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 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 needs 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. After this initial testing and evaluation, more item scraping was done to make the search system more complete, Solr's searching features were explored to further enhance the system, and finally more testing was done in order to compare the new behavior to the original system. In the end, an intuitive and appealing user interface was developed, to provide a good experience when using our system.

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 [13]. An ever-growing number of activities depend on the ability to extract value from information so, 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. [22]

To solidify our knowledge on information processing and information retrieval, 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.

We start by detailing 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 [15]. We found it via Kaggle [14], 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

Information Processing and Retrieval,
2021.

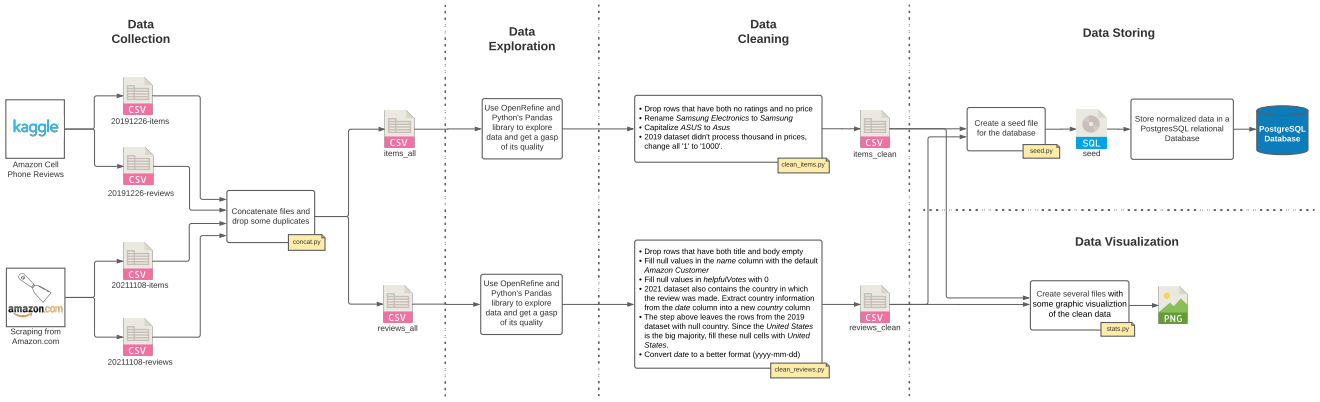

Figure 1: Data Preparation Pipeline

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.

Griko Nibras was generous enough to provide the code with which he performed the scraping [16], 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, obtaining 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, some items did not have reviews but were still being captured by the scraper. We decided to leave these items 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 [3], 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 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 [18], 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

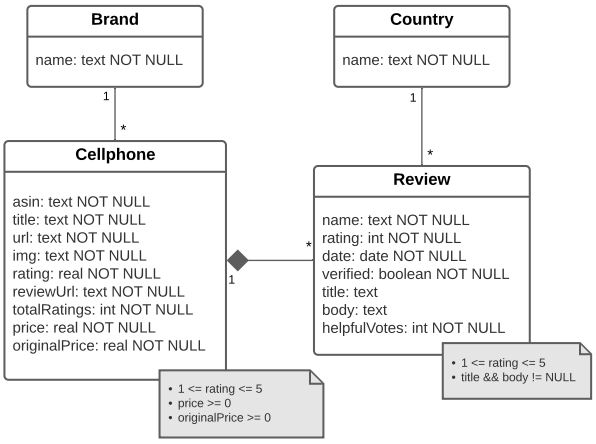

Figure 2: Conceptual Model

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 product 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 number 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 customer 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 were submitted
- *country* is the country of the customer 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 the text of the reviews
- *helpfulVotes* is the number 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.

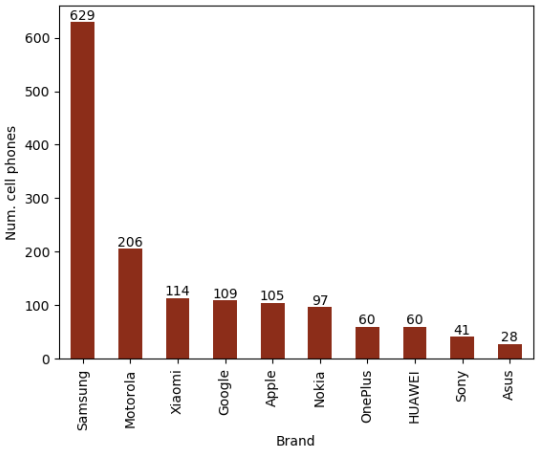

Figure 3: Amount of cell phones per brand

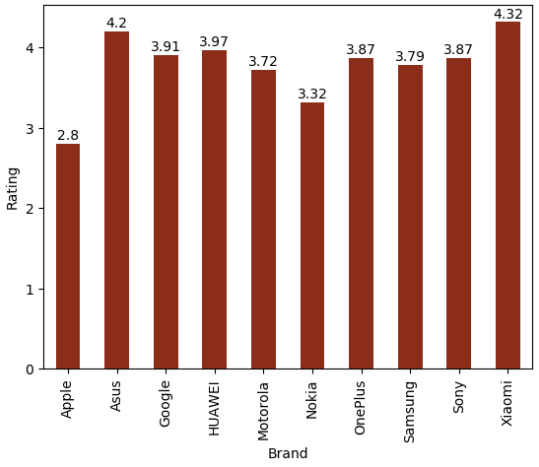

Figure 4: Average cell phone rating per brand

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.

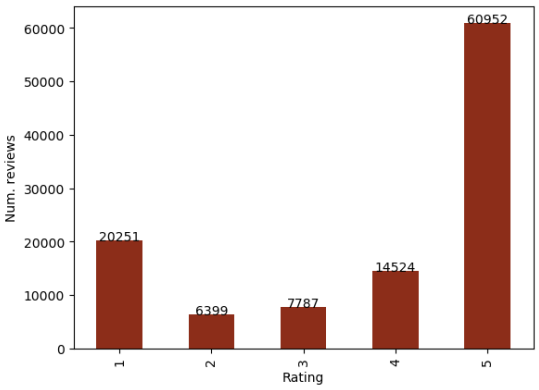

Figure 5: Number of reviews per rating

	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

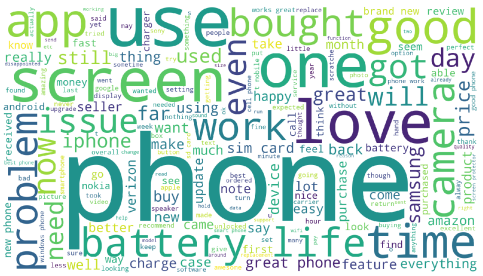

Figure 7: Reviews' body Word Cloud

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 [20]. 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 [11] and Elasticsearch [2]. 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 [23] and deals better with scaling, data analytics, and processing time series data [17], 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
	brand	true
	url	false
	image	false
	reviewUrl	false
	name	false
	country	true
title	title_item	true
	title_review	true
text_body	body	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, *title* and *text_body*, all use Solr's StandardTokenizerFactory as their tokenizer, but use different filters, that are more appropriate to some than others, present bellow:

- *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	ASCIIFoldingFilterFactory	✓	✓
	EnglishMinimalStemFilterFactory	✓	✓
	LowerCaseFilterFactory	✓	✓
text_body	StopFilterFactory	✓	✓
	ASCIIFoldingFilterFactory	✓	✓
	EnglishMinimalStemFilterFactory	✓	✓
	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 [8], the Extended DisMax (eDismax) query parser [9] 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

The weights given above were determined while analyzing how Solr calculated the scores of each returned document and how the weights affected the order of the documents. For the items, we gave more weight to the title and kept the default weight for the brand, as the title contains information about the item's model and some features. For the reviews, we gave slightly more weight to the body content and to the title, keeping the country's weight 1. We decided to give a bigger weight to the review title than to the body since that concise and relevant information is likely to be found in that field.

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. In the end, we calculate the mean average precision (MAP) for each system, to see which performed better overall.

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.

Rank	System 1	System 2	System 3
1	N	N	N
2	R	R	R
3	R	R	R
4	R	R	N
5	R	R	R
6	N	R	R
7	R	N	R
8	R	R	N
9	N	R	N
10	N	N	N
AvP	0.70	0.73	0.63
P@10	0.60	0.70	0.50

Table 5: Query 1 Results for initial systems

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.

Rank	System 1	System 2	System 3
1	R	R	R
2	N	R	R
3	N	R	R
4	N	R	R
5	N	R	R
6	N	R	R
7	N	R	R
8	N	R	N
9	N	N	R
10	N	R	R
AvP	1.00	0.99	0.98
P@10	0.10	0.90	0.90

Table 6: Query 2 Results for initial systems

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

Rank	System 1	System 2	System 3
1	R	R	R
2	N	R	R
3	R	R	R
4	R	R	R
5	N	R	R
6	N	R	R
7	N	R	R
8	R	R	N
9	N	N	N
10	N	N	N
AvP	0.73	1.00	1.00
P@10	0.40	0.80	0.70

Table 7: Query 3 Results for initial systems

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.

Rank	System 1	System 2	System 3
1	N	R	R
2	N	R	R
3	N	N	N
4	N	N	N
5	N	N	N
6	N	R	R
7	N	R	R
8	N	N	N
9	N	R	R
10	N	R	R
AvP	0	0.70	0.70
P@10	0.00	0.60	0.60

Table 8: Query 4 Results for initial systems

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.

Rank	System 1	System 2	System 3
1	N	R	R
2	N	R	R
3	N	R	R
4	N	R	R
5	N	R	R
6	N	R	R
7	N	R	R
8	N	R	R
9	N	R	R
10	N	R	R
AvP	0	1	1
P@10	0	1	1

Table 9: Query 5 Results for initial systems

After running all the queries in the different systems, we gathered the average precision and the precision at 10 for each query, presented in Table 10. From here we can conclude that systems 2 and 3 are the better ones and perform similarly. A change to the values of the weights in system 3 could possibly improve its performance.

Query	System 1		System 2		System 3	
	AvP	P@10	AvP	P@10	AvP	P@10
1	0.70	0.6	0.73	0.7	0.63	0.5
2	1	0.1	0.99	0.9	0.98	0.9
3	0.73	0.4	1	0.8	1	0.7
4	0	0	0.7	0.6	0.7	0.6
5	0	0	1	1	1	1
MAP	0.286		0.884		0.862	

Table 10: Initial System results for each query

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 SEARCH SYSTEM IMPROVEMENTS

In this section, it will be presented and discussed the several approaches taken to improve the search system previously developed. In the end, we present the results of this improved system and compare the results of the previously developed system, presented in section 12.

14.1 Items data improvement

A concern when choosing the dataset was the textual fields, whether there were enough fields and if they contained enough text and information to be later queried on. After the data preparation phase and its usage on the search system, we noticed there was room for improvement and to make the system more complete and mature.

We were satisfied with the textual information in the reviews' data, and it revealed to be enough to match the information needs, however, the items' data seemed to not be enough. Even though it is rich in numeric data, which allows us to sort and filter the queries in several ways, we had some difficulty when trying to conceive more complex information needs. To address this issue it was decided to scrape more data from the items Amazon product page, to retrieve more textual data. On the contrary to the section 2, where we used and adapted previous scrapping code, this time we developed our own simple Python script to scrape new sections of the products page, and obtained a new .csv file with each items' *description*, *about* and *more* section.

- **Description** - long text field, with some of the item's characteristics and catchy features
- **About** - main points of the product and some details that the buyer should be aware of
- **More** - multi-valued key-value field that contains the items features

After the scraping, the data was analyzed and we extracted the main features, from the *more* field, present in the majority of the products: wireless carrier, screen size, color, memory storage capacity, cellular technology, operating system. Afterward, the new data was refined and concatenated with the existing item data, proceeding with the pipeline to merge the data, to be later uploaded to Solr. The schema for these new fields can be consulted in Table 11.

Type	Field	Indexed
text_body	description	true
	about	true
title	more	true
string	wireless_carrier	true
	operating_system	true
	color	true
	screen_size	true
	memory_storage_capacity	true
	cellular_technology	true

Table 11: Items' additional fields

Now that we have new textual fields to query over the items' information, the weights of those fields had to be considered. The *title* continues to be the more relevant field, as it contains information about the brand, phone model, and some characteristics like storage capacity, cellular connectivity, etc., so naturally, it was the field that we gave the highest boost. Next was the *description*, due to its long text and information provided, we considered it to be the second most relevant field. The *about* and *more* fields were left with the default weight because some of the features of these fields are already mentioned in the previous ones. The boosts for the items fields can be seen in Table 12.

Field	Weight
title_item	2
description	1.5
about	1
more	1

Table 12: Item field's weights

With the new data, the search system will be able to satisfy more complex information needs, have more fields to filter the queries and provide more relevant information.

14.2 Information Retrieval tool improvements

Following the improvements of the data, it was time to improve the developed search system. Solr provides several tools that can be used to achieve a more robust system, and from the topics covered in the documentation [10] we examined: *Faceting*, *Spell Checking*, *Query Re-Ranking*, *Suggester*, *MoreLikeThis* and *Pagination of Results*. After analyzing the tools, considering the *quality of improvement/time to implement* relation and our intentions to developed a User Interface (later discussed in section 14.3), we decided to explore *Faceting*, *Pagination of Results*, and *Query Re-Ranking*.

14.2.1 Faceting. Faceting is the arrangement of search results into categories based on indexed terms. Solr's JSON Facet API[5] was used due to its usability and easier integration with requests from the User Interface to be developed.

Terms facets, which serve to bucket the domain based on the unique values in a field, were used to retrieve all the unique values on the specified fields and their respective count, which prove helpful to build a filter section. The parameters used from the API were the *field* parameter, which specifies the field name to facet over, and the *limit*, which limits the number of buckets returned. Regarding the *field* parameter, the fields chosen were *brand*, *wireless_carrier*, *operating_system*, *color*, *screen_size*, *memory_storage_capacity*, *cellular_technology*, for the items, and *country* and *verified*, for the reviews. The facet count, returned from Solr, for the *wireless_carrier* field can be seen in Figure 18, to serve as an example. Regarding the *limit* parameter, we used -1, rather than a positive integer, to return all the buckets available. There were no concerns with having an unlimited number of buckets, since the field with the most unique values is the *color*, and it "only" has 100 values.

Finally, we also made use of the Tagging and Excluding Filters functionality[12], in order to implement multi-select filter fields in the UI, able to display the values and their respective counts other than the selected filters, and that are updated every time a value for a specific filter is selected.

14.2.2 Pagination of Results. In most applications, the matching results of a query are displayed to the user in pages containing a fixed number of results. Since we want to develop a UI, it was clear that pagination must be implemented when showing the results to the user, to provide a better user experience.

Solr supports basic pagination[7] using the *start* and *rows* parameters:

- *start* lets us define the current "page", by calculating the initial offset of the documents to show. This can be achieved by this simple equation *start = page_number * rows*, treating the "first" page number as "0".
- *rows* specifies how many results we want to display per page. To avoid making the user scroll too much to see all the results of just one page, we opted to show 12 items, and 10 reviews, on their respective page.

Solr's documentation alerts to problems regarding index updates and "deep paging", but none of them were a concern. Since our data is static, there will not be any modifications while paginating, which might cause the same document to be returned on multiple pages, or documents to not be shown. Furthermore, our datasets are not large enough to cause memory issues on the last pages, and we do not need to recur to the *Cursors* functionality that Solr provides to fetch a large number of results.

14.2.3 Query Re-Ranking. Re-Ranking allows us to run a simple query for matching documents and then re-rank the top N documents using the scores from a different, more complex query. With the Learning to Rank (LTR) module that Solr provides, we can configure the features and the ranking models, and to better understand and implement this feature we followed not only Solr's documentation[6], but also a very comprehensive tutorial available on Github[1].

To begin we need to define the features that will be provided to the ranking models. For the items, we used the item's `rating`, `total ratings`, and `discount (\%)`, and for the reviews, we used the reviews's `number of helpful votes`, its `recency` (relative to the current date), and `length of the text`. Important to highlight that all those features were calculated using Solr's Function Query tools, and are normalized between 0 and 1, with a linear or logarithmic distribution, to provide a better data range for the models. A common feature used on all models was the `original score`, calculated by Solr, on the initial query.

Next, we define the models. Solr provides *linear*, *trees*, and *neural network* models, but since the last two require data to train the model, and for that large user input data on our system would need to be collected, we decided to implement the simpler LinearModel. For this model, it is only necessary to specify the features to be used (previously presented) and their weights, which can be consulted for both items and reviews model on Table 13 and 14 , respectively

Feature	Weight
originalScore	0.4
itemRating	0.1
itemTotalRatings	0.25
itemDiscount	0.25

Table 13: LTR items linear model

Feature	Weight
originalScore	0.4
reviewVotes	0.3
reviewRecency	0.1
reviewBodyLength	0.2

Table 14: LTR reviews linear model

In both models, the `original score` has the highest importance, as it calculates the score of the initial query input, and we want to make sure only results that match that query are shown. Regarding the items model, `total ratings` number and `discount` are the second most important features, with the `rating` being the least. This is because we want the rating to be justified, as an item with a 5-star rating, but few ratings, does not serve a lot. In regards to the reviews model, `original score` is again the most relevant feature for the same reasons. The number of `helpful votes` comes in second, as it proves the usefulness to the community, next is the `length` of the text, as more text should better describe the experience the buyer had, and lastly is the `recency` that we think should be handle carefully, as recent reviews, but for a recent phone, will only prove the first impressions, and not the full experience.

On the last note, the LTR plugin requires modifications to the *solrconfig.xml* file, in order to enable it and be able to upload the engineered features and models to Solr, but it is all very well described on both of the guides followed. The number of documents we decide to re-rank after the initial query is 200, which provides enough relevant results to the user on the initial "pages".

14.3 User interface

Besides improving the data in itself and the information retrieval tool, we also wanted to provide a better interface for the user to interact with the system. We opted for a modular approach, separating the frontend, with the user interface, from the backend, where the requests to Solr are made and the responses processed to provide the information to the frontend.

14.3.1 Frontend. The technologies chosen to build the frontend were React [19], an open-source and widely used JavaScript library, and MUI [21], Material to build UIs, a React UI library that enabled us to build our design system and develop an application faster. Our goal was to build something simple and intuitive so the user could access the main functionalities without much effort. To achieve this we implemented three main interfaces:

(1) *Item search interface* - figure 19
This interface allows the user to search through all cellphones in our data set, filtering the results according to the criteria of their choice.
The search bar is used for text queries. The section on the left has filters for all the relevant fields that search can be performed on, which are: rating, price, brand, carrier, operating system, color, screen size, storage capacity, and cellular connectivity. The first two filters need range-based input while the others use checkboxes to get the input from the user. When analyzing the checkboxes used as filters, we can notice that, for each checkbox, the number of possible results matching that filter appears in front. These values and the counts are retrieved using facets, as mentioned in section 14.2.1, and each time a value is selected, a request to Solr is made, to show only the facets available accordingly to the selections.
On top of the returned results section, there is a sort button to allow sorting by relevancy (which uses the items model defined in section 14.2.3), rating, price, and best discount. Both rating and price can be ascending or descending.
At the bottom of the results, there is a pagination component, making use of the pagination feature described in section 14.2.2.
(2) *Item page* - figure 20
This page shows an item and all its features in detail. At the bottom of the page, a link to the reviews search interface of that specific phone is present, as described in the next item.
(3) *Review search interface* - figures 21 and 22
Similar to the first interface described, item 1, multiple filters and a search bar for full-text search can be used to search through a large amount of data, this time cellphone's reviews data. The available filters are: rating, number of helpful votes, date, country, and verified. Both rating and helpful votes filter based on the provided range, date receives to date inputs and returns the results between those dates, and the country and verified fields have checkboxes that make use of the facet functionality (as mentioned in item 1).
The results can be sorted according to relevancy (with the reviews model defined in section 14.2.3), rating, helpful votes, and date, and are once again displayed using pagination.
There is also the possibility to search reviews for a single item, if the user comes from the item page, with this same interface.

14.3.2 Backend. The backend was developed with Node Express [4], a web application framework, full of HTTP utility methods and middleware, making the connection between the frontend and Solr extremely easy. The main reason to separate the backend from the frontend was to have a proxy like structure since Solr does not have CORS enabled by default and it is needed for our React app, and follow the MVC (Model-View-Controller) architecture. The frontend is responsible only for the views and sends to the backend controllers the input query and filters chosen by the user. Afterward, the backend processes that information and builds the request with the correct parameters for Solr's REST API. Then it processes the incoming Solr's JSON response, parses the item/review information and its facets, finally returning the information to the frontend, to be displayed to the user.

14.4 Results

After applying the previous improvements, we re-run our queries that represented our information needs from section 12, and the

new results for each query can be consulted in Tables 16, 17, 18, 19 and 20. The queries suffered minor changes to fit better with the updates, namely, for queries that return a set of reviews, the queried fields are now `title_review` and `body`, removing the `country` field. For queries that return a set of items, the queried fields are now `title_item`, `description`, `about` and `more`, removing `brand`. These fields were removed since they are now used with facets to filter the query.

Systems 1 and 2, apart from the changes above, remained the same. For system 3, we revised the weights used for the item's fields, as presented in table 12, and added query re-ranking with the LTR feature, as described in section 14.2.3. Only for query 5, which results are depicted in table 20, we did not use query re-ranking, because sorting is applied to the results. Since the re-ranking was made after the sort, we would not have any relevant documents returned.

Compared to the initial systems, we now have better results, where there's a clear distinction between each improved system, with system 3 providing the most relevant results in the initial returned documents, as seen in table 15 and the results tables in annex A. The initial system 1 and improved system 1 show different results because, due to the new scraping made, we also remade the list of relevant documents we expected would be returned, and in the improved systems, system 1 performed better, but still failed as expected in queries 4 and 5 because of the lack of filters.

For systems 2 and 3, we have a decrease and increase in performance respectively. We expected system 3 to return better results based on the new weights, query fields, and LTR, which it did. For system 2 we had worse results for queries 2 and 3, this may be because of the changes mentioned above or ultimately because of the inherent bias that exists in the list of relevant documents since this list was created by team members and not an unrelated authority to the project. Because of this, the list of relevant documents for the initial systems might have had more relevant documents that were in line with the results of the queries made for the initial system 2 than the improved system 2, resulting in better results for the initial system.

Query	**System 1**		**System 2**		**System 3**	
	AvP	P@10	AvP	P@10	AvP	P@10
1	0.79	0.4	0.77	0.6	1	0.8
2	1	0.2	0.71	0.4	0.96	0.7
3	0.39	0.3	0.46	0.4	0.8	0.6
4	0	0	0.98	0.8	1	0.9
5	0	0	1	1	1	1
MAP	0.436		0.784		0.952	

Table 15: Improved System results for each query

15 REVISIONS INTRODUCED

Following the search system implementation and its testing, some revisions were made, and are present in this section to clarify the corrections.

Regarding Solr's custom field types created, we ended up not making use of the *EdgeNGramFilter*, since no auto-complete feature was implemented. Therefore, that filter was removed from the title field, and only two custom field types, *title* and *text_body*, were used. This correction can be consulted in the Table 3. These custom types changes also affected the schema fields, which can be seen in Table 2.

The values for the *AvP* and *P@10* in section 12 were corrected, a table with the relevant/non-relevant docs for each query was added, and values for *MAP* metric were added to better compare the several systems.

16 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.

For the final section, overall improvements to the work developed in the previous sections are implemented. We start by describing the improvement to the data, with more scraping for items textual data, and the changes necessary to accommodate those additions in the dataset. Next, we discuss some of Solr's searching tools available, detail those that were explored, how they were implemented in our system, and their purpose to achieve our goals. An intuitive user interface was developed, the pages and functionalities implemented are described, and prints of the UI are present. Finally, we re-run the evaluation for the previously defined information needs and compare the new results with the initial system. We were able to evidence the impact of all the improvements made: the system became more complete with the added data, the searching tools implemented allow the implementation of a UI with better options,

and conclude that the system performs better with the new weights and LTR feature.

Regarding future work, we would like to implement more complex models for LTR, to provide a better *relevancy* filter, and explore more features like *Suggestions* and *Spell Checking* for a better user experience on the developed UI.

REFERENCES

[1] Michael A. Alcorn. 2020. From Zero to Learning to Rank in Apache Solr. https://github.com/airalcorn2/Solr-LTR Accessed: 2022-01-16.
[2] Elasticsearch B.V. 2021. Elasticsearch. https://www.elastic.co/ Accessed: 2021-11-10.
[3] Antonin Delpeuch. 2021. Openrefine User Manual. https://docs.openrefine.org/ Accessed: 2021-11-14.
[4] Express. 2021. Express. https://expressjs.com/ Accessed: 2022-01-16.
[5] Apache Software Foundation. 2021. JSON Facet API. https://solr.apache.org/guide/8_11/json-facet-api.html Accessed: 2022-01-16.
[6] Apache Software Foundation. 2021. Learning to Rank. https://solr.apache.org/guide/8_11/learning-to-rank.html Accessed: 2022-01-16.
[7] Apache Software Foundation. 2021. Pagination of Results. https://solr.apache.org/guide/8_11/pagination-of-results.html Accessed: 2022-01-16.
[8] Apache Software Foundation. 2021. Query Syntax and Parsing. https://solr.apache.org/guide/8_11/query-syntax-and-parsing.html Accessed: 2021-11-14.
[9] Apache Software Foundation. 2021. Query Syntax and Parsing. https://solr.apache.org/guide/8_11/the-extended-dismax-query-parser.html Accessed: 2021-11-14.
[10] Apache Software Foundation. 2021. Searching. https://solr.apache.org/guide/8_11/searching.html Accessed: 2022-01-16.
[11] Apache Software Foundation. 2021. Solr. https://solr.apache.org/ Accessed: 2021-11-10.
[12] Apache Software Foundation. 2021. Tagging and Excluding Filters. https://solr.apache.org/guide/8_11/faceting.html#tagging-and-excluding-filters Accessed: 2022-01-16.
[13] Information and Communications Technology. 2021. Information and Communications Technology. https://en.wikipedia.org/wiki/Information_and_communications_technology Accessed: 2021-12-10.
[14] Kaggle. [n. d.]. Find open datasets and Machine Learning Projects. https://www.kaggle.com/datasets Accessed: 2021-11-07.
[15] Griko Nibras. 2019. Amazon cell phones reviews. https://www.kaggle.com/grikomsn/amazon-cell-phones-reviews Accessed: 2021-11-14.
[16] Griko Nibras. 2020. Grikomsn/amazon-cell-phones-reviews: Scrape (UN)locked cell phone ratings and reviews on Amazon. https://github.com/grikomsn/amazon-cell-phones-reviews Accessed: 2021-11-14.
[17] Dragan Ivanović Nikola Luburić. 2016. Comparing Apache Solr and Elasticsearch search servers. http://www.eventiotic.com/eventiotic/files/Papers/URL/icist2016_54.pdf Accessed: 2021-11-14.
[18] The pandas development team. 2021. Pandas. https://pandas.pydata.org/ Accessed: 2022-01-14.
[19] Meta Platforms. 2022. React. https://reactjs.org/ Accessed: 2022-01-16.
[20] Berthier Ribeiro-Neto Ricardo Baeza-Yates. 2011. Modern Information Retrieval: The Concepts and Technology behind Search. http://www.mir2ed.org/ Accessed: 2021-11-14.
[21] Material-UI SAS. 2022. MUI. https://mui.com/pt/ Accessed: 2022-01-16.
[22] Robert Laubacher Thomas Malone, Daniela Rus. 2020. Artificial Intelligence and the Future of Work. https://workofthefuture.mit.edu/wp-content/uploads/2020/12/2020-Research-Brief-Malone-Rus-Laubacher2.pdf Accessed: 2022-02-17.
[23] Asaf Yigal. 2020. Solr vs. Elasticsearch: Who's The Leading Open Source Search Engine? https://logz.io/blog/solr-vs-elasticsearch/ Accessed: 2021-11-14.

A ANNEX

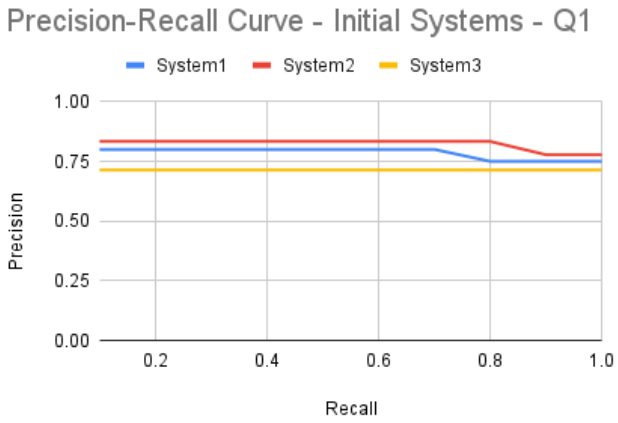

Figure 8: Query 1 Precision-Recall curve for Initial Systems

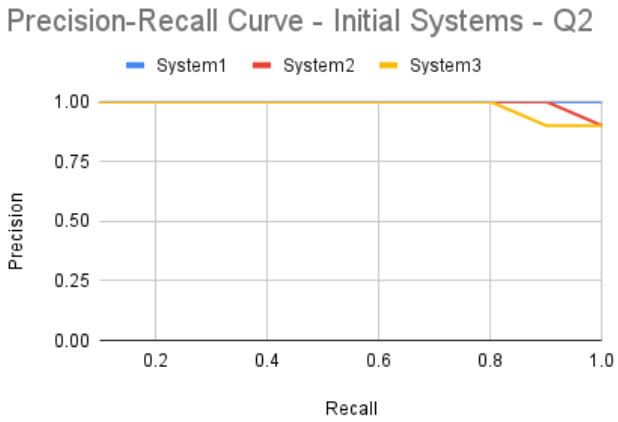

Figure 9: Query 2 Precision-Recall curve for Initial Systems

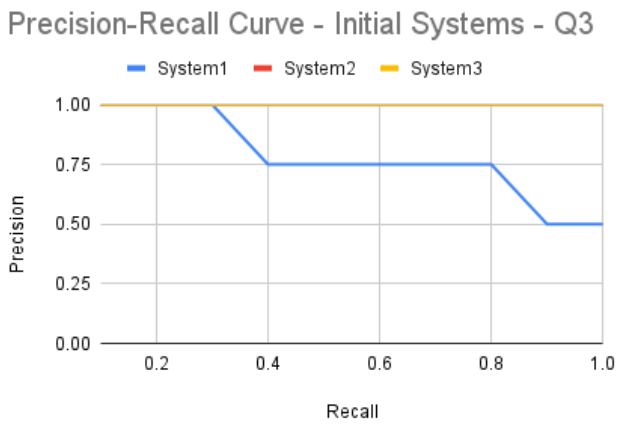

Figure 10: Query 3 Precision-Recall curve for Initial Systems

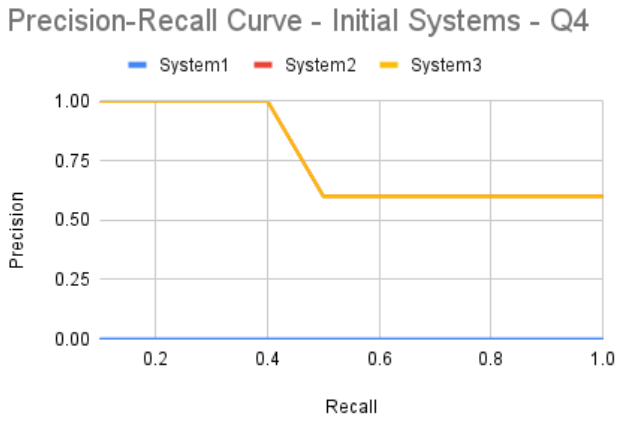

Figure 11: Query 4 Precision-Recall curve for Initial Systems

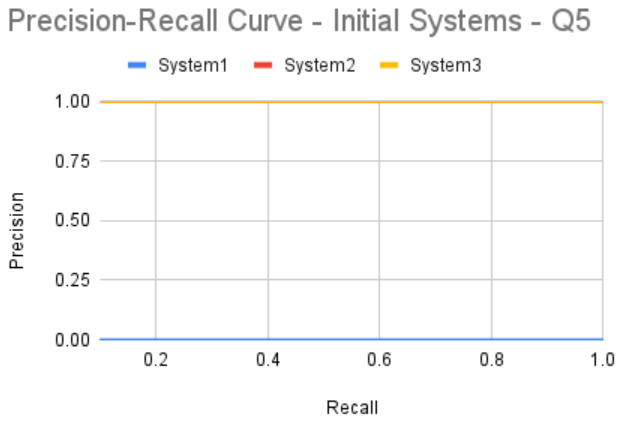

Figure 12: Query 5 Precision-Recall curve for Initial Systems

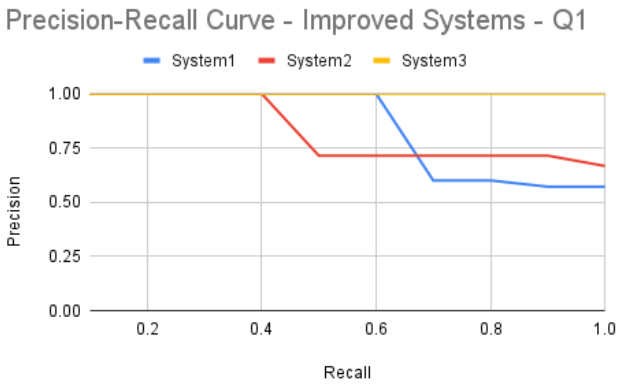

Figure 13: Query 1 Precision-Recall curve for Improved Systems

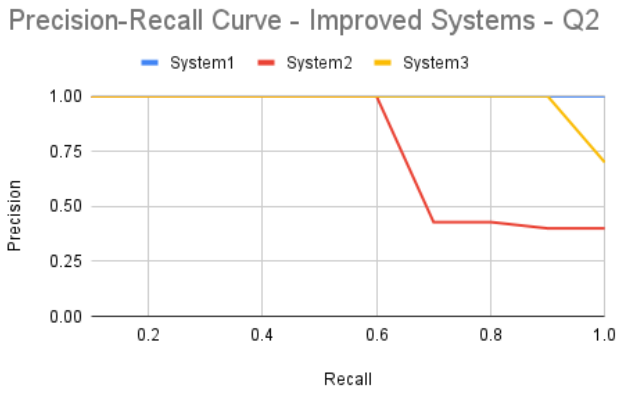

Figure 14: Query 2 Precision-Recall curve for Improved Systems

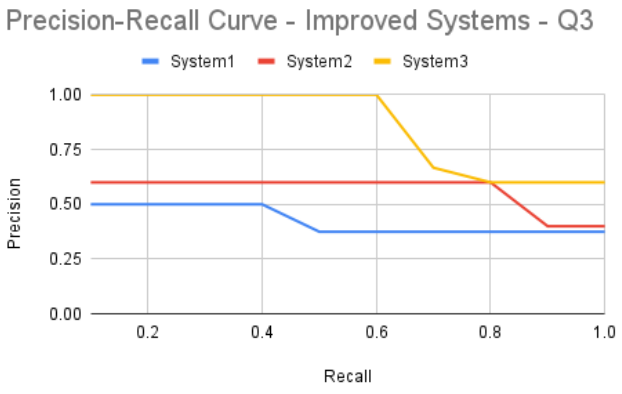

Figure 15: Query 3 Precision-Recall curve for Improved Systems

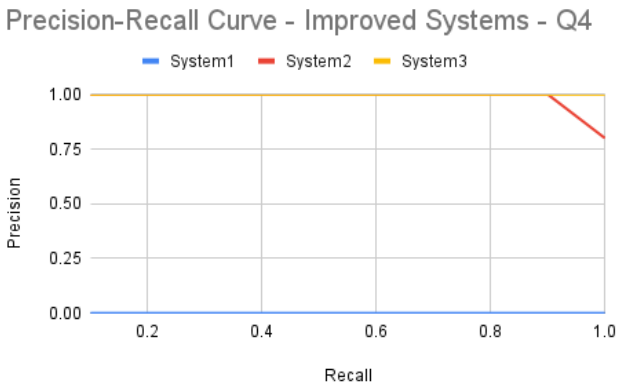

Figure 16: Query 4 Precision-Recall curve for Improved Systems

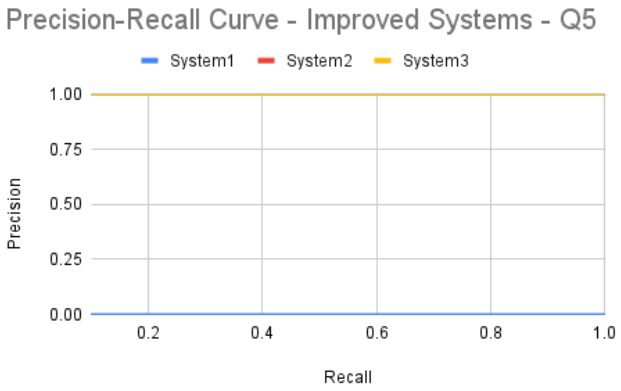

Figure 17: Query 5 Precision-Recall curve for Improved Systems


```
"facet_counts":{
  "facet_queries":{},
  "facet_fields":{
    "wireless_carrier":[
      "Unlocked",705,
      "T-Mobile",226,
      "AT&T",171,
      "Verizon",166,
      "Sprint",34,
      "Straight Talk",20,
      "Tracfone",17,
      "3",12,
      "Cricket",12,
      "Boost Mobile",11,
      "Verizon Wireless",9,
      "Net10",7,
      "Simple Mobile",7,
      "WhatsApp SIM",7,
      "Virgin Mobile",6,
      "MVNO",4,
      "Telcel",4,
      "Blue",3,
      "Go Mobile",3,
      "Movistar",3,
      "Deutsche Telekom",2,
      "Everything Everywhere",2,
      "Nextel",2,
      "Ting",2,
      "VoiceStream Wireless Provider Type",2,
      "Alltel Cellular",1,
      "China Mobile",1,
      "Embratel",1,
      "Etisalat",1,
      "Metro by T-Mobile",1,
      "O2",1,
      "Orange",1,
      "TIM",1,
      "Total Wireless",1,
      "Uscellular",1]},
  "facet_ranges":{},
  "facet_intervals":{},
  "facet_heatmaps":{}}}
```


Figure 18: Wireless carrier facet counts

Rank	System 1	System 2	System 3
1	R	R	R
2	R	R	R
3	N	N	R
4	N	N	R
5	R	R	R
6	N	R	R
7	R	R	R
8	N	N	R
9	N	R	N
10	N	N	N
AvP	0.79	0.77	1
P@10	0.4	0.60	0.8

Table 16: Query 1 Results for improved systems

Rank	System 1	System 2	System 3
1	R	R	R
2	R	R	R
3	N	N	R
4	N	N	R
5	N	N	R
6	N	N	R
7	N	R	N
8	N	N	N
9	N	N	N
10	N	R	R
AvP	1	0.71	0.96
P@10	0.2	0.4	0.7

Table 17: Query 2 Results for improved systems

Rank	System 1	System 2	System 3
1	N	N	R
2	R	N	R
3	N	R	R
4	N	R	N
5	N	R	N
6	N	N	R
7	R	N	N
8	R	N	N
9	N	N	R
10	N	R	R
AvP	0.39	0.46	0.8
P@10	0.3	0.40	0.60

Table 18: Query 3 Results for improved systems

Rank	System 1	System 2	System 3
1	N	R	R
2	N	R	R
3	N	R	R
4	N	R	R
5	N	R	R
6	N	R	R
7	N	R	R
8	N	N	R
9	N	N	R
10	N	R	N
AvP	0	0.98	1
P@10	0	0.8	0.9

Table 19: Query 4 Results for improved systems

Rank	System 1	System 2	System 3
1	N	R	R
2	N	R	R
3	N	R	R
4	N	R	R
5	N	R	R
6	N	R	R
7	N	R	R
8	N	R	R
9	N	R	R
10	N	R	R
AvP	0	1	1
P@10	0	1	1

Table 20: Query 5 Results for improved systems

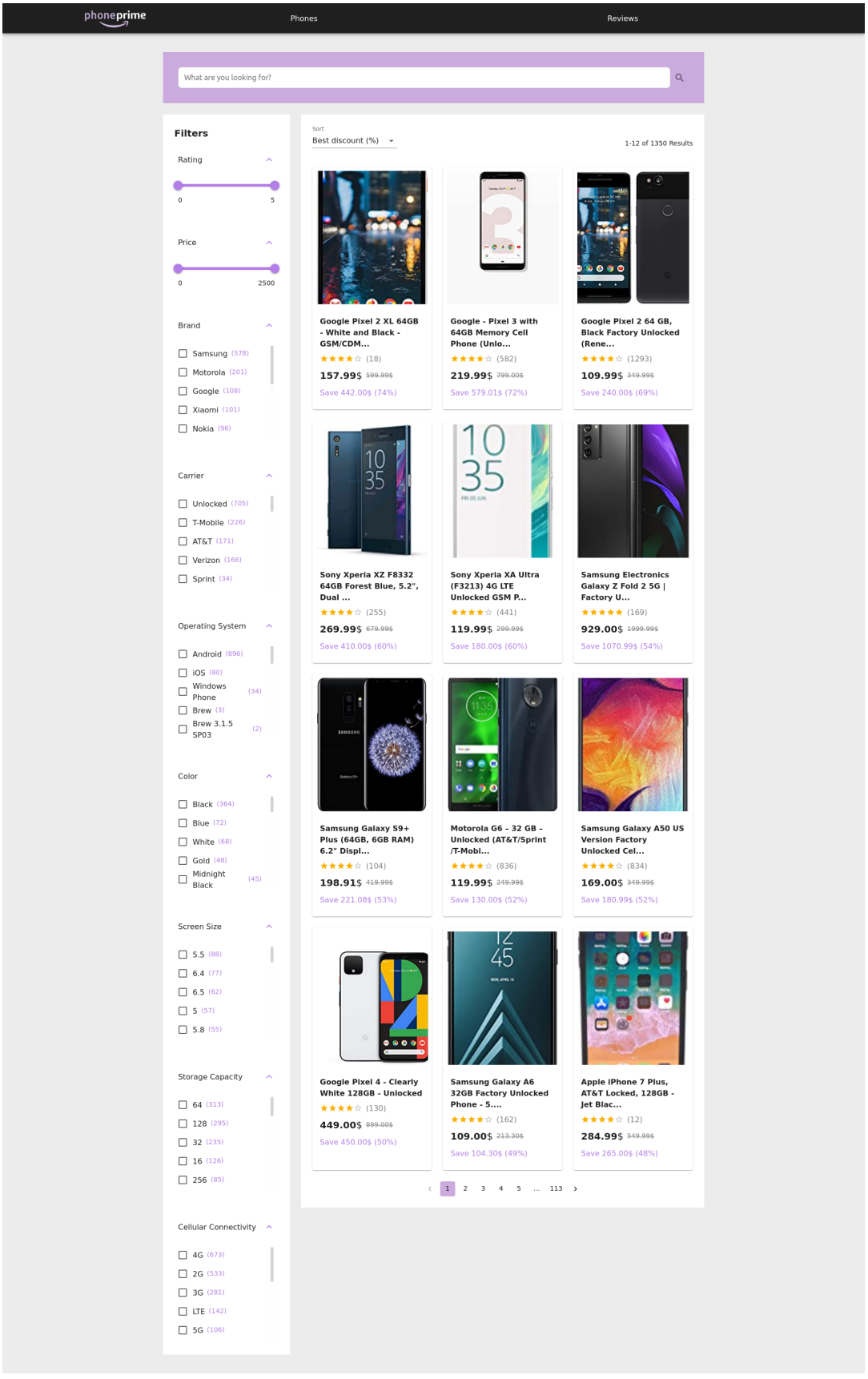

Figure 19: Search items page

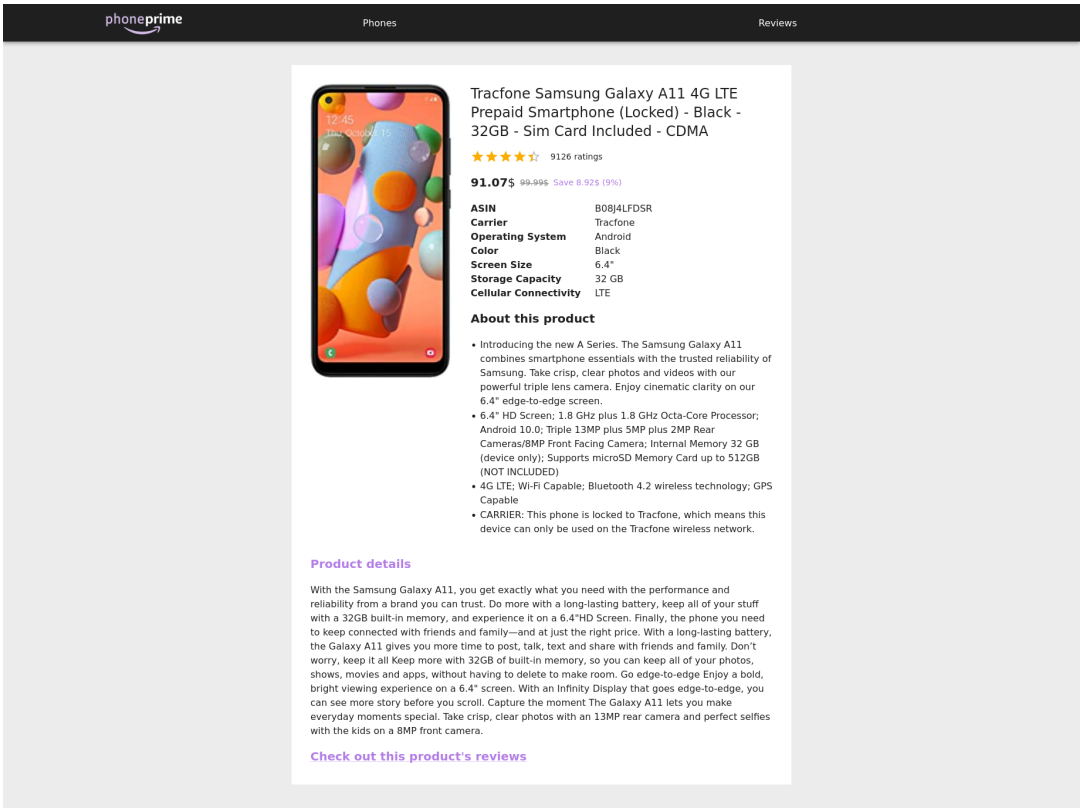

Figure 20: Item page

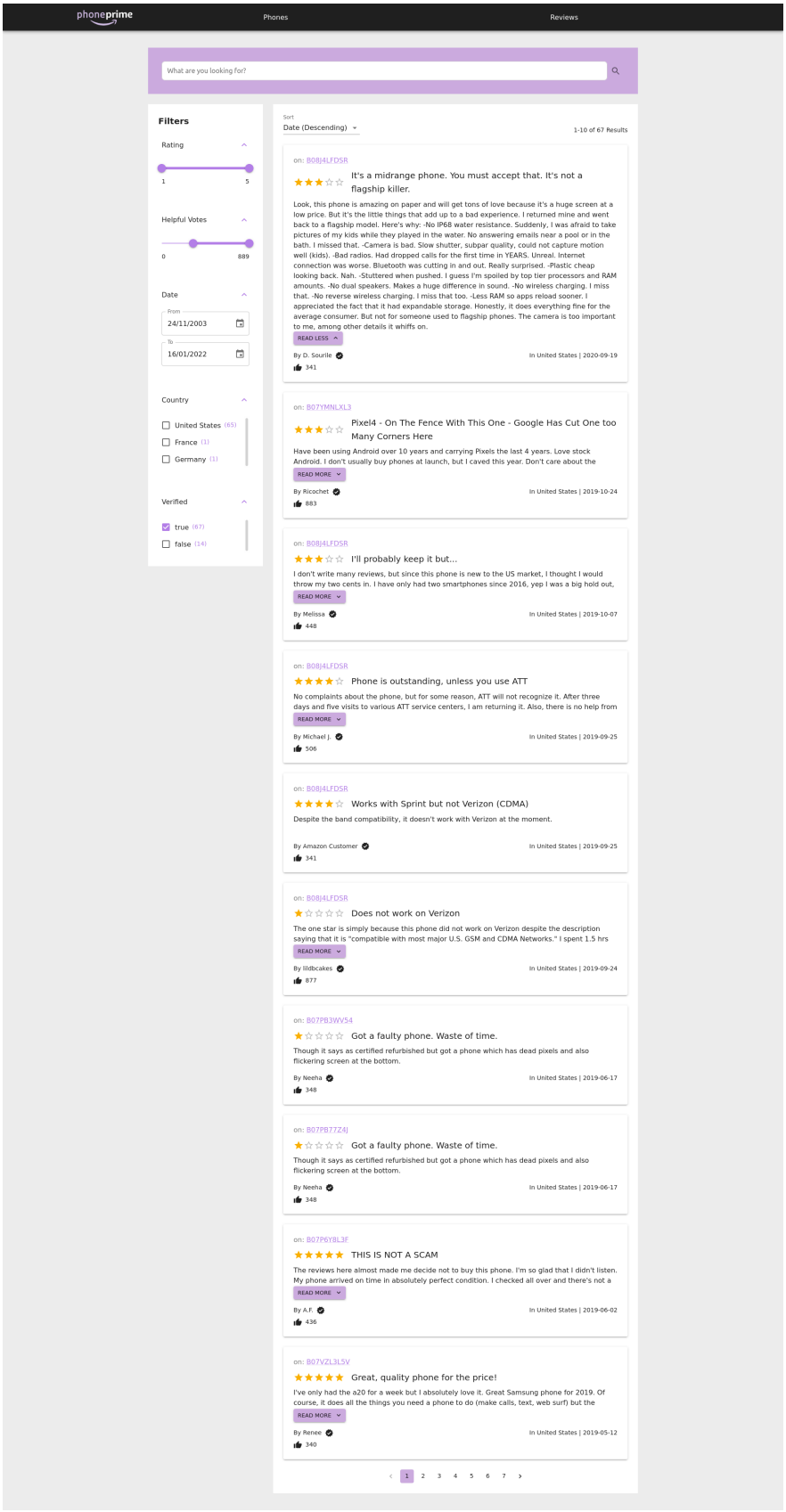

Figure 21: Search reviews page

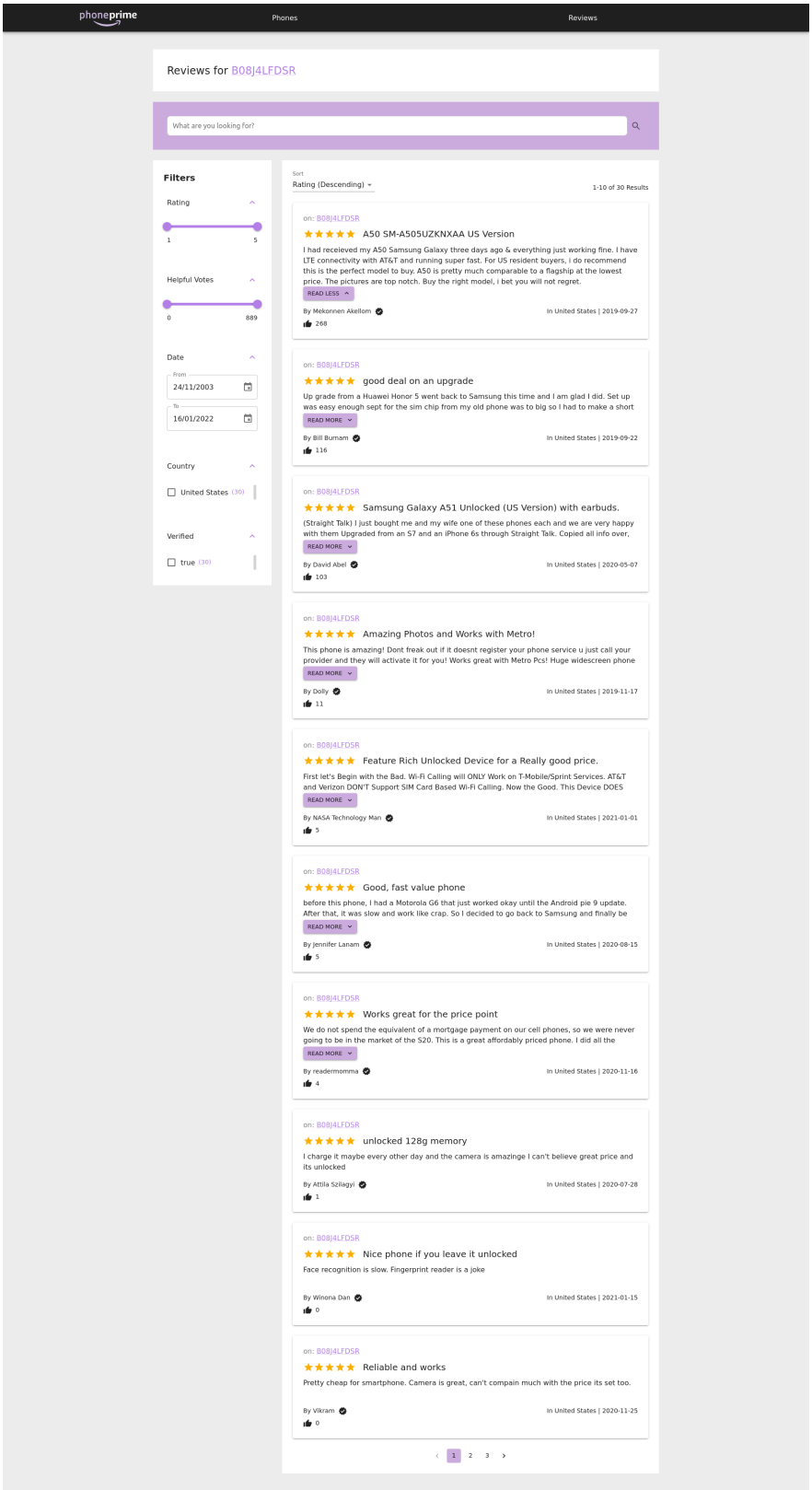

Figure 22: Search reviews from a certain item page