HackerSearch: an Information Retrieval system

Milestone 1

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

In this project, textual data, corresponding to the year 2018, was collected from the "HackerNews" link aggregator and corresponding linked URLs. After refinement and analysis, this data will be used to develop an Information Retrieval system.

KEYWORDS

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

1 INTRODUCTION

"HackerNews" is a link aggregator website, 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 first milestone of the development of "HackerSearch". This platform will serve as an intuitive and powerful search engine for "HackerNews" posts/content.
This report is divided into 2 main sections. Each of these sections describes different tasks completed in order to achieve the final goal for this **milestone: data preparation**. 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* [4] 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**. It was decided 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.

Supervised by Sara Fernandes and Sérgio Nunes.

2021.

The API [4] does not provide a way to perform bulk downloads. In order to streamline the process of fetching data, a starting dataset [5] was used. This dataset contained the raw data of stories collected using the API [4]. 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 are joined into a single one using the **jq** [1] 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 [4] was used to download the data of the top two comments of each story. This translates to around 100000 comments.
To download this data, curl with persistent connections was used, thus reducing the time expended establishing connections with the API [4], 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. This file was built using **jq** [1] (to select the IDs of the top comments) and **awk** (to build the URLs using the IDs that were being piped). 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 [1] 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 are frowned upon on "HackerNews", most stories (90%) do not have textual content (apart from the title). Instead, users are incentivized to share URLs to the content they want to discuss/share. This limits the searchability of stories.

To work around this, a small program was developed using Mozilla's Readability tool [3] (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** [6] to fetch the target website's data and **jsdom** [2] to feed this data to the readability tool [3].

Jq [1] is used to select the stories containing a URL, and GNU's split is used to divide the information of these stories into multiple files (each used by a respective sub-process). The website's extracted

textual content is stored in a JSON object alongside its post ID. Each sub-process stores the content it is responsible for in a file. The JSON objects in these files are subsequently concatenated into a single JSON list using **jq** [1] and stored into the **html_content.json** file.

3 DATA PROCESSING

While being collected and before being analyzed, the data is passed through a pipeline where it was transformed and filtered.

3.1 Initial Processing

The processing and filtering processes described in this subsection were applied 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 is reported by enough users, 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** [1] script that encompasses all conditions mentioned above.

Comments suffered a similar filtering process (using **jq** [1]): dead and deleted comments were filtered out. 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 were filtered 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 is compressed 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 all the data was gathered, it was necessary to perform some final cleanups and save it in a more fitting format: at this point, the data was saved in three JSON files which needed to be unified.

The 3 JSON files containing the data were imported into python's **pandas [8] DataFrames** to be worked on. The steps described below were taken using the **pandas[8] and numpy [7]** modules in python3.

All the stories in the dataset had a **type** field with the value **story**. This happened because only the posts of that type were downloaded (as mentioned in the Posts/Stories Data collection subsection. This field was replaced by 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).

It was 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 [4], 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 database (described in this section), using the **sqlalchemy python module**.

For the data domain, a conceptual model was created 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 when the story was posted.
- 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).

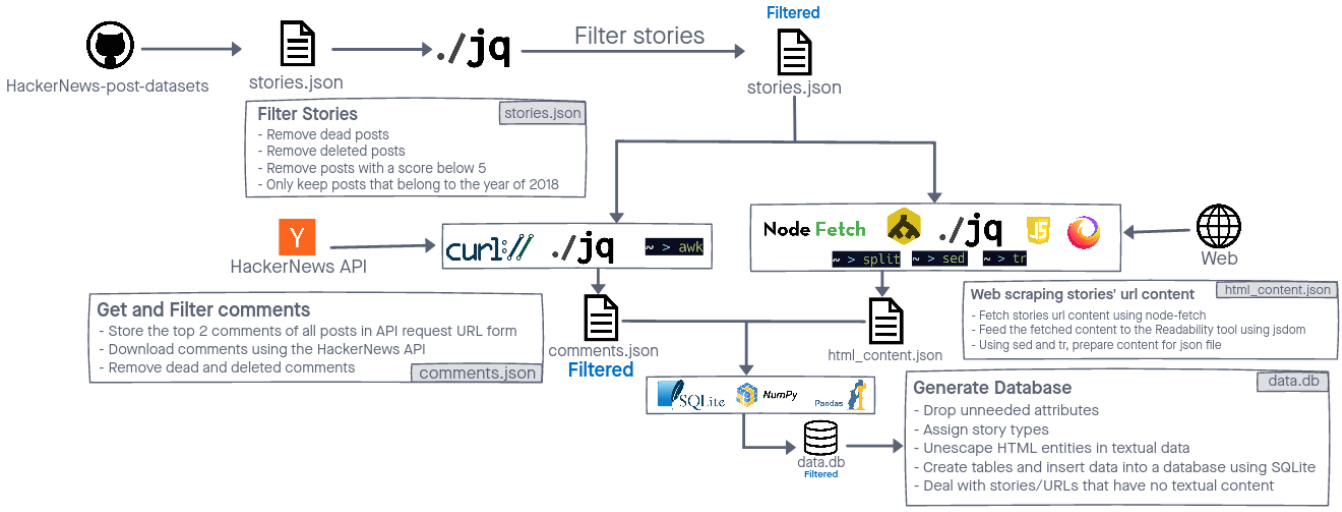

Figure 1: Dataflow Diagram

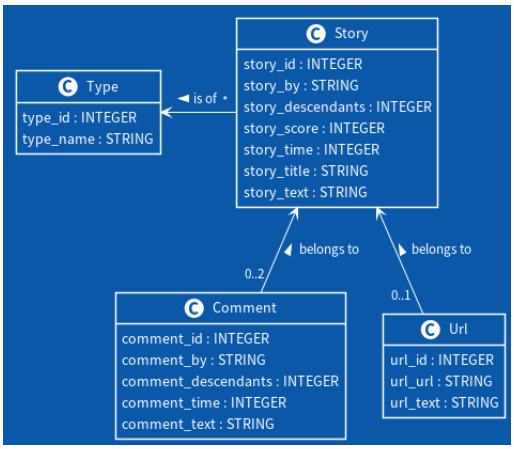

Figure 2: Conceptual Model

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. These contents were obtained as described in the Data Collection section.

Finally, the *Comment* table is used to represent 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.

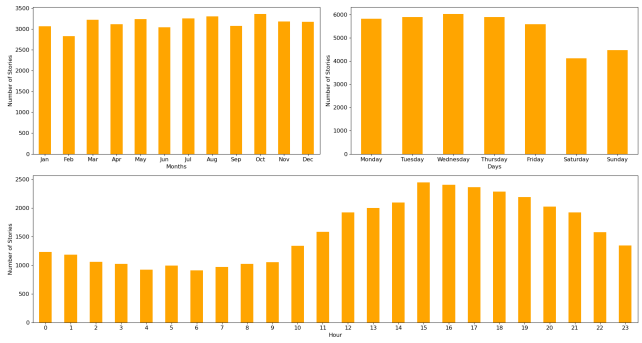

Figure 3: Number of stories per time

By analyzing the dataset, it can be concluded that there is 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.

The distribution of posts based on type is unbalanced. 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.

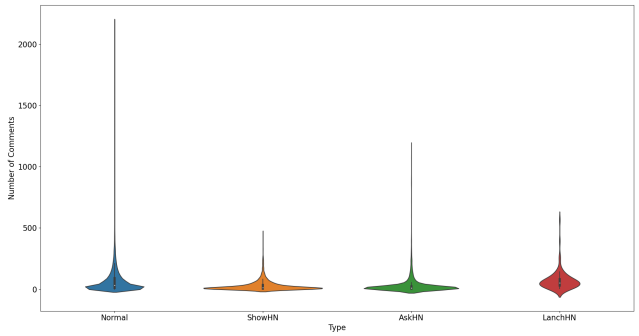

Figure 4: Number of comments per type

'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).

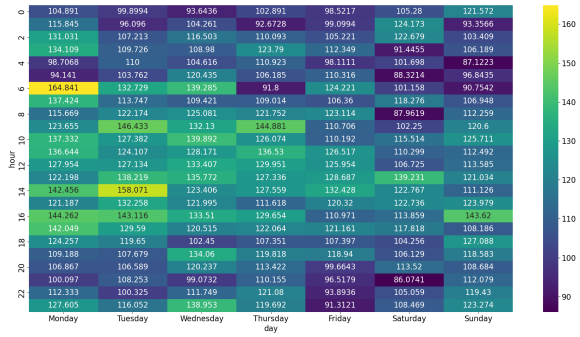

Figure 5: Average score in reference to time and day of the week

By analyzing the heatmap that relates the score of a story with the time and day of the week of when the story 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 average higher scores are achieved by stories posted between 6AM and 2PM on a Monday and from 9AM to 6PM on a Tuesday. Most of 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 upvotes) was posted on Wednesday at 11PM, which has an average score of 138, while its neighboring cells have a score of 111 and 100.

The average score of a story can be related to the number of comments it has (user engagement). Usually, a higher number of

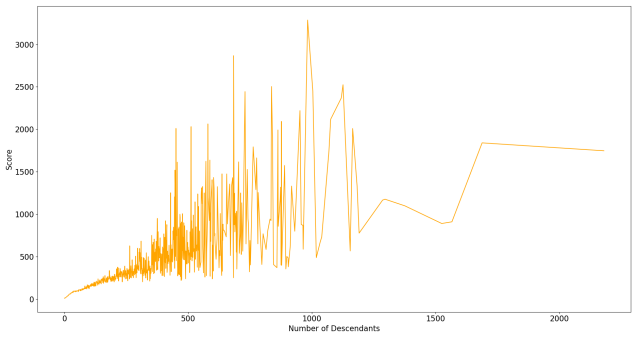

Figure 6: Score of story per number of comments

comments, leads to a higher score. Up until reaching 500 comments, stories' scores stay below 1000 points. Stories with the highest scores are found between the interval of 750 and 1000 number of comments. There isn't enough data to conclude anything about stories with more than 1000 comments, but the ones that exist in the dataset, appear to score lower than expected (about the same as stories with 750 to 1000 comments)

6 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. All of these factors and the use of a relational database, to store the data in its final form, will provide the flexibility and ease of use to advance in the project.

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, ...).

REFERENCES

[1] jq. 2021. jq. https://stedolan.github.io/jq/
[2] jsdom. 2021. jsdom. https://github.com/jsdom/jsdom
[3] Mozilla. 2021. Mozilla's Readability. https://github.com/mozilla/readability
[4] Hacker News. 2021. Hacker News API. https://github.com/HackerNews/API
[5] Masatoshi Nishimura. 2020. HackerNews Post Datasets. https://github.com/massanishi/hackernews-post-datasets
[6] node fetch. 2021. node-fetch. https://www.npmjs.com/package/node-fetch
[7] numpy. 2021. numpy. https://numpy.org/
[8] pandas. 2021. pandas. https://pandas.pydata.org/