Amazon Cell Phone Reviews

André Gomes
FEUP
up201806224@edu.fe.up.pt

André Nascimento
FEUP
up201806461@edu.fe.up.pt

Catarina Fernandes
FEUP
up201806610@edu.fe.up.pt

ABSTRACT

This paper presents the data collection, preparation, analysis and querying of datasets. An initial search was made to find a suitable dataset to be used for the project. After settling with a dataset about Amazon cell phones reviews retrieved from Kaggle, there was an initial data analysis of the data, followed by a scraping of updated data. A merge of updated and already existing data was made to have a more complete dataset, followed by a data exploration using Python's Pandas library. Afterward, the data was cleaned and refined process using Python scripts, and, to finalize the pipeline, SQL files to create and seed the database. Through the usage of Solr, an information retrieval tool, we define a collection and its respective documents, and describe their indexing process. The defined information need are queried, while also exploring some of the tool's features, and evaluated according to some relevant metrics that compare the different system's configurations.

KEYWORDS

scraping, data retrieval, data processing, Amazon cell phone reviews, information retrieval

1 INTRODUCTION

Information communication technologies are ubiquitous in modern societies. It is a broad subject that covers any product that will store, retrieve, manipulate, transmit, or receive information electronically in a digital form [8]. An ever-growing number of activities depend on the ability to extract value from information and Human progress and welfare are largely dependent on efficient management of the life cycle of information.

With this rise in the importance of data and its processing, new professional profiles have been appearing, such as data engineers, data architects, data analysts, data scientists.

In this course, we were subjected to an introduction to information processing and information retrieval. To solidify our knowledge, we were tasked with developing an information search system, including work on data collection and preparation, information querying and retrieval, and retrieval evaluation.

The project is divided into three milestones, namely: data preparation, information retrieval, and final search system.

This report covers the first milestone, data preparation, in which we performed the following actions:

- search repositories for datasets;
- select convenient data subsets;
- assess the authority of the data source and data quality;
- perform exploratory data analysis;
- prepare and document a data processing pipeline;
- characterize the datasets, identifying and describing some of their properties;
- identify the conceptual model for the data domain;
- identify follow-up information needs in the data domain.

2 DATA COLLECTION

While searching for datasets, we faced common problems amongst the data, such as the low amount of rows and/or columns, low word count in each entry for a given column (we need columns with large bodies of text to later perform full-text search) and not so diversified text bodies (we were looking for datasets which weren't too repetitive).

The first dataset that caught our eyes was related to wine reviews. It consisted of a single .csv file with 130.000 entries. Each entry had information about a bottle of wine and its review. We could gather information about the wine, like its designation, country of origin, region, province, and price, and information about the taster and its reviews, like the taster's name, his classification of the wine, and his description of the taste.

We ended up ditching this first dataset, not because it fell under any of the common problems mentioned earlier, but because we thought we did not have enough information to work with, since this dataset only gave us one table to manage.

Later on, we found the dataset explored in this report: Amazon cell phone reviews [10]. We found it via Kaggle [2], a platform used to host data science competitions and explore, analyze and share datasets. This dataset is composed of two .csv files: a list of cell phones and a list of cell phone reviews. The data was obtained from amazon via scraping by Griko Nibras, the author of the dataset and the scraping code, which wanted to aggregate the amazon reviews of cell phones from 10 brands of his choice: Asus, Apple, Google, HUAWEI, Motorola, Nokia, OnePlus, Samsung, Sony, and Xiaomi.

This dataset was obtained from amazon in 2019, making it a quite recent aggregate of data, a good point that inclined us to use it in this project.

From the first .csv file, the phones list, we can retrieve information about various phones brands, the titles of the amazon listing, the amazon URL, the main image URL, and the URL of the reviews, as well as some metrics like the original price and current price of the device, the total number of reviews and the rating of the product, from one to five. This file contains 720 entries.

From the second .csv file, the reviews list, we have various reviews of the cell phones, which include information about the reviewer, like its name and its validity (an Amazon metric), and information about the review: its title and body of text, the date of the submission of the review, the score given to the product and the amount of "helpful votes" of that review, more commonly known as upvotes, which is how many people signed that review as helpful. This file contains 68 thousand entries.

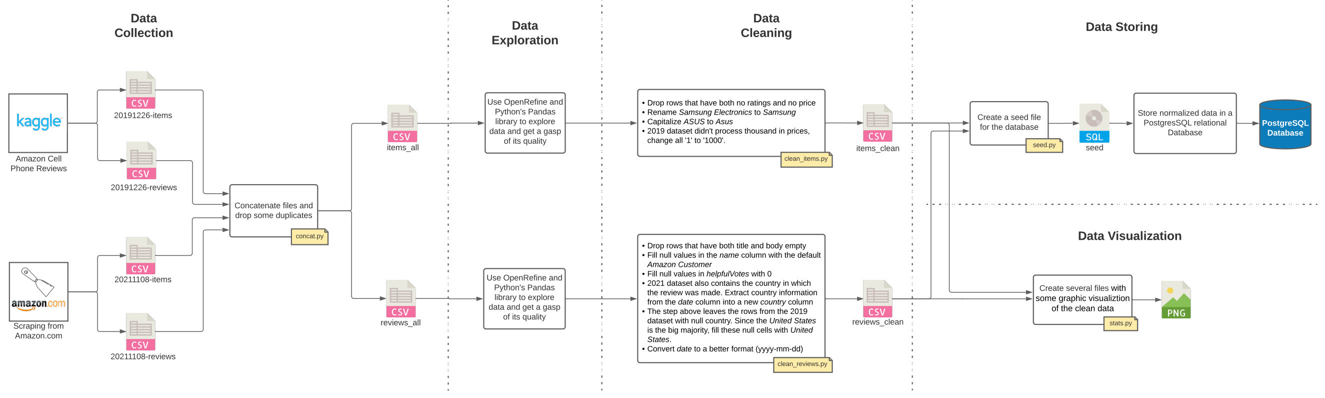
Data Collection
Data Exploration
Data Cleaning
Data Storing
Data Visualization
kaggle
Amazon Cell Phone Reviews
amazon.com
Scraping from Amazon.com
CSV
20191226-items
20191226-reviews
20211108-items
20211108-reviews
Concatenate files and drop some duplicates
concat.py
items_all
reviews_all
Use OpenRefine and Python's Pandas library to explore data and get a gasp of its quality
• Drop rows that have both no ratings and no price
• Rename *Samsung Electronics* to *Samsung*
• Capitalize *ASUS* to *Asus*
• 2019 dataset didn't process thousand in prices, change all '1' to '1000'.
clean_items.py
• Drop rows that have both title and body empty
• Fill null values in the *name* column with the default *Amazon Customer*
• Fill null values in *helpfulVotes* with 0
• 2021 dataset also contains the country in which the review was made. Extract country information from the *date* column into a new *country* column
• The step above leaves the rows from the 2019 dataset with null country. Since the *United States* is the big majority, fill these null cells with *United States*.
• Convert *date* to a better format (yyyy-mm-dd)
clean_reviews.py
items_clean
reviews_clean
Create a seed file for the database
seed.py
SQL
seed
Store normalized data in a PostgresSQL relational Database
PostgreSQL Database
Create several files with some graphic visualiztion of the clean data
stats.py
PNG

Figure 1: Data Preparation Pipeline

Griko Nibras was generous enough to provide the code with which he performed the scraping [9], and since the data is at least two years old, we decided to get another set of .csv files by running the scraping ourselves. The downside of this approach was that we needed to update the initial code, this was mainly because the amazon website was updated and some things were not getting captured correctly, for example, the Samsung phones are now on a category named "Samsung Electronics", the HTML selectors were outdated and the parsing of prices above 1000 dollars was being done incorrectly. With these small bugs fixed, we were able to perform the scraping, which took around – minutes. In the end, we were able to obtain another pair of .csv files with the most updated values possible.

3 DATA LIMITATIONS

Some limitations regarding the dataset enrichment were encountered. The first one was about the text body of the reviews of text: The scraping was performed on the English version of the Amazon website, but the customers are from around the world, as well as the reviews left on the products, in this sense, we were faced with reviews from some countries that were written in languages that contained special characters and accented letters. We decided to leave these reviews in the dataset and we'll see how they fare in the future stages of the project.

Another problem we faced was that the "brand" column in the items table would sometimes be empty, not due to a scraping error, but a data error, since the product would not have an established brand selected. Luckily, most of the time, the brand name is incorporated in the title of the amazon product, which opened the possibility of filling the empty brand names with information from the title.

Another common occurrence in the data was that, in some products, the current price would be 0, which from a business standpoint does not make much sense. After verifying the URLs of those products, we concluded that the products with a price equal to 0 were either, at the time of scraping, unavailable, for various motives, for example, out of stock, or not on sale anymore.

Lastly, there were also some items that did not have reviews but were still being captured by the scraper. We decided to leave these items just, because they can still be searched for and have all the other columns with valid data.

4 DATA PREPARATION

For a first approach, we decided to try out OpenRefine to process the dataset. After reading the official OpenRefine Documentation [5], we had several steps ready to perform on the data and some facets made up to analyze the data. Using the facets we were able to automatically detect the data types of some columns, find which columns have null values, and convert dates to ISO-8601-compliant extended format with time in UTC: YYYY-MM-DDTHH:MM:SSZ.

With OpenRefine, we can make the data processing steps via the GUI and export a JSON file describing these steps, to be able to recreate the steps all at once without going one by one manually again. But the problem that emerged from here is that OpenRefine does not come with a CLI out-of-the-box to perform these operations. Together with the fact that OpenRefine is a server that needs to be initiated, instead of a process that can be run via a .exe or via the command line, it made the process of creating a pipeline in a Makefile more difficult than it should be. To run a pipeline using OpenRefine in a Makefile, we would need to make sure that the OpenRefine server is first running, and then we would need to find a way to send the configuration file and the data to the server. This is not specified in a simple way in the OpenRefine documentation, so we decided not to invest in OpenRefine for the processing of the data, but we still used it for its Facet's capabilities.

After dealing with OpenRefine, we decided to use python, together with the pandas library [3], for the data processing phase. We first merged the 2019 and 2021 datasets to have a more complete .csv file. In theory, the 2021 dataset would have all of the contents of the 2019 dataset, but that was not the case, we had 730 items in 2019 and 823 items in 2021, but only 88 items were on both the datasets, so by merging the two datasets, we were able to expand our data, more than we expected.

After merging, we made the following changes to the dataset in Python, starting with the Items file:

- Drop rows that have both no ratings and no price (items unavailable to buy and that have no reviews, we have no interest in these)
- Rename Samsung Electronics to Samsung, Amazon changed the brand name so the 2021 dataset contained this brand instead.
- Capitalize ASUS to Asus (some cells had ASUS and others Asus)
- 2019 dataset didn't process thousand in prices. Change all '1's to '1000'. Only 2 occurrences, one in the price column and another in originalPrice.

And the Reviews file:

- Drop rows that have both title and body empty (they also have no helpfulVotes)
- Fill null values in the name column with the default Amazon Customer (only 2 occurrences)
- Fill null values in helpfulVotes with 0
- 2021 dataset also contains the country in which the review was made, split date column into two columns: country and date
- The step above leaves the rows from the 2019 dataset with null country. Since the United States is the big majority, fill these null cells with United States.
- Convert date to a more data-friendly format (yyyy-mm-dd)

The finalized pipeline can be seen in 1.

5 CONCEPTUAL MODEL

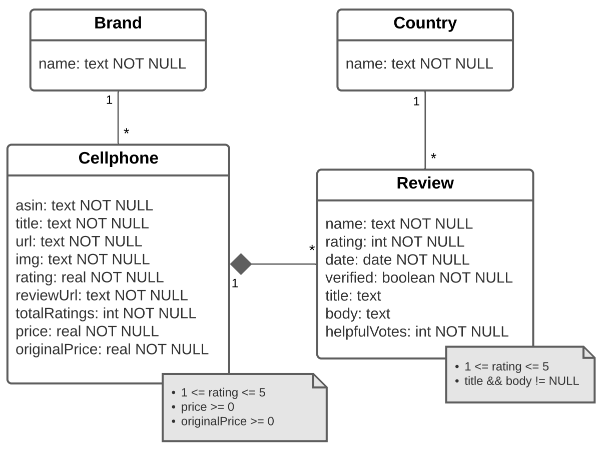

Figure 2: Conceptual Model

The conceptual model consists of a few classes, with the *Cellphone* class being the main one. The other classes help to complement it with additional information. Those classes are *Brand*, *Review* and *Country*. The main class has the following attributes that together result in the complete information of the cellphone:

- *asin* stands for Amazon Standard Identification Number. Each products has a unique asin.
- *title* corresponds to the title of the product in the publication.
- *url* is the link to the amazon products.
- *img* is a link to the main image of the product.
- *rating* is a number, from 1 to 5, corresponding to the average of all ratings.
- *reviewUrl* is a link to the first page of reviews of that item.
- *totalRatings* is the amount of ratings that the product has, each rating goes from 1 to 5.
- *price* is the current price of the product. The price could be 0, in which case, the product is not currently available.
- *originalPrice* is used when the item is in discount, this attribute would have a value and *price* would have a lower value. If the item is not on discount, this value is 0 .

The *Brand* class only has one attribute, name, which corresponds to one of the 10 brand names, as well as the *Country* class, while the *Review* class has the following attributes:

- *name* is the name of the costumer that submitted the review
- *rating* is the score, from 1 to 5, that was given to the item
- *date* is the date of when the reviews was submitted
- *country* is the country of the costumer from where he submitted the review
- *verified* is a boolean value that represents if the reviewer has bought the item
- *title* is the title of the review
- *body* is the body of text of the reviews
- *helpfulVotes* is the amount of people that marked that review as helpful

6 SEARCH TASKS

After retrieving the data, merging, and cleaning, we can perform queries to get some insights about the dataset. Here are some of the possible queries to be made to the database:

- Search a cell phone by its title, brand, rating, and price
  - Returns a list of cell phones filtered by the desired parameters
- Search a review by the reviewer's name, its title, body, country
  - Returns a list of reviews filtered by the desired parameters
- Check which brands are more developed
  - Return the Number of cell phones on cell globally for each brand
- Check which products and brands are more liked by the reviewers
  - Return the number of 1/2/3/4/5 stars reviews per item and per brand
  - Return the brands with most/least positive feedback

7 DATASET CHARACTERISATION

In order to better understand and characterize the collected data, charts were developed concerning the following aspects of the database:

Regarding the cell phones data, we would like to have a visual understanding of how many cell phones are currently being sold by brand name, and we can conclude from figure 3 that, in the amazon website, Samsung has the market dominance while ASUS only has 28 cell phones for sale.

From figure 4 we can check that Xiaomi is the brand with the highest average rating, while Apple has the lowest, this may be due to the fact that most of the Apple phones that are being sold on Amazon are refurbished, and not original.

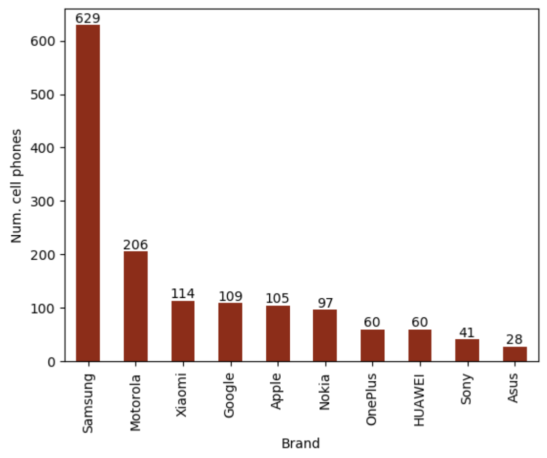

Figure 3: Amount of cell phones per brand

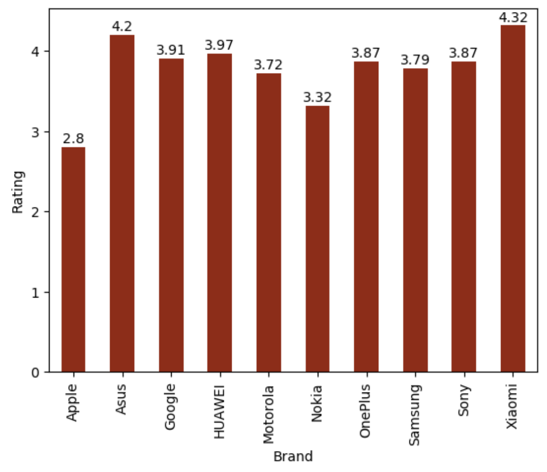

Figure 4: Average cell phone rating per brand

Concerning the reviews, from figure 5 we can conclude that the rating that is given the most is 5 stars, followed by 1, while the lowest is 2. This means that amazon users prefer to either give a 1-star rating or 5-star rating, instead of a middle ground. Since 5 stars are the norm, they are also not that valuable.

There are three main textual fields in the system: items' titles, reviews' titles and reviews' bodies. The count of total and unique words can be seen in Table 1.

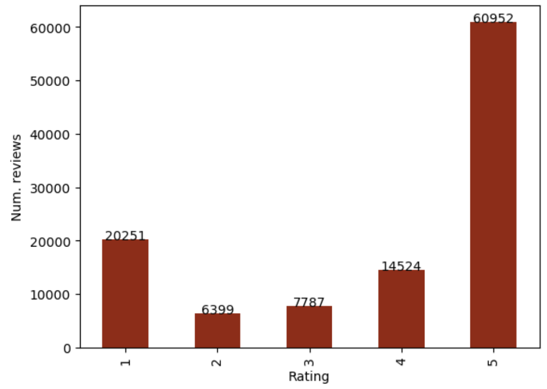

Figure 5: Number of reviews per rating

	Total	Unique
Item Titles	21820	2852
Review Titles	479417	25539
Review Bodies	5745338	185471

Table 1: Word count

Word Clouds were also plotted for these 3 main textual fields. In regards to the items' titles (fig. 6), we can see that most of the words describe the cell phone features: if it is carrier unlocked, what type of connectivity (e.g. 4g), and storage space. When it comes to the reviews' titles (fig. 7), the most used words are the rating and the main positive points of the phone.

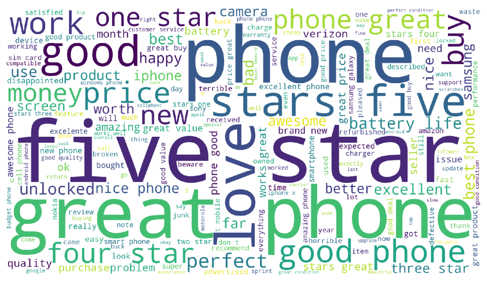

Figure 6: Reviews' title Word Cloud

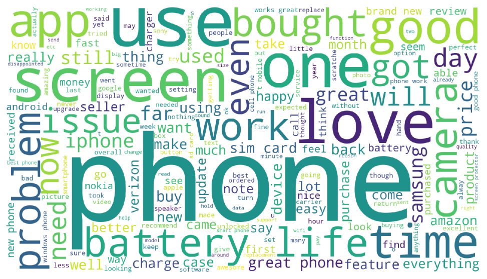

Figure 7: Reviews' body Word Cloud

8 INFORMATION RETRIEVAL

Information retrieval is a field concerned with the structure, analysis, organization, storage, searching, and retrieval of information. The goal is to build a system that implements such principles and provides the respective analysis.

The key goal of the system, as any Information Retrieval system, is to retrieve all the items that are relevant to a user query, while retrieving as few non-relevant items as possible [12]. This study will focus on ad hoc search, the most common retrieval task. Ad hoc search refers to the task of returning information resources related to a user query formulated in natural language.

In order to analyze the results of possible retrieval tasks on the Amazon Cell Phone Reviews domain, specified previously, we used three different systems and five information needs, which we describe in section 12.

The set of information specified in section 10 will be translated to a natural language query and tested in each system. Due to the high number of results, the criteria used to evaluate each query were Precision at 10 (P@10), Average Precision (AvP), and a Precision-Recall curve. P@10 measures the precision, which is the fraction of the retrieved results that are relevant, at a fixed low level of retrieved documents, 10 in this case. AvP, for a single information need, is the average of the precision value obtained for the set of top k documents existing after each relevant document is retrieved. Finally, for each query, we will plot a Precision-Recall curve.

9 INFORMATION RETRIEVAL TOOL

The two main information retrieval tools considered for this project were Solr [4] and Elasticsearch [1]. Both are search platforms based on the Lucene library tools, open-source, and offer a variety of features, such as near real-time indexing, distributed full-text search, high availability, and support for NoSQL data.

Solr is a mature tool, with a broad user community, and the group members already had some experience. It also works best in search applications that use significant amounts of static data. Although Elasticsearch has been the most popular search engine since 2016 [13] and deals better with scaling, data analytics, and processing time series data [11], it was not our first option as Solr has the capability of performing more complex queries, which was needed for this study.

Even though exploration of ElasticSearch and its practical uses was made, Solr was the selected tool for the project due to work and experiments previously made.

10 COLLECTION AND DOCUMENTS

Our system has two different types of documents: items, that represent Amazon's cell phones, and their respective reviews. There is a composition relation between the items and reviews, since the reviews don't have a purpose in the system if the reviewed item didn't exist. We took that into consideration when defining how to upload the documents to Solr. The items and reviews' data could be uploaded separately, as two different collections to the same core, and later query on both to retrieve aggregated data, but the most obvious approach was to take advantage of the Solr's nested documents feature, natively preserving the relationship between items and reviews, and index all the documents in a single collection where all the information needs will be queried upon.

In order to index the reviews as child documents of the respective reviewed item document, items and reviews' data were merged and aggregated into a single JSON file, with an array of JSON objects, where the reviews are an array inside each item.

11 INDEXING PROCESS

Before starting the indexing process, we studied which fields from each document should be indexed. We decided to only index the fields necessary to fulfill the defined information needs, by serving as a query parameter. The created schema can be consulted in Table 2.

Type	Field	Indexed
pint	totalRatings	true
	rating_review	true
	helpfulVotes	true
pfloat	rating_item	true
	price	true
	originalPrice	true
boolean	verified	false
string	asin	false
	content_type	true
	url	false
	image	false
	reviewUrl	false
title	title_item	true
	title_review	true
text_body	body	true
text_standard	brand	true
	name	true
	country	true

Table 2: Schema fields

The document's fields with numeric values were defined with Solr's *pint* and *pfloat* types, according to their precision, the date field uses *pdate* type, and the textual fields that contain simple data, such the brand, and that won't be used in queries, were defined with the *string* type. The more complex and important textual fields, that will be used in most of the queries, were indexed using custom field types that provide extra filters to enhance the queries and provide more relevant results. The defined custom field types are *title*, *text_body*, and *text_standard*, all use Solr's StandardTokenizerFactory as their tokenizer„ but use different filters, that are more appropriate to some than others, present bellow:

- *EdgeNGramFilterFactory* - generates edge n-gram tokens, in our case from 2 to 5 letters for each word. This is useful for FTS.
- *ASCIIFoldingFilterFactory* - 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.

- *LowerCaseFilterFactory* - applies a lower case filter that converts all letters in a token to lower case, so matches occur no matter the capitalization.
- *EnglishMinimalStemFilterFactory* - stems plural English words to their singular form.
- *StopFilterFactory* - this filter discards, or stops analysis of, tokens that are on the given stop words list. We use the standard stop words list included with Solr, named stopwords.txt, which is appropriate for typical English language text.

A summary of the custom field types and their filters can be seen in Table 3.

Field Type	Filter	Index	Query
title	EdgeNGramFilterFactory	✓	×
	ASCIIFoldingFilterFactory	✓	✓
	EnglishMinimalStemFilterFactory	✓	✓
	LowerCaseFilterFactory	✓	✓
text_body	StopFilterFactory	✓	✓
	ASCIIFoldingFilterFactory	✓	✓
	EnglishMinimalStemFilterFactory	✓	✓
	LowerCaseFilterFactory	✓	✓
text_standard	ASCIIFoldingFilterFactory	✓	✓
	LowerCaseFilterFactory	✓	✓

Table 3: Custom field types

12 RETRIEVAL PROCESS

After the indexing process is complete, the next step was to decide how to retrieve the information from those documents. The retrieval process consists of two main phases: choosing the query parser, and selecting and optimizing the parameters of the selected parser. After analyzing the variety of parsers provided by Solr [6], the Extended DisMax (eDismax) query parser [7] was chosen. It is an improved version of the DisMax query parser, which already adds relevant parameters in relation to the Solr's Standard query parser, and provides us more query parameters and better overall support. From all the eDismax's parameters available the ones chosen were:

- *q* - defines a query for the documents
- *qf* - a list of fields, each of which is assigned a boost factor to increase or decrease that particular field's importance in the query. The fields used in the queries, according to their respective needs, were our main textual fields: *brand*, *title_item*, *title_review*, and *body*
- *fq* - defines a query that can be used to restrict the superset of documents that can be returned, without influencing score. Helpful to filter dates and ratings in our documents.
- *sort* - arranges search results in either ascending or descending order.

Five different information needs were defined, based on the possible retrieval tasks previously defined in Section 6. For each information need, we provide a simple description, the textual query, and the query parameters.

To evaluate the results of these information needs, and the impact of our schema and parameters, three different systems were configured:

- S1 - a baseline, filterless system, where all the documents have either Solr's default types or our custom types but without any filters (only the Standard tokenizer).
- S2 - a system with our complete specified custom schema.
- S3 - a system with the schema, and also with weights where the relevant fields are boosted in the retrieval process. The weights can be seen in Table 4. The *brand* and *country* textual fields were not boosted since they only contain single words.

Field	Weight
title_item	1.5
title_review	1.7
body	1.3

Table 4: Fields' weights

To evaluate each system performance, for each information need, we analyzed the top 10 results relevance and calculated the P@10, AvP, and plotted the Precision-Recall curve.

12.1 Mexican reviews

Information Need: Reviews from Mexico.

In this query we intend to retrieve reviews related to Mexico, by searching the word "Mexico" in the *country* field, but also the review's *title* and *body* to search for references to that country, in order to check people's opinions.

Query: Mexico

Parameters:

- qf: title_review body country
- fq: content_type:review

Conclusion: This query serves as a baseline to evaluate the different systems. from the plot in figure 8 we can check that all 3 systems behaved similarly with a minimal difference, the line for the filterless system coincides with the line for the system using a schema. Since the query is also simple, it wouldn't be correct to make more general assumptions or conclusions.

12.2 Christmas gifts

Information Need: Gift reviews in the 2020 Christmas season.

In this query, we intend to retrieve reviews that contain the word "Gift", either on its title or its body, and that were made between November 2020 and January 2021. We will see several opinions on phones, so we can choose the best gift for this Christmas.

Query: Gift

Parameters:

- qf: title_review body country
- fq: content_type:review date:[2020-11-01T00:00:00Z TO 2021-01-31T00:00:00Z]

Conclusion: Analysing the plot in figure 9, we can now see a difference between the unfiltered query and the queries using schemas. The queries on systems using schemas not surprisingly perform better, but the system without filters was not completely useless since the precision did not drop below 0.5.

12.3 Relevant 5-star reviews

Information Need: Positive and relevant 5-star reviews to the community.

In this query, we intend to retrieve reviews that contain the word "Great" or "Good", either on its title or its body, positively describing an item, with a 5-star rating, that were verified, and have at least 10 helpful votes from other users.

Query: Great Good

Parameters:

- qf: title_review body
- fq: content_type:review verified:true rating_review:5 helpfulVotes:[10 TO *]

Conclusion: This query was not complex to the point of using sentiment analysis beforehand to discretize positive and negative reviews. We decided to query by positive terms and check which kind of reviews were returned. In this query, the disparity between the schema and schemaless systems is noticeable, just like in the previous query. Since we searched for capitalized words, the filterless system struggled to find the relevant documents because some of those had "great" or "good". The results can be seen in figure 10

12.4 Google or Apple phones

Information Need: Google phones with 4g connectivity or refurbished iPhones.

Query: (Google AND 4g) OR (iphone AND refurbished)

Parameters:

- qf: title_item brand
- fq: content_type:item

Conclusion: In figure 11 the results for both schema systems overlapped, meaning they had similar performance, which is still better than the performance for the filterless system. This query is the most complex in terms of textual fields, and here it can be seen the importance of having a well-defined schema since the filterless system got a precision of 0.

12.5 Samsung's discounts

Information Need: Samsung cell phones with the highest discount

In this query, we retrieve items that contain "Samsung", either on the brand or title, and sort them by the highest absolute discount (original price - current price).

Query: samsung

Parameters:

- qf: title_item brand
- fq: content_type:item -originalPrice:0.0
- sort: dist(2, originalPrice, 0, price, 0) desc

Conclusion: For the final query we wanted to check the differences between systems using a simple query but a complex sort attribute. By searching "samsung" and not "Samsung" the filterless couldn't return the desired documents and so performed badly (figure 12). After changing the query to "Samsung" the results would be the same as the other systems, meaning that the sorting doesn't alter the performance between systems, and the query terms are more important.

After running all the queries in the different systems, we gathered the average precision and the precision at 10 for each query, presented in the table **??**.

Query	**System 1**		**System 2**		**System 3**	
	AvP	*P@10*	*AvP*	*P@10*	*AvP*	*P@10*
1	0.70839	0.7	0.70839	0.7	0.624603	0.6
2	1.0	0.1	1	1	0.969798	0.9
3	0.729167	0.4	1	0.8	1	0.7
4	0	0	0.710768	0.7	0.710768	0.7
5	0	0	1	1	1	1

13 RETRIEVAL TOOL EVALUATION

Since Solr was the only tool used, there is no way to evaluate it and compare it to other information retrieval tools available. Nonetheless, it is possible to draw some conclusions from the experience when designing this system. The negative aspect of Solr is its documentation, which is not clear enough and doesn't provide relevant examples in some parts, increasing the learning curve to perform the required tasks. Most of the time was spent learning how to correctly upload nested documents and work with them, so we could retrieve items associated with the retrieved reviews, which proved not so trivial, and doesn't work with boosted fields. On the other hand, Solr offers a variety of options, very helpful for the needed IR tasks, including multiple ways to both index and query documents.

In conclusion, despite Solr's disadvantages, it still allows the implementation of a good information retrieval system, and the previously discussed results prove that it is possible to achieve satisfying solutions.

14 CONCLUSIONS AND FUTURE WORK

In the initial sections of the paper, regarding the Data Preparation phase, the processing pipeline is documented, as well as all the tools used, showing all the steps that led to the creation of a complete and coherent dataset about cell phones and their reviews. Some graphics and statistics are also present to better visualize the data. All of the goals for this phase were accomplished given that there is a better understanding of the chosen domain, the only setback we had was the data preparation using OpenRefine, which was midway changed to Python.

Regarding the Information Retrieval phase, the search tool selected, Solr, is described and the motives for that choice. The previously cleaned datasets were merged to a single JSON and uploaded to Solr as a file with nested documents, indexing each imported document. In this process, custom types were defined for some fields, to better accommodate our data and information retrieval needs, which allow the removal of stop words, stemming, lower case transformation, among others, in indexing and/or querying time. Finally, to explore the retrieval process, we conceived five user information needs and performed comparisons between three different systems, evaluating the performance through the precision from the first ten documents retrieved and the average precision. We were able to evidence the impact of Solr's filters and parameter boosting in each query and we concluded that the schema definition is an important step in any IR project. The most difficult and time-consuming part was trying to understand how to work with nested documents while also boosting fields in both parent and child documents. Apparently, it is not possible to retrieve parent documents when we want to apply weights, so we ended up just querying either the items or reviews as if they were separate documents.

REFERENCES

[1] [n. d.]. Elasticsearch. https://www.elastic.co/
[2] [n. d.]. Find open datasets and Machine Learning Projects. https://www.kaggle.com/datasets
[3] [n. d.]. Pandas. https://pandas.pydata.org/
[4] [n. d.]. Solr. https://solr.apache.org/
[5] Antonin Delpeuch. 2021. Openrefine User Manual. https://docs.openrefine.org/
[6] Apache Software Foundation. [n. d.]. Query Syntax and Parsing. https://solr.apache.org/guide/8_11/query-syntax-and-parsing.html
[7] Apache Software Foundation. [n. d.]. Query Syntax and Parsing. https://solr.apache.org/guide/8_11/the-extended-dismax-query-parser.html
[8] Information and Communications Technology. 2021. Information and Communications Technology. https://en.wikipedia.org/wiki/Information_and_communications_technology
[9] Griko Nibras. [n. d.]. Grikomsn/amazon-cell-phones-reviews: Scrape (UN)locked cell phone ratings and reviews on Amazon. https://github.com/grikomsn/amazon-cell-phones-reviews
[10] Griko Nibras. 2019. Amazon cell phones reviews. https://www.kaggle.com/grikomsn/amazon-cell-phones-reviews
[11] Dragan Ivanović Nikola Luburić. [n. d.]. 287Comparing Apache Solr and Elasticsearch search servers. http://www.eventiotic.com/eventiotic/files/Papers/URL/icist2016_54.pdf
[12] Berthier Ribeiro-Neto Ricardo Baeza-Yates. 2011. Modern Information Retrieval: The Concepts and Technology behind Search. http://www.mir2ed.org/
[13] Asaf Yigal. [n. d.]. Solr vs. Elasticsearch: Who's The Leading Open Source Search Engine? https://logz.io/blog/solr-vs-elasticsearch/

A IMAGES

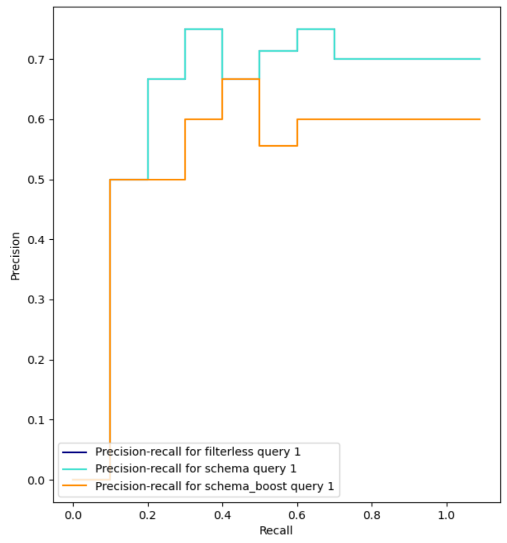

Figure 8: Query 1

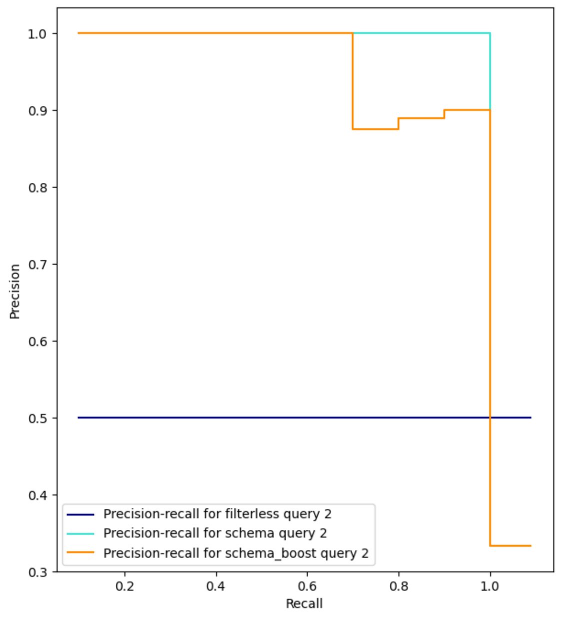

Figure 9: Query 2

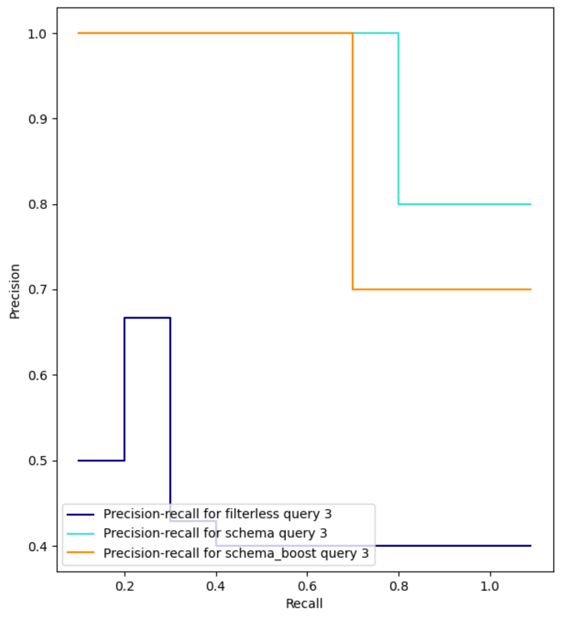

Figure 10: Query 3

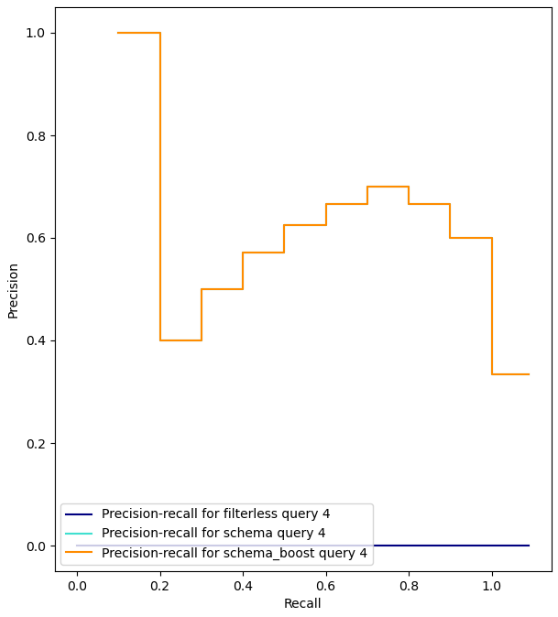

Figure 11: Query 4

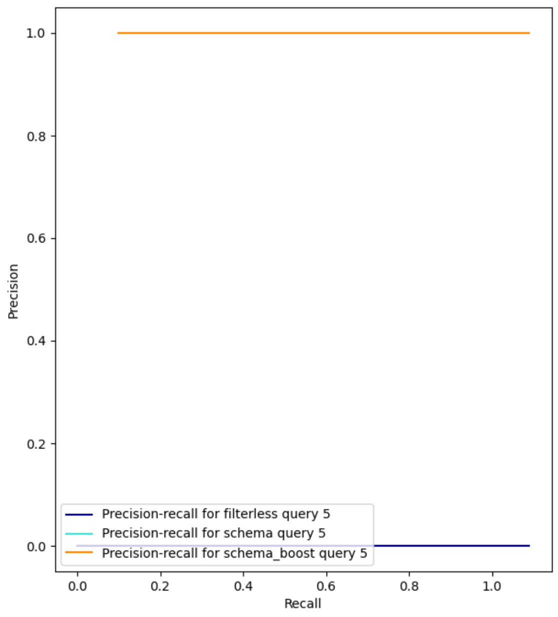

Figure 12: Query 5