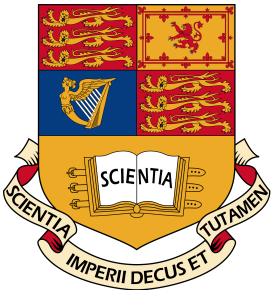


Imperial College London

MEng Computing Final Report

NewSumm

News Aggregator and Summariser

Author:
Kunal M L Wagle

Supervisor:
Anandha Gopalan

Abstract

Abstract to be written

Acknowledgements

I would like to offer my sincere gratitude to my supervisor, Anandha Gopalan, for his total support and advice during the course of this project, and for ensuring I never strayed too far from the right track.

I would also like to thank my family for their unbounded love and support, and tolerance as this project went through its most trying moments.

Finally, I'd like to thank Joe Letts, a friend who would always be willing to take time out of his schedule to help test the system, and who would prove an invaluable sounding board in times of crisis.

Contents

List of Figures

List of Tables

Listings

1 Introduction

The onset of the digital age has resulted in an insurmountable volume of news for a consumer to read. Not only are all the traditional news outlets publishing stories (and not just about local news, but news from all over the globe), but blogs are getting in on the act as well.

In May 2017 President Donald Trump visited the Vatican. If I were to search Google News on the 24th May 2017, I'd find a story about the President's meeting with the Pope (see Figure 1.1).

On further inspection, I can see that there are several pages of results about the meeting itself, from dozens of different news sources. They've all covered the story from different perspectives as well. Some have covered the contents of the meeting itself, some covered the general concept of the meeting in the wider context of Trump's trip to Saudi Arabia earlier that week, and some even chose to focus on everyone's attire at the meeting.

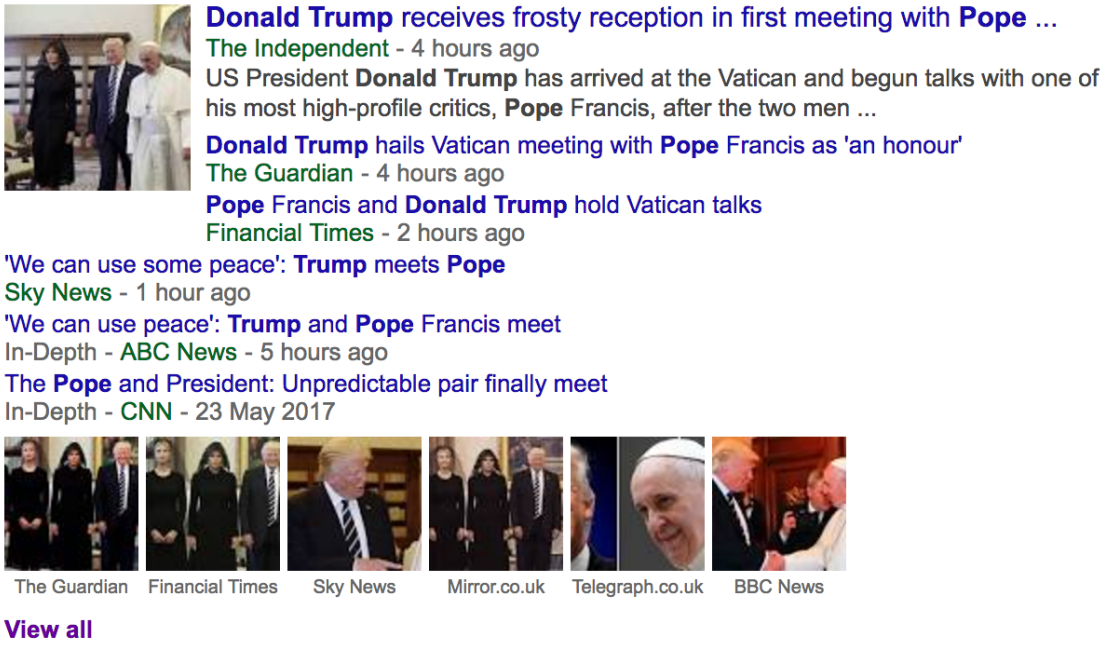


Figure 1.1: *A Google News search for stories about President Trump's visit to the Vatican on 24th May 2017*

When a consumer has to search for the news and find these sources themselves, it's more than probable that they will end up missing something. Also, which sources

should they go to? And how many sources are enough to get a full and accurate picture?

News Aggregators, such as Flipboard and Apple News, remove the requirement for a user to source articles themselves. The aggregators find the news and present it to the user in a way that is nicely grouped by topic. Other aggregators, such as Google News, allow consumers to search for news topics of their choosing as well, and present all the articles it finds on the same topic.

But these solutions don't go far enough, as they don't address a major issue - there is still a very high volume of news.

In July 2016, a study was conducted by the Statistic Brain Research Institute that said that the average attention span of a person is now as low as 8.25 seconds. In addition to this, the Institute's findings also said that the average person only reads 28% of a webpage of average length (593 words). Another key finding was that users spend only 4.4 seconds to cover each additional 100 words on a webpage. It's simply not feasible for a consumer to go through several different sources' take on a piece of news, which is what many aggregators still expect them to do.

The primary objectives of my project are therefore two-fold. Firstly, the solution to the project needs to tackle the issue of users missing information because they need to find multiple sources for a topic. Secondly, the project will have to present a solution to the very high volume of news that a consumer needs to read.

My solution is for a news aggregator that goes further than current products. NewSumm would have a primary aim of not only aggregating news, but of combining multiple sources' articles on a piece of news, and summarising them into one short piece. Using the example from before, the intention would be that the consumer only has to read one summary that incorporates the facts from several different sources on Trump's visit to the Vatican.

There is also the issue of reliability and trust to consider. The Digital News Report 2016 found that in the United Kingdom only 50% of readers trust what they see in the news. Could this be a case that there is simply so much news that it is hard to tell which is real? This could be the case because some readers will also be aware that a consequence of the new digital age is that anyone with an internet connection can create a blog and start spreading 'fake news'.

Leading from this issue of trust, a secondary objective of the news aggregator would be that it produces content that users can be near certain is trustworthy and reliable - effectively a news aggregator that is free from 'fake news'.

Whilst it should be emphasised that the primary aim of NewSumm would be to

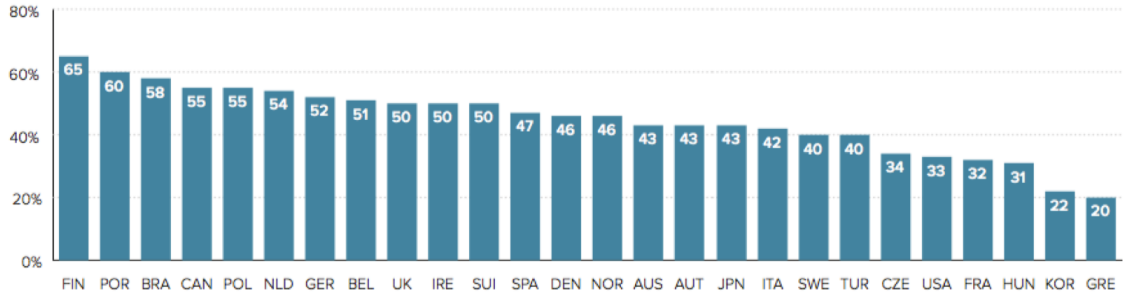


Figure 1.2: *Digital News Report findings showed that when it came to trust in the news, in only a few countries was trust as high as 50% in the accuracy of a piece of news. In countries such as the United States it is far lower.*

reduce the sheer volume of news that a consumer has to read, it could be argued that the problem of 'fake news' could be indirectly minimised in the same way that other news aggregators attempt to tackle the issue. This would be by limiting articles to trusted sources from across the political spectrum. This would be reputable news organisations, and articles from random blogs on the internet would not be considered. In addition to this, a clear indication when reading the summary of what facts different outlets all agree on, and what facts they either differ on or omit, will allow the user to easily corroborate parts of the story.

Beyond this, another objective of the project should be for the News Aggregator to allow users to eliminate elements of news, or sources themselves that they deem to be untrustworthy, thus alleviating consumer concerns about fake news.

A further secondary aim to this could be to allow a user to customise their summary, taking in information from the outlets of their choice (from the original 'trusted' list). The advantage to this would potentially be that if a user decides that too many issues from one source are uncorroborated, they can remove them, and see only information from the other sources that have written about a topic. The disadvantage to this could be that if a user dismisses an entire Section of the political spectrum as uncorroborated continuously, they'll pick up a political bias on the facts of the topic. Although, it could be said that the Digital News Report 2016 findings suggest that this is a feature that won't be used much. One of the more interesting findings in the report was that more than 60% (in the United Kingdom) were concerned about the potential impact of personalising their news feeds. They felt that it could lead to both 'missing information' and 'missing challenging viewpoints'.

The remainder of this report will follow the development of this concept, as both a website and as an iOS application. The following sections will set out New-

Summ to be an ambitious project that aims to automatically fetch, label, cluster and summarise articles without supervision. NewSumm will be outlined as a user-friendly application that allows consumers a large level of control over what they read, through topic subscriptions and the ability to alter sources for summaries, as well as providing a daily digest for users who want them.

Whilst the ultimate goal of an abstractive summarisation tool in the back end will prove to be infeasible, NewSumm, after thorough evaluation, will be found to be an application that provides more than competent summarisation of articles and effective clustering that goes a large way to reducing the large volume of news that a consumer has to get through.

Going side by side with this evaluation of the fundamentals of the system itself is a rigorous analysis of the summaries produced to establish whether the system can be affected by the various political biases of its sources. The root causes of any political bias will also be fully investigated, and the role of the neutral source in a summary will also be discussed.

2 Market Research

2.1 News Reading Habits

2.1.1 Digital News Report 2016

Conducted by the Reuters Institute for the Study of Journalism [24] in Oxford University [31], the Digital News Report [23] is an annual study that serves to investigate how people access and find out about news in various countries across the globe, including the United Kingdom.

The study attempts to cover, amongst other things how people get the news, how they use social media, what types of news they trust, and both the reasons to use and concerns about news aggregators. Findings of the report relevant to my concept are listed below:

Reasons to use News Aggregators

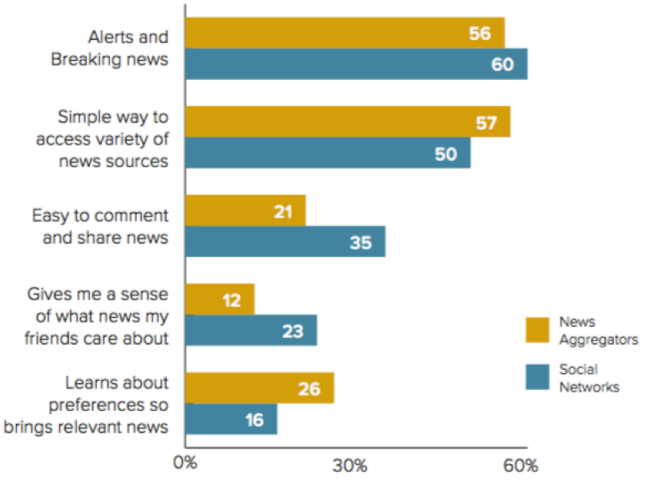


Figure 2.1: *A graph showing why people use News Aggregators and Social Networks as news sources*

As shown in the graph in Figure 2.1 the key to the popularity of a News Aggregator is that it's simplicity and speed. It's important to bring news as soon as it happens, and it needs to provide access to a variety of different news sources. Of lesser importance is the social aspect - the need to comment and share the news, although the fact that 26% of people want their preferences to influence the news that they're interested in should not be discounted.

Concerns about Personalised News

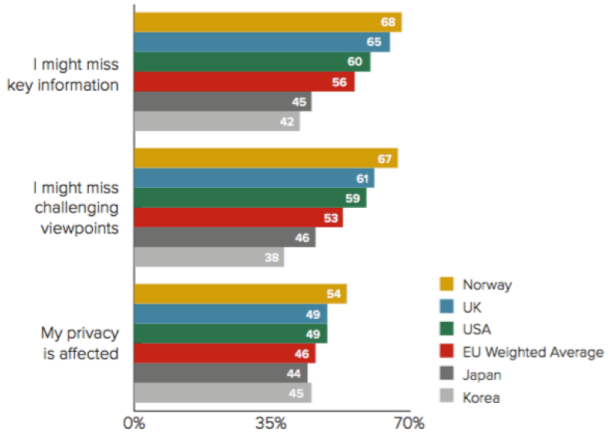


Figure 2.2: *A graph showing the main three concerns that people have with Personalised News Feeds determining what they read.*

Key concerns, especially in the United Kingdom, with Personalised News is that information might be incomplete. It is not difficult to see why this may be a concern - if you limit your news to a subset of the sources that might be available then it's possible to miss parts of the story - the parts that challenge aspects of the articles you're reading. It's therefore important that the project reflects this concern, which is why it is considered a secondary aim (see Section 1).

What should determine what someone reads next

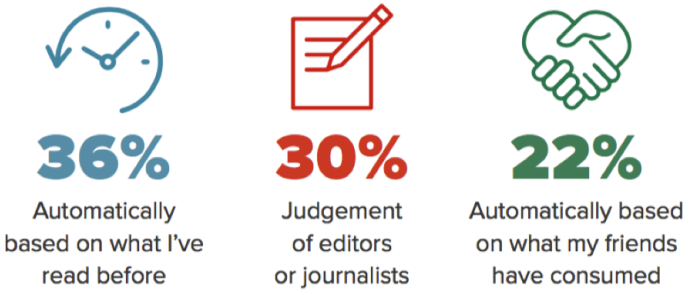


Figure 2.3: *Key statistics in response to the question: Is this a good way to get the news?*

Least popular are algorithms that present news based on what people's friends have been reading. In general, according to the Reuters Institute [24], 'People think

they are the best judge of what's important to them.' More interesting, is that people are now less inclined to allow journalists or editors to push articles to the top of reading lists than having an algorithm suggest similar stories to what they've just consumed. This could perhaps indicate that they aren't as trusting in journalists and editors as before - something that the Reuters Institute touches upon in an essay near the end of the report.

Conclusions

The Reuters Institute for the Study of Journalism is a trusted and respected organisation, and this study has been supported by institutions varying from media outlets such as the BBC [3], news search engines such as Google, and other Universities, such as University of Canberra [30]. Importantly, it's also supported by regulators, including Ofcom [22], which is the regulator in the United Kingdom.

The survey was conducted on their behalf by YouGov [39], which is a highly respected polling company in the United Kingdom, and it surveyed more than 50,000 people, including over 2000 in the United Kingdom. In addition, the report is coupled with numerous essays on various key points that arose from the study. The writers of these essays varied from the Director of Research at the Reuters Institute to the Chief Executive Officer of The New York Times [29].

As a result of this, I deemed the Digital News Report 2016 [23] to be a very reliable source of information with regards to how people read the news, and their preferences in the field.

2.1.2 Potential User Survey

Following this initial research I decided to conduct a survey of potential users to delve into more specific aspects of News Reading Habits. I was more focused on the number of sources people use to ascertain the facts about a piece of news. I also asked questions regarding the summarisation of news and news digests.

I was interested in the questions about summarisation and digests after reading some data from a study conducted by the Statistic Brain Research Institute [25]. Statistic Brain conducted a study in July 2016 that suggested that the average attention span of a human is now only 8.25 seconds [16]. They coupled the presentation of this data with statistics about people browsing the internet taken from the paper *Not Quite the Average: An Empirical Study of Web Use* [33]. The statistics suggest that the average user only reads 28% of the words on the average webpage, and that for each additional 100 words beyond that, the user would only spend an additional 4.4

seconds on the page.

Admittedly, the paper was written in 2008, and so might not reflect average web browsing activity, but it raised a question worth answering in my opinion. Do users actually read full articles on the news, and if not, would summarising articles help make sure they didn't miss out on key points, thus counteracting an issue discovered in the Digital News Report 2016 [23] (Section 2.1.1)?

The survey was presented on Google Forms [10] and was answered by 96 respondents. Further findings are shown below:

1. **How often do you read the news?**

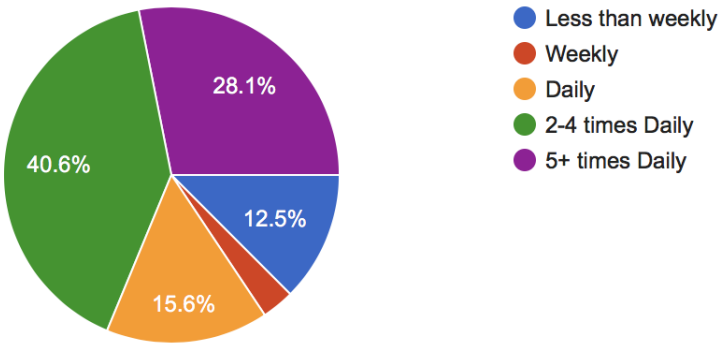


Figure 2.4: *A pie chart showing how often people read the news.*

The first question (Figure 2.4) on the survey was designed to ascertain how often people read the news. Overall, well over 80% of respondents said that they read the news at least daily, and nearly 60% more than once per day.

2. **How many sources do you read?**

Next I asked about how many sources a user read. It was quite rare for someone to use only one source for a piece of news. In fact, over 90% said that they used two or more sources, with three and five sources being the most prominent selections.

3. **Are these the same sources every time?**

The respondents were a lot more split on the question of consistency of sources, although more than half said that they used the same sources every time, with

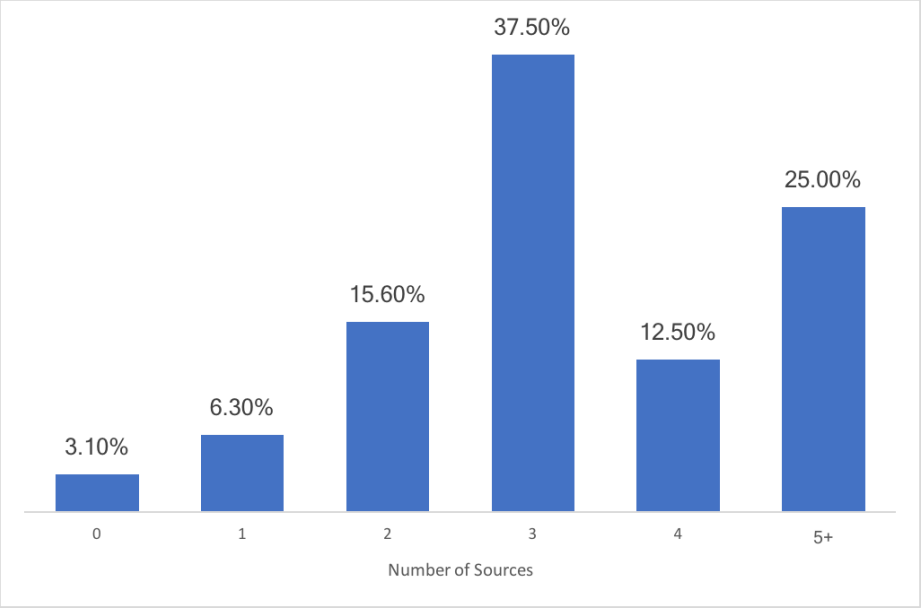


Figure 2.5: *A bar graph showing how many sources people normally use for a source of news.*

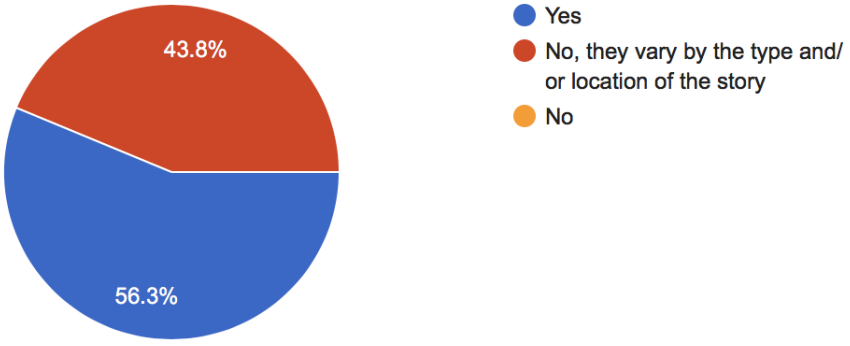


Figure 2.6: *A pie chart showing whether people use the same sources every time.*

43.8% saying that they vary their sources based on the type and/or location of the news story.

4. **How much of an article do you normally read?**

   I further expanded upon the initial reading I had done about attention spans by asking potential users how much of articles they normally read. Only 18.8%

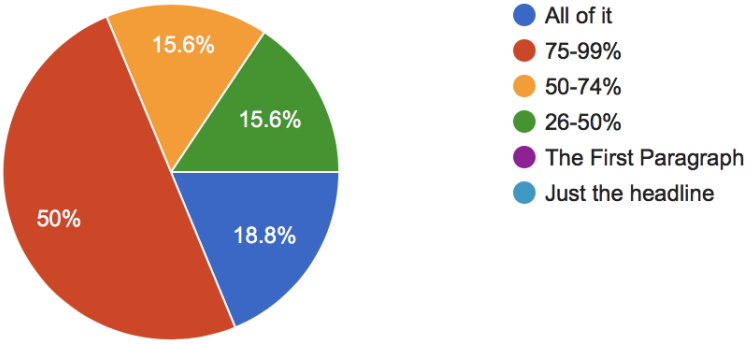


Figure 2.7: *A pie chart showing how much of articles people normally read.*

of respondents said that they read the entirety of articles (Figure 2.8), and as much as 15.6% confess that they read less than 50% of an article on average. It's plausible that the users could be missing key facts through not reading the entire article. On reflection, I could have also asked if this was a concern to those users.

5. **If an article was presented as a shorter summary, would you read more/all of it?**

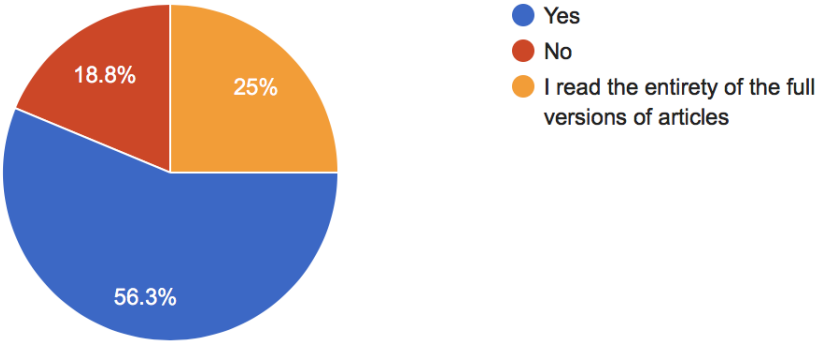


Figure 2.8: *A pie chart showing what percentage of people would read a shorter summary of an article.*

Another question asked was whether reading a shorter summary would result in users reading more or all of the article. Encouragingly, more than half

(56.3%) of respondents said that they would read a shorter summary.

6. **Would you read a summary of different articles on the same topic combined?**

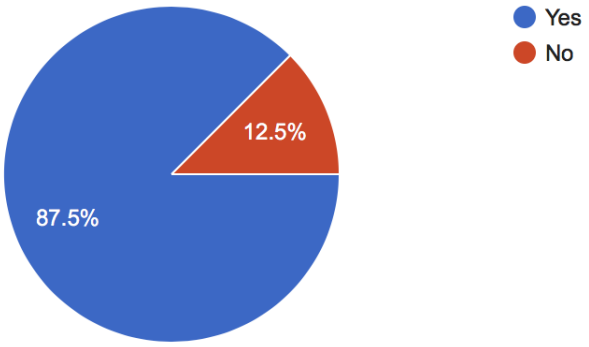


Figure 2.9: *A pie chart showing whether people would read a summary of different articles on the same topic combined.*

With this question (Figure 2.9) I aimed to obtain an idea from potential users as to whether combining articles about the same topic from different sources and then summarising it would be considered useful. The question got a resounding response, with as much as 87.5% saying that they would read a summary like this.

7. **Would you use a News Aggregator that summarised articles?**

Consolidating the findings of the previous question was the fact that 81.3% said they would use a News Aggregator that summarised articles (Figure 2.10). Amongst those who said they wouldn't, there were comments along the lines of 'I'm not great with technology' given as explanation. An interesting comment however, said that 'nuance could be lost in the summarisation'. This is a valid point, and so a key point of the evaluation process will have to be focused on the summarisation of articles itself, to ensure it's not losing important information at any stage.

8. **How often do you search for news on a specific topic?**

For the questions in Figure 2.11 I tried to get an idea of how much people search for news on a specific subject of interest to them. In general, this came

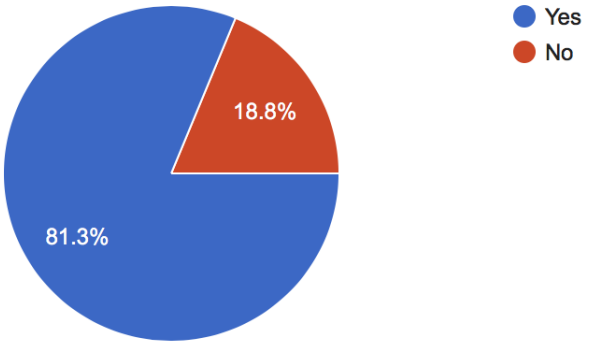


Figure 2.10: *A pie chart showing answers to the question: 'Would you use a News Aggregator that summarised articles'*

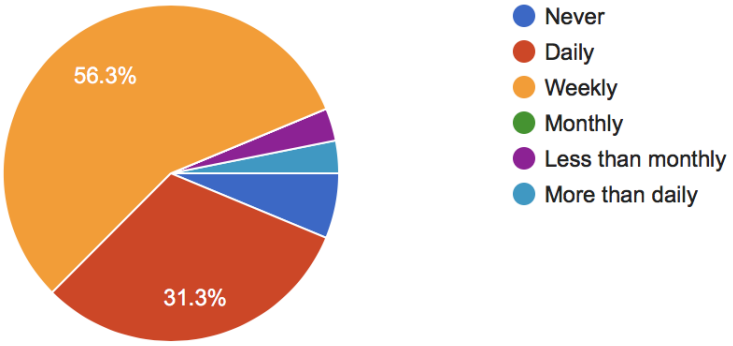


Figure 2.11: *A pie chart showing how frequently people search for news on a specific topic.*

out to be less frequent then reading the news itself, with just over half the respondents saying that they search for a specific topic weekly. However, a healthy proportion (31.3%) search daily for specific topics, and some search even more often than this.

9. **Should the news aggregator have the ability to search for a specific topic?**

   As much as 81.3% of respondents agreed that the News Aggregator should have a function to search for specific topics.

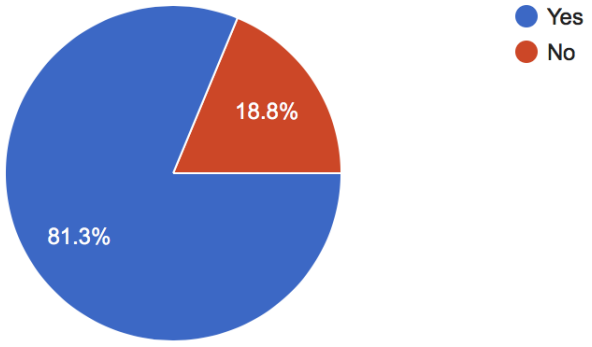


Figure 2.12: *The answers to a question asking if the News Aggregator should have the ability to search for a specific topic.*

10. **Do you like services such as news digests that prepare lists of articles each day that apply to a topic you may be interested in?**

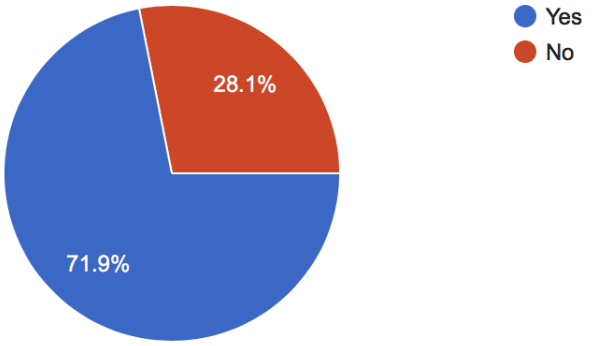


Figure 2.13: *A pie chart showing people's attitudes towards News Digests.*

The next question, shown in Figure 2.13 centred around News Digests. Nearly 72% of respondents said that they liked services that provide news digests.

11. **How often should these news digests be updated?**

People who were in favour of news digests in general felt that these should be updated at least daily (47.8%), with a further 34.8% on top of that saying it

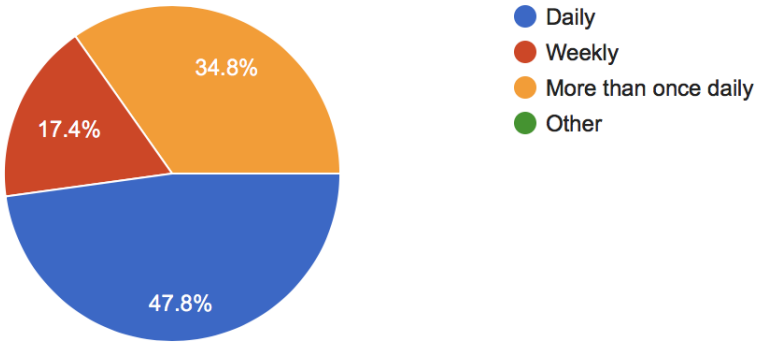


Figure 2.14: *Responses regarding how frequently news digests should be updated.*

should be updated more than once.

12. **What platform should the aggregator be developed for?**

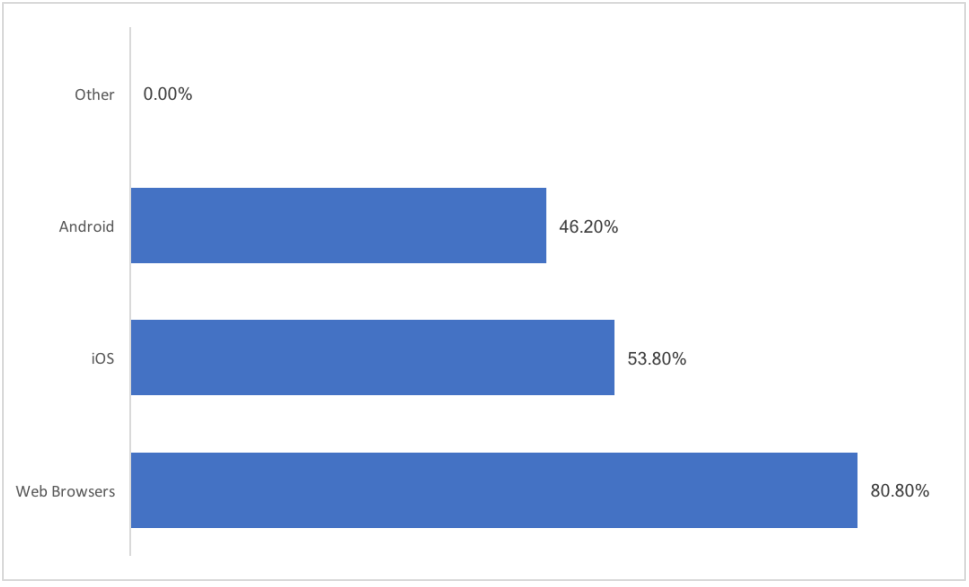


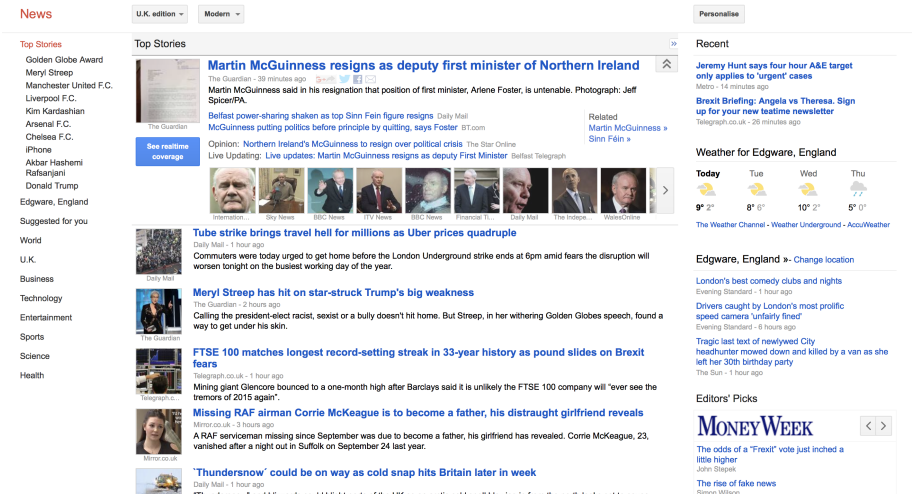
Figure 2.15: *A bar chart showing responses when asked which platform they'd rather see the news aggregator developed for.*

Finally, respondents were asked which platform they would prefer to see the news aggregator developed for. 80.8% said they'd want a version for web browsers, and 53.8% for iOS and 46/2% for Android. This would suggest that

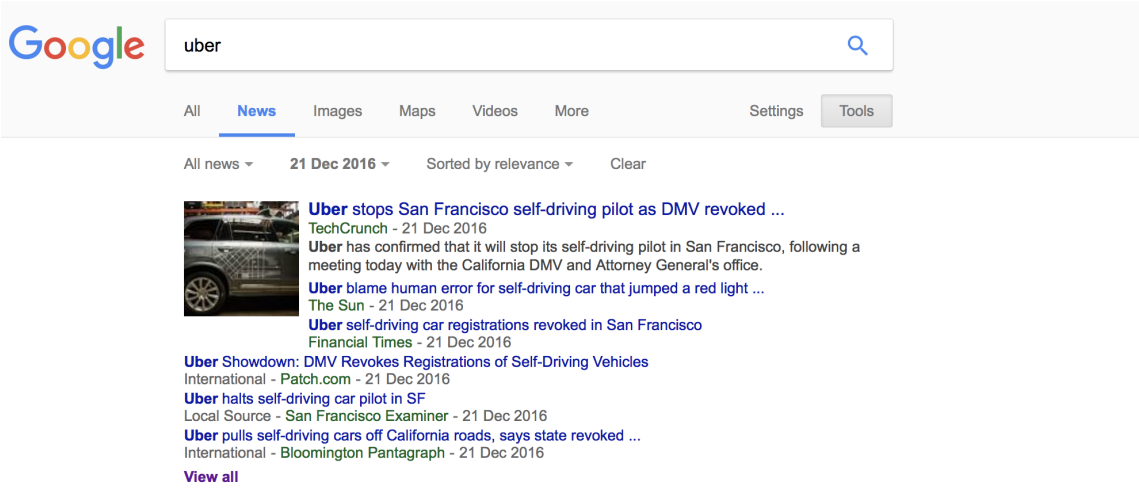
I should develop a web solution and then move to a mobile platform afterwards if there is time.

2.2 Related Products

2.2.1 Google News



(a) A Section of the homepage for Google News. [11]

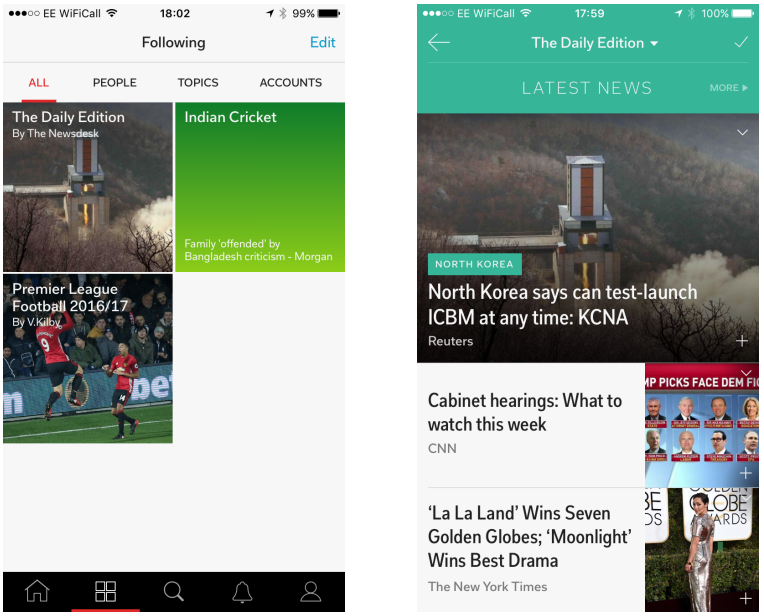


(b) Google News uses clustering techniques to group articles about the same topic together.

Figure 2.16: *Screenshots from Google News*

Initially developed early in the century and released in 2006, Google News [11] is a free-to-use news aggregator. Google News operates in a similar manner to the traditional Google [9] search engine, thus making it a go to aggregator when searching for a specific topic. Google News also groups 'similar' articles together using Clustering techniques [15]. Google News operates purely as a go-between - when clicking on an article the user is taken straight to the media source itself, rather than being able to read the article on Google News itself.

2.2.2 Flipboard



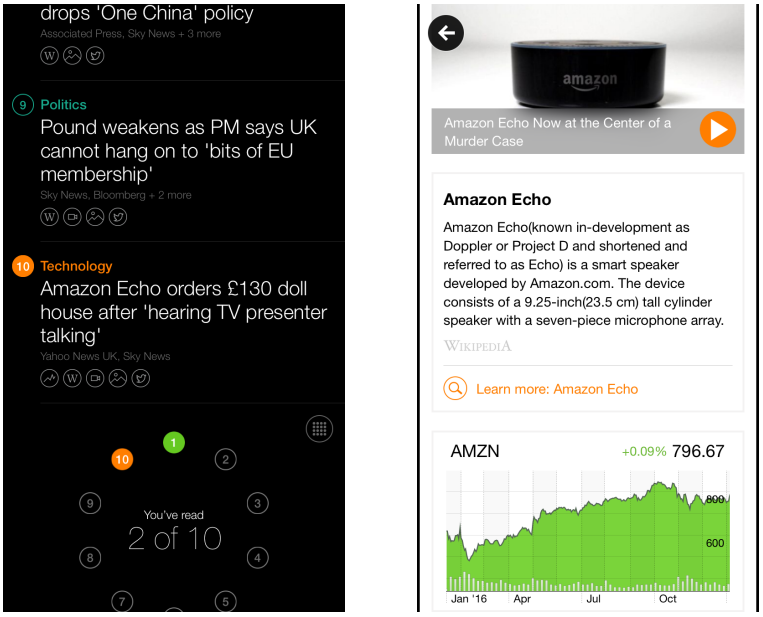
(a) *Users can subscribe to 'magazines' that they may be interested in.*

(b) *All users are subscribed to* The Daily Edition, *which puts together the latest most popular news.*

Figure 2.17: *Screenshots from the Flipboard app*

Flipboard [6] is a much more recent attempt at a news aggregator (developed in 2010), and relies on the concept of users subscribing to 'magazines' on different topics. There's a central 'cover page' on the home page that shows the most recent stories from across all a user's subscriptions. Like Google News, links from the desktop website send the user to the original media source, while links from within the mobile applications open a browser within the app itself.

2.2.3 Yahoo News Digest



(a) *A Section of the home screen, which displays the top ten stories for the day.*

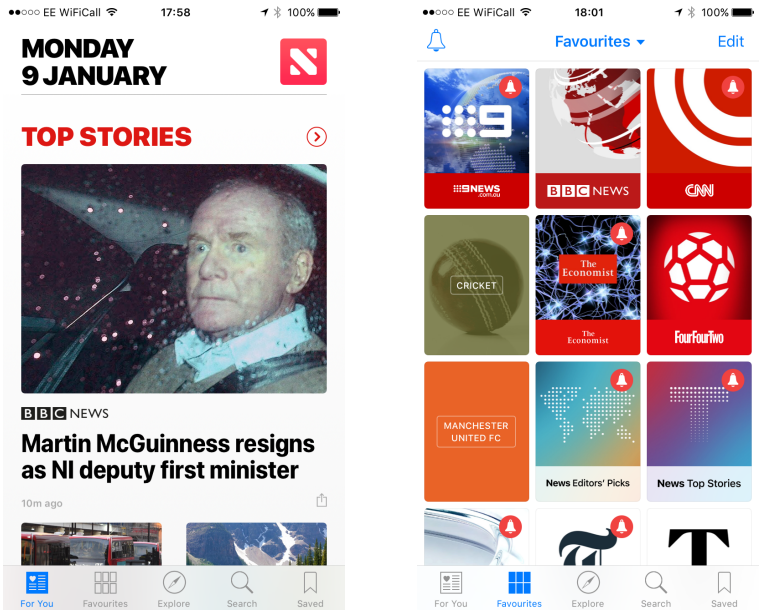
(b) *The Yahoo News Digest app provides links and infographics that are relevant to the article.*

Figure 2.18: *Screenshots from the Yahoo News Digest app [38]*

Yahoo News Digest [38] is a direct evolution from Summly [26], which was an app that summarised news. Yahoo News Digest is a phone application that creates two digests a day: one in the morning and one in the evening. Each digest contains the ten leading articles from the previous 12 hours. Each article provides a summary of the story and links to relevant other pages - such as articles from Wikipedia [36].

2.2.4 Apple News

Apple News [2] is an application that is installed by default on all recent iOS devices. Apple's default attempt at a news aggregator allows users to select their preferred news sources, and from a selection of topics. Apple [1] then presents on a home screen news from those sources and topics. Clicking on each article keeps it in the native application, rather than sending the user to the media source itself.



(a) The home screen for the Apple News app

(b) Apple News allows users to select sources and topics to be their favourites.

Figure 2.19: *Screenshots from the Apple News app [2].*

2.2.5 Comparing the existing products

The existing products are compared in Table 2.1 on page 30.

Product	What it does well	What it doesn't do well
Google News	• Groups similar articles together • Similar to a traditional search engine and so is easy to use • Obtains articles instantly, thus providing most up-to-date information when searching	• Doesnt́ host articles itself - therefore making it harder to navigate than perhaps could be possible. Although, this could be to avoid any copyright issues.
Flipboard	• Provides a home page that allows a user to see the most popular stories at the time related to the user's subscriptions. • Available as a mobile application	• Topics are much broader than Google News, thus meaning that users can't necessarily search for topics specific-enough for them. • Flipboard requires registration before being able to read articles
Yahoo News Digest	• Yahoo News Digest won the 2014 Apple Design award [37]. • The articles also provide key infographics, quotes, and other information potentially relevant to the article.	• There are only ten articles available per digest, and no capability for searching by topic. • Digests are only produced in the morning and the evening, so the news articles presented could be out of date.
Apple News	• Allows selection based off both topics and the news sources themselves. • The home screen for the app allows users to see a lot of headlines at a glance.	• Navigation on the application is not simple. If a user has accessed many articles from push notifications, then the user could have to press the back button several times to get back to the home screen. • The topics that a user can subscribe to can be quite limited, and aren't reactive to current affairs - for example, if a natural disaster occurs, you couldn't then subscribe to that natural disaster as a topic.

Table 2.1: *Comparing the relative benefits and drawbacks of Google News, Flipboard, Yahoo News Digest and Apple News*

3 Background Research

3.1 Machine Learning Techniques

3.1.1 Topic Modelling Techniques

Topic modelling is a subsection of Machine Learning that aims to determine what topic a given document is about. The topics wouldn't be named at this stage, they would simply be given generic names such as Topic A and Topic B. Assigning names to topics will be done at a later stage (see Section 3.1.2).

Latent Semantic Indexing

Also known as Latent Semantic Analysis [19], Latent Semantic Indexing (LSI) was one of the initial forerunners in the field of topic modelling. It uses singular value decomposition to locate patterns in the text of a document and thus form a basis on which to categorise the document. A major benefit of LSI is that it is fast to train, but in general it has lower accuracy when compared to models that are probabilistic, such as Latent Dirichlet Allocation [35].

Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) is a generative probablistic model that assumes that the topic distribution has a Dirichlet prior. The theory behind LDA is that a document of words contains a mixture of different topics, and this is reflected in the final answers given for the algorithm.

A rough algorithm for performing LDA [34]:

1. **Set n to be the number of topics there are in the document.** We can do this by trial and error.
2. **Assign every word w in the document d to a random topic.** These topics are temporary. At this stage we can remove function words, such as 'the'. However, we keep duplicates - in fact, at this stage they could be in different topics.
3. **Check and update topic assignments.** To do this, we loop through each word in the document, taking note of how prevalent the word is across documents, and prevalent those topics are in the document. These two probabilities are then passed to a sampling algorithm to generate a new topic for the word. This step is usually completed using the statistical model *Gibbs Sampling*.
4. **Repeat step 3 until there are no more topic-reassignments**.

3.1.2 Topic Labelling Techniques

Topic Labelling techniques in the project for labelling the topics that are generated from the LDA analysis of the document in Section 3.1.1. When conducting research into this topic specifically, I found a paper (*Automatic Labelling of Topic Models* by Lau, Grieser, Newman and Baldwin in 2011 [20]) that documents the creation of an algorithm that labels topics using Wikipedia [36] titles. This could work very well in my project, as Wikipedia titles as title headings would allow users to search for topics that are both broad and specific.

A rough version of Lau, Grieser, Newman and Baldwin's algorithm:

1. **Calculate the top 10 topic terms.** This is done by finding the marginal probabilities of each word from the original topic models, and taking the top ten. The marginal probability of a term is the probability of that term being randomly selected given a topic t.
2. **Search Wikipedia using these terms.** We also search Google [9] using a site restricted search (to Wikipedia) and take the top eight results from each. These are called the *primary labels*.
3. **Isolate all 'noun chunks' from the terms.** In this case noun chunks are combinations of words from the terms that appear next to each other. For example, with the term 'Summer Olympic Games' the noun chunks would be 'Summer', 'Olympic', 'Games', 'Summer Olympic', 'Olympic Games' and 'Summer Olympic Games'. Note that 'Summer Games' is not a noun chunk as the words don't appear juxtaposed. These noun chunks are added to the primary labels from step 2 and are deemed *secondary labels*.
4. **For each noun chunk:**
   - Check to see if the noun chunk is the title for a Wikipedia article
   - Remove the noun chunk if it doesn't correspond to a Wikipedia article
5. **Calculate the *Related Article Conceptual Overlap* scores.** Related Article Conceptual Overlap (RACO), developed by Grieser et al in 2011 [12], is a calculation designed to identify the strength of relationship between terms by inspecting the category overlap between the terms' corresponding articles. We do this for each secondary label still remaining - details on how to calculate the RACO scores are explained in further detail below.
6. **Discard all secondary labels with RACO score of less than 0.1.**
7. **Add five highest topic terms to the list.** Now we return to the original list

of topic terms from step 1 and add the five highest to the remaining candidates.

8. **Perform candidate ranking.** There are multiple ways to do this, but the original paper recommends using a variety of statistical methods, based around a T-test, the Chi-squared test and a log-likelihood test. The aim is to estimate how closely related the candidate is to all the terms in the topic. We then take the top candidate as our final answer.

Calculating the *Related Article Conceptual Overlap*

Related Article Conceptual Overlap (RACO) was first introduced as a concept in the paper *Using Ontological and Document Similarity to Estimate Museum Exhibit Relatedness* [12] by Grieser, Baldwin, Bohnert and Sonenberg in 2011. It compares two terms and estimates their similarity by comparing the two terms' similarity on Wikipedia. The core calculation for RACO goes as follows:

$$Category - Overlap(a, b) = \left| \left(\bigcup_{p \in O(a)} C(p) \right) \bigcap \left(\bigcup_{p \in O(b)} C(p) \right) \right|$$

In this equation, *O(a)* represents the outlinks (the links to other articles from the Wikipedia article) of an article a and *C(p)* represents the set of categories that article p is a part of.

An issue with the Category-Overlap calculation as it is, is that there's a bias for articles that are larger than others as they will have more outlinks but won't necessarily be in more categories. As a result it's normalised using Dice's coefficient to produce the final RACO equation:

$$sim_{RACO}(a, b) = \frac{2 \times \left| \left(\bigcup_{p \in O(a)} C(p) \right) \bigcap \left(\bigcup_{p \in O(b)} C(p) \right) \right|}{\left(\bigcup_{p \in O(a)} C(p) \right) + \left(\bigcup_{p \in O(b)} C(p) \right)}$$

3.1.3 Clustering Techniques

Cluster analysis is a machine learning technique that is used to put items that are similar to each other into groups. In practice it's used by Google News [11] to put articles about the same topic together for a user [15]. There are multiple commonly used types of cluster analysis:

Centroid Clustering

In Centroid Clustering [4], also known as k-means clustering, there are k clusters. A vector is calculated for each article in the list. The article is then assigned to the cluster that is closest to it's vector score. A major downside to this method however is that it requires k to be defined in advance. In the context of this project, that's not applicable as we don't know how many different articles we are clustering based on.

Density Clustering

Density Clustering [4] also involves calculating a vector score for each article. Once these have been calculated, they can be graphed, and the areas of the graph that have highest density are chosen as the clusters. An advantage of this over the centroid clustering methods is that we don't need to know the number of clusters beforehand. However, density clustering can become less accurate as it requires areas of sparse density on the graph to precisely separate the different groups, which isn't always possible.

Hierarchical Clustering

Hierarchical Clustering [14], which is also known as Connectivity Clustering, is based on the idea of using distance measures between articles to identify which ones are most similar. There are two approaches to Hierarchical Clustering:

- *Agglomerative*, which is a bottom-up approach, assigns each item to its own cluster and then merges pairs that are closer together.
- *Divisive*, a top-down approach, that begins with all items in a single cluster, and then proceeds to split it into multiple clusters.

Creating a vector score for each item

The first step in each of the three clustering techniques is to create a vector score for each item. As this will play a big part in the final results, it's important to get this stage right. This can be split into two steps:

1. **Find definitive terms within the article.** This can be done using techniques such as the popular *Term Frequency-Inverse Document Frequency* (TF-IDF) , which is explained in further detail below. Proper nouns would be useful

in this step, as they are more likely to be relevant to what the article is specifically about.

2. **Create a vector.** This vector, based from the definitive terms from the first step, would be a set of keywords and corresponding weights.

Term Frequency-Inverse Document Frequency

TF-IDF [27] is designed to identify terms that appear frequently in one article that don't occur a lot over the entire set of articles (also known as the *corpus*). Variations of it are commonly used by search engines to identify search results that are most relevant to a query. The calculation for TF-IDF is as follows:

Term Frequency

Term Frequency can be most simply calculated as the frequency of a term in a document. However, this could result in a bias towards terms that appear in longer articles. A more accepted way to calculate the Term Frequency therefore is to use a normalising function, called augmented frequency:

$$tf(t,d) = 0.5 + 0.5 \times \frac{f_{t,d}}{max\{f_{t',d} : t' \in d\}}$$

Inverse Document Frequency

Inverse Document Frequency is used to check in how many documents of a corpus D a given term t occurs. It is an inverse function, so as to minimise the value for the term when it is common amongst various different documents. It is also logarithmically scaled. It is calculated using the following formula:

$$idf(t,D) = \log \frac{|D|}{|d \in D : t \in d|}$$

Term Frequency-Inverse Document Frequency

$$tfidf(t,d,D) = tf(t,d) \times idf(t,D)$$

3.2 Summarisation Techniques

Summarisation techniques will be (predictably) used for summarising the merged articles that were identified by the processes of Topic Modelling , Topic Labelling and Clustering. There are two types of summarisation:

- **Extractive Summarisation**, which consists of taking sentences that are important from the original text, and discarding the rest [13].
- **Abstractive Summarisation**, which aims to generate a piece of text using natural language techniques. A key to this is that some words in the final summary may not have been the original piece of text.

3.2.1 Extractive Summarisation Techniques

LexRank and TextRank

There are two well known examples of extractive summarisation: LexRank and TextRank.

LexRank and TextRank have very similar methods for extracting a summary [5]. Initially a graph is constructed, that consists of one node for each sentence in the corpus. Then a clustering algorithm is applied, using a TF-IDF calculation to determine similarities.

There are a couple of key differences between LexRank and TextRank. The first arises in the calculation. Both use a TF-IDF calculation, but they are varied slightly. LexRank uses a cosine similarity function in order to weight the final calculation, whereas TextRank uses a more simple logarithmic weighting to perform the calculation.

Both use Google's PageRank algorithm to then rank the importance of each sentence based on the calculations in the previous step. With d being a damping factor, the PageRank of a node u is given as:

$$p(u) = \frac{d}{N} + (1 - d) \sum_{v \in adj[u]} \frac{p(v)}{deg(v)}$$

After this has been calculated, the top ranked sentences are taken by the TextRank algorithm to form the summary. However, in the LexRank algorithm sentence position and length are also taken into consideration. Also, when adding a sentence to the summary, the LexRank checks the sentence against the summary to ensure that

it won't be redundant. As a result of this extra step, LexRank is considered more suitable than TextRank when summarising multiple documents, whereas TextRank is only normally used for summarising a single document.

3.2.2 Abstractive Summarisation Techniques

There are six common methods for abstractive summarisation, which can be split evenly into two distinct categories [17]. These two categories centre around the creation of a representation of the given document:

- **Structure based methods** consist of techniques that involve determining the important information in a document by considering its structure [18]. Ways of doing this include fitting the given document to a template, or converting the text in a tree-like structure.
- **Semantic based methods** involve building a semantic representation of the given document and then feeding that into an algorithm that generates natural language that forms the final summary.

Structure based summarisation

Tree based summarisation

With tree based structuring, the first step is to create a dependency tree to represent the document. A dependency tree is a tree with a node for each word in a sentence, the links between the trees show which words depend on each other. For example, given the sentence *This is an example*, we can construct a tree as in Figure 3.1:

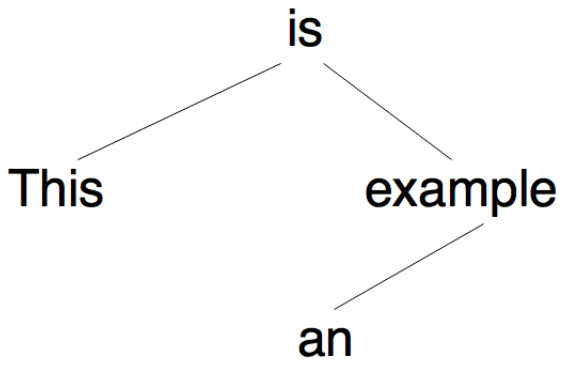


Figure 3.1: *A possible dependency tree for the sentence* This is an example

In the tree (Figure 3.1), the word *This* is dependent on *is* and so is linked. *an* is linked to *example* in a similar way, and *an example* is dependent on the verb *is* and so also has a link to it.

Once dependency trees have been created for each of the sentences a similarity algorithm is used to find sentences that are similar. The common phrases between these similar sentences are then taken and form the basis of the final summary. A language generator then combines the common phrases and arranges it to create a final set of summary sentences.

An obvious disadvantage to this method is that if only common phrases are taken from the sentences, context could be lost from some of the sentences. However, on the other hand, the use of a language generator means that a more coherent, more grammatically correct summary is formed.

Rule based summarisation

In rule based summarisation [8], the process is centred around a list of pre-determined categories. With each of these categories, there is a pre-determined set of questions to be answered.

For example, with a theoretical category *Product Launch* we could have the following questions:

1. What is the name of the product?
2. What company is launching it?
3. What type of product is it?
4. What's new about it?
5. What price is it retailing at?

This represents only a subset of the possible questions we could have for this category.

To perform rule based summarisation, the first step is to analyse the document and determine which category it fits best. Once that's been determined, the next stage is to analyse the text to find answers to the questions corresponding to that category. These answers are then fed in to a natural language generator that forms sentences, and thus the summary.

Results for this method of abstractive summarisation have been promising, but the method has a major disadvantage in that the list of categories and questions needs to be pre-determined. As a result, it might not be an optimal algorithm to use for

the ever-changing world of news reporting.

Ontology based summarisation

Methods of ontology based summarisation have been developed, primarily using domain based ontology. In this method a 'domain expert' defines a domain ontology for a news event. Each new document is then classified into a topic using these domain ontologies. Important phrases are determined by how close they are to items in the ontology. These phrases are then passed into a natural language generator to form the final summary sentences.

A key disadvantage to this method is that a lot of the domain ontologies has to be manually determined by the 'domain experts' and so can be very time consuming. As a result this may also not be particularly optimal for summarisation of news events.

Semantic based summarisation

Multimodal Semantic summarisation

Multimodal semantic summarisation can work on documents that contains both text and images. First, a semantic model is built to represent the document. This model is made up of different concepts that are surmised from the text. For example, the sentence *Multimodal Semantic summarisation is an example of an algorithm that performs abstractive summarisation* could form the concept shown in Figure 3.2:

Concepts are gradually filled with more information as the entire text is analysed. Links are also added between concepts that share some relationship. For example in Figure 3.2 if there was another sentence that surmised a concept called *Abstractive Summarisation* then there would be a link from *Algorithm1* to that new concept.

The next step is to rank the concepts. This is done by taking into account the completeness of the concept, and the number of links that the concept as to others. This way, the concepts are ranked by which is most important to the original document. Once the key concepts have been identified summary sentences can be generated featuring these concepts.

Information Item based method

Information Item based summarisation [7] relies on the content of the summary being determined from an abstract representation of the original document, rather

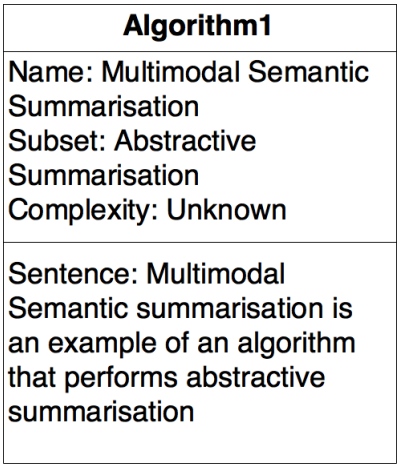


Figure 3.2: *A possible concept created from the analysis of the sentence* Multimodal Semantic summarisation is an example of an algorithm that performs abstractive summarisation

than the sentences from the document themselves. To do this, the document is first scanned so that Information Items (InIt) can be generated. An information item is defined as being 'the smallest element of coherent information in a text or sentence'.

Once the information items have been created, they are then ranked using frequency analysis to find the most important predicates and entities from the original document. This step is near identical to the term frequency stage in extractive summarisation (Section 3.2.1). These information items that are ranked highly are then combined and fed into a natural language generator to form the final summary sentences.

Semantic Graph summarisation

Semantic Graph summarisation centres around a rich semantic graph (RSG). The document is first converted into a RSG. Each node in the RSG represents a noun or verb in the document, and the links between the nodes represent the semantic and topological relations between these nouns and verbs. In the second stage, heuristic rules are used to reduce the semantic graph to a more minimalistic version. This will form the basis of the final summary. In the final step, the minimised RSG is passed into a generator that creates the final summary sentences.

The steps are outlined in the flowchart provided in Figure 3.3.

Semantic Graph summarisation has had success with producing an abstractive summary that has fewer redundant sentences, and is also good at producing grammati-

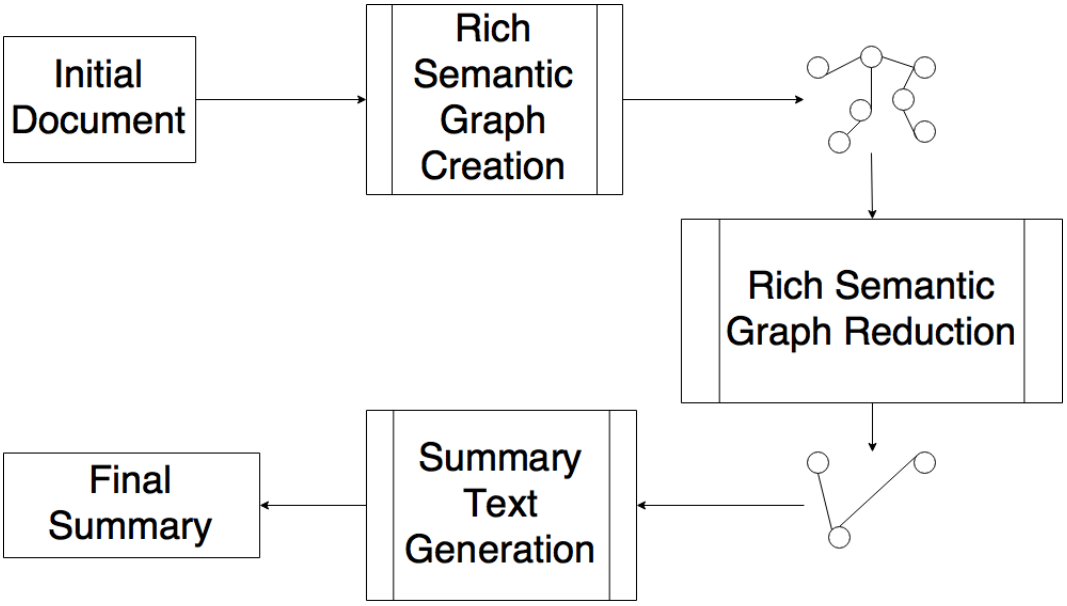


Figure 3.3: *This shows the process of semantic graph summarisation in a flowchart based on those provided in [17, 18]*

.

cally correct sentences. However, it's only designed for use with a single document as input. As a result it might not be suitable as a method for summarising the multiple documents needed for the aggregator, unless it's combined with another method.

3.3 Natural Language Processing Libraries

A lot of the summarisation techniques specified earlier, especially for abstractive summarisation, would require a full analysis of the semantics of a body of text. As a result, it's clear that a Natural Language Processing library will be needed for this task.

In the book *Natural Language Processing in the kitchen*, Anthony Pesce says 'Natural Language Processing is a field that covers computer understanding and manipulation of human language, and it's ripe with possibilities of for newsgathering. You usually hear about it in the context of analyzing (sic) large pools of legislation or other document sets, attempting to discover patterns or root out corruption.'

Therefore, a Natural Language Processing (NLP) library could you be used for various aspects of summarisation, as well as potentially in areas such as clustering

and labelling. With this in mind I started to do some research in the area to identify what features could prove useful going forward.

3.3.1 Aspects of Natural Language Processing

There are several different tasks that a Natural Language Processor. A few key ones that are most likely to be required for the project are explained briefly below:

- **Tokenisation** is the splitting of words in a given document.
- **Sentence Segmentation** is the splitting of sentences in a given document.
- **Part-of-Speech (POS) tagging** takes a list of tokens and analyses them, returning tags for each. A tag represents the grammatical function of the token in the sentence (for example if it is a noun, verb or adjective).
- **Named entity extraction** identifies the proper nouns within a document, such as people, dates or locations, as well as other categories of noun.
- **Chunking** takes a list of tags from a POS tagger and groups sets of tokens by their function in a sentence. Examples can include noun phrases and verb phrases.
- **Parsing** takes a sentence and develops a tree showing the functions of each Section of the sentence.
- **Coreference Resolution** identifies noun phrases within a document that are the same. This can commonly be used for replacing pronouns with the original noun phrase.

3.4 Conclusions

After conducting the background research I was able to come up with the following basic conclusions as to how to lead in to the design phase of the project.

- In **Topic Modelling** (Section 3.1.1) the better initial approach will be to employ (or to use a library) that employs Latent Dirichlet Allocation (LDA) as opposed to Latent Semantic Indexing (LSI). LDA uses a generative probabilistic model, which has in practice produced significantly more accurate results. Time shouldn't be of much concern in this case, as the model won't have to be generated very often. As a result, LSI's major advantage is rendered irrelevant.

- For **Topic Labelling** (Section 3.1.2) following the algorithm set out by Lau, Grieser, Newman Baldwin would be the best option, as it has had strong success in its methods (although admittedly this is their own claim). A potential issue to consider however is that the algorithm would require several (and perhaps dozens) of calls to Wikipedia. This could be time intensive.
- With **Clustering** (Section 3.1.3), there are significant disadvantages with the techniques of centroid clustering and density clustering. With the first, the number of clusters need to be known, and with density clustering accuracy can be lower than needed. Out of the two hierarchical clustering approaches, agglomerative would probably be the most useful, as it could be slightly faster than a top-down divisive approach. A vector score based on Term Frequency-Inverse Document Frequency is a tried and tested metric for clustering, and will also come in handy.
- For **Summarisation** (Section 3.2), a combination of extractive and abstractive summarisation would probably be best. The LexRank approach for extractive summarisation could help combine the multiple articles into one document, which could then be fed into the semantic graph abstractive summarisation algorithm.
- **Natural Language Processing**, (Section 3.3.1) will come in handy, especially in the summarisation phase. For example, with abstractive summarisation, it could be used to analyse the grammatical structures of the text in the semantic graphing phase.

4 Design

In this section I set out the overview of the design for the product. There are three important aspects to consider.

Firstly, how the front end should look, and what the underlying infrastructure and flow of it should be.

Secondly is the back end, and more importantly the process of an article being summarised, starting with taking an article from an outlet and going all the way until the summary itself is generated.

The last point that is considered is the infrastructure of the back end, namely the storage system and the company used to host the server.

4.1 Front End Architecture Diagram

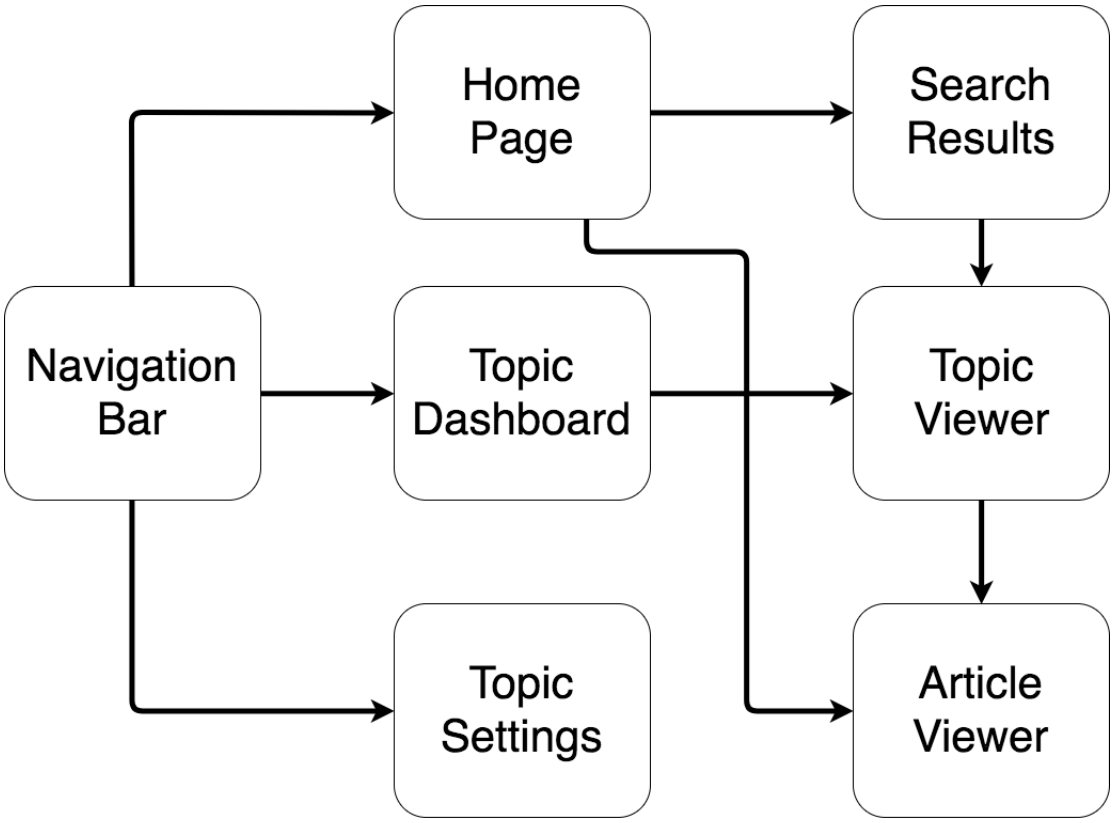


Figure 4.1: *A diagram showing the expected architecture of the front end application*

Figure 4.1 depicts the expected flow for the front end. A navigation bar will be present on all screens, and will provide links to the Home Page (Section 4.2.1), the Topic Dashboard (Section 4.2.2) and Settings pages (Section 4.2.5).

From the Home Page, the user can search for a topic, taking them to the Search Results screen (Section 4.2.3). They can alternatively click on one of the latest articles, which will in turn take them to the Article Viewer (Section 4.2.6).

Both the Topic Dashboard and the Search Results screen will display a list of topics in some form - in the case of the dashboard, it will be a list of the user's subscriptions, and in the latter a list of search results. They thus both have a direct link to the Topic Viewer (Section 4.2.4), which can be accessed by clicking on one of the items in the list.

The Topic Viewer presents the articles for the topic, and clicking on one will take the user to the aforementioned Article Viewer.

The Topic Settings screen is a standalone screen accessible from the Navigation Bar on any page.

4.2 User Story

4.2.1 Home Page

If a user has not logged in, the first page they will see on opening to the website will be the Home Page. The key to the design of the home page is that it's easy to get started for a user. They are presented with a large search box that they can use without having to sign in. There'll also be a row of trending articles displayed below the search bar. This row will consist of icons consisting of images and titles below, as shown in the wireframe. There's a navigation bar at the top, that is present on all screens, with a button to register and a button to sign in on the top right, and a link to information about the application itself on the top left.

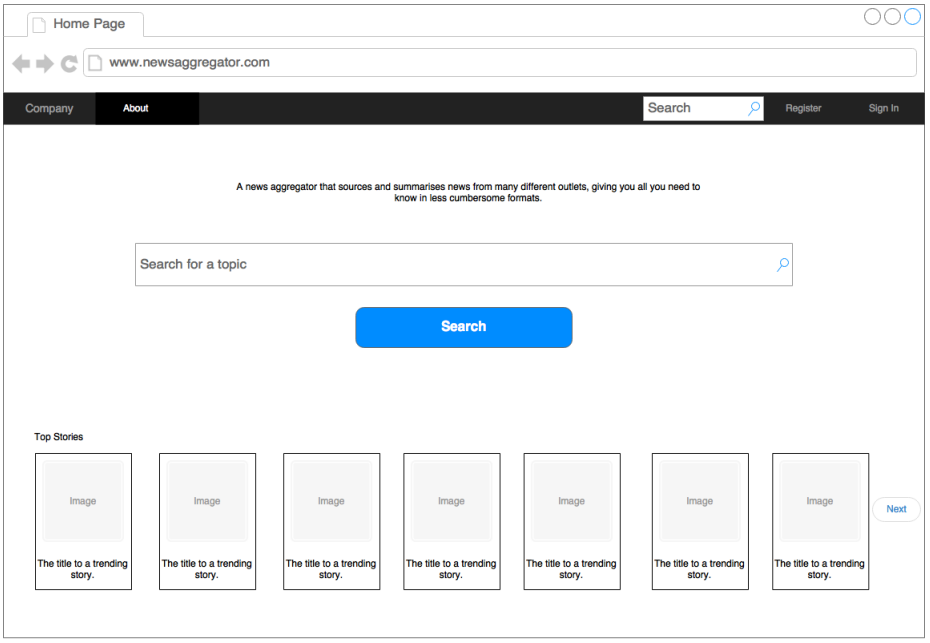


Figure 4.2: *A key focus of the home page is that the user can easily search for articles without having to sign in or register.*

4.2.2 Topic Dashboard

For a user that is logged in, or alternatively a user that has logged in (or signed up) from the home page will first see the topic dashboard screen that shows sets of articles corresponding to each topic they are subscribed. For users who don't have any topics that they are subscribed to, they will instead go back to the home page. The topic dashboard initially shows leading articles from across all the topics they are subscribed to. They can drag to one direction (akin to a sideways swipe on a phone or tablet) to see articles pertaining to the next topic.

Articles themselves are presented as panels. On a panel is an image for the article, with the title overlaid. In the top right hand corner are icons corresponding to the sources that the article has been summarised from. Leading articles (those out of the most recent that have the most sources attached) are placed in the larger panels on the top left and bottom right of the screen, as shown in the diagram.

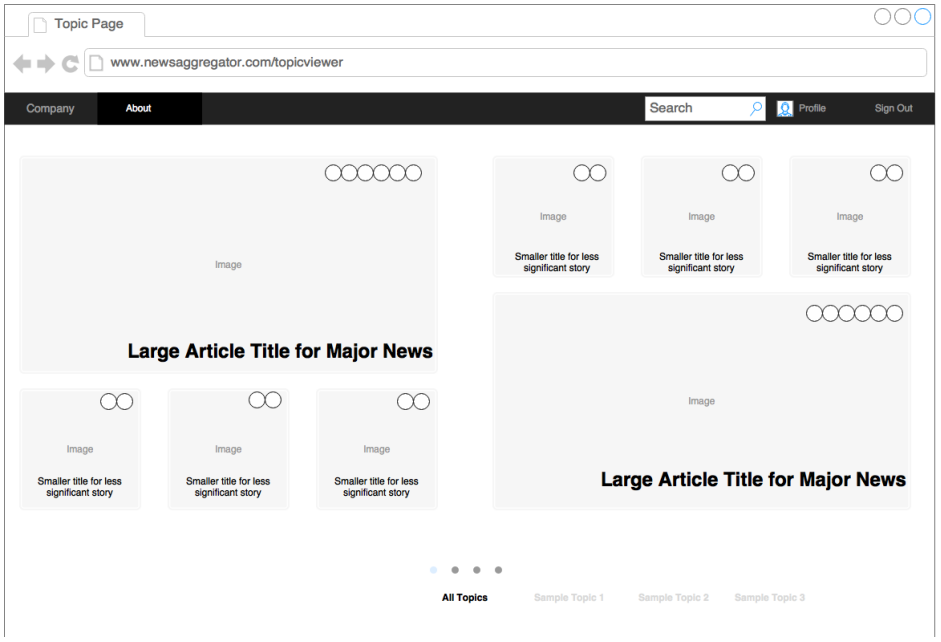


Figure 4.3: *The topic dashboard has the ability to show articles across all topics, or pertaining to a specific topic.*

4.2.3 Search Results Page

The search results page is designed to allow users to see as much as possible and do as much as possible without having to leave the page. The results are presented in

the application's panel design, with an image corresponding to each panel. Below the panel, is a description of the topic, that is likely to be created from the first paragraph of the Wikipedia [36] article about that particular topic. Below this are two buttons, one to view the topic's articles, which will go to a topic viewer page (shown later), and a subscribe button. If logged in, the user is taken to the topic settings page, where they can choose any particular preferences they have for this specific topic. If the user isn't logged in, they are taken to a page allowing them to log in or register, before being taken to the preference page.

It's necessary at this stage to have a user sign in if they want to subscribe to a topic. This way, they can access their topics from anywhere. In addition to this, the search bar is still prominent on the search results page, so that users can edit their queries easily in case of minor errors.

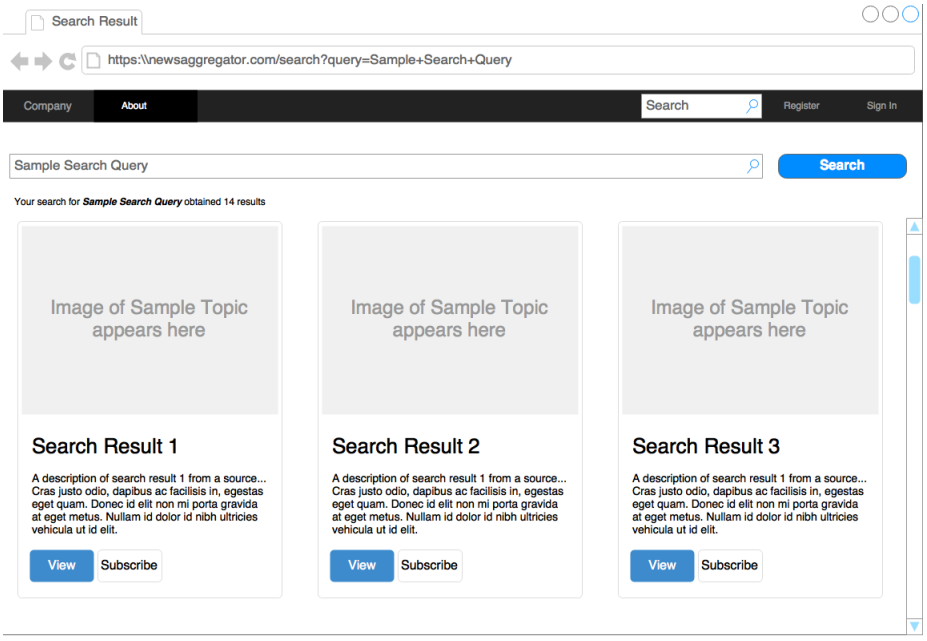


Figure 4.4: *The Search Results page has a focus on showing users as much information as possible about each topic.*

4.2.4 Topic Viewer

The topic viewer is accessed if a user has clicked on the view button from the search results page. This screen is similar to that of the topic dashboard. Minor changes include the page control at the bottom of the screen being replaced by page statistics, including figures about how often articles are created, the number of subscribers,

and the total number of hits that the page has had. The other major difference is the presence of a subscribe button in the top right hand side of the screen. As with the search results page, this leads to the topic preferences screen if logged in, and to a page prompting a user to sign in or register when not already logged in.

The remaining aspects of the screen are the same - namely the panelled articles, and the icons in the top right of each panel that indicate which sources the article originated from.

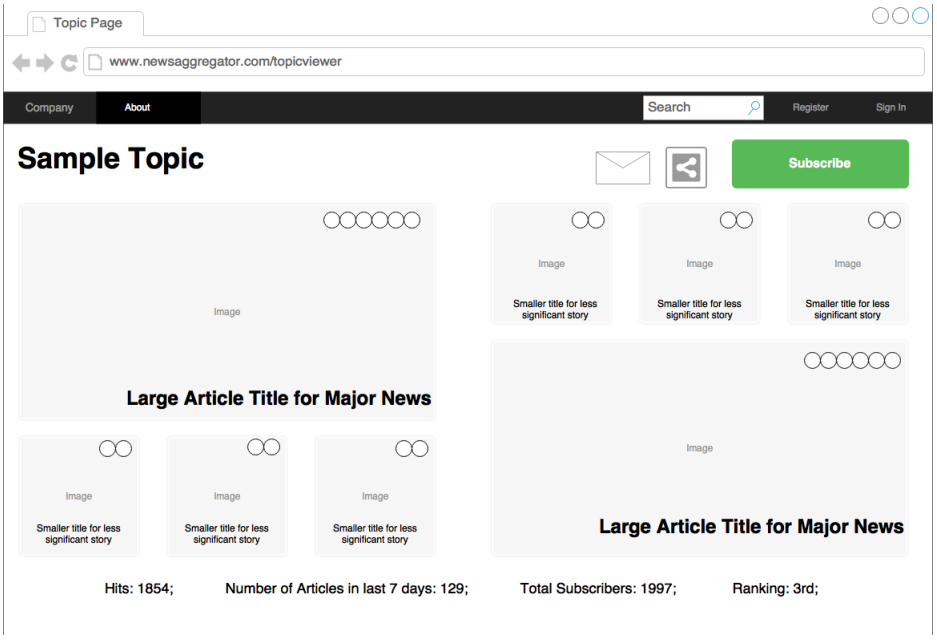


Figure 4.5: *The topic viewer is similar in structure to the topic dashboard, but with the addition of statistics and a subscribe button.*

4.2.5 Topic Settings

The aforementioned Topic Settings page sets out two preferences from the user for a specific topic. These are namely:

- **Should the topic's articles be included in the user's email digest?**

  This is answered by a simple on/off switch.

- **Which sources should be used to construct summaries**

This aspect is done using the panelling effect used in other screens. Each news outlet's logo forms the basis of the panel, with the title of the outlet below. These panels form buttons that the user can click on to either select or deselect an outlet. The panel will either have a coloured outer rim, or an embedded effect to indicate selection.

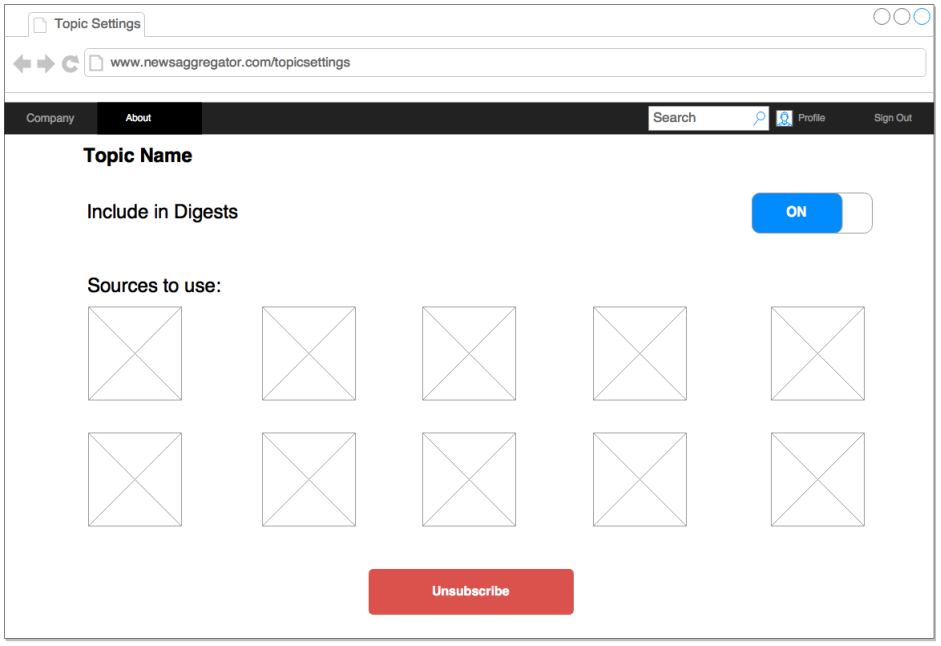


Figure 4.6: *The Topic Settings page allows users to specify key preferences about the topic.*

4.2.6 Article Viewer

When the user clicks on an article panel in the topic viewer or topic dashboard they are presented with the summarised article itself. This screen is modelled on standard news interfaces, and produces the headline, image and article body on the left hand side of the page. On the right hand side are links to the original articles that the summary was generated from, and links to other articles that are in the same topic.

Like most other screens in the application, there is also a share button on the top right, and an email button.

Figure 4.7: *The Article Viewer is modelled on standard news outlets' interfaces.*

4.2.7 Digest Viewer

The digest viewer will show users (who have requested them) their daily digest. The form would be nearly identical to that seen in the topic dashboard of Figure 4.3. However the articles would be made up of the ten latest articles across the user's subscriptions, rather than from a single topic. The link to this page would be sent daily to the user via email.

4.3 Back End Flow Diagram

In Figure 4.8 the expected flow of the machine learning aspects of the back end is presented. The process of an article being summarised is as follows:

1. The **Article Fetcher** queries the various News APIs used for new articles.
2. The **Article Curator** takes the response from the News APIs and performs any scraping that may be necessary, before passing the article to the Machine Learning phases.
3. The article is then passed to the **Topic Modelling** phase, which identifies which topics it consists of.

4. The **Topic Labelling** Section takes the results of the Topic Modelling phase, and proceeds to label its topic.
5. **Clustering** then occurs. The article, along with its topic label is passed to this phase, and the program pulls articles from the database that have already been assigned the same topic. These articles are then clustered, and the cluster containing this article is passed to the next phase.
6. Now that the cluster has been passed, **Summarisation** occurs on the article, which is then put in the Summarised Articles database, ready to be called for a user to read.

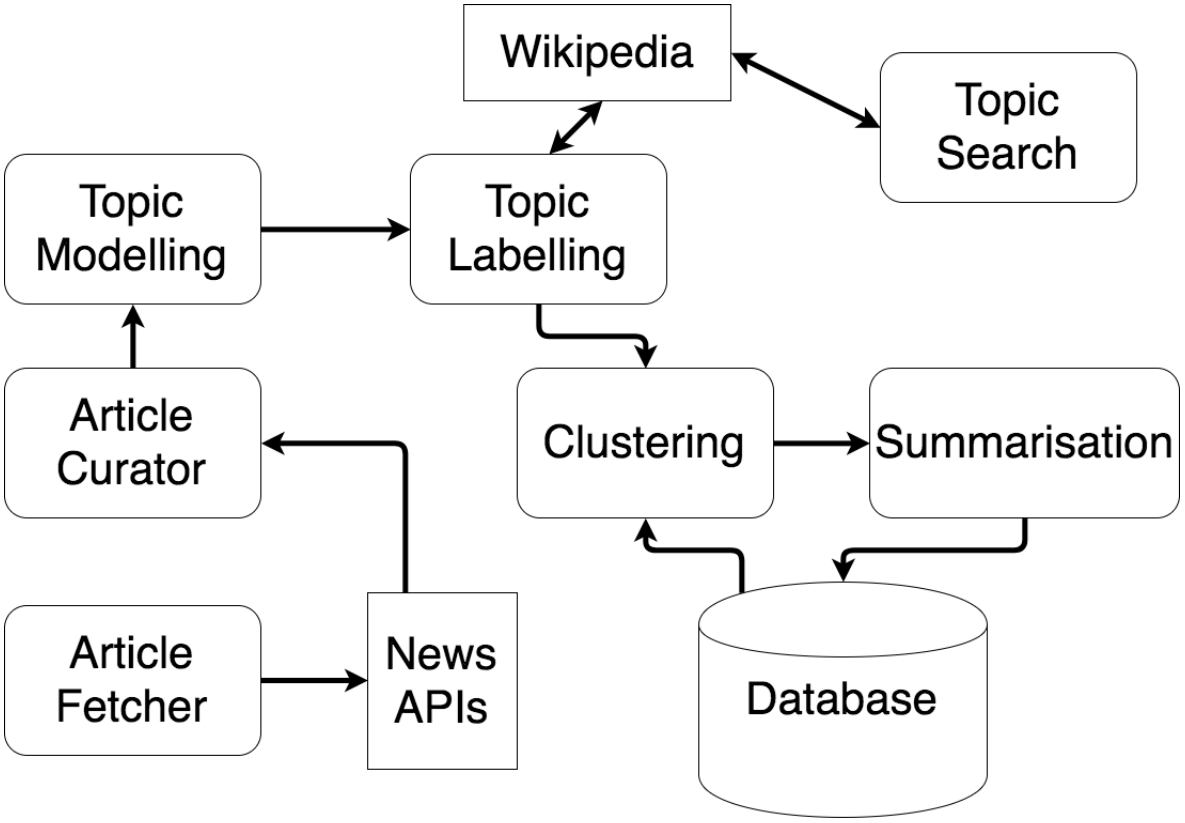


Figure 4.8: *The expected flow of the Back End.*

4.4 Infrastructure

4.4.1 Hardware

One of the first key decisions I made about the infrastructure surrounded the hardware that the server would be deployed on.

Amazon Web Services

Amazon Web Services (AWS) is a set of cloud computing platforms provided by Amazon. A key benefit to it is that it has a free tier, which allows for a subset of their services to be used without charge.

Some of the key platforms offered by AWS include:

- **EC2** allows a user to create a single virtual machine for free on which to run software. This would be with a limit of 750 hours of use per month. In my case this would be used for running the back end of the News Aggregator.
- **S3** is a storage system provided by Amazon that allows a user 5GB of space for free.

Digital Ocean

Digital Ocean is an infrastructure provider that provides 'cloud computing for developers'. Digital Ocean offers 'droplets' that can act as virtual machines, on which I would deploy my backend. There are no free tiers, but the student pack provided by GitHub would cover the first two months of charges.

In addition to this, one of the advantages of Digital Ocean is that if resizing of the virtual machine is necessary, it can be done in a matter of minutes, thus making the final solution very highly scalable.

In contrast though, unlike Amazon, Digital Ocean don't provide alternative microservices such as S3. Having said that, it could be argued that the droplet itself would offer that with its storage capabilities. In fact, the cheapest droplet offered by Digital Ocean (priced at $5 a month), entails a virtual machine with a 20GB SSD disk, which is four times the amount provided by the free tier in AWS.

4.4.2 Database

MongoDB

The first database solution that I researched was MongoDB, as I have had prior experience with it.

MongoDB is a document database that is designed to be both flexible and highly scalable. Documents are written in a format that is highly similar to JSON. This format is what results in a Mongo database being flexible, as the schema of documents in the collection can be changed quickly.

MongoDB allows for the connection of items between tables through linking. Auto-generated 'ObjectId's (that are generated based on the timestamp) allow for easy referencing of an item in another.

A MongoDB database would have to remain in a virtual machine on either Amazon or Digital Ocean, and will therefore have to count against the storage provided by that virtual machine.

DynamoDB

DynamoDB is a No-SQL database provided by Amazon Web Services as a separate micro-service to EC2 and S3.

The structure of a document in DynamoDB is more rigid than in MongoDB. It is primarily designed as a key-value system, with a primary key. Searching in the database is driven by this primary key, or by 'secondary indexes'. These secondary indexes need to be declared when the table is first created. Therefore this means that if there needs to be a schema change, the table needs to be recreated.

A DynamoDB database doesn't need to be stored in the virtual machine, but there are trade offs. Whilst Amazon Web Services provide 25GB of storage for the database as part of the free tier, there are throttles on the throughput of the database. As part of the free tier, a user is allowed to allocate 25 'units' of throughput to both reading and writing from a database, where one unit corresponds to transfer of 1KB per second. Beyond these 25 units, Amazon would start charging.

4.4.3 Infrastructure Decisions

Comparison table

Table 4.1 shows the key points of comparison between the four main options for hardware and database combinations.

Decision Process

Infrastructure	Advantages	Disadvantages
AWS and MongoDB	• AWS provide free services • Mongo provides a flexible schema, meaning that the table structure can be changed further down the line.	• Mongo would need to be hosted on the EC2 instance, or on S3, which would limit the free storage to 5GB.
AWS and DynamoDB	• DynamoDB would provide 25GB of storage on its own, meaning that I wouldn't have to use the S3 storage.	• DynamoDB's key-value schema is quite rigid, meaning that there isn't much flexibility on the model. • The throttling of reads and writes by DynamoDB could severely impact the performance of key back end tasks.
Digital Ocean and DynamoDB	• Digital Ocean offers the ability to resize the server as needs be within a matter of minutes.	• As before, there are some key issues regarding the speed of DynamoDB whilst on the free tier. • This solution would involve maintenance of a database solution provided by one company, and a server on another, which is potentially quite wasteful.
Digital Ocean and MongoDB	• MongoDB database can be stored comfortably on the Digital Ocean droplet • The flexibility of the MongoDB model would mean that fields can be removed and added easily from the schema, allowing for further changes down the line.	• Whilst covered by vouchers for the first two months of use at least, the Digital Ocean instance would require expenditure if used for longer than that period, versus AWS EC2 which would be free for at least twelve months.

Table 4.1: *Comparing the advantages and disadvantages of potential combinations of hardware and database options.*

I tried all possible combinations listed in table 4.1 before finally arriving at the decision of using Digital Ocean and MongoDB.

I initially started with the configuration with AWS and MongoDB. However, it became clear quite quickly that there was a distinct possibility that the memory provided by AWS would not be sufficient to hold a large number of articles (about six weeks or greater) in the database. There certainly would not be scope for adding extra outlets near the end of the project.

In an attempt to counter this, and trying to avoid solutions that would require the spending of money, I then switched the database option to DynamoDB. This worked well for a time, but as the database grew, it became clearer that the throttling of the read and write operations to my database was going to cause major issues down the line for my background jobs on the server, as well as for simple API methods. In order to counteract this, I resolved that I would use vouchers for AWS from the GitHub student pack in order to fund some extra read-write operations per second.

Having resolved to do this, I then decided to make use of Digital Ocean. This was because I wanted to experiment to see the effects of having an instance with extra RAM would do for the running of my program, and knew that the vouchers on the GitHub student pack would easily cover a Digital Ocean droplet until the end of the project. What I found was that my background jobs (such as pulling in articles, labelling, clustering and summarisation) were being performed faster in general, and so I decided to permanently move to Digital Ocean.

In the end, a final nail in the coffin for my use of Amazon Web Services came when I wanted to change the way my articles table worked in the database. I had wanted the ability to search by date published (as I was considering at the time removing articles published before a certain date), but found that I would have to recreate my entire table in order to add this functionality to my table. As a result, and combined with the fact I now had a much larger amount of storage space on my new Digital Ocean droplet, I took the decision to make a new MongoDB database, knowing that it was still early enough in the development process that losing all my previous database work would not affect the finished product, or the development timeline.

5 Implementation

5.1 Language and Platform Choices

5.1.1 Front End

The survey that was conducted in Section 2.1.2 clearly presented the result that a website would be the preferred option for the application, followed by a mobile application in either iOS or Android. Therefore I've decided to follow this and create a WebApp as a primary platform. If time permits, I'll then develop the application for a second platform, which will be a mobile platform.

For the WebApp I will primarily use Bootstrap for the Interface. One design framework I used to great effect during my Industrial Placement in Summer 2016 was React and Redux. React is a JavaScript library developed by Facebook that is designed for building scalable and reusable user interfaces. Redux is a design pattern for JavaScript that is designed around a central predictable state container that is the single source of truth for a User Interface.

The benefit of this framework is that it allows for very clean code in the user interface. This is because the concept of a single source of truth allows for data flow that is purely in only one direction, making it easier to understand.

5.1.2 Back End

The back end will be constructed using Java. The reasoning behind this is that I have experience from my Industrial Placement using Java as the basis of a server and a back end.

5.2 Database Schema

5.2.1 Schema Diagram

Figure 5.1 shows the basic components of the various collections in the Mongo Database, along with some of the connections involved. Further details on each are given in each of the following sections.

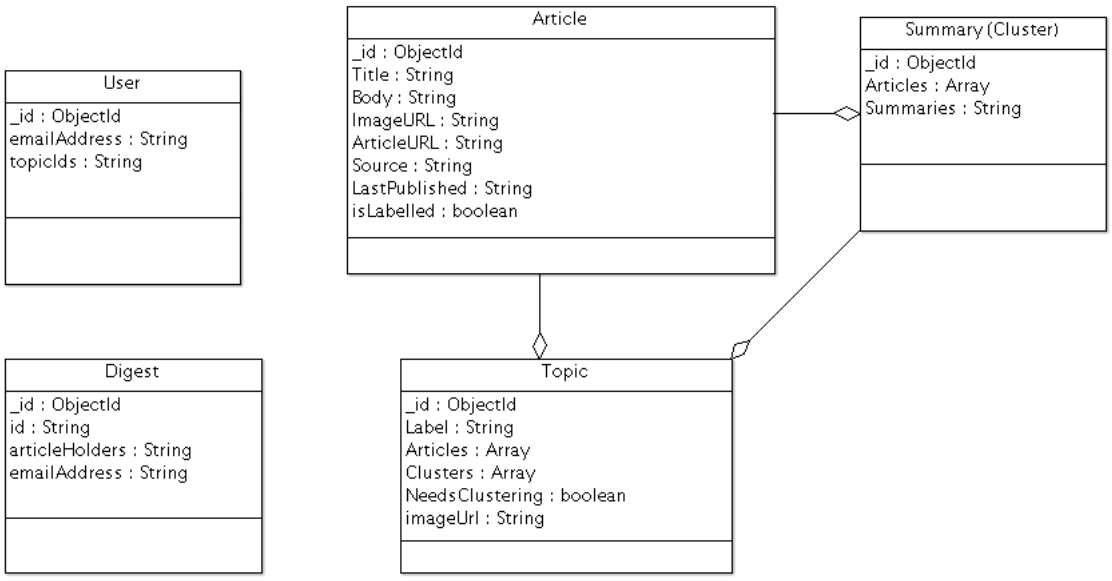


Figure 5.1: *A diagram of the core MongoDB schema.*

5.2.2 Articles

A sample document for an article can be seen below:

```
{
    "_id" : ObjectId("591df7cfacea820abe29709f"),
    "Title" : "Kirsten, Simons on five-man panel in hunt
       for new SA coach",
    "Body" : "Cricket South Africa has nominated a five-man
        panel including two former national coaches, Gary
       Kirsten and Eric Simons, to recommend a suitable
       candidate for the position of head coach, which it
       aims to fill by the beginning of September
    ...
    ",
    "ImageURL" : "http://www.espncricinfo.com/db/PICTURES/
       CMS/157600/157626.5.jpg",
    "ArticleURL" : "http://www.espncricinfo.com/southafrica
       /content/story/1098337.html",
    "Source" : "espn-cric-info",
    "LastPublished" : "2017-05-18T10:42:38Z",
```

```
    "isLabelled" : true
}
```

Listing 5.1: *A sample document in the Article table*

The purpose of the article table is for storage of raw article data that has been brought in by the article fetcher. Most fields are self explanatory. The *isLabelled* field is defaulted to false when an article is first brought in, and is then changed to true when the Topic Labeller has labelled it. This is present so that articles that either weren't labelled, or caused an error when being labelled, can be found with a simple search in the database.

5.2.3 Topics

A sample document for a topic can be seen below:

```
{
    "_id" : ObjectId("591df830acea820aebf55465"),
    "Label" : "Opposition (politics)",
    "Articles" : [
        ObjectId("591e172dacea820aebf55a74"),
        ObjectId("5925f959acea821a75b831bc"),
        ObjectId("59281cddacea821ab611d3ed"),
        ObjectId("59291a47acea82594af9ef53"),
        ObjectId("592aa5daacea8264125210d3")
    ],
    "Clusters" : [
        ObjectId("591f8bc7acea821e031f4dc6")
    ],
    "NeedsClustering" : true,
    "imageUrl" : "https://upload.wikimedia.org/wikipedia/
      commons/d/d1/Stand_in_opposition_city_hall_boston.
      jpg"
}
```

Listing 5.2: *A sample document in the Topics table*

A topic object is a core concept in the back end. It keeps references to all articles that have been labelled with that topic, and also holds references to all the clusters (summaries) that are produced from these articles. The *imageUrl* field holds either the lead image for the corresponding Wikipedia article, or a placeholder if the Wikipedia article doesn't have an image.

The *NeedsClustering* flag performs a similar function to the *isLabelled* flag in the Articles table. It is defaulted to true, and is changed to false once all the clustering (but not necessarily summarisation) is completed. It is used again to find topics that may have 'slipped through the cracks', either due to an error, or for some other reason.

5.2.4 Clusters

A sample document for a cluster can be seen below:

```
{
    "_id" : ObjectId("591e47f7acea8221dcb97c42"),
    "Articles" : [
        ObjectId("591df7cfacea820abe2970a9"),
        ObjectId("591df7cfacea820abe2970b2")
    ],
    "Summaries" : "{\"[http://www.cnn.com/2017/06/01/
      politics/trump-paris-climate-decision/index.html,
      http://www.reuters.com/article/us-usa-climatechange-
      eu-idUSKBN18S5IT]\":[{\"sentence\":\"BRUSSELS The
      European Union said on Thursday it had made its
      position on climate change clear and was not engaged
       in last-minute lobbying of the Trump administration
       to keep the United States aboard the Paris climate
      accord.\",\"sentencePosition\":0.0,\"
      absoluteSentencePosition\":0,\"identifier\":26,\"
      relatedNodes\":[],\"source\":\"reuters\"}
    ...
    ]}"
}
```

Listing 5.3: *A sample document in the Summaries table*

An object in this table represents a Summary, and corresponds to the Clusters array in a Topic. It contains an array of references to the original articles used for the summarisation.

The *Summaries* field is a string representation of a map object, with the values representing to a summary, and the corresponding key representing a combination of articles used to make that summary (represented using their URLs). For example, in the full version of the sample document, there would be three keys in the string representation:

- **[http://www.cnn.com/2017/06/01/politics/trump-paris-climate-decision/index.html,http://www.reuters.com/article/us-usa-climatechange-eu-idUSKBN18S5IT]**, for a summary using articles from *CNN* and *Reuters*.
- **[http://www.cnn.com/2017/06/01/politics/trump-paris-climate-decision/index.html]**, for a summary using an article from *CNN*.
- **[http://www.reuters.com/article/us-usa-climatechange-eu-idUSKBN18S5IT]**, for a summary using an article from *Reuters*.

These represent all the possible permutations (of size greater than zero) of the articles in the cluster. Its primary use is in the customisation offered to users, that can be seen in more detail in Section 5.9.

5.2.5 Users

A sample document for a User can be seen below:

```
{
    "_id" : ObjectId("592c5288acea82147ca7f0d1"),
    "emailAddress" : "fake@gmail.com",
    "topicIds" : "[{\"topicId\":\"591e3c74acea821e2b7e6c02
      \",\"sources\":[\"wikipedia\",\"business-insider-uk
      \",\"daily-mail\",\"espn-cric-info\",\"metro\",\"
      mirror\",\"newsweek\",\"sky-sports-news\",\"the-
      telegraph\",\"the-times-of-india\",\"bbc-news\",\"
      bbc-sport\",\"bloomberg\",\"cnn\",\"cnbc\",\"espn
      \",\"four-four-two\",\"the-washington-post\",\"the-
      wall-street-journal\",\"associated-press\",\"the-
      guardian-uk\"],\"digests\":false}]"
}
```

Listing 5.4: *A sample document in the Users table*

A user object is very simple. The only field that is not necessarily self-explanatory is the *topicIds* field.

This is an array of strings, with each string forming a representation of a user's topic subscription (the topic being represented by the field marked *topicId*), and the settings for that topic. The two core settings for each subscription is the digests setting (a boolean flag indicating whether a user wants to receive email digests that

include this topic), and a list of sources that the user would like their news to default to (further information on this feature can be seen in Section 5.9). By default, when a user subscribes to a topic, defaults is set to false, and sources is represented by a list comprising all possible sources.

5.2.6 Digests

A sample document for a digest can be seen below:

```
{
    "_id" : ObjectId("591ec5c433625db581d77cc6"),
    "id" : "591ec5c433625db581d77cc6",
    "articleHolders" : "[{\"topicId\":\"593020
       de5826a72aa697a201\",\"articleId\":\"59322
       bc5acea820dd0c7ef37\",\"title\":\"Elon Musk '
       intrigued' by India's objective of all-electric cars
        by 2030 - Times of India\",\"imageUrl\":\"http://
       timesofindia.indiatimes.com/photo/msid
       -58964385/58964385.jpg?53722\",\"lastPublished
       \":\"2017-06-03T08:40:00Z\"},{\"topicId\":\"5930210
       d5826a72aa697a227\",\"articleId\":\"59324084
       acea820dd0c7f65b\",\"title\":\"Kathy Griffin loses
       ALL of her tour gigs in wake of Trump scandal\",\"
       imageUrl\":\"http://i.dailymail.co.uk/i/pix
       /2017/06/03/05/410C672100000578-0-image-a-41
       _1496464334669.jpg\",\"lastPublished\":\"2017-06-03
       T04:39:14Z\"},{\"topicId\":\"593020
       de5826a72aa697a201\",\"articleId\":\"59323
       dddacea820dd0c7f566\",\"title\":\"Putin says hacking
        of Democratic Party may have been CIA false flag op
       \",\"imageUrl\":\"http://i.dailymail.co.uk/i/pix
       /2017/06/03/05/410C6AD500000578-0-image-a-92
       _1496463783859.jpg\",\"lastPublished\":\"2017-06-03
       T04:28:04Z\"}
    ...
    ",
    "emailAddress" : "specialk109@gmail.com"
}
```

Listing 5.5: *A sample document in the Digests table*

A digest document represents a single daily digest for a user. Thus, a user who gets daily digests would have a single object for each day. Having said a user would have a single object for each day, it's worth pointing out that the digests are not linked to the User object. Whilst there is no concrete justification for this to be the case either way, a reason for the digests table and users table to not be linked in some way is that they simply don't need to be. A user is never called when a digest is retrieved, and similar a digest is never called when a user is retrieved.

The *articleHolders* field in the document above represents the articles in a digest. It is an array of strings, with each string a JSON representation of an object containing a topic id, article id, and basic information about the cluster (title, image url, and a publishing date). The reasoning behind having the topic id in addition to the cluster is due to the pattern matching employed in the URL for the article viewer on the website (this is explained in further detail in Section 5.9).

5.3 Useful APIs

5.3.1 News Outlets

Readership statistics for selected outlets in the United Kingdom

Statistics below are taken from the *Digital News Report 2016* [23] by the Reuters Institute for the Study of Journalism [24]. Viewership is given as a percentage of people who say they use the outlet at least weekly.

Outlet	Viewership	Political Stance
BBC	51%	Neutral
Mail Online	17%	Right
Huffington Post	14%	Left
The Guardian	14%	Left
Sky News Online	11%	Neutral
The Telegraph Online	9%	Right
The Independent	6%	Centrist

Table 5.1: *Readership statistics for selected outlets in the United Kingdom*

The Guardian

The Guardian [28] is a UK newspaper that commands about 14% of the viewership as outlined in Table 5.1. It is also known for having an API that allows a user full

and free access to all articles, including the article text itself. The API can access any article dated on or after 1999, and as long as not used for a commercial purpose, is completely free to use.

To obtain an article from *The Guardian* there are two calls that need to be made. First, we make a call to the search endpoint, which returns a JSON object similar to this:


```
{
    "response": {
        "status": "ok",
        "userTier": "free",
        "total": 1,
        "startIndex": 1,
        "pageSize": 10,
        "currentPage": 1,
        "pages": 1,
        "orderBy": "newest",
        "results": [
            {
                "id": "politics/blog/2014/feb/17/alex-
                   salmond-speech-first-minister-scottish-
                   independence-eu-currency-live",
                "sectionId": "politics",
                "sectionName": "Politics",
                "webPublicationDate": "2014-02-17T12:05:47Z
                   ",
                "webTitle": "Alex Salmond speech - first
                   minister hits back over Scottish
                   independence - live",
                "webUrl": "https://www.theguardian.com/
                   politics/blog/2014/feb/17/alex-salmond-
                   speech-first-minister-scottish-
                   independence-eu-currency-live",
                "apiUrl": "https://content.guardianapis.com
                   /politics/blog/2014/feb/17/alex-salmond-
                   speech-first-minister-scottish-
                   independence-eu-currency-live"
            }
        ]
    }
}
```

Listing 5.6: *A sample response to an API call to The Guardian*

From this, we extract the apiUrl, and call this, with the API key in the request headers. This will return the raw text for the article.

News API

A major issue is that most newspapers (for example, *The New York Times* [29]) only give the first paragraph to an article in their API, and don't give out their full articles as standard. There are solutions, such as Webhose [32], that offer feeds into this outlets, but at either a cost, or with a very limited number of requests per month.

News API [21] is a company that provides an API to get headlines for over seventy different news outlets. A call returns data in the JSON format as follows:

```
{
"status": "ok",
"source": "the-next-web",
"sortBy": "latest",
"articles": [
{
        "author": "Abhimanyu Ghoshal",
        "title": "Are these ridiculous headphones the way
           forward for music tech?",
        "description": "All the amazing tech we now have at
            our disposal for enjoying music is closing us
           off from other people instead of bringing us
           together. Is there hope yet?",
        "url": "https://thenextweb.com/gear/2017/01/26/can-
           music-tech-make-us-sociable-again/",
        "urlToImage": "https://cdn3.tnwcdn.com/wp-content/
           blogs.dir/1/files/2017/01/Vinci-hed-1.jpg",
        "publishedAt": "2017-01-26T13:12:10Z"
}
]
}
```

Listing 5.7: *A sample response to an API call to News API*

We can use the url provided for each of these results to get to the webpage of the corresponding article. However at this stage I'll need to create a scraper to extract the raw content of the article. There could be copyright implications for this step, even though the final production summary won't look the same as the raw content. These implications will be fully examined in the final report.

News API counts amongst its 70 sources the following:

- BBC News
- Mail Online
- *The Telegraph*
- *The New York Times*
- Associated Press
- *The Independent*
- *The Daily Mirror*

News API also has *The Financial Times* and *The Guardian*, although these are left off the list above as they have their own fully accessible APIs.

Wikipedia

Wikimedia (the parent company for Wikipedia [36]) provides documentation for the Wikipedia API on its website MediaWiki website. Here, I can search using the search term that a user has used (perhaps in searching for a topic), and get a result that is as follows:


```
{
    "query": {
        "searchinfo": {
            "totalhits": 4152
        },
        "search": [
            {
                "ns": 0,
                "title": "Albert Einstein",
                "snippet": "&quot;<span class=\"searchmatch
                  \">Einstein</span>&quot; redirects here.
                   For other uses, see <span class=\"
                  searchmatch\">Albert</span> <span class
```

```
                    =\"searchmatch\">Einstein</span> (
                    disambiguation) and <span class=\"
                    searchmatch\">Einstein</span> (
                    disambiguation). <span class=\"
                    searchmatch\">Albert</span> <span class
                    =\"searchmatch\">Einstein</span> (/?alb?
                    rt ?a?n?ta?n/; German:",
                "size": 124479,
                "wordcount": 13398,
                "timestamp": "2015-05-10T12:37:14Z"
            },
            ...
}
```

Listing 5.8: *A sample response to an API call to Wikipedia's API*

The search query can be altered to provide different information, such as the URL for the lead image, which can be used to show previews of various items to users. The Wikipedia API can also be used in the topic labelling aspect of the Machine Learning techniques (see Section 3.1.2).

5.4 Useful libraries

5.4.1 Mallet

Mallet (Machine Learning for Language Toolkit) is a java library that provides an interface for various natural language-based machine learning tasks. These tasks include:

- Document Classification
- Sequence Tagging
- Clustering
- Topic Modeling
- Information Extraction

Mallet in Topic Modeling

Mallet is particularly useful for Topic Modeling. Mallet uses a corpus in the form of a list of strings in order to train a set of topics. The library trains the topics using Gibbs Sampling and Latent Dirichlet Allocation.

Mallet also provides an inferencer that allows a user to submit a new document and receive a list of probabilities. Each of these probabilities shows the likelihood that the corresponding topic is an accurate classification of the document we are testing. Thus the user can then infer the actual topics that the document is about by simply taking the topics with the highest probabilities.

5.4.2 Natural Language Processing

There are several Natural Language Processing libraries available. The following are designed for use in Java:

Apache OpenNLP

OpenNLP is an open source library developed by Apache. The aim of the project is to support the basic aspects of natural language processing.

OpenNLP supports all of the basic tasks that were specified in Section 3.3.1 with varying degrees of success, plus 'document categorisation'.

Document Categorisation, as the name suggests, takes a document and categorises it. This could potentially be used in the topic modelling phase, but has the disadvantage that it doesn't come with a model for training, and the user of the library has to create their own. This is a contrast to the Mallet library, which can create a model when the user passes in the raw data.

When it comes to the other key tasks of Natural Language Processing, OpenNLP provides flexibility on models. They provide a default model in a variety of languages, including English. However, there is at least one different model file for each task, so space constraints need to be considered.

Name Finder

For using the name finder, there are different model files, depending on what types of proper nouns a user is looking for. These are:

- Date
- Location

- Money
- Organisation
- Percentage
- Person
- Time

Space Constraint Table

Table 5.2 shows each model and their respective sizes. As Apache's recommended method to bring the files in to the program is through the Java class FileInputStream their sizes can be important, as they make use of the Java heap.

Model Name	**Task**	**Size (MB)**
en-chunker	Chunking	2.6
en-ner-date	Name Finder: Dates	5.0
en-ner-location	Name Finder: Locations	5.1
en-ner-money	Name Finder: Money	4.8
en-ner-organization	Name Finder: Organisations	5.3
en-ner-percentage	Name Finder: Percentages	4.7
en-ner-person	Name Finder: People	5.2
en-ner-time	Name Finder: Time	4.7
en-pos-maxent	POS Tagging	5.7
en-sent	Sentence Detection	0.01
en-token	Tokenisation	0.44
Total if all used		**43.55**

Table 5.2: *A table showing the space required for the various models provided for use with OpenNLP*

Stanford CoreNLP

The Stanford CoreNLP is a toolkit provided for Java by the Stanford Natural Language Processing Group. Like Apache OpenNLP, CoreNLP also offers the same functionalities that were discussed in Section 3.3.1. However, Stanford go above and beyond this list in what they can provide. Their coreference and name recognition solutions are 'competition winning', whilst they also provide functionality for semantic analysis of text.

As a result, CoreNLP could have a major benefit for abstractive summarisation, through its coreference resolution.

Extracting elements from a document

The following is a code snippet, taken from the Stanford CoreNLP documentation that demonstrates how to perform certain Natural Language tasks from a given annotated document.

```
1
2  // these are all the sentences in this document
3  // a CoreMap is essentially a Map that uses class objects
     ↪ as keys and has values with custom types
4  List<CoreMap> sentences =
     ↪ document.get(SentencesAnnotation.class);
5
6  for(CoreMap sentence: sentences) {
7    // traversing the words in the current sentence
8    // a CoreLabel is a CoreMap with additional
       ↪ token-specific methods
9    for (CoreLabel token:
       ↪ sentence.get(TokensAnnotation.class)) {
10     // this is the text of the token
11     String word = token.get(TextAnnotation.class);
12     // this is the POS tag of the token
13     String pos = token.get(PartOfSpeechAnnotation.class);
14     // this is the NER label of the token
15     String ne = token.get(NamedEntityTagAnnotation.class);
16   }
17
18   // this is the parse tree of the current sentence
19   Tree tree = sentence.get(TreeAnnotation.class);
20 }
21
22 // This is the coreference link graph
23 // Each chain stores a set of mentions that link to each
     ↪ other,
24 // along with a method for getting the most representative
     ↪ mention
25 // Both sentence and token offsets start at 1!
```

```
26 Map<Integer, CorefChain> graph =
27   document.get(CorefChainAnnotation.class);
```

Listing 5.9: *Analysing an annotated document using Stanford's CoreNLP*

WordNet

WordNet is a large scale database developed by Princeton that offers a lexical database of the English language. It can primarily be used as a thesaurus, and could therefore be useful in the natural language generation of abstractive summarisation, as well as in finding similarities between sentences.

5.5 Machine Learning

5.5.1 Topic Modelling

For the Topic Modelling task I used the Mallet library, which is described in full detail in Section 5.4.1.

There are two sections to the topic modelling phase: Training, and Estimating.

Training

For training the model, I used the ParallelTopicModel class provided by the Mallet library. The class provides a parallel implementation of Latent Dirichlet Allocation. There are several key parameters that need to be passed in to the model before it can be estimated:

- **numTopics**, the number of topics to model the corpus into.
- **alphaSum** is distributed evenly across all topics. For example, an alphaSum of 1 over 100 topics would result in each topic having an alpha score of 0.01. An alpha score is viewed as a smoothing term. It ensures that the probability of a given topic being in a document is never zero.
- **beta** is similar to alpha, and is the corresponding smoothing factor ensuring that the probability of a word being in a given topic is never zero.
- **numIterations** is the number of iterations the trainer should run through before presenting a final model.

Initially I attempted to model topics by feeding in articles and seeing what topics came up. However, it was clear that this wasn't ideal, as frequently occurring words, such as 'a', 'the' and 'and' were too prominent in the list of words in each topic, and thus were skewing any results from the estimation stage.

I then tried to use the provided 'stoplists' file. With this file, Mallet aims to filter out common words with no real significance out of the given texts, before training topics. It became noticeable quickly though that the stoplists file wasn't extensive enough, as it was missing contracted words, such as 'it's'. Given the frequency of these words, I dismissed this idea and set about a new solution.

As a final solution, I decided to remove all insignificant words by removing all non-nouns from the article bodies. I did this using the Apache OpenNLP library, and found that this made (to the naked eye) the topic models look a lot better.

My final parameters can be seen in Table 5.3.

Parameter	**Final Value**
alphaSum	1.0
beta	0.01
numTopics	500
numIterations	1500

Table 5.3: *A table showing the final parameters chosen for the topic modelling training*

Initially, the articles would be used to retrain the model each time a set of articles were about to be labelled. However, in the optimisation phase of the project, I changed this so that the training would happen once a week, and a model saved on file. Full details and justification can be found in Section 7.1.1.

Estimation

The body of the article to be estimated is passed in to the model. This was initially the whole article, but after I decided to keep only nouns for training, I did the same for estimation.

There are three key parameters for estimation. These are:

- **numIterations**, which is the number of times to iterate through the model before producing a final estimate.

- **thinning**, which is the number of iterations to run before saving an interim model.
- **burn-in** is the number of iterations to perform before beginning the estimation. The reasoning behind this is that since the first iteration begins with a random distribution, the first set of iterations are unlikely to be truly representative of the actual model. The burn-in period could potentially be compared to the warm-up of an athlete.

After the topic model for the new article has been estimated, I prepare the result for topic labelling. To do this, I parse the data into a new custom class I created, that stores the word and the 'distribution'. The distribution is calculated by the Mallet library, and is an indication of the likelihood that an article belongs to a certain topic.

The final parameters chosen can be seen in table 5.4.

Parameter	**Final Value**
numIterations	2500
thinning	40
burn-in	50

Table 5.4: *A table showing the final parameters chosen for the topic modelling estimation*

5.5.2 Topic Labelling

When I first implemented the Topic Labelling phase of the backend, I aimed to emulate the algorithm of Lau, Grieser, Newman and Baldwin as closely as possible. This algorithm is detailed in Section 3.1.2. However, once this algorithm had been implemented, it was clear that there were some potential flaws when it applied to my project.

A key issue in the topic labelling phase was that of speed. The algorithm required dozens (and potentially hundreds) of calls to the Wikipedia API. There are ten calls in step two (searching Wikipedia for the original top ten topic terms), followed by calls on every single noun chunk found in step three of the algorithm. Calculating the Related Article Conceptual Overlap scores (RACO, see Section 3.1.2 would also require every single 'outlink' to be found, and every category of it. Given this needs to be done for every single noun chunk found earlier, it was easy to see why this could become prohibitively costly in terms of speed.

A simple test run using five articles from *The Guardian* of varying lengths resulted in each taking at least fifteen minutes to complete the topic labelling process. Given the potential volume of articles expected per day, I reluctantly dismissed the algorithm as too expensive for the project.

In order to streamline the labelling process I removed major aspects of the algorithm. This included removing the creation of noun chunks, and the RACO calculation. I also looked to some simple heuristics, such as isolating names using the OpenNLP library and automatically adding them as labels.

Brief overview of final labelling algorithm

1. **Find all proper names** in the article body using the Apache OpenNLP library. I used the models for Locations, Organisations and People in this phase.
2. **Search Wikipedia** for the top fifteen results for each of the ten topic words provided by the modelling algorithm.
3. **Retrieve text from Wikipedia articles** for all results in step two that aren't in the list of names found in step one.
4. **Perform Candidate Ranking.** This step is described in more detail below.
5. **Add top 18 results to candidates from step one.** The number 18 was found via trial and error, with the aim of striking a balance between too many potential labels and too few. Too many could result in a very slow clustering process, as there are too many topics that need clustering, causing a backlog. Too few could reduce accuracy.

This significantly increased the speed of the algorithm, and articles that were taking fifteen minutes or more on the previous algorithm were now taking less than a minute, which I decided would be fast enough for the expected volume of articles.

A full analysis of the speed of the new topic labelling algorithm, and its role in the bigger picture of the summarisation process is outlined in the evaluation section.

Brief overview of candidate ranking algorithm

1. **Isolate the nouns from the Wikipedia articles of every label**
2. **Use nouns from step one as a corpus in a Term Frequency-Inverse**

Document Frequency (TF-IDF) model

3. **Isolate the nouns from the original article**
4. **For each label:**
    - Calculate the sum of performing TF-IDF on each individual noun from the label's article. The TF-IDF algorithm is explained in full detail in Section 3.1.3.
    - Find the total number of nouns that are present in the original article that are also present in the candidate article. This is known as the crossover.
    - Normalise the crossover by dividing it by the total number of nouns in the candidate article and the original article.
    - Multiply the crossover by the sum of TF-IDF operations found earlier to give a final total.
5. **Order the candidates by score**, highest to lowest.

5.5.3 Clustering

For the project, I developed my own clustering algorithm, based on the metric of Term Frequency-Inverse Document Frequency.

The intention of the make up of my clustering algorithm is that the following two items must be true for a pair of articles to be clustered together:

- **Noun similarity**: The content of the articles themselves needs to be similar. To do this I'll again look for similarity between 'significant words' (nouns) in the articles.
- **Timestamp similarity**: When a piece of news is published, if other outlets are going to cover it, they will publish as soon as possible. Therefore, the difference between timestamps should be used as a factor in deciding if articles are genuinely about the same topic or not, or just about similar topics.

Initialisation

1. **Create a TF-IDF corpus** from the content of every article to be clustered.
2. For every article **perform TF-IDF on each noun**.

3. For every article **generate a 'timestamp' score**. The timestamp is represented by the number of milliseconds since midnight on January 1, 1970. This is the result of the Java Date class' getTime() method.
4. **Create clusters**, with each article initially assigned its own cluster.
5. **Generate a similarity matrix** with initial values for each cell determined by the 'Finding Similarities' algorithm detailed below. The purpose of this is to store the 'distance' between a cluster and all other clusters, and to quickly and easily find similarities.

The article, along with a list of TF-IDF scores for each noun, and the timestamp score, are kept in a class known as an ArticleVector. These ArticleVector instances form the cluster items in the algorithms.

Finding Similarities

The principle behind this step of the algorithm is to compare two clusters and generate a score to keep in the similarity matrix. To do this, each ArticleVector from one cluster is compared with every ArticleVector from the second cluster. These similarity scores are then added together to form a final similarity score between the clusters. So, for example, if there was a cluster with three articles, and a second cluster with four articles, then the final similarity score would be the sum of the twelve possible combinations. The algorithm I used for determining the similarity between two ArticleVector instances is detailed below:

```
1         List<TfIdfScores> firstNounScores =
             ↪ firstArticleVector.getNounScores();
2         List<TfIdfScores> secondNounScores =
             ↪ secondArticleVector.getNounScores();
3
4         double similarityRunningTotal = 0;
5         double differenceRunningTotal = 0;
6
7         double averageWords = (firstNounScores.size() +
             ↪ secondNounScores.size()) / 2;
8         double totalSimilarWords = 0;
9
10        for (TfIdfScores word : firstNounScores) {
11            String noun = word.getLabel();
12            if (secondNounScores.hasWord(noun)) {
```

```
13                  TfIdfScores score =
                        ↪ secondNounScores.scoreForWord(noun);
14                  double number = word.getTfIdf() -
                        ↪ score.getTfIdf();
15                  similarityRunningTotal += Math.pow(number,
                        ↪ 2);
16                  totalSimilarWords++;
17              } else {
18                  differenceRunningTotal += word.getTfIdf();
19              }
20          }
21          for (TfIdfScores word : secondNounScores) {
22              String noun = word.getLabel();
23              if ( ! firstNounScores.hasWord(noun)) {
24                  differenceRunningTotal += word.getTfIdf();
25              }
26          }
27
28          double multiplicationFactor = totalSimilarWords /
                ↪ averageWords;
29
30          similarityRunningTotal =
                ↪ Math.pow(similarityRunningTotal, 0.5) *
                ↪ multiplicationFactor;
31
32          return similarityRunningTotal /
                ↪ differenceRunningTotal;
```

Listing 5.10: *Finding similarities between two ArticleVectors*

1. **Initialise variables**: similarityRunningTotal and differenceRunningTotal will form the backbone of the final calculation, and act almost like a running total. averageWords stores (as the name suggests) the average number of nouns in the two ArticleVector items - this will help to normalise the calculation somewhat.
2. **For each word in the first ArticleVector:**
    - If the word is in the second ArticleVector, increment a counter for the total similar words. Add the square of the difference between the word's TF-IDF scores in each item to the similarityRunningTotal.
    - If the word is not in the second ArticleVector, add its TF-IDF score from

the first ArticleVector to the differenceRunningTotal.

3. **For each word in the second ArticleVector that's not in the first**, add the TF-IDF score to the differenceRunningTotal.
4. **A multiplicationFactor is calculated** to be the number of similar words divided by the average words.
5. **The final similarityRunningTotal** is calculated to be the square root of the similarityRunningTotal multiplied by the multiplicationFactor calculated earlier.
6. **Calculate the final similarity** by dividing the similarityRunningTotal by the differenceRunningTotal.

Factoring in the timestamp

Once the similarity from listing 5.10 has been calculated, the timestamp is then factored in.

The theory behind this is simple.

If the similarity is above a certain threshold then the timestamp difference between the two ArticleVector's is considered and the final similarity score adjusted accordingly.

If not, then the final similarity is reset to -1, as it is determined that the ArticleVector are not about the same topic, and therefore there is no need for a timestamp check.

When the timestamp is considered, the absolute difference between the two timestamps is calculated, and the similarity score is divided by the result. The logic behind this is that it means that a small difference between the timestamps results in the similarity score increasing significantly, whilst the converse is also true.

The timestamps are also subjected to a multiplication factor before the absolute difference between them is calculated. The theory behind this is that this will mean that similarity scores are penalised more heavily, rather than on a standard linear scale.

Combining clusters

Once the similarity scores have been calculated for all clusters and the full similarity matrix has been generated, the clusters can be combined.

To do this, the highest similarity score from the matrix is found. If it is above

a threshold (all thresholds are tabled below) then the two clusters are combined. Once the clusters have been combined, the entire process is started again, with a new similarity matrix being generated.

If the similarity score was below a threshold, then the clusters are not combined. At this point the clustering process is declared to be finished, as no clusters have changed in this iteration.

Parameter Selection

I tried different values for the three parameters in the clustering algorithm, each with varying degrees of success. I conducted a survey that involved the clusters produced by testing multiple clustering parameters. The results of this survey are documented in the evaluation Section. The final parameters chosen are listed below:

Parameter	Final Value
Similarity score threshold to bring in the timestamp	0.0005
Timestamp multiplication factor	36000
Threshold for combining clusters	0.005

Table 5.5: *A table showing the final parameters chosen for the clustering algorithm*

5.6 Summarisation

For the summarisation of articles, I initially intended to use LexRank for extractive summarisation, followed by Semantic Graph summarisation for abstractive summarisation. However, it became clear that abstractive summarisation would not be time efficient, which is detailed in Section 5.6.3.

5.6.1 Extractive Summarisation

Modified LexRank algorithm

The core aspects of the extractive summarisation follow the LexRank algorithm that was investigated in Section 3.2.1.

1. **Create a Graph**, with each sentence in each article corresponding to a node in the graph. Information stored in the node includes:
    - The original sentence

- The position of the sentence (as a decimal) in the original article
- The absolute position of the sentence in the original article
- An identifier for the node
- A list of related nodes, initialised as an empty list
- The source of the original article

2. **Apply cosine similarities** as in the original LexRank algorithm. I initially attempted this with just nouns, but that approach is not particularly suitable for summarisation, as verbs become important in identifying similarities between two individual sentences. As a result, I decided to use the stoplists file that was provided by Mallet for topic modelling. Words are filtered out before the LexRank cosine similarity function is applied.
3. **Filter the graph**, removing connections between nodes that has similarity values that are below a selected threshold.
4. **Page Rank is applied**, using the equation given in Section 5.6.3.
5. **Redundant nodes are filtered out**. This is done by iterating through the nodes returned by the page rank algorithm. For each node, if there are nodes which share more than 75% of the same words (minus the words in the stoplist file provided by Mallet), it is marked as a related node. When a node is marked as a related node, it is added to the related nodes list in the original node, and the related node is not considered for the final summary.

At this stage the summary is considered to be ready to go through quotation and coreference resolution, in order to ensure the summary is as readable as possible. These processes are detailed below.

Quotation resolution

In order for the summary to be as readable as possible, it's important that quotations are complete. An incomplete quotation could mean that the reader is reading a quotation from someone that can easily be taken out of context, or that just doesn't make much sense.

In the quotation resolution phase of the summarisation, each node is analysed in order. There are four types of quotation that the algorithm looks for:

1. A quote is started in the sentence, but is not completed.
2. A quote is ended in the sentence, but wasn't started in the same sentence.

3. There are no quotation marks in the sentence, but it is the middle sentence in a longer quotation.
4. A quote is both started and ended in the same sentence.

There is no issue with a quote outlined as in (4). However, with the first three, resolution needs to take place. With resolution, one of two things will happen:

- **If the rest of the quote is in the summary**
  The nodes comprising the quotation will be combined, ensuring that in the final summary they are kept together.
- **If the rest of the quote is not in the summary**
  Depending on whether the current node starts or ends the quote, adjacent sentences from the original are added to the summary (this is done using the absoluteSentencePosition field on the node). The adjacent sentences are combined with the node to ensure that they are kept together in the final summary.

Coreference resolution

Pronouns in a sentence can also cause readability issues, as the sentence could be out of context, meaning it would not be clear what the pronoun is referring to.

Initially I aimed to fix these issues using the coreference resolution part of the Stanford CoreNLP library, which is outlined in Section 5.4.2. However, there were two issues with this. On test runs, it would take nearly a minute to produce the coreference resolution for an article. In a summary that has several original articles this would become prohibitively expensive. In addition to this, it was also clear that whilst Stanford's coreference resolution might be competition-winning, it certainly isn't perfect. In a climate such as journalism, any incorrect coreference resolution could alter the meaning of a sentence significantly. If the article is about a sensitive subject, it could create some liability issues.

As a result, I decided that the best way to approach coreference resolution would be a similar manner to quotation resolution. Using the Apache OpenNLP library to identify pronouns, I would then add the previous sentence to the summary (if it wasn't already present), and combine the two nodes to ensure that they were kept together in the final summary, thus preserving readability.

Ordering

Another key aspect to the summarisation's readability is that sentences perform the same task that they were meant to in the original article. For example, a sentence that was meant to set the scene near the start of an article, should also appear near

the start of the summary.

This is achieved by ordering the sentences according to the sentencePosition field on the Node object.

5.6.2 Summary Evaluation Resource

In order to provide an easy way to track progress on how my summarisation algorithm was doing, I created a simple web page to test it out.

On this web page I could input the URL to an article from *Reuters*, an article from *The Independent* and an article from *The Telegraph*. This would then submit these three articles to the server, which would obtain the article text from the websites. It would then summarise it, and send it back to the web page.

This was particularly useful as it would mean that I could fully test summarisation without having to go through the process of modelling, labelling and clustering the articles.

A screenshot of the screen can be seen in Figure 5.2.

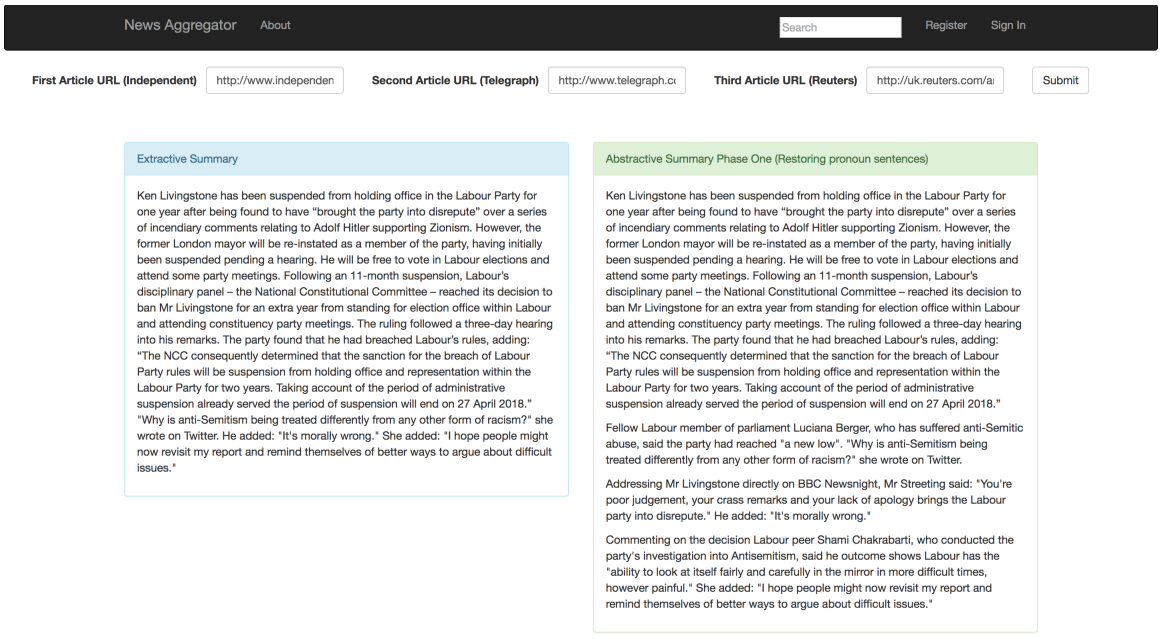


Figure 5.2: *A screenshot of the Summary Evaluation Resource. On the left is the extractive summary, and on the right is the extractive summary after the coreferences and quotations have been resolved as in Section 5.6.1*

5.6.3 An attempt at Abstractive Summarisation

I made an initial attempt at abstractive summarisation, before reluctantly abandoning it on the grounds of infeasibility (further details on why are provided on why this was the case are given further on in this Section).

Brief overview of Abstractive Summarisation algorithm

The following algorithm presents in more detail the ideas presented in the Semantic Graph Section in Section 3.2.2.

1. **Rich Semantic Graph creation**
   In this process, the first step is to perform an analysis on each of the original articles using the Stanford CoreNLP library. I then perform coreference resolution on the articles, replacing relevant nodes in the summary. I do this because it reduces the length of the summary significantly. The Rich Semantic Graph is then created from the Rich Semantic Subgraphs, which are created according to the procedure below.
2. **Rich Semantic SubGraph creation**
   In this step, the original summary is converted into several rich semantic subgraphs. First, the Stanford analyses computed in the first step are used to find a collection of 'Relation Triples'. Relation Triples are groups of subject, verb, object patterns. Each of these triples will be given its own subgraph. Each word in the triple is fed into the WordNet library (Section 5.4.2 to find all the possible meanings of the word.
   Once every possible meaning has been found, the cartesian product of every possible meaning is generated, and assigned a score based off the WordNet usage statistics. The graph is created from this cartesian product, with each meaning of an individual word constituting a node.
3. **Rich Semantic Graph reduction**
   The graph is reduced by applying heuristics to combine sentences. For example, if two consecutive sentences both have the same subject, they can be combined into one sentence, with a conjunction for the object.
4. **Natural Language Generation**
   The final stage is to pass the reduced semantic graph into a natural language generator, in order to generate the final summary sentences.

Feasibility issues

The major issue with this algorithm again centred around time. The designer of the algorithm had only presented results on simple sentence structures. As a result, when it came to the more complex structures and sentence lengths that inevitably appear in news articles, the algorithm began to struggle.

The major issue came in the creation of rich semantic subgraphs. Take the following relation triple for example.

Computing is hard

According to WordNet, there are two meanings to the word 'computing', eleven for the word 'is' and twelve for the word 'hard'. All told, that alone would result in 264 different combinations presented in the cartesian product. Given this would be just one out of potentially dozens of relation triples in an article, the amount of time to generate a single summary would become prohibitive. As a result, I decided that it would make more sense to focus on making the extractive summarisation as readable as possible instead.

5.7 Restlet

For my server I used the Restlet framework for Java. Restlet is a powerful open source framework that allows users to create vastly scalable Rest APIs and back end servers. There are three key classes in Restlet that I used.

- **ServerResource** is a wrapper class, that provides the class with both a request object, and a response object. A server method can be created by using the annotation @Get or @Post as appropriate above a method.
- **Application** is the base class for Restlet. Here, in an overrided method createInboundRoot I set the routes for the API to follow, and which ServerResource subclass to use when a request comes in to a specific api address.
- **TaskService** is a service that allows the asynchronous scheduling and running of 'tasks'. I used this to run the server jobs required for fetching, labelling, clustering and summarisation.

Documentation on the Web API I created is provided in Appendix B and for the server jobs in Section 5.8.

5.8 Server Tasks

The following jobs are back end server tasks using the Restlet framework's TaskService (Section 5.7). I created them with the purpose of fetching, labelling, clustering and summarising articles. They also perform other important tasks such as sending digests to subscribed users.

5.8.1 Scheduling of tasks

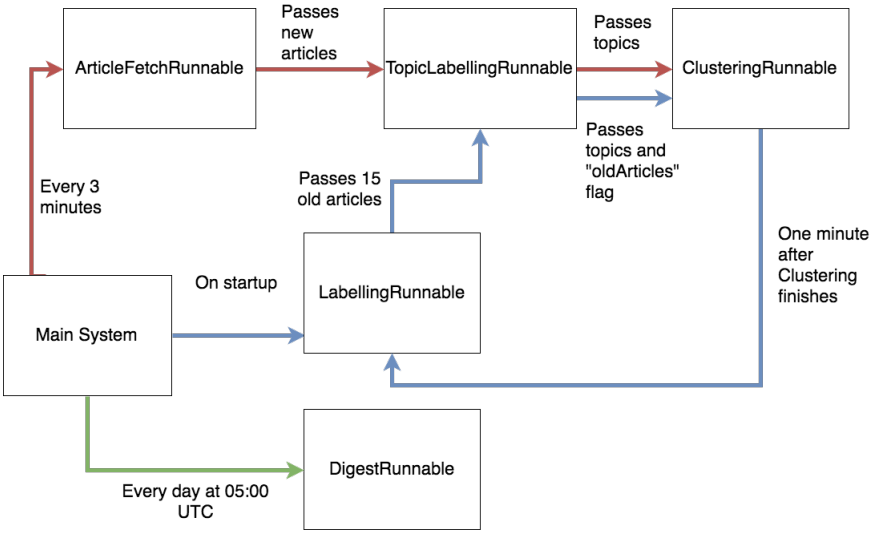


Figure 5.3: *A diagram showing the scheduling of tasks using the TaskService and the flow of information in the server tasks*

In order to make sure no new articles from external outlets are missed, the ArticleFetchRunnable is run every three minutes. The ArticleFetchRunnable, on completion, fires the TopicLabellingRunnable with the new articles as a parameter (this only happens if there is at least one new article). Once the TopicLabellingRunnable has completed, it in turn fires the ClusteringRunnable.

The principle behind this is to create a cascading effect. With the ArticleFetchRunnable being called so frequently, the TopicLabellingRunnable should rarely have to deal with a large batch of articles in a single run (the largest recorded so far is 22 articles in one Runnable). This cascading effect also means that the ClusteringRunnable wouldn't have to deal with a massive batch of topics at any one time either. This will all minimise the delay between an article first being published on an external outlet, and then reaching the NewSumm website.

With the LabellingRunnable, once the articles that have been picked up have gone through the whole process, and clustering has finished, the ClusteringRunnable then schedules the LabellingRunnable to start again a minute later. This means that only a maximum of one thread in the thread pool is dedicated to summarising old articles, and as many threads as possible can be dedicated to bringing news to the website as soon as possible after publication.

The DigestRunnable is fired once a day, at five in the morning in the UTC time zone.

Figure 5.3 shows how tasks are scheduled in the back end, and how the information flows between the separate server tasks.

5.8.2 ArticleFetchRunnable

The ArticleFetchRunnable runs through every source and fetches the latest articles. Each article is then checked against the database to see if it exists. If it does, then it is removed from the list of latest articles, whereas it is kept if it isn't present in the database.

Each article has the 'isLabelled' flag set to false, and is then saved to the database.

5.8.3 TopicLabellingRunnable

The concept of the TopicLabellingRunnable is to model and label a set of articles. It has a constructor that takes as a parameter a list of articles.

The Runnable models and labels each article, changing the 'isLabelled' flag on the articles to true, and the 'NeedsClustering' flag on the labels to true, before saving all to the database.

5.8.4 ClusteringRunnable

The ClusteringRunnable is responsible for both clustering topics, and for producing the summaries.

Initially, each label is clustered. Then, for each cluster, a 'power set', consisting of all possible combinations of articles in the cluster (of size greater than or equal to one) is created. For each combination of articles an extractive summary is produced (this allows for the customisation of survey sources).

Once the summaries have been generated, the clusters have been saved. They are added to the list of the labels' clusters in the database. In addition to this, for each label, a call to the Wikipedia API is made to find the list of categories that the label belongs to. The clusters are added to these labels as well.

In a time-saving measure, a list of clusters that have been summarised during the course of the runnable is kept. This means that if a cluster is created that has already been created for a different label, it doesn't need to be summarised again.

The 'NeedsClustering' on the labels are also set to false.

5.8.5 LabellingRunnable

The LabellingRunnable's main function is to find articles that, for various reasons, could have 'slipped through the cracks'. It is fired at regular intervals, and takes fifteen articles from the database that have been marked as needing labelling, and calls the TopicLabellingRunnable with these articles.

5.8.6 DigestRunnable

The DigestRunnable is run each day. It goes through each user in the database, creating digests for those who have requested them. It then saves these digests in the database, thus creating an ID that is used to create a link to each. It then sends an email to each of the users with a digest, each email equipped with the generated link to view their digest.

5.9 Front End

For the implementation of the front end, I created a website and an iOS app.

The iPad version of the iOS app is designed to mirror the interface of the desktop website, whilst the iPhone version mirrors the interface of the mobile website.

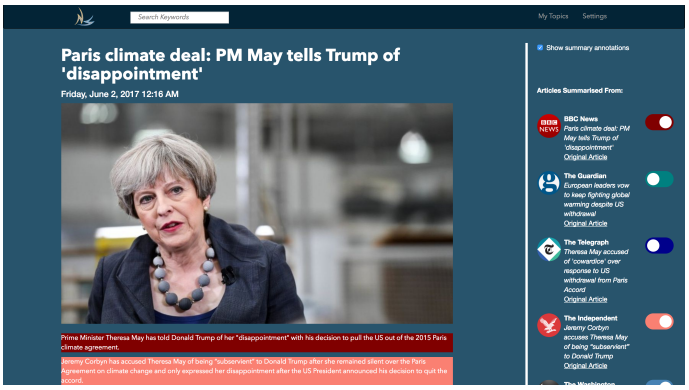
There are a few key differences from the initial design. These are:

- **The Home Page** - In the initial design, it was stated that a signed in user's home page would be their topic dashboard screen. This is now changed, so that they still receive the original home page, but the 'Top Stories' Section is now tailored to their specific subscriptions. This means that the large search bar still has major prominence, which, in my opinion, emphasises the search

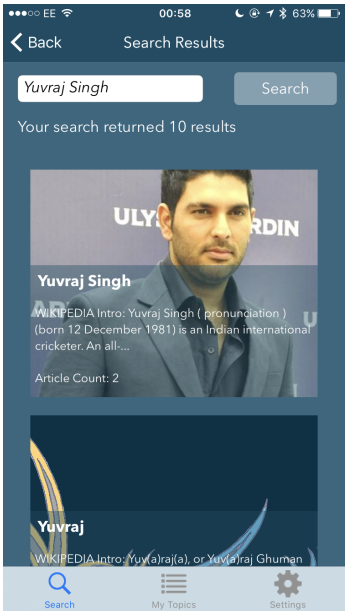
capabilities of NewSumm, whilst still providing the user with their articles at a glance.

- **Profiles** - Users no longer have a profile picture, and in fact only an email address is stored to link to their accounts, and so there is no longer a profile settings page.
- **Article Viewer** - This screen now allows a greater amount of customisation on a user's part. Each article that the summary is created from is now accompanied by a switch. Turning this switch off will remove the article from the summary. The same happens in the opposite direction.
- **Summary Annotations** - Users can now see where the individual sentences of their summary are coming from. There is now a checkbox marked 'Show Summary Annotations' in the corner, and when selected the text from the summary is highlighted according to its source (the source colours are shown on the corresponding switches on the right of the screen). Clicking on a highlighted piece of text will reveal a popover showing the source, along with the original sentence. If a sentence has some related sentences that aren't in the summary, the text won't be highlighted, and the popover will show the original, as well as all related sentences.

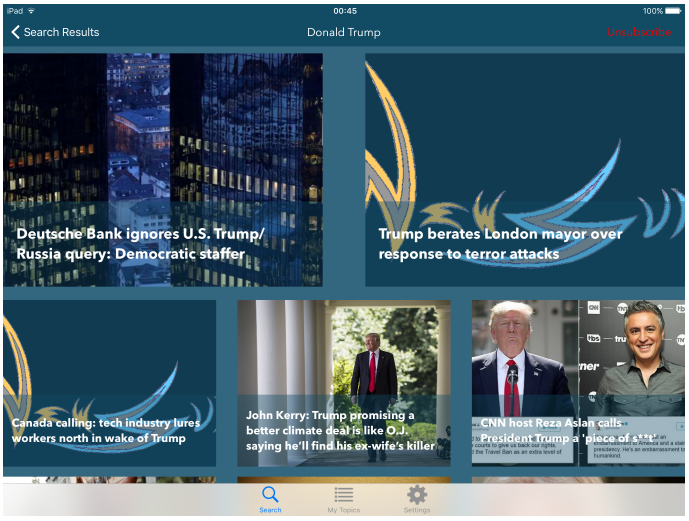
Screenshots from the website and apps can be seen in Figure 5.4, and further details are provided in the user guide in Appendix C.



(a) *The article viewer, with summary annotations on, on the desktop website*



(b) *Search Results on the iPhone application.*



(c) *The topic viewer on the iPad application.*

Figure 5.4: *Screenshots from the website and iOS Applications.*

6 Evaluation

6.1 Machine Learning and Summarisation

6.2 Summary Analysis

6.3 User Interface Evaluation

7 Optimisation

7.1 Speed Optimisations

7.1.1 Topic Modelling

7.1.2 Topic Labelling

7.1.3 Clustering

7.2 Memory Optimisations

8 Conclusion

9 Future Work

9.1 Foreign Languages

9.2 Further Optimisations

9.3 Other Apps

References

[1] *Apple*. www.apple.com.

[2] *Apple News*. www.apple.com/uk/news/.

[3] *BBC*. www.bbc.co.uk.

[4] *Cluster Analysis*. URL: https://en.wikipedia.org/wiki/Cluster_analysis (Retrieved Dec. 29, 2016).

[5] Gunes Erkan and Dragomir R Radev. 'LexRank: Graph-based Lexical Centrality as Salience in Text Summarization'. In: *Journal of Artificial Intelligence Research* (December 2004).

[6] *Flipboard*. www.flipboard.com.

[7] Pierre-Etienne Genest and Guy Lapalme. 'Framework for Abstractive Summarization using Text-to-Text Generation'. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics* (June 2011).

[8] Pierre-Etienne Genest and Guy Lapalme. 'Fully Abstractive Approach to Guided Summarization'. In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics* (July 2012).

[9] *Google*. www.google.com.

[10] *Google Forms*. www.forms.google.com.

[11] *Google News*. www.news.google.com.

[12] Karl Grieser et al. 'Using Ontological and Document Similarity to Estimate Museum Exhibit Relatedness'. In: *ACM Journal ACM Journal of Computing and Cultural Heritage* (2011).

[13] Vishal Gupta and Gurpreet Lehla. 'A Survey of Text Summarization Extractive Techniques'. In: *Journal of Emerging Technologies in Web Intelligence* (2010).

[14] *Hierarchical Clustering*. URL: https://en.wikipedia.org/wiki/Hierarchical_clustering (Retrieved Dec. 29, 2016).

[15] *How does Google News cluster stories?* URL: https://www.quora.com/How-does-Google-News-cluster-stories/answer/Bharath-Kumar-M?srid=Qord (Retrieved Jan. 8, 2017).

[16] Statistic Brain Research Institute. *Attention Span Statistics*. URL: http://www.statisticbrain.com/attention-span-statistics/.

[17] N. R. Kasture et al. 'A Survey on Methods of Abstractive Text Summarization'. In: *International Journal for Research in Emerging Science and Technology* (November 2014).

[18] Atif Khan and Naomie Salim. 'A review on abstractive summarization methods'. In: *Journal of Theoretical and Applied Information Technology* (January 2014).

[19] *Latent Semantic Analysis*. URL: https://en.wikipedia.org/wiki/Latent_semantic_analysis#Latent_semantic_indexing (Retrieved Dec. 22, 2016).

[20] Jey Han Lau et al. 'Automatic Labelling of Topic Models'. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics* (June 2011).

[21] *News API*. URL: www.newsapi.org (Retrieved Jan. 27, 2017).

[22] *Ofcom*. www.ofcom.org.uk.

[23] *Reuters Institute Digital News Report 2016*. Reuters Institute for the Study of Journalism, 2016.

[24] *Reuters Institute for the Study of Journalism*. www.reutersinstitute.politics.ox.ac.uk.

[25] *Statistic Brain Research Institute*. www.statisticbrain.com.

[26] *Summly*. www.summly.com.

[27] *Term Frequency - Inverse Document Frequency*. URL: https://en.wikipedia.org/wiki/Tf%E2%80%93idf#Example_of_tf.E2.80.93idf (Retrieved Dec. 29, 2016).

[28] *The Guardian*. www.theguardian.com.

[29] *The New York Times*. www.nytimes.com.

[30] *University of Canberra*. www.canberra.edu.au.

[31] *University of Oxford*. www.oxford.ac.uk.

[32] *Webhose*. URL: https://webhose.io (Retrieved Jan. 27, 2017).

[33] Harald Weinreich et al. 'Not Quite the Average: An Empirical Study of Web Use'. In: *ACM Transactions on the Web* (February 2008).

[34] *What is a good explanation of Latent Dirichlet Allocation?* URL: https://www.quora.com/What-is-a-good-explanation-of-Latent-Dirichlet-Allocation/answer/Edwin-Chen-1?srid=Qord (Retrieved Jan. 8, 2017).

[35] *What's the difference between Latent Semantic Indexing and Latent Dirichlet Allocation*. URL: https://www.quora.com/Whats-the-difference-between-Latent-Semantic-Indexing-LSI-and-Latent-Dirichlet-Allocation-LDA/answer/Joseph-Turian?srid=Qord (Retrieved Dec. 22, 2016).

[36] *Wikipedia*. www.wikipedia.org.

[37] *WWDC Apple Design Award Winners for 2014*. URL: http://www.macworld.com/article/2358481/wwdc-apple-design-awards-winners-for-2014.html (Retrieved Dec. 29, 2016).

[38] *Yahoo News Digest*. www.uk.mobile.yahoo.com/newsdigest/.

[39] *YouGov*. www.yougov.com.

Appendices

A Source Code and Documentation Repositories

The various aspects of the project are implemented on four different GitHub repositories:

The source code for the **back end and WebApp** can be found at:

https://github.com/kunalwagle/newsaggregator-backend

The source code for the **iOS application** can be found at:

https://github.com/kunalwagle/newsaggregator-ios

The python notebook that was used for the **evaluation and analysis** can be found at:

https://github.com/kunalwagle/newsaggregator-evaluation

The repository for the **report** can be found at:

https://github.com/kunalwagle/newsaggregator-reports

The report repository also includes all graphs and charts that were displayed at points in the report.

B API

The following constitutes the API that the iOS application and WebApp uses to make requests to the server.

All calls below should be prefixed with *http://kunalnewsaggregator.co.uk:8182*.

B.1 Searching for Topics

GET /api/wikipedia/{searchTerm}

Request Parameters:

- **searchTerm**: The keyword(s) to search the Wikipedia API for.

Search Wikipedia for a list of topics pertaining to the user's search request. Combine with the contents of the topic database to return the Wikipedia titles, first paragraphs, links to images and an article count for each topic in the database.

B.2 Getting a topic

GET /api/topic/{topic}/user/{userId}

Request Parameters:

- **topic**: The id of the topic to return the contents for.
- **userId**: (optional) The id of the logged in user, if applicable.

Returns the basic information (titles, ids and image links) of the articles within a topic in a wrapper with a list of settings for the topic. The settings are set to the user preferences. If none exist, or the user is not provided, the default topic settings are provided.

B.3 Settings

POST /api/settings

Request Parameters:

- **user**: The id of the user changing their settings.
- **digest**: True if the user is requesting digests for the topic. False otherwise.

- **sources**: A list of sources that the user wants to set as their default sources for the given topic
- **topicId**: (optional) The id of the topic to apply changes to. If not present, then apply changes to all of the user's subscriptions.

Given the request parameters in a multipart form, the server sets the user's settings for the topic (or for all their topics if a topicId is absent). Returns the updated User object.

B.4 Subscriptions

B.4.1 Subscribing and Unsubscribing

GET /api/user/subscribe/{user}/{topicId}
GET /api/user/unsubscribe/{user}/{topicId}

Request Parameters:

- **user**: The id of the user subscribing or unsubscribing to the topic.
- **topicId**: The id of the topic concerned.

Subscribes, or unsubscribes the user to/from the topic provided. Returns the topic object.

B.4.2 Getting a user's subscriptions

GET /api/user/subscriptions/{user}

Request Parameters:

- **user**: The id of the user.

Returns a User object, populated with a list of the user's subscriptions and the relevant settings for them.

B.5 Digests

GET /api/digest/{digestId}

Request Parameters:

- **digestId**: The id of the digest to fetch.

Returns a list of titles, ids and image links for the articles in the corresponding digest.

B.6 Getting an Article

GET /api/article/{articleId}/topic/{topicId}/user/{user}

Request Parameters:

- **articleId**: The id of the article.
- **topicId**: The id of the topic the article is in.
- **user**: (optional) The id of the user making the request (if logged in).

Returns the requested article in a wrapper with a list of default sources to display the summary for. The default sources are generated using the topicId and user parameters.

B.7 Home Page

GET /api/home/{userId}

Request Parameters:

- **userId**: (optional) The id of the user making the request (if logged in).

If the userId parameter is present, returns the ten latest articles from across the user's subscriptions. If the userId isn't passed, or if the user doesn't have subscriptions, returns the ten latest articles in the database.

C User Guide

Index