Information Search System for Versioned Portuguese News Articles about Technology

João Romão
Viseu, Portugal
up201806779@edu.fe.up.pt

Rafael Cristino
Cucujães, Portugal
up201806680@edu.fe.up.pt

Xavier Pisco
Moita do Boi, Portugal
up201806134@edu.fe.up.pt

ABSTRACT

In this document, we present our information search system for versioned Portuguese news articles about technology. Our data is obtained from the Portuguese web archive, more specifically from technology news articles stored by the Arquivo.pt service. We discuss the steps we took and the processes we used to arrive at the final result.

We started with the collection of a dataset by using the publicly available Arquivo.pt APIs to retrieve data and employing data pre-processing, cleaning and refinement techniques.

We then defined our document schema, indexed the dataset into Solr, and showcased our system's data retrieval capabilities by defining possible queries.

To evaluate our query results, we applied evaluation methods such as Precision-Recall Curve and Mean Average Precision to different states of the system and different query configurations.

Lastly, we experimented with new functionalities with the objective of improving the results given by the system.

KEYWORDS

websites, news, articles, technology, information, search

1 INTRODUCTION

Arquivo.pt [1] is a service that allows anyone to visit a past version of any webpage or to visit web pages that are no longer active. This service was created by *Fundação para a Ciência e Tecnologia* [2] and it does so by crawling the Portuguese web and storing information about all of the webpages that it finds. This has been going on since 1996, therefore it contains a huge amount of data that can be fetched and used. All of this data is publicly available by the means of multiple API's that are described in the Arquivo.pt repository [3].

For this project, we are going to develop an information search system centred around a dataset that is gathered from Arquivo.pt. For collecting this dataset, because we are working in an academic setting and the service's data is very extensive, we decided to use a smaller portion of what's available by defining a small scope for the information that will be retrieved. Of all the available website's archives, our work will focus on Portuguese technological news articles indexed by Arquivo.pt in 2021, from the 1st of January to the 1st of November, and that were published by 3 Portuguese news media companies: *Notícias ao Minuto* [4], *Jornal de Negócios* [5] and *Exame Informática* [6]. The data collection resulted in a 183.5MB CSV dataset, with 19580 unique articles, and in total, counting with all of the different versions of each article, with 83838 entries.

2 DATA PROCESSING AND RETRIEVAL PIPELINE

In this section, we describe the steps we took to process and retrieve the data from Arquivo.pt. This is a very important process because the final result will be used as the dataset for our information search system.

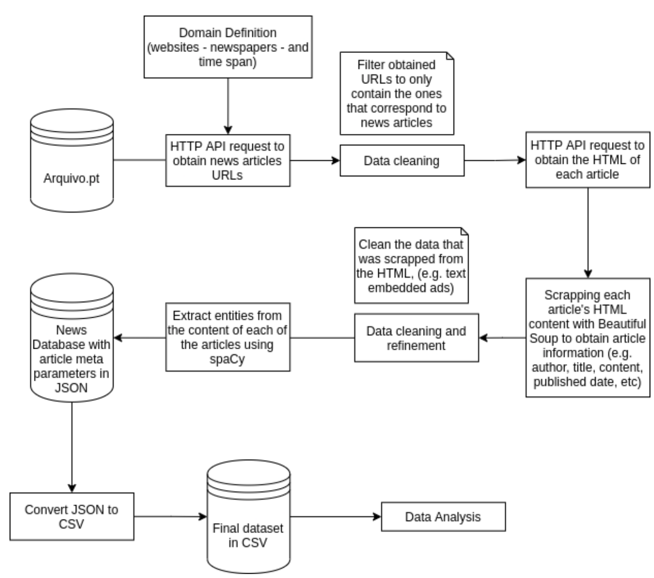

Figure 1: Pipeline diagram
Describes the data collection process.

2.1 Domain Definition

To get the data we need from Arquivo.pt, first, we need to do API requests to get the websites Arquivo.pt has stored and select only the ones we want.

We used the CDX server API [7], which returns an entry for each of the indexations that exist for the query we give it. We chose to only get tech news from the three different websites already mentioned, *Noticías ao Minuto*, *Jornal de Negócios* e *Exame Informática* and restricted them to the ones that were indexed in 2021, from the 1st of January to the 1st of November. This part of the pipeline was executed in a python script named "pull-links.py".

2.2 Filter Domain Data

After this initial gathering of data, some of it was not useful for our project. Some of the links we had were from websites that did not contain news but had a similar URL to the ones that did. We decided to remove those links before downloading the websites because that would reduce the time needed to process the next step of our pipeline. For this, we created and used a python program that we called "remove-non-news.py".

2.3 Getting the HTML

After having the links to the websites that we want to get our data from, we need to download the HTML from those websites and store it together with all the other available information about that website.

In this part, we used the NoFrame API [8] and saved the HTML from all the websites we had previously filtered, to be used to get the information needed for our project. Our script "pull-html.py" is responsible for this.

2.4 Scrapping of HTML

The HTML we got was too big and had too much code that we just didn't need for our final goal. We decided that we needed to scrap that HTML and store only the important parts.

Thus, we created a python program called "parse-information.py" where we read the HTML using the Beautiful Soup [9] library and we retrieved the parts that were important to us, for example, the title, author, text, etc.

2.5 Data Cleaning

Since some of the retrieved information that we got on the previous step had some irrelevant text and some inconsistencies, we decided to clean those parts of the data. As an example, some of the news had ads in the middle of the text and sometimes the newlines were represented by more than one \n.

Most of this was made at the same time as the scrapping to improve the time efficiency and avoid an unnecessary extra loop on the data, except for the embedded ads, that were removed in the "clean-ads.py" script.

2.6 Extracting Entities

One thing we thought was interesting to have in our project was a list of all the entities referred to in each of the articles we had. This may allow us, in the future, to get a better grasp of each article's content and possibly to define relations between multiple articles. So, for retrieving the entities from the articles' contents, we decided to use the python library spaCy [10] in our "parse-entities.py" program to do that.

2.7 Convert JSON to CSV

To easily analyze and characterize the data we thought we should change its format, thus, we decided we converted it to CSV format. As a result, we created 5 separate CSV files whose contents are better explained in the Data Storage section. In the makefile, this is represented by the "json_to_csv.py" program.

3 CONCEPTUAL DATA MODEL AND DATA STORAGE

To more easily manipulate the data we decided to change its format from JSON to CSV. For that, basing ourselves on the conceptual data model (figure 3), we split it into 5 different CSV files:

- "domain.csv": maps to the "Website" in the conceptual data model.
- "urlkeys.csv": maps to the "News" in the conceptual data model.
- "news.csv": maps to the "Version" in the conceptual data model.
- "entities.csv": maps to the "Entity" in the conceptual data model.
- "news_entities.csv": maps to the many-to-many relation between "Version" and "Entity".

For simplicity's sake, we decided to keep the author's name in the "news.csv" file, even though that slightly increases the redundancy in our dataset.

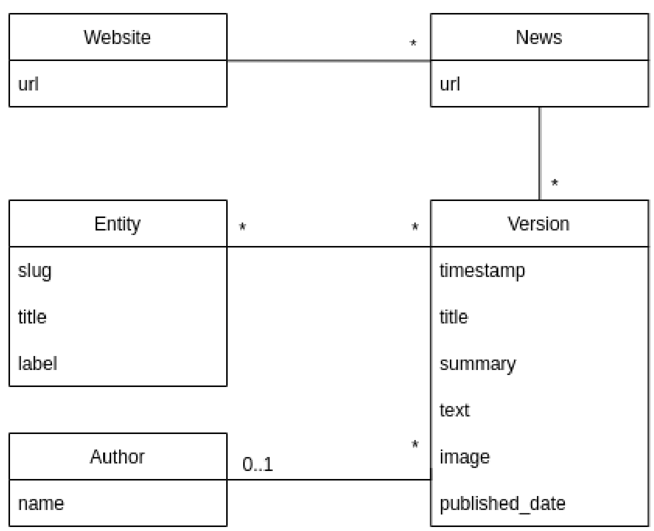

Figure 2: Conceptual Data Model

Describes the relations between our data.

On the website table, we have the URL corresponding to the news companies that we defined in our problem domain, being identified by "noticiasaominuto", "jornaldenegocios" and "exameinformatica".

On the news table, we have the urlkey for each of the news, this key is a specific name given by Arquivo.pt that is unique to each of the websites it has stored and based on the URL that was scraped. This table is connected with the website table since each of the urlkeys belongs to one of the domains.

Each of the news has one or more versions that correspond to each of the indexations made by Arquivo.pt for that particular article. Those versions have a timestamp, that stores the date of indexation, and the article's metadata and contents, as described in the diagram.

For each version of the news article, there is a specific author, and each author can correspond to more than one article version.

At last, the versions have entities in them that are stored in a separate table, the entities table, and have a title, a slug, which is an identifier based on its title, and a label. The label can be one of PER (person), ORG (organization), LOC (localization), and MISC (miscellaneous).

4 DATA CHARACTERIZATION

In this section, we seek to characterize the data we collected to achieve a better understanding of our domain.

4.1 Amount of news

To better understand the amount of data we are dealing with and to know from which domain the majority of news come from, a plot mapping each domain to the number of distinct news indexed in 2021 was built.

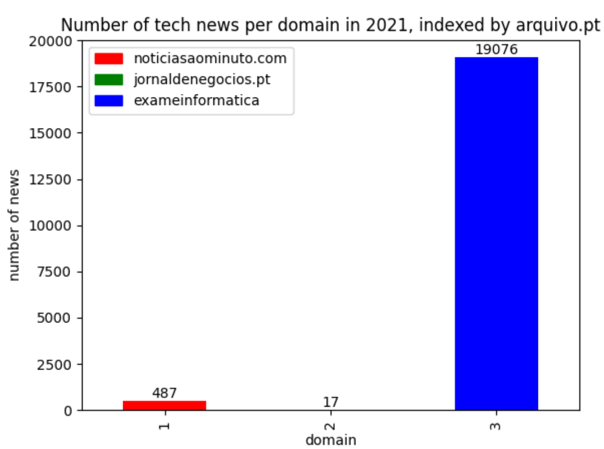

Figure 3: Amount of unique articles indexed per domain

Since each article can be indexed at Arquivo.pt more than once, the total amount of indexed articles was also plotted to each domain.

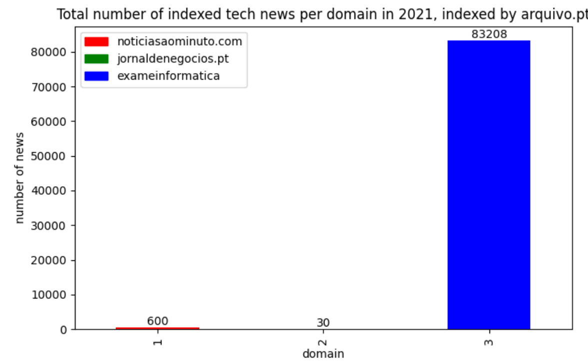

Figure 4: Total number of indexed articles, including multiple versions, per domain

4.2 Indexation count

As it was shown in the previous subsection, each article can be indexed more than once. This way, it would be interesting to know how many times each article has been indexed at the Arquivo.pt database. To visualize this aspect, a plot mapping the number of entries to the number of articles grouped by the number of times they were indexed was built (for example, in figure 6, 1665 articles were indexed only once, while 9996 articles were indexed twice).

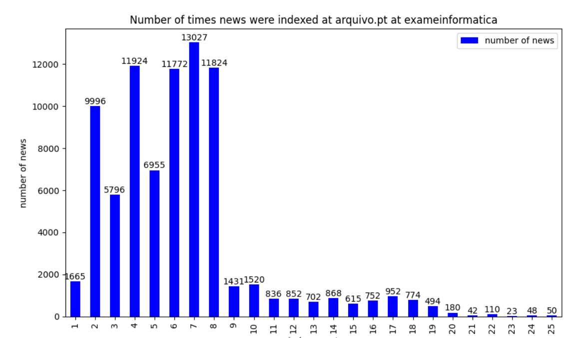

Figure 5: Count of the number of news grouped by the number of times they were indexed

4.3 Average article length

Another characterization we found interesting was to compare the length of articles from different sources. To visualize this aspect we plotted each domain to the average length of its articles taking a character as a measurement unit.

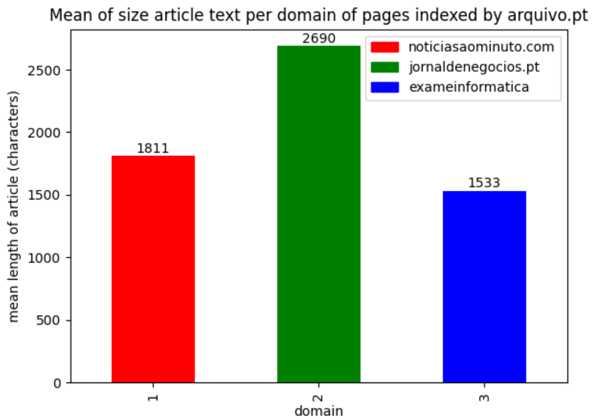

Figure 6: Average article length (characters)

4.4 Group news by publication year

Even though only news indexed at Arquivo.pt in 2021 were taken into consideration, the article's published date may differ a lot from the date it was indexed. To understand how the articles spread over time, the articles were grouped by the corresponding publication year.

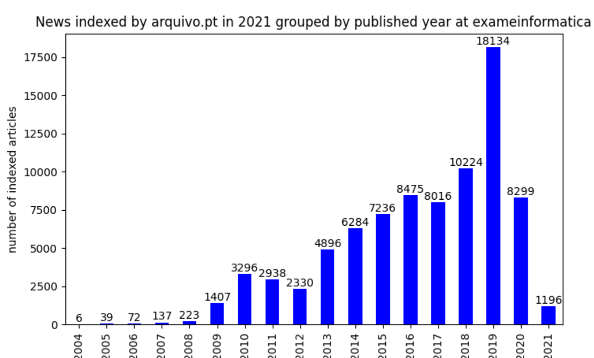

Figure 7: Count of news grouped by publishing year

4.5 Entities count

Because we have parsed all of the entities in the articles' contents, we can now get a better grasp of the topics that are most mentioned in the articles we collected. We did so by calculating the entities that are mentioned in most articles (for example, in figure 9, "Google" was mentioned in 10756 indexed articles).

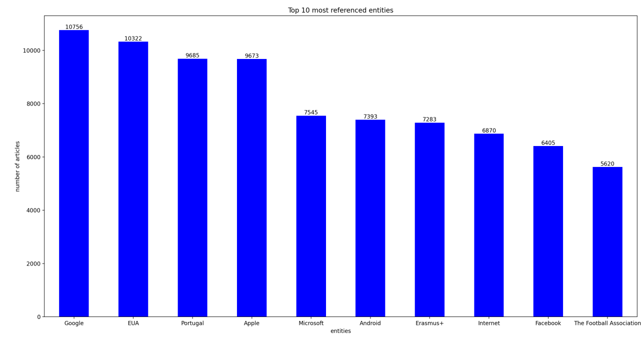

Figure 8: Top 10 referenced entities

5 COLLECTION

For the implementation of the information retrieval system, we decided to use the Solr [11] tool and, thus, we needed to split our information into multiple documents to be useful in their indexation and query system.

We already have our data split into different newspapers, inside of which it is split into unique articles and, inside each article, into different versions. However, we needed to morph this into a collection of documents, so that we have an entry for each unique article version.

Furthermore, we needed to format the dates into a specific format used by Solr.

5.1 Documents

When setting up a Solr instance, we upload a JSON file containing a list of documents, each one of those documents corresponding to a version of an article that we previously retrieved.

Each document contains the following information: *urlkey*, *timestamp*, *URL*, *article*, *newspaper*.

5.1.1 urlkey. The urlkey is a string created by Arquivo.pt that works as an identifier for every page in the database and that's equal for the same webpage across different versions. It is based on the URL and has a constant format across all articles.

5.1.2 timestamp. The timestamp of an article version is the date on which the page was indexed by Arquivo.pt containing the year, month, day, hour, minute and second information. Since all the dates are in Lisbon time we decided to ignore the timezone that was needed by the Solr date format.

5.1.3 URL. It's a string with the URL from which the article was fetched.

5.1.4 article. The article contains all the important information about the article that the document represents. It has the title from the article, a small summary of its contents, a link to the article's cover image, the date in which the article was published (in the same format as the timestamp), the name of the author (or, in the case where it was missing, the name of the newspaper), the text representing the article content and a list with all the entities that were found on the article.

Since the information about the article is as a nested document inside the main documents, Solr will "flatten" the information to a single document and create entries in the format "article.title". Due to this fact, we will refer to each part of the article in that same format throughout the remainder of this document.

5.1.5 newspaper. A string containing the name of the newspaper in all minuscules and without any non-ASCII characters or spaces.

6 INDEXATION

In his section, we discuss how we used our collection schema to guide Solr while indexing our documents, which should give us better results in the information retrieval tasks.

For quicker results in our information retrieval system, we decided to index most of the information we will be searching for in our future queries. To improve our results, we applied tokenizers and used stemming filters in some of our documents' textual fields.

6.1 Date fields

The dates that are used in our documents should be treated as dates by Solr, so we set their field type class to *solr.DatePointField*. This makes it easier to search for specific dates and for dates between a range, which will be used in our queries later on in the document.

The article publication date will be useful, for example, for the user to search for articles published between two different dates, and since we want that to be fast it should be indexed.

The timestamp of the webpage version will be used in most queries to only show the most recent version of an article to the final user and, in specific cases, order the versions based on that time, so this date will also be indexed.

6.2 Text fields

When dealing with text we separated the text fields into two different groups: text fields that will be used as a whole, (text fields that will be used as they are for exact matching), and text fields that need to be tokenized and stemmed (the fields that will be split into parts that can and will be used both in queries and indexes).

In the first type of text field, we have both the urlkey and the newspaper, both have the *string* type from Solr. The urlkey will be used as a way of comparing if two different documents belong to the same webpage. It is the identifier for each unique article so there will be an index based on it. In this category we also have the name of the newspaper that the news belongs to, in this case, since there are only three different newspapers, we decided to not index this field.

For the fields that we wanted to do full-text search on, we decided to tokenize them with the *solr.StandardTokenizerFactory* tokenizer, we changed the non-ASCII characters to ASCII with *solr.ASCIIFoldingFilterFactory*, we saved everything in lower case, using *solr.LowerCaseFilterFactory*, to have the queries give the same results both in lower or upper case. To improve our documents and queries even further we wanted to lemmatize, but we couldn't find a way to do it in Solr with Portuguese data so we decided to stem the fields with *solr.PortugueseStemFilterFactory*.

In this category, we have the following fields: article.title, article.summary, article.text, article.entities.title, article.authors. All of these fields are indexed as they will all be used in the multiple queries that will be presented below.

Field Name	Type	Indexed
url	tokenizedText	X
urlkey	string	X
timestamp	date	X
newspaper	string	
article.title	tokenizedText	X
article.summary	tokenizedText	X
article.text	tokenizedText	X
article.entities.title	tokenizedText	X
article.entities.label	string	
article.authors	tokenizedText	X
article.publish_date	date	X

Table 1: Summarization of the dataset schema

7 INFORMATION RETRIEVAL

In this section, we present and explain how we implemented different queries. We have them ordered by increasing implementation complexity to have greater diversity.

7.1 Search for different versions of an article based on a URL

In our information retrieval system, we have the option for users to find different versions of the same news and one of the ways to do that is to search the full URL of a website and that will present all the versions of that website.

In Solr, we just have to search with the following query: *url:"https://www.noticiasaominuto.com/tech/1125833/sonda-da-nasa-ja-aterrou-em-marte"*. The Solr response will be a JSON containing all the available versions of the articles with that URL, in this case, two documents.

7.2 Date filtered search

Since each stored article has a field relative to its published date it is possible to select articles that were published in a specific time interval. As it was previously mentioned date fields are being stored in *solr* as *solr.DatePointField* types, so to build the query the date fields have to follow the format specified by solr:

YYYY-MM-DDThh:mm:ssZ, where YYYY is the year, MM is the month, DD is the day of the month, hh is the hour of the day as on a 24-hour clock, mm is minutes, ss is seconds and Z is a literal 'Z' character indicating that this string representation of the date is in UTC.

Taking into consideration that multiple versions of the same document are stored, we decided to show only one version of an article in the results of this query, so we can receive better results. To do this, results were grouped by URL, which is unique for each article. In solr this was achieved with the following instruction: *fq: {!collapse field="urlkey" sort='timestamp desc'}*.

Bearing in mind both these aspects the following query was built:

- q: article.publish_date:
  [2021-05-20T00:00:00Z TO 2021-08-15T00:00:00Z]
- fq: {!collapse field="urlkey" sort='timestamp desc'}

In this case, the articles published from 20-05-2021 to 15-08-2021 are displayed.

7.3 Search for the number of times a page was indexed in an interval of time

One of the motivations for basing our search system on the Arquivo.pt API was the possibility to verify the frequency of indexation for each article and to find if any patterns can be observed in the way articles are stored and updated.

Doing so, it is relevant to include a search query that illustrates this aspect. The following query was built to provide answer to this question:

- q: url:"https://visao.sapo.pt/exameinformatica/noticias-ei/2010-04-26-microsoft-touch-pack-para-windows-7-ja-disponivel/" AND timestamp:[2021-03-01T00:00:00Z TO 2021-03-31T23:59:59Z]
- Raw Parameter Queries: group=true&group.field=urlkey&group.sort=timestamp desc

The *Raw Parameter Queries* was selected to group the different versions of the same article since the content of the articles is not relevant to the query result. The *collapse* utility that was referenced previously was not used since it decreases the number of results found (the parameter of main interest) to one. The selected method groups the results and keeps track of their count in the field named *numFound* in the *response* object.

7.4 Text search

Text search is included in the vast majority of search systems. The text search query implemented allows the search of a specified string. Since the selected document has multiple text fields and it may be of interest to search for a sequence of characters that may be present in any of them, the built query has these possible fields into consideration. To improve results, we also added weights to different fields, favouring the article's title and entities because normally the most relevant results would include some query words in them. Bearing this in mind, the following query was built:

- q: aterragem em Marte
- defType: edismax
- qf: article.title^3 article.entities.title^3 article.text article.summary article.publish_date

The query has the intention of simulating the reception of input provided by the user, this way the same string is passed to each query field, in this case, the selected string was *aterragem em Marte*.

7.5 Combination of multiple parameters

To simulate a possible use case of the search system a query was built that would take multiple parameters with different values. The proposed query could be phrased the following way: "Search for an article that has *Sistema Operativo Android* in its text, was authored by someone called *Pedro* and published in the *Exame Informática* newspaper before 20-05-2011, and has the entity titled *Google* associated with it".

- q: article.entities.title:"Google" AND newspaper:exameinformatica AND article.publish_date:[* TO 2011-05-20T00:00:00Z] AND article.authors:Pedro AND article.text:Sistema Operativo Android
- fq: !collapse field="urlkey" sort='timestamp desc'

Once again the *collapse* is used so that an article is only shown once.

7.6 Proposed search queries that are not included

One of the queries that were taken into consideration was to search for articles that have differences in their texts in each indexed version.

- q: url:"https://visao.sapo.pt/exameinformatica/noticias-ei/software/2019-08-08-direcoes-em-realidade-aumentada-chegam-ao-google-maps/"
- fq: !collapse field="urlkey" sort='timestamp desc'
- Raw Parameter Queries: expand=true&expand.field=article.text

Firstly we search by a *URL* and then as mentioned previously we use the *collapse* utility to group the different article versions.

Finally, in the *Raw Parameter Queries* parameter, we expand the results. Since the *expand* field is the article text, only articles that have distinct text fields will be shown.

Although the query works, at first it did not return the expected results. With this in mind, the dataset was analysed to verify if there were meaningful changes between texts. After building a python script capable of detecting changes between documents it was possible to conclude that the only differences between different versions of the same article were space and newline characters. These characters are removed by solr, which explains the obtained results.

For this reason, this query that was previously included in the set of proposed queries is not included in the presented queries.

8 EVALUATION

In this section, we measure the effectiveness of our information retrieval system. For each information need, we will measure the effectiveness of three different systems: without our custom schema or weighted attributes (**System 1**), with our custom schema but without weighted attributes (**System 2**), and with our custom schema and weighted attributes (**System 3**).

The evaluation will be based on the precision, which represents the percentage of relevant results in all the retrieved results, average precision, for the average precision over the 10 documents retrieved, mean average precision, to have a very succinct summary of the precision of our systems over different queries, recall, representing the percentage of retrieved relevant results in all the relevant results of the system, and f-measure, which combines both precision and recall in just one value.

We decided not to include *BPREF*, *Kendall tau coefficient* or *DCG* because we would have to give a ranking to the relevance of each of the articles and that would make the results even more biased and we wanted this to be as neutral as possible.

8.1 Information Needs

The main type of information need our system must attend is text search. When a user inserts a natural language query, the system must be able to retrieve the most relevant documents that match the query.

To evaluate our system we will use two different information needs:

- *aterragem em Marte* - search for an article that talks about something landing on Mars (there are 11 relevant articles in total).
- *Microsoft Teams* - search for an article that talks about the Microsoft Teams software (there are 10 relevant articles in total).

These information needs can be expressed as *Solr* queries. For that, we'll use the text search query presented in the section 7.4. Note that the last two bullet points in each query correspond to attribute weighting, which as we said will only be enabled in the third evaluation step.

8.1.1 "aterragem em Marte" query.

- q: aterragem em Marte
- defType: edismax
- qf: article.title^3 article.entities.title^3 article.text article.summary article.publish_date

8.1.2 "Microsoft Teams" query.

- q: Microsoft Teams
- defType: edismax
- qf: article.title^3 article.entities.title^3 article.text article.summary article.publish_date

8.2 Query "aterragem em Marte"

8.2.1 System 1. Evaluation without custom schema or attribute weighting. For this state of the system, the average precision measurement is 0.93 and the f-measure is 0.89.

k	1	2	3	4	5	6	7	8	9	10
Relevant	R	R	R	R	R	N	R	N	R	R
P@k	1	1	1	1	1	0.83	0.86	0.75	0.78	0.8
R@k	0.13	0.25	0.38	0.50	0.63	0.63	0.75	0.75	0.88	1

8.2.2 System 2. Evaluation with custom schema and without attribute weighting. For this state of the system, the average precision measurement is 0.66 and the f-measure is 0.82.

k	1	2	3	4	5	6	7	8	9	10
Relevant	N	R	R	N	R	R	R	R	N	R
P@k	0	0.50	0.67	0.50	0.60	0.67	0.71	0.75	0.67	0.7
R@k	0	0.14	0.29	0.29	0.43	0.57	0.71	0.86	0.86	1

8.2.3 System 3. Evaluation with custom schema and attribute weighting. For this state of the system, the average precision measurement is 0.90 and the f-measure is 0.82.

k	1	2	3	4	5	6	7	8	9	10
Relevant	R	R	R	N	R	R	R	N	R	N
P@k	1	1	1	0.75	0.80	0.83	0.86	0.75	0.78	0.7
R@k	0.14	0.29	0.43	0.43	0.57	0.71	0.86	0.86	1	1

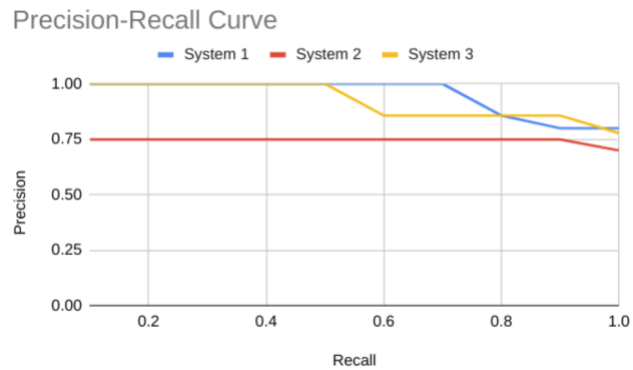

Figure 9: Precision-Recall curves for "aterragem em Marte" query for each of the systems

8.3 Query "Microsoft Teams"

8.3.1 System 1. Evaluation without custom schema or attribute weighting. For this state of the system, the average precision measurement is 0.1 and the f-measure is 0.18.

k	1	2	3	4	5	6	7	8	9	10
Relevant	N	N	N	N	N	N	N	N	N	R
P@k	0	0	0	0	0	0	0	0	0	0.1
R@k	0	0	0	0	0	0	0	0	0	1

8.3.2 System 2. Evaluation with custom schema and without attribute weighting. For this state of the system, the average precision measurement is 1 and the f-measure is 0.33.

k	1	2	3	4	5	6	7	8	9	10
Relevant	R	R	N	N	N	N	N	N	N	N
P@k	1	1	0.67	0.5	0.4	0.33	0.29	0.25	0.22	0.2
R@k	0.50	1	1	1	1	1	1	1	1	1

8.3.3 System 3. Evaluation with custom schema and attribute weighting. For this state of the system, the average precision measurement is 0.58 and the f-measure is 0.33.

k	1	2	3	4	5	6	7	8	9	10
Relevant	N	R	R	N	N	N	N	N	N	N
P@k	0	0.5	0.67	0.5	0.4	0.33	0.29	0.25	0.22	0.2
R@k	0	0.5	1	1	1	1	1	1	1	1

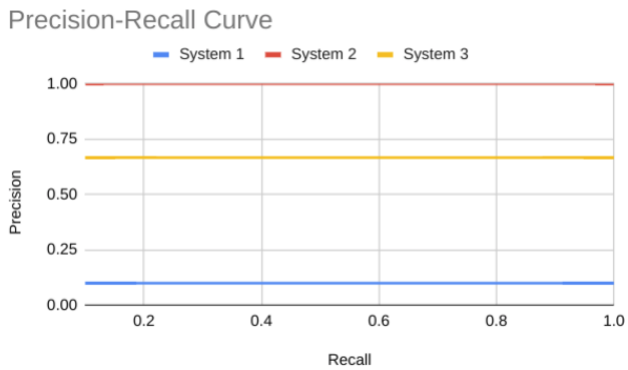

Figure 10: Precision-Recall curves for "Microsoft Teams" query for each of the systems

8.4 Evaluation Results

	Section 8.2 AvP	Section 8.3 AvP	Mean AvP
System 1	0.93	0.1	0.515
System 2	0.66	1	0.83
System 3	0.90	0.58	0.74

Table 2: Mean average precision for each of the systems

8.4.1 System 1. For the first system, surprisingly, we get a good average precision result on the first query. That happens because the query words are rarely used and when used they are used in almost the exact way they were searched. This system performs poorly for the second query, which features words that are commonly used in multiple documents separately. This results in the system not being able to deliver relevant results.

8.4.2 System 2. By adding a schema to our system, we expect the results of both queries to improve. However, that only happens for the second query. The first query delivers slightly worse results because now the words are being matched with more derivates due to the use of stemming and other filters, and these derivates aren't necessarily what we were looking for in this case.

8.4.3 System 3. Adding weights to our queries allows us to guide our system into weighing the significance of each match on a field basis. This way we can give more importance to more significant fields like the title of the article or the name of the entities that are mentioned. We can see that this improves the results for the first query but lowers the results for the second query. However, these results are more balanced and in most cases, this system should deliver more relevant results than system 2.

8.4.4 Conclusion. From these results, we can understand that without a schema that indexes and tokenizes our data, our information system would be very inconsistent. It would give good results in queries with words rarely used, like "aterragem" and "Marte", and would give bad results in specific queries with words that are used frequently through many documents, like "Microsoft".

Adding weights to the queries after adding the schema did not have a good impact overall, in most cases, it did not change the ten documents we got, it just changed the order of those documents, and that's why we have close P@5 and equal P@10 between both cases.

9 REVISIONS INTRODUCED

- Data processing and retrieval (section 2)
  - Improved the dataset by improving the entity cleaning methods and further removing invalid detected entities
  - Tried to find new newspapers that had relevant amounts of tech news (comparable to *Exame Informática*) but couldn't find any
- Indexation (section 6)
  - Added a table summarization of the dataset schema
- Evaluation (section 8)
  - Restructuration of data presented
  - Correction of recall calculations
  - New graphs were generated with new recall values
  - Added more result discussion (section 8.4)

10 QUERY WEIGHTS

To improve our search system we tested multiple 3 different weights on 2 different queries.

10.1 Systems

The systems we studied used the fields title, entities, text and summary from articles with different weights to try to find the one that fits our use cases the best.

10.1.1 Important Title. The first system we used was a system with normal weights on the entities of the article, the text of the article and its summary, and we increased the importance of the title to be 10 times more important than the other fields.

We chose this system because we thought that people who use our search system will likely choose if an article is relevant enough for them to read it based on its title.

In *Solr* we used the query field: article.title^10 article.entities.title article.text article.summary

10.1.2 Important Entities. In this system, we thought that when people search by names of persons, companies or anything else that was considered an entity, if it appears in the text it will be relevant to the user.

To not deviate too much from the previous system we decided to keep the title, text and summary of the query with normal weight and the entities with 10 times those fields, thus, getting the query field, article.title article.entities.title^10 article.text article.summary

10.1.3 Overall Importance. At last, we decided to have a system that would have some more weight on the summary of the article but without making it more important than the title or the entities present. Therefore, we decided to give a weight of 5 to both the title and the entities and 3 for the summary, the text kept a weight of 1.

The resultant query field was: article.title^5 article.entities.title^5 article.text article.summary^3

10.2 Queries

To test the systems described in the previous subsection, we used 2 different queries

10.2.1 Microsoft Teams. With this query, we wanted to search for all the news that were related to the Microsoft Teams program. As expected, this query returned some results which included any of the different Microsoft programs, or just the name of the company, and articles about some of its competitors, like Zoom, or about remote work and remote classes and not specifically about the Microsoft Teams program.

10.2.2 Prémio Nobel. In this query, the user would be searching for news related to the Nobel prizes and their owners. When executing searches we ended up with results for different prizes and with news non-related to the Nobel prizes but that mentioned it somewhere in the text.

10.3 Results

For the "Microsoft Teams" query with the first set of weights, the average precision for the first 10 results is 1 and the f-measure is 0.75.

k	1	2	3	4	5	6	7	8	9	10
Relevant	R	R	R	R	R	R	N	N	N	N
P@k	1	1	1	1	1	1	0.86	0.75	0.67	0.6
R@k	0.17	0.33	0.5	0.67	0.83	1	1	1	1	1

In the same query but giving more weight to the entities the AvP is 0.33 and the f-measure is 0.18 based on this results:

k	1	2	3	4	5	6	7	8	9	10
Relevant	N	N	R	N	N	N	N	N	N	N
P@k	0	0	0.33	0.25	0.2	0.17	0.14	0.13	0.11	0.1
R@k	0	0	1	1	1	1	1	1	1	1

When using the last system the resulting AvP is 0.73 and the f-measure is 0.82.

k	1	2	3	4	5	6	7	8	9	10
Relevant	N	R	R	R	R	R	R	N	N	R
P@k	0	0.5	0.67	0.75	0.8	0.83	0.86	0.75	0.67	0.7
R@k	0	0.14	0.29	0.43	0.57	0.71	0.86	0.86	0.86	1

For this query, we get this Precision-Recall curves, where "Title" represents the first system, "Entities" the second one and "Summary" the last system.

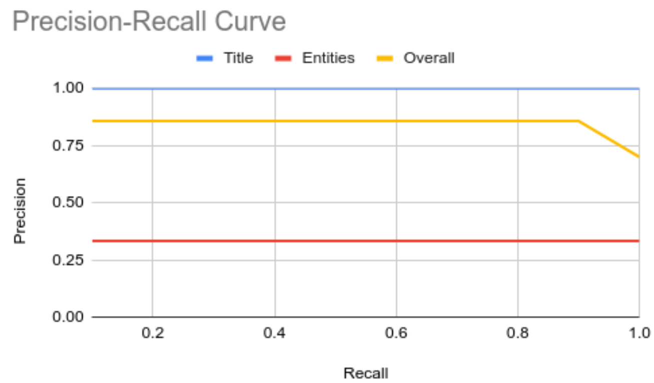

Figure 11: Precision-Recall curves for "Microsoft Teams" query with different weights

For the "Prémio Nobel" query the first system gave us an AvP of 0.75 and f-measure of 0.82.

k	1	2	3	4	5	6	7	8	9	10
Relevant	N	R	R	R	R	R	R	R	N	N
P@k	0	0.5	0.67	0.75	0.8	0.83	0.86	0.88	0.78	0.7
R@k	0	0.14	0.29	0.43	0.57	0.71	0.86	1	1	1

With the second system we had an AvP of 0.27 and f-measure of 0.33:

k	1	2	3	4	5	6	7	8	9	10
Relevant	N	N	R	N	N	N	N	N	N	R
P@k	0	0	0.33	0.25	0.2	0.17	0.14	0.13	0.11	0.2
R@k	0	0	0.5	0.5	0.5	0.5	0.5	0.5	0.5	1

When using the last system the resulting AvP is 0.73 and f-measure is 0.82.

k	1	2	3	4	5	6	7	8	9	10
Relevant	N	R	R	R	R	R	R	N	N	R
P@k	0	0.5	0.67	0.75	0.8	0.83	0.86	0.75	0.67	0.7
R@k	0	0.14	0.29	0.43	0.57	0.71	0.86	0.86	0.86	1

The Precision-Recall curves resultant from these tables can be seen in the image below.

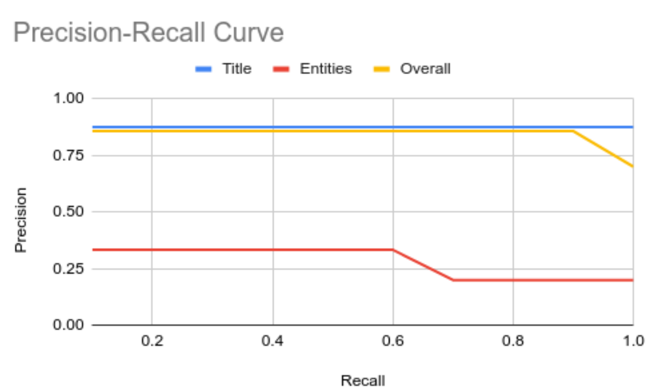

Figure 12: Precision-Recall curves for "Prémio Nobel" query with different weights

In both queries the second system we used gave bad results and, thus, we decided not to use it anymore. This can be explained because the entities extracted are based on the article text and, thus, simple references to the topics that were searched would have a high impact on the score of the document.

The other 2 systems had similar results. In the first query, in the first system, the "Microsoft Teams" query gave less relevant results but gave them all in the first 6 results, while in the third one, it gave 1 more relevant result but the first one was not relevant, thus decreasing the precision at 5. With the "Prémio Nobel" query, both systems had 7 relevant results, but the first system gave the last relevant result before the third system, making its precision-recall curve better.

With those results, we decided that for the rest of the queries, the first system will be used on all future queries, unless specified otherwise.

11 SYNONYMS

To make our search system more robust we decided that when a person searches for a specific term, the results should not only show that word, and its derivatives, but also its synonyms.

As a means to do that we added the *solr.SynonymGraphFilterFactory* filter both to the indexing of the documents and at query time. We used a synonyms file with some entries for Portuguese words that were used as synonyms, for example, "cair, baixar, descer", which means that when any of these 3 words are used, all 3 will be searched for.

To test whether the synonyms would improve our system or not, we decided to use 2 new queries to compare them and choose if we wanted to keep the synonyms or not.

11.1 App Televisão

As the first query we wanted to know any technology news about television apps, we searched this query because sometimes the word "televisão" appears as "TV" or "ecrã" and so, we wanted to check if, with those synonyms, we would have better results so we search "app televisão" on *Solr*.

When doing the query without synonyms we got an average precision of 1 and f-measure of 0.67 based on these results:

k	1	2	3	4	5	6	7	8	9	10
Relevant	R	R	R	R	N	N	N	N	N	N
P@k	1	1	1	1	0.8	0.67	0.57	0.5	0.44	0.4
R@k	0.25	0.5	0.75	1	1	1	1	1	1	1

With synonyms the AvP is 0.72 and f-measure is 0.57:

k	1	2	3	4	5	6	7	8	9	10
Relevant	R	N	R	R	N	N	R	R	N	N
P@k	1	0.5	0.67	0.75	0.6	0.5	0.57	0.63	0.56	0.5
R@k	0.2	0.2	0.4	0.6	0.6	0.6	0.8	1	1	1

Based on these results we get these precision-recall curves

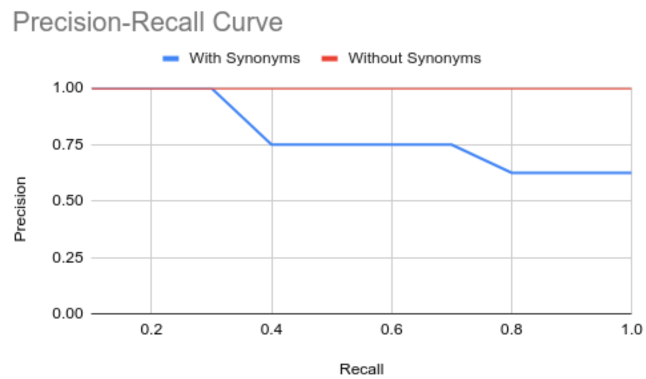

Figure 13: Precision-Recall curves for "App Televisão" query with and without synonyms

11.2 Aterragem em Marte

The other query we used was one we had already used, "aterragem em Marte", in which we wanted to get news about the landing on Mars and the perseverance robot that landed on Mars last year. It is relevant in this section because the verb "aterrar" has synonyms like "chegar" and "pousar" that appeared in some news instead of the word we searched.

In Solr we used the query "aterrar Marte" and got an AvP of 1 and f-measure of 0.95 without synonyms:

k	1	2	3	4	5	6	7	8	9	10
Relevant	R	R	R	R	N	N	N	N	N	N
P@k	1	1	1	1	0.8	0.67	0.57	0.5	0.44	0.4
R@k	0.25	0.5	0.75	1	1	1	1	1	1	1

And an AvP of 0.96 and a f-measure of 0.57 with synonyms:

k	1	2	3	4	5	6	7	8	9	10
Relevant	R	R	R	R	R	R	N	R	R	R
P@k	1	1	1	1	1	1	0.86	0.88	0.89	0.9
R@k	0.11	0.22	0.33	0.44	0.56	0.67	0.67	0.78	0.89	1

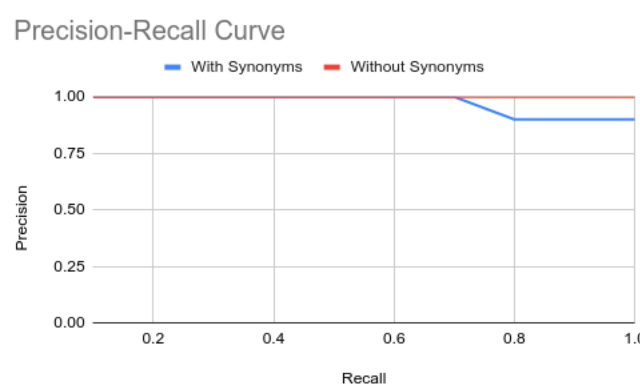

Figure 14: Precision-Recall curves for "Aterrar em Marte" query with and without synonyms

11.3 Conclusions

In both queries, the system with synonyms had more relevant results than the system without synonyms but, the non-relevant results when synonyms were not used were always shown after the relevant ones, which was not true on the other system. This made the precision-recall curves seem better for the system without synonyms but, when looking directly at the results, we found that when we used synonyms, we had more relevant results, thus, making our search system more appealing to its users.

12 FACETS

While trying to improve our system, we thought that one of the most important things would be faceting. Faceting is the arrangement of search results into categories based on indexed terms. The use of facets means that we would be able to implement a filtering functionality, that is, once the user obtains the results of the query, he can then further filter them according to the facets that they have, which improves the user's ability to explore the results.

12.1 Facet Creation

For the fields that would be turned into facets, we chose the newspaper, the authors and the entities. Taking into consideration the domain of our search system, these are the fields that provide the most value, allowing the user to filter by news of a certain newspaper, that were authored by a certain person or that mention some specific entities.

After selecting the fields that would be used as facets, we needed to copy their values to a new field (`${ORIGINAL_NAME}_facet`) in the schema, so that that field would preserve its original value, and not be changed by filters that were being applied to the original field, like the stemming filter. The exception was the filtering by 'newspaper' as the correspondent field is already stored as a string.

12.2 Facet Usage

The way we use the facets is by turning on the respective option in the solr query (`facet=true`). This will make the query return the number of documents that are categorized in each of the facet field values, for example, there would be many articles with the facet 'Google'.

By knowing the different facet values and how many articles are categorized by each, we can give that information to the user and let him choose which facet values to use in the filter query (fq) field of the solr query. If the user chooses to filter by the entity 'Google', that would be equivalent to the filter query
`fq=article.entities.title:Google`.

13 FRONTEND

To improve the interaction between the user and the search system, we implemented a simple graphical user interface. You can see the result in figure 15.

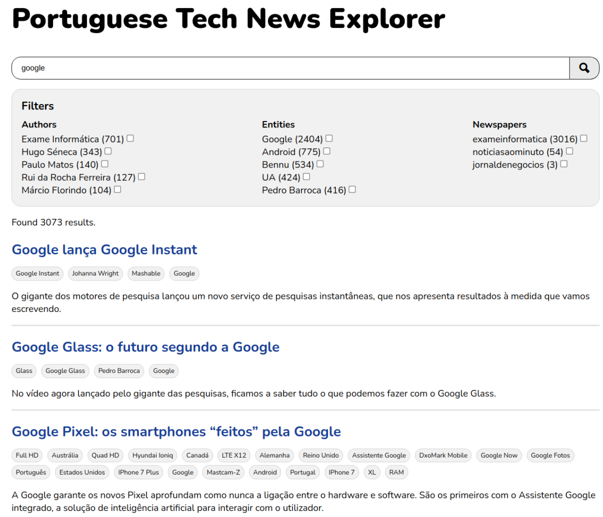

Figure 15: Frontend

With this implementation, we allow our search system to be used by more users, including ones without the technical knowledge to build solr queries.

To execute the search, we used the full-text search query that gave us the best results. This query searches over the following fields: article.title^10 article.entities.title article.text article.summary, while giving a weight of 10 to the matches in the title of the article.

Furthermore, because we implemented facet support into our search system, we were able to include query results filtering. Our implementation allows the user to filter the query results by author, entities detected and origin newspaper.

14 CONCLUSION

During the development of this project, we managed to explore Arquivo.pt and its APIs, as well as experiment with techniques such as HTML scrapping and entity extraction from natural language text.

We acquired new knowledge on how to extract useful information from web pages and on the actions that must be taken to clean and refine it into a format that we can use. We also got to analytically characterize this information by manipulating the resulting dataset, which left us with a better understanding of what is possible to be extracted from it.

We then converted the refined dataset into documents that can be indexed by Solr, defined our document schema and executed information retrieval tasks. We evaluated our system's performance and compared it between multiple configurations: without custom schema or attribute weights, with custom schema and without attribute weights, and with custom schema and attribute weights.

Lastly, we improved our system's results by experimenting with new functionalities, like synonyms and facets, and improving the weights we were using. We also made it more user-friendly by implementing a front-end for interaction with the user. In its current state, our system is able to deliver relevant results to natural language queries.

REFERENCES

[1] Arquivo.pt, https://sobre.arquivo.pt/
[2] Fundação para a Ciência e Tecnologia, https://www.fct.pt/
[3] Arquivo.pt APIs, https://github.com/arquivo/pwa-technologies/wiki/APIs
[4] Notícias ao Minuto, https://www.noticiasaominuto.com/
[5] Jornal de Negócios, https://www.jornaldenegocios.pt/
[6] Exame Informática, https://visao.sapo.pt/exameinformatica/
[7] Arquivo.pt CDX Server API, https://github.com/arquivo/pwa-technologies/wiki/URL-search:-CDX-server-API
[8] Arquivo.pt NoFrame API, https://github.com/arquivo/pwa-technologies/wiki/Arquivo.pt-API
[9] Beautiful Soup, https://beautiful-soup-4.readthedocs.io/en/latest/
[10] spaCy, https://spacy.io/
[11] Solr 8.10, https://solr.apache.org/guide/8_10/