`newsreduce.org`

A personalised and privacy-respecting newsreader

Seán Healy

August 2020

Abstract

A web-based newsreader is designed and implemented using a cocktail of software engineering, natural language processing and machine learning techniques. The newsreader aims to tackle some issues in the current newsreader space: user privacy concerns, information overload, advertising influence and low coverage.

Acknowledgements

I must thank Dr. Kevin Koidl for his valuable advice and help throughout my dissertation. I must also thank Visma (my employer) for granting me three months leave in order to complete this large project and dissertation.

Contents

Chapter 1

Introduction

1.1 The task

The task of personalised news aggregation is a complex one, but it can be broken down into several more manageable tasks. These tasks are well described by the following questions:

1. What is news?
2. What is a news source?
3. What is a good/bad news source?
4. What is the official homepage of a news source?
5. How do we discern article links on a homepage from other links (ads, etc.)?
6. Who is the author of an article?
7. When was an article released/published/printed?

8. Which parts of a webpage actually correspond to article content, i.e. headline, byline, body, visual media and captions?

9. What is a highly importance topic for the current news cycle?

10. How do identify similar articles, and eliminate all but the best sources for that article?

11. Is the *best source* the paper that “got there first”, or the paper with highest rank, or the paper that covered the topic the best, or some other heuristic based on a combination of these metrics?

1.2 What is news?

This question is mostly a philosophical one. But for the practical purposes of this project, an estimated answer is given by way of examples. People may agree that things you read in a newspaper are news, for example *The Irish Times*, *Reuters*, *The Financial Times.* But then there are other forms of media that lie in an ontological sense somewhere between ‘online newspaper’ and ‘online radio station’, or between ‘online newspaper’ and ‘quiz website’ (*BuzzFeed*), or between ‘online newspaper’ and ‘TV station’ (*BBC*). For most of the examples given so far, at least sometimes, these sources contain information people would describe as news. In this project, the answer to the question ”Is something a newssource?” is determined by supervised machine learning. Different encyclopedia pages are labelled with a binary class, *news source* or *not news source.* From this data, a program is generated that takes unlabelled encyclopedia pages as input, and returns a probability score that the encyclopedia page is a *news source*.

This pattern of supervised machine learning is repeated for several of the questions surrounding news aggregation, including “What is the official homepage of a news source?”, “Given a webpage that is an article, what is the author of the article, the date the article was published, and what is the headline? The process is described in more detail in Section 3.6.

Chapter 2

Background

2.1 The web

2.2 News aggregation

Newsreaders and aggregators have been around for quite some time now. The RSS specification first emerged in 1999 (RSS Advisory Board, 1999), and quickly grew in popularity, eventually becoming the key ingredient in the now defunct *Google Reader*.

From the experiences outlined in Section 3.2, RSS today seems to be a poor source for the task of "high-coverage news crawling", the process by which large amount good quality news data are extracted from the public web.

The style of such legacy news aggregation lives on in some way in today's popular news feeds, including the Twitter and Facebook timelines, as well as in multimodal media platforms such as YouTube. On these platforms, a sequential lists of news items are presented to users, and to some extent, the user chooses what they see, by ‘following’ or ‘liking’ a news source. But the choice of items now follows a new methodology, and there are clear differences between the legacy *RSS feeds* approach, and the current algorithmic approach.

There is a tradeoff. Previously, users had granular control over what they would read, and what they would not read, but the cost was the time and effort it took to amass a collection of RSS feeds that was well curated to the user. Depending on the user’s choice of newsreader, there may have not been additional filter settings. Some newsreaders had powerful keyword functionality, that could allow a user, for example, to block all COVID-19 news on Sundays.

Now, users may not need to spend as much time setting up their own curation service, or manually navigating to different sites intermittently throughout the day, in order to read the news. It’s highly automated. A news source they never heard of before may appear before them on their social media feed. It could be a welcome addition to their news diet. On the other side, if a news article is boring or unimportant, a user may be less likely to see it now, since its appearance on the ‘timeline’ is often based on its overall performance in the community (numbers of likes, karma, etc.).

The shift from the old news landscape to the new news landscape has created new problems, however. There is growing concern over user privacy, leading to the introduction of GDPR laws in Europe (“General Data Protection Regulation”, 2016). The same machine learning features used to get user news feeds so ‘right’ also happens to be very valuable for spurious ad campaigns.

In 1998, Brin and Page argued for general search engine development to be pushed into "the academic realm". They stated concerns over the influence of advertising on quality search results. But google's search service can no longer truly be described as "in the academic realm". Its new inner workings are now a business secret, protected by proprietary software licenses. Furthermore, *Google News*, the news aggregator launched in 2002, has never really been in the academic realm, except for the release of a large dataset. With this in mind, another motivation for developing `newsreduce.org` was to provide a news aggregation service within the academic realm. The algorithms used, and software developed, are all open-sourced, and accessible on the public git repository: `https://github.com/sean-healy/newsreduce/`. The code repository includes `BASH` automation scripts intended to allow others to set up their own NewsReduce servers. The architecture is modular, so users may be interested in using the web crawler alone, or the various parsers, or even the entire system. As a diclaimer, some technical knowledge in Linux and `BASH` are needed to successfully launch a NewsReduce instance.

2.3 Large-scale information retrieval services

2.3.1 Google

Now ubiquitous, Google search, and the technology behind it, were first introduced by Brin and Page (1998). Much of the software engineering techniques outlined in Section 3 draw from the topics introduced in that paper, including the need for appropriate compression, distributed web crawling, appropriate word and URL ID methods, and of course, a ranking algorithm. Pagerank is essentially an application of Markov processes and Monte Carlo simulation, though there is no mention of either of those topics by Brin and Page (1998).

The task of ranking the importance of webpages, based on a directed graph made up of web links, can be explained through an analogy to a statistical problem in the game 'Monopoly' (Project Euler, 2004). In a simple variant of Monopoly, where each tile carries the same fine (€1), consider the problem: *What is the best tile to purchase*? The answer, of course, is the tile that people land on most frequently. To determine which exact tile that is, a Monte Carlo simulation is initiated, with a probability value of 1 on the start tile, and a value of 0 on every other tile. This reflects the fact that before any moves, a player is bound to be on the start tile. Next, that probability of 1 is split into 36 pieces (possible dice combinations), and each piece is distributed to the next round of possible player destinations. The links between tiles are not always trivial, given additional complicated Monopoly rules.

This probability splitting process is repeated, and over time, popular destinations accumulate a lot of *rank* (probability), and less popular destinations lose rank. As the game progresses, the variance in tile ranks from one move to the next begins to converge, and at a certain variance threshold, the algorithm doesn't evaluate the next move. The tile ranks are then finalised.

Pagerank basically works the same way as this example, except that the tiles are webpages, and the dice rolls are links on the pages.

2.4 Personalised search

After the development of Pagerank (Brin & Page, 1998), techniques were suggested by Page, Brin, Motwani, and Winograd (1999) to make search 'personalised' to users. A simple approach to this emerged when Page et al. were solving the problem of *rank sinks*: webpages with at least one backlink, but no outgoing links, or pages that form a cyclic clique, where no page in the clique references a page outside of the clique. In a naive Pagerank implementations, these pages would gradually accumulate large amounts of rank, as they would essentially correspond to final states in a finite state automaton. Betting on a user being in a the final state would always yield large returns.

One solution was to use a set of predetermined pages as *escape routes* from rank sinks. In other words, if we consider Pagerank to represent the way a user might navigate through the web, then the set of escape route pages may be thought of as the random user's home button, or links on the user's bookmarks toolbar. By tweaking this set of escape links, and ensuring they not only accumulate rank from rank sinks, but also from a *tax* applied to all pages, Pagerank can generate rankings relevant to a given user (or at least to the image of that user formed from their 'bookmark links').

This simple approach is used by `newsreduce.org` (Section 3.7) to calculate webpage ranks from the perspective of users in different English speaking regions. This helps ensure that an Australian reader receives Australian news, an Irish person receives Irish news, and so on.

2.5 The open corpus problem

A key topic in the area of news aggregation is the open corpus problem, as outlined by (Henze & Nejdl, 2001), Brusilovsky and Henze (2007) and Koidl (2013). Henze and Nejdl (2001) encountered the problem from the domain of e-learning software, but others have noted the complexity of an *open-ended* corpus, albeit with different terminology. In the much cited paper of Brin and Page (1998), for example, large difficulty is mentioned with regards "unstructured" web content, and the task of gradually crawling and storing it. Some noted problems include the tendency of websites to break, returning 400 or 500 errors. The former error code corresponds with the broader issue of *link rot*. Markwell and Brooks (2003) studied the demise of "URL viability" over the course of 25 months in the area of biochemistry and molecular biology education, and found that 27.5% of links were lost over that period (47.9% for .com URLs).

A naive solution would be to discard a resource immediately after the webpage begins to return an error code, but this would erroneously discard resources from websites with short, intermittent errors.

Another difficulty arises from tendency of websites to have vastly different `HTML` patterns. News sources will often correctly wrap headlines in `H1` tags, wrap author names in the appropriate metatags, times and dates in the correct `TIME` tag. But some will wrap the headline immediately within a `DIV`, others will place each paragraph within separate `SECTION` tags, and so on. The possibilities for `HTML5` misuse are vast, especially as a result of `CSS`, and the web developers' ability to simulate the expected functionality of one `HTML` tag using `CSS3` rules applied to different `HTML` tag.

Some of the issues Brin and Page (1998) faced are no longer as relevant today. Parsing syntactically invalid `HTML` is now trivial, given the wide availability of robust web scraping libraries in popular programming environments. `newsreduce.org` uses the `JSDOM` library (Section 3.4). Another issue that is far less daunting is network latency. With image and `AJAX`-style content loading disabled, popular websites appear to load much faster today than in the 90s, and the web crawling task has a smaller bottleneck as a result.

The huge lengths Brin and Page (1998) went to in order to optimise their implementation of a search engine for sequential disk acess seems less relevant now, given the widespread adoption of solid state drives, and huge increases in random access memory in today's servers. Sequential access still provides a performance boost, but in certain areas of the application it may be more worthwhile to rely on the filesystem for efficiency. This was the reasoning behind `newsreduce.org`'s storage methodology: one file per resource version format, and each group of resource version stored within a compressed archive (Section 3.3).

All that said, there are new issues relating to crawling in today's web. Some websites have adopted *single-page application* (SPA) architectures, which involve initially loading a web page with a small to medium sized chunk of JavaScript, but no actual content. The chunk of JavaScript code is then responsible for fetching individual pieces of content, laying them out in the browser, and updating parts of the display if necessary. SPAs and the originally underlying technology (`AJAX`) have presented a problem to web crawling for over a decade at the time of writing (Matter, 2008), (Mesbah, Van Deursen, & Lenselink, 2012).

There are documented solutions to crawling non-static pages, but these solutions rely on allowing JavaScript code to execute in a virtual environment, waiting some amount of time, and then taking a snapshot of the document's DOM structure. JavaScript is of course Turing complete, so this approach is cumbersome, and without a rendering time limit, the problem is actually *undecidable.* This has led to sites like LinkedIn and Twitter becoming difficult to crawl without strong processing power. At the time of writing, inspecting the HTML source of arbitrary LinkedIn and Twitter webpages (via Firefox) reveals no textual similarities to the eventual text that a user would see.

Fortunately, the largest online news sources that aren't social networks tend to stick with static `HTML`, but there is no guarantee that will always be the case, and detecting if a web resource involves SPA architecture in any form adds a layer of complexity to the task of crawling the web. For the purposes of this thesis, the problem is set to the side, and only static `HTML` is considered.

The web of today plays a default role in the dissemination of infromation. Information retrieval papers used to regularly cite the growth rate of the web (Brin & Page, 1998), (McBryan, 1994). But today it's hardly worth the trouble, since the vast majority of news sources and businesses are now on the web. In fact, finding news sources that are limited to an offline audience now appears to be the harder task. This ubiquity is beneficial for corpus construction, as there is a lot more data available. But it is now significantly harder to verify sources. The term *fake news* has been in the public consciousness and parlance since approximately 2016[1]. The apparent rise of *fake news* has now been studied. Researchers have even tried to solve the problem using various data mining techniques (Shu, Sliva,

[1]Google Trends data observed in August 2020

Wang, Tang, & Liu, 2017).

All of the problems discussed represent a tricky open-corpus, far beyond the complexity of building ML models for a large but static data set. In the best case, the `newsreduce.org` crawler can go days without a problem, though in earlier versions, many bugs would arise, stopping the crawler after a few minutes or a few hours (Section 3.2).

2.6 Machine learning

Within machine learning, there is currently a distinction between two sets of algorithms, broadly described as *explainable AI* (XAI) and *black box* models. The `newsreduce.org` service will use both methods in some sense, so these terms should first be disambiguated.

2.6.1 Explainable AI

As stated by Goebel et al. (2018), XAI is not a new is not a new field. The expert systems of the 80s were one clear example of explainable AI. The result of a program could be easily explained by examining the logic rules of the program. When we speak of XAI, reference is usually being made to the observed inability of the artificial neural networks (ANN) to *explain* their classification decisions. ANNs can display near-human performance on certain classification tasks (Krizhevsky, Sutskever, & Hinton, 2012), but the absence of explanations can make debugging impossible, and bias difficult to spot. In a system of decision trees, on the other hand, the *paper trail* of a trained program's decision making can be easily extracted by observing the paths through a decision tree that were taken, and in boosted systems, by observing the weights at each branch. The popular example of a situation where XAI is important emerges in the field of computer vision, and the racial bias therein. Much research has no been published in attempts to tackle this issue, (e.g. Wang, Deng, Hu, Tao, & Huang, 2019). The previously cited paper of (Goebel et al., 2018) makes some progress towards XAI by applying neural networks to the task of assigning textual explanations to generated image descriptions.

Generally speaking, a relatively *explainable* approach was sought for the main task of this thesis, and that ruled out ANN models. There are many models that fit the XAI requirement and still perform well on classification tasks. An implementations of AdaBoost decision trees, with various configuration settings, is outlined later in Section 3.6. The precision and recall of this approach appears sufficient for the task of news feed personalisation (more results in Section 3.9). Decision trees allow the user to gradually build up a model of their news preferences, while still maintaining the ability to examine the reasons behind negatively labelled news items. This offers readers insights into their underlying preferences, both conscious and subconscious. The training data for the decision trees are assembled from the `likes` and `dislikes` of the user. The `newsreduce.org` website stores the data on the user's side, and the data is assembled through the display of an upvote and downvote button alongside each news item. Section 3.8 outlines how this is accomplished using web browser technology.

Chapter 3

Methodology

3.1 Hashing

Words, URLs and every other *entity* stored in `newsreduce.org` databases, is identified by an ID, in the form of a fixed length cryptographic hash of the entity's content. Before hashing, the content is prefixed by the the entity's own type name.

For example, the URL `https://example.com/` would be identified by the hash of `url:https://example.com/`. The resulting ID is the first 12 bytes of the cleartext's `SHA-3` hash. This hashing schema was chosen in order to fulfil several requirements:

1. The ability to know the ID of an entity without interacting with the database.
2. Uniqueness of IDs across the entire database (over per-table uniqueness).
3. An infeasible hash collision rate among entities.

Requirement 1 ensures that small programs could reason about the data in the `newsreduce.org` system, without necessarily needing a database connection in order to obtain the ID of a word, or of a URL, or of anything else. Knowing the ID of an entity before storing it is also valuable in terms of concurrency and performance. If a system with a large amount of data were to rely on the auto-increment feature (a popular 'ID generator' in MySQL), then huge amounts of communication to and from the database would be needed in order for various machines to collaborate.

For example, in the tokenisation stage outlined in Section 3.5, several processes or machines are tokenising web resources in parallel. Most of the documents processed will contain the word '*the*' at least once. Relying on auto-increment would lead each machine to send the following SQL query to the database server: '`SELECT ID FROM Word WHERE value = `the';`'. In any period of time, this costly ID check would need to be carried out for each word in the lexicon of documents being processed during that period. Knowing with certainty that the ID of 'the' is `A91A5E09C79E0221159C8DFF`, without needing a database connection, is a highly valuable part of a competitive IR service.

The choice of 12 bytes for the ID can be explained by considering the birthday paradox and the pigeonhole principle. If an IR service may potentially store billions of unique objects, then the ID space needs to be in the region of trillions, to avoid ID collisions (when two unrelated machines assign the same ID to different objects). This is illustrated in Figure 3.1. Using 12 byte IDs across different SQL tables renders a collision practically impossible.

Unfortunately, MySQL doesn't have a native 12 byte number type, so the ID

Entities	Probability of collision
4,194,304	0.00000000000001%
8,388,608	0.00000000000004%
16,777,216	0.00000000000017%
33,554,432	0.00000000000071%
67,108,864	0.00000000000284%
134,217,728	0.00000000001136%
268,435,456	0.00000000004547%
536,870,912	0.00000000018189%
1,073,741,824	0.00000000072759%
2,147,483,648	0.00000000291038%
4,294,967,296	0.00000001164153%
8,589,934,592	0.00000004656612%
17,179,869,184	0.00000018626451%
34,359,738,368	0.00000074505805%
68,719,476,736	0.00000298023219%
137,438,953,472	0.00001192092824%
274,877,906,944	0.00004768370445%
549,755,813,888	0.00019073468138%
1,099,511,627,776	0.00076293654275%
2,199,023,255,552	0.00305171124684%
4,398,046,511,104	0.01220628622226%
8,796,093,022,208	0.04881620601105%
17,592,186,044,416	0.19512188925244%
35,184,372,088,832	0.77820617397564%
70,368,744,177,664	3.07667655236558%
140,737,488,355,328	11.75030974154045%
281,474,976,710,656	39.34693402873665%
562,949,953,421,312	86.46647167633872%
1,125,899,906,842,624	99.96645373720974%
2,251,799,813,685,248	99.99999999999873%
4,503,599,627,370,496	100.00000000000000%

Figure 3.1: The probability of a hash collision when the number of entities reaches different stages, given 12 byte hash IDs.

column on each table in this project uses the type `DEC(30)`, which is not as performant as a `BIGINT`, and lacks certain functionality, such as bitwise operations. Another frustration is that `DEC(30)` usually maps to a string instead of a big integer in various programming environments. For this reason, in hindsight, and in future projects, `BIGINT` (8 bytes) is recommended.

Using the hashed ID method outlined here, with 8 bytes the probability of a hash collision in a database of up to 4,294,967,296 identifiable entities is roughly 39%. In any case, a few hash collisions is arguably acceptable in the area of news aggregation (as opposed to, say, the area of banking).

3.2 Crawling

In order to crawl large amounts of web pages fast, a distributed system of crawlers was set up, along with automation scripts written in `BASH`. The scripts allowed machines to be added to the pool of 'crawlers'. Files were shuttled to a master machine using `rsync` periodically, and various cleanup scripts were used in order to ensure that crawler instances retained a certain level of available disk space. This approach was was inspired by distributed crawling method outlined by Brin and Page (1998).

One difference is that `newsreduce.org` relies heavily on Redis, whereas Brin and Page write about a custom URLServer that was designed from scratch. Another key difference is the method of distributing and storing files. Rather than some distributed, fault-tolerant filesystem, `newsreduce.org` simply stores all the compressed web pages on a single RAID-1 disk. This design choice was made for three reasons. Firstly, the task of this project is limited to recent news, so not as much disk storage is needed, as compared to a tool for the general web. Secondly, disk space has advanced a lot since Google was launched; renting a dedicated server with 4TB of disk space now costs as low as €33 a month[1]. Lastly, setting up a distributed filesystem is costly, requires time, specialised expertise, and ties the project to 'the cloud', effectively ruling out the possibility of development without an internet connection. Relying on an old-fashioned single hard drive allowed `newsreduce.org` to be developed fast, and at times without a connection to the internet.

3.3 Compression

Compression was used to reduce the disk space needed to store different versions of web pages. The Zstandard (`zstd`) package from Facebook (Collet, 2015) was chosen as a tradeoff between read/write speed and compression ratio. This tradeoff was a significantly easier choice than the one made by (Brin & Page, 1998), when Zstandard was not an option. Cursory comparisons between `zstd` and other compression options (both faster and more space-efficient) revealed that the library was among the fastest, and yet its compression ratio was second only to `xz`.

Brin and Page (1998) write that they compress each `HTML` file, but an alternative approach was taken in this project. News sites are subject to much larger amounts of change than the general web. Large news sources such as *The New York Times* often release hundreds of articles per day, and each arti-

[1]Price estimated from Hetzner's server auction `https://www.hetzner.com/sb`, 2020

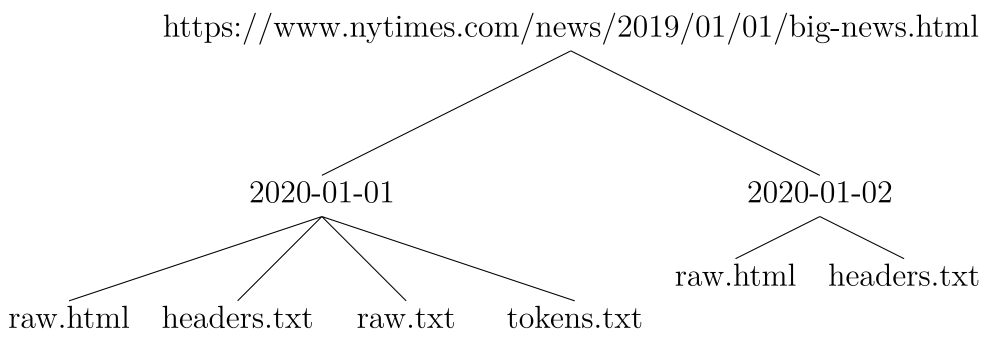

Figure 3.2: An example of what a resource archive's hierarchical structure.

cle published will usually appear on a homepage or section page at least once. This causes webpages to move from one state to the next, with small differences between neighbouring states, and large differences between distant states (i.e. today's front page versus last year's front page). For this reason, `newsreduce.org` doesn't compress each individually fetched `HTML` file, but instead maintains one archive per resource, and many versions per archive.

Figure 3.2 illustrates the archiving format used to store the corpus of resource versions. In reality, the number of formats is much larger than those shown in the hierarchical diagram.

Figure 3.3 is a more realistic portrayal of a resource's archive. The archive is placed at a location resembling the URL's ID (Section 3.1). Under each archive, there is a file named prefixed with the timestamp of the version it represents, and suffixed with the unique name of a version format. The different formats are explained in more detail in Section 3.4.

```
https://www.nytimes.com/news/2019/01/01/big-news.html
└── (ywwmgz6zwg8vnp02pe.tzst)
    ├── 1596573378626_link-hits.bin
    ├── 1596573378626_min-tokens.txt
    ├── 1596573378626_bol.bin
    ├── 1596573378626_rbin-botg.bin
    ├── 1596573378626_bow.bin
    ├── 1596573378626_ndoc-vec.bin
    ├── 1596573378626_doc-vec.bin
    ├── 1596573378626_raw-words.txt
    ├── 1596573378626_bin-botg.bin
    ├── 1596573378626_rbobg.bin
    ├── 1596573378626_rbow.bin
    ├── 1596573378626_raw-links.txt
    ├── 1596573378626_rbosg.bin
    ├── 1596573378626_bin-bol.bin
    ├── 1596573378626_headers.txt
    ├── 1596573378626_rbotg.bin
    ├── 1596573378626_bin-bosg.bin
    ├── 1596573378626_bin-bow.bin
    ├── 1596573378626_raw.html
    ├── 1596573378626_word-hits.bin
    ├── 1596573378626_rbin-bobg.bin
    ├── 1596573378626_rbin-bow.bin
    ├── 1596573378626_tokens.txt
    ├── 1596573378626_bobg.bin
    ├── 1596573378626_bosg.bin
    ├── 1596573378626_rbin-bosg.bin
    ├── 1596573378626_bin-bobg.bin
    ├── 1596573378626_sub-docs.txt
    ├── 1596573378626_anchor-paths.txt
    └── 1596573378626_botg.bin
```

Figure 3.3: A realistic portrayal of a resource's archive.

3.4 Intermediate representations

As mentioned in Section 3.2, crawled resources are stored in their entirety as HTML documents, and the HTTP headers are stored in separate plaintext files. For clarity, each resource will have separate HTML and header files for each time point at which the resource was crawled, which makes `newsreduce.org` a version archiving program.

But these initial resource formats are a small fraction of the resource representations stored. To compare the performance of machine learning models across different representations, many representations are generated. Below is a non-exhaustive list of intermediate formats:

- A sub-documents file
- A document vector
- A links file
- A tokens file
- A tokens file, with stop words removed
- A HTML title file
- The following formats are generated separately from the two token files, with and without stop words.
  - Bag of words (BOW)
  - Binary bag of words (BBOW)

- Binary bag of n-grams
    - Binary bag of bigrams (BBOBG)
    - Binary bag of trigrams (BBOTG)
    - Binary bag of skip-grams, with skips ≤ 2, bag-size $= 2$ (BBOSG)

This extensive number of formats was intended to allow different representations to be easily compared with machine learning algorithms. Different representations perform well on different tasks.

3.4.1 The sub-documents file

This file is designed specifically for extracting a particular class of information from a webpage. The format extracts all the leaf nodes from a HTML document, and associates the attributes (text, `href`, title, etc.) with the path of the leaf node. The path is made up of the HTML class names, IDs, and tag names sorted by reverse order of tree height in the document DOM. In a fictitiously simplified webpage, the sub-document file might look like this:

```
{"text":"Seán Healy"} span.author p div
{"text":"My Homepage"} h1#headline
{"text":"Home", "href"="https://seanh.sh/} a header
```

Each row in the sub-documents file can be considered a document in itself, and information extraction may then be accomplished with standard document classification techniques, such as Naive Bayes, decision trees, etc.

3.4.2 Parsing HTML

3.5 Data cleaning

3.5.1 Alphabet normalisation

Before constructing prediction models with the textual data from web pages (Section 3.6.4), the characters appearing within each page are normalised to ASCII (a-z). This reduces the total number of features needed by the model. The drawback is a loss of specificity in languages that rely heavily on accents (French, for example). That said, as mentioned in Section 1.1, this project targets English language news only. Alphabet normalisation also reduced the disk space needed to store a mapping of word IDs to word values. In order to collect a large number of candidate characters for normalisation, the names file from Consortium (2020) was parsed, and simple string lookups for 'LATIN A', 'LATIN B', and so on, were performed.

Figure 3.4 illustrates just a handful of the different forms the letter 'a' can take. In most representations of resources, each of these forms is replaced with the ASCII 'a'. The same process is applied for all other Latin characters, along with punctuation, since there are many ways to quote, hyphenate, parenthesise, insert pauses into sentences, etc. All language representation models have drawbacks, and as stated, the drawback in this procedure (*Unicode equivalence*) is a potential loss of specificity. But the larger benefit (for a language like English) is that commonly anglicised words have a lower chance of being identified as separate symbols within texts. A simplified example: if a news article about a | A a À Á Â Ã Ä Å à á â ã ä å Ā ā Ă ă Ą ą Ǎ ǎ Ǟ ǟ Ǡ ǡ Ǻ ǻ Ȁ ȁ Ȃ ȃ ...

Figure 3.4: A few of the different (accented) forms the letter ‘a’ can take.

Tomáš Mikolov instead uses the incorrect form, *Tomas Mikolov*, after alphabet normalisation, the article would still identify *Tomáš Mikolov* as a topic.

Alphabet normalisation is only applied on internal representations, and the output that news readers see must always be the original unicode form.

3.5.2 Tokenisation

3.6 Machine learning

3.6.1 Decision trees

As mentioned in Section 2.6, AdaBoost with decision trees was the algorithm of choice for `newsreduce.org`’s news personalisation. The algorithm was applied to several tasks:

1. Determining which Wikipedia webpages correspond to news sources.
2. Finding the official homepage of a news source from a wikipedia webpage.
3. Determining if a webpage is an index.
4. Determining if a webpage is a news item.
5. Personalising the selection of articles a user sees.

3.6.2 Bayesian classifiers

Although bayesian classifiers aren't as powerful as decision trees, they are simple and efficient, and they do play their part in certain parts of the `newsreduce.org` application. For example, a classifier that determine if different newspapers might be from the same geographical region uses Bayesian classification with a few pre-labelled newspapers. Although this is supervised learning, the labelling is trivial, since `newsreduce.org` currently only supports news from 6 countries.

3.6.3 Feature engineering

Traditional computer science algorithms played their part in feature engineering, particularly in the task of identifying a news source's homepage from its Wikipedia article. As mentioned in Section 3.4.1, the sub-documents file is made up of all the leaf nodes in a HTML file's DOM. For anchors, the left hand side of a sub-document will contain attributes for the text and the `href`. The domain name is extracted from the `href`. Next, two features are built, firstly by comparing domain to the last part of the current resource's path, and secondly by comparing the anchor's text to the last part of the path. This comparison uses the Levenshtein distance between two strings, and normalising the distance to a scale between 0 and 1 (0 being low similarity, and 1 being high similarity). This normalisation is carried out as follows:

$$Similarity(a, b) = 1 - \frac{LevenshteinDistance(a, b)}{max(|a|, |b|)} \tag{3.1}$$

Normalisation fills two purposes here. Firstly, it makes separate results from *LevenshteinDistance* somewhat comparable. E.g. two 100,000 word texts with 8 differences will no longer have the same difference/similarity score as the score between 'newspaper' and 'zookeeper'. In other words, the size of the texts being compared will normalise the similarity score. Secondly, plotting similarity from 0 to 1 is convenient for decision tree algorithms, and it's also convenient (in the case of `newsreduce.org`) to place the *similarity* end closer to 1 than to 0. This is because `newsreduce.org`'s DT implementation uses a default of 0 when a feature is missing from a document or sub-document.

3.6.4 Training

3.6.5 Predicting

3.7 Ranking

A definition of Pagerank Pagerank (Page et al., 1999) is explained in simple terms in Section 2.3.1, but a more specific specification should also be outlined. The implementation of Pagerank used by `newsreduce.org` treats webpages as nodes p in a graph, $p_i \in P$, $1 \leq i \leq |P|$, $i \in \mathbb{N}$. Links from one page to another are represented as directed edges between nodes in that graph, $l_{i,j} \in G$, $1 \leq i, j \leq |P|$, $i, j \in \mathbb{N}$, $G \subseteq P \times P$. It is convenient to represent the link structure with two function B and F, such that $B(j) = \{i \mid l_{i,j} \in G\}$, and $F(i) = \{j \mid l_{i,j} \in G\}$. In English, B returns the other webpages that link to a webpage (otherwise known as its *backlinks*: indexes to the pages that link to the

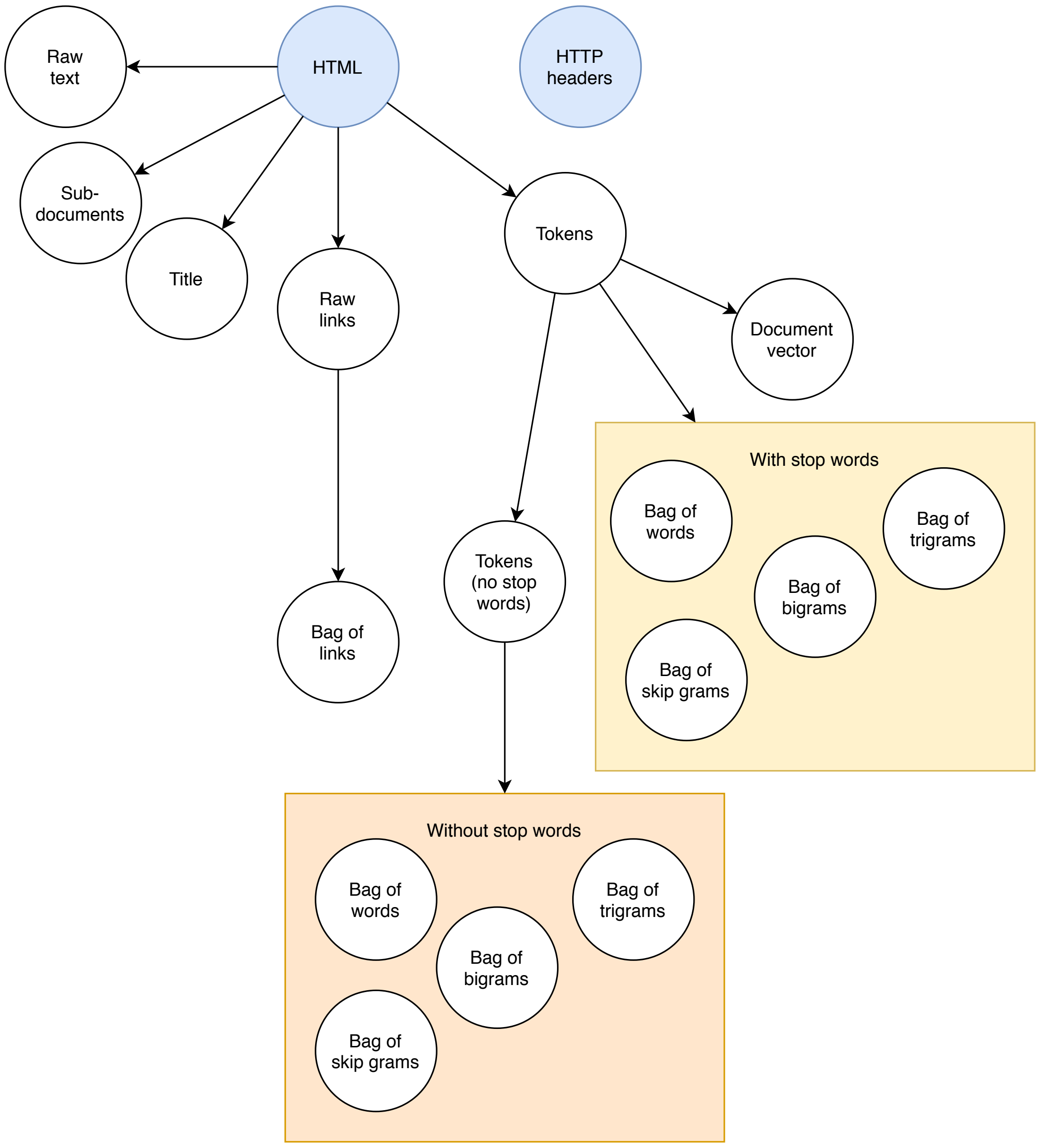

Figure 3.5: The process by which different experimental resource models are created. Blue nodes are models created by the crawler. The others are created later. A comparison of these models, and their relative performance in text classification, is provided in Section 3.9.1.

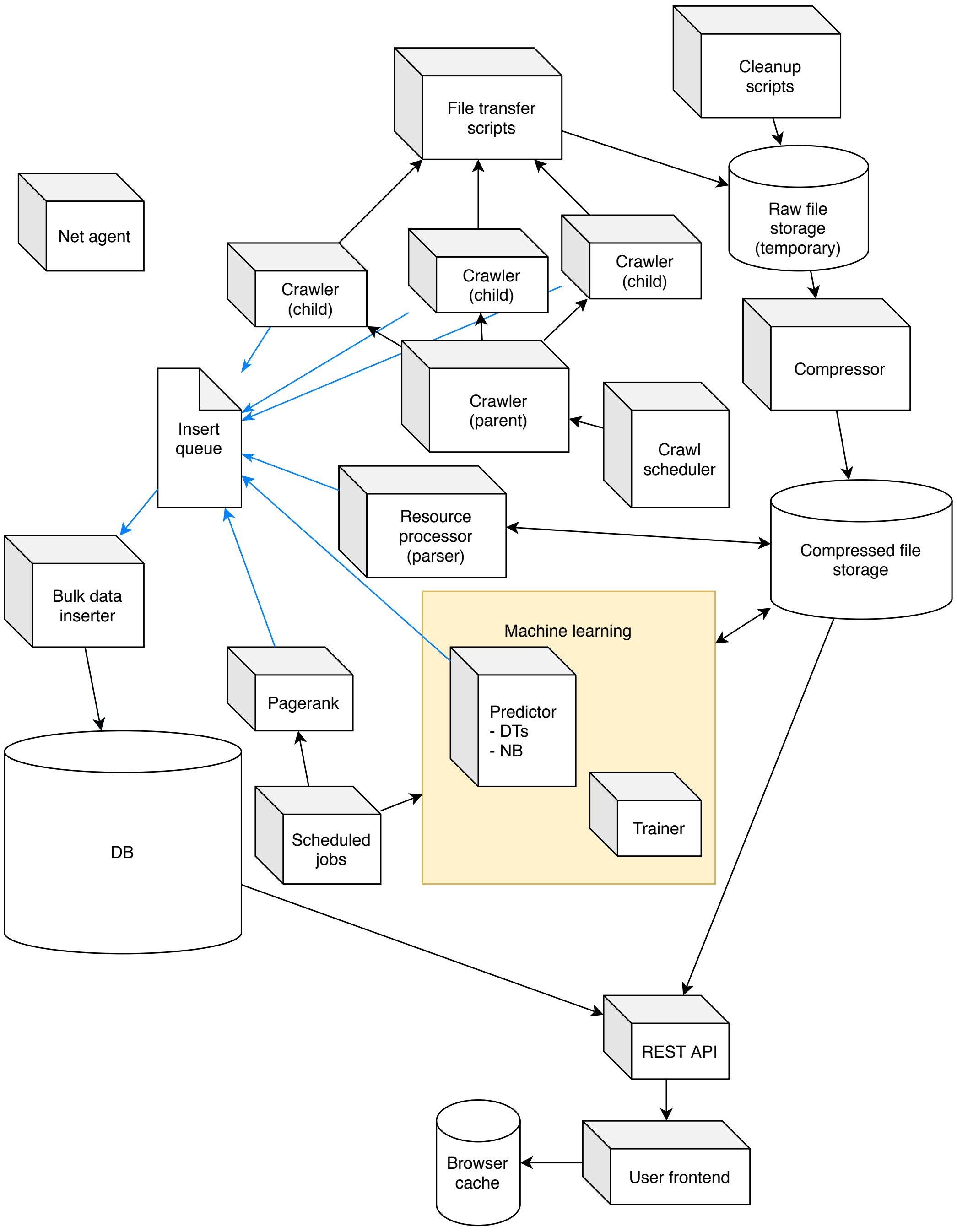

Figure 3.6: Broad overview of the system's architecture.

webpage at index i). F returns the *forward links*, the links on a webpage. This corresponds more closely to the general idea of a 'link' on the web. A node cannot link to itself. `HTML` files are pre-processed to ensure the such self-referential links don't arise as input to the algorithm.

The implementation of Pagerank converges at some stage, leaving the final rankings of the webpages stored in memory. But before then, the *running estimates* of the webpage rankings can be defined with the binary function R. The two arguments to R are: the webpage index i, and the stage in the algorithm t, $t \geq 1$, $t \in \mathbb{N}$. A *stage* represents one cycle through the algorithm's central loop (Algorithm 1, line 5). Finally, Pagerank can be defined recursively:

$$R(i,t) = \begin{cases} \frac{1}{|P|}, & t = 1 \\ \sum_{j \in B(i)} \frac{R(j,t-1)}{|L(j)|}, & t \neq 1 \end{cases} \tag{3.2}$$

Algorithm 1 is the naive implementation, and in real world scenarios it would

be open to manipulation through rank sinks. As mentioned in Section 2.3, one solution was found through a finite set of *escape* links, or *sources of rank* (Page et al., 1999). These links are chosen purposefully in various invocations of Pagerank, in order to produce geographically sensitive rankings. Recalling that `newsreduce.org` is currently designed to work for English language regions, some example escape links are presented in Figure 3.7. To simplify things, only 6 English-speaking countries are currently supported. These countries were chosen by population size.

Algorithm 1 Simplified Pagerank algorithm

```
Require: $T$, an error threshold, needed to halt Pagerank.
Require: A set, $webpages$, where each element ($webpage$) has a property, $links$.
    This returns a set of other webpages linked on the $webpage$.
        Each $webpage$ also represents an index, so it can be used to address the
    memory location of a $webpage$'s rank.
 1: for each $webpage \in webpages$ do                    ▷ Initializing variables
 2:     $previousRanks[webpage] \leftarrow \frac{1}{|webpages|}$
 3: end for
 4: $error \leftarrow \infty$
 5: while $error > T$ do
 6:     for each $webpage \in webpages$ do
 7:         $nextRanks[webpage] \leftarrow 0$
 8:     end for
 9:     for each $webpage \in webpages$ do
10:         if $|webpage.links| > 0$ then
11:             $previousRankPiece \leftarrow \frac{previousRanks[webpage]}{|webpage.links|}$
12:             for each $link \in webpage.links$ do
13:                 $nextRanks[link] \leftarrow nextRanks[link] + previousRankPiece$
14:             end for
15:         end if
16:     end for
17:     $error \leftarrow variance(previousRanks, nextRanks)$
18:     $tmp \leftarrow nextRanks$          ▷ Swapping $nextRanks$ and $previousRanks$
19:     $previousRanks \leftarrow nextRanks$
20:     $nextRanks \leftarrow tmp$
21: end while
22: $finalRanks \leftarrow previousRanks$
```

Region	Pagerank escape links
AU	`www.news.com.au`, `www.theaustralian.com.au`
IE	`www.irishtimes.com`, `www.independent.ie`
NZ	`www.nzherald.co.nz`, `www.stuff.co.nz`
UK	`www.theguardian.com`, `www.dailymail.co.uk`
US	`www.nytimes.com`, `www.wsj.com`
JM	`jamaica-gleaner.com`,`www.loopjamaica.com`

Figure 3.7: Pagerank escape links by ISO-3166-2 country code.

In reality, many escape links are used per region, and these links are chosen based on a combination of factors: The ccTLD (e.g. `.co.nz`), the presence of substrings resembling the region name within the hostname (e.g. `irishtimes.com`), and bayesian classification applied to the weighted lexicon of words appearing in crawls of the various domains.

Algorithm 2 is a modified version of Pagerank (Algorithm 1), aiming to tackle rank sinks, while also implementing personalised ranking. The argument $escapeWebpages$ is filled with a set of webpages that may be of particular relevance to a geographic region.

3.8 Privacy

Algorithm 2 Pagerank algorithm, with measures against rank sinks

```
Require: $T$, the error threshold from Algorithm 1.
Require: The set $webpages$ from Algorithm 1.
Require: $escapeWebpages$, a set of webpages useful for dealing with rank sinks.
1: for each $webpage \in webpages$ do                    ▷ Initializing variables
2:     $previousRanks[webpage] \leftarrow \frac{1}{|webpages|}$
3: end for
4: $error \leftarrow \infty$
5: while $error > T$ do
6:     for each $webpage \in webpages$ do
7:         $nextRanks[webpage] \leftarrow 0$
8:     end for
9:     $redistribute \leftarrow 0$
10:    $redistributeRatio \leftarrow 0$
11:    $decayRate \leftarrow \frac{1}{2 \times 85}$ ▷ Set with the expected iterations (in this case $\approx 85$).
12:    for each $webpage \in webpages$ do
13:        $previousRank \leftarrow previousRanks[webpage]$
14:        if $|webpage.links| > 0$ then
15:            $rankPiece \leftarrow \frac{previousRank \times (1 - redistributeRatio)}{|webpage.links|}$
16:            for each $link \in webpage.links$ do
17:                $nextRanks[link] \leftarrow nextRanks[link] + rankPiece$
18:            end for
19:            $redistribute \leftarrow redistribute + previousRank \times redistributeRatio$
20:        else
21:            $redistribute \leftarrow redistribute + previousRanks[webpage]$
22:        end if
23:    end for
24:    for each $webpage \in escapeWebpages$ do
25:        $nextRanks[webpage] \leftarrow nextRanks[webpage] + \frac{redistribute}{|escapeWebpages|}$
26:    end for
27:    $error \leftarrow variance(previousRanks, nextRanks)$
28:    $tmp \leftarrow nextRanks$                ▷ Swap $nextRanks$ and $previousRanks$
29:    $nextRanks \leftarrow previousRanks$
30:    $previousRanks \leftarrow tmp$
31:    $redistributeRatio \leftarrow redistributeRatio + decayRate$
32: end while
33: $finalRanks \leftarrow previousRanks$
```

3.9 Results and discussion

3.9.1 Comparison of resource models

3.10 Conclusion

References

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, *30*, 107-117.

Brusilovsky, P., & Henze, N. (2007). Open corpus adaptive educational hypermedia. In *The adaptive web* (pp. 671–696). Springer.

Collet, Y. (2015). Zstandard [Computer software manual]. Retrieved from `http://facebook.github.io/zstd/`

Consortium, U. (2020). Unicode 13.0.0 final names list [Computer software manual]. Retrieved from `https://unicode.org/Public/UNIDATA/NamesList.txt`

General data protection regulation [Computer software manual]. (2016). Retrieved from `https://gdpr-info.eu/`

Goebel, R., Chander, A., Holzinger, K., Lecue, F., Akata, Z., Stumpf, S., … Holzinger, A. (2018). Explainable ai: the new 42? In *International cross-domain conference for machine learning and knowledge extraction* (pp. 295–303).

Henze, N., & Nejdl, W. (2001). Adaptation in open corpus hypermedia. *International Journal of Artificial Intelligence in Education*, *12*(4), 325–350.

Koidl, K. (2013). *Cross-site personalisation* (Unpublished doctoral dissertation). The University of Dublin, Trinity College Dublin.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems 25* (pp. 1097–1105). Curran Associates, Inc.

Markwell, J., & Brooks, D. W. (2003). "link rot" limits the usefulness of web-based educational materials in biochemistry and molecular biology. *Biochemistry and Molecular Biology Education*, *31*(1), 69–72.

Matter, R. (2008). *Ajax crawl: making ajax applications searchable* (Unpublished master's thesis). Eidgenössische Technische Hochschule Zürich, Department of Computer Science.

McBryan, O. A. (1994). Genvl and wwww: Tools for taming the web. In *Proceedings of the first international world wide web conference* (Vol. 341).

Mesbah, A., Van Deursen, A., & Lenselink, S. (2012). Crawling ajax-based web applications through dynamic analysis of user interface state changes. *ACM Transactions on the Web (TWEB)*, *6*(1), 1–30.

Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). *The page rank citation ranking: Bringing order to the web* (Tech. Rep.). California: Stanford InfoLab.

Project Euler. (2004). Problem 84 [Computer software manual]. Retrieved from `https://projecteuler.net/problem=84`

RSS Advisory Board. (1999). RSS 0.90 specification [Computer software manual]. Retrieved from `https://www.rssboard.org/rss-0-9-0`

Shu, K., Sliva, A., Wang, S., Tang, J., & Liu, H. (2017). Fake news detection on social media: A data mining perspective. *ACM SIGKDD explorations newsletter*, *19*(1), 22–36.

Wang, M., Deng, W., Hu, J., Tao, X., & Huang, Y. (2019). Racial faces in the wild: Reducing racial bias by information maximization adaptation network. In *Proceedings of the ieee international conference on computer vision* (pp. 692–702).