Challenges facing privacy-focused news aggregation

Seán Healy

M.Phil. Speech and Language Processing

Trinity College Dublin

2020

Declaration

I declare that this dissertation has not been submitted as an exercise for a degree at this or any other university and that it is entirely my own work.

Seán Healy

September 2020

(Discursive text: approximately 15,000 words.)

Permission to lend or copy

I agree to deposit this thesis in the University's open access institutional repository or allow the library to do so on my behalf, subject to Irish Copyright Legislation and Trinity College Library conditions of use and acknowledgement.

Seán Healy

September 2020

Abstract

Crucial tasks in implementing a self-hosted news aggregation service are explored. A web-based newsreader is designed using various corpus, computational linguistics and software engineering techniques. The design aims to tackle some issues in the current newsreader space: user privacy concerns, information overload, advertising influence and low coverage. Ultimately, this dissertation presents a negative outlook on the efficacy of a self-hosted solution in the intelligent news aggregation space. Barriers of entry include monetary cost, limited processing power of the standalone server, and the impracticalities of server orchestration for the general user. Several open-source implementations are presented as solutions to news aggregation problems, with proposed performance enhancements for k-NN and random forest.

Contents

Acknowledgements

"I have not failed. I've just found 10,000 ways that won't work."

Thomas Edison

Firstly, I must thank Dr. Kevin Koidl for his valuable advice and help throughout my dissertation. I must also thank my girlfriend Isabel for her loving support and inspiration throughout this project. To my parents, friends and peers for their support, the journey would have been far more difficult on my own.

Finally, thank you to my employer, Visma, for granting me three months leave in order to complete this large project and dissertation.

Chapter 1

Introduction

The continuing decline of print news readership has been documented (Ma, Hui, Tong, Tse, & Wu, 2014; Thompson, 2019), alongside the growth of online social networks (OSN), and their expanding role in news dissemination (Goel, Watts, & Goldstein, 2012). Within the space of online news, there are highly automated news recommendation services, or *news aggregators*. Two popular news aggregators are *Google News* and *Yahoo News*. But there are many more niche news aggregators. In Sweden, a service named *Omni* exists, focussing on multi-source news from a Swedish perspective. In information technology circles, *Hacker News* is popular, a news aggregator by definition, with some social features.

There are open-source tools allowing people to create their own aggregators, including `tt-rss` (*Tiny Tiny RSS*, 2020) and Thunderbird's newsfeeds feature. These open-source tools currently lack intelligent functionality however, and sources must be added manually. Some online news sources aren't supported, particularly if the website doesn't provide an RSS or Atom feed link. Furthermore, RSS is currently considered legacy, and many newspapers fail to provide an RSS feed link. Some papers provide only one RSS feed for all news items, leading to information overload. In the worst cases, the items on the RSS feed may not match any of the items on the news site's current homepage.

1.1 The task

The goal of personalised news aggregation is a complex one, but it can be broken down into smaller 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 high importance topic for the current news cycle?
10. How does one identify similar articles, and eliminate all but the best sources from that cluster?
11. Is the *best source* the paper that "got there first", the paper with highest rank, the paper that covered the topic the best, or some other heuristic based on a combination of these metrics?

1.1.1 Motivation

Some of the 11 questions above may appear trivial. Arguably, most people know *news* when they see it. As for the more complicated problems (e.g. Question 11), a large team of experts from the world of news could create a pipeline to provide high-quality news

aggregation. But an operation on that scale would cost a lot of money. Furthermore, the task of news aggregation, i.e. republishing the work of other writers and journalists, could be considered very tedious work that ought to be automated.

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 news source?" 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 3.7.

1.3 Terminology

Some news related and technical terminology are used throughout this dissertation. These terms will now be introduced.

Class This term is used to mean some boolean or label that can be applied to a document. In most cases, boolean classes (*true* or *false*) are used instead of labels (many-valued logic). Though it is common to represent a document's class as a real number between 0 and 1, 0 being *most false* and 1 being *most true*.

False positive This defines a situation when a positive classification is made incorrectly, e.g. when some fact is said to be true for a document, but the fact does not actually hold true for that document.

False negative This defines a situation when a negative classification is made incorrectly, e.g. when we *miss* a document for which a fact would be true.

True positive and true negative As the names suggest, these two terms indicate when a classification is made (*true* or *false*), and the classification is correct.

News index This refers to certain kinds of webpages that list a number of news items. The category webpages within a news website (*World*, *Sport*, etc.) are examples of news indices.

News item This refers to webpages that contain news in some longform way (a few paragraphs, a headline, etc.). The term *article* is avoided since it relates mostly to newspapers, and people don't only obtain news from newspapers.

Resource This term refers to news items that have been crawled and stored by a news aggregation system. The term 'file' was avoided, since a resource could use many files (one file per format), and could also be represented by information stored in database tables.

Precision This is the rate of accuracy within positive classifications. It can best be understood as a question: "If a classifier labels an item as positive, what is the likelihood the correct label is in fact *positive*?"

Recall The ratio of items correctly classified as positive to the actual total of positives. This measures how well a classification system performs at the task of exhaustively finding all positive items in a dataset.

F1 This is a score that combines both precision and recall into a single metric, such that $F1 = 2(P \times R)/(P + R)$. It is a useful metric in determining the overall business value of a classifier. 100% precise classifiers aren't much use if recall is low, and the reverse is also true. High recall classifiers are rather useless if the precision is very low.

Coverage Throughout this paper, the term coverage is used in a sense similar to the idea of recall. A news aggregator with good coverage will be aware of the most important news articles, from the average reader's perspective. Low coverage news aggregators will miss important news stories entirely, or perhaps show those stories long after the stories are considered current and important.

Accuracy is another popular term in machine learning (correct classifications over total classifications), but that term is avoided in this dissertation, because it can be misleading when classes are not evenly distributed. For example, a classifier that says everything is a positive match would perform with 95% accuracy when 95% of items just happen to be positive matches.

1.4 The corpus

As will be discussed in Section 2.4, the unstructured web poses a challenge to structured data extraction. For example, in Figures 1.1 and 1.2, three different positions for 'official homepage' links appear on newspapers' Wikipedia articles. An intelligent web scraper is needed, capable of recognising all different scenarios, and extracting the links most likely to represent official homepages. Another task arises after the official homepages for news sources are determined. Which links on a homepage or other index page correspond to news items and other news indices? (Figures 1.3 and 1.4)

Figure 1.2: The position of official homepage links at the bottom of another Wikipedia webpage.

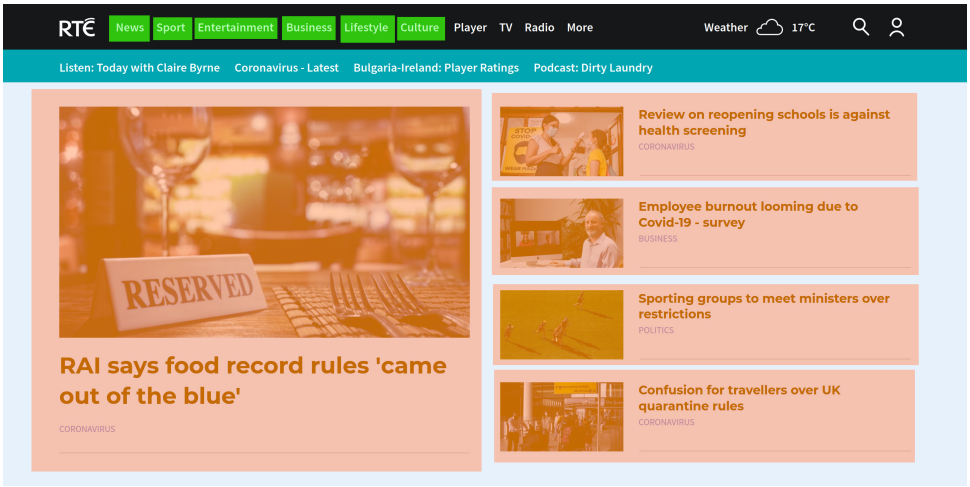

Figure 1.3: Index links (green) and item links (orange) on the RTE homepage.

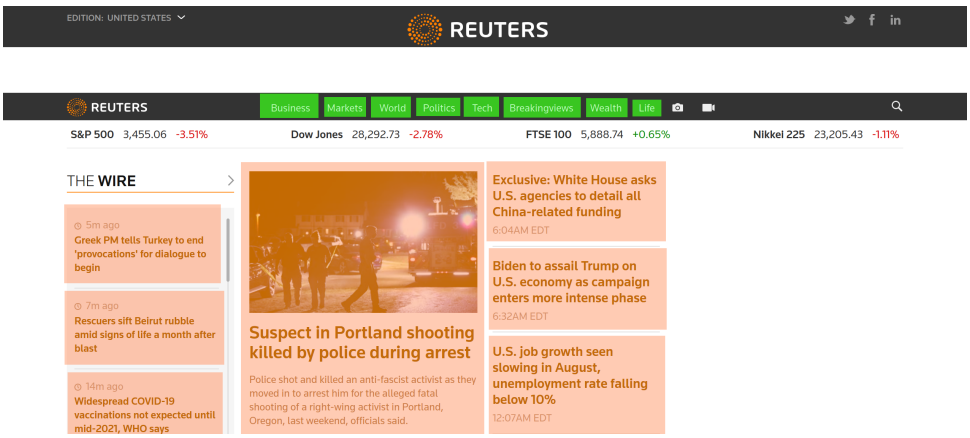

Figure 1.4: Index links (green) and item links (orange) on the Reuters homepage.

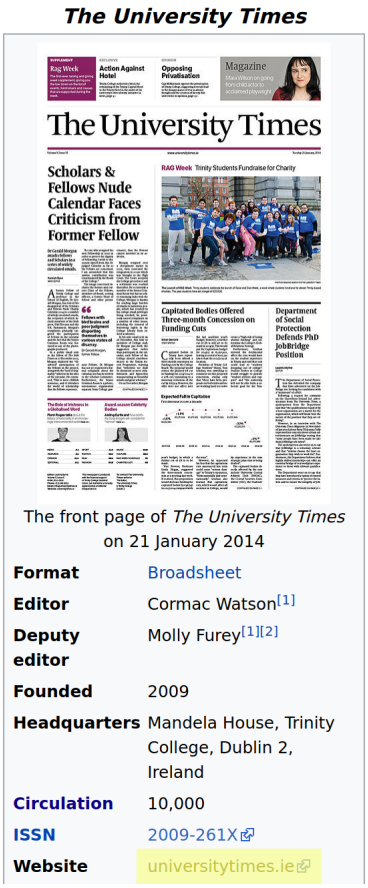

Figure 1.1: The position of an 'official homepage' link on a Wikipedia webpage's `vcard` (highlighted in yellow).

Within news items on each page, there are several more relation extraction tasks. Figures 1.5 and 1.6 show some of the key labelled information associated with news items. Note that in one news item (1.5), there is a single author, and in the other there are two authors (1.6). The publication date is labelled relatively in one news item, "a day ago", and literally in the other news item. One news item has a byline, the other one doesn't.

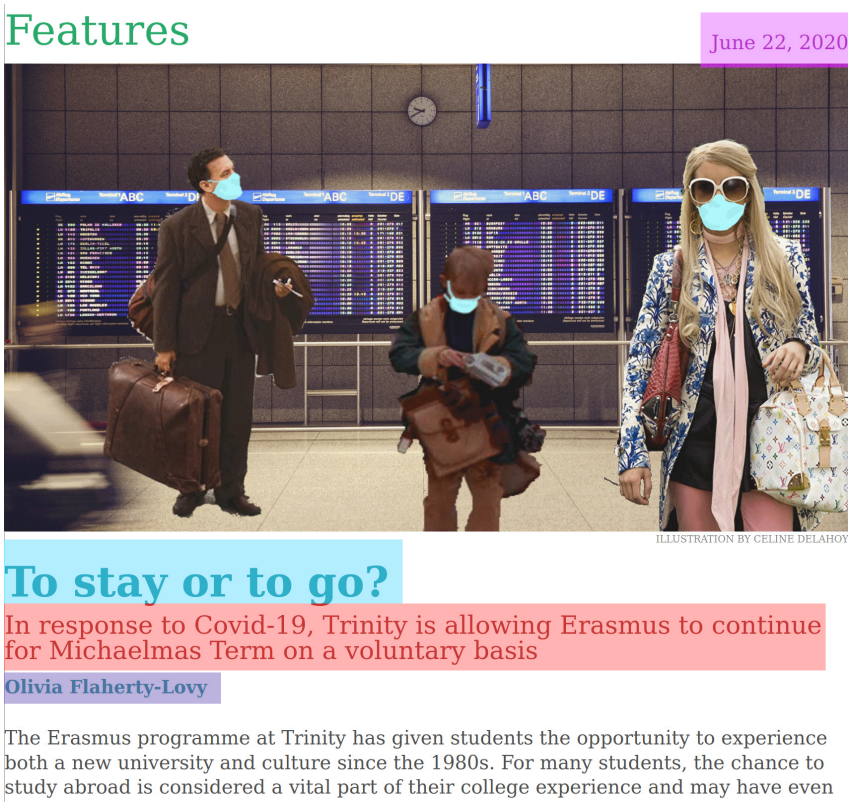

Figure 1.5: Information of interest on a news item page: publication date (pink), headline (blue), byline (red), author name (purple).

Figure 1.6: The same information types from Figure 1.5, on a different news item page.

The underlying HTML for each of these webpages also varies greatly. This level of variance necessitates a dynamic, probabalistic approach to web scraping, rather than manually writing rules based on the assumption of semantically correct and uniform HTML.

With a firm grasp on the tasks, terminology, and the structure the dataset will take, the next chapter will cover background research related to these tasks.

Chapter 2

Background

2.1 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 in Section 3.3, 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 list of news items are often 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 often 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 not have 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 this dissertation 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, there are many bugs remaining to be fixed, and some technical knowledge in Linux and `BASH` are needed to successfully launch a NewsReduce instance.

The software was initially intended to be run as a *private cloud* solution to news aggregation, similar to `tt-rss`. As will be discussed in Section 4.3, however, the task of implementing large scale news aggregation of a comparable quality to current options (e.g. *Google News*) proved more difficult that expected.

2.2 Large-scale IR services

Now ubiquitous, Google search, and the technology behind it, were first introduced by Brin and Page (1998). Some of the software engineering techniques outlined in Section 3 draw from the techniques outlined 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 (PR). PR is essentially an application of Markov processes and Monte Carlo simulation.

The task of ranking the importance of webpages, using 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.

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

2.3 Personalised search

After the development of PR, techniques were suggested by Page, Brin, Motwani, and Winograd (1999) to make search *personalised* to users. A simple approach 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 the clique. In naive PR 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 eventually being in a final state would 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 PR to model the way a user might randomly 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, PR can generate rankings relevant to a given user (or at least relevant to the image of that user formed from their 'bookmark links').

This simple approach will later be explored to to calculate webpage ranks from the perspective of users in different English speaking regions (Section 3.10). This would help ensure that an Australian reader receives some Australian news, an Irish person receives some Irish news, and so on.

2.4 The open corpus problem

A common difficulty in applications using web data (such as news aggregation) is the open corpus problem (Henze & Nejdl, 2001; Brusilovsky & Henze, 2007; Koidl, 2013). Henze and Nejdl 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. Some noted problems include the tendency of websites to break, returning 400 and 500 errors, or simply no error. One of the error codes (404) 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 corpus-related 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 3.1.3). Another issue that is far less daunting is network latency. When image and `AJAX`-style content loading are disabled, websites tend to load much faster today than in the 90s. As a result, the web crawling task has a smaller bottleneck. Nonetheless, network issues were still a major source of difficulty in the implementation portion of this dissertation.

The huge lengths Brin and Page (1998) went to in order to optimise their implementation for sequential disk access initially seemed less valuable today, given the widespread adoption of solid state drives, and huge increases in random access memory on business and commodity servers. Sequential access still provides a performance boost, but in certain areas of the application it may be more worthwhile to rely on the filesys-

tem or RDBMS for efficiency. This was the reasoning behind this disseration's storage methodology: one file per resource version format, and each group of resource versions stored within a compressed archive (Section 3.4). This storage format worked well in terms of space efficiency, but led to many bottlenecks, in particular the need to invocate compression programs N times to assemble datasets of N resources.

There are many new issues relating to crawling today's web. Some websites have adopted *single-page application* (SPA) architectures. Thse 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` (Section 3.3), 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 dissertation, the problem is set to the side, and only static `HTML` documents are considered in the design process.

2.4.1 The Web

The web at the time of writing plays a default role in the dissemination of information. Information retrieval papers used to regularly cite the growth rate of the web (McBryan, 1994). But today it's hardly worth the trouble, since the vast majority of news sources and businesses are now on the web, and have been for quite some time. In fact, finding news sources that are limited to an offline audience could now be a more difficult 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]. Its documented rise has led machine learning researchers to make attempts at tackling the problem (Shu, Sliva, Wang, Tang, & Liu, 2017). As with the issue of non-HTML content, this problem was not a priority while designing `newsreduce.org`. Although, the system design overview does leverage ranking algorithms strongly, which could cause the incidence of fake news to be less probable.

All of the problems discussed represent a tricky open-corpus, and this background literature was taken into consideration while building the crawler. Traditionally, data scientists work with static data sets, fixed in size, and assembled before programming begins. Highly performant models, in terms of precision and recall, can be developed using a static data set. But these models can be difficult to transition into a production pipeline, where the dataset is no longer static.

2.5 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 aimed to benefit from both methods in some sense; any project using word embeddings has some unexplainability in it. The XAI term should first be disambiguated.

[1]Google Trends data observed in August 2020

2.5.1 Explainable AI

As stated by Goebel et al. (2018), XAI is not a new field. The expert systems of the 80s were one clear example of XAI. 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 an artificial neural network (ANN) to *explain* its 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 tree in a forest. 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 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 3.7. The precision and recall of this approach appears sufficient for the task of news feed personalisation (more results in Section 4). 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.

2.5.2 Decision trees

When it comes to natural language text classification, a single decision tree will generally perform poorly in all metrics, including precision and recall (Section 4.1). But techniques

Day of week	Overtime	Annual leave	Bank holiday	Sick	Emergency	Worked
1	F	F	F	F	F	T
2	F	F	F	F	F	T
3	F	F	F	F	F	T
4	F	F	F	F	F	T
5	F	F	F	F	F	T
6	F	F	F	F	F	F
7	F	F	F	F	F	F
1	F	T	F	F	F	F
2	F	F	T	F	F	F
3	F	F	F	T	F	F
4	F	F	F	F	T	F

Table 2.1: Data used to generate the decision tree in Figure 2.1

have been invented to combine many decision trees into a *decision tree forest*. The final decisions from trees in the forest are combined in some way to produce results with higher precision and recall. One such technique is random forest, introduced by Ho (1995). and extended by Breiman (2001). Another technique is AdaBoost, first introduced by (Schapire & Singer, 1999).

A decision tree is comparable to the common notion of a *flowchart*, but with no cycles, and each node has exactly one parent. This fits closely with the description of a *taxonomy* from biology. Manually created decision trees are useful for classifying objects with certain features into different categories. Beginning at the root of the tree, a question is asked regarding the object. The answer to this question determines which branch is followed next. (Generally, there are two child branches per node.) A new question is then asked, and the process continues until a *leaf* is reached. Leaves are the nodes with no child branches, at the bottom of the tree. Each leaf corresponds to a class. The class is then assigned to the object.

Figure 2.1 is a very basic example of a decision tree. The tree could in fact be built automatically, from a spreadsheet of data regarding someone's work and life patterns. That spreadsheet may look something like Table 2.1

Some of the data in Table 2.1 is categorical (boolean), but the *day of the week* variable is numerical, non-continuous. When implementing a robust decision tree learning algorithm, it is important to acccount for both categorical, numerical and continuous

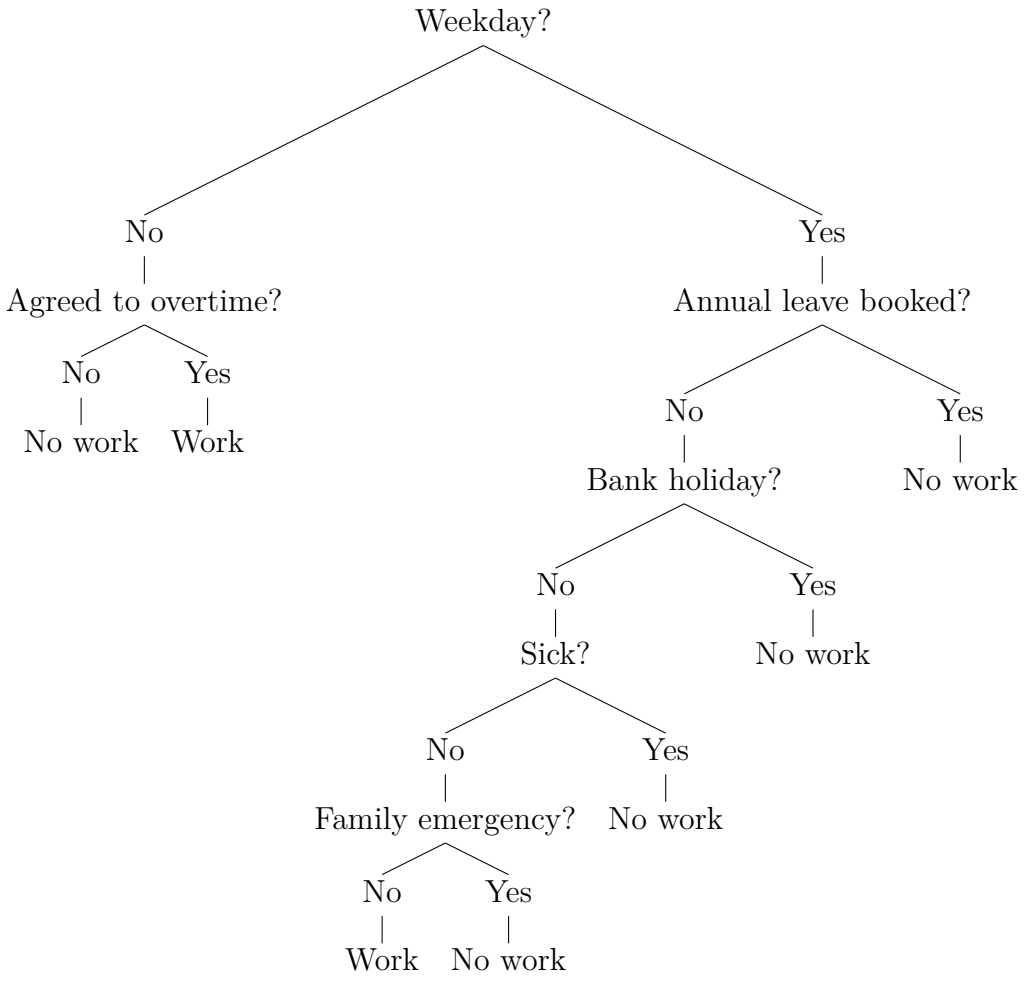

Figure 2.1: An example of a decision tree people may implicitly use to determine if they should go to work each morning.

variables. Otherwise, valuable features for classification could be excluded from selection. The CART algorithm[2] (Breiman, Friedman, Stone, & Olshen, 1984) is one way to achieve this robustness.

2.5.3 Choosing the right questions

Decition tree learning algorithms generally run top-down, first determing the best question to ask in order to split a dataset into more manageable parts. For each part, the same process is applied in order to determine the best followup questions. *Best* is determined using various methods, including *information gain* (Quinlan, 1986), *gini impurity* (from CART) and *entropy* (from information theory). Gini impurity is the most straightforward to calculate:

$$GI(p, C) = 1 - \sum_{c \in C} p(c) \tag{2.1}$$

C is a set of categories. p returns the ratio of the items that match the category c over the total number of items. In English, Gini impurity is the probability of the following event. If a random item and a random category are taken at a DT node, what is the probability that the category matches the item? For a node where all items have the same category, the probability is 1. The more categories represented at a node, the lower the probability becomes. In other words, using gini impurity to determine the best question to ask gradually leads to nodes with fewer distinct categories. This corresponds closely with the idea of minimising entropy.

2.5.4 Random forest

Random forest (Ho, 1995) applies *bagging* to a training data set, and builds a number of trees from these bags. Bagging works by splitting data into random chunks, and applying machine learning to each chunk separately, in order to build multiple classifiers (an ensemble). With random forest, the bags are built by randomly selecting N items from a dataset of size N. The items are selected with replacement, meaning that the

[2]Classification and regression trees

same item can be selected more than once. For each bag, a full decision tree is built (without pruning). While building each node in a tree, a random sampling of k features is taken from the full set of features, K. There is no required value for k, but $k = \sqrt{K}$ is a popular choice.

Trees in a random forest may be built in parallel. Pruning is not necessary if many trees are trained in the forest. Classification of items involves a vote among all the trees in the forest. A threshold is set according to desired precision and recall scores, and above that threshold, the result is a positive match. Below the threshold, the result is a negative match (for a binary classifier).

2.5.5 AdaBoost (AB)

AdaBoost (Schapire & Singer, 1999) also generates decision tree forests, but *boosting* is applied to the best performing trees in the forest, so that their *votes* in a classification task carry more sway than other less performant trees. The way the forest is constructed also differs from RF. In RF, trees can be built in parallel, because no tree affects the others in the ensemble. In AB, trees are built sequentially, since the performance of the previous tree determines the best questions to ask in the next tree. It has been shown by (Schapire & Singer, 1999) that the AB technique corresponds to gradient descent over a convex loss function.

A more detailed implementation of AB is outlined later, but in simple terms, the algorithm works in the following way:

1. Give each item in the training data initially equal *weight*
2. Determine the best question to ask in order to split the total weight of the data evenly into positive and negative classifications.
3. Classify all items in the data set using this question (called a *decision stump*).
4. Calculate the error, ϵ, as the sum of weights misclassified over the total sum of weights.

5. Update a parameter α using the error ϵ.

6. Assign the α parameter to the current decision tree. Those items that were misclassified receive a bump in weight, and those items that were classified correctly receive lower weight while training the next decision tree. This process is repeated until a forest of weighted decision trees emerges.

7. Classification problems are resolved by applying each decision tree to the object being classified, and taking the class with the greatest amount of weighted votes.

This algorithm in effect attempts to explain all edge cases in the data, by chasing misclassified samples via an increase in weight. *Weight* can be considered a measure of importance: how important is it that future rounds of training correctly classify the labelled sample? The magnitude of the sample's weight is the answer to this question of importance.

2.6 Word Vectors

Mikolov, Sutskever, Chen, Corrado, and Dean (2013) are often cited as the paper that marked the popular rise in the use of word embeddings for natural language processing applications. However, at this stage several implementations of word embeddings have been supplied, including another popular implementation named Glove, from Pennington, Socher, and Manning (2014). Word embeddings are vectors that embed semantic meaning for words in higher dimensional space. The training process is slow, and involves simultaneously training a neural network. However, once the training process is completed, the resulting word embeddings may be applied in many various tasks including synonym mining and analogy mining. One year after Mikolov et al. (2013), Le and Mikolov (2014) presented a performant embedding model for documents. This model again relied on a slow and non-sequential training process. For this reason, weaker document vectors, using pretrained word embeddings were chosen as the document embedding technique in this dissertation. This is necessary when corpus is open and indefinitely large.

2.7 Wikipedia

There is a long tradition of researchers mining data from Wikipedia in order to extract meaning (Nguyen, Matsuo, & Ishizuka, 2007; Nakayama, Hara, & Nishio, 2007; Lehmann et al., 2015). The hierarchical category structure is particularly useful. Nguyen et al. (2007) used this hierarchical structure, along with the first sentence in each article, to extract relations between entities. Two important *relations* mentiond in Section 1.1 are those of *is news source page* and *is homepage*. The text mining approach of Nguyen et al. (2007) will be used by `newsreduce.org` to carve out the collections of entities that satisfy these relations. Here, *relation* and *entity* are meant in the predicate logic sense, where an entity is some object in the world (real or imaginary): e.g. *CNN*, *Newspaper*, etc. A relation is some factual template that evaluates to true for an object, e.g. ____ *is an author*, ____ *is not a news source*.

A popular argument against using Wikipedia as a source of truth is of course reliability. Rector (2008) found that professional encyclopedias were indeed more reliable than Wikipedia, in terms of the number of inaccurate statements per randomly selected topics. But the cost of using anything other than Wikipedia at the time of writing is coverage. Intuitively, most local newspaper are unlikely to have an article in Encyclopedia Britannica. The encyclopedia would need to pay writers and editors from every local area to achieve that level of coverage. Wikipedia has high coverage as a result of worldwide volunteer collaboration. It is true that most articles can be edited by anyone, but there are also many quality procedures in place, and anyone can revert flagrant falsities as easily and as quickly as they were written. For very important articles, there are limits on who can edit. Finally, `newsreduce.org` will also use Pagerank to choose which sources take precedence (Sections 2.2 and 3.10). For this reason, vandalism on *stub articles*[3], leading to false positives or negatives in the two previously mentioned relations, are expected to have lower impact. For the reasons outlined, Wikipedia was chosen as the source of truth while addressing the questions of "What is a news source?" and "What is the news source's homepage?"

[3] *Stubs* are short, generally less relevant articles. `https://en.wikipedia.org/wiki/Wikipedia:Stub`

Chapter 3

Methodology

3.1 System overview

A broad overview of the `newsreduce.org` backend is provided in Figure 3.1. The connecting arrows between different components usually denote some form of inter-process communication (IPC). In the case of `newsreduce.org`, this communication is carried out most of the time via Redis queues and another feature within Redis named pubsub. Pubsub is not fault-tolerant, so wherever it is used, the process on the receiving end also relies on frequent polling of some queue holding output from another module. The modules are small services that generally read some information, and then pass derived information to other services. Message queues are used extensively throughout the system, but only the most important queue is represented in Figure 3.1 (the insert queue). Most services communicate with the database in some way, but this fact isn't presented in the diagram. The most important services are outlined in greater detail in their own sections (e.g. Sections 3.3, 3.4, 3.7), but some of the smaller services are briefly described next.

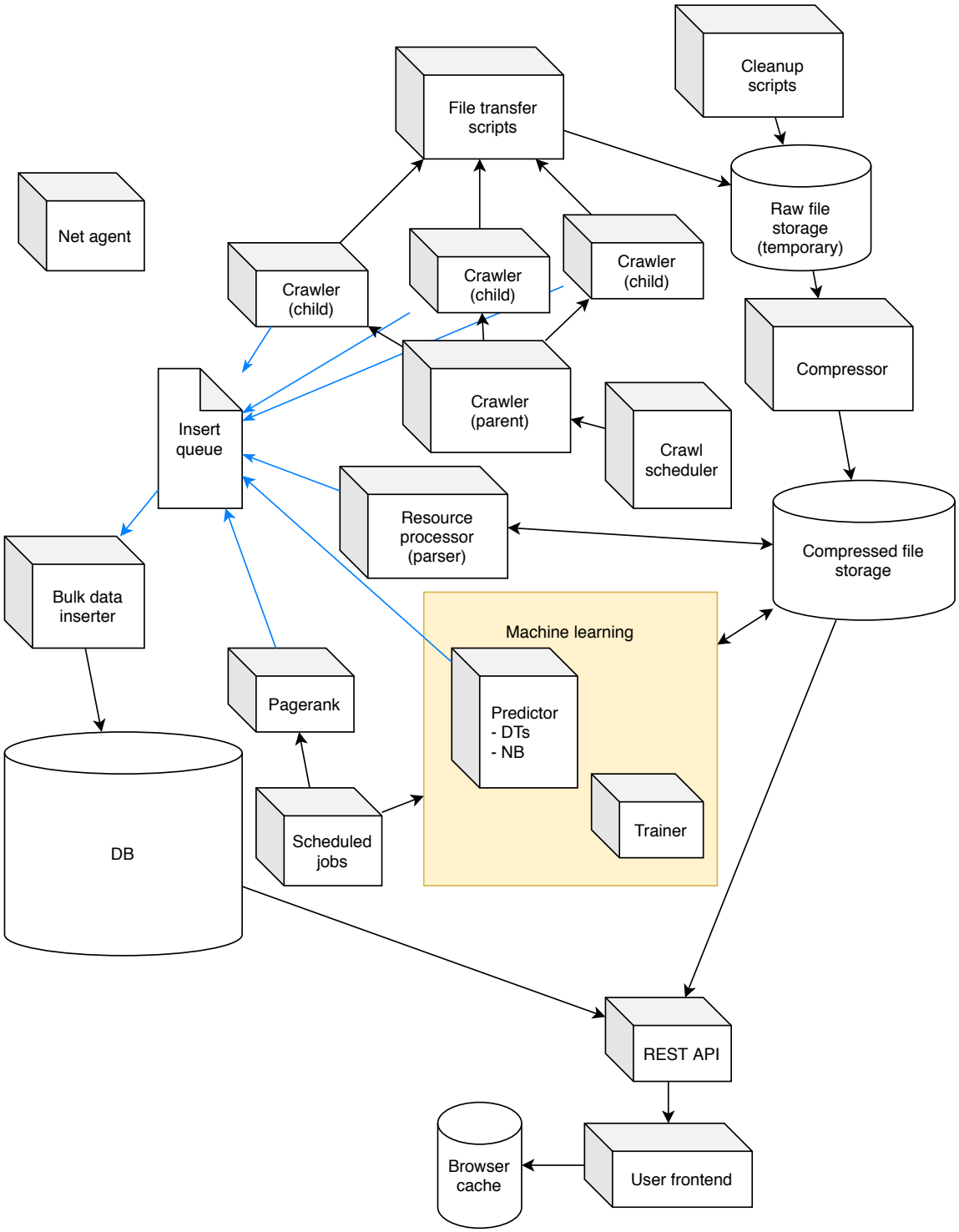

Figure 3.1: Broad overview of the system's architecture.

3.1.1 The inserter service

In order to crawl and process large amounts of news data, a strategy for inserting large amounts of data into a database was needed.

In the `newsreduce.org` system, rather than inserting rows into a database one at a time, in most cases, rows were appended to a Redis queue. This queue is polled at an interval of 400ms, and bulk insert jobs are created. Experiments were set up to compare the performance of a large SQL `INSERT` statement with that of a batch CSV import. The latter was faster, so the completed inserter service actually writes data to CSV files, and uses MySQL `LOAD` statements (every 400ms) to place data into relevant tables. This may not be wise when storing critical and inter-connected personal data, but for large amounts of corpus data, the few resulting inconsistencies were accepted. The following parameters were placed in a MySQL configuration file in order to speed up the DB server[1]:

```
[mysqld]
local-infile = 1
innodb_doublewrite = 0
innodb_buffer_pool_size = 20G
innodb_log_file_size = 1G
innodb_flush_log_at_trx_commit = 0
```

3.1.2 Hashing

Words, URLs and every other *entity* stored by the `newsreduce.org` database, 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 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 deduce 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.

[1]Parameters obtained from various answers on `https://stackoverflow.com`

Requirement 1 ensures that small programs can reason about data in the `newsreduce.org` system, without necessarily needing a database connection in order to obtain the ID of a word, 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 3.6, 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";`'. This costly ID check would need to be carried out for each word in the lexicon of documents being processed during a given 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 the same ID is assigned to two different objects). This problem is illustrated in Figure 3.2. 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 column on each table in this project uses the type `DEC(30)`, which is not as performant as an 8 byte `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` is recommended.

Using the hashing method outlined here with 8 bytes instead of 12, the probability of a hash collision over a database of 4,294,967,296 identifiable entities is roughly 39%. In

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

any case, a few hash collisions is arguably acceptable in the area of news aggregation (as opposed to, say, the area of banking).

3.1.3 Parsing and processing

After crawling various websites, HTML and headers are stored locally in files. But various other formats need to be derived from these two, in order to compare different machine learning language models, and choose the best model per task. Figure 3.3 illustrates these different formats. The storage strategy for these different formats is outlined in Section 3.4, and the motivation for different formats is explained in more detail in Section 3.5.

To parse HTML, the `JSDOM` library was used (Denicola, 2020). This library was chosen because of its similarity to the browser's builtin DOM manipulation and tree traversal functions. Special care was taken by the developer of the library to ensure backend developers could write code as though they are interacting with a real browser DOM, rarely with any difference. This allowed some code to be reused both for backend (server) and frontend (browser) programs. For example, in a later section, AdaBoost decision tree models are intended for application to documents via the user's browser. The model is stored within the browser's cache. More information about this process is provided in Section 3.12.

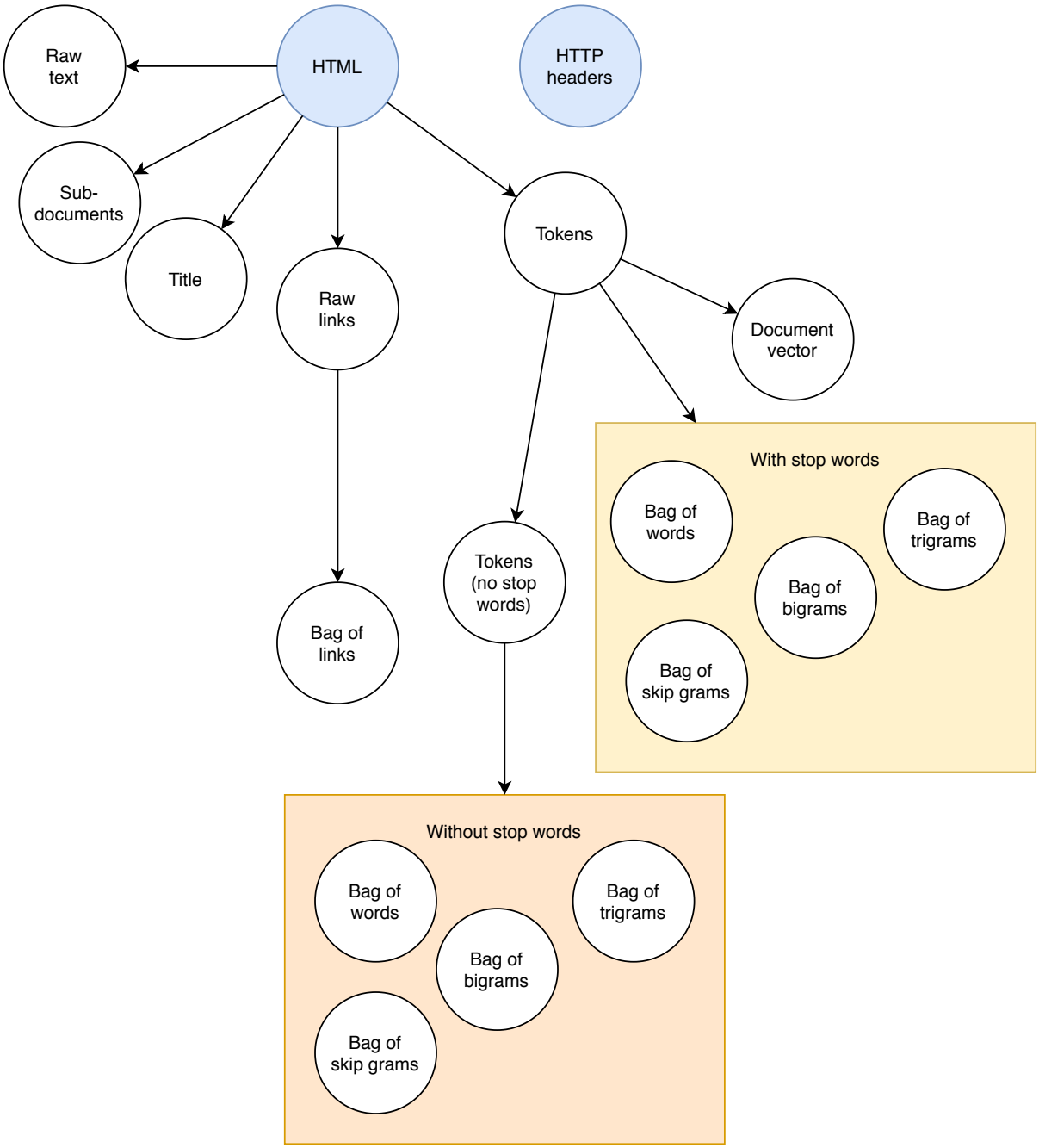

Figure 3.3: The process by which different experimental resource models are created. Blue nodes are models created by the crawler. The others are created later.

3.1.4 Net agent

A *net agent* program runs on each machine in the `newsreduce.org`'s network of machines. The program has two varieties, one for the parent node (where files and the database are

located), and another for child nodes, where crawling takes place. The program is used for system motitoring, propagating settings from the parent node, and other *housekeeping* tasks.

3.2 Scheduling

McBryan (1994) aimed to "tame the web" in a general sense, and many others have done the same since then. But `newsreduce.org` is only interested in news data, specifically recent news data. For this reason, a directed crawl of the web was desired, and a custom scheduler service was set up to accomplish this.

At regular intervals, the service schedules URLs to be crawled via custom `SQL` queries run against the main database. The queries only retrieve URLs for the following criteria:

1. Wikipedia category pages that are listed under other Wikipedia category pages.
2. Wikipedia pages that are listed under Wikipedia category pages.
3. URLs marked as news source homepages.
4. Same-origin links on a news source homepage.
5. Pages that are marked as news indexes.
6. Same-origin links on a news index.

The web pages to be crawled are inserted into a Redis *sorted set*. This set is polled in parallel by several crawler machines.

3.3 Crawling

In order to crawl large numbers of web pages fast, a distributed system of crawlers was set up, along with automation scripts written in BASH. The scripts allowed machines and processes 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.

Rather than some distributed, fault-tolerant filesystem, `newsreduce.org` simply stores all the compressed web pages on a RAID-1 file system. 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 was 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[2]. 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 and usage 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. But the disk and database limitations eventually led to the website's failure (Section 4.3).

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

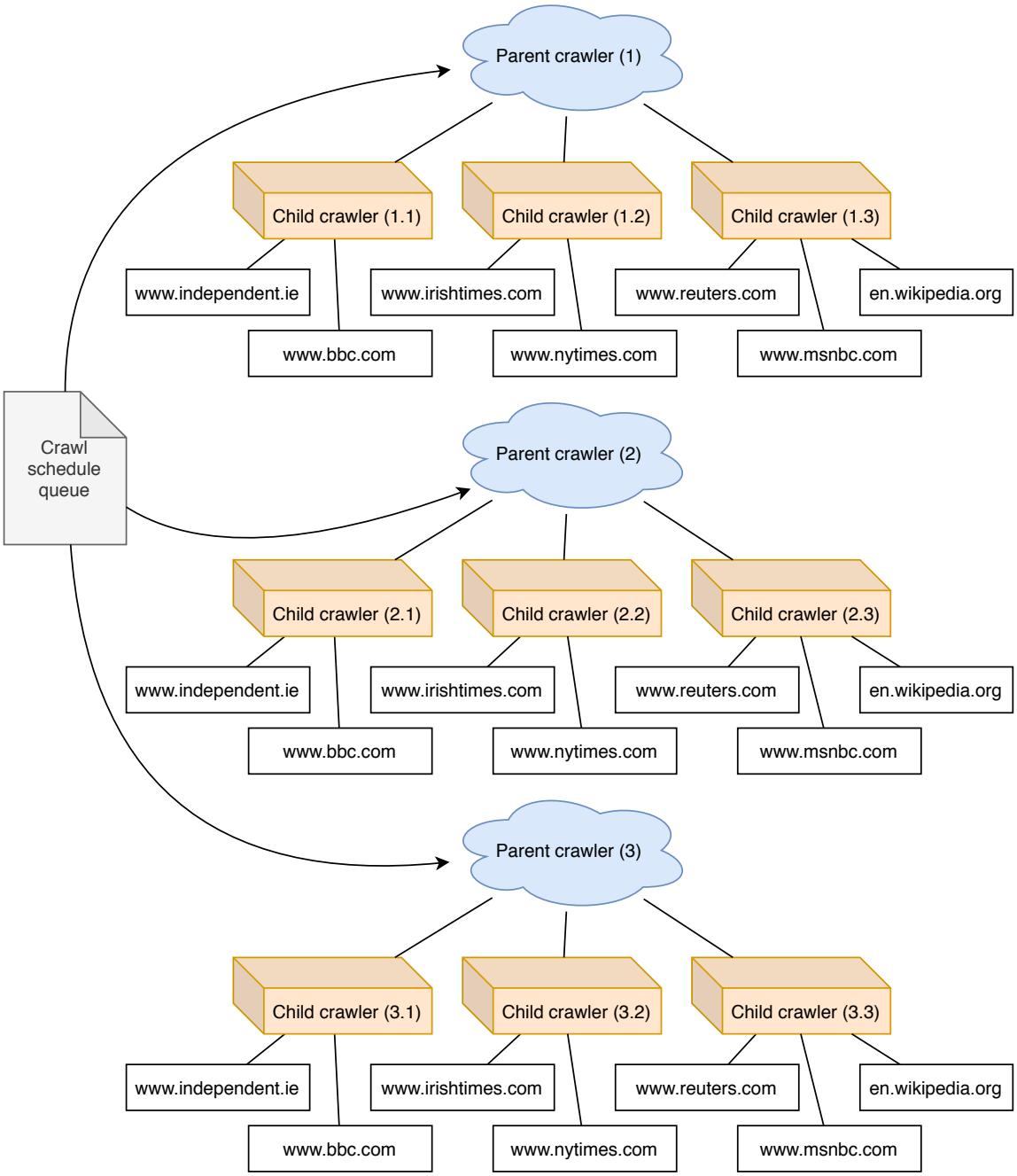

Figure 3.4: A closer look at the distributed crawler from Figure 3.1.

3.4 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

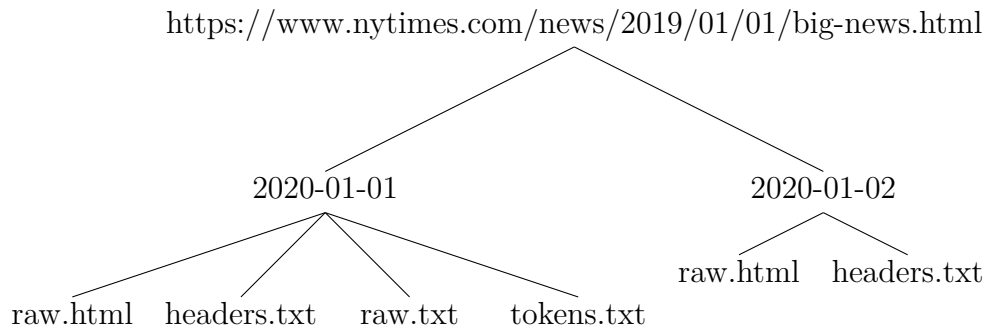

Figure 3.5: An illustration of a resource archive's hierarchical structure.

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/header 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 article 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 conversely, many versions of that resource per archive.

Figure 3.5 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 3.6 is a more realistic portrayal of a resource's archive. The archive is placed at a location resembling the URL's ID (Section 3.1.2). Under each archive, there is a file name, with a prefix of the version's timestamp, a suffix of version format name, and an underscore separating both parts. The different formats are explained in more detail in Section 3.5.

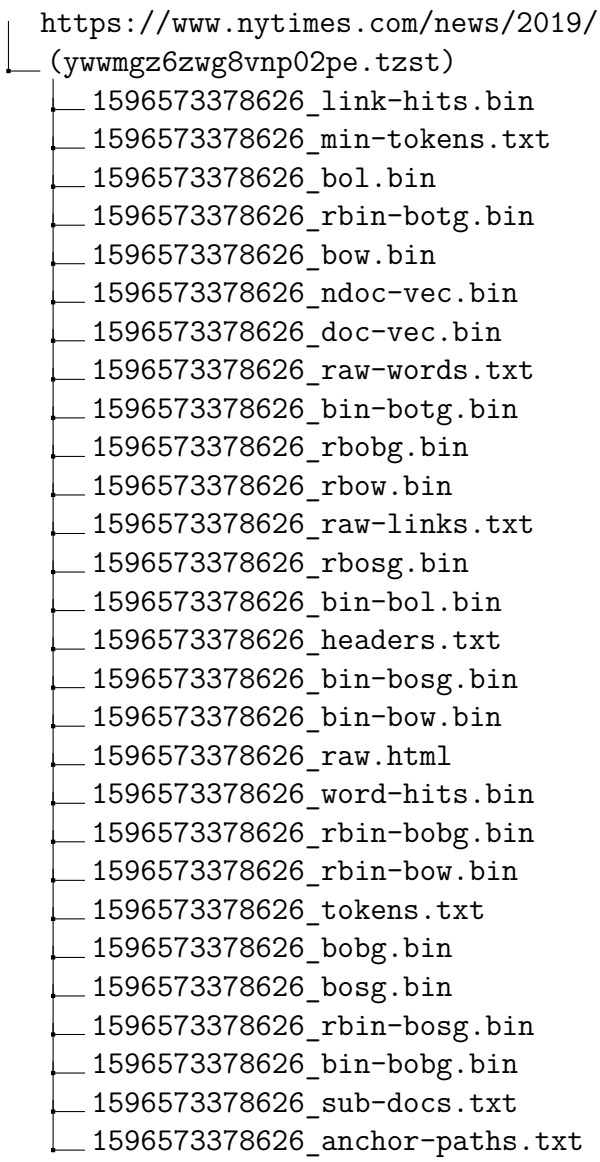

Figure 3.6: A realistic portrayal of a resource's archive.

3.5 Intermediate representations

As mentioned in Section 3.3, 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)

3.5.1 The sub-documents file

This file is designed specifically for extracting a particular piece of information (a relation) 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, in 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. Extracting relations may then be accomplished with standard document classification techniques, such as Naive Bayes or decision trees, by applying document classification to the sub documents rather than the documents themselves. This format turns out to be highly effective in the task of structured data extraction.

3.6 Data cleaning

3.6.1 Alphabet normalisation

Before constructing prediction models with the textual data from web pages (Section 3.7.5), 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 Unicode Consortium (2020) was parsed, and simple string lookups for 'LATIN A', 'LATIN B', and so on, were performed.

Figure 3.7 illustrates just a handful of the different forms latin letters can take. In most representations of resources, each of these forms is replaced with the ASCII letters. The same process is applied for numbers and 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 *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.

3.6.2 Tokenisation

Due to time and scope constraints, a rather simple approach to tokenisation was taken, whereby the input is assumed to be alphabetic rather than logographic or otherwise. Documents were split by paragraph, sentence, and word, and these parts were stored in a tokens file. Stop words were removed from the tokens file. These stop words were taken from a frequency table of common English words.

3.7 Machine learning

3.7.1 Supervised learning setup

Machine learning is an exciting field, but the data it relies on, and more specifically, the manner in which it is collected, is often the opposite. Primarily, the problems machine learning can solve tend to be those problems that are boring and routine for humans, for example, selecting 2,000 items at random from a set of Wikipedia articles, and manually labelling whether each article describes a news source. There are unsupervised solutions to the classification problem, based on clustering, but these solutions don't have the same

ASCII-form	non-ASCII forms
a	A a À Á Â Ã Ä Å à á â ã ä å Ā ā Ă ă Ą ą Ǎ ǎ Ǟ ā Ǻ ā Ắ ắ À…
b	B b Ḃ ḃ Ḅ ḅ Ḇ ḇ…
c	C c Ç ç Ć ć Ĉ ĉ Ċ ċ Č č Ҫ ҫ…
d	D d Ď ď Đ đ Δ D ḋ Ḍ ḍ Ḏ ḏ…
e	E e È É Ê Ë è é ê ë Ē ē Ĕ ĕ Ė ė Ę ę Ě ě Ǝ ə È è E ē E…
f	F f f Ф F f…
g	G g Ĝ ĝ Ğ ğ Ġ ġ Ģ ģ Ǧ ǧ Ǵ ǵ Ԍ ḡ…
h	H h Ĥ ĥ Ħ ħ Ḧ ȟ Ḥ ḥ Ḥ ḥ Ḧ ḧ Ḫ ḫ ẖ…
i	I i Ì Í Î Ï ì í î ï Ĩ ĩ Ī ī Ĭ ĭ Į į İ ı Ǐ ǐ Ỉ ỉ I ī I…
j	J j J̃ ĵ ǰ ȷ…
k	K k Ķ ķ Ḱ ǩ K Ḱ ǩ Ḳ ḳ Ḵ ḵ…
l	L l Ĺ Í Ļ ļ Ľ ľ Ŀ l· Ł ł Ḷ ḷ Ḻ ḻ Ḽ ḽ…
m	M m M Ḿ ḿ Ṁ ṁ Ṃ ṃ…
n	N n Ñ ñ Ń ń Ņ ņ Ň ň Ṉ ṅ N Ṅ ṅ Ṇ ṇ Ṉ ṉ…
o	O o Ò Ó Ô Õ Ö Ø ò ó ô õ ö ø Ō ō Ŏ ŏ Ő ő Ơ ơ Ǒ ǒ Ǫ ǫ Ǭ ǭ Ǿ…
p	P p Π P ṕ P ṗ…
r	R r Ŕ ŕ Ŗ ŗ Ř ř Ȑ ȑ R r̃…
s	S s ß Ś ś Ŝ ŝ Ş ş Š š Σ S ṡ Ṣ ṣ Ś ś Ṧ ṧ Ṣ ṣ…
t	T t Ţ ţ Ť ť Ṭ ṭ T T t Ṭ ṭ Ṯ ṯ ẗ
u	U u Ù Ú Û Ü ù ú û ü Ũ ũ Ū ū Ŭ ŭ Ů ů Ű ű Ų ų Ư ư Ǔ ǔ Ǖ ū Ǘ…
v	V v Ṽ ṽ Ṿ ṿ…
w	W w Ŵ ŵ Ẁ ẁ Ẃ ẃ Ẅ ẅ W ẇ Ẉ ẉ ẘ…
x	X x Ξ Ẍ ẋ X ẍ…
y	Y y Ý ý ÿ Ỹ ŷ Ÿ Y ȳ Y ẏ ẙ Ỳ ỳ Ỵ ỵ Ỷ ỷ Ỹ ỹ…
z	Z z Ź ź Ż ż Ž ž Z Z ẑ Ẓ ẓ Ẕ ẕ…

Figure 3.7: A few of the different letters that can be normalised to ASCII.

proven track record as supervised approaches, so the latter approach was used here.

In the task of determining what exactly is a news source, a random subset of webpages from a crawl of Wikipedia was fed through a browser frontend, with left and right keystrokes mapped to positive and negative classes. I then manually labelled two thousand of these articles over the course of two hours. This supervised learning setup can be achieved rather quickly using an `iframe` (to hold the web pages being classified), and some JavaScript code to cycle between different classification items.

Collecting training data for the relation extraction task was a little more complicated. For the task of classifying news index links, for example, various news sources with high Pagerank were visited manually, and any link resembling a news category was opened in a new tab. Firefox allows users to shift-select (multi-select) tabs, and store them all in a bookmark folder. Furthermore, the browser allows users to copy a bookmark folder as plaintext. These functionalities coincidentally provided an excellent tool for extracting training data for relations between links in hypertext documents. Classification over the sub-documents of web pages, as previously outlined, was then used in order to extract relations within documents. The same procedure was used to identify news article links on homepages, and official news source URLs on wikipedia articles.

3.7.2 Decision trees

As mentioned in Section 2.5, AdaBoost with decision trees was the algorithm of choice for `newsreduce.org`'s news personalisation.

Decision thresholds Depending on the task, different *decision thresholds* were used. Moving this threshold closer to zero increases recall at the cost of precision. Moving the threshold closer to one increases precision at the cost of recall. F1 tends to peak around the point where precision and recall are most similar.

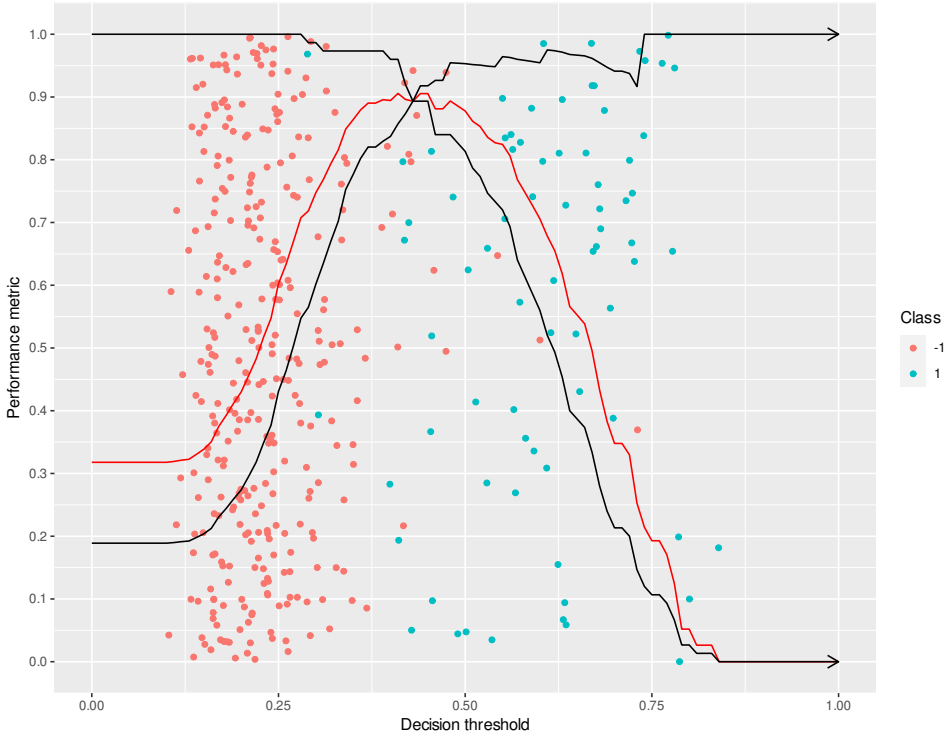

Figure 3.8: Comparing the effects of decision threshold choices (the descending black line is recall, the ascending black line is precision, and the curved red line is F1 score).

3.7.3 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 incomplete `newsreduce.org` application. For example, a classifier that determines if different newspapers might be from the same region uses Bayesian classification with a few pre-labelled newspapers. Although this needs supervised learning, the labelling is trivial, since `newsreduce.org` was intended to support news from 6 countries.

3.7.4 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 3.5.1, 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 the 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 (LD) between two strings (Levenshtein, 1966), and normalises 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:

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

Normalisation fills two purposes here. Firstly, it makes separate results from LD somewhat comparable. E.g. two 100,000 word texts with 6 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 decision tree implementation uses a default of 0 when a feature is missing from a document or sub-document.

3.7.5 Training

The training data for the decision trees was intended to be assembled from the *likes* and *dislikes* of the user, but due to time and legal constraints, this portion of the `newsreduce.org` system is yet to be developed. The website was intending to store this data on the user's side (Figure 3.12). The data would be assembled through interactions with upvote and downvote buttons displayed alongside each news item. Section 3.12 outlines how this could be accomplished without storing user data on a server, using web browser technology.

3.7.6 Specific implementation

In order to compare the performance of AdaBoost with Random Forest, and also with a hybrid of the two algorithms, an implementation of decision tree learning, with bagging and boosting, was programmed in C/C++ (code linked in Appendix A). The implementation is general enough to cover both Random Forest, AdaBoost, as well as the hybrid approach, depending on which parameters are used. A parameter for toggling boosting on or off is present (`-a` for AdaBoost). Another parameter is used to determine which ratio of the feature set should be considered as possible pivots at each decision tree node. In AdaBoost, this parameter would simply be set to 1. In standard Random Forest, a special value is used to indicate that the F features (for whatever size of F at a given node) should be sampled using a rate of $\sqrt{F}$. Other ratios between 0 and 1 may be used, but cursory experiments revealed that these static ratios were rather impractical compared with the standard $\sqrt{F}$. Another parameter sets the number of trees to train, and a yet another parameter determines the maximum depth to use per tree. The final parameter is the location of a file containing the training data in binary format. The training data arrives from a higher level language with access to the broader file system and database. This pattern of higher level languages spawning C scripts for heavier tasks was repeated in other scenarios: namely for PR and bulk k-NN. For more technical implementation specifics, readers are advised to navigate to the `src` directory within the code repository linked in Appendix A.

Random selection without replacement One of the challenges and benefits of using a low-level language like C comes in the necessity to allocate memory in a very thoughtful manner, generally near the entry point of the program. After an initial implementation of the decision tree learner was written for AdaBoost, it was extended to cover Random Forest, but for this, some implementations would use additional memory while determining which features to randomly select at a given node. To avoid this, an online algorithm for random feature selection (without replacement) was used, relying on the predicted frequency of gaps between random samples from a dataset. To obtain n random samples

from a population of N items, a random number generator need only be called n times, and no memory needs to be used in order to keep count of which samples have already been selected. The algorithm works like this:

To calculate the intial gap (between the left end of a list of length N, and the first random sample), a random number p generated between 0 and 1 is passed into a trinary function g, along with the sample size and population size:

$$g(p, n, N) = \begin{cases} 0 & \text{when } n = N \\ \log_{(1-n/N)}(p) & \text{otherwise} \end{cases}$$

The result of the function is later floored $\lfloor g(p, n, N) \rfloor$ to obtain a random gap length to the next sample. For large enough N, this function closely estimates the real probability distribution of various gaps, but when N is low, some further massaging of the graph is needed. This is due to the fact that the function g plots a continuous graph, whereas in reality, many gap lengths are not just improbable, but impossible. Only gap lengths between 0 and $N-n$ are allowed. Anything above that range would not leave enough space in the remaining population $N - g(p, n, N)$, and $n-1$ samples could not be subsequently obtained as the algorithm scans sequentially through the data. After a gap is selected, the subsequent gap is calculated with updated values for N and n. Namely, the next invocation would be $g(p, n-1, N-gap)$, where *gap* is the previous randomly selected gap length. This avoids the otherwise common difficulty when the random function overshoots the end of the list. The exact way in which the graph was *massaged* (i.e. stretched along the x-axis, and vertically shifted) is described by the source code in Appendix B. The effectiveness of this method was obersved visually by plotting randomly generated samples without replacement using this technique and a control technique: the memory-based lookup method outlined in the section on sampling from Cormen, Leiserson, Rivest, and Stein (2009). Figure 3.9 illustrates the results. It is hard to guess which technique involved the gaps approach. This level of reliability in randomness was sufficient for the task of Random Forest, and it is one of the implementation details that allowed for `newsreduce.org`'s decision tree learning algorithm to train many trees efficiently.

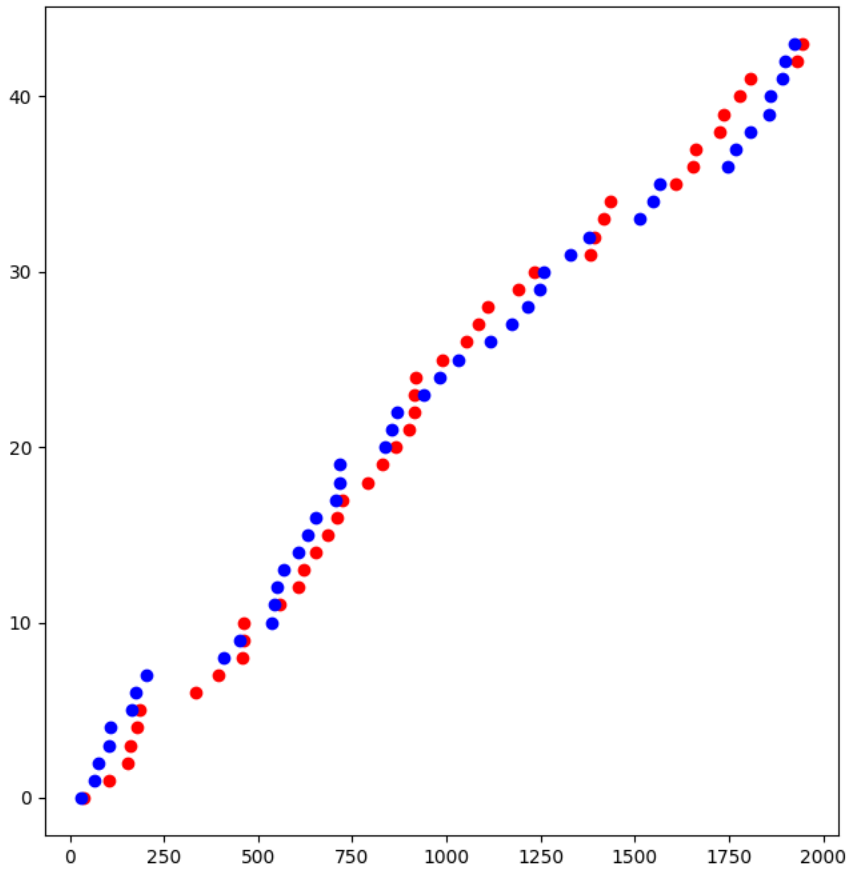

Figure 3.9: Random samples (without replacement) generated using the gaps approach and the approach from Cormen et al. (2009), in blue.

3.8 Clustering

A metric for article relevance was sought, and traditional nearest neighbour algorithms formed the basis for this measure. Using k-NN methods for small numbers of query documents yields k documents ranked by their cosine similarity to the given document. The cosine similarity refers to angular distance between the two documents plotted in a vector space (Section 2.6). There are many ways to place documents in a vector space. As discussed in Section 2.6, a *state of the art* solution trains document vectors on a large document corpus. This has drawbacks, however, since it relies on knowing the documents

that are being clustered in advance, and results from the algorithm are only available at the end of execution time (several hours). In other words, the state of the art is not an *online* algorithm.

`newsreduce.org` used pre-trained word-level vectors to produce document vectors on the fly. This way, a document vector can be calculated using only the document and the static set of word vectors. Each word in the document forms a piece of a weighted aggregate vector. Words nearer to the beginning of the document play a stronger role in the overall document vector than words closer to the end. This dampening was achieved using survival function formulae. The motivation for using these formulae comes intuitively from the notion of reader attention. The model assumes most readers don't *survive* to the end of the document, and that documents contain more important information nearer to top. These word vectors are referred to as *weak document vectors* to distinguish them from higher quality word vectors bulk-produced using strong neural networks.

The efficacy of the weak document vectors were tested by applying them as the singular feature source in AdaBoost document classification. Although the resulting F1 scores were lower than when all feature sources were used, the score was still sufficiently high (roughly 75% on the test set) to determine that semantic meaning was present in the weak document vectors. These document vectors could then be used in order to cluster documents together by cosine similarity. Rather than finding k nearest neighbours to a given document or small set of documents, a heavier task was undertaken, determining similarity scores across all news resource pairs crawled in a set time period.

3.9 Bulk k-NN in parallel

A naive solution, when applied to 1,000,000 vectors, would take roughly 57 hours to complete (based on observations taken using an X1 ThinkPad laptop), longer than a news cycle. This naive approach would iterate through the cartesian product of the dataset, and find the similarity score for each tuple pair.

An improvement would be to split the dataset into P chunks (where P is the number

of processor cores available), and calculate the similarity for all possible tuple pairs in respective chunks. After this first stage, each chunk would be left *calculated*, i.e. the similarity between each tuple pair from the chunk would be known. All cores would be used in the initial production of P chunks, but in subsequent steps, when similarities across disparate chunks are sought, the core utilisation would decrease. In the final stage, when two large chunks are being merged, only one core would be used. This problem is comparable to the diminishing returns observed in parallel implementations of divide-and-conquer sorting algorithms (merge sort and quick sort).

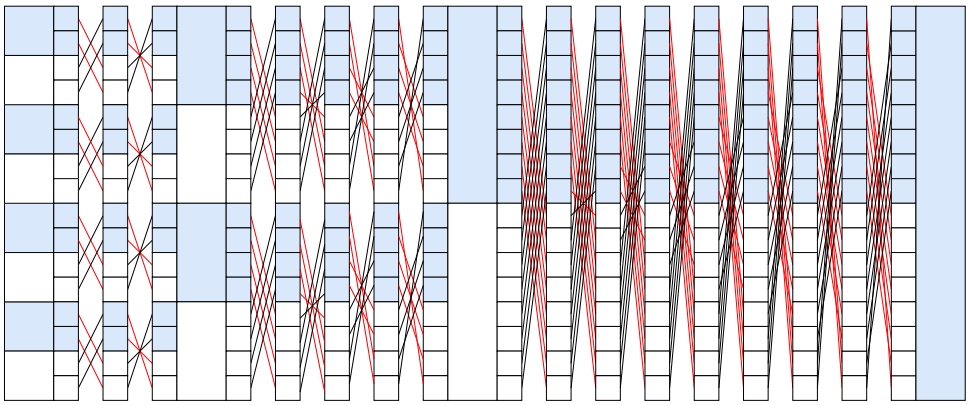

Figure 3.10: An illustration of the parallel execution pattern used in bulk nearest neighbour processing.

`newsreduce.org` uses a bulk nearest neighbour algorithm that ensures all cores are being used at all stages of execution. The formal foundation of the algorithm is divide and conquer. Figure 3.10 attempts to illustrate the algorithm's execution, but a more formal definition will be given later. Two chunks are *merged* into one chunk by splitting chunks A and B into equal *subparts*, and calculating the similarities between items from each combination of subparts from A and B. Red lines signify the process of *merging* two chunks. Merging effictively turns two chunks into one chunk, where the previously mentioned *calculated* property holds true within that chunk. The black lines are present only to illustrate the reflexive nature of calculating similarity; the cosine similarity between vectors V and U is the same as the cosine similarity between vectors U and V.

The implementation for this (and other processor-heavy tasks) was written in C/C++.

The codebase URL is provided in Appendix A, but a short description of the algorithm, and its formal verification is provided here.

Firstly, the task must be formally described. The input to the algorithm is a set D of fixed-length vectors. This length is popularly set to 300 when dealing with word vectors. Here, a vector, for those readers leaning more into the field linguistics, is simply an ordered set of real numbers (a tuple), for example $\langle 0.01, -0.095, 0.567, -0.03 \rangle$. These numbers are considered dimensions in a high-dimensional space, and semantic information is embedded in the position of items within this space. $|D|$ could be in the order of millions, though the number of dimensions for items $d \in D$ tends to be low enough for practical purposes.

A bulk k-NN function maps each item d to a set of k documents d'_k with maximum similarity scores to the the document d. Simultaneously, one variant of the algorithm should gradually calculate mean and standard deviations for similarity scores, so that significance of similarity scores can later be determined. In a system of only one dimension, transitive properties of the similarity score relation could be leveraged in order to speed up the calculation. In other words, the dataset would essentially be a list of numbers, which is then sorted, revealing the implicit similarity between items. When things go into 2 dimensions, 3 dimensions, and other low dimensions, k-d trees could be applied, in order to narrow down the search space of similar vectors. Unfortunately, when dimensions get as high as 50 or 300, k-d trees become ineffective, and a sequential search performs just as well in an asymptotic sense, in fact with lower overhead. The naive algorithm first introduced in this section has $O(n^2)$ running time, and the aim of the parallel algorithm proposed here is to reduce that figure to $O(n^2/P)$, with little overhead.

Before explaining, more terminology is needed. A *similarity group* is defined as a collection of vectors d for which the similarity, between any vector d' in the group to any other vector d'', is known. I.e. the cosine similarity between the vectors has been calculated, and integrated into a results data structure, if appropriate. Naturally, any set of vectors with a cardinality of one can be considered a similarity group. The predicate SG is used here to describe when a set is a similarity group. Two vectors can be considered a similarity group if one calculation has occurred: $S(d', d'')$. This is a predicate

which indicates that the similarity between the two vectors has been calculated already. Any larger set of vectors (cardinality above two) can be defined as a similarity group recursively, if some element d' in the set forms a similarity group with all other elements, $\forall_{d'' \neq d'} S(d', d'')$, and if the set without d' is itself a similarity group: $SG(D - \{d'\})$. This recursive definition relies on the fact that a set of cardinality below 2 is intuitively a similarity group:

$$SG(D) \text{:-} \begin{cases} T & \text{when } |D| < 2 \\ \exists_{d' \in D} \forall_{d'' \neq d' \in D} S(d', d'') \wedge SG(D - \{d'\}) & \text{when } |D| \geq 2 \end{cases} \tag{3.2}$$

Using the naive sequential algorithm with P threads across P splits of the data results in P similarity groups. This stage in the algorithm corresponds with the eight leftmost blocks in Figure 3.10. All blocks in the diagram represent similarity groups. These groups carry a *closure property* under subset, meaning that any subset of items from a similarity group can also be called a similarity group, $SG(D) \implies \forall_{D' \subseteq D} SG(D')$. Two similarity groups D and D' can be merged into a set that can be considered its own similarity groups only after certain calculations are applied across component-wise across items from each of the sets, in other words when the following logical sentence holds true: $\forall_{d \in D} \forall_{d' \in D'} S(d', d'')$. In English, this means when every possible ordered paired of items from D (the first item in the pair) and D' (the second) have been considered. In programming, this is achieved with nested loops. But in order to ensure that all processors are being utilised, the merging of two similarity groups must use two processors in the first stage, instead of just one. In the second stage (turning four vertical blocks from Figure 3.10 into two vertical blocks), four processors must be applied per merging tasks. For this, each block is split into 2^n evenly sized sub-blocks (n begins as one, and increments at each stage of the algorithm). The larger block from which the sub-blocks are split is called the parent block. To merge two parent blocks optimally using P processors (for some P being a power of two), all possible merge operations, from all of one of the parent's sub-blocks to all of the other parent's sub-blocks, must be performed (the red

lines in Figure 3.10). But these merge operations can now be applied in parallel, and if applied in the executio order shown in Figure 3.10, currency bugs will not occur. This is explained by the fact that at no stage in the algorithm are two red or black lines ending or beginning on the same block. In practice, this means that at any given time, no memory is shared across threads. Each thread is operating on its own window (a sub-block pair) of the data. For more detailed insights, the code is available, and links are provided in Appendix A.

The running tallies of k nearest neighbours are maintained during execution using a max heap, and two figures (mean and standard deviation) are gradually built up as the algorithm progresses. After implementing this parallel algorithm, the bulk k-NN calculation predictably completed in one eighth the amount of time ($\approx$ 7 hours) on an 8-core ThinkPad laptop.

3.10 Ranking

A definition of PR PR (Page et al., 1999) is explained in simple terms in Section 2.2, but a more specific specification should also be outlined. The implementation of PR used by `newsreduce.org` treats webpages as nodes p in a graph G, such that $p_i \in P$, $1 \leq i \leq |P|$, $i \in \mathbb{N}$. 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 functions 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 webpages that link to a given webpage (otherwise known as *backlinks*: indexes to the pages that link to the webpage at index i). F returns the *forward links*, simply 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 PR 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, PR can be defined recursively:

$$R(i,t) = \begin{cases} \frac{1}{|P|}, & t = 1 \\ \sum_{j \in B(i)} \frac{R(j,t-1)}{|L(j)|}, & t \neq 1 \end{cases} \tag{3.3}$$

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

Algorithm 1 Simplified PR algorithm

Require: T, an error threshold, needed to halt PR.

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: $finalRanks \leftarrow previousRanks$

manipulation through rank sinks. As mentioned in Section 2.2, one solution was found

Region	PR 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 3.11: PR escape links by ISO-3166-2 country code.

through a finite set of *escape* links, or *sources of rank* (Page et al., 1999). These links are chosen purposefully in various invocations of PR, 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 3.11. To simplify things, only 6 English-speaking countries are currently supported. These countries were chosen by population size.

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 PR (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.

3.11 Trend ranking

A major problem in applying PR on a corpus comprising primarily news data is the tendency within the news industry not to reference other articles that reported on a story first, or at least the tendency not to reference using an external hyperlink. Possible reasons for this tendency may include the speed at which articles must be published in the competive news industry. In any case, a solution to this problem was sought.

PR remains a superb solution once some link structure between resources can be established. In order to establish this link structure, a design for an *inferred links* approach

Algorithm 2 PR 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.

for each $webpage \in webpages$ **do** ▷ Initializing variables
  $previousRanks[webpage] \leftarrow \frac{1}{|webpages|}$
end for
$error \leftarrow \infty$
while $error > T$ **do**
  for each $webpage \in webpages$ **do**
    $nextRanks[webpage] \leftarrow 0$
  end for
  $redistribute \leftarrow 0$
  $redistributeRatio \leftarrow 0$
  $decayRate \leftarrow \frac{1}{2\times 85}$ ▷ Set with the expected iterations (in this case ≈ 85).
  for each $webpage \in webpages$ **do**
    $previousRank \leftarrow previousRanks[webpage]$
    if $|webpage.links| > 0$ **then**
      $rankPiece \leftarrow \frac{previousRank\times(1-redistributeRatio)}{|webpage.links|}$
      for each $link \in webpage.links$ **do**
        $nextRanks[link] \leftarrow nextRanks[link] + rankPiece$
      end for
      $redistribute \leftarrow redistribute + previousRank \times redistributeRatio$
    else
      $redistribute \leftarrow redistribute + previousRanks[webpage]$
    end if
  end for
  for each $webpage \in escapeWebpages$ **do**
    $nextRanks[webpage] \leftarrow nextRanks[webpage] + \frac{redistribute}{|escapeWebpages|}$
  end for
  $error \leftarrow variance(previousRanks, nextRanks)$
  $tmp \leftarrow nextRanks$ ▷ Swap *nextRanks* and *previousRanks*
  $nextRanks \leftarrow previousRanks$
  $previousRanks \leftarrow tmp$
  $redistributeRatio \leftarrow redistributeRatio + decayRate$
end while
$finalRanks \leftarrow previousRanks$

is proposed (though not implemented), that relies on the bulk k-NN algorithm outlined in Section 3.9, along with the PR algorithm outlined in Algorithm 2. The variant of the bulk k-NN algorithm that simultaneously calculates means and standard deviations is used. Once the k nearest neighbours to each news item (represented as weak document embeddings) are calculated, those with a similarity score high enough (some multiple of the standard deviation outside the mean) are considered for potential inferred links. An inferred link can only be placed from a news item n to another news item n' if the publication date (or crawl date) of n occurred after n'. Some threshold could be set in order to exclude temporally distant news items from consideration for inferred links. Finally, PR could then be applied to the graph formed from the inferred links, using initial web page weights obtained from a more traditional PR. This proposal remains to be implemented, due to the difficulties encountered while completing the large scale crawling task (Section 4.3).

3.12 Privacy

As introduced, a privacy-oriented solution to personalised news aggregation was sought, and the following system was designed in order to achieve this (Figure 3.12). The system design relies on modern browser and phone app technology, and particularly the ability of these technologies to store relatively large amounts of data when compared with their earlier versions.

A web browser can now store 10MB of persistent data in a variable named `localStorage`, which is accessible via JavaScript embedded in HTML documents. This combination of a programming language, GUI formatting language, and access to newly increased sums of persistent storage, effectively transforms the browser from its original purpose as a linked document navigator, into something comparable to a small-scale operating system. Others have leveraged this with excellent results. The website `lichess.org`, for example, applies a Stockfish implementation for chess game analysis, entirely within the browser.

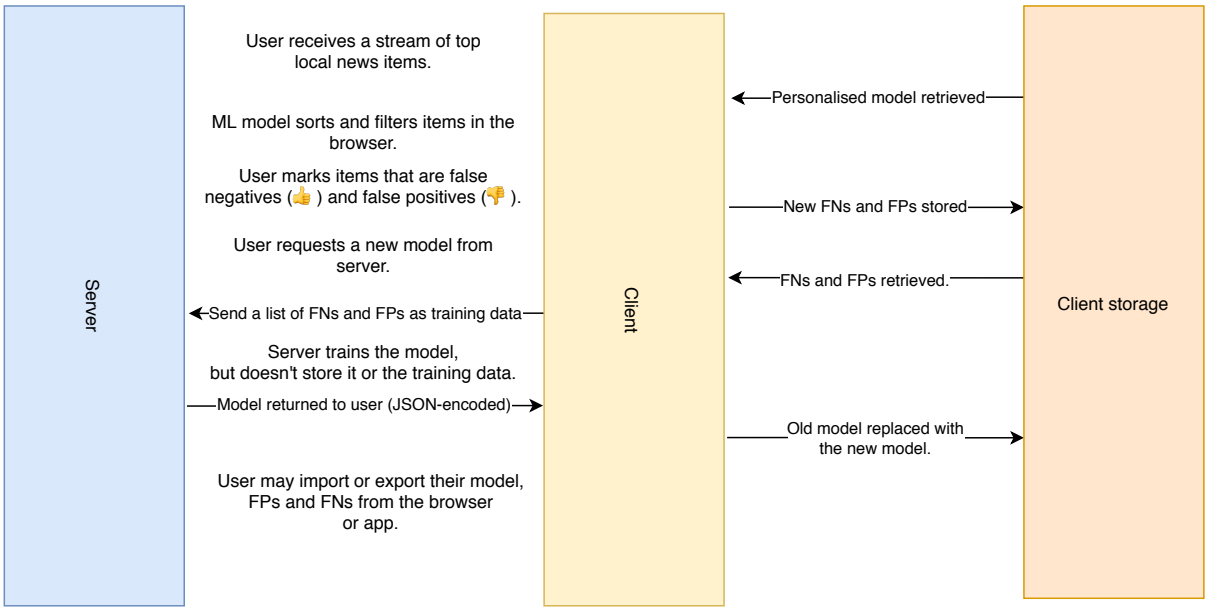

Figure 3.12: A privacy-oriented approach to machine learning

The proposed design of `newsreduce.org` focussed on more of a hybrid approach, in which model training remains a task of the backend server, but model storage and application is moved to the frontend. This minimises the role of the server, leaving data primarily in the hands of the user (stored in the `localStorage` variable). Arguably, this system could be misused by businesses, and user data could be leaked when received by the server. One possible solution to this, without moving the model training entirely to the browser, would be to entrust the browser with the construction of a *training input file*, with data anonymisation applied to it. The anonymisation process could simply replace the features with sequential numbers, and store a map of these numbers to the original features in local storage.

The training input file sent to the server would be implemented to work immediately within a backend decision tree learning algorithm, without needing to first translate training data (in the form of *likes* and *dislikes*) into a more suitable format. This would leave backend services entirely blind to the meaning of the data for which they are building a model. The browser, on receiving newly trained models, would then be responsible for de-anonymizing the model into the original features located at tree nodes.

Section 3.7.6 outlines a program which has been designed to train decision tree forests on arbitrary classification problems. The program is agnostic to whether the data is derived from written language, spoken language or any other data source, so working with these anonymised features should have no impact.

Chapter 4

Results and discussion

4.1 Classifier comparison

Comparative results for the task of identifying news articles that correspond with news sources are presented in Figure 4.1. The hybrid approach mentioned used boosting along with random feature selection. This technique speeds up the building process significantly, at the cost of precision and recall. Initially, an AdaBoost decision tree learner was prototyped in TypeScript (Figure 4.3), but this implementation didn't allow for the scale of comparative testing necessary for choosing the best ML models per task. The benefits of the efficient C implmentation are reflected in Figure 4.1, and in the fact that initially, deep AdaBoosted decision trees seem significantly better than AdaBoosted decision stumps in the classification task. As training time progresses however, the F1 scores for the stumps approach those of the other techniques, and in some instances surpass the performance of the deeper decision tree forests. Furthermore, training decision stumps with AdaBoost was by far the fastest machine learning algorithm in the comparison.

The results are even clearer in Figure 4.2, when analysing the task of extracting official homepage links from Wikipedia articles. In this case, the decision stumps surpass the performance of depths 2 and 3 somewhere between 50 and 100 trees. An explanation here may involve training data overfitting, a common problem in deeper decision trees. The original and inefficient prototype decision algorithm used to generate Figure 4.3

would not have spotted the eventual success of decision stumps, as experiments were limited by running time. Several random splits of test and training data at a ratio of 60% training data were used to obtain these figures. In the case of Figures 4.1 and 4.2, trials were continued until aggregate data produced the comparatively smooth lines and curves observed in the two figures. This level of certainty with regards which models are best in which tasks was not so clear when the number of trials were low (in Figure 4.3).

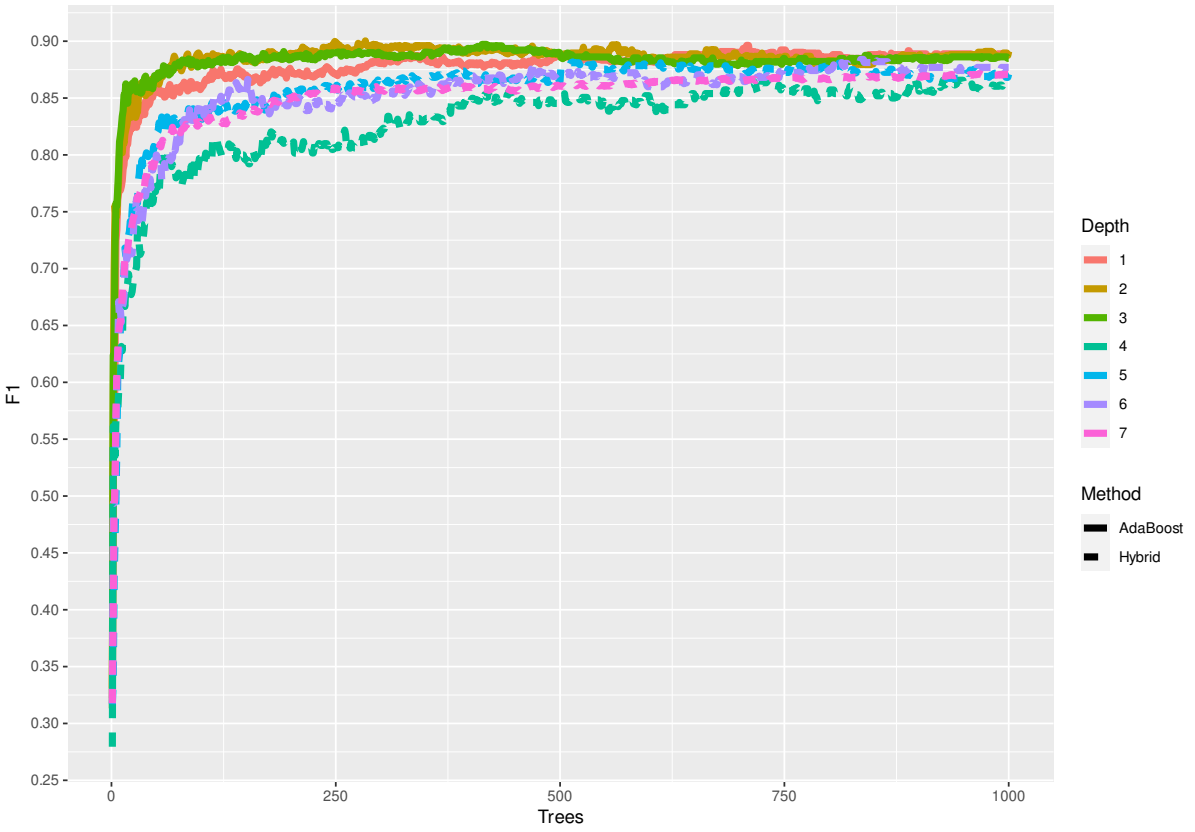

Figure 4.1: A comparison of F1 scores across different max tree depths, methods and forest size (binary classification)

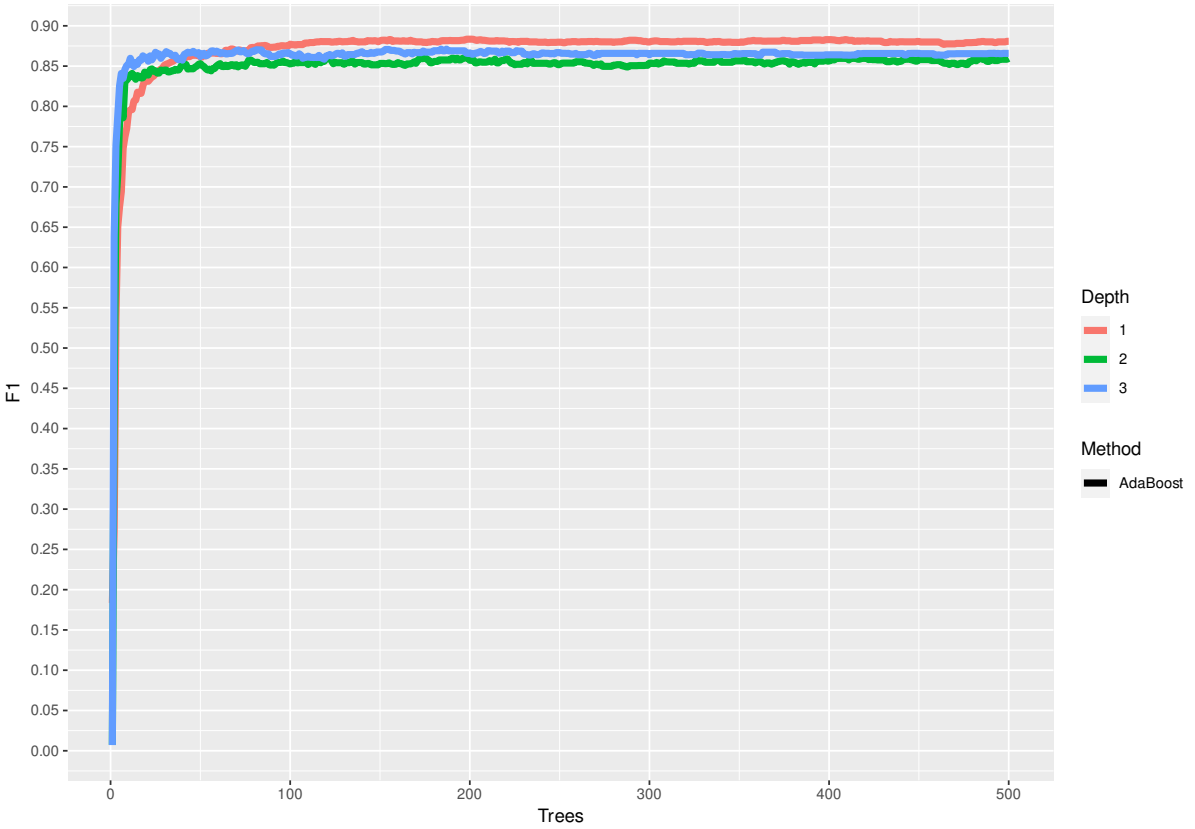

Figure 4.2: A comparison of F1 scores across different max tree depths, methods and forest size (relation extraction).

4.2 Feature sources

Figure 4.3: A comparison of early precision and recall scores when different combinations of feature sources were used.

Figure 4.3 illustrates the lower performance when relying entirely on the weak document vectors introduced in Section 3. These were vectors built only from the pretrained word vectors comprising documents, aggregated and weighted by relative position in the document. Despite this lower performance, fluctuating around 65% recall and 85% precision, it was enough to demonstrate the value of simple yet weak document vectors in various natural language processing applications. For reference, a classifier that always returns true would have had a precision score of 18% [1].

This particular experiment formed the basis for why weak document vectors were chosen as the building blocks of the *trend rank* algorithm proposed in Section 3.11.

4.3 Discussion

In the task of categorising wikipedia articles as news sources, and mining text to discover *official homepages*, there were good results, as measured by the high precision and recall

[1] 18% of articles from the media-related subsection of Wikipedia crawled corresponded to news sources

scores in both of these tasks (observations in Section 4.1). Arguably, the most important results in this dissertation came in the form of the tasks that failed to reach implementation, notably the task of implementing the proposed *Trend rank* introduced in Section 3. Discussing the reasons for these failures may aid future research in this area.

Problems The primary cause of failure in the implementation of `newsreduce.org` was in the reliance on the model of a single database server situated on a single machine. This was predicted to be a problem, but the choice of a single database was made in order to reduce costs and thereby reflect the realities of a private cloud user's situation; people are probably not going to pay hundreds or thousands of dollars a month to achieve the result of *Google News*, but with added privacy.

Even after several performance enhancements were applied (Section 3.1.1), clear bottlenecks began to emerge when the crawling task was expanded beyond the boundaries of Wikipedia, and into the web of news content. `newsreduce.org` can be considered a system of queues, with data passing from one queue to the next after different rounds of processing. This is very useful for debugging and fault tolerance, since failures can be explained by inspecting the data sitting in particular queues (in this case, queues are Redis sets). Additionally, suspended processes can be resumed without data loss. Unfortunately, as is the case outside of computers, queues sometimes have a tendency to get very large. The problem is made worse when the flow of data through different queues forms loops. These loops cannot be avoided in the crawling task, which is inherently recursive.

Shortly before `newsreduce.org` was taken offline permanently, there were backlog queues over 800,000 items long for various high-frequency tables, such as the links table, and the URLs table. Due to cost restrictions, a 2TB HDD[2] was used to host the database server. In other words, the predicted speed boosts of modern solid state drives didn't come into effect. When data is continually being inserted into a database housed on a mechanical harddrive, many other servers and applications on a machine start grinding to a halt.

[2]mechanical hard disk drive

In order to keep costs low, the database, many microservices, the blob file storage, and the planned frontend, were all housed on one primary server, with 8 CPU cores and 32GB of RAM. This was rented for approximately €33 a month from `hetzner.de`. The crawling task was distributed, but data processing and machine learning tasks were not distributed across machines, since the system was relying on a traditional single disk filesystem, and distributing data processing tasks in that case would only replace disk IO problems with *later* disk IO problems.

The focus on compression was not unfounded. Before `newsreduce.org` was permanently shut down, approximately 13GB of disk storage was filled by compressed HTML and related resource representations. This figure would have stretched to at least 95GB if compression was not used, and the project was shut down not too long after it began crawling the public web, so compression increased the maximum time the single disk approach would have remained feasible by approximately seven-fold. After this, pruning old file versions would be necessary.

Compressing different resource versions in the same archive contributed to the high compression ratio (approximately 1:7). However, compressing each resource led to yet another bottleneck in the machine learning stage of the project, the Pagerank stage, and in the general pre-processing stage, when intermediate resource representations were being generated. The problem lies in the need to compress and decompress resource archives whenever a file is being read or written. Versions of the same resource (e.g. a BOW representation and a links file) could not be written concurrently. Firstly, this caused many bugs, the majority of which were solved, but at a huge development time cost. Secondly, this forms a compression bottleneck. Similar to the database issue, compressing at scale uses yet another data queue, a queue that is prone to overfill when the machine is under strain (e.g. IO strain from a database running behind on insertion tasks). `rsync` was used as a low cost means of shuttling files from low-cost crawler machines to the main machine. Compressing on the main machine was paused during this process, and similarly, during compression, `rsync` was paused.

Alongside the database and the file storage design choices, difficulties arose from

the choice of which file formats to generate (Figure 3.3). Early on, a choice was made to generate many different intermediate representations of HTML resources, in order to investigate which representations perform best on various classification and ranking tasks. In the end, only a handful of these representations were used in classifying, clustering and ranking, namely the tokens file, a list of URLs, the sub-documents file (which proved very useful in the task of structured information extraction), and a document vector, generated from the tokens file. Having residue formats wouldn't be such an issue in small-scale data science projects, but in a live production systems using exponentially growing web content as input, it certainly led to additional disk bottlenecks. I would advice future researchers to choose smaller file formats and language models while working with open corpus environments like the web, even with today's faster disks.

Many of the centralised design choices were made in the interests of keeping costs and barriers of entry low, so that a *private cloud* solution to news aggregation could be offered to general users. Commonly, a private cloud application will run on a single server, with a straightforward installation frontend. This is the case, for example, with Wordpress (blogging), Syncthing (personal file storage), Nextcloud (A Google suite alternative). These services can be difficult to set up, but nowhere near the difficulty of setting up a huge web of microservices, and a distributed, fault-tolerant file system, such as Apache Hadoop.

Crawling and news aggregation are two incredibly time-sensitive tasks. If data doesn't go from websites to the user in a matter of hours, or perhaps minutes, then a modern news aggregation service has failed the task of aggregating news. In hindsight, I would warn those who are interested in applying algorithms to language: Chasing open corpus problems can lead a researcher down a programming and code maintenance route, never fully reaching the goals of doing what they would like to do with the data they are collecting.

There are so many technical roadblocks in the task of crawling, it proved difficult for one developer in 2020 to overcome them while on temporary work sabattical. Many of these problems are trivial to solve (in terms of programming complexity), but the real

difficulty is in the quantity of the problems. For example, after implementing Pagerank, I discovered that many URLs have a huge number of synonyms. The page's rank would collect in the most popular synonym. This wouldn't be a problem if rank were only used in web search results. But Pagerank was also integrated into the crawling algorithm used by `newsreduce.org`, in order to crawl the most important pages first. I spent some time wondering why `http://www.nytimes.com` was way down at the bottom of the crawl schedule, before realising it was because another URL using the SSL prefix (`https://www.nytimes.com`) had taken most the rank. Defects like this one usually take at least half a day of development time to resolve. In this example case, the solution would be to set up the notion of a 'URL group', and reaccumulate ranks from individual resources into their associated URL group instead.

These gradually emerging defects can slowly start taking up a large portion of research time. This is why I decided to shut down the `newsreduce.org` website roughly midway through the methodology, to focus my efforts on outlining algorithms in an abstract way, and to present the challenges I faced to others researching in this area.

More positive aspects Before undertaking this project, I wasn't quite sure how large an undertaking it was. The journey along the way has had many positive results. Firstly, an implementation of a hybrid decision tree learning algorithm was provided. The efficiency of this part of the `newsreduce.org` system allowed many trials to be carried out, with varying training parameters. This was used in order to demonstrate the sufficiency of decision tree stumps in text classification, versus deeper trees using otherwise identical bagging and boosting techniques.

Secondly, a methodology for structured data extraction from HTML documents (author names, dates, etc.) using a generated sub-documents file was outlined. This method was shown to be effective in the selected task of recognising news source homepages on a wide array of Wikipedia articles, including stubs. A corpus of news sources, mapped to their homepages, and a probability of correctness, was produced. This data could be useful in further research, or in the text mining industry.

A parallel implementation of bulk k-NN calculations over vectors with high dimen-

sionality was provided (Figure 3.10). This bulk process forms the basis of a proposed novel method for ranking news items not only by link structure, but by inferred link structure. This could be valuable in environments where link citations are rare, not only in news, but in message logs and emails.

Chapter 5

Conclusion

The original intention of this dissertation was to design, implement and deploy a news aggregator with coverage similar to *Google News*, but with a focus on privacy. In this goal, there was limited success. This dissertation outlines what didn't work, as well as what did work. Firstly, the monumental goal of personalised news aggregation was broken down into smaller tasks. Solutions to each task, based on background research in similar areas, were then attempted. The tasks of news source discovery, retrieving official homepages in a probablistic manner, and ranking, were addressed successfully, and several improvements to various algorithms were implemented or proposed along the way (section 3).

However, the huge task of implementing a personalised and self-hosted news aggregator was blocked by several unforeseen barriers. These barriers were documented in Section 4.3, so that others might have more success in future, or simply avoid the challenge entirely.

It is possible that a cheap and private server solution for intelligent news aggregation can be developed. But the results of this dissertation and project cast doubt on the possibility of achieving this within the academic realm. For now, it remains difficult for news consumers to simplify and aggregate their news diet in a highly automated fashion, without using the current ad-based services.

References

Breiman, L. (2001). Random forests. *Machine learning*, *45*(1), 5–32.

Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). *Classification and regression trees*. CRC press.

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/`

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms*. MIT press.

Denicola, D. (2020). *Jsdom homepage.* Retrieved from `https://github.com/jsdom/jsdom#readme`

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).

Goel, S., Watts, D. J., & Goldstein, D. G. (2012). The structure of online diffusion networks. In *Proceedings of the 13th acm conference on electronic commerce* (pp. 623–638).

Henze, N., & Nejdl, W. (2001). Adaptation in open corpus hypermedia. *International*

Journal of Artificial Intelligence in Education, *12*(4), 325–350.

Ho, T. K. (1995). Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition* (Vol. 1, pp. 278–282).

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.

Le, Q., & Mikolov, T. (2014). Distributed representations of sentences and documents. In *International conference on machine learning* (pp. 1188–1196).

Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., … others (2015). Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web*, *6*(2), 167–195.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady* (Vol. 10, pp. 707–710).

Ma, W. W., Hui, M.-L., Tong, Y.-Y., Tse, O.-K., & Wu, P.-Y. (2014). Exploring news reading behavior in hong kong: Identification of distinctive reader profiles. *Hong Kong Association of ducational Communications and Technology*.

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.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111–3119).

Nakayama, K., Hara, T., & Nishio, S. (2007). Wikipedia mining for an association web thesaurus construction. In *International conference on web information systems engineering* (pp. 322–334).

Nguyen, D. P., Matsuo, Y., & Ishizuka, M. (2007). Relation extraction from wikipedia using subtree mining. In *Proceedings of the national conference on artificial intelligence* (Vol. 22, p. 1414).

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

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (emnlp)* (pp. 1532–1543).

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

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, *1*(1), 81–106.

Rector, L. H. (2008). Comparison of wikipedia and other encyclopedias for accuracy, breadth, and depth in historical articles. *Reference services review*.

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

Schapire, R. E., & Singer, Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine learning*, *37*(3), 297–336.

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.

Thompson, M. (2019). *The new york times company annual report.*

Tiny tiny rss. (2020). Retrieved from `https://tt-rss.org/`

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

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).

Appendices

Appendix A

Code

The code for NewsReduce is available in the public git repository located at `https://github.com/sean-healy/newsreduce/`.

Appendix B

Random gaps

```
float pGapBasis(float probability, float sample, float population) {
    if (sample == population) return 0;
    float p = sample / population;
    float q = 1 - p;

    float base = q;
    return log(probability) / log(base);
}

unsigned int pGap(float probability, float sample, float population) {
    if (sample == population) return 0;
    float aboveMaxGap = population - sample + 1;
    float pAboveMaxGap = normalisedGapFrequency(aboveMaxGap, sample, population);
    float correction = -pGapBasis(1 + pAboveMaxGap, sample, population);
    float yScale = aboveMaxGap / (aboveMaxGap + correction);
    float xShiftedP = pGapBasis(probability + pAboveMaxGap, sample, population);
    float originIntercept = xShiftedP - aboveMaxGap;
    float yScaledP = originIntercept * yScale;
    float originalIntercept = yScaledP + aboveMaxGap;

    return (unsigned int) floor(originalIntercept);
}

const float RAND_MAX_FLOAT = (float) RAND_MAX;
unsigned int randomGap(float sample, float population) {
    float p = rand() / RAND_MAX_FLOAT;

    return pGap(p, sample, population);
}
```

Appendix C

Examples of high confidence results

Here are some of the results that were obtained when tackling the two tasks of finding news sources, and determining their official homepage, using data from Wikipedia. False positives and negatives are included.

```
https://en.wikipedia.org/wiki/National_Mortgage_News = https://www.nationalmortgagenews.com/
            https://en.wikipedia.org/wiki/LankaeNews =            http://www.lankaenews.com/
  https://en.wikipedia.org/wiki/Cashiers_du_Cinemart =    http://www.cashiersducinemart.com/
      https://en.wikipedia.org/wiki/Senses_of_Cinema =        http://www.sensesofcinema.com/
        https://en.wikipedia.org/wiki/Zimbabwe_Metro =         http://www.zimbabwemetro.com/
          https://en.wikipedia.org/wiki/Durma_Melhor =           http://www.durmamelhor.com/
            https://en.wikipedia.org/wiki/Myhomepage =            http://www.myhomepage.com/
       https://en.wikipedia.org/wiki/Breaking_Tweets =        http://www.breakingtweets.com/
        https://en.wikipedia.org/wiki/Portadown_News =         http://www.portadownnews.com/
https://en.wikipedia.org/wiki/Christian_Film_Database = http://www.christianfilmdatabase.com/
             https://en.wikipedia.org/wiki/GameFront =             http://www.gamefront.com/
      https://en.wikipedia.org/wiki/The_Fiscal_Times =        http://www.thefiscaltimes.com/
     https://en.wikipedia.org/wiki/Smashing_Magazine =          http://www.smashingconf.com/
          https://en.wikipedia.org/wiki/SemiAccurate =          http://www.semiaccurate.com/
   https://en.wikipedia.org/wiki/World_Chess_Network =     http://www.worldchessnetwork.com/
             https://en.wikipedia.org/wiki/Dont_Party =             http://www.dontparty.com/
               https://en.wikipedia.org/wiki/OhmyNews =    http://international.ohmynews.com/
            https://en.wikipedia.org/wiki/PistonHeads =           http://www.pistonheads.com/
              https://en.wikipedia.org/wiki/LifeZette =             http://www.lifezette.com/
       https://en.wikipedia.org/wiki/Ovi_(magazine) =               http://ovimagazine.com/
https://en.wikipedia.org/wiki/The_Pan-Arabia_Enquirer =     http://www.panarabiaenquirer.com/
               https://en.wikipedia.org/wiki/E-novine =              http://www.e-novine.com/
               https://en.wikipedia.org/wiki/Netvibes =              http://www.netvibes.com/
https://en.wikipedia.org/wiki/Winding_Road_(magazine) =           http://www.windingroad.com/
               https://en.wikipedia.org/wiki/TamilNet =              http://www.tamilnet.com/
```

https://en.wikipedia.org/wiki/The_Mud_Connector	http://www.mudconnect.com/
https://en.wikipedia.org/wiki/IT_News_Africa	http://www.itnewsafrica.com/
https://en.wikipedia.org/wiki/The_Week_in_Chess	http://www.theweekinchess.com/
https://en.wikipedia.org/wiki/Blistering	http://www.blistering.com/
https://en.wikipedia.org/wiki/BellaNaija	http://www.bellanaija.com/
https://en.wikipedia.org/wiki/IvyGate	http://www.ivygateblog.com/
https://en.wikipedia.org/wiki/Weardrobe	http://www.weardrobe.com/
https://en.wikipedia.org/wiki/NASASpaceFlight.com	http://www.nasaspaceflight.com/
https://en.wikipedia.org/wiki/ScienceBlogs	http://scienceblogs.com/
https://en.wikipedia.org/wiki/ReadWrite	http://readwrite.com/
https://en.wikipedia.org/wiki/The_Groton_Line	http://thegrotonline.com/
https://en.wikipedia.org/wiki/Junkee	http://junkeemedia.com/
https://en.wikipedia.org/wiki/Examine.com	http://examine.com/
https://en.wikipedia.org/wiki/Flavorwire	http://www.flavorpill.com/
https://en.wikipedia.org/wiki/Newsvine	http://newsvine.com/
https://en.wikipedia.org/wiki/Consumerist	http://consumerist.com/
https://en.wikipedia.org/wiki/Cromos	http://www.cromos.com.co/
https://en.wikipedia.org/wiki/Publishers_Weekly	http://www.booklife.com/
https://en.wikipedia.org/wiki/Easyriders	http://www.easyriders.com/
https://en.wikipedia.org/wiki/Antique_Trader	http://www.antiquetrader.com/
https://en.wikipedia.org/wiki/Campaigns_and_Elections	http://www.campaignsandelections.com/
https://en.wikipedia.org/wiki/Guns_(magazine)	http://www.gunsmagazine.com/
https://en.wikipedia.org/wiki/Country_Living	http://www.countryliving.com/
https://en.wikipedia.org/wiki/Natural_History_(magazine)	http://www.naturalhistorymag.com/
https://en.wikipedia.org/wiki/Pro_Football_Weekly	http://www.profootballweekly.com/