall for this system is shown in Figure 13.

8 Conclusions and Future Work

The data pipeline produced a sizeable and representative sample of texts and reviews in the English language that served as a basis for the information retrieval tasks. Using Solr and the produced dataset, the information search system was proven capable of catering to the proposed informational needs by covering the most relevant search scenarios and evidencing the positive impact of carefully crafted schemata and query optimization tools on the search engine's performance. Furthermore, the addition of advanced search tools and more complex Solr utilities, paired with a solid UI and API for the search engine, added great value to the information retrieval system. As for future work, support for named entity-based queries and natural language interpretation can be added, as well as live search suggestions.

References

[1] 2008. Dropping common terms: stop words. https://nlp.stanford.edu/IR-book/html/htmledition/dropping-common-terms-stop-words-1.html [Accessed Oct. 9, 2022].

[2] 2017. The Extended DisMax Query Parser | Apache Solr Reference Guide 6.6. https://solr.apache.org/guide/6_6/the-extended-dismax-query-parser.html [Accessed Nov. 13, 2022].

[3] 2017. Named-entity recognition. https://deepai.org/machine-learning-glossary-and-terms/named-entity-recognition [Accessed Dec. 11, 2022].

[4] 2017. The Standard Query Parser | Apache Solr Reference Guide 6.6. https://solr.apache.org/guide/6_6/the-standard-query-parser.html [Accessed Nov. 13, 2022].

[5] 2017. The Standard Query Parser | Apache Solr Reference Guide 6.6. https://solr.apache.org/guide/6_6/the-standard-query-parser.html#TheStandardQueryParser-SpecifyingTermsfortheStandardQueryParser [Accessed Nov. 13, 2022].

[6] 2017. Welcome to apache opennlp. https://opennlp.apache.org/ [Accessed Dec. 11, 2022].

[7] 2018. textblob Documentation. https://www.djangoproject.com [Accessed Dec. 11, 2022].

[8] 2019. English idioms: Ef: Global site. https://www.ef.com/wwen/english-resources/english-idioms/ [Accessed Nov. 13, 2022].

[9] 2022. Apache Solr Tutorial: What Is, How It Works & What Is It Used For - Sematext. https://sematext.com/guides/solr/ [Accessed Nov. 13, 2022].

[10] 2022. Meet your next favorite book. https://www.goodreads.com/ [Accessed Oct. 9, 2022].

[11] 2022. OpenNLP Tools Models. https://opennlp.sourceforge.net/models-1.5/ [Accessed Dec. 11, 2022].

[12] 2022. Project gutenberg. https://www.gutenberg.org/ [Accessed Oct. 9, 2022].

[13] 2022. React – A JavaScript library for building user interfaces. https://reactjs.org [Accessed Dec. 11, 2022].

[14] 2022. The selenium browser automation project. https://www.selenium.dev/documentation/ [Accessed Oct. 9, 2022].

[15] 2022. The web framework for perfectionists with deadlines | Django. https://www.djangoproject.com [Accessed Dec. 11, 2022].

[16] 2022. Welcome to Apache Solr. https://solr.apache.org [Accessed Nov. 13, 2022].

[17] Zipf G. 1936. The Psychobiology of Language. [Accessed Oct. 9, 2022].

A Pipeline

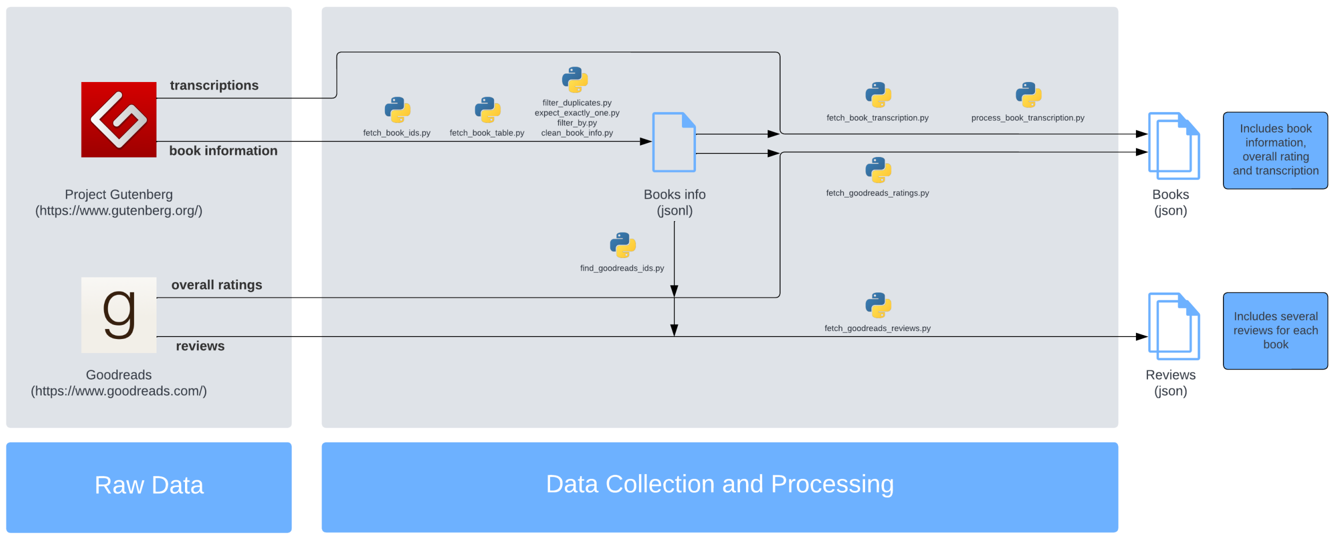

transcriptions
book information
Project Gutenberg
(https://www.gutenberg.org/)
overall ratings
reviews
Goodreads
(https://www.goodreads.com/)
fetch_book_ids.py
fetch_book_table.py
filter_duplicates.py
expect_exactly_one.py
filter_by.py
clean_book_info.py
Books info
(jsonl)
find_goodreads_ids.py
fetch_book_transcription.py
process_book_transcription.py
fetch_goodreads_ratings.py
fetch_goodreads_reviews.py
Books
(json)
Includes book information, overall rating and transcription
Reviews
(json)
Includes several reviews for each book
Raw Data
Data Collection and Processing

Figure 8. Data Scraping, Collection and Processing pipeline.

B Improvements from Milestone 1

- Reference all figures and tables in the text.
- End all captions in figures and tables with a period.
- Add access date to references of websites.
- Add scatter plot of Figure 6 that compares two variables (global rating and average rating of reviews) in order to see how representative the sample of reviews is compared to the actual rating of books.
- Update abstract, introduction and conclusion to reflect work done on milestone 2.
- Create Section 4, Section 5 and Section 6 to reflect work done on milestone 2.
- Add appropriate en and em dashes.
- Fix overfull \hboxes.

C Improvements from Milestone 2

- Correct average precision and mean average precision calculations.
- Interpolate P-R curve graphs.
- Create section 7 to reflect work done on milestone 3.
- Update abstract, introduction and conclusion to reflect work done on milestone 3.

D P-R Curves

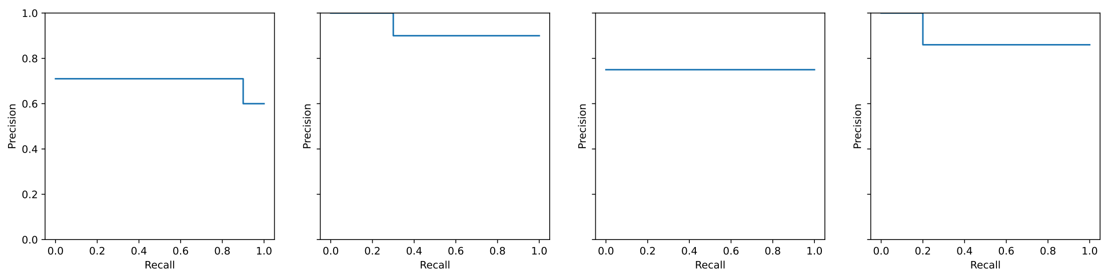


Figure 9. Interpolated P-R Curve for Information Need 1. From left to right: sys1, sys2, sys1_syn, sys2_syn.

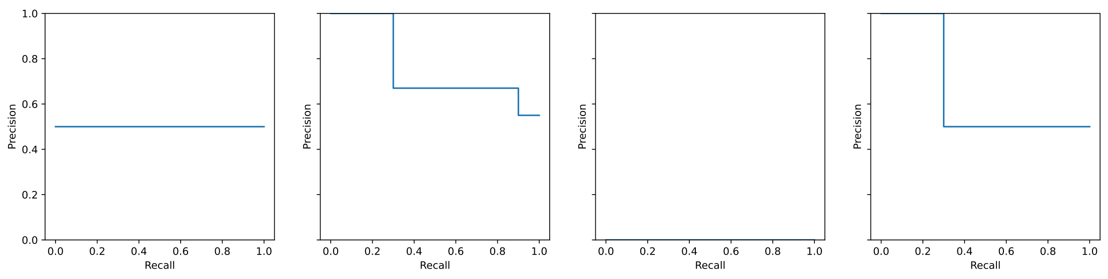


Figure 10. Interpolated P-R Curve for Information Need 2. From left to right: sys1, sys2, sys1_syn, sys2_syn.

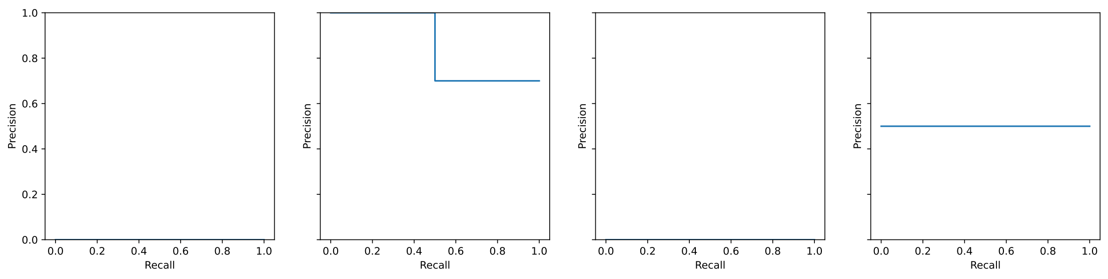


Figure 11. Interpolated P-R Curve for Information Need 3. From left to right: sys1, sys2, sys1_syn, sys2_syn.

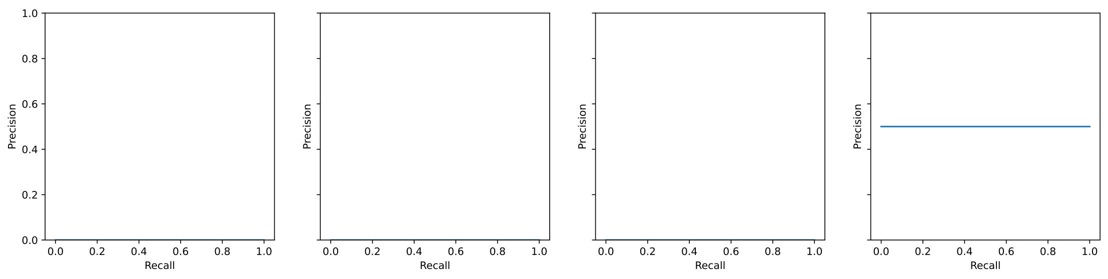


Figure 12. Interpolated P-R Curve for Information Need 4. From left to right: sys1, sys2, sys1_syn, sys2_syn.

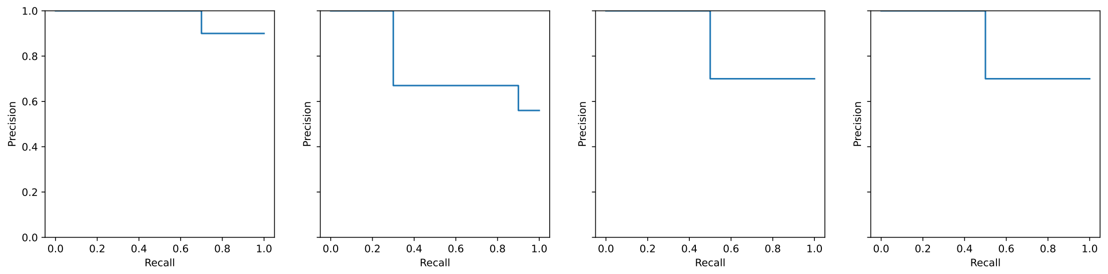
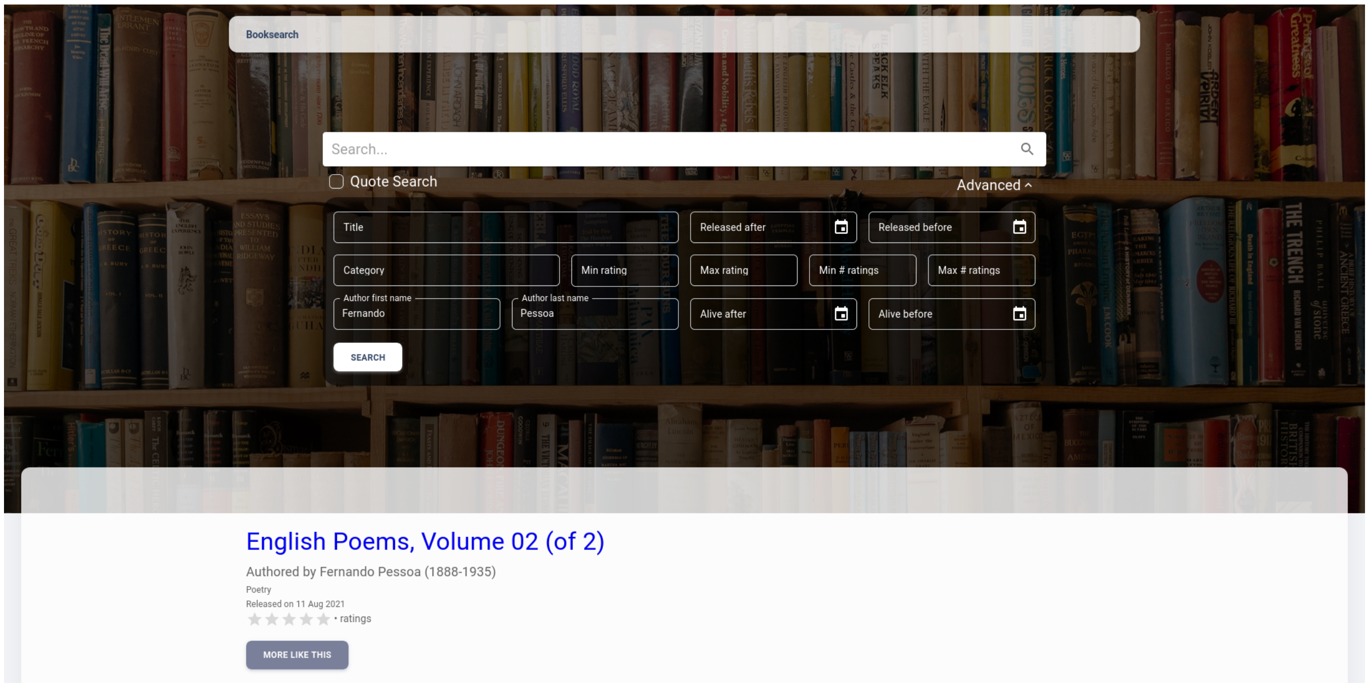


Figure 13. Interpolated P-R Curve for System 3. From left to right: IN1, IN2, IN3, IN4.

E Website User Experience



English Poems, Volume 02 (of 2)

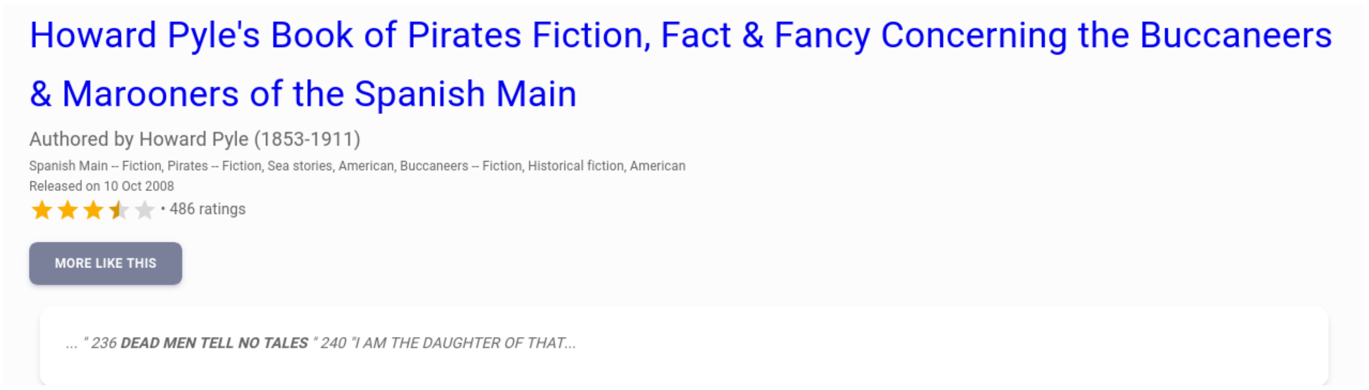
Authored by Fernando Pessoa (1888-1935)

Poetry

Released on 11 Aug 2021

• ratings

Figure 14. Advanced search with match.



Howard Pyle's Book of Pirates Fiction, Fact & Fancy Concerning the Buccaneers & Marooners of the Spanish Main

Authored by Howard Pyle (1853-1911)

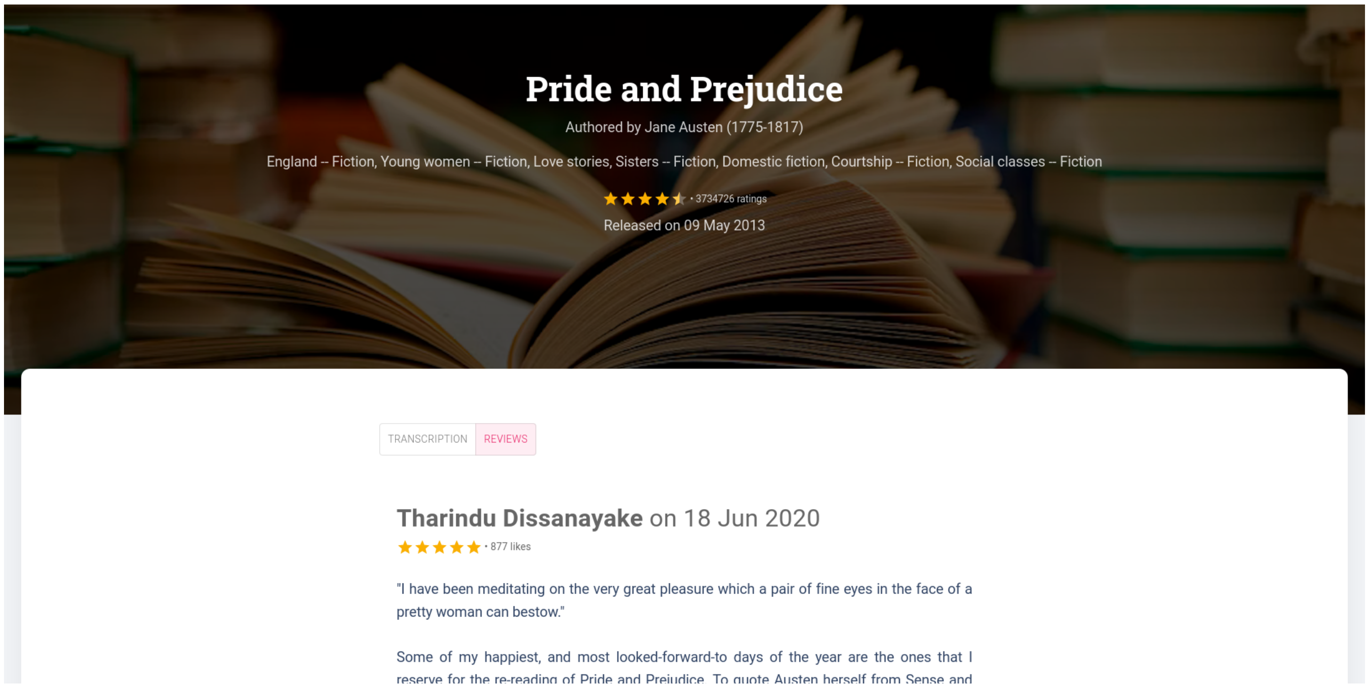
Spanish Main -- Fiction, Pirates -- Fiction, Sea stories, American, Buccaneers -- Fiction, Historical fiction, American

Released on 10 Oct 2008

• 486 ratings

... " 236 ***DEAD MEN TELL NO TALES*** " 240 "I AM THE DAUGHTER OF THAT...

Figure 15. Book card with quote highlighting and relevant information.



Pride and Prejudice

Authored by Jane Austen (1775-1817)

England -- Fiction, Young women -- Fiction, Love stories, Sisters -- Fiction, Domestic fiction, Courtship -- Fiction, Social classes -- Fiction

★★★★½ • 3734726 ratings

Released on 09 May 2013

Tharindu Dissanayake on 18 Jun 2020

★★★★★ • 877 likes

"I have been meditating on the very great pleasure which a pair of fine eyes in the face of a pretty woman can bestow."

Some of my happiest, and most looked-forward-to days of the year are the ones that I reserve for the re-reading of Pride and Prejudice. To quote Austen herself from Sense and

Figure 16. Book page with reviews.