HackerSearch: an Information Retrieval system

Ana Barros
feup
up201806593@edu.fe.up.pt

João Costa
feup
up201806560@edu.fe.up.pt

João Martins
feup
up201806436@edu.fe.up.pt

ABSTRACT

This document describes the development of an Information Retrieval system for the course unit of *Information Processing and Retrieval* taken in 2021 at *FEUP*. This system takes textual data, corresponding to the year 2018, collected from the "HackerNews" link aggregator and corresponding linked URLs. The following sections explain in detail the processes of collecting, processing and analyzing the collected data, alongside with describing the developed information search system. More specifically, this includes its indexing, query parsing and results evaluation.

KEYWORDS

datasets, information retrieval, search engine, HackerNews, Y Combinator, HackerSearch, data analysis, indexing, Solr, JSON

1 INTRODUCTION

"HackerNews" is a link aggregator, mainly used to share news and products. It focuses on tech-related topics, such as computer science, and entrepreneurship. "HackerNews" is backed by "Y Combinator" foundation, which funds early startups biannually.
The community in this website usually interacts by sharing links to articles (blogs, news websites, and product landing pages) and then, voting and commenting on them. What sets this community apart is the vast technical knowledge shown in most comments, giving them a high readability value which sometimes surpasses the value of the post discussed.

This article aims to describe the development of "HackerSearch". This platform will serve as an intuitive and powerful search engine for "HackerNews" posts/content.
There are 4 main sections in this report. Each of these sections describes different tasks completed in order to achieve the final goal for this project. The first section, Data Collection, describes the process of retrieving the data. The second section, Data Processing, describes the filtering and preparation of the gathered data. The final section, Data Analysis, lists the process of exploring the data with statistics, characterizing the final datasets and their properties, the conceptual model as well as the follow-up information needs in the data domain.

2 DATA COLLECTION

"HackerNews" provides an *official public API*[5] to fetch website's data. This data is output in **JSON format**, and the **curl** command line utility is used to fetch it.

2.1 Posts/Stories

There are three types of posts in "HackerNews": **story**, **job**, and **poll**. There was a decision that our dataset would only contain stories, as those comprise around 99.99% of posts on the website. From now on, all references to post/story are interchangeable.

The API[5] does not provide a way to perform bulk downloads. In order to streamline the process of fetching data, a starting dataset[6] was used. This dataset contained the raw data of stories collected using the API[5]. It is fetched by cloning the GitHub repository using the git command-line utility. The data comes divided into several JSON files containing a single list with multiple objects representing stories. This was done to bypass GitHub's limit on file size. Upon downloading, these lists merge into a single one using the **jq**[2] command-line utility, so the stories can be filtered more easily and stored in the **stories.json** file.
Given the large volume of data available, it was decided to only deal with information related to stories posted in 2018. This is equivalent to around 50000 stories.

2.2 Comments

The dataset mentioned above only contains data about stories. To complement this, the API[5] was used to download the data of the top two comments of each story. This translates to around 100000 comments.
It was possible to download the data by using curl with persistent connections, thus reducing the time expended establishing connections with the API[5], and multiple sub-processes, to parallelize work. In order to easily parallelize the work and make use of persistent connections in curl, it was necessary to use a file containing URLs as input. Using **jq**[2] (to select the IDs of the top comments) and **awk** (to build the URLs using the IDs that were being piped) helped to build the file. Afterwards, it was split info multiple files (one for each sub-process) using GNU's split, so each sub-process could take care of their part of the URLs.

Jq[2] was used to concatenate all the downloaded comment data into a single JSON list stored in the **comments.json** file.

2.3 URL content

As stories without a link don't really do well upon on "HackerNews", most stories (90%) do not have textual content (apart from the title). Instead, the website incentivizes users to share URLs to the content they want to discuss/share. This limits the searchability of stories.

To work around this, the group developed a small program using Mozilla's Readability tool[4] (part of the Firefox browser) to fetch the main content of the URLs linked by the stories in text form. This allows us to perform general web scraping in many websites with different layouts/content, which wouldn't be feasible with a specialized 'web spider'.
This code uses **node-fetch**[7] to fetch the target website's data and **jsdom**[3] to feed this data to the readability tool[4].

Jq[2] selected the stories containing a URL, and GNU's split divided the information of these stories into multiple files (each used by a respective sub-process). A JSON object stored the website's

2021.

extracted textual content alongside its post ID. Each sub-process stores the content it is responsible for in a file. The JSON objects in these files join into a single JSON list using **jq**[2] and stored into the **html_content.json** file.

3 DATA PROCESSING

During collection and before analysis, the data went through a pipeline, illustrated by *Figure 1*, where it was transformed and filtered.

3.1 Initial Processing

The processing and filtering processes described in this subsection happened during the data collection process.

In order to reduce the number of stories (while increasing their relevance), they were filtered out based on the following criteria:

- Stories deleted by their owner.
- Dead stories (flagged by the users): If a story has enough reports, it is deleted.
- Stories that 'failed': A story 'fails' when it scores lower than 5 points, thus not being able to reach the front page of the website. Every story starts with 1 point and gains 1 extra point for each vote it receives.

These filtering steps were all performed by a single **jq**[2] script that encompasses all conditions mentioned above.

Comments suffered a similar filtering process (using **jq**[2]): Removing dead and deleted comments. It should be noted that some less popular stories have less than 2 comments in the final dataset, because users might not have engaged enough in their discussion or all their comments don't appear in the final dataset out for the reasons above.

The content of the URLs of the stories isn't always useful as textual data: some URLs link to binary data (images, PDF files, videos, etc…) and others might lead to web pages that our tool isn't able to extract information from (e.g.: YouTube's website). In these cases where the URL doesn't lead to any useful information and the story does not have textual content, it was decided to 'drop' this from the dataset.
When the URL yields useful information (most cases), the yielded data passes through a compression process to remove repeated blank spaces and newlines, and filtered to remove non-printable characters. This is done using a series of piped **sed** and **GNU's tr** commands. These processes are important because the website data is frequently unstructured and large.

3.2 Post-Processing

After gathering all the data, it was necessary to perform some final cleanups and save it in a more fitting format: at this point, the data is in three JSON files which needed to be unified.

The 3 JSON files containing the data turned into python's **pandas[9] DataFrames** to be worked on. The steps described below use the **pandas[9] and NumPy[8]** modules in python3.

All the stories in the dataset had a **type** field with the value **story**. This happened because the dataset contained only posts of that type (as mentioned in the Posts/Stories Data collection subsection. This field turned into a new one of the same name that divides the stories into the dataset in four categories, recognized by the "HackerNews" website, that can be filtered in their tabs. These categories are:

- **LaunchHN** - these are stories about new "Y Combinator" backed start-ups. This type of stories start their title with *"Launch HN:"*.
- **AskHN** - these are questions to the website's community. This type of story can't have a URL, and their title almost always starts with *"Ask HN:"*. Every story that doesn't have a URL falls in this category.
- **ShowHN** - these are stories that usually want to share a product landing/main page. This category is very similar to the **Normal** category discussed below, being only distinguished by their title starting with *"Show HN:"* and thus being shown in their filtered page.
- **Normal** - this is the category where most stories fall under. These always have a URL and rarely contain a textual description. Most time these link to news and scientific articles, or blog posts.

The percentage distribution of each of these categories in the final dataset will be discussed below in the Data Analysis section.

Both the story data and the comments contain a **'kids'** field. This field is a list of the direct descendants of the respective story/comment. In the case of the stories, their data also contains a **descendants** field that represents the number of child comments in total (across all nesting levels).
The group decided that the **'kids'** field has no value for the final dataset and, in the case of the comments, should be used to derive a **descendants** field before being dropped. This solution is not optimal as it doesn't take into account the number of child comments across all nesting levels, but only the top-level. It was still decided to use this solution, as it was infeasible to fetch the data of every comment (over 2.5 million) of the chosen stories in order to count the number of child comments.

Some data acquired from the "HackerNews" API[5], namely the **text** fields of both the stories and the comments, contained HTML special chars escapes. These were 'unescaped' so they could be stored cleanly.

4 CONCEPTUAL MODEL

After the gathering and passing through the data processing pipeline, the final data on the **pandas' DataFrames** was exported to a sqlite3 [11] database (described in this section), using the **sqlalchemy python module**.

For the data domain, the team created a conceptual model (*Figure 2*) in order to understand the data organization. There are four tables: *Type*, *Story, Comment*, and *URL*. The main attributes of the *Story* table are:

- story_by: the username of the user who posted the story.
- story_descendants: the number of comments of the story.
- story_score: the score of the post (a sum of its upvotes).
- story_time: the (UNIX) timestamp of the story posting.
- story_title: the title of the story.
- story_text: the textual content of the story.
- story_type: the type of the story (foreign key to the *Type* table).

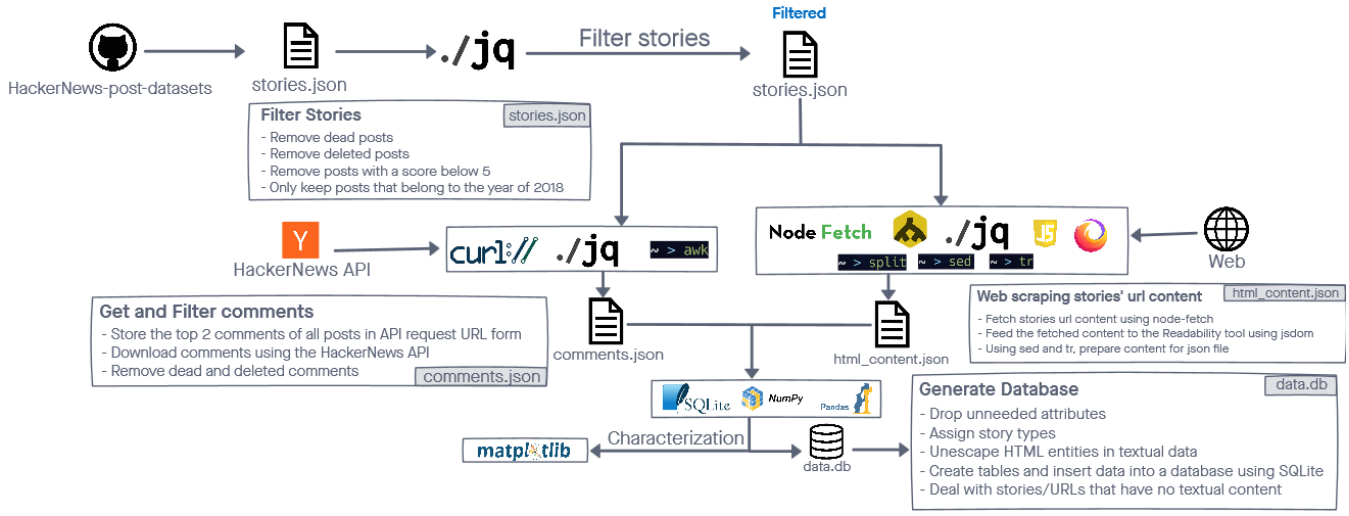

Figure 1: Dataflow Diagram

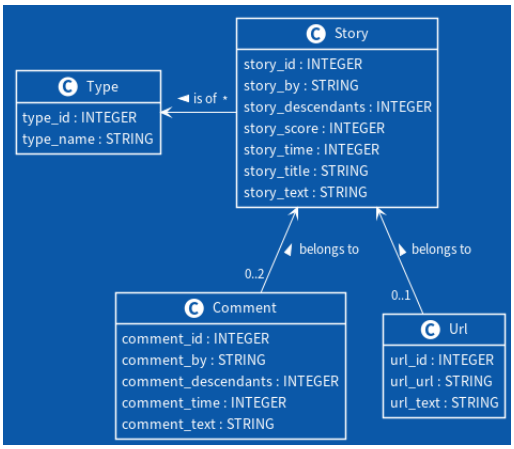

Figure 2: Conceptual Model

The *Type* class exists in order to represent the type of story. All stories have a type. This class contains only two attributes:

- type_id: the integer that identifies a story type.
- type_name: the name of the type.

The *Url* class represents the textual data obtained from URLs that are linked inside a story. Not every story needs to have a URL. However, a story cannot have more than one URL. The main attributes of this table are:

- url_url: the URL of the resource.
- url_text: the filtered textual contents of the web resource the URL links to.

Finally, the *Comment* table represents a comment of a story. Each comment only belongs to one story. Each story might have one, two, or no comments. These are its main attributes:

- comment_by: the user who posted the comment.
- comment_descendants: the number of replies of a comment (as described in the Post Processing subsection.
- comment_time: the (UNIX) timestamp of when comment was posted.
- comment_text: the content of the parent story.

5 DATA CHARACTERIZATION/ANALYSIS

This section describes the data characterization using graphics, plots, and textual descriptions.

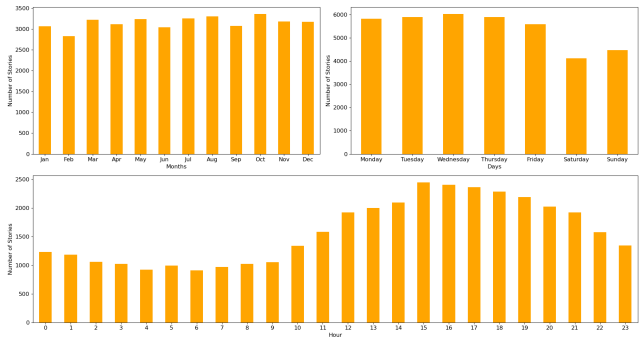

Figure 3: Number of stories per time

By analyzing *Figure 3*, it can be concluded that there is a higher frequency of story posts during the afternoon. Throughout the week, posts are more frequent during the weekdays as in contrast with the weekend. Finally, there is not a particular time of the year when the number of stories posted differentiates from the rest.

There is a unbalanced distribution of posts based on type, as evidenced by *Figure 4*. As described previously, there are four categories of stories: “Normal”, “AskHN”, “ShowHN”, and “LaunchHN”. The reality is that 85, 5% of all posts (which translates to almost 33000 posts) are classified as “Normal” posts while there are less than 100 posts classified as “LaunchHN”. This is expected considering that most posts on “HackerNews” intend on sharing a news article or blog post (most common category) and “LaunchHN” posts are a special category for new Y Combinator startups (less common category). The other two categories fall somewhere in between.

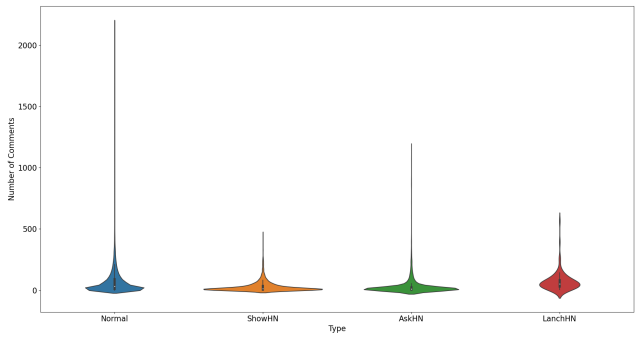

Figure 4: Number of comments per type

'Normal" stories have a higher ceiling for number of comments (followed by "AskHN"), but the median number of comments is roughly the same for all story types. This is surprising considering that stories that do not contain a URL (the case of "AskHN") are penalized in terms of exposition, requiring more engagement from users in order to reach the front page (succeeding).

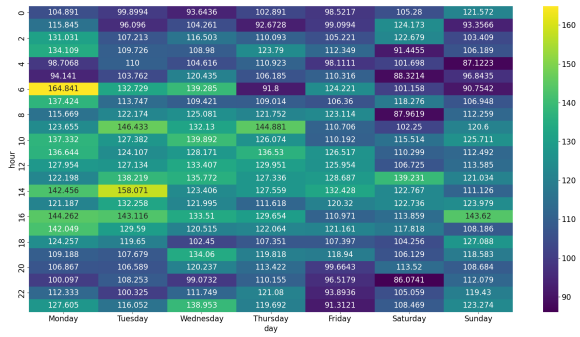

Figure 5: Average score in reference to time and day of the week

By analyzing *Figure 5*, a heatmap that relates the score of a story with the day and hour of when it was posted, it can be concluded that stories posted at the beginning of the week (Monday, Tuesday, Wednesday) and in the middle of the day (10AM-4PM), achieve higher scores. The cells with a higher average happen in stories posted between 6AM and 2PM on a Monday and from 9AM to 6PM on a Tuesday. The high discrepancy between the cells with the highest score values and its neighboring cells can be explained by the fact that most outliers occur within these timeframes. For example, one of the highest rated stories (2776 up-votes) was posted on Wednesday at 11PM, which has an average score of 138, while its neighboring cells have a score of 111 and 100.

It can be concluded by analyzing *Figure 6* that the average score of a story can be related to the number of comments it has (user engagement). The trend is that a higher number of comments leads to a higher score. Stories with the highest scores appear between the interval of 750 and 1000 number of comments.

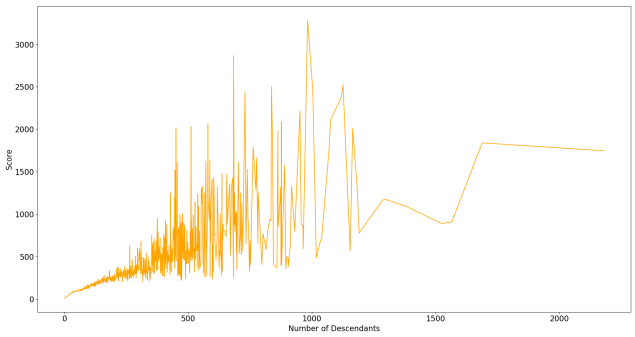

Figure 6: Score of story per number of comments

6 INDEXING PROCESS

The indexing process consists of the preparation of the documents, the creation of a collection and the definition of the schema. For this task, the chosen information retrieval tool was "Solr"[10].

6.1 Documents

As previously mentioned, the group created a SQLite[11] database containing the data after all the processing. However, in order to import the data to the information retrieval tool chosen, the database transformed into a single JSON file (documents). This meant performing a join of all the tables shown in the conceptual model (*Figure2*). The relation between Story and Comment is one-to-many. With this, preforming a join of these tables would result in duplicated entries in the search results (one for each comment on a relevant story). As such, the comments of each story are sub-documents of the story document.
The group converted the UNIX timestamps in the database to time strings, so they could be used in Solr's range queries.

A document contains all information about the story it refers to (title, text, URL, etc…), the scrapped content of the website it links to (if any), and a sub-document for the story's comments. The most import field of the comment's document is its content (the comment's text). These documents, compose the *hackersearch* collection.

6.2 Schema definition: indexed fields

The structure of the schema can be analyzed in Table 1. Each entry represents a field of the document: one cell indicating the type of the field, and another whether it is indexed or not. The latter was a decision made based on the search tasks defined on the Search Tasks section: fields that aren't indexed, can't be searched on (this includes filtering and faceting tasks).

The fields of the comment's sub-document are present on the schema table and prefixed by *comments.*. The two last fields are Solr's copy-fields.

6.3 Schema definition: copy-fields

Table 2 contains the copy-fields' definition: the *source* and the *destination* of the copy operation.

The *search* field is the default searching field: when no field to search on is defined, all fields copied to the *search* field will be

Field	Type	Indexed
story_id	int	false
story_author	singleToken	false
story_descendants	int	true
story_score	int	true
story_time	date	true
story_title	text	true
story_text	text	true
story_type	category	true
url	url	true
url_text	text	true
comments.comment_id	int	false
comments.comment_author	singleToken	false
comments.comment_descendants	int	false
comments.comment_time	date	false
comments.comment_text	text	true
newssite_filter	newsfilter	true
search	text	true

Table 1: Document Fields

Source	Dest
url	newssite_filter
story_type	search
story_title	search
story_text	search
url	search
url_text	search
comments.comment_text	search

Table 2: Copy Fields

searched on. The *newssite_filter* fields filters news stories from non-news stories.

6.4 Schema Definition: analyzers, tokenizers, and filters

In Solr, each field has a field-type defining how fields are to be processed for indexing and querying purposes using tokenizers and filters:

- **title** – The type of the stories' titles. Very similar to the **text** type.
- **text** – The type of all the text fields (story text, URL content, comment text, and search).
- **url** – The type of the stories' URLs. Allows the user to focus on their search on specific domains.
- **newsfilter** – The type of the *newssite_filter* field. Allows the user to tailor their results towards news or non-news content.
- **category** – The type of the stories' categories.
- **singleToken** – The type of text fields that aren't indexed (e.g.: a story's author).
- **date** – The type of fields representing points in time. Useful for sorting/filtering for recent stories.
- **int** – The type of integer fields (e.g.: the number of descendants of a story).

6.4.1 Field-type: text. Stream treatment with this field-type is the same during index and query time. Before being passed to the tokenizer, fields of these type pass through a **charFilter**. This filter removes HTML tags from the textual content of the field, for example:
`<a href=https://solr.apache.org>Solr</a>` becomes `Solr`. This is important because, although the content obtained using the "HackerNews API" was stripped of its HTML special entities, it still contains HTML tags.

Even after passing through the **charFilter**, these contents can still contain URLs. In order to deal with this, the tokenizer used is the *UAX29URLEmailTokenizerFactory*. This tokenizer identifies special tokens, like URLs and email addresses, and leaves each of them as a single token. Besides the special treatment described above, this tokenizer splits tokens on whitespace, punctuation, and hyphens.

The filtering pipeline starts after the tokenization process of the tokens:

- The filter converts non ASCII characters, to their ASCII equivalents, if one exists. Since the language of the website is exclusively English, this helps deal with typos/rare special terms. This also discards the originals.
- Characters change to lower case.
- There is a synonym filter to increase the detection of related topic/words. As of now, this filter is non-exhaustive and only used as a proof of concept of what is possible. It appends occurrences of the phrase **version control** with names of version control systems, like **git**, **svn**, and names of products/companies, like **GitHub** and **GitLab**.
- English words reduce to their root form and possessive cases disappear.
- Removal of stop words (e.g. the, a, an, etc…).

6.4.2 Field-type: title. Stream treatment with this field-type is the same during index and query time. This field-type is very similar to the **text field-type**. The main differences are the tokenizer used and some filters. Since titles in "HackerNews" can't contain URLs or email addresses, the used tokenizer splits tokens on whitespace, punctuation, and **@** characters. In contrast with the **text field-type**, stop words in the title of a story are not filtered. Also, the original tokens are kept when converted to ASCII: stylized product names often appear in the title.

6.4.3 Field-type: url. When searching on the *url* field, the focus is on the domain of the URL. During index time, the tokenizer creates one token for each (sub-)domain of the URL. During query time, the tokenizer splits the queries as standard (on whitespace, punctuation, etc…), and the filters convert all characters to lowercase, so it can match the (sub-)domains, which are all lower-case as well.

6.4.4 Field-type: newsfilter. During index time, the tokenizer of this field-type extracts the domain (not the subdomains) of a URL. Afterwards, the filters keep only the domains present on a news website list. The tokens output from this process can only be **empty** (not a news website) or the word **news** (a news website).

During query time, the tokenizer splits the stream as standard (on whitespace, punctuation, etc…). Afterwards, only the tokens **new** and **news** will match with the URLs of news-related websites.

This field allows for a dynamic list of news content domains (news networks, journalism website, etc…) and for users to know how many results in their query are news-related. Furthermore, this field allows the user to apply a filter to their search, so only news websites or non-news websites show up.

6.4.5 Field-type: category. During index time, the categories stay as a single token and change to lower-case. During query time, it splits the stream as standard and converts it to lower-case as well. This allows the user to filter results by their categories, and the system to count the number of results that fall on each category.

6.4.6 Field-type: other. The last three field types are just *wrappers* around built-in Solr[10] field types, without any defined tokenizers or filters:

- **singleToken** – solr.StrField – text fields that aren't searched on.
- **date** – solr.TrieDateField – fields that represent points in time.
- **int** – solr.IntPointField – fields that represent integer numbers.

7 INFORMATION RETRIEVAL

7.1 Search Tasks

Users of this information retrieval tool are able to search for items fitting a description. For example, when searching for "cli work with JSON", the tool would retrieve results talking about/showcasing jq[2] (the tool used to process the JSON files that power "HackerSearch"). An option to separate news related content from technical blog posts could also improve the system., e.g., searching for "github" leading to version control system usage/guides instead of update news and policies changes of GitHub. This could be done by keeping a list of news/journalism websites. In addition, ranking based on popularity, how active the story is, and time is desirable. If possible, the meaning of words should be associated with the context.

7.2 Query demo

Query intention. As a user, I want to see the most recent news stories about Hong Kong. The ones that interest me are the highest rated ones. I don't want to see anything related to China or its relation with Hong Kong.

Query. `"Hong Kong" +(newssite_filter:news) -China`

Sort field. `story_time desc, story_score desc`

Number of results found. 27

8 EVALUATION

This section describes four queries that display the capabilities of the designed system, and evaluates them. Three different systems (or more/less in some cases) run each query:

- **schemaless** – system with only the **search** field described in the schema section above, without any filters.

Field	Weight
story_type	10
story_title	3
story_text	3
url	2
url_text	1
comments.comment_text	1

Table 3: Field weight

- **schema** – system as described in the schema section.
- **weighted schema** – the same as schema, but the search fields have weights as described on the weight table of the query demo.

8.1 Precision Metrics

The followed evaluation process consists of calculating, for each query, a set of precision metrics. To prevent result redundancy, only the systems deemed relevant were chosen, for example, news filtering isn't relevant/possible on the **schemaless** system as it doesn't have the required fields.

For each query and correspondent relevant systems, each metric uses a universe composed of the **top 20** results obtained by the query/system pair. Afterwards, the set of relevant and irrelevant documents are within the top 10 results.

The chosen metrics to describe the query results are as follows:

- **Recall@N** - Ratio between the number of retrieved relevant documents and the total number of documents in the *universe* for the N first retrieved documents.
- **Precision@N** - Ratio between the number of retrieved relevant documents and the total number of retrieved documents for the N first retrieved documents.
- **Average Precision (AP)** - The mean of the precision values calculated for the first N documents).
- **F$_\beta$Score** - Calculated with precision (P), recall (R), and β (controls the amount in which the recall is more important than the precision).

$$(1+\beta^2)\cdot\frac{P\cdot R}{(\beta^2\cdot P)+R}$$

This is useful to verify the balance between the number of relevant documents missed and the number of retrieved irrelevant documents [12].

For the problem at hand, the group decided that a good baseline is to consider the precision as twice as important as the recall, i.e., a β value of 0.5. This is justified by the fact that, in the context of "HackerNews"[1], a user wants to find the first relevant matches with very few irrelevant documents, instead of all relevant matches with some irrelevant documents. Still, the recall metric is valuable, as the system must be able to retrieve an ample part of all relevant documents to provide the user with diverse results.

8.2 Queries

8.2.1 Eclipse IDE query.

Relevance Definition. The goal of this query is to find posts that talk to the Eclipse IDE or related plugins/tools.

Query. eclipse

Results. This query is intentionally ambiguous. Due to this ambiguity, almost all retrieved documents aren't relevant. The majority discuss astronomy and the solar/lunar eclipse phenomena. Others simply use the verb *eclipse* as a way to refer to something that has fallen into disuse. In the end, very few documents reference the Eclipse IDE across all systems, as can be confirmed by the PR-Curve.

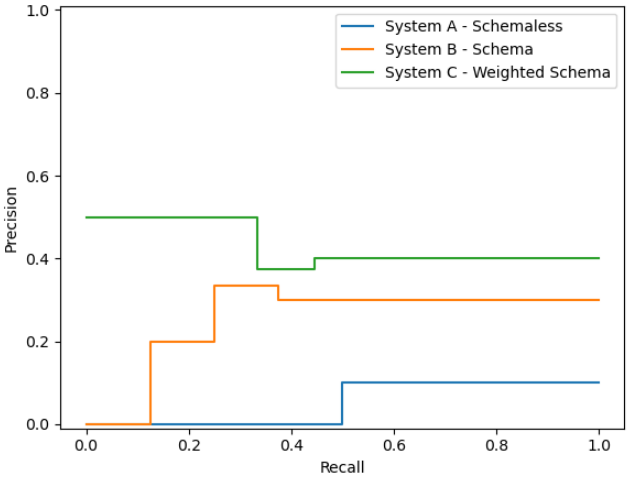

Figure 7: Eclipse query precision recall curve

System	AP	P@10	R@10	F_β
Schemaless	1.00%	10.00%	50.00%	11.90%
Schema	22.20%	30.00%	37.50%	31.25%
Weighted Schema	54.15%	40.00%	44.44%	40.82%

Table 4: Eclipse Query Results

8.2.2 *JSON command line tools.*

Relevance Definition. This query intends to retrieve all posts that showcase tools that manage files/streams in the JSON format, on the command line.

Query. +json tool "command line"

Results. System B achieved better average precision than system A, with a difference of 20%. The usage of the term "command line" seems to reduce the number of irrelevant documents retrieved on all systems, as the usage of the words command and line is relatively common. In fact, the group found that the results of the same query with these terms matching separately proved to produce significantly worse results (reducing precision and recall by 20% in all systems). PR-Curve.

8.2.3 *Version Control Tools.*

Relevance Definition. This query aims to discover posts that showcase tools for software version control or news related to the topic.

Query. +"version control" tools -svn

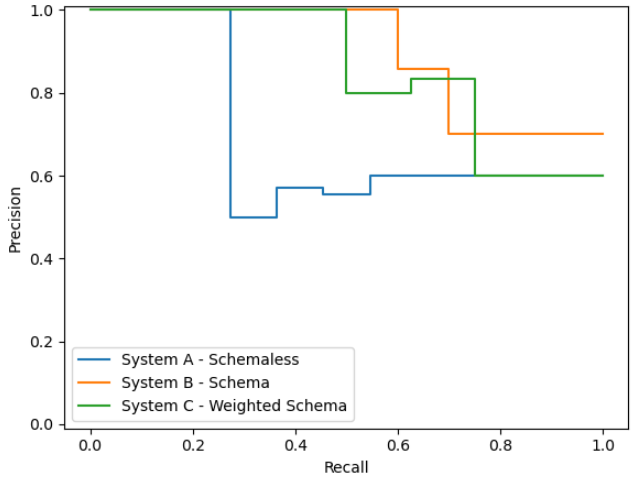

Figure 8: JSON command line query precision recall curve

System	AP	P@10	R@10	F_β
Schemaless	72.02%	60.00%	54.55%	58.82%
Schema	92.10%	70.00%	70.00%	70.00%
Weighted Schema	85.07%	60.00%	75.00%	62.50%

Table 5: JSON Query Results

Results. All systems seem to produce similar average precision results, while portraying a significant disparity between their recall values, as can be seen below. The majority of results found referred to GitHub[**?**] or GitLab[**?**] news.

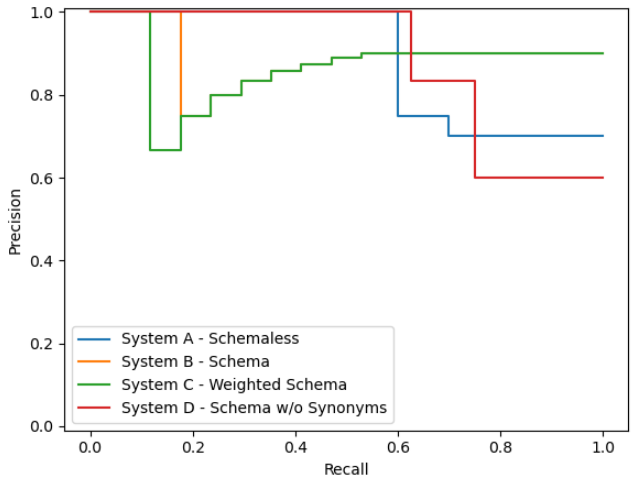

Figure 9: Version control query precision recall curve

8.2.4 *News filtering.*

Relevance Definition. This query aims to find stories from news websites discussing topics relating to China.

Query: `China +(newssite\_filter:news)`

System	AP	P@10	R@10	F_β
Schemaless	90.85%	70.00%	70.00%	70.00%
Schema	89.04%	90.00%	52.94%	78.95%
Weighted Schema	85.71%	90.00%	52.94%	78.95%
Schema w/o synonyms	87.07%	60.00%	75.00%	62.50%

Table 6: Version Control Query Results

Results. The intention of this query is showcasing the news filter effectiveness. As such, only the **weighted schema** performed the query; once with the news filter, and once without the filter. As shown on the PR-Curve, when using the filter, the system manages to only find relevant documents, while the other system only finds one relevant document. When not using the filter, most stories are personal blog posts or guide discussing how to optimize content for the Chinese market, and personal takes on Chinese policies/culture.

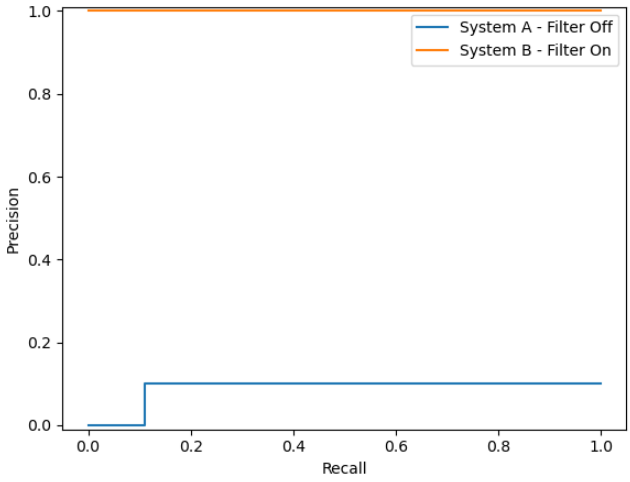

Figure 10: News query precision recall curve

System	AP	P@10	R@10	F_β
Schema w/ news filter	2.11%	10.00%	11.11%	10.20%
Schema w/o news filter	100.00%	100.00%	50.00%	83.33%

Table 7: China Query Results

9 FUTURE WORK AND IMPROVEMENTS

Regarding data collection/processing improvements, the dataset could be improved by collecting all types of posts from all time, collecting all comments (instead of just the top two of each story), and by finding the real descendants count of comments. Also, the web scrapping part of the pipeline should be able to handle all websites, e.g., YouTube.com by extracting the video description, converting PDF files to text (processing binary data), and handle cases of 404 errors/missing content.

Regarding search improvements, the weighted system seems to be performing below expectations. This is likely a problem of the chosen weights, and can be fixed by experimentation. The synonym list proved to be helpful, specially when considering more than the top 3 results, but in our test query, the results were worse for these top 3. This could be improved by weighting the exact match of the original term higher than the synonyms.

In the future, n-grams on story titles can be used to provide auto-completion features for the system's front-end. Finding a way to identify stories that could be classified as *niche* would allow users to find results about newer/unknown/unpopular tools and topics, when they wish to explore those areas.

In order to improve user experience, Solr's faceting mechanism can be used to tune queries using the story type field and the news story filter. This would allow the user to see how many results retrieved match those categories, and filter their search with the desired ones.

10 CONCLUSION

The data analysis allows for the following conclusions: there is enough data to feed the planned Information Retrieval system; it is varied in topics and scopes; the data is of a high quality.

With more time, it would be possible to obtain the data relative to all stories, comments, and possibly even URLs. This would make the temporal exploration of the data more interesting, allow for the correction of the approximation made on the comment's descendants count (mentioned in the Post Processing section), and create an even more interesting Information Retrieval system. Also, it would prove fruitful to improve the web scrapping part of the pipeline in order to be able to retrieve textual content from all websites (e.g. YouTube.com) and some file types (e.g. PDF, PNG, JPEG, ...).

Both the indexing and retrieval processes are fast and streamlined. The information retrieved is frequently relevant and the system with the schema performs well. The weights defined for the **weighted system** didn't improve the results in all cases, and can likely be fine-tuned.

"Solr" allows for expressive and rich queries, and the filters defined improve results significantly.

REFERENCES

[1] hackernews. 2021. hackernews (accessed on 12/12/2021). https://news.ycombinator.com/
[2] jq. 2021. jq (accessed on 20/11/2021). https://stedolan.github.io/jq/
[3] jsdom. 2021. jsdom (accessed on 20/11/2021). https://github.com/jsdom/jsdom
[4] Mozilla. 2021. Mozilla's Readability (accessed on 20/11/2021). https://github.com/mozilla/readability
[5] Hacker News. 2021. Hacker News API (accessed on 16/11/2021). https://github.com/HackerNews/API
[6] Masatoshi Nishimura. 2020. HackerNews Post Datasets (accessed on 17/11/2021). https://github.com/massanishi/hackernews-post-datasets
[7] node fetch. 2021. node-fetch (accessed on 20/11/2021). https://www.npmjs.com/package/node-fetch
[8] numpy. 2021. numpy (accessed on 20/11/2021). https://numpy.org/
[9] pandas. 2021. pandas (accessed on 20/11/2021). https://pandas.pydata.org/
[10] solr. 2021. solr (accessed on 11/12/2021). https://solr.apache.org/
[11] sql. 2021. sql (accessed on 21/11/2021). https://sqlite.org/index.html
[12] Wikipedia. 2021. F-score (accessed on 12/12/2021). https://en.wikipedia.org/wiki/F-score](https://en.wikipedia.org/wiki/F-score)