PRI: Building an Information Retrieval System using University Data

Ilina Kirovska
up202301450@fe.up.pt

Gonçalo Almeida
up202308629@fe.up.pt

Žan Žlender
up202302230@fe.up.pt

Abstract

The primary objective of the project is to develop an information retrieval system, prioritizing comprehensive exploration of European higher education. In *Milestone 1* a collection of documents was initially obtained from Kaggle which went through different data collection and data cleaning processes using scripts written in Python. It was enriched with data from the Wikipedia and Wikidata APIs for each respective university and city, resulting in a collection of 529 University documents. Furthermore, to better understand the quality and context of the data, a more in-depth analysis with visual representations was performed. The final result was a deeper understanding of the distribution of universities through geolocation, attribute comparison as well as text analysis, which gave the best results given the nature of the data is mostly plain text. *Milestone 2* focuses on using the aforementioned documents in the open-source search engine, Solr. From a set of 5 defined information needs Solr queries were created to extract those documents. Those same queries were evaluated, improved, and evaluated again to see the differences in results. In *Milestone 3* the improved solution from *Milestone 2* was chosen and further improved using different methods, such as MoreLikeThis, Semantic search, re-ranking, TF-IDF, and relevance feedback. After evaluating all methods it was concluded that the optimal method for the university search engine is a boosted query with semantic search re-ranking and relevance feedback, with an approximate 77% mean average precision (MaP) across all tested information needs. Using this method gave an improvement of 9% over the boosted query's 68% MaP. This information was used to construct a graphical user interface as well as the API backend which was used to query the Solr engine. In the end, we obtained an improvement of 14% for the MaP from the final solution of *Milestone 2* to the functional retrieval system that is the result of *Milestone 3*.

CCS Concepts: • **Information systems → Information retrieval**.

Keywords: data processing, datasets, data preparation, full-text search, conceptual data model, information retrieval system

1 Introduction

University rankings offer extensive assessment of higher education institutions. They are powerful tools, providing insightful analysis of the quality, performance, and general credibility of universities worldwide. Furthermore, they help students find educational institutions that align with their academic interests.

The Quacquarelli Symonds (QS) World University Rankings [3] debuted in 2004 and has since grown to become the foremost source of comparative data on university performance. As a result, we chose the 'QS World University Rankings 2024' Kaggle dataset [2] to be the base of our data. The dataset was then enriched using Wikipedia [5] and Wikidata [4] APIs, through which we extracted data about the universities and the cities they are located in.

The goal of the project is to build an information retrieval system by working with the extensive data made accessible through these APIs and the dataset.

This document explains the process of building our search engine, from the initial stages of data collection until the final result. Section 2 focuses on the collection of the data and its preparation. It explains how we used both the Kaggle dataset and the Wikidata API to construct a document collection where each document represents a university and its characteristics. Additionally, it contains the data characterization analysis that was done to better understand the domain and the gathered data. Section 3 describes how the Solr search platform was used for the initial implementation of the information retrieval system. It also demonstrates the evaluation process—the information needs used and its results.

2 Milestone 1

2.1 Data Collection and Preparation

The scope of this project includes all European universities. As such, the first step was to filter out the original University ranking dataset to get the European ones. For that purpose, we used the Python library Pandas to load and filter the universities by their country of origin ISO code, by comparing them to the European countries' .csv file[1]. After filtering, this dataset included 529 universities located throughout Europe, including ranking information on their campuses and faculties.

At this point, we encountered a problem, which was that some university names were written in their local language instead of English. Since the idea was to use the English Wikipedia API, which searches by the name of the university, the second step was to manually translate all the incorrectly written names into English. Using a simple Python script we identified the problematic names, then manually searched for them on Google. After finding the English version and the respective Wikipedia page, we then used those found names and wrote them into a new column in the dataset.

The following step included fetching the WikiData information for that university. With this information, we could identify any problematic entities which cannot be found, as well as get the relevant information about them. The most important is the WikiData identification number for that university. Only one university was not found because it had the symbol & in its name, which was quickly solved by exchanging the symbol with its UTF-8 representation %26.

After that, we could safely start getting the actual information we wanted, which was the plain text of the universities' Wikipedia pages. Using the Wikipedia API and the university names, as well as a Wikitext parser library, we quickly managed to get the text we wanted.

The next step was to include the cities' information about where the universities are located. Since the WikiData was fetched in the third step, we can easily query all of that university's information stored in WikiData.

Since all the data is structured, it includes many useful properties and relations, with the relevant ones being: located in the administrative territorial entity (P131) and location (P276).

Using those two properties, we could then fetch that city's WikiData information, which would return us the city's name, which could finally be used to get that city's Wikipedia page plain text as well.

Finally, during the analysis, we determined that there were a few columns which were irrelevant or had too many incorrect or missing values, so we removed them.

After the final conversion, the dataset included 529 universities with a file size of around 33MB, as a JSON file.

The process defined in this section is expressed as a flowchart in Figure 1.

The final dataset is a JSON file containing an array of universities, where each university object consists of the following attributes: 2024_rank, 2023_rank, institution_name, country_code, country, size, focus, age, status, academic_reputation_score, academic_reputation_rank, employer_reputation_score, employer_reputation_rank, faculty_student_score, faculty_student_rank, citations_per_faculty_score, citations_per_faculty_rank, international_students_score, international_students_rank, international_research_network_score, foundation_year international_research_network_rank, employment_outcomes_score, employment_outcomes_rank, overall_score, institution_name__wrong, wikidata, wikipedia_text, city_name, city_wikipedia_text, coordinates, university_vector, and url.

2.2 Conceptual Data Model

The conceptual data model, as seen in Figure 2, represents the relationship between each entity in the domain. In this case, the model is straightforward since most of the information is centered around the university. The university information includes its rank, as well as past year's ranking. Additionally, there are many metrics that determine the rank, like Employer Reputation Score, Employer Reputation Rank, Faculty Student Score, Faculty Student Rank, etc. One of the most important metrics is the **Overall SCORE**. Although we had extracted different information about the university, for example, the Wikipedia text, it's still connected to a specific university. Finally, each university is located in a city, which has its name as well as the City Wikipedia text extracted from its Wikipedia web page.

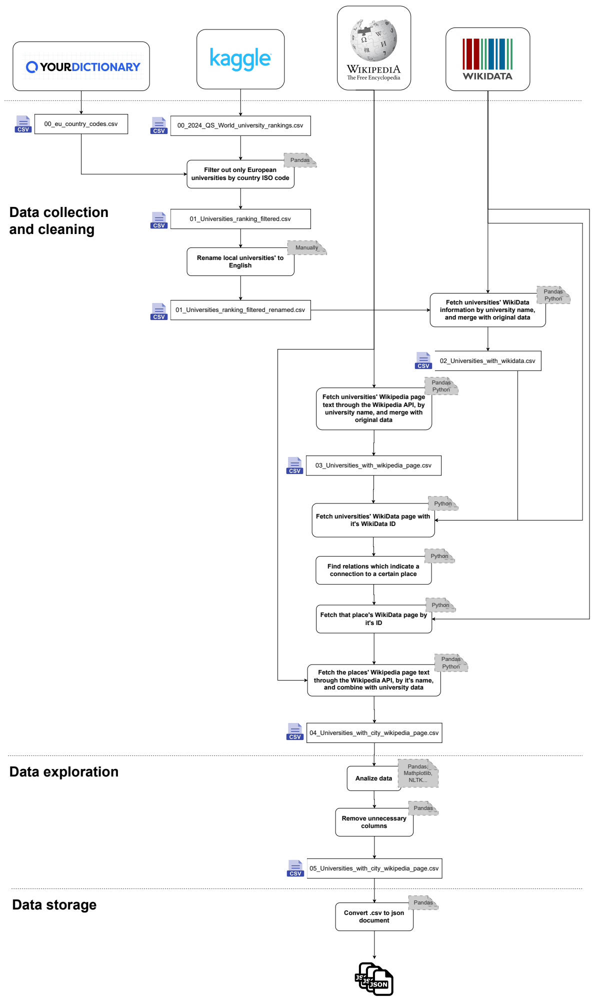
YOURDICTIONARY
kaggle
WIKIPEDIA The Free Encyclopedia
WIKIDATA
Data collection and cleaning
CSV
00_eu_country_codes.csv
00_2024_QS_World_university_rankings.csv
Pandas
Filter out only European universities by country ISO code
01_Universities_ranking_filtered.csv
Manually
Rename local universities' to English
01_Universities_ranking_filtered_renamed.csv
Pandas Python
Fetch universities' WikiData information by university name, and merge with original data
02_Universities_with_wikidata.csv
Fetch universities' Wikipedia page text through the Wikipedia API, by university name, and merge with original data
03_Universities_with_wikipedia_page.csv
Python
Fetch universities' WikiData page with it's WikiData ID
Find relations which indicate a connection to a certain place
Fetch that place's WikiData page by it's ID
Fetch the places' Wikipedia page text through the Wikipedia API, by it's name, and combine with university data
04_Universities_with_city_wikipedia_page.csv
Data exploration
Analize data
Pandas, Mathplotlib, NLTK...
Remove unnecessary columns
05_Universities_with_city_wikipedia_page.csv
Data storage
Convert .csv to json document
JSON

Figure 1. Data collection and preparation pipeline

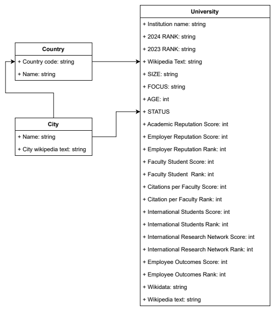
Country
+ Country code: string
+ Name: string
City
+ Name: string
+ City wikipedia text: string
University
+ Institution name: string
+ 2024 RANK: string
+ 2023 RANK: string
+ Wikipedia Text: string
+ SIZE: string
+ FOCUS: string
+ AGE: int
+ STATUS
+ Academic Reputation Score: int
+ Employer Reputation Score: int
+ Employer Reputation Rank: int
+ Faculty Student Score: int
+ Faculty Student Rank: int
+ Citations per Faculty Score: int
+ Citation per Faculty Rank: int
+ International Students Score: int
+ International Students Rank: int
+ International Research Network Score: int
+ International Research Network Rank: int
+ Employee Outcomes Score: int
+ Employee Outcomes Rank: int
+ Wikidata: string
+ Wikipedia text: string

Figure 2. Conceptual data model

2.3 Data Characterization

Understanding the collected data and identifying its key features is of great importance when building an information retrieval system. This was accomplished by doing additional data analysis, through which we gained more insights about the structure and context of the data. The results, along with proper visualisation, are explained in the following subsections.

2.3.1 Initial analysis of the acquired dataset. In the beginning stages of the process, a basic statistical analysis of the Kaggle dataset [2] was done. The aim was to identify how many of the universities fit our criteria of being a European university. As seen in Figure 3, 35.34% of all universities were in Europe and hence, kept in the dataset.

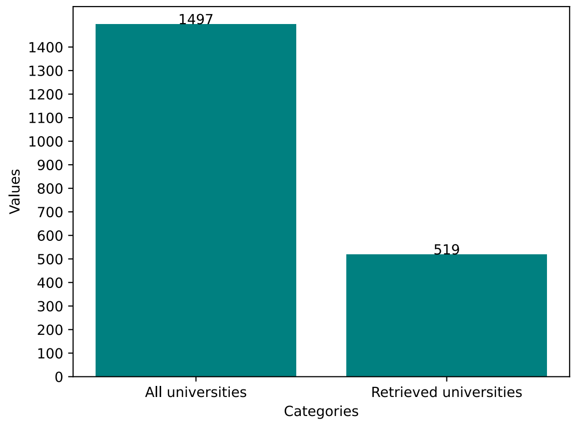

Figure 3. Number of all universities versus the number of the universities that were kept

2.3.2 Location analysis. As our focus is on European universities, their distribution over Europe is vital information. Therefore, it is illustrated using a variety of visual representations.
Figure 4 shows that the countries with the most universities are the United Kingdom (90) and Germany (49).

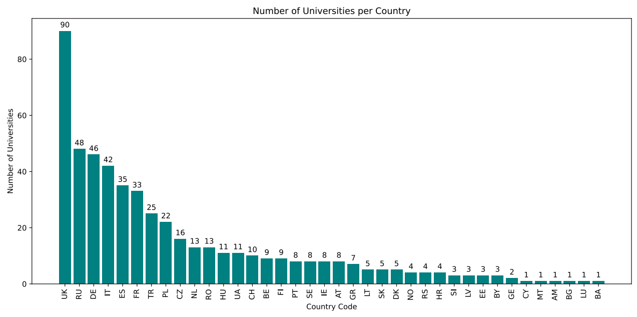

Figure 4. Number of universities grouped by country

Moving on to geographic visualization, Figure 5 is a map that displays the cities where universities are located. Each city is pinpointed on the map, providing a clear spatial perspective of the university distribution.

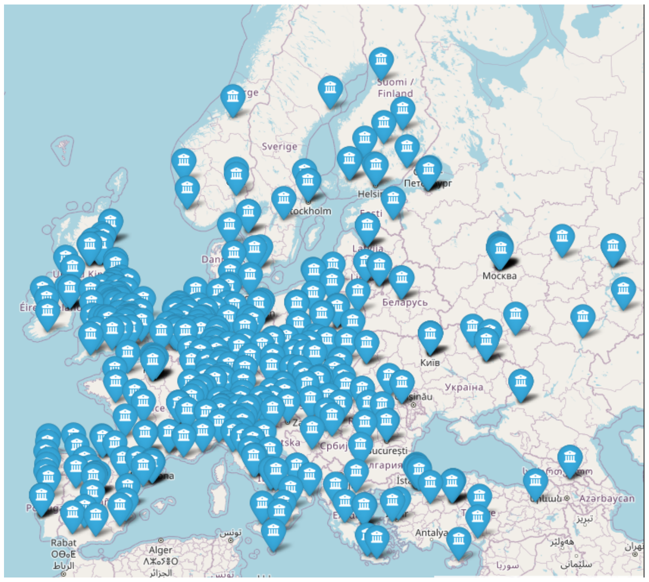

Figure 5. Map of the universities analysed

Figure 6 is a heat map that visualizes the distribution of universities across Europe, and it helps to identify clusters of universities in specific regions.

2.3.3 Ranking analysis. Figure 7 shows the changes in university rankings over time, by comparing the university ranks in 2024 (2023/2024) and 2023 (2022/2023). This dynamic visualization highlights universities that have experienced rank changes, such as improvements or declines, between the two years.

2.3.4 Size and Age Analysis. Figure 8 illustrates the number of universities falling into different size categories (XL, L, M, S) while considering their age, which ranges from new (Category 1) to older (Category 5). The teal-coloured bars represent the count of universities in each size category, with

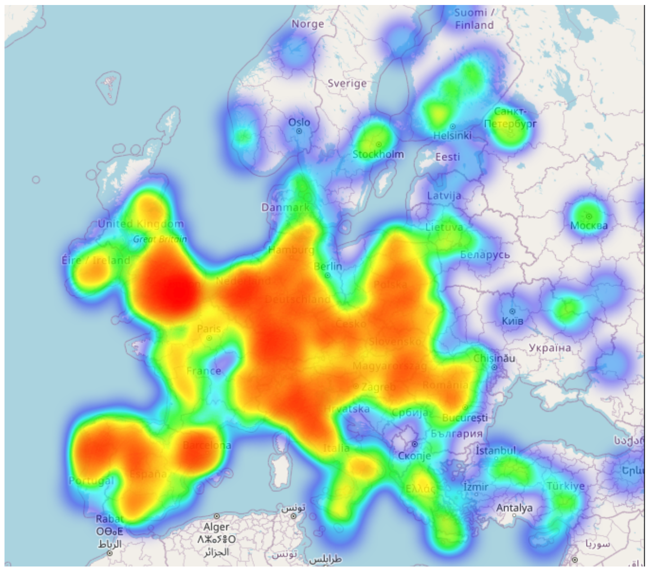

Figure 6. Heat map of the universities analysed

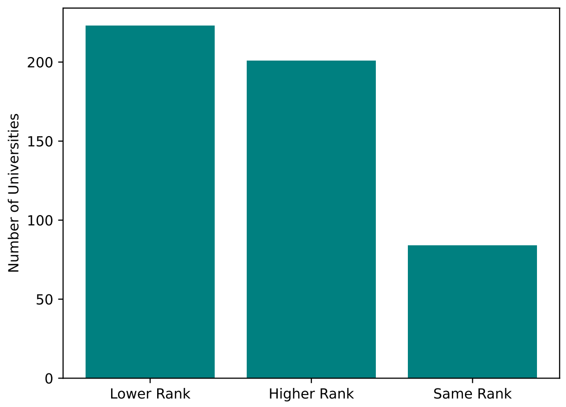

Figure 7. Distribution of universities grouped by rank

age categories distinguished by different shades within each bar. This visualization helps us understand how university sizes vary across different age categories.

2.3.5 Text analysis. The majority of the data in our system is stored as plain text, hence one of the most efficient ways to analyse it is to identify the most common words. Through them, we can easily find out the tone and content of the text.

We used the NLTK library [6] which offers NLP techniques. After the text was preprocessed using NLTK, we calculated the word frequencies and constructed the word clouds. As shown in the first photo in Figure 9 some of the most common unigrams in the universities' text data were university (appears 31.918 times), student (9.584 times), faculty (8.344

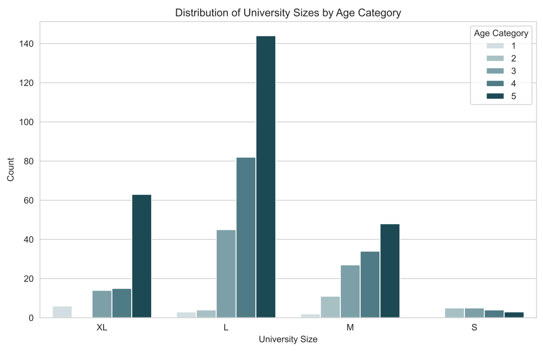

Figure 8. Distribution of university sizes by age category

times) and science (7.946 times). The other 3 photos show the most common bigrams, trigrams and fourgrams. When we calculated these wordclouds we specifically removed the n-grams which consisted of only stopwords, this mostly affected the bigrams.

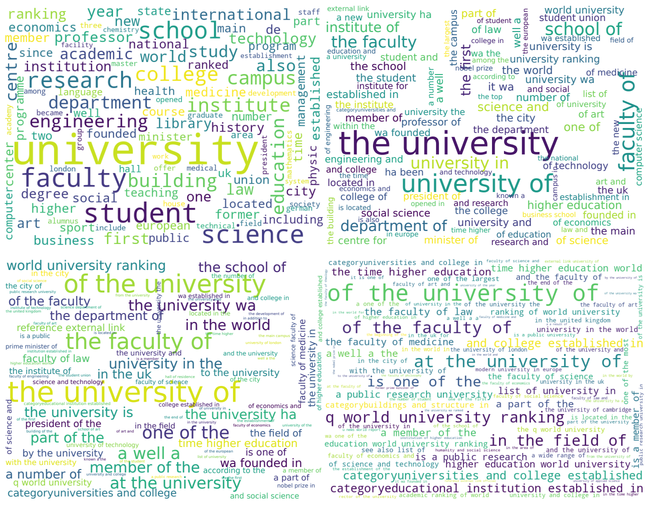

Figure 9. Wordclouds for the universities using unigrams, bigrams, trigrams and fourgrams respectively (starting from the top left corner).

The left photo from Figure 10 shows that in the cities' data, the most common unigrams were city (41.723 times), also (10.365 times), one (8.811 times) and century (8.583 times), and on the right we can see the most common bigrams.

2.4 Prospective search tasks

The focus of this search engine is to provide information about the available universities. Taking that into consideration, these are some of the possible search tasks:

1. Looking for universities that are top-ranked in computer science and are located in cities that have rich cultural heritage

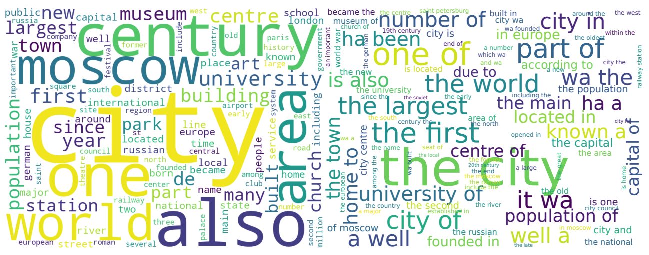

Figure 10. Wordclouds for the cities using unigrams and bigrams.

2. Looking for universities located in the United Kingdom that have courses in biology and are ranked in the top 150
3. Looking for universities in Germany that have a dental medicine faculty/dentistry and a large number of students
4. Looking for universities that have a faculty of engineering or a faculty of science and are located in a city with a Mediterranean climate
5. Looking for top-ranked universities in the north of Europe with a focus on the Computer Science field

3 Milestone 2

In the following sections, we will focus on the Information Retrieval part of the project. To be able to carry out this task we chose to use Solr - a powerful open-source search platform based on Apache Lucene. Solr allows document indexing and retrieval as well as efficient query processing and document ranking. Using Solr queries, for each of the information needs defined in Section 2.4 we retrieved the first 30 results and consequently evaluated our search engine.

3.1 Document indexing

As mentioned in Section 2.1, the result of Milestone 1 was a collection of 529 documents stored in one JSON file. To be able to retrieve the wanted information from these documents using Solr, we first need to index them using a schema. Schemas define various characteristics of the document fields, such as their type and how they should be processed during indexing and querying.

The indexing process was completely automatised by using a shell script. To upload the schemas and index the documents we used the Solr API and the commands shown on Listing 1.

Listing 1. Solr commands for uploading the schema and documents

```
curl -X POST -H 'Content-type:application/json' \
--data-binary "@new_schema.json" \
http://localhost:8983/solr/universities/schema

curl -X POST -H 'Content-type:application/json' \
--data-binary "@datasets/06_University_documents.json" \
http://localhost:8983/solr/universities/update?commit=true
```

3.1.1 Schema design.

In order to optimize and improve the query results for a more efficient search experience, we designed two schemas — one with a basic structure (Version 1 - V1) and another enhanced version incorporating various filters (Version 2 - V2). Both schemas have the same fields, as shown in Table 1, but with different field types. We indexed only the most important fields which help identifying the relevant documents for each query.

When we designed the two schemas the goal was to have one schema with little to no optimization and another one utilizing multiple Solr features. As seen in Table 1 for the first schema we used the same field type for all fields, except for the coordinates, where we used the LatLonPointSpatialField class. Moreover, no filters were used during indexing or querying apart from two very basic ones - ASCII Folding Filter and Lower Case Filter.

The second schema uses a lot more of the features Solr offers. In terms of field types, we used basic types such as int and float for fields containing standard data, whereas, for the fields 'wikipedia_text' and 'city_wikipedia_text', which contain the most complex and important data, we defined a custom type called wikipediaText. An important addition to this schema is the last field in the table - universityVector. The field is of type DenseVectorField and it was used to perform a semantic search.

The tokenizer used for this schema is the Classic Tokenizer. For both, indexing and querying, we applied several different filters in the following order:

- ASCII Folding Filter - converts alphabetic, numeric, and symbolic Unicode characters which are not in the Basic Latin Unicode block (the first 127 ASCII characters) to their ASCII equivalents, if one exists
- Lower Case Filter - converts all characters to lowercase
- Classic Filter - strips periods from acronyms and "'s" from possessives
- English Minimal Stem Filter - stems plural English words to their singular form
- Porter Stem Filter - applies the Porter Stemming Algorithm

By using these filters we make sure to have some generality and reduce the ambiguity while querying. After this pipeline and underlying all of the filter transformations tokens have a more general form which is a crucial part of query matching.

3.2 Retrieval

To be able to evaluate our search engine, we first need to query our information needs and retrieve the results. Solr offers numerous retrieval features that allow the user to tailor the search queries in order to get the best possible results. While querying, we used Solr's eDismax Query Parser. We decided to use this parser because it extends the abilities of

Field	Indexed	Field type	
		Version 1	Version 2
2024_rank	false	text	int
2023_rank	false	text	float
institution_name_-_wrong	false	text	text
institution_name	false	text	text
country_code	false	text	text
country	true	text	text
size	true	text	text
focus	false	text	text
age	false	text	text
status	false	text	text
academic_reputation_score	false	text	float
academic_reputation_rank	false	text	text
employer_reputation_score	false	text	float
employer_reputation_rank	false	text	text
faculty_student_score	false	text	float
faculty_student_rank	false	text	text
citations_per_faculty_score	false	text	float
citations_per_faculty_rank	false	text	text
international_students_score	false	text	float
international_students_rank	false	text	text
international_research _network_score	false	text	float
international_research _network_rank	false	text	text
employment_outcomes_score	false	text	float
employment_outcomes_rank	false	text	text
overall_score	false	text	float
foundation_date	false	text	tdate
wikidata	false	text	text
wikipedia_text	true	text	wikipediaText
city_wikipedia_text	true	text	wikipediaText
coordinates	false	coordinates	coordinates
universityVector	true	/	DenseVectorField
url	false	/	text

Table 1. Schema fields and their types.

the Dismax Query parser, allowing for more flexibility while querying.

Out of the various features this parser offers, we decided that the most appropriate one to use in our case is the field-boosting feature. Through this feature, we can assign different weights or importance to different fields of the schema. This allows us to influence the relevance of documents based on the fields where the query terms appear. By assigning higher boosts to a field, we make sure that if a term appears in it, it will result in that document having a higher relevance score. The boosts that we defined are:

- country^4
- wikipedia_text^3
- city_wikipedia_text^2

We chose these three fields to have boosts because we deem them the fields that contain the most valuable information for each university. The country of the university is a significant property, and as for wikipedia_text and city_wikipedia_text, they contain all additional information that was retrieved from the Wikidata API.

In the following section, we will adopt the use of abbreviations for the following eDisMax parameters:

- q (Query): specifies the user's query
- qf (Query Fields): specifies the fields to search in

3.2.1 Queries. Each information need from Section 2.4 was translated into one of the following queries that were then used for document retrieval:

1. **Information need:** Looking for universities that are top-ranked in computer science and are located in cities that have rich cultural heritage.
   q: top universities in computer science in city with rich cultural heritage
   qf: country^4, wikipedia_text^3, city_wikipedia_text^2, size
2. **Information need:** Looking for universities located in the United Kingdom that have courses in biology and are ranked in the top 150.
   q: top universites in united kingdom biology courses
   qf: country^4, wikipedia_text^3, city_wikipedia_text^2, size
3. **Information need:** Looking for universities in Germany that have a dental medicine faculty/dentistry and a large number of students.
   q: large universities in Germany dental medicine dentistry
   qf: country^4, wikipedia_text^3, city_wikipedia_text^2, size
4. **Information need:** Looking for universities that have a faculty of engineering or a faculty of science and are located in a city with a Mediterranean climate.
   q: faculty of engineering faculty of science in city with a mediterranean climate
   qf: country^4, wikipedia_text^3, city_wikipedia_text^2, size
5. **Information need:** Looking for top-ranked universities in the north of Europe with a focus on the Computer Science field.
   q: top universities in North of Europe computer science
   qf: country^4, wikipedia_text^3, city_wikipedia_text^2, size

3.3 Evaluation

3.3.1 Manual evaluation process. As we explained in Section 3.1.1 we have defined two different Solr schemas and therefore we have two different system configurations. When we evaluated the first, basic configuration, during the querying process we used the queries without any boosts. For the second, optimized configuration, we also used the field boosts. This process was executed for all 5 information needs.

For each query, we recovered the first 30 results, which were then deemed relevant or not relevant, based on whether they satisfied the corresponding information need requirements. The process of evaluating if a result was relevant was the following: we went through the data stored in each of the indexed fields and checked if they contained the needed information that would meet the user's needs. For example, in the first information need, the user is interested in a university that, among other things, is located in a city with a rich cultural heritage. Hence, we had to examine if a retrieved document contains mentions that might suggest that the city of the university has a long-standing history, cultural locations, or even just the word heritage.

3.3.2 Precision metrics.

After manually determining the relevance of each document, for each query, we calculated the following performance measures:

- Precision at n (P@n): Measures the proportion of retrieved documents in the top n results that are relevant to the information need.
- Average Precision (AvP): Calculates the average precision across all relevant documents, providing a single-value summary of the precision-recall curve.
- Recall at n (R@n): Evaluates the ability of the system to retrieve relevant documents by measuring the proportion of relevant documents among the top n results.
- Mean Average Precision (MAP): Calculates the average precision across multiple queries, providing an overall measure of the system's performance. It considers both precision and recall, making it a comprehensive metric for our information retrieval system.
- F-measure at n (F@n): Combines precision and recall into a single metric, balancing both aspects of the system's performance, being useful when there is a need to consider both false positives and false negatives in the top n results.

3.3.3 Results and discussion.

Tables 2 and 3 show the P@n, AvP, R@n, and F@n for each of the five queries, for the basic system configuration and optimized system configuration, respectively. We only display P@10, R@10, and F@10 because the first 10 search results are the ones that we deem to be the most relevant. For the calculation of AvP we used all 30 results.

If we look at Table 2 we see that the P@10 values for the non-boosted system configuration vary across queries. While some queries achieve high precision (e.g., 0.8 for Query 1), others exhibit lower precision with P@10 values of 0.1. A more consistent trend can be noticed in Table 3 where most of the queries achieve high precision (e.g. 1.0 for Query 2). The biggest difference here is that the system struggled with Query 4, resulting in a P@10 of 0.

It is worth mentioning that both configurations gave similar P@10 values for queries 1, 3, and 5, whereas for queries 2 and 4, we have opposite values.

Although by looking at Tables 2 and 3 an improvement from the non-boosted to the boosted system configuration cannot be clearly determined, if we look at Table 4 which shows the Mean average precision (MAP) for both configurations we can see that we obtained an improvement of 0.06 from the first to the second configuration.

Table 2. Information retrieval effectiveness measures for non-boosted queries.

Query	P@10	AvP	R@10	F@10
Query 1	0.8	0.8	0.36	0.5
Query 2	0.1	0.09	0.5	0.17
Query 3	0.1	0.26	0.33	0.15
Query 4	1.0	0.93	0.48	0.65
Query 5	0.6	0.57	0.43	0.5

Table 3. Information retrieval effectiveness measures for boosted queries.

Query	P@10	AvP	R@10	F@10
Query 1	0.7	0.8	0.33	0.45
Query 2	1.0	0.85	0.56	0.71
Query 3	0.4	0.49	0.36	0.38
Query 4	0.0	0.18	0.0	0.0
Query 5	0.7	0.6	0.47	0.56

Table 4. Mean Average Precision (MAP) for both search engines.

Metric	Version 1	Version 2
MAP	0.53	0.59

For a more visual perspective on the performance trends, we can use the precision and recall query metrics results and plot Precision-Recall curves (P-R curves). In Figure 11, we can see the interpolated P-R curves for V1 and V2. For each system configuration, we combined the queries into a single curve and then plotted them in a single chart. Both of the curves follow a similar downward trend, with V2 consistently having higher values. These two curves confirm our previous conclusion that we obtained an improvement from V1 to V2.

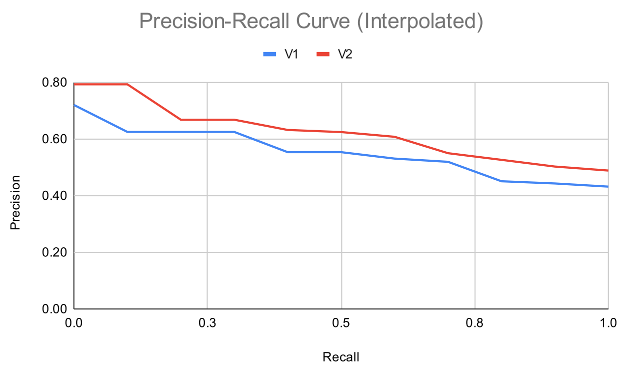

Figure 11. Interpolated P-R curves for V1(system configuration without boosts) and V2(system configuration with boosts).

4 Milestone 3

4.1 Improvements explored

Before defining the final approach for our search engine, we decided to explore a few possible improvements to the system we developed during Milestone 2. In order to choose the improvement that is going to yield the best results, we evaluated them using only two information needs. The P-R curves from this evaluation can be seen in Figure 13.

4.1.1 Semantic search. Semantic search is a search engine technology that understands the user's words based on their intent and the context of their query. The goal of this kind of search is to go beyond surface-level matching and provide better search results by more precisely and contextually understanding natural language. This means that the results of a semantic search are based on interpreting and matching the meaning of the query's words and phrases instead of exactly matching them with the documents' content.

This is achieved by using natural language processing and machine learning. Different models can be used to encode the query and the documents into their respective embeddings (their vector representations). Then they are matched based on the vectors' similarity in this new vector space in which we encoded them.

To be able to do this in Solr we used a sentence transformer to get the embeddings and then used the KNN-Query Parser to find the k-nearest documents to the query.

This feature has been introduced quite recently, so we wanted to explore its abilities. This is why we decided to evaluate the semantic search using two different input types:

1. information need
2. query

Because the key concept of semantic search is to use the word's meaning and context when document matching, our first approach was for the user's input to be the whole information needed. Then, for the second approach, for the user's input, we used the queries that were defined in Section 3.2.1.

Evaluation of semantic search. The results of the concluding evaluation are shown in two tables. Table 5 contains the semantic search evaluation results using information needs as input, whereas Table 6 contains the results using queries as input.

Table 5. Information retrieval effectiveness measures for Semantic search using information needs as input

Information need	P@10	AvP	R@10	F@10
Information need 1	0.6	0.66	0.35	0.44
Information need 2	0.6	0.67	0.3	0.4
Information need 3	0.4	0.39	0.33	0.36
Information need 4	0.6	0.67	0.29	0.39
Information need 5	0.7	0.73	0.41	0.52

Table 6. Information retrieval effectiveness measures for non-boosted queries.

Query	P@10	AvP	R@10	F@10
Query 1	0.1	0.27	0.09	0.09
Query 2	0.6	0.84	0.75	0.67
Query 3	0.4	0.58	0.67	0.5
Query 4	0.6	0.68	0.32	0.41
Query 5	0.6	0.62	0.46	0.52

Table 7. Mean Average Precision (MAP) for Semantic search using information needs (SS-IN) and using queries (SS-Q).

Metric	SS-IN	SS-Q
MAP	0.62	0.6

The interpolated P-R curves in Figure 12 show the trend of both system configurations. Both of the curves demonstrate a downward trend, indicating decreasing values. However, the downward trend for the SS-IN curve is less noticeable. Tables 5 and 6 show that both approaches produced either the same or very similar P@10 values for each information need, the only exception being information need 1 where the information need approach performed better, with a difference of 0.5. Additionally, Table 7 shows that the difference between the Mean average precision values of both configurations is only 0.02. Based on these findings, a clear winner between the two approaches cannot be determined.

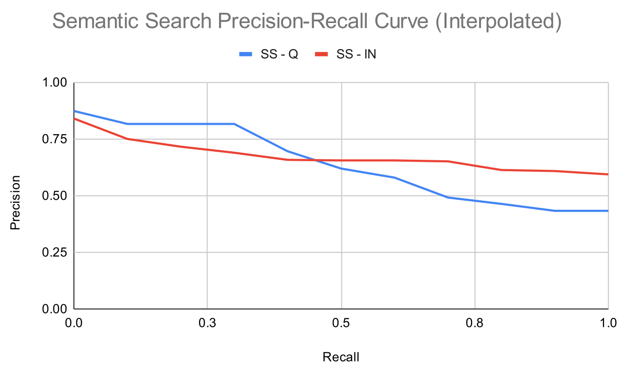

Figure 12. Interpolated P-R curves for Semantic search using information needs (SS-IN) and using queries (SS-Q).

4.1.2 Re-ranking using Semantic search. Semantic search in Solr can also be used to re-rank queries. As such, we applied this functionality on top of the boosted query to further improve it.

The re-ranking is executed by calculating a score for the document using the k-nearest neighbours algorithm, multiplying it by a multiplicative re-rank weight, and applying it on top of the initial query results. This results in a list of the same number of documents, but rearranged.

The evaluation of re-ranking using semantic search was compared to the other methods mentioned in the latter section, as they are all improvements on top of the initial, boosted system configuration (V2).

This method demonstrated superior efficacy and emerged as the most optimal solution, outperforming all other explored methods, as can be seen in Figure 13.

4.1.3 Relevance / Pseudo Relevance Feedback. Relevance Feedback actively involves users in refining search results based on their feedback, aiming to improve the result set, by considering the user's judgments on the initial set of results from its search. This approach acknowledges the difficulty of formulating a precise query without prior knowledge of the collection, allowing users to evaluate document relevance based on their information needs.

Pseudo Relevance Feedback shares the core concept of Relevance Feedback but streamlines the process by automatically analyzing results. Instead of relying on user classification, it assumes the top k, in our case the 3 first ranked results, as relevant and considers the rest as non-relevant, by default already non-relevant, providing a method for swift, automatic local analysis.

We use the Rocchio Algorithm in the Relevance Feedback to incorporate the user feedback (or the automatically generated one) into the vector space model. It seeks to optimize the query vector, maximizing the distinction between the average vector of relevant documents, and non-relevant documents.

After applying the Rocchio Algorithm, we give each term in the inverted matrix a weight based on its occurrence in the relevant or non-relevant documents. The greater the weight, the more relevant the term is presumed to be added to the query. Also, before adding them to the query we filter the terms to remove stop words and symbols. However, this ideal situation did not always hold, because most of our documents have the words "University", "Universities", and "city" in them and that influenced in a bad way the results.

Step-by-Step Example of The Relevance Feedback:

Original Query. The user initiates a search with the original query [universities in Portugal].

Result Retrieval. The system retrieves the top 10 results for the query.

Relevance Feedback Process. As part of Relevance Feedback, each result is presented to the user for classification:

- Doc 1: Relevant
- Doc 2: Relevant
- Doc 3: Non-Relevant
- ...
- Doc 4: Non-Relevant

Inverted Matrix Construction. The system builds an inverted matrix, featuring terms associated with document IDs and positions of occurrence.

{Rocchio Algorithm Application The inverted matrix and the list of user-classified results are fed into the Rocchio Algorithm. It computes the Query Vector, associating each term with a weight:

$$\begin{aligned}
[\text{university}] &\longrightarrow [0.9] \\
[\text{europe}] &\longrightarrow [0.75] \\
&\cdots \\
[\text{capital}] &\longrightarrow [0.46] \\
&\cdots \\
[\text{west}] &\longrightarrow [0.31]
\end{aligned}$$

Modified Query. The system selects the top three highest-weighted terms to enhance the original query: [university in portugal europe capital west] is the modified query

4.1.4 MoreLikeThis. MoreLikeThis (MLT) is a method of enhancing queries in Solr that enables retrieval of documents similar to the referenced documents. As this is a built-in function of Solr, and results in similar to relevance algorithms we decided to experiment with it. As with the re-ranking using semantic search, this method was applied to the boosted system configuration to further enhance the relevance of the results.

Term frequencies are computed by re-tokenizing the text in the desired fields. In other words, the frequencies are calculated by the number of times a specific term appears in the text. Terms can be excluded by different parameters, by setting a threshold on the maximum and minimum number of characters allowed or the frequency itself.

However, the effectiveness of this method was found to be notably suboptimal. As can be seen in Figure 13, the results performed worse than any other of the explored methods.

4.1.5 TF-IDF ranking.

TF-IDF, or Term Frequency-Inverse Document Frequency, is a statistical measure used in natural language processing and information retrieval to evaluate the importance of a term within a collection of documents. In terms of search engines, it helps in determining whether a document is relevant for a given query. The value is comprised of two frequencies:

- TF or term frequency, measures how often a term appears in a document.
- IDF or inverse document frequency, measures how common/rare is a term within a collection of documents

When we use TF-IDF, common terms such as "the", "a" and "an" have a low TF-IDF score, because they don't tell us much information about the document. On the other hand, words that appear frequently in a particular document but are rare throughout the collection of documents have a high TF-IDF score. These terms help in determining the document's relevance. If a term with a high TF-IDF matches a term from our query, then this occurrence increases the ranking score of the document.

In practice, this method didn't turn out to be efficient for us, as seen in Figure 13 it was the second worst improvement we tried.

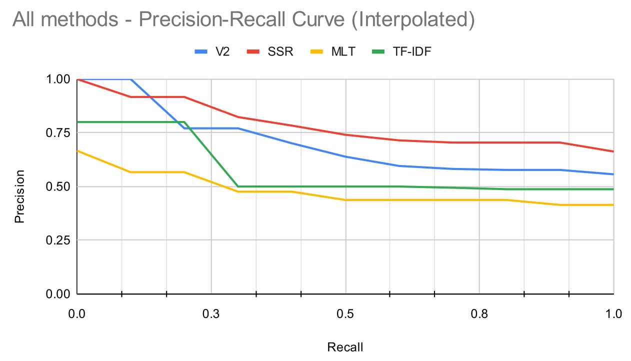

Figure 13. Interpolated P-R curves for the boosted system configuration from M2 (V2), Reranking with Semantic search, MoreLikeThis and TF-IDF.

4.2 Final approach

After trying multiple possible improvements, we decided on our final approach: re-ranking using semantic search with a relevance feedback algorithm. This means that we use our boosted system configuration from Milestone 2 as the base of the system. Then we use semantic search, more precisely the KNN Query Parser, to rerank the results from the first Solr search. Furthermore, the pseudo-relevance feedback algorithm changes the input query by adding the most frequent terms and does a new Solr search. The ending results are the results of this search. Additionally, the user can manually choose the relevant document and use the relevance feedback algorithm to obtain new results.

It is worth mentioning that for the user interface of the search engine we used Solr's highlight option to get highlighted parts of the text within the selected fields which Solr recognizes as matching for the given query. The plain text is wrapped in the HTML <em> tag, which makes styling it easier.

4.2.1 Evaluation and discussion.

We evaluated our final approach using all five information needs defined in Section 2.4. The evaluation reflects the results obtained when the pseudo-relevance feedback algorithm was used.

Table 8 shows that this approach is the most consistent out of all the solutions presented in this project. All of the P@10 values are higher than 0.5 and do not differ a lot from each other. This is very different from the P@10 values for our starting point, the system from Milestone 2. As seen in Table 3 these values varied, from 0.0 for Query 4 to 1.0 for Query 2.

By comparing the Mean average precision values in Tables 4 and 9 which are equal to 0.59 for the boosted system configuration and 0.73 for our final approach, we see that we obtained an improvement of 0.14 from one system to another. This further proves that we chose the right improvements.

Table 8. Information retrieval effectiveness measures for our final approach.

Information need	P@10	AvP	R@10	F@10
Information need 1	0.7	0.75	0.3	0.42
Information need 2	0.9	0.89	0.43	0.58
Information need 3	0.7	0.64	0.41	0.52
Information need 4	0.9	0.82	0.47	0.62
Information need 5	0.5	0.56	0.31	0.38

Table 9. Mean Average Precision (MAP) for our final approach.

Metric	Final solution
MAP	0.73

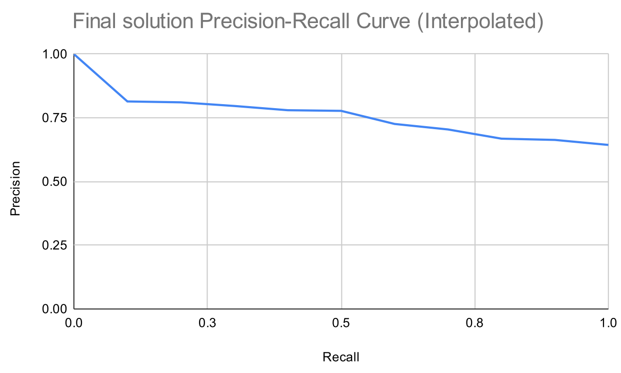

Figure 14. Interpolated P-R curve for our final approach.

4.3 Application

The application consists of 3 main parts: the Solr engine which contains all the documents, the NextJS application which serves both as a graphical user interface and a web server, and finally a Flask API server written in Python that provides a way to execute queries to the Solr engine API from the graphical user interface.

The API server accepts multiple parameters on the search endpoint which are used to construct and execute the Solr query. Firstly, it accepts the user's search input, then the limit and offset for document pagination, and finally an optional query vector. If the query vector is not present it uses the sentence_transformers Python library to create a vectorized representation of the search input, however, if the query vector is provided then it uses it. This vector is used as part of the semantic search re-ranking as mentioned in 4.1.2.

The graphical user interface provides the user with an input field, filters for the age and size of the university, the possibility of marking documents as relevant and viewing the results. If the user marks any of the results as relevant, the Rocchio algorithm is used as explained in 4.1.3 to calculate a new vector that will be used in the Solr query. After the API call finishes it will display the augmented results.

4.4 Conclusion

This report details the three milestones of the project, whose goal is to build an information retrieval system.

In the first milestone, we focused primarily on dataset selection, preparation, and exploration. The dataset that was used includes all of the basic information about European universities, including rankings, locations, and institutional characteristics. Wikidata and Wikipedia API were used to streamline the data-collecting process and ensure data relevance and accuracy for the study. At the end of the milestone, we had a document collection with all the necessary university information.

In the second milestone, we successfully implemented Solr, indexed the document collection, and retrieved and evaluated the query results based on our information needs. Our efforts were concentrated on refining document indexing, improving the basic schema, crafting effective retrieval queries, and evaluating the results manually to assess Solr's performance. Furthermore, for each query, we made a boosted version where we got significantly better results, as expected.

In the third and final milestone, we further improved the boosted system configuration from milestone 2 using different methods described in 4.1. We concluded that the most performant method and our final approach was applying semantic search re-ranking on top of the boosted query. To further better the performance of our retrieval system, we used the (pseudo) relevance feedback algorithm. The final evaluation showed that by doing this we obtained an improvement of 0.14, from the boosted system configuration with a $MaP = 0.59$ to our final approach with a $MaP = 0.73$. The outcome is a functional University information retrieval system accompanied by a user interface.

References

[1] [n. d.]. List of European countries. Retrieved October 10, 2023 from https://www.yourdictionary.com/articles/europe-country-codes

[2] [n. d.]. QS World University Rankings 2024 Dataset. Retrieved September 28, 2023 from https://www.kaggle.com/datasets/joebeachcapital/qs-world-university-rankings-2024/data

[3] [n. d.]. Quacquarelli Symonds (QS) World University Rankings. Retrieved October 10, 2023 from https://www.qs.com/

[4] [n. d.]. Wikidata API. Retrieved October 10, 2023 from https://www.wikidata.org/wiki/Wikidata:REST_API

[5] [n. d.]. Wikipedia API. Retrieved October 10, 2023 from https://www.mediawiki.org/wiki/API:Main_page#Endpoint

[6] Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit.*