PRIMED: A Medicine Search System

Pedro Simões
up202403063@up.pt
Faculdade de Engenharia da Universidade do Porto
Porto, Portugal

Miguel Garrido
up202108889@up.pt
Faculdade de Engenharia da Universidade do Porto
Porto, Portugal

Emanuel Maia
up202107486@up.pt
Faculdade de Engenharia da Universidade do Porto
Porto, Portugal

Guilherme Martins
up202403106@up.pt
Faculdade de Engenharia da Universidade do Porto
Porto, Portugal

Abstract

The ***PRIMED*** project aims to enhance access to structured pharmaceutical data for healthcare and research by collecting and processing information on medicines, diseases, manufacturers, and user reviews. The project started with a data collection phase, from diverse sources such as Kaggle and Wikipedia, followed by a comprehensive data pipeline implemented in `Python`. Key steps in this pipeline include data cleaning, text normalization, and standardization of formats, ensuring the data is structured and easily searchable. The model stores data in a `JSON` format, making it compatible with future integration into larger systems. In the final milestone, the focus was mostly on developing a usable frontend, providing users with an easy way to access the different versions of the search engine to perform their queries, a re-ranking system, which aims to re-order search results based on a polarity analysis of user reviews, and a new schema based on semantic search. Results presented in this paper indicate that the processed data, along with the new additions to our querying system improve accessibility and organization, providing a valuable resource for healthcare professionals and researchers in making informed decisions.

CCS Concepts

• **Information systems** → Information retrieval query processing.

Keywords

Medicine, Treatments, Sickness, Pipeline, Data, Gathering, Scraping, Preparation, Search Engine

ACM Reference Format:
Pedro Simões, Miguel Garrido, Emanuel Maia, and Guilherme Martins. 2024. PRIMED: A Medicine Search System. In *Proceedings of PRI (G51).* ACM, New York, NY, USA, 16 pages.

G51, October 13, 2024, Porto, PT

ACM ISBN 978-1-4503-XXXX-X/18/06

1 Introduction

Nowadays, accurate and comprehensive medicine information plays a vital role in decision-making on the healthcare and investigation sectors. Having access to clean and relevant information is crucial to enhance the efficiency and safety of treatments.

The ***PRIMED*** project aims to compile and organize such data so that it can be accessed in an easy and organized way. It provides insights that can be applied both in practical cases or simply in developing new health safety politics.

In this part of the project, data from various sources was collected, processed and analysed with the goal of creating a solid basis for a pharmaceutical system. The whole process is described in this document, from the choice of the theme itself up to the analysis of gathered data and its subsequent classification.

The choice to focus on medicines as the central theme for this project stems from the critical role they play in modern healthcare. Medication is the primary tool for treating multiple health problems and conditions, making them an essential component in both public healthcare systems and individual patient care. The data surrounding these substances, such as active substances, applicable cases and clinical trials, offers valuable insights that can improve decision-making in medical practice and pharmaceutics.

2 Data Collection

This chapter outlines the data selection process and the methods utilized to gather it.

2.1 Selection of Data

One of the main challenges of the healthcare sector is ensuring that accurate and up-to-date information on medication is available to healthcare professionals. Given these needs, the selection of data to be collected was made:

- `Medicines`: The main component of the data, consisting of medicines and their respective relevant, intrinsic information.
- `Diseases`: To complement the collected data, information on some diseases was collected so that it would be possible to get more information to allow for an easier medicine selection process.
- `Pharmaceutical Companies`: Some pharmaceutical companies are more trusted than others due to their credibility and higher quality products, which can impact the decision-making process.

- `Reviews`: It is important to know the success rate of the presented medicine and how the people who use it feel about it.

With this data, the aim is to provide the required information to the users.

2.2 Gathering

Finding data suitable for the project proved to be a challenge; not only did it have to be relevant and accurate, but it also had to meet some criteria in terms of quantity and quality. Therefore, the data had to be gathered from multiple sources using different methods - most of the data came from prepared datasets found on Kaggle[1], while the rest came from scraping Wikipedia[2].

2.2.1 Medicines. The *Medicines* dataset[3], retrieved from Kaggle, contains a list of pharmaceutical treatments and some relevant, mostly textual, information about the cases where it is used.

The present information on this dataset is:

- `Medicine Name`: The name of the medicine.
- `Composition`: The active substance present in the medicine.
- `Uses`: A list of cases where the medicine is used (specific diseases, for example).
- `Side Effects`: Lists possible side effects resulting from the medicine's usage.
- `Manufacturer`: The name of the company responsible for producing the medicine.
- `Reviews`: Three additional columns containing the percentage of "Excellent", "Average" and "Poor" reviews for each medicine's treatment results.

This dataset, which contains 11824 different medicines, possesses a ***CC0 1.0 Universal***[4] license, which means the data is part of the public domain, allowing for the copying and modification of the data.

2.2.2 Diseases. This dataset was originally scraped from the tables of a Wikipedia page containing a list of autoimmune diseases[5]; by gathering this data, the goal was to complement the previous dataset's "Uses" and "Side Effects" columns by collecting more information on this specific subset of diseases.

The information extracted from this dataset consists of:

- `Disease`: The name of the disease.
- `Primary Organ/Body Part Affected`: Information on the organs or body parts affected by the disease.
- `Autoantibodies`: The antibodies associated with each specific disease.
- `Acceptance as an Autoimmune Disease`: Classification for each disease related to its acceptance as an autoimmune condition, based on the current scientific consensus and level of evidence supporting its autoimmune nature.
- `Prevalence Rate (US)`: The percentage of people affected by the disease in the United States of America.

The gathered dataset contains about 110 lines of diseases, being available under the ***Creative Commons Attribution-ShareAlike 4.0 International***[6] license, which allows for the sharing and adaptation of the contents.

This dataset was expanded in the context of the third milestone with more data from the CUF website page *Sicknesses from A-Z*[7]. The gathering process encompassed the scraping of the website, normalization of its contents and translation from Portuguese to English using the *deep_translator* pakage; this new set of data effectively replaced the old one, consisting of:

- `Name`: The name of the disease.
- `Description`: A small description of the disease.
- `Symptoms`: Description of symptoms commonly associated with the disease.
- `Causes`: What may cause someone to contract the disease.
- `Diagnosis`: Information on how to monitor whether someone has, in fact, contracted the disease.
- `Treatment`: Common treatments for the particular disease (ranging from medication to surgeries).
- `Prevention`: How to avoid contracting the disease itself.

This dataset contains about 556 entries, being publicly available for usage and consultation. It allowed for a better understanding of our existing diseases while also complementing missing information.

2.2.3 Pharmaceutical Companies. To complement the data on companies present in the *Medicines* dataset, more data on pharmaceutical companies was gathered through another round Wikipedia scraping - this time from a page containing an extensive list of pharmaceutical companies[8]. Approximately 700 companies' names and founding dates were gathered, alongside a short description from each one's Wikipedia article.

The collected information follows this structure:

- `Company Name`: The company's name.
- `Year`: The year the company was created and, if available, when the company was shut down.
- `Description`: A short description of each company, its values and some extra information.

This dataset falls under the same license as the previous *Diseases* dataset (***Creative Commons Attribution-ShareAlike 4.0 International***), as the data was collected in a similar way.

2.2.4 Reviews. Lastly, a dataset containing reviews for the collected medication data with more personal descriptions from users[9] was retrieved from UC Irvine's Machine Learning Repository[10].

The data collected had the following structure:

- `Unique ID`: The unique identification of the review.
- `Drug Name`: Name of the drug/treatment.
- `Condition`: The condition of the patient where it was used.
- `Review`: Written review of the experience of taking the drug.
- `Rating`: Number from 0-10 that expresses the quality of the drug.
- `Date`: Date of when the drug was taken.
- `Useful Count`: Similar to a "like" system, this showcases the number of people who found this review useful.

Each review can only be associated with a single condition/disease, which is a limitation of the dataset itself.

This dataset contains around 215000 entries and is covered by the ***Attribution 4.0 International***[11] license, which allows for the

sharing and adaptation of the datasets for any purpose, provided that the appropriate credit is given.

3 Pipeline Description

As mentioned in section 3.2 of this report, ***PRIMED***'s data comes from various sources. Due to the often unstructured nature of this data, it is vital to have a streamlined and automated way of normalizing and processing all the information into similar formats, which is the main role of the data pipeline, present in figure [1].

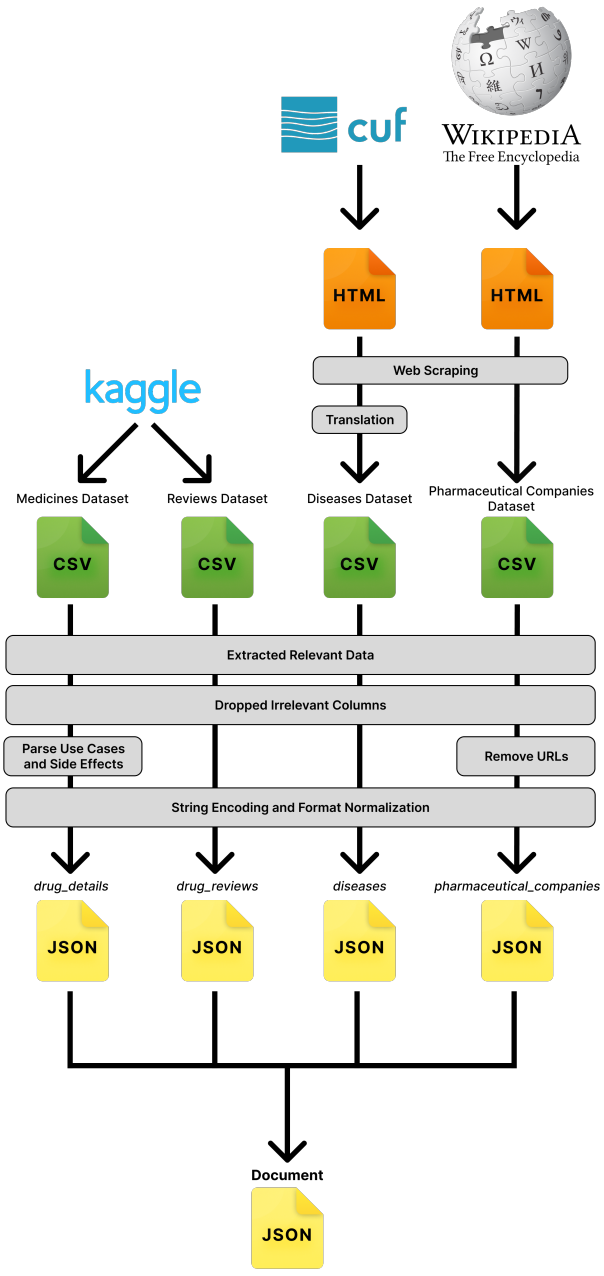

Figure 1: Data Pipeline

The pipeline consists of `Python`[12] scripts utilizing libraries such as `pandas`[13], `unidecode`[14], `html`[15], `json`[16] and `csv`[17] with the crux of the data processing occurring on the `to_json.py` script, which converts the `CSV` files into `JSON` format.

3.1 Elimination of Null Values and Rows

When processing data, another key aspect to consider is that not all data may be correct or even present. For this reason, before doing anything else with the `CSV` source files, the pipeline uses `pandas` *dataframes* to check for and remove rows containing only null values, or rows in which key values, such as `Medicine Name`, for instance, aren't present.

3.2 Text Normalization

When scraping information from websites, it is important to make sure all the text is normalized. The data pipeline ensures, via `Python`'s `unidecode` function, that the dataset only contains `ASCII` characters. This helps prevent future issues when searching the datasets for information.

Another character normalization problem resulting from the scraping process arises due to `HTML`'s nature - more specifically, escape codes used to represent special characters. To convert these codes into ASCII characters, the `unescape`[15] function from `Python`'s `html` module, wrapped inside a call to the previously mentioned `unidecode` function, to convert the `Unicode` output into `ASCII`.

3.3 Standardization of Formats

For the best possible result, formats such as dates should be standardized; therefore, part of the pipeline deals with transforming this data into `yyyy-mm-dd` format, which makes the process of sorting and searching substantially easier.

3.4 Data Storage

The final output of the pipeline (the processed data that the search engine will be working with) is stored in `JSON` format. This decision stems from the requirement of a document-based data storage model, where each `JSON` object represents a document encapsulating all the relevant fields and values for a particular record - in this case, a specific medicine. This approach provides flexibility in handling unstructured data, since there are no hard constraints in place.

This model also ensures the search engine can query and retrieve data efficiently, based on specific fields within these documents, ensuring that the system can scale as the dataset grows; it also supports complex data relationships and makes it easier to analyse information across different records.

4 Conceptual Data Model

Having finalized the data collection and the subsequent transformation and storage processes in the pipeline, a conceptual data model for the combined dataset was developed.

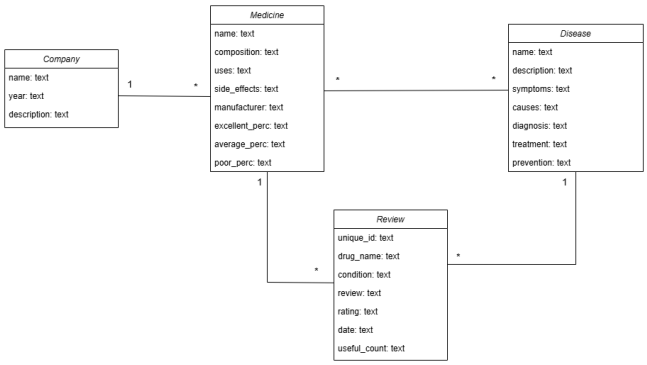

Figure 2: Conceptual Data Model

Each medicine is associated with one and only one pharmaceutical company; a company, however, can produce multiple medicines. A single medicine can be linked to multiple diseases and reviews, though the reverse is true only for diseases - a disease can be associated to many medicines (and many reviews). A review, however, can only be associated with a single medicine and a single disease.

5 Dataset Characterization

With regard to the characterization of the dataset, the data resulting from the pipeline was analysed. Graphs, tables and even word clouds, despite not being the most reliable method, were created to make it easier to understand the structure of the data and the patterns that guide the system itself. The final version of the dataset, representing the collection of documents, contains 11498 entries. Each entry represents a medicine along with its related information, such as associated diseases and possible side effects, user reviews, ratings and manufacturer details.

By analysing figure [8], which shows the distribution of the main pharmaceutical companies responsible for manufacturing medicines where the five that possess the largest share are highlighted, we are given a clearer understanding of each manufacturers' production capabilities.

A box plot, present in figure [9], is utilized to analyse the distribution of "Excellent", "Average" and "Poor" review percentages across the different medicines. The graph shows that, for instance, "Excellent" reviews percentages are more evenly spread than the "Average" reviews percentages which, despite having a lower interquartile range[18], possesses a higher number of outliers. The "Poor" review percentages have the largest variations, but also the lowest values. In figure [10], we can see the average percentage of reviews for the top ten use cases, which supports the findings from figure [9].

By analysing figure [11], we can identify the most common side effects caused by medication - nausea, headaches, diarrhea - and how many medicines are associated to each one of them. As expected, the data shows that side effects that occur more frequently are those that are milder in nature.

Figure [12] contains a word cloud with the most common use cases for medicines. "Bacterial Infections" and "Hypertension", for example, are clearly highlighted in the graph, being conditions that are found and treated more frequently.

Figure [13] identifies the the most common drug combinations available in the market, with "Levocetirizine + Montelukast" being the most prevalent, followed by other combinations such as "Luliconazole" and "Domperidone + Rabeprazole".

By analysing figure [14], it's possible to observe the years in which the most pharmaceutical companies were founded, with 2003 standing out. This temporal analysis gives us some information about periods of significant growth in the pharmaceutical industry.

6 Information Needs

The information needs can vary depending on the person using ***PRIMED***. If the individual who is using the tool is, for example, a doctor or a pharmacist, these needs might be related to the compositions of a medicine or cases where it can be applied. If the user is a patient (an average person), they might be more interested in checking which company manufactures the medicine to check if it can be trusted or to assess other user's reviews of that treatment, as well as possible side effects, to have a better understanding of what can happen to them.

Here is a list of possible information needs, explaining what type of data is needed, why and for whom:

- `Medicine Compositions`: The composition of medication is an important aspect since some might be harmful to the patient, depending on what kind of active substances are present.
- `Uses`: Not every medicine has clearly defined use cases; some may have a primary use while also being beneficial for other less common conditions.
- `Side Effects`: The users should be aware of possible side effects resulting from their treatment, so that they can be informed and prepared for potential symptoms or strange events.
- `Manufacturer`: Some people have a preference for specific manufacturers who they trust more than others, or can simply want to gather more information on the company producing the medicine they were prescribed.
- `Reviews`: For all possible types of users, existing reviews are always important - whether they are textual or not. These indicate other people's experiences with a particular medicine and can even report some peculiar cases where the patients experienced rare effects from that treatment.
- `Diseases' Primary Organs/Autoantibodies`: In some cases, experts might need to know what part of the body or organs are most affected by a disease, or what antibodies are linked to it, in order to prescribe the best solution.
- `Prevalence Rate`: It can also be relevant to be aware of how common a certain disease is in a population, whether that influences the diagnosis of the patient or just for statistical purposes.

Based one these needs, we can answer some questions like:

⇒ Identify pharmaceutical companies or organizations that develop, produce, or provide treatments for malaria.
⇒ Find customer reviews, opinions, or feedback regarding Eli Lilly's products, services, or the company's reputation.

⇒ Obtain information about the weight gain side effect caused by the use of antidepressant medications, including data on specific antidepressants.
⇒ Search for medication or treatments that are effective for managing or curing migraines.

As previously stated, a wide range of people can take advantage of this type of information - from regular patients to medicine experts and even investigators, everyone can benefit from having access to the data present in ***PRIMED***.

7 System Overview

Apache Solr[19] is a search and analysis platform which allows the indexation of big data volumes, while also enabling quick and efficient queries on that same data. It works by indexing documents from a collection and then efficiently associating words with the respective documents they appear in. Thanks to this process, the queries performed are faster, since when the user searches for a word, Solr checks the index; Solr also allows the usage of other "tools" via query parameters, such as boosters that enhance the querying process by prioritizing certain terms or fields, or fuzziness to try and mitigate the effects of typos on user queries.

7.1 Project Objectives

The primary objective of the project is to give users access to structured information for healthcare and research, focusing on medicines, associated diseases, user reviews or even manufacturers. The use of Solr means to allow the system to deliver an efficient querying process to the ***PRIMED*** data collection able to rapidly retrieve the desired results. Additionally, utilizing Solr's capabilities offers the opportunity to enhance the document collection itself by boosting specified fields in search results and creating of custom data types, among other features.

7.2 System Architecture

The search engine receives a document collection in `JSON` format, generated by the data pipeline previously implemented in `Python`. This collection is then indexed by Solr according to a preconfigured schema[20]. When a user submits a query to the system, it is sent to Solr via a `HTTP` request[21], which returns the results for the query in a `JSON` file.

The goal is then to evaluate the results obtained from Solr by using several different performance and precision metrics. Figure [15] illustrates the system's overall architecture.

8 Document Analysis

In this section, the structure of the documents is analysed, as well as the systems utilized to index the document collection and parse the queries made by the users, which are treated as a variable in each of the aforementioned system's configurations.

8.1 Document Characteristics

The final version of a document in the data collection, generated as a result of the previously implemented data pipeline, contains the following fields:

- `drug`: The name of the medicine in question.
- `composition`: The active substance(s) present in the drug.
- `applicable_diseases`: A list of diseases for which the medicine is usually taken or applied.
- `diseases_info`: A list of information on the medicine's applicable diseases.
- `possible_side_effects`: A list of possible side effects provoked by the drug when taken.
- `excellent_review_perc`: Percentage of reviews with a rating superior to 7.
- `average_review_perc`: Percentage of reviews with a rating between 4 and 7.
- `poor_review_perc`: Percentage of reviews with a rating inferior to 4.
- `reviews_average_rating`: The average review score for the medicine, based on all the reviews it received (rounded to two decimal places).
- `reviews`: Reviews associated with each medicine (in some specific cases, the association is made based on the active principle due to the reviewer not revealing the medicine's name itself).
- `manufacturer`: The name of the company responsible for producing a specific medicine.
- `manufacturer_desc`: A short description of each company, containing some extra information.
- `manufacturer_start`: The year the company was created.
- `manufacturer_end`: The year the company was shut down, if available.

However, only a few of these document attributes are utilized in the retrieval process; the query is performed solely on the `diseases_info`, `reviews` and `manufacturer_desc` fields in the index, since these are attributes which primarily contain unstructured text.

The three review percentage categories were defined to allow for a more detailed comparison between different medicines, particularly in cases when average review ratings are similar.

8.2 Schema Definition

Solr stores details about the field types and which fields it is expected to understand in a schema. A schema allows for a specific search system to be configured on the indexing process. For system and performance evaluation purposes, a decision was made to create two distinct systems.

As a result, two different schemas were created - one for the baseline system, with minimal configuration, and another, more advanced schema for the improved system.

Since every field has the *stored* property set to *true* in both schemas, it won't be displayed in the upcoming (advanced) schema table. Of all the fields present, only two are not indexed (have the *indexed* property set to *false*): `manufacturer_start` and `manufacturer_end`, since they are only used to display additional information about a medicine's manufacturer and are not actively used in any search tasks. The only fields with the *multiValued* property enabled are those that store a list of strings: `reviews`, `diseases_info`, `applicable_diseases` and `possible_side_effects`.

For the basic schema, only native Solr data types were used - no additional field types were created. For this reason, no tokenizers[22] or filters[23] were added in the basic schema; however, the text

fields utilize the *text_general* type, which tokenizes with a standard tokenizer. The percentage fields type, as well as the field representing the average rating of the reviews, were assigned the *pdouble* type.

For the advanced schema, since the goal is to perform queries on textual fields, a new field type, ***textBoosted***, was created, based on the *TextField* class, which makes use of both tokenizers and filters - while the tokenizers are responsible for breaking field data into lexical units (tokens), filters examine the stream of tokens and keep them, discard them or transform them depending on the specific filter being used. The ***textBoosted*** field type includes:

⇒ ***StandardTokenizerFactory*:** Tokenizer that uses whitespaces and punctuation as delimiters to split text into tokens.
⇒ ***ASCIIFoldingFilterFactory*:** Filter that converts Unicode characters like special characters or characters with accents into their ASCII equivalents, if one exists.
⇒ ***LowerCaseFilterFactory*:** Filter that converts all letters in a token to lowercase.
⇒ ***SnowballPorterFilterFactory*:** Filter that applies stemming[24] - the process of reducing a word to its root form (or *stem*) - to tokens during text processing by generating pattern-based word stemmers. While less accurate than table-based stemmers, it is faster and less complex. It was also chosen for its aggressive stemming[1], making it suitable to normalize word variations in the reviews and manufacturer descriptions.

Two additional field types, similar to ***textBoosted***, were also created:

- ***diseasesBoosted***: Very similar to ***textBoosted***, but relies on ***EnglishMinimalStemFilterFactory*** for stemming instead, which is a lightweight filter, better suited to handle fields that contain more technical terms - particularly the diseases' information.
- ***shortText***: Very similar to ***textBoosted*** but without a stemming filter, as it is applied to short textual fields that contain very specific terms.

For each of three field types, the same tokenizer and filters were used in both the query analyser and the index analyser.

The table below provides an overview of the advanced schema's structure, detailing the field names, their respective types and whether they are indexed or multi-valued.

[1]The *SnowballPorterFilterFactory* filter often applies more drastic reductions to words than other stemming filters; for instance, the words *general*, *generous*, *generation* can be reduced to the stem ***gener***, despite having different meanings.

Field	Type	Indexed	Multi-Valued
drug	shortText	✓	✗
composition	shortText	✓	✗
applicable_diseases	shortText	✓	✓
diseases_info	diseasesBoosted	✓	✓
possible_side_effects	shortText	✓	✓
excellent_review_perc	pdouble	✓	✗
average_review_perc	pdouble	✓	✗
poor_review_perc	pdouble	✓	✗
reviews_average_rating	pdouble	✓	✗
reviews	textBoosted	✓	✓
manufacturer_desc	textBoosted	✓	✗
manufacturer	text_general	✓	✗
manufacturer_start	text_general	✗	✗
manufacturer_end	text_general	✗	✗

Table 1: Advanced Schema

8.3 Retrieval Process

With both schemas implemented, the next step in the system development was to configure the query parsers and their respective parameters. Both systems utilize the Extended DisMax (eDisMax) query parser[25], which is an improved version of the DisMax query parser[26]. Despite neither the baseline nor the advanced making use of any parameters specific to the eDisMax parser, it remains more flexible than the DisMax parser, while also offering additional features that could prove useful in future developments of the search engine.

For the query parameters used by the aforementioned schemas, the following common ones are utilized:

- `q`: Defines the raw input string for the query, inserted by the user.
- `q.op`: Applies a logical OR/AND to query operations.
- `sort`: Specifies the way the documents returned are to be sorted.
- `start`: Specifies an offset applied to the documents returned from a query being displayed (decides whether Solr should skip over a set number of documents).
- `rows`: Sets the maximum number of documents to be returned as a result.

Additionally, other parameters focused on optimizing search results are used:

- `qf` (**Field Boosts**): Represents list of fields that are assigned a particular boost factor, which increases (or decreases) its importance in the query.
- `pf` (**Phrase Match**): Once the list of matching documents has been identified by the `qf` parameter, `pf` can be used to boost the score of documents in which all of the terms present in the user's query appear in close proximity.
- `ps` (**Phrase Slop**): Represents the amount of phrase slop[27] to apply to queries specified with the pf parameter.
- `bf` (**Independent Boosts**): Used to apply additional boosting to the documents based on certain fields or functions, independent of a document's score calculated by other parameters.

Other techniques, external to the query parameters, were explored via direct application to the user's query within the system. However, due to either constraints in the system specifications and query handling or worse overall performance, techniques like term boosting[28] and fuzziness[29] were ultimately not included in the final version of the advanced system.

The baseline system exclusively uses the common parameters, along with the `qf` parameter, to process a query and attempt to find matches in the designated query fields (`diseases_info`, `reviews` and `manufacturer_desc`); however, all three fields have the same weight in the ranking process. The system then returns the top 30 ranked documents, sorted in descending order based on their average review rating.

Parameter	Value
q	$query
q.op	AND
sort	reviews_average_rating desc
start	0
rows	100
qf	diseases_info reviews manufacturer_desc

Table 2: Simple Query Parser

Alternatively, the advanced system makes use of the all the common and optimization parameters, with the exception of the `sort` parameter, allowing for theoretically better results to be achieved when processing a query. In this case, the common parameters have the same values as the ones in the baseline system, but the designated query fields, albeit similar in structure to those in the baseline system, have different weights in the ranking process: the `reviews` field has a weight of 4 and the `diseases_info` field has a weight of 3, as they are the most important sources of information; `manufacturer_desc` maintains a weight of 1.

The document receives an additional boost if all of the terms in the query are found in close proximity of each other (exact phrase match) within the `reviews` field, due to the more informal nature its text. Nevertheless, to account for some variation in word order or the presence of extra words, a phrase slop of 2 is applied to allow for more flexibility when searching for the phrase matches. We settled on the value of 2 for the phrase slop parameter after some trial and error - it seemed to be the value that provided the best balance between the tolerance for minor variations in word order and precision.

Finally, an independent boost is added, based on the `excellent_review_perc` and `poor_review_perc` fields, with values of 1.5 and 0.5, respectively. This adjusts the scoring of each document based on its respective percentage of positive and negative reviews, effectively replacing the sorting utilized in the baseline system.

Parameter	Value
q	$query
q.op	AND
start	0
rows	100
qf	diseases_infoˆ3 reviewsˆ4 manufacturer_desc
pf	reviewsˆ3
ps	2
bf	excellent_review_percˆ1.5 poor_review_percˆ0.5

Table 3: Advanced Query Parser

9 System Evaluation

This chapter outlines the evaluation process used to assess the precision of the search engine - that is, its ability to accurately retrieve the data a user is searching - as well as the metrics employed for this purpose.

9.1 Ground Truth Development

To evaluate the performance of each search engine system, we manually created QRELS[30] files - one for each information need selected for the evaluation process. This involved searching through our data collection for suitable medicines and selecting the top 30 documents for each information need.

Each QRELS file was created manually; despite being a cumbersome task, it ensured that each file contained a list of relevant documents for the corresponding information need, which was critical for evaluating the precision of each version of the search engine's retrieval process.

9.2 Performance Metrics

To evaluate the effectiveness of the information retrieved from querying, specific metrics were established and analysed. The evaluation process, however, requires predefined queries, derived from specific information needs, to be utilized:

⇒ *Q1*: Companies with treatment for malaria.
⇒ *Q2*: Reviews from Eli Lilly.
⇒ *Q3*: Weight gain from antidepressants.
⇒ *Q4*: Medicine for migraines.

Each one of these queries was properly tested, with the number of documents retrieved for each query in both systems being recorded.

Query	Simple Results	Advanced Results
Q1	30	60
Q2	54	61
Q3	1391	1573
Q4	3897	4842

Table 4: Query Results

9.2.1 Precision@k. To evaluate the performance of our systems, the P@k metric[31] is used, with the choice being made of evaluating the first ten documents returned for each query. This metric

gauges the accuracy of the system by analysing the top k results and calculating the proportion of those documents that are relevant.

9.2.2 Average Precision. The Average Precision[32], or AP, is a commonly used performance metric that calculates the average of the precision values at every position in the ranked collection where relevant documents are retrieved (and in which the order of retrieval of these same documents is considered).

9.2.3 P-R Curve. A Precision-Recall (P-R) Curve[33] is calculated for each individual query and system, based on the ranked collection of documents returned by Solr. For instance, the larger the area under the P-R curve[34], the better a system performs.

9.3 Performance Analysis

In this section, the results for every single query in both systems will be examined by analysing the different values for each performance metric, which will help to determine the accuracy of each system in regards to retrieving relevant information. For each query, only the first 30 results were utilized for metrics calculation.

Query	Simple P@10	Advanced P@10
Q1	0.8	0.9
Q2	0.9	1
Q3	0.7	1
Q4	1	1

Table 5: P@10 Results

The Average Precision is calculated for each query result; in this particular case the values obtained via the P-R Curve graph, as it provides the AP value for all queries and systems. The figures containing each P-R Curve are placed in the appendix.

Query	Simple AP	Advanced AP
Q1	0.8661	0.8934
Q2	0.8632	0.8719
Q3	0.3805	0.5361
Q4	0.3329	0.4095

Table 6: AP Results

As we expected, the values of the P@k and AP in the advanced system (represented with an *A*) consistently outperformed or, at the very least, matched those of the baseline system (represented with an *S*), indicating that there's a clear advantage in using not only the advanced schema, but the advanced query parameters as well.

10 System Improvements

Having created two initial versions of the information retrieval system, there was a subsequent need improve upon the foundations of the search engine in order to mitigate possible limitations present in the previously developed systems.

Due to the clear performance difference between the baseline and advanced systems, there was no clear path to follow regarding possible enhancements to the system. At first, ideas like query processing utilizing external tools, to explore links between symptoms and diseases and subquerying certain fields within the `bf` parameter in the schema sounded promising; however, they ended up not being implemented. Query processing utilizing external tools was functional, and an interesting idea overall, but perhaps due to the nature of our data and, subsequently, our queries, there were no significant gains in terms of overall accuracy (in fact, the opposite occurred). Regarding the subquerying within the `bf` parameter, it was indeed valid, but generating dynamic subqueries was troublesome and it never appeared to be able to yield great performance returns.

For this reason, we decided to focus on three pillars regarding the system improvements: a polarity analysis-based metric[35], query re-ranking[36] using the ReRank[37] query parser and semantic search[38], utilizing dense vectors embeddings[39].[2]

10.1 Polarity Analysis

The concept of polarity is essentially about analysing the overall tone of a particular text (whether it's negative, neutral or positive - sentiment analysis) and scoring it from a range of -1 (very negative) to +1 (very postive). For this reason, we deemed this an interesting metric to further enhance our search system, since it would allow us to strike a better balance between the review scores and the actual sentiment of the reviews' text. The review scores offer an objective evaluation of a particular medicine, but polarity/sentiment analysis is capable of performing a deeper (and perhaps more meaningful) analysis of a review's contents by examining the tone of the review and the way the text itself was crafted.

To calculate our custom polarity rating, we used the NLTK[40] package, specifically the VADER sentiment analysis tool[41]. A `SentimentIn-tensityAnalyzer` object was created, and for each document in the collection the `polarity_scores` method was utilized to retrieve a float value representing the sentiment strength for each review in a particular document. Next, the reviews average rating for each document was normalized, so as to fit in the $[-1, 1]$ range. Finally, the polarity rating was calculated as the average of the sum of the normalized reviews average rating and the mean of the polarity scores from all reviews in a document.

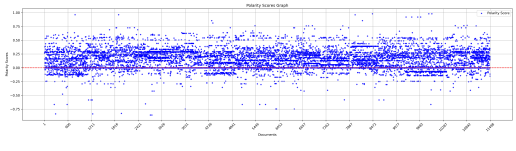

Figure 3: Polarity Scores Distribution

[2]Despite being analysed independently, the polarity analysis-based metric and query re-ranking are deeply connected, since the metric itself is utilized for the query re-ranking process.

By analysing the graph containing the distribution of the polarity scores across all documents in the collection [3], we can actually determine that there's a much larger amount of documents with positive polarity scores than with negative ones. As the last step, a ***polarity_rating*** field of type *pdouble* was created for the schema.

10.2 Re-ranking

The goal of the previously mentioned polarity analysis was to provide a more robust document ranking process by making sure the contents of the reviews for a specific medicine were in line with the reviews average rating and, therefore, with the needs of the users.

By utilizing the metric from the previous section, we were able to implement a query re-ranking system based on the weighted sum of the average review and polarity ratings obtained for each document.

Query re-ranking basically allows a primary query to be ran for matching documents; from this set of documents, the top N can be re-ranked using the scores from a secondary (and usually more specific) query. Despite allowing for a more efficient system (due to the usually more expensive secondary query only being performed on the top N documents as opposed to every document retrieved), the downside to this is that documents that could potentially score better with this secondary query aren't taken into consideration when re-ranking.

The `rerank` parser utilizes the following parameters:

- `reRankQuery`: Defines the query string to be utilized as the ranking query (a variable in most cases).
- `reRankDocs`: Top N documents from the original query that should be re-ranked (acts as the minimum number of documents to re-rank).
- `reRankWeight`: Factor applied to the score of each re-ranked document.

As mentioned earlier, the re-ranking query contains the weighted sum of the average review and polarity ratings, obtained for each document. It was decided that the reviews average rating would still have more influence in the result due to the data skew observed in the polarity scores graph [3]. As for the number of documents, we arbitrarily decided that 30 would be an appropriate number, both to verify improvements in the system, the reduced number of results that some queries yield and due to the slightly intensive nature of the query in terms of computational power that might eventually be required. In the weight parameter, we decided to utilize the default value present in the ReRank query parser.

Parameter	Value
rq	!rerank reRankQuery=$rqq reRankDocs=30 reRankWeight=2.0
rqq	!funcsum(product(reviews_average_rating, 4), product(polarity_rating, 2))

Table 7: ReRank Query Parser Fields

It was then decided to test the influence of the re-ranking process on the system's performance by incorporating the re-ranking fields into the advanced system, in order to compare its performance to the "regular" advanced system.

Query	Re-Ranked AP
Q1	0.8133
Q2	0.8288
Q3	0.1878
Q4	0.1953

Table 8: Re-rank AP Results (Advanced System)

As shown in the P-R Curve graphs for queries 1 [24] and 2 [25], the loss in average precision in the advanced system utilizing the ReRank query parser, albeit not negligible, is minimal. However, this doesn't mean the system itself isn't performing a more suitable ranking of the relevant documents.

However, in the P-R Curve graphs for queries 3 [26] and 4 [27] there is a sharp drop both in terms of average precision and recall.

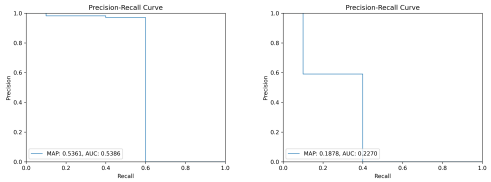

Figure 4: Q3 P-R Curve Comparison -> Advanced System without and with Re-Ranking

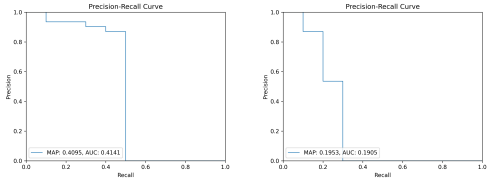

Figure 5: Q4 P-R Curve Comparison -> Advanced System without and with Re-Ranking

These abnormal results may be a consequence of Solr's ability to internally increase (automatically) the value of `reRankDocs` in order to rank enough documents to satisfy the query, which means the actual value of this parameter for queries 3 and 4 (which return a large number of relevant documents) might actually be, for instance, the number of rows, as stated in Solr documentation for the ReRank query parser.

10.3 Semantic Search

The goal of the semantic search utilization is to generate more accurate queries by focusing on understanding the contextual meaning of the terms, rather than relying solely on keyword matches. This approach makes use of dense vectors that transform words or phrases into numerical representations, which allow the system to find correlations between concepts.

Semantic search was implemented in the project using vectors for the `reviews` field; we were unable able to implement it on any extra fields due to the very high computational cost of generating embeddings. These embeddings are generated by AI language models[42] capable of capturing semantic nuances in data - relationships between the queries and diseases, side effects or even reviews, if implemented correctly.

This approach is expected to allow the search system to deliver more relevant and accurate results, even when there is no direct lexical match between the query terms and the indexed documents. As a result, semantic search theoretically enhances the interaction with data, making it more intuitive and efficient ensuring the query results are in line with the context and meaning of that same query. The main difference when compared to the other systems was the change to the `q` field, which now matches documents based on vector similarity, using an approach based on the K-Nearest Neighbors (KNN) algorithm [43], and the embeddings themselves. It then returns the top 100 documents in terms of similarity.

For the semantic system's schema, a new field type, ***diseaseVector***, was created, based on the Solr *DenseVectorField* class, and a `vector` field was added to the schema. The semantic system's query parser parameters are as follows:

Parameter	Value
q	!knn f=vector topK=100
q.op	AND
start	0
rows	100

Table 9: Semantic Query Parser

No boosts were used due to the particular characteristics of the semantic search process. Having set the query parser parameters for the semantic system, we decided evaluate its performance by plotting the P-R Curves for each one of the four queries previously utilized in both the baseline and advanced systems.

Query	Semantic AP
Q1	0.0414
Q2	0.0092
Q3	0.0077
Q4	0.0663

Table 10: Semantic System AP Results

We expected a significant drop in performance when compared to the baseline and advanced systems, due to the high precision of the results registered in those systems (that would always be difficult to improve upon); however, we did not anticipate such an extreme drop in precision. This could be caused by a wide number of factors: no meaningful relationships between the queries and the documents themselves, the model used for the embeddings not being suitable for the particular nature of our dataset, data skews in the dataset, the lack of boosts in the semantic system's query parameters (which themselves were undoubtedly a factor that contributed to the improved results in the advanced system) or even issues with the QRELS themselves.

This search system only focuses on the generated embeddings, however. One approach to potentially fix this issue could involve the incorporation of a hybrid approach into the search system, which would combine lexical search and embeddings in a single query[44]. This would require the use of the Boolean query parser[45] to build hybrid queries utilizing various boolean conditions, which themselves would ensure a high degree of flexibility and perhaps ensure better results overall.

11 User Interface

A web-based user interface was also developed for the ***PRIMED*** search system to provide users with a more accessible interface for interacting with the various systems present in the search engine. The design prioritized usability over frontend aesthetics or even backend complexity, due to the nature of the project.

11.1 Technologies

The user interface was built using `Rails`[46], which utilizes `Ruby`[47] as the main language, with dependency management configured through `Bundler`[48]. It can support `Docker`[49] containers, which would make an eventual deployment easier to due its controlled environment.

The user interface's project structure is organized as follows:

- **app**: Contains the core logic of the application, such as controllers, models, and views[50].
- **config**: Includes configuration files required for the application's operation.
- **public**: Stores static assets such as images, `CSS` files, and `JavaScript` files.
- **test**: Contains test cases for validating the website's functionality.
- **vendor**: Stores external dependencies not managed directly by `Bundler`.

The project dependencies are managed using the `Gemfile` and `Gemfile.lock` files, which define the libraries required for execution. Additionally, the `Dockerfile` and `.dockerignore` file configure the runtime environment for `Docker` containers.

11.2 Web Interface

The search engine's web interface (for testing purposes, hosted locally on one's machine) consists of an index page [6] that acts as the website's landing page. In it, the user has access to a text input field where queries can be inserted and the ability to switch between three system options - each representing one of the different systems supported by the search engine. This option provides an easy way to switch between systems (schemas and query parameters) - baseline, advanced, and the semantic search. To achieve this, however, it is required to be running multiple cores, so that the end user can quickly switch systems without requiring a reset of the `Docker` container.

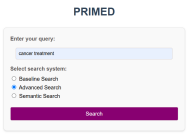

Figure 6: User Interface - Landing Page

Once a query is submitted, the interface redirects the user query and the system option to the page controller's `Ruby` script, which then creates an instance variable[51] containing the query and makes calls to the appopriate `Python` scripts, depending on the chosen system. As before, the results from the query are returned from Solr in `JSON` format and dumped into a file.

After the file is created, the page controller parses its contents and redirects the user, alongside the response contents, to the search page [7]. Here, the relevant documents retrieved from Solr are displayed in a more user-friendly format, compared to a raw `JSON` file. This interface allows the user to easily consult the most important aspects of a medicine relevant to their query: the drug's name, its composition, applicable diseases, the average review rating and the reviews themselves.

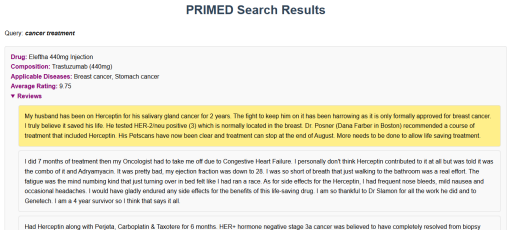

Figure 7: User Interface - Search Page

12 Conclusions

Completing these project milestones marks a significant step in the development of our system. The work carried out to this date has made it possible to transform an initial set of diverse, loosely related data into a cohesive and structured basis for ***PRIMED***. By carrying out data cleaning operations, we have been able to maintain the quality of the information while simultaneously reducing inconsistencies and duplicates. With this refined data collection, we created two initial versions of the ***PRIMED*** search engine by indexing the data in Solr, performing querying operations in both of these systems, and evaluating and comparing the subsequent results. The biggest challenge was perhaps the creation of the third version of the search engine, utilizing semantic search, due to the domain-specific nature of the ***PRIMED*** dataset and the computational requirements for generating embeddings for it. Nevertheless, there are still clear improvements that could be made to the search system overall that would more than likely increase its precision and efficiency, such as implementing the hybrid search system mixing both the lexical and semantic search processes, as well as using synonym filters in the schema or even utilizing Learning To Rank.

References

[1] Kaggle. https://www.kaggle.com, 2024.
Accessed on: November 9, 2024.
[2] Wikipedia. https://www.wikipedia.org/, 2024.
Accessed on: November 9, 2024.
[3] Kaggle. 11000 medicine details. https://www.kaggle.com/datasets/singhnavjot2062001/11000-medicine-details, 2024.
Accessed on: November 9, 2024.
[4] Creative Commons. Cc0 1.0 universal. https://creativecommons.org/publicdomain/zero/1.0/.
Accessed on: November 9, 2024.
[5] Wikipedia. https://en.wikipedia.org/wiki/List_of_autoimmune_diseases, 2024.
Accessed on: November 9, 2024.
[6] Creative Commons. Creative commons attribution-sharealike 4.0 international license. https://en.wikipedia.org/wiki/Wikipedia:Text_of_the_Creative_Commons_Attribution-ShareAlike_4.0_International_License.
Accessed on: November 9, 2024.
[7] CUF. Saúde de a-z. https://www.cuf.pt/saude-a-z.
Accessed on: December 2, 2024.
[8] Wikipedia. https://en.wikipedia.org/wiki/List_of_pharmaceutical_companies, 2024.
Accessed on: November 9, 2024.
[9] Surya Kallumadi and Felix Grer. Drug reviews. https://archive.ics.uci.edu/dataset/462/drug+review+dataset+drugs+com, 2018. UC Irvine Machine Learning Repository
Accessed on: November 9, 2024.
[10] UC Irvine. https://archive.ics.uci.edu/.
Accessed on: November 9, 2024.
[11] Creative Commons. Attribution 4.0 international. https://creativecommons.org/licenses/by/4.0/legalcode.
Accessed on: November 9, 2024.
[12] Python Software Foundation. *Python Programming Language*. Python Software Foundation, 2023. Version 3.x
Accessed on: November 9, 2024.
[13] The pandas development team. *pandas: Python Data Analysis Library*, 2023. Version 2.x
Accessed on: November 9, 2024.
[14] Takahiko Wada. *unidecode: Text transliteration for Python*, 2023. Version 1.x
Accessed on: November 9, 2024.
[15] Python Software Foundation. *html: HTML and XHTML processing for Python*, 2023. Python Standard Library
Accessed on: November 9, 2024.
[16] Python Software Foundation. *json: JSON encoder and decoder for Python*, 2023. Python Standard Library
Accessed on: November 9, 2024.
[17] Python Software Foundation. *csv: CSV file reading and writing for Python*, 2023. Python Standard Library
Accessed on: November 9, 2024.
[18] Pritha Bhandari. How to find interquartile range (iqr) | calculator & examples. https://www.scribbr.com/statistics/interquartile-range/, 2023.
Accessed on: November 9, 2024.
[19] Sematext. Getting started with apache solr. https://sematext.com/guides/solr/. solr introduction and explanation
Accessed on: November 11, 2024.
[20] Apache Software Foundation. Solr's schema file. https://solr.apache.org/guide/6_6/overview-of-documents-fields-and-schema-design.html#solr-s-schema-file, 2017.
Accessed on: November 11, 2024.
[21] MDN. Http request methods. https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods, 2024.
Accessed on: November 17, 2024.
[22] Apache Software Foundation. Tokenizers. https://solr.apache.org/guide/solr/latest/indexing-guide/tokenizers.html.
Accessed on: November 11, 2024.
[23] Apache Software Foundation. Filters. https://solr.apache.org/guide/solr/latest/indexing-guide/filters.html.
Accessed on: November 11, 2024.
[24] Jacob Murel Ph.D. and Eda Kavlakoglu. What is stemming? https://www.ibm.com/topics/stemming, 2023.
Accessed on: December 7, 2024.
[25] Apache Software Foundation. The extended dismax query parser. https://solr.apache.org/guide/6_6/the-extended-dismax-query-parser.html, 2017.
Accessed on: November 13, 2024.
[26] Apache Software Foundation. The dismax query parser. https://solr.apache.org/guide/6_6/the-dismax-query-parser.html, 2017.
Accessed on: November 13, 2024.

[27] Apache Software Foundation. Using 'slop'. https://solr.apache.org/guide/8_6/the-extended-dismax-query-parser.html#using-slop, 2020.
Accessed on: November 17, 2024.
[28] Apache Software Foundation. Boosting a term. https://solr.apache.org/guide/7_2/the-standard-query-parser.html#boosting-a-term-with, 2024.
Accessed on: December 14, 2024.
[29] Apache Software Foundation. Fuzzy searches. https://solr.apache.org/guide/6_6/the-standard-query-parser.html#TheStandardQueryParser-FuzzySearches, 2024.
Accessed on: December 14, 2024.
[30] Sérgio Nunes. Understanding relevance judgements. https://gitlab.up.pt/pri/tutorials/-/blob/main/06-evaluation/README.md?ref_type=heads#2-understanding-relevance-judgements, 2024.
Accessed on: November 17, 2024.
[31] Keylabs. Understanding precision at k (p@k). https://keylabs.ai/blog/understanding-precision-at-k-p-k/.
Accessed on: November 17, 2024.
[32] Deval Shah. Mean average precision (map) explained: Everything you need to know. https://www.v7labs.com/blog/mean-average-precision, 2022.
Accessed on: November 17, 2024.
[33] Doug Steen. Precision-recall curves. https://medium.com/@douglaspsteen/precision-recall-curves-d32e5b290248, 2020.
Accessed on: November 17, 2024.
[34] Amber Roberts. What is pr auc? https://arize.com/blog/what-is-pr-auc/, 2022.
Accessed on: November 17, 2024.
[35] Thematic. Sentiment analysis: Comprehensive beginners guide. https://getthematic.com/sentiment-analysis.
Accessed on: December 13, 2024.
[36] Apache Software Foundation. Query re-ranking. https://solr.apache.org/guide/6_6/query-re-ranking.html, 2024.
Accessed on: December 14, 2024.
[37] Apache Software Foundation. Rerank query parser. https://solr.apache.org/guide/solr/latest/query-guide/query-re-ranking.html#rerank-query-parser, 2024.
Accessed on: December 14, 2024.
[38] Sérgio Nunes. Solr semantic search. https://gitlab.up.pt/pri/tutorials/-/blob/main/07-semantic-search/README.md?ref_type=heads, 2024.
Accessed on: December 11, 2024.
[39] Alessandro Benedetti. Apache solr neural search. https://sease.io/2022/01/apache-solr-neural-search.html, 2022.
Accessed on: December 12, 2024.
[40] NLTK Project. Natural language toolkit. https://getthematic.com/sentiment-analysis, 2024.
Accessed on: December 13, 2024.
[41] C.J Hutto and E.E. Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. Eighth International Conference on Weblogs and Social Media (ICWSM-14), 2014.
Accessed on: December 13, 2024.
[42] Cloudflare. What is a large language model (llm)? https://www.cloudflare.com/learning/ai/what-is-large-language-model/.
Accessed on: December 14, 2024.
[43] IBM. What is the knn algorithm? https://www.ibm.com/topics/knn.
Accessed on: December 16, 2024.
[44] Alessandro Benedetti. Hybrid search with apache solr. https://sease.io/2023/12/hybrid-search-with-apache-solr.html, 2023.
Accessed on: December 13, 2024.
[45] Apache Software Foundation. Boolean query parser. https://solr.apache.org/guide/solr/latest/query-guide/other-parsers.html#boolean-query-parser, 2024.
Accessed on: December 14, 2024.
[46] The Rails Foundation. Ruby on rails. https://rubyonrails.org/, 2024.
Accessed on: December 14, 2024.
[47] Ruby Community. Ruby programming language. https://www.ruby-lang.org/en/, 2024.
Accessed on: December 14, 2024.
[48] Bundler. Bundler. https://bundler.io/, 2024.
Accessed on: December 16, 2024.
[49] Docker Inc. Docker. https://www.docker.com/, 2024.
Accessed on: December 16, 2024.
[50] MDN. Mvc. https://developer.mozilla.org/en-US/docs/Glossary/MVC, 2024.
Accessed on: December 17, 2024.
[51] Ruby User's Guide. Instance variables. https://ruby-doc.org/docs/ruby-doc-bundle/UsersGuide/rg/instancevars.html.
Accessed on: December 9, 2024.

A Annexes

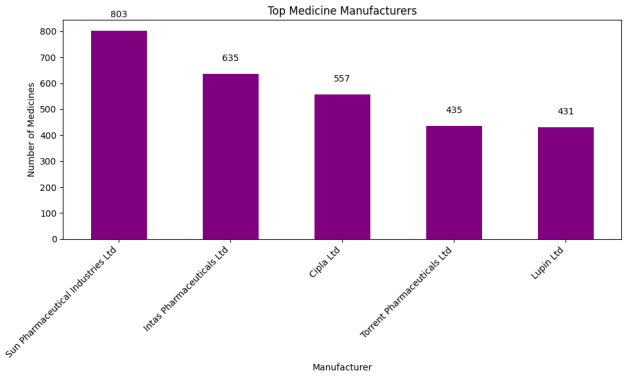

Figure 8: Top Drug Manufacturers

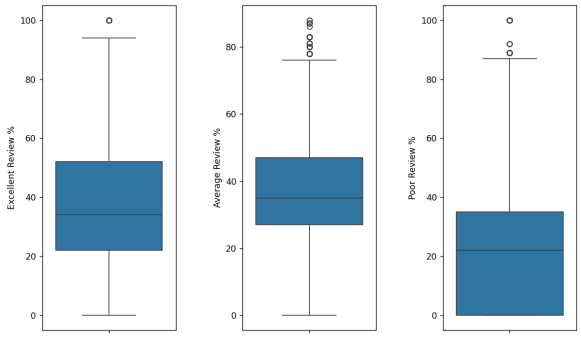

Figure 9: Distribution of Reviews

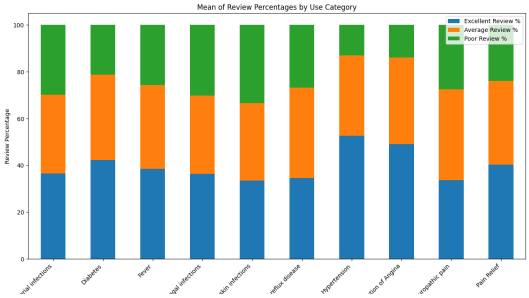

Figure 10: Mean of Reviews Percentages by Uses

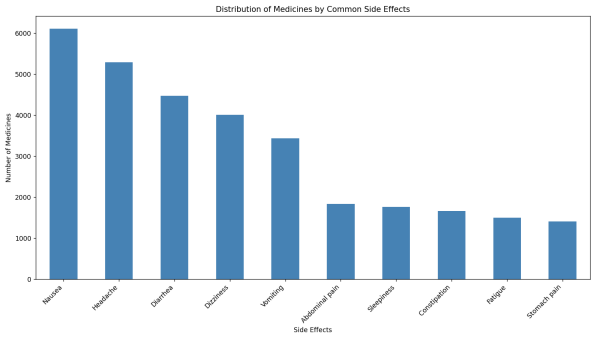

Figure 11: Distribution of Side Effects

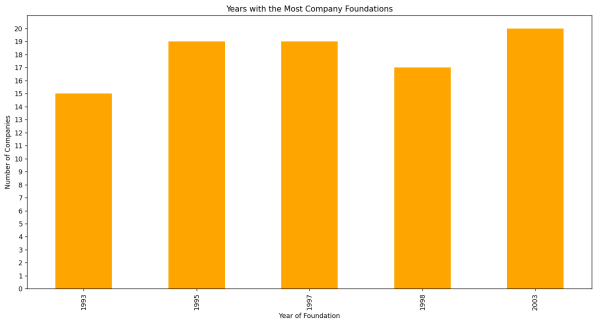

Figure 14: Years with the Most Companies Foundations

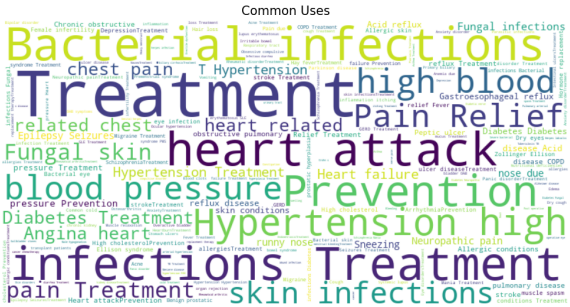

Figure 12: Common Uses Word Cloud

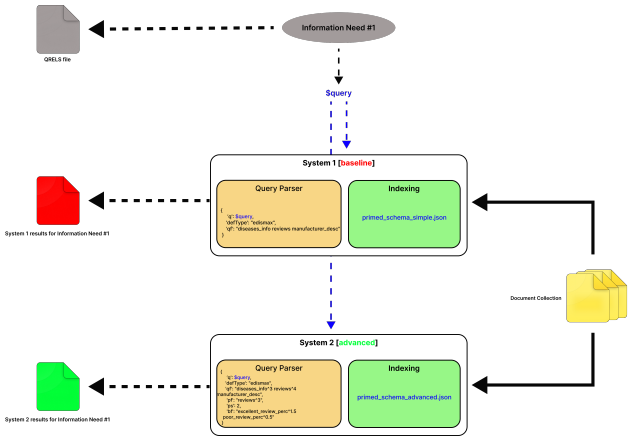

Figure 15: System Architecture

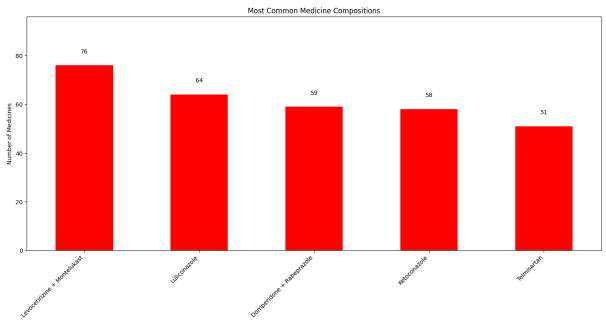

Figure 13: Most Common Drug Compositions

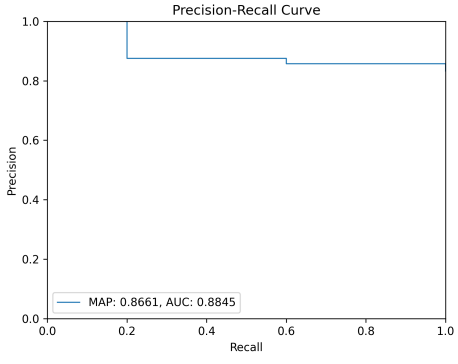

Figure 16: Q1 P-R Curve -> Baseline System

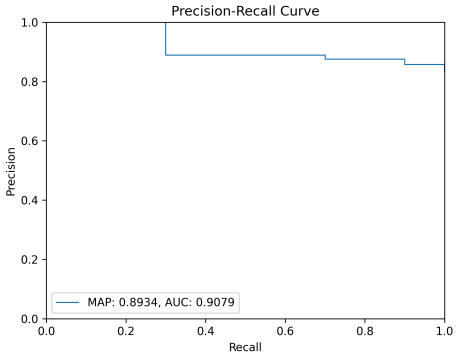

Figure 17: Q1 P-R Curve -> Advanced System

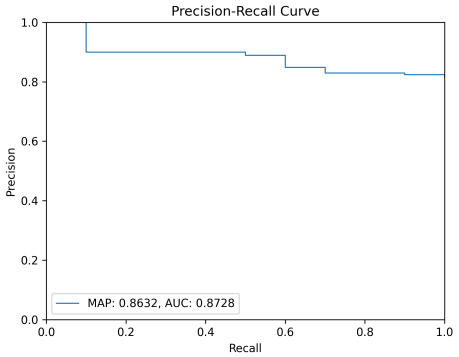

Figure 18: Q2 P-R Curve -> Baseline System

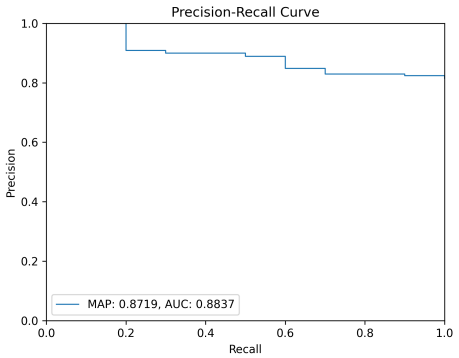

Figure 19: Q2 P-R Curve -> Advanced System

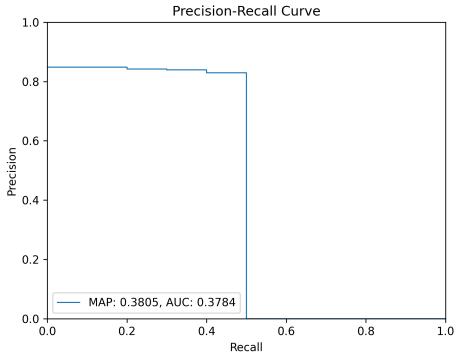

Figure 20: Q3 P-R Curve -> Baseline System

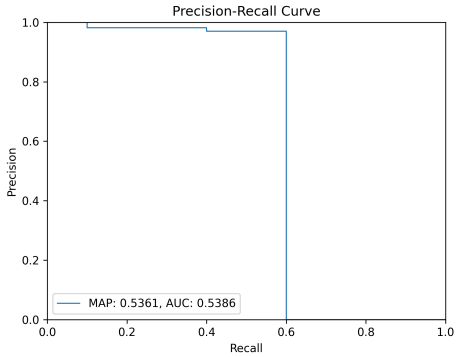

Figure 21: Q3 P-R Curve -> Advanced System

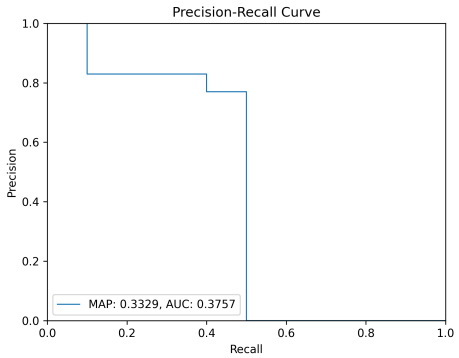

Figure 22: Q4 P-R Curve -> Baseline System

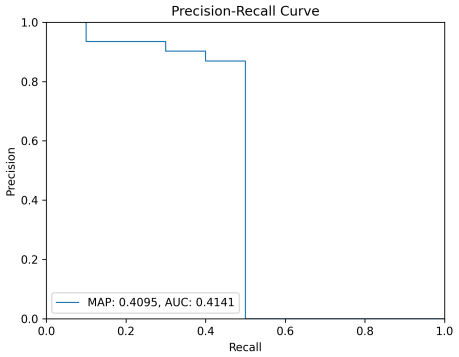

Figure 23: Q4 P-R Curve -> Advanced System

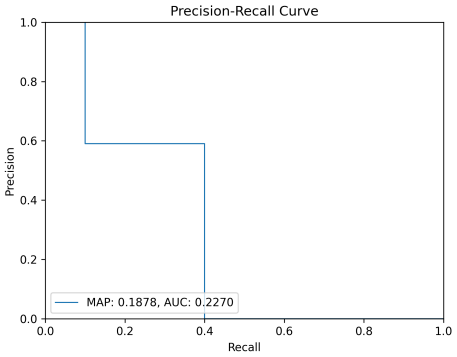

Figure 26: Q3 P-R Curve -> ReRank Advanced System

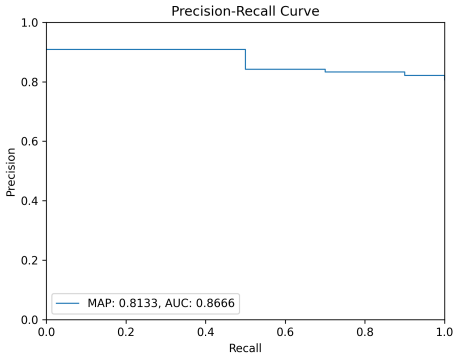

Figure 24: Q1 P-R Curve -> ReRank Advanced System

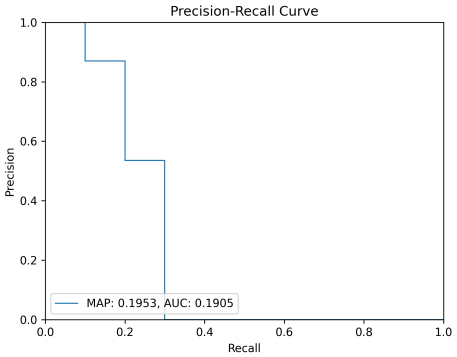

Figure 27: Q4 P-R Curve -> ReRank Advanced System

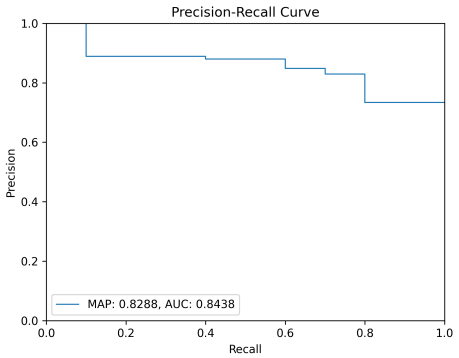

Figure 25: Q2 P-R Curve -> ReRank Advanced System

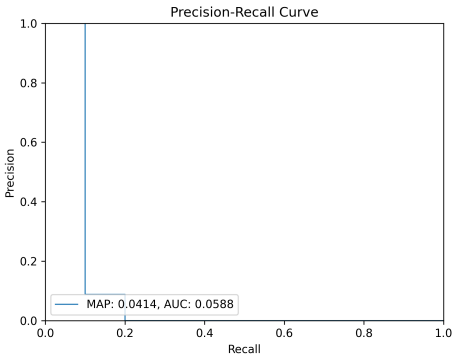

Figure 28: Q1 P-R Curve -> Semantic System

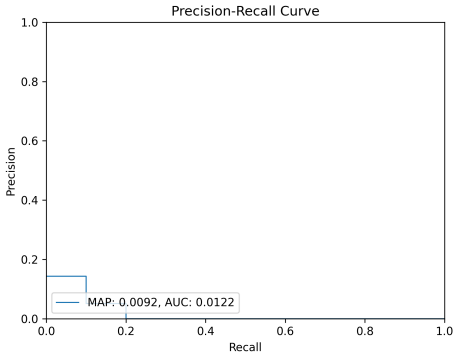

Figure 29: Q2 P-R Curve -> Semantic System

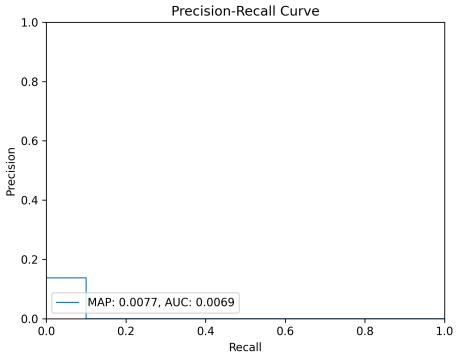

Figure 30: Q3 P-R Curve -> Semantic System

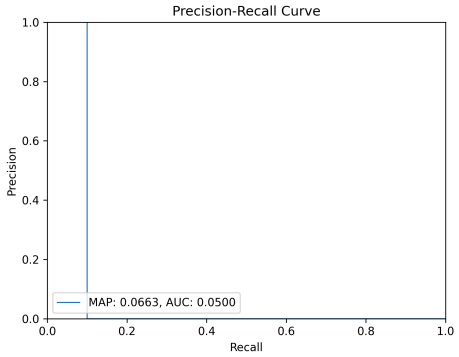

Figure 31: Q4 P-R Curve -> Semantic System