Efficiently Crawling, Collecting, and Condensing News Comments

by

Gobaan Raveendran

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2013

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Traditionally, public opinion and policy is decided by issuing surveys and performing censuses designed to measure what the public thinks about a certain topic. Within the past five years social networks such as Facebook and Twitter have gained traction for collection of public opinion about current events. Academic research on Facebook data proves difficult since the platform is generally closed. Twitter on the other hand restricts the conversation of its users making it difficult to extract large scale concepts from the microblogging infrastructure.

News comments provide a rich source of discourse from individuals who are passionate about an issue. Furthermore, due to the overhead of commenting, the population of commenters is necessarily biased towards individual who have either strong opinions of a topic or in depth knowledge of the given issue. Furthermore, their comments are often a collection of insight derived from reading multiple articles on any given topic. Unfortunately the commenting systems employed by news companies are not implemented by a single entity, and are often stored and generated using AJAX, which causes traditional crawlers to ignore them. To make matters worse they are often noisy; containing spam, poor grammar, and excessive typos. Furthermore, due to the anonymity of comment systems, conversations can often be derailed by malicious users or inherent biases in the commenters.

In this thesis we discuss the design and creation of a crawler designed to extract comments from domains across the internet. For practical purposes we create a semiautomatic parser generator and describe how our system attempts to employ user feedback to predict which remote procedure calls are used to load comments. By reducing comment systems into remote procedure calls, we simplify the internet into a much simpler space, where we can focus on the data, almost independently from its presentation. Thus we are able to quickly create high fidelity parsers to extract comments from a web page.

Once we have our system, we show the usefulness by attempting to extract meaningful opinions from the large collections we collect. Unfortunately doing so in real time is shown to foil traditional summarization systems, which are designed to handle dozens of well formed documents. In attempting to solve this problem we create a new algorithm, KLSum+, that outperforms all its competitors in efficiency while generally scoring well against the ROUGE SU4 metric. This algorithm factors in background models to boost accuracy, but performs over 50 times faster than alternatives. Furthermore, using the summaries we see that the data collected can provide useful insight into public opinion and even provide the key points of discourse.

Acknowledgements

I would like to acknowledge my advisor Charles Clarke for his mentorship and guidance in completing this thesis. I would also like to acknowledge Frauke Zeller and Abby Goodrum for their co-operation and discussions on the uses for our unique data set. Zach Weiner of SMBC-comics also deserves some thanks for giving me permission to use his comic within my thesis. Last but not least, I would like to thank the research and commercialization network GRAND NCE for their sponsorship of my research.

Dedication

This thesis is dedicated to my friends and family for always being interested in my rantings about news comments. I would also like to thank Andrew Russell and my dog Shiba for dealing with my one sided discussions about what to add to my thesis. Daniel Nicoara should also receive a special mention, as our constant competition is likely the only thing that kept me on course.

Table of Contents

List of Tables **viii**

List of Figures **ix**

List of Algorithms **x**

1 Introduction **1**

1.1 Motivation 1

1.2 Goal 3

1.3 Contributions 4

2 Related Work **6**

2.1 Crawlers 6

2.2 Parsers 8

2.3 Summarization 12

3 Implementation **17**

3.1 AJAX Simulation Framework 19

3.2 Parsing comments 26

3.3 Results 31

4 Diverse Opinion Summarization for Scalable Opinion Mining **32**

4.1 Problem Description 33
4.2 KLSum+ 35
4.2.1 Summary of Enhancements 35
4.2.2 Implementation 35
4.3 Baseline Methods 38
4.3.1 Iterative Random Summarization 38
4.3.2 SumBasic 38
4.3.3 MEAD Summarization 39
4.3.4 LexRank Summarization 41
4.3.5 Topic Model Based Summarization 41
4.3.6 Aspect Mining with Sentiment Methods 42
4.3.7 Other Methods 46
4.4 Experimental Setup 46
4.4.1 Data 46
4.4.2 Efficiency 49
4.4.3 ROUGE Scores 50
4.4.4 Example Summaries 53
4.5 Discussion 62

5 Conclusion **65**

Appendix **67**

A HTTP Request Header **68**

B Data Description **70**

References **72**

List of Tables

2.1 Readability's Regular Expressions 10

3.1 Sample Google Parameters 18
3.2 List of Parameters for Comment Fetching 25
3.3 Example Values for Parameters 25
3.4 Parser for HTML Comments 28
3.5 Parser for JSON Comments 30
3.6 Parser for Embedded HTML Object 31

4.1 List of Key Terms for discussing Summarization 34
4.2 Sentiments for every word sense of the word 'Smart' 43
4.3 Syntactic Patterns Predicting Key Noun Phrases 45
4.4 Document Count for Top 5 Websites 47
4.5 Document Count for Top 5 Topics 47
4.6 First 100 words for summaries with the query "SOPA" 55
4.7 Continued: First 100 words for summaries with the query "SOPA" 56
4.8 Top 25 terms from each of algorithm 57
4.9 Example Summaries from KLSum+ 59
4.10 Example Summaries from KLSum+ 60
4.11 Example Summaries from KLSum+ 61

List of Figures

1.1 Sample CNN.com Comment 2
1.2 "How Internet Fighting Works" by Zach Weiner 2

2.1 AJAX Model of the Web 8
2.2 Example News Article 9
2.3 Example HTML and associated DOM Tree 11
2.4 Graphical model representation of LDA 15

3.1 Overview of Crawler Architecture 17
3.2 Sample Result Page 19
3.3 How AJAX Works image by Kevin Liew 20
3.4 Graphical Representation of Least Upper Bound 27
3.5 CBC JSON Layout 29

4.1 Graph of Domains vs Number of Documents 48
4.2 Graph of Topics vs Number of Documents 48
4.3 Summarization Time versus Number of Sentences 50
4.4 DUC Scores 51
4.5 Cross Evaluated DUC Scores 53
4.6 Snapshot from gobaan.com:8000 64

List of Algorithms

3.1 Monitoring User Traffic 21
3.2 Dynamic programming solution to Longest Common Subsequence problem 24
4.1 Snippet Selection Algorithm 37
4.2 Sentiment Scoring Algorithm 45

1

Introduction

Huxley feared the truth would be drowned in a sea of irrelevance.

– Neil Postman, *Amusing Ourselves to Death*, Penguin Books

1.1 Motivation

Social media has reformed the way the world shares information. In the last few years, we have seen social media used to enhance presidential campaigns, monitor crime around the world, and even start global revolutions[27]. The news itself has become a giant mixture of voices that wish to be heard. The read-only days of print media, where articles are written by journalists and consumed by the masses, is over. In its place the web has created a fluid space of discourse, where every article can be commented on, discussed and analyzed. Readers can now converse, debate, and share related articles, with the ease of pouring a coffee. However, with this ease comes information overload.

In the past five years the number of blogs has increased five fold. Furthermore, online news has been shown to be preferred over offline news for most individuals[43]. The barrier of entry for creating and releasing news articles is much lower and hundreds of domains exist that rehash content or provide an editors specific spin on a topic. One of the core advantages to online news is the ability for the audience to create comments, such as the one presented in Figure 1.1, which enable users to discuss and highlight the key points of a story. Furthermore, commenters often debate using context derived from outside articles, and thus comments as a whole contain much more context than the original article.

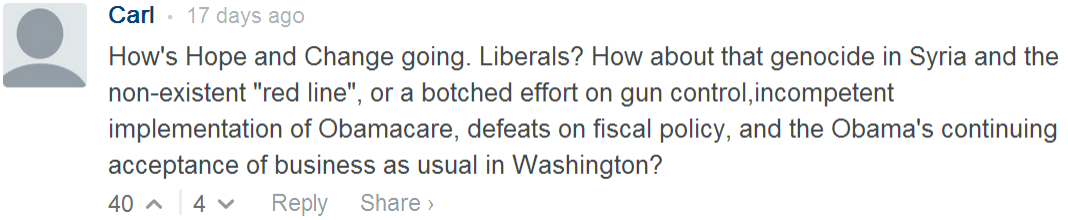


Figure 1.1: Sample comment from cnn.com article about Obamacare

By analyzing these comments, some journalist have gained valuable feedback that may help drive future articles and discussion on the site[26]. Websites such as Reddit have shown that news aggregation also benefits greatly from commenting, as commenters discuss both the strengths and flaws of the article, along with the long reaching implications of the issue being discussed. With controversial issues, the comments can often be extremely polar and become representative of distinct groups with varying interests in the topic at hand. Unfortunately, often it seems that the loudest voice is the most prominent, as highlighted by the SMBC[62] web comic below. Thus despite comments providing a rich set of opinions, the overall opinion can be drowned out by the noisiest individuals.

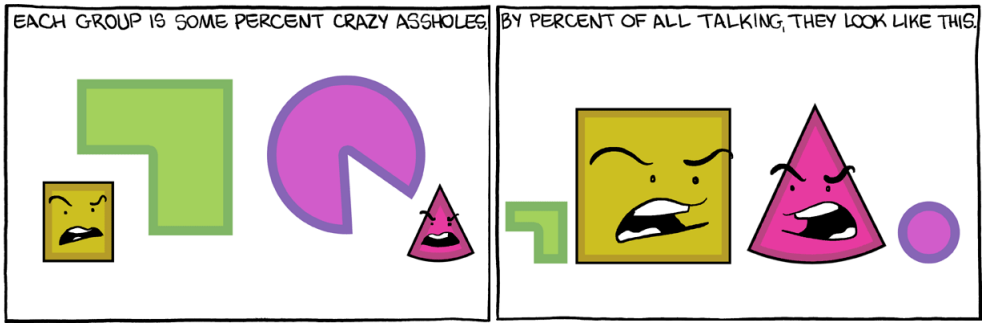


Figure 1.2: Excerpt from SMBC comic titled "How Internet Fighting Works" by Zach Weiner

Gathering these comments for analysis is foiled by various sites using different methods to track and store comments. One of Twitter's core uses is for commenting on news articles publicly and spreading opinion. The expressiveness of tweets however, is restricted by a character limit that can make analysis and clustering of tweets difficult. Furthermore, links on Twitter are often to outside domains, and thus ignored by summarization research because they are difficult to parse and interpret. The general academic community has recognized the value in tweets however, and experiments such as the TREC microblogging

task focus on attempting to classify and retrieve tweets based on keywords. Supervised data for the microblogging task requires filtering through hundreds of thousands of tweets, providing a fundamental challenge to the task.

1.2 Goal

The goal of this thesis is to describe the creation and uses of my open source custom comment crawler[1]. This crawler is uniquely designed such that it can quickly and efficiently crawl news articles and their comments. I accomplish this by creating a framework for writing specialized parsers that are able to extract content from a single domain. The proposed framework is designed to fetch only the data necessary for collecting comments, ignoring images, ads, videos, and other extraneous data located on various news domains. This is unique from the general approaches described later in the thesis that usually require rendering the entire page and predicting which components are needed.

We accomplish this by exploiting the implementation of Asynchronous JavaScript and XML (AJAX) as described in Chapter 3. By determining which AJAX calls are required to acquire the comments for a given article, I can figure out what specific parameters fetch the comments, and then directly guess the AJAX calls for future articles.

With this framework in place I then describe some of the tools created to aid in the creation of customized parsers. Once I collected the large set of data, I dug into understanding the information hidden within the data. However, the data collected was found to be very noisy, and difficult for traditional methods to summarize. In Chapter 4, I finish this thesis by discussing my algorithm, KLSum+, that was designed to efficiently summarize the opinions presented in noisy comment datasets.

[1] `https://github.com/Gobaan/NewsCrawler`

1.3 Contributions

My crawler provides the following contributions:

- The ability to crawl dozens of domains quickly and efficiently. Moreover, by minimizing the amount of data requested and naturally throttling the number of connections to a given domain, I avoid negatively affecting the target domain. This also allows me to use a single machine to crawl these domains saving on infrastructure costs.
- The ability to quickly be updated to domain changes or to add new domains. Previous solutions were found to be difficult to customize and require days of analysis and be relatively fragile to variation. By fetching the comments directly I disconnect myself from any dependence on layout code.
- The ability to be crowd sourced. The modular nature of the parsers means that a community can work together and add new parsers with virtually no interaction with the other parsers. This plugin structure allows the system to be distributed and decentralized.
- The ability to . Other solutions often attempt to predict what section of text belongs to the comments. This often means gathering extra unwanted data or incorrect sections. By fetching the comments directly I acquire raw structured data, which is designed to be processed by layout code later in the website rendering process.
- The expectation of failure. Given the amount of variety and the special care required to craft each parser, my crawler was designed with failure in mind. This means that I implement various logging and recovery techniques to quickly detect when failures occur rather than letting them silently disappear.

By analyzing my dataset I provide the following contributions:

- I create an open source baseline[2], called KLSum+, for summarization on multidocument sets. Specifically, much of the literature focuses on document sets with dozens to hundreds of documents. While my corpus can easily contain hundreds of thousands of microdocuments.

[2] `https://github.com/Gobaan/Summarization`

- I show the uniqueness of the data. The data itself is quite different from other document sets such as books and blogs. The data is much less structured and can contain much more variety, while not having the limited context of similar sets such as the Twitter dataset.
- I analyze efficiency of popular algorithms with a new dataset. Since much of the summarization literature focuses on a dataset with various differences, this work helps evaluate these algorithms in a different context.

2

Related Work

If I have seen further it is by standing on ye shoulders of Giants.

– Sir Isaac Newton, *Letter to Robert Hooke*

The problem of extracting comments that can be dynamically generated has received some attention in the past few years. Google for example has created a proposal [18] for webmasters that allows them to specify how to make their AJAX content searchable. However, this solution only works if webmasters follow the designed solution and requires creation of a custom crawler that understands these specialized rules.

Alternatively, large domains such as YouTube.com have created comment APIs that allow comments to be gathered from a given url via HTTP requests. However, this approach is not employed by many large news web domains, and unlikely to be implemented by smaller domains.

For this work I worked on to writing a crawler and parser that is able to navigate and extract comments from various news portals, for example CNN.com which is presented in Figure 2.2. In order to simplify this task I used metasearch techniques via Google, thus I was able to ignore the much larger task of finding relevant articles.

2.1 Crawlers

When implementing a search engine, crawlers are designed to explore a domain, finding all the links connected to a given web page. When it comes to comment extraction this involves finding all the comment pages attached to a given article.

The field of web crawling is full of diverse tasks that researchers have attempted to address. In 2010 Olston and Najork [45] performed a comprehensive survey of the state of the art in web crawling. Traditionally research has focused on politely extracting content from across the web, while maintaining either freshness or convergence.

A crawler is considered polite if it has a fetching policy that follows the web servers rules while avoiding overloading the target domains by making excessive requests within a finite period of time. A simple policy to encourage politeness is to navigate the web and add URLs to a priority queue. In this queue, urls are sorted by the time to next fetch:

$$b + K * t_{fetch} \tag{2.1}$$

where b is the base time to wait before fetching urls, K is an arbitrary constant, usually 10, and t_{fetch} is the time it took to fetch the last resource we fetched from the given domain. This policy helps ensure that slower domains, with less resources to handle requests, receive less traffic. This priority can be extended to factor in domains that update more frequently, or are more relevant, for example by using Google PageRank [46].

On the other hand, the ideal crawler should be able guarantee freshness and thus ensure that as a page updates the crawled version of the page updates with it. In order to maintain freshness, visited urls are regularly revisited, while maintaining the politeness policy mentioned above.

Unfortunately, when attempting to cover a single topic a general purpose crawler can quickly become encumbered by extraneous links throughout the web. A topical crawler attempts to rerank links based on how relevant they are to a given topic. Given a topic crawler we can say the crawler converges if two separate executions, with a disparate seed set of urls, covers the same set of articles.

These mechanisms and measures focus on urls and links to determine what to extract. However, with many sites comments are setup to load dynamically using AJAX. When AJAX is used no hyperlink exists, so a crawler may be unable to find content. Furthermore, crawlers rely on caching websites and parsing the documents offline, decoupling the data extraction from the data fetching. However with dynamic content, links can vary and change before a page is revisited from the breadth first queue. Traditional deduplication based on url no longer works, because an AJAX website will update content without changing the url.

Research around AJAX crawling focuses on simulating a browser that can load each page and then simulating user behaviour [13, 42, 47]. Specifically, these approaches require creating an event model that represent JavaScript events. This model assigns a unique

node to each comment page, and maps each JavaScript event as a transition between states. In this approach the model of the web is extended, and is no longer a series of nodes connected via hyperlinks. Instead the model becomes more like Figure 2.1, where each node (representing a page) contains a series of substates and transitions between states.

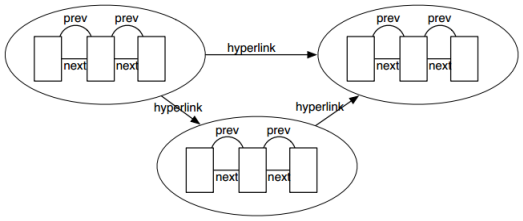


Figure 2.1: AJAX Model of the Web

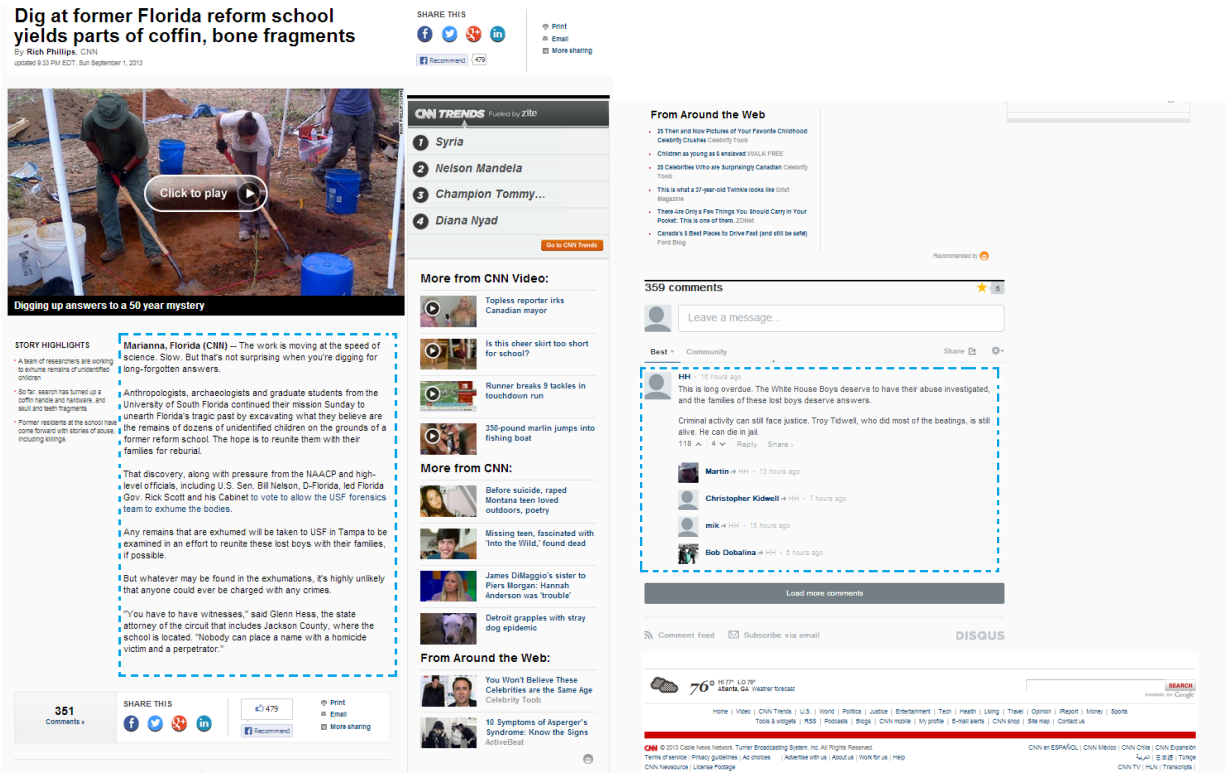
Under this approach the traditional priority queue for crawlers is extended to include triggers which navigate between states within a given web page. Choosing which triggers to apply depends on the application, with some approaches targeting clickable elements [13] and others preferring to populate search bars using term lists in order to attempt full coverage of a given domain [37, 42].

This kind of system can be considered a full page crawl, and can get expensive quickly due to state explosion from events that generate infinite unique states, or from requiring hundreds of calls to the server to emulate all the events. Rules can be created to only follow certain events, however determining which events to follow is difficult. The easiest resolution is to implement caching to AJAX function calls with equal parameters, but this will still inevitably lead to excess fetches.

It should also be noted that Olston's[45] work briefly discusses the lack of current literature focus on both crawling scripts and vertical crawling. A vertical crawling architecture is one that focuses on creating high fidelity crawling systems for a few domains, rather than a general purpose solution that uses no domain specific knowledge.

2.2 Parsers

For my work I am only interested in article text and comment data from each of the articles I find. However, news domains often contain erroneous data, such as ads, summaries, and related articles. For example, Figure 2.2 shows an article, where the dotted boxes



(a) Article Section (b) Comment Section

Figure 2.2: Example news article. The interesting text is highlighted with the dotted box.

highlight the only text not considered superfluous for document summarization. Thus, the second task involves extracting and classifying the data such that I have the comments and article data properly separated from this extra metadata. For news articles, this is a well recognized problem and various solutions have been proposed. Two of the most well adopted solutions include heuristic classifiers [9, 30] and text categorization via machine learning [14, 59, 60, 61]

With heuristic classifiers a set of rules are created that help determine if the section in question is contains relevant information. The project Readability[66] is an example of an associative rule miner designed to extract article text. This project uses regular expressions that search HTML elements and attempt to predict which ids are important.

To explain this approach we first give a quick primer on HTML. HTML consists of a series of nested tags that browsers read and interpret. This structure can be described as an acyclic tree often called the Document Object Model (DOM). Figure 2.3b is an example

Classification	Regular Expression
Unlikely Candidates	combx|comment|disqus|foot|header|menu| nav|rss|shoutbox|sidebar|sponsor
Probable Candidates	and|article|body|column|main
Article Text	article|body|content|entry|hentry|page| pagination|post|text
Not Article Text	combx|comment|contact|footer|footnote| link|media|meta|promo|related|foot| scroll|shoutbox|sponsor|tags|widget
Video	http://(www.)?(youtube|vimeo).com

Table 2.1: List of regular expressions used in Readability to classify content

of a DOM tree corresponding to the HTML in Listing 2.3a.

Each tag may have any number of optional attributes. These attributes are normally used by the browser to determine how to format the data (such as text or images) attached to the tag. Other attributes such as class and id help uniquely identify a single tag or a set of tags. Normally this is used by AJAX or CSS formatting, however parsers can also use these formatting tips to gain some insight into what sections of an article are important.

Heuristic parsers construct this DOM Tree for each website they visit. They then use heuristics to determine what sections of the DOM tree are important. In the case of Readability the regular expressions in Table 2.1 are used to guess at candidates or determine which sections should be extracted.

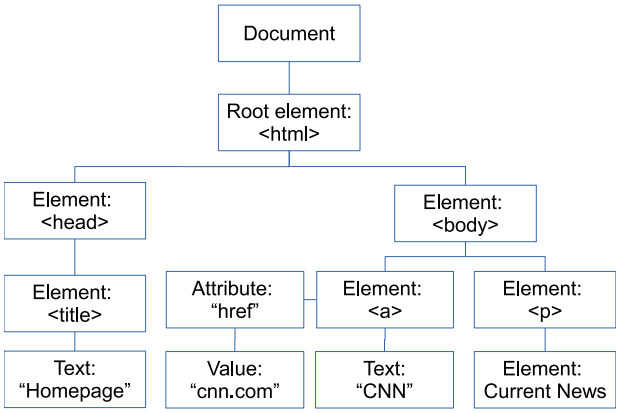
These regular expressions were handcrafted in the hope that they are general enough for most articles on the internet. However, this approach is tied to English sources, and often these rules will not be valid among multiple domains. Furthermore these perform well in general, but are likely to fail for edge cases.

Alternatively, I can create specialized wrappers which are templates designed to extract text from a special domain. This approach however is costly, since each domain needs a custom template to be designed. These templates can be difficult to maintain, as they are fragile and may break whenever the layout of a page changes. In order to decouple the extraction from the creation of templates, support vector machines are often employed in order to automatically build templates for domains. Junfeng Wang et al. [60] describes how to extract news articles using this approach.

This work relies on content features and spatial features. The content features employ the layout of the DOM tree and formatting to determine what sections are important. For

```
<html>
  <head> <title> Homepage </title> </head>
  <body>
  <a href="cnn.com"> CNN </a>
  <p>Current News </p>
  </body>
</html>
```

(a) Sample HTML



(b) Associated Document Object Model

Figure 2.3: Example HTML and associated DOM Tree

example, titles often use a larger font than the rest of the text. The spatial features on the other hand require rendering the page and then tracking where each element in the DOM tree appears. The relative position can then be used to help the system learn where titles and text are often placed and to separate ads from content.

This approach has proven successful for extracting articles. However it suffers from three key shortcomings. First this approach can be slow, since rendering a page requires loading all the ads, videos, and miscellaneous content for a site. Second, many of the feature values that help classify article text causes false negatives with user comments. For example, user comments will often occur at the bottom of a page, possibly after ads or related articles. Thus relative position may dismiss comments as unnecessary. The third problem comes from the asynchronous nature of comments. Large sites often do not load all user comments during page load in order to save bandwidth. This means comments are on a separate page or require clicks to be tracked. Determining which icon or which clickable resource fetches comments, versus ads or other articles, can prove difficult to accomplish using spatial or content features.

Diffbot[59] is a startup centered around the concept of 'a visual learning bot that allows users to create APIs for any site'. The tool is designed to extract content from across the web, and although the implementation is a trade secret, they often discuss the use of machine learning and visual cues to determine how a website is laid out. This system also allows customers to help by specifying rules when components fail to extract in cases where the robot fails. Despite Diffbot's great set of features, the comment extraction component does not work as of the writing of this thesis. This serves to exemplify the difficulty in extracting user comments.

2.3 Summarization

The task of summarization is an open field of research that has acquired much attention in the last decade. The goals of a summarization system may vary greatly depending on the target audience. News summarization in particular may focus on extracting events, tracking memes, or determining relevant details. On the other hand, email summarization and review summarization focus on extracting out themes for a given thread or determining opinions about different aspects of a product. In recent years, microblog summarization over Twitter has also gained some traction. This work focuses on extracting meaningful content from large collections of documents that are all under 140 characters, many of which are hashtags, links, or other metasequences.

The most traditional approach to summarization depends on scoring terms and sentences, in an attempt to find important passages within the article. Term importance based summarization attempts to determine the importance of each n-gram within the collection. Using these n-grams, sentences are then selected based on the overall score of the n-grams within them. Generally, term importance is decided using a mixture of TF-IDF scores [25, 55, 64], sentiment scores [5, 23], and temporal scores [8, 41].

Some canonical examples of summarization using these scores include SumBasic [40] and KLSum [21]. SumBasic scores terms based on their likelihood of occurring within the corpus, iteratively selecting sentences that are highly probable but different from the currently selected sentences. These scores are generated from simple frequency counters, ranking terms by how often they occur in the corpus. KLSum extends this concept by attempting to generate a summary whose term-wise K-L Divergence mirrors that of the initial comment collection. This extension directly penalizes any summary that adds extraneous words, or any summary that misses frequent words. In order to find this solution, sentences are selected greedily based on which sentences cause the summary to most mirror the word distribution of the target documents.

Temporal summarization focuses on the extraction of key events from text by analyzing breakpoints in public opinion[8]. This approach works over real time data and has worked well over Twitter as tweets naturally spike in popularity around key events. One way of accomplishing this task relies on creating a Hidden Markov Model (HMM) trained to recognize breakpoints in opinion. In this HMM all tweets that occur during a given time window are considered a single observed state. By feeding tweets into this HMM it can be used to determine how the distribution of words diverges from one state to the next. Using the Viterbi algorithm[49] the events are segmented and summaries can be generated that present words that spiked during a given time frame.

Graph based summarization on the other hand focuses on creating relationships between sentences within the corpus and then isolating sentences that are especially important. Two well known algorithms in this field are LexRank [15] and TextRank [39]. LexRank creates sentence by sentence adjacency matrix and uses cosine similarity between sentences in TF-IDF space to measure how related two sentences are. Then by applying PageRank, the system generates a score for the centrality of all the sentences within the collection. Summaries are generated by selecting the top highest scoring sentences.

On the other hand TextRank which was developed at the same time creates the same sentence by sentence graph but defines similarity of two sentences as

$$Similarity(S_i, S_j) = \frac{|\{w_k | w_k \in S_i \& w_k \in S_j\}|}{log(|S_i|) + log(|S_j|)} \tag{2.2}$$

It is mentioned that this similarity score leads to a highly connected graph and thus the resulting graph must be weighted. For both LexRank and TextRank, if diversity is preferred then documents are preclustered before summarizing, however the choice of clustering algorithms is up to the implementation.

The two dominating clustering based summarization approaches are centroid based summarization and topic model based summarization. Centroid based summarization is based on clustering documents then choosing sentences that have terms that are statistically important to the clusters' centroids. MEAD is a widely cited and publicly available summarization toolkit that uses CIDR based clustering to perform centroid based summarization [51]. Topic models on the other hand, rely on factoring a term-doc matrix into a term-topic and topic-term matrix. The term-topic matrix then allows me to decide which terms are important for each of the separate topics and sentences can be selected to describe each topic. This approach often uses Latent Dirichlet Allocation(LDA) to factor the term-doc matrix [1].

LDA is a generative probabilistic model of a corpus where we assume all words and documents are generated by mixtures over latent topics as visualized in 2.4. In a generative probabilistic model we make the simplifying assumption that all documents are generated from a random process. This random process generates sentences by selecting words from a topic based on some statistical distribution. Documents themselves are a mixture of topics, and thus every word is drawn from a set of topics. We can infer the original distribution from from analyzing a corpus. In this model w represents the known distribution of words that we observe in our corpus. The words belong to N topics and attempt to estimate θ and ϕ. ϕ is the latent model that determines what words are drawn from a topic, and theta determines what topics are drawn for each document. α and β are priors over the documents that we can assign based on domain specific knowledge of our corpus. The literature discusses estimating ϕ and θ using variational expectation maximization or Gibb's Sampling.

Query based summarization[58] extends LDA for cases where we need to solve an information need, expressed by a query. In this model, each time a word is sampled, we flip a coin which decides if we should draw a word from the query distribution or from the document specific distribution. The goal of incorporating this query information is to make the summaries more closely related to the query. Furthermore, queries may have multiple topics, and thus multiple summaries may be generated, which may highlight different aspects of the query.

Once the topics are extracted, sentences can be selected using a similar approach to the centroid based summarizer, choosing sentences that either represent one topic well

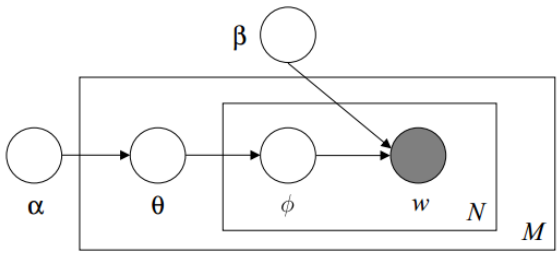


Figure 2.4: Graphical model representation of LDA. The boxes are "plates" representing replicates. The outer plate represents documents, while the inner plate represents the repeated choice of topics and words within a document.

or represent multiple topics. Unfortunately, on a large scale the parameter estimation required to execute PLSA and LDA can become a bottleneck due to the time required to have the algorithms converge.

Natural language research has started using sentiment analysis to perform opinion extraction[4, 23, 24]. Sentiment based summarization systems work by attempting to determine words used to express opinion within a given corpus. This is based on the assumption that opinion words are often tied to the most important aspects of a story or product. By using natural language processing and syntactic patterns sentiment systems are able to achieve some success in classifying terms as expressing an opinion. The sentiment score of a word i.e. how positive or negative, is usually decided either via machine learning or by sentiment dictionaries such as SentiWordNet[3, 24].

The machine learning systems are built by bootstrapping a classifier, which users initial human judgment and then constant feedback from humans to refine a classifier until it is able to accurately predict sentiment scores. These systems can be enhanced by using synonym sets and other natural language resources to create word nets and propagate sentiment throughout a corpus [5]. Once a sentiment scoring mechanism is created, buckets are created for positive and negative sentiment and individual summaries are created for each bucket. These systems usually focus on selecting either extremely positive, extremely negative, or completely neutral sentences. These systems may also be built to focus on noun sequences and the sentiments associated with them.

Evaluation of summarization systems comes in two flavors, extrinsic or intrinsic. In extrinsic evaluation we decide on a second task and evaluate the system based on this task. For example, one possible task is attempting to determine if a new document is relevant to a collection based on the summary[11]. Another alternative is creating a search system

that compares the rank order of documents given a query, before and after summarization. The ideal summary should preserve the rank order since only information that is repeated or irrelevant to the core content of the article is dismissed.

Intrinsic evaluation on the other hand relies on evaluating summaries against a gold standard. The DUC 2007 task[7] was one approach to automated summarization that took this approach. In this task, judges were tasked with reading a set of up to twenty five documents, and generating 100 - 250 word summaries of the collections. Automated summaries were then evaluated against these manual summaries using ROUGE-N and ROUGE-LCS scores[35]. Another group of work by Inouye and Kalita[25] discuss generation of human readable summaries of thousands of tweets. Their work relies on clustering to present judges tweets, that the judges then use to create summaries. They then evaluated ten systems using these gold standards and determined that simple term frequency based systems performed slightly better than alternative systems over tweets.

Overall the field of summary evaluation is a rich in research over the last four years. The National Institute of Standards and Technology runs a conference every year where one track is dedicated to the Automatically Evaluating Summaries of Peers[44]. In this task contestants attempt to create unsupervised metrics for linguistic quality and topic coverage that correlate well with human generated ground truth. As of the writing of this thesis no clear winner has surfaced however some systems show promise for opinionated and news data[28, 32].

3

Implementation

A good idea is about ten percent and implementation and hard work, and luck is 90 percent.

– Guy Kawasaki via Twitter

The first task in extracting news comments was the creation of a vertical crawling framework, designed to quickly create specialized crawlers for news domains around the internet. My crawler is composed of three main components. These components and interactions are described in Figure 3.1.

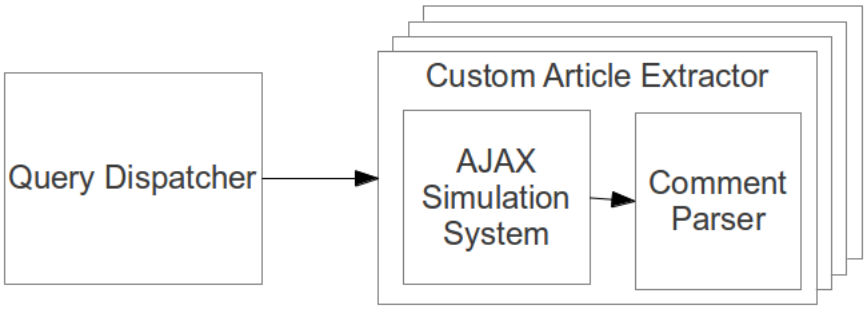


Figure 3.1: Overview of Crawler Architecture. Query dispatcher fetches article URLs and forwards them to customized parsers for further processing.

The first and simplest component, called the Query Dispatcher, is designed to find relevant articles from the internet, given a query. This is accomplished by using metasearch techniques. Metasearch implies forwarding my search requests to Google and parsing the

Parameter	Description	Example Value
query	Relevant terms	US Election
tbs	Date restriction	cdr:1,cd_min:10/1/2012,cd_max=10/31/2012
site	Domain restriction	www.cnn.com
start	Result page selection	0
num	Results per page	100
complete	Disable Google Instant	0

Table 3.1: Parameters used to fetch US Election Articles from CNN during October, 2012

search results. Unfortunately, Google's API lacks parameters for filtering based on a date range. Thus, I open a single connection to Google.com and simultaneously search each domain for articles related to a user provided query. For my work I implemented the ability to search for all articles that fall within a given year, for example 2012, or alternatively around a given day, for example a month of articles before and after November, 6, 2012. Below is a sample query that requests up to 100 articles between 01-10-2012 and 31-10-2012 for www.cnn.com about the topic "US Election".

```
https://www.google.com/search?q=US+election+site%3Awww.cnn.com
&num=100&tbs=cdr%3A1%2Ccd_min%3A10%2F1%2F2012%2Ccd_max%3A10%2F
31%2F2012&start=0&complete=0
```

This is equivalent to a request to www.google.com/search with the parameter values in Table 3.1. By executing this query I get a search page, such as that in Figure 3.2. Once the results are fetched, I may simply parse the HREFs, which specify link destinations, to determine the article URLs. Since the search page contains various other links, I restrict on links that formatted properly. I do this by restricting on HREFs found within subtrees of the form:

```
<li class=g><a class='l' href="..."></a></li>
```

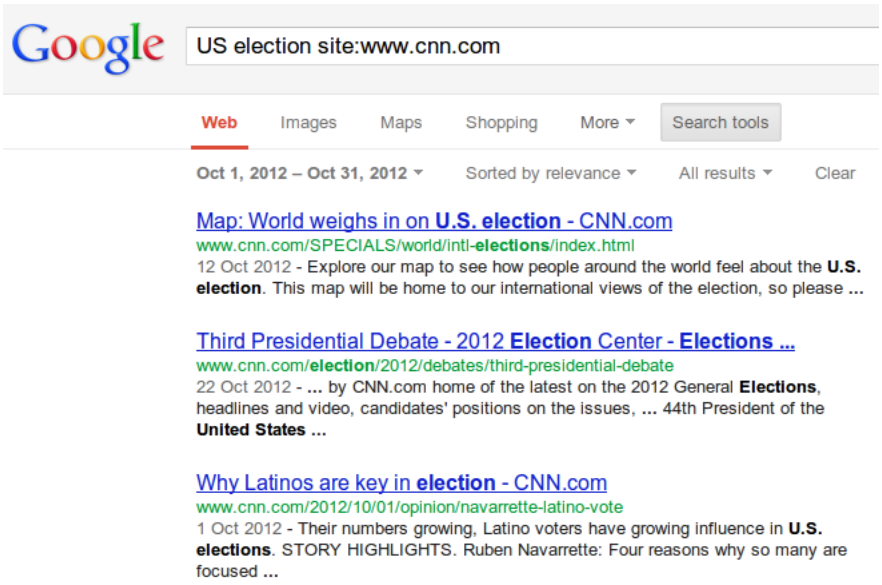


Figure 3.2: Sample Result Page

3.1 AJAX Simulation Framework

Once the relevant URLs have been fetched the next step is to extract the article and comment text from the results. I solve this problem by implementing a framework for content extraction. Given a URL from a given domain, the goal of a customized parser should be to extract every page of comments. In order to accomplish this two fundamental tasks need to be completed: finding all the comment pages, then extracting all the comments while eliminating unnecessary metadata.

Since comments are likely to be fetched from the web server and positioned by AJAX, link crawling generally fails. Thus, various crawlers often fall back to loading the page, executing all the JavaScript, and simulating user clicks. However, browsers implement AJAX requests by performing traditional HTTP requests to servers. Furthermore, during preliminary work we found that all surveyed websites used HTTP get requests. Thus, the necessary parameters for the server to understand the remote procedure call are embedded in the request url. This process is visualized in the image by Kevin Liew shown in Figure 3.3[17].

Thus, if I am able to visit a url, and determine how to generate the HTTP request that fetches the comments, I am able to fetch the comments directly. In order to accomplish this

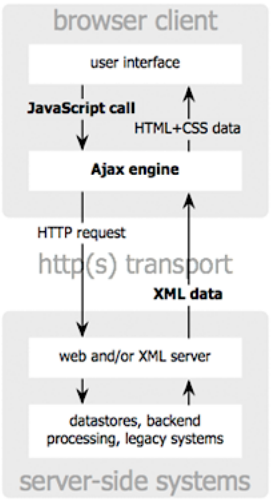


Figure 3.3: How AJAX Works image by Kevin Liew

I create a semi-supervised parser generator that attempts to determine what the important requests are, based on some simple user feedback. I start by attaching a monitor to a browser (Mozilla Firefox in my case), and monitoring all traffic opened by this browser. I then request that the user navigates to an article and provide the system with at least three comments from the target website. Next I attempt to determine the most efficient way to extract the comments, which varies based on how comments are presented by the target domain.

In order to instrument the browser I use a library designed for browser automation called Selenium [33]. Using Selenium I prompt the user to navigate to their desired domain, and fetch two comments from each of the second and third page of comments. Selenium enables the ability to capture network traffic, specifically the HTML Response messages from each URL fetched. For completeness I provide an example HTTP Response messages in Appendix A. For my work it is only necessary to understand that the URLs for all HTTP requests made are contained within the URL parameter of these headers. Thus, I extract the URLs and filter out only those that contain text, and filter once more for URLS that contain the comments I provided to the system earlier. The pseudocode for this is shown in Algorithm 3.1. Listing 3.1 shows an example of the data I request for one article on the news website arstechnica.com.

Once the URLs are fetched, the next goal is to determine how to transform the starting url into the comment url. I found that a few simple heuristics could be used to estimate

Algorithm 3.1 Pseudocode for monitoring user traffic and extracting AJAX calls containing comments.

```
function GetTargetedUrls(Fields[1..n])
    browser = Selenium.FirefoxBrowser()          ▷ Start an instrumented session
    targetURLs = empty-map
    for all field in fields do
        targetText = User.request(field)     ▷ Request text to search for from the User
        headers = browser.networkTraffic()          ▷ Fetch a list of http requests made
        for all header in headers do
            url = header["url"]
            if ContainsText(url) then:  ▷ Dismiss URLs that link to images, videos, etc.
                data = curl.fetch(url)
                if data.contains(targetText) then
                    targetURLs[field] = url
                end if
            end if
        end for
    end for
    return targetURLs
end function
```

```
http://arstechnica.com/apple/2012/10/the-ghost-of-jobs-apples
-challenge-to -decide-what-would-steve-do/?comments=1&start=40

http://arstechnica.com/apple/2012/10/the-ghost-of-jobs-apples
-challenge-to-decide-what-would-steve-do/?comments=1&start=80
```

Listing 3.1: Two sample comment urls

```
["http", "://", "arstechnica", ".", "com", "/", "apple",
 "/", "2012", "/", "10", "/", "the", "-", "ghost", "-",
 "of", "-", "jobs", "-", "apples", "-", "challenge", "-",
 "to", "-", "decide", "-", "what", "-", "would", "-",
 "steve", "-", "do", "/?", "comments", "=", "1", "&",
 "start", "=", "40"]
```

Listing 3.2: Token stream created from the first URL in Listing 3.1

which values in the url are parameters, and where these parameters come from. In order to explain this process I will provide example URLs, and explain how I dealt with issues I discovered during implementation.

I start off by requesting two comment pages from the same article. For example two comment URLs from arstechnica.com as shown in Listing 3.1. I then tokenize the URLs by splitting them on non-alphanumeric sequences creating a token sequence like the presented in Listing 3.2.

The next task is splitting this list into components such that the parameters are isolated and easily compared. This entails finding the values that change between the two token sequences. This problem is similar to the well known diff algorithm where I take the diff of token sequences rather than two files. In order to accomplish this I use solution to the longest common subsequence problem shown in Algorithm 3.2.

This solution uses dynamic programming based on two cases: First, if both sequences contain the i^{th} term I copy it into the final solution. Second, if both sequences do not contain the i^{th} term I take the larger of the two subsequences created by removing the current term from each list and recursing down both paths.

Given this subsequence I can group tokens in the original two streams into two groups, base url or parameter, depending on their existence in the common subsequence. Specifi-

```
{
    "sentences": {
        "Author": "Jacqui Cheng",
        "Pg2 Comment 1": "oogle wanted was Google branding and Latitude
            integration and you would have had the same awesome Android
            Google Maps experience as",
        "Pg2 Comment 2": "gle is especially good at that, while Maps appears
             to be especially bad. But I don't think different map data would
             help much with that",
        "Pg3 Comment 1": "happen anymore scares me about the future of the
            comp",
        "Title": "he ghost of Jobs: Apple",
    },
    "urls": {
        "Author": [
            "http://cdn.api.twitter.com/1/users/show.json?screen_name=
                eJacqui&callback=twttr.setFollowersCount",
            "http://arstechnica.com/apple/2012/10/the-ghost-of-jobs-apples-
                challenge-to-decide-what-would-steve-do/"
        ],
        "Pg2 Comment 1": [
            "http://arstechnica.com/apple/2012/10/the-ghost-of-jobs-apples-
                challenge-to-decide-what-would-steve-do/?comments=1&start=40"
        ],
        "Pg2 Comment 2": [
            "http://arstechnica.com/apple/2012/10/the-ghost-of-jobs-apples-
                challenge-to-decide-what-would-steve-do/?comments=1&start=40"
        ],
        "Pg3 Comment 1": [
            "http://arstechnica.com/apple/2012/10/the-ghost-of-jobs-apples-
                challenge-to-decide-what-would-steve-do/?comments=1&start=80"
        ],
        "Title": [
            "http://arstechnica.com/apple/2012/10/the-ghost-of-jobs-apples-
                challenge-to-decide-what-would-steve-do/"
        ],
        "starting_url": "http://arstechnica.com/apple/2012/10/the-ghost-of-
            jobs-apples-challenge-to-decide-what-would-steve-do/"
    }
}
```


Listing 3.3: Output of URL Extraction Phase

Algorithm 3.2 Dynamic programming solution to Longest Common Subsequence problem

```
function LCS(X[1..m], Y[1..n])
    C = array(0..m, 0..n)                          ▷ Initialize known sequences to 0
    for i from 1..m do
        for j from 1..n do
            if X[i] == Y[j] then                   ▷ Include matching characters
                C[i, j] = C[i-1, j-1] + 1
            else                                   ▷ Decide which token to remove
                C[i, j] = max(C[i, j-1], C[i-1, j])
            end if
        end for
    end for
    return C
end function
```

```
["http://arstechnica.com/apple/2011/10/the-ghost-of-jobs
-apples-challenge-to-decide-what-would-steve-do/?comments=1&
start=", ("40", "80")]
```

Listing 3.4: Output of running the diff algorithm on the two urls in Listing 3.1

cally, any token in the longest common subsequence is considered part of the base url and any variation is considered a parameter. The output of this grouping, on the example, is shown in Listing 3.4.

My next task is to discover how parameter values are likely determined. Table 3.2 below summarizes some of the basic parameter classifications and how their values are extracted. For conciseness I construct fake urls using the given parameters in my examples. These example values are shown in Table 3.3.

I now explain each of these parameters in finer detail. The three parameters, URL, Encoded URL, and Article ID, are used to determine which article to fetch the comments from. The first two parameters are generally easy to detect by separating the url path from the article url, and then encoding it if necessary. Once this is done I can replace all instances of the parameter with placeholders. I do this before running the LCS algorithm as these components often contain a mix of alphanumeric and non-alphanumeric terms that foil tokenization. An advantage to this approach is the simplification of the output from the LCS algorithm and a reduction in possible errors.

Parameter Type	Description	Extraction
Base URL	Initial URL	URL visited by crawler
URL	Unique slice of the url	Extract path from original url
Encoded URL	Safely encoded url	Encode URL using standard
Article ID	ID of this article	Use regex on URL or article body
Comment ID	ID of first comment to fetch	Use regex on article/comment response
Page number	Page/Comment number to fetch	Increment counter before fetching
Arbitrary	Useless Parameters, or call ID's	Set to 0 or empty

Table 3.2: Most common parameters for comment fetching

Parameter Type	Example
Base URL	http://www.fake.com/post/this-is-my-article–14131
URL	/post/this-is-my-article–14131
Encoded URL	%2fpost%2fthis-is-my-article–14131
Article ID	14131
Comment ID	914131
Page number	5
Arbitrary	0

Table 3.3: Example Values for Parameters

The article id on the other hand is a unique identifier sometimes assigned to an article. By crafting regular expressions targeting these ids I can ease the extraction. These regular expressions are autogenerated by searching urls and the original HTML source for lines containing the ids. I then present the user with a set of possible regular expressions that will extract the necessary id. The comment ID is similar to article ID, however usually marks the first comment in a set to search. Therefore, the comment ID usually changes with each page that needs to be extracted, thus the regular expression needs to be reapplied before each page is fetched.

The last two parameters are the page number and arbitrary ids. Most websites use some form of counter to decide which comment to get, this counter can either represent a page number or the index of the first comment. In order to determine what the counter represents I compare the values from the second and third page of comments, if the values between two separate articles match up then I guess that this value comes from counter. To determine how to increment the counter I can subtract the two values. For example, if I have **`comment=40`** and **`comment=80`**, I know to increment my counter by 40 between each page. Sometimes arbitrary id's exist to help track function calls, or provide unique names for callbacks. I detect these by setting their values to 0 and checking if the same comments are fetched.

Once I have my comment crawling systems complete, I am able to automatically convert article urls to comment urls. When generating URLs I insert them into a priority queue that allows me to maintain my politeness policy. However, since we only need to request a few kilobytes of text data per a comment page, we fetch multiple pages in one connection. This has the added benefit of lowering the number of connections I make to the server, and thus lowering the CPU overhead on the target's end.

3.2 Parsing comments

Once the comment url is determined the next task is extracting the comments. In my work I found that most sites belong to one of three different systems: comments embedded in HTML, comments in JSON objects, or comments embedded in strings within a larger request.

The simplest comment system involves embedding the comments in HTML, where each page request fetches an entirely new url containing the comments. This is traditionally implemented via an HREF attribute, and links to an HTML page that needs to be processed to extract the comments. However, many domains also have AJAX fetch simple html

responses containing the comments, and these comments are loaded within a frame. In order to parse comments in this form I use a library called BeautifulSoup[53]. I choose to use this library because most websites do not follow HTML standards. For example, a study done in 2008 by Opera, found that only 4.13% of websites visited passed WC3 validation [63]. Browsers, and libraries such as BeautifulSoup, are designed to recover from common mistakes that users make and create a deterministic parse of a given website. Furthermore, as a consequence to the malformed nature of HTML, alternatives such as regular expressions can prove both difficult to create, and fragile to website changes.

However when parsing hundreds of pages per a domain BeautifulSoup can be slow, due to creating an in-memory DOM Tree representing all the nodes within an HTML website. Furthermore, my system was designed to be relatively simple for newcomers to generate new parsers and the overhead of learning BeautifulSoup foiled this goal. Thus I create a simplified wrapper around BeautifulSoup and boosted efficiency by restricting which sections of the DOM Tree are created.

By using example comments provided by users, I can restrict BeautifulSoup to create nodes only for the subtree rooted at the least upper bound of all comment nodes. The least upper bound of two nodes in a tree is the youngest ancestor of the two nodes as shown in Figure 3.4.

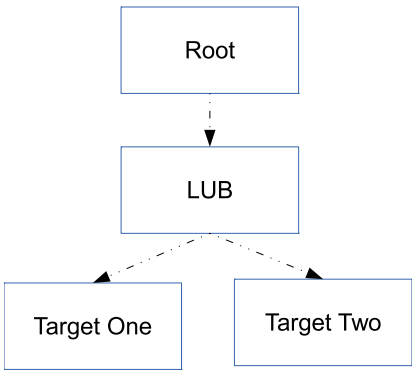


Figure 3.4: Graphical Representation of Least Upper Bound. Dotted lines represent an arbitrary amount of intermediate nodes

I determine the least upper bound by finding the path to both comment examples, then navigating these paths and return the last node that is shared by both paths. BeautifulSoup also provides the ability to find all nodes that have certain attributes or tags. For example it is possible to extract all nodes of the form:

```
1  def discussions_chicagotribune_rename(self, url, site):
2    title = self.regex.search(url).group("title")
3    return Template(self.template.substitute({"title":title}))
4
5  def discussions_chicagotribune_postprocess(self, comment_list):
6    for comment in comment_list:
7      text = comment.text
8      comment.text = text[text.find("\n \t \n"):]
9    return comment_list
10
11 discussions_chicagotribune = HtmlParser(
12   regex = re.compile(".*?chicagotribune.com/.*/(?P<title>.*?),.*"),
13   rename = discussions_chicagotribune_rename,
14   postprocess = discussions_chicagotribune_postprocess,
15   site = "www.chicagotribune.com",
16   template = Template("http://discussions.chicagotribune.com
17     /20/chinews/$title/10"),
18   strainer = SoupStrainer("div", {u"id": u"comment-list"}),
19   targets = SoupStrainer("div", {u"class": u"comment"}),
20 )
```

Example 3.4: Parser for HTML Comments. Lines 1-3 specify how to convert the base template into a comment url, Lines 5-9 remove extra characters, Line 12 is used to extract parameters, Line 18 creates the DOM Tree for the LUB of all comments, Line 19 fetches all comments

```
<p class="comment.*">
```

Thus, once I have the tree for the least upper bound extracted, my system automatically generates the necessary BeautifulSoup code to extract just the comments. Unfortunately, some domains insert extra data such as authorship or dates at the beginning or end of a comment. Thus I allow users to specify Python code that can trim away the necessary characters. I present an example of this type of parser in Example 3.4.

The second case is similar, however instead the server returns a JSON object. In this case, the url is usually generated via a JavaScript function call. The JSON object is formatted as needed by the calling function, and thus it often contains extra metadata.

```
{
  "Envelopes": [
    {
      "Payload": {
        "Items": [
          {
            "Body": "Comment One"
          },
          {
            "Body": "Comment Two"
          }
        ]
      }
    }
  ]
}
```

Figure 3.5: Example CBC comment JSON Layout

,

For my purposes, I can determine the JSONPath that represents all comments within the object and extract those directly.

JSONPath is a notation for selecting components from a JSON object that match a given pattern, much like regular expressions are used over text data. JSONPath is built on navigating a JSON object by specifying which elements to match at each level of a JSON object. It has three tools I require for navigation: keyword, integer index, and a catch-all character, *. Keyword navigation allows me to navigate JSON dictionaries by selecting the element with a given key, e.g. 'url'. Index navigation selects elements of a list with a given index, and the * character selects all elements at a given level of a hierarchy.

An example JSONPath for CBC is '$.Envelopes[1].Payload.Items.*.Body'. Here I navigate the JSON object in Listing 3.5 and return the two comments, while removing any extra data, which I do not show in this example.

I can determine the JSONPath by finding the JSONPath of two comments on the same article. I then split the path on the '.' token and replace any differences with the catch all operator, *. The parser in Example 3.5 extracts comments from articles by the Washington Post.

Finally, sometimes the target comments are part of a larger object or webpage, for example the comments may be embedded in a larger JavaScript page. In this case I have to isolate both which lines contains the necessary comment, and how to extract the

```
1  def washingtonpost_rename(self, url, site):
2    url = urllib.quote(self.regex.search(url).group("url"))
3    return Template(self.template.substitute({"url":url}))
4
5  washingtonpost = JSONParser(
6    regex = re.compile("(?P<url>.*?washingtonpost.com.*?.html).*"),
7    paths = ["$.entries.*.object.content"],
8    rename = washingtonpost_rename,
9      site = "washingtonpost.com",
10   template = Template("http://echoapi.washingtonpost.com/v1/
11   search?callback=jsonp1326063163544&q=((childrenof%3A+$url
12   +source%3Awashpost.com+++))+itemsPerPage%3A+100+sortOrder%3A
13   +reverseChronological+safeHTML%3Aaggressive+children
14   %3A+2+++&appkey=prod.washpost.com"),
15 )
```

Example 3.5: Parser for JSON comments. Generally the same as HTML Parser, however in Line 7 I track the JSON Paths used to store comments.

comment. In order to determine which lines contain the comments, I can use the comments provided by users to preprocess the document and remove any unnecessary lines. Once I have isolated the lines containing the comments, the next stage is determining how to parse them. This often means finding a subset of the line that is understandable by either the JSON parser or HTML Parser. I can then craft a regular expression that extracts the necessary comment component of the line for other articles. This reduces the problem to one of the first two cases, and I apply whichever solution is appropriate. An example of this type of parser is shown in Example 3.6.

Using all these heuristics I generate a group of potential parsers and present them to the user. The user then inspects the output of each parser and chooses which parameter values select the targeted data.

```
1  def cbsnews_preprocess(self, comment_list):
2     return [line for comment in comment_list for line in comment
3      if line.startswith("</script> <sp")]
4
5  cbsnews = HtmlParser(
6    site = "www.cbsnews.com",
7    strainer = SoupStrainer("div", {u"id": u"commentWrapper"}),
8    preprocess = cbsnews_preprocess,
9    targets = SoupStrainer("dd", {u"id": re.compile(u"body.*")}),
10 )
```

Example 3.6: Parser for Embedded HTML Object, In Lines 1-3 I strip out any extraneous lines before parsing out comments

3.3 Results

Using the vertical crawling system described above I created crawlers to extract comments on 43 different domains. Creation of each parser took approximately 10 minutes. The average parser is implemented in 20 lines of Python code, approximately half of which is autogenerated. A full list of sites domains targeted by my system is located in Appendix B.

My system was originally conceived in order to extract comments from news websites discussing the controversial 'Northern Gateway Pipeline'[56]. As such, my seed set of parsers targeted news domains from Canada and the west coast. In order to get breadth, one week was spent monitoring `www.reddit.com/r/news` and writing parsers for the domains that appeared frequently. It should also be noted that this system was designed generically such that it could easily be extended to other domains, such as review mining, but this was not tested as part of this work.

4

Diverse Opinion Summarization for Scalable Opinion Mining

> *With a click of the "Post Comment" button, Netizens can quickly bring down the level of dialogue ... In an era in which making noise is essential to standing out and breaking through the clutter, naughty will, by definition, win out over nice.*
>
> – Willow Bay, *Bile in the Blogosphere*, Huffington Post

When attempting to isolate the opinions expressed about a given topic, be it a political revolution or a new product, many come up short. A survey[19] on marketers found that approximately 29% report having "little or no consumer data". Furthermore, only 35% collect social media data, and those that do find it difficult to measure their return on investment. This lack of utilization occurs despite tools such as Twitter, generating 175 million tweets and Facebook generating over 100 terabytes per day[19].

One issue with Twitter is the generally conversational nature of posts, with only 3.6% of tweets discussing the news[48]. Facebook data on the other hand is generally considered sensitive by users and thus difficult to obtain. Using my system I was able to collect a large collection of news comments about any given topic. Unfortunately, once I harvest this discourse I still must determine how to understand this large collections of comments.

In this chapter I perform a case study attempting to use my novel collection of data to evaluate various opinion summarization algorithms. This both serves to present the usefulness of the data my tool collects and attempts to address an interesting and unsolved problem. I start by performing a brief survey on the field of multidocument summarization.

Next I introduce my algorithm KLSum+. This algorithm starts with the general skeleton of faster algorithms such as KLSum and SumBasic. However, by borrowing from topic modeling literature, I am able to factor in background models. The background model allows me to remove fragility introduced by stop word lists, while better encoding term-wise information entropy. Next, by adding length and redundancy penalties, I am able to quickly extract passages that are more meaningful and understandable. Finally, I evaluate progress by benchmarking the system against various other systems, which are well known for their ability to summarize multiple documents. This allows me to show that my results are consistent with these alternatives while both executing faster and producing more coherent summaries.

4.1 Problem Description

News comments generally have no limit on their length. Along with the free-form nature of comments, the general lack of skill or motivation to write consistently strong comments, can cause the quality to drop. Although I fetch comments by issuing a query to a search engine and getting related articles, comments may not contain information directly related to the query as a comments relevance is determined by the article they are fetched from. Instead, comments represent the discourse occurring over articles on a given topic. Thus, by analyzing the comments I hope to acquire insight into the general opinion about a given query, and the events related to the query. The problem this thesis attempts to address is:

> *Given a query, q, and a set of comments related to the query C_q, present to the user a summary S, which is a collection of sentences that presents the various viewpoints related to q.*

The difficulty in this problem is three-fold. The first task is determining the number of different viewpoints related to the query. The next task is generating sentences that summarizes each of these viewpoints. Lastly, creating a ranked list of sentences representing the prevailing opinions.

For my work I focus on extractive summarization, where summaries are generated by selecting a representative set of sentences from the target corpus. This is in contrast to generative summarization, where I attempt to generate a summary by crafting summaries that follow both natural language rules and the probability distribution of the target corpus.

Term	Term description
t	Term
s	Sentence
c	Comment
S	Summary
C	Corpus of comments
n_t	Instances of term t
N_t	Count of unique terms
N_s	Count of unique sentences
N_c	Count of unique comments
N_{sav}	Average count of sentences per comment
N_{tav}	Average count of terms per sentence

Table 4.1: List of Key Terms for discussing Summarization

When analyzing time and memory complexity I use average case complexity, as the worst case, where every comment has every term, is unlikely. For ease of reference I define some useful terminology in Table 4.1.

4.2 KLSum+

4.2.1 Summary of Enhancements

In this section we go over the design and implementation of KLSum+. KLSum+ is a ultrafast scalable algorithm for summarizing news comments and creating a set of diverse opinions represented in the comments. Its design mixes together the effectiveness of frequency based summarization with the injection of domain specific knowledge created from background models in clustering based methods. Furthermore, this system is designed to extract snippets, which are a collection of sequential sentences that are part of a larger document. Due to the English language naturally referencing content beyond sentence boundaries this focus on snippets, rather than sentences, adds more context and boosts our ability to understand the opinions presented.

A list of innovative components within KLSum+ is as follows:

- Starting from KLSum work I use simple and efficient frequency based models to score terms and present contrastive summaries.
- Merging in knowledge from topic model work, I add background models which incorporate information from a larger corpus to increase the relevance of our summaries to the target corpus.
- By adding length penalties I am able to punish overly short and overly long snippets. This helps reduce noise from comments which on average contain many short sentences, while also avoiding run on sentences.
- By focusing on snippet extraction, selecting multiple contiguous sentences rather than a single sentence from each comment, I decrease the amount of noise and boost the context in my summaries.

Thus, I create a simple yet effective system that incorporates reasoning from various different approaches to summarization. In doing so I hope to set up a simple baseline and show that more complex approaches may be unnecessary.

4.2.2 Implementation

A brief explanation of my algorithm is as follows: I start by isolating important terms for the documents I wish to summarize. This is accomplished by finding terms that are more

prominent in my collection than in my general background model. This background model is generated by a much larger corpus of comments generated from 50+ other topics. Once important terms are scored, I score snippets within comments based on the term scores. I present the most interesting snippets to the user, lower the score of any terms that I showed, then select new snippets. This implicitly selects orthogonal comments that isolate different sides of the story.

Before I discuss the details of KLSum+ I start with a discussion of KLSum[21]. The goal of KLSum, is to generate a summary S^*, of comment collection C that minimizes Equation (4.1).

$$S^* = \min_{S:word(S)\leq l} KL(P_C \| P_S) \tag{4.1}$$

Where l represents the target summary length, P_C represents the empirical unigram distribution of the document collection, P_S represents the empirical unigram distribution of the summary, and $KL(P\|Q)$ represents the K-L Divergence between P and Q. Traditionally, this is estimated greedily by adding sentences to the summary as long as they decrease K-L Divergence.

I extend this concept in many ways in order to generate contrastive summaries. First I borrow from topic and language modeling work, which sometimes uses a background model in lieu of stop words, to eliminate unimportant terms. These background models are based on the assumption that naturally all documents in the English language have some quantity of words that are low in information. This decoupling deals with domain specific noise, which is common in comments where extra characters and misspellings are common. In topic model approaches this model is generated by flipping a coin that decides if a term belongs to a background model. However, I decided to generate a background model directly, by using a much larger corpus of documents. Thus, I define $p(t)$ and $p_q(t)$:

$$p(t) = \frac{|\{c_i \in C | t \in c_i\}| + \delta}{|C| + \delta} \tag{4.2}$$

$$p_q(t) = \frac{|\{c_i \in C_q | t \in c_i\}| + \delta}{|C_q| + \delta} \tag{4.3}$$

where C is a large collection of comments, and $C_q \subset C$ is a much smaller set of comments related to query, q. In order to simplify calculations we add δ comments that contain every term to both corpuses. A simple way of performing this is linear smoothing, adding δ to the numerator and denominator of the calculation for p and p_q.

It should be noted that I only count each term once per a comment. This filtering helps deal with errors in sentence extraction, which are common due to the noise in comments.

For example the string '.....' may generate six uninteresting sentences, which would bias my results if I counted based on sentences. Comment based counts are easier as I already have the comments separated in the corpus. Once I have $p(t)$ and $p_q(t)$ I can use term-wise K-L divergence to score each term:

$$score(t) = \begin{cases} KL(p_q(t) \| p(t)) & p_q(t) > p(t) \\ 0 & otherwise \end{cases} \tag{4.4}$$

This score tells me how important the current term is to the current query. Given this term-wise score I can score snippets using Equation 4.5.

$$score(\hat{s}) = \begin{cases} \frac{\sum_{unique(t) \in \hat{s}} score(t)}{|\hat{s}| + \nabla} & |\hat{s}| < \epsilon \\ 0 & otherwise \end{cases} \tag{4.5}$$

Next I generate a series of candidate snippets by applying Algorithm 4.1 to each comment. In Algorithm 4.1, when given a comment I generate every contiguous subsequences of two or more sentences. Since summaries much longer than ∇ are generally ignored, the score function quickly dismisses any snippets of length longer than ϵ.

Algorithm 4.1 Snippet Selection Algorithm

```
Require: Sentences[1..n] List of Sentences
Require: ScoreFn Function used to score sentences
  function GetSnippet(Sentences[1..n], ScoreFn)
      contiguousSubsequences = empty-list
      for all start, end in combinations(1..n, 2) do    ▷ Generate all sublists of sentences
          contiguousSubsequences.push(sentences[start:end])
      end for
      return max(contiguousSubsequences, key=ScoreFn)
  end function
```

The complete KLSum+ snippet selection process is outlined below:

1. Find a snippet $\hat{s}$ that maximizes the score in Equation (4.5), where $|\hat{s}|$ is the number of words in $\hat{s}$. For my work I set $\delta = 1$, $\nabla = 40$ and $\epsilon = 90$.

2. Present the highest scoring snippet $\hat{s}^*$ to the user.

3. Set $score(t) = score(t) - \sigma(t)\ \forall t \in \hat{s}^*$. For this work I let $\sigma(t) = score(t)$

4. Repeat step 1-3 until score($\hat{s}^*$) drops below a threshold, λ, or when the desired number of viewpoints are harvested.

To ease analysis, the results in this paper are from selecting the top five snippets. KLSum+ has $O(N_t)$ memory efficiency as the background file must be loaded to determine the pointwise term scores. The time efficiency is $O(\binom{N_s/N_C}{2} * N_C * N_{tav})$ which can be simplified to $O(N_s * N_{sav} * (N_{tav}))$.

4.3 Baseline Methods

In this section I discuss the alternate summarization methods I chose to implement and compare against.

4.3.1 Iterative Random Summarization

I start with random summarizer to set up a trivial baseline for my system. This system can be implemented extremely quickly by generating random indices from a sentence list, which are used to select what sentences to include in the summary. On the other hand my random summarizer iteratively assigns each sentence a random score and selects the sentence with the highest score. This adds a slight overhead, as I must iterate over the sentences and recalculate the score before each selection, however this complication provides a more reasonable baseline for diversive sentence extraction methods. For my work I set the desired length to 250 words. Due to the random nature of this summarizer no additional effort was exerted to create contrastive summaries. This algorithm is $O(N_s)$ in terms of computation and $O(1)$ in terms of memory.

4.3.2 SumBasic

SumBasic[40] defines the estimate of a term's likelihood as:

$$score(t) = \frac{n_t}{N_t} \tag{4.6}$$

Given these scores, SumBasic selects sentences with the highest average term score:

$$score(s) = \frac{\sum_{t_i \in s} score(t_i)}{|s|} \tag{4.7}$$

In order to gravitate towards contrastive summaries, each time a sentence is selected, I reweigh each term such that $score(t) = score(t) * score(t)$ before selecting a new sentence. I remove stop words and punctuation before running SumBasic in order to improve the results. Sentences are selected until the desired length is reached, in my case 250 words. This algorithm is $O(N_s * N_{tav})$ in terms of computation and $O(N_t)$ in terms of memory.

4.3.3 MEAD Summarization

The MEAD summarizer[51] is a freely available and widely used summarizer, designed for summarizing news articles, such as those found in the DUC task. According to their manual, the platform implements various summarization algorithms such as position-based, centroid-based, largest common subsequence[57], and query based summarization. Position based summary generation is a trivial gold standard for news summarization, where sentences are deemed important based on how far down the page they are. This approach works well since the starting section of an article, called a lead, should contain a brief statement of the stories essential facts [2]. For comments however this is less likely, since the individuals writing are not professional journalist, and they are not motivated to propagate the same information.

Centroid based summarization attempts to split the various sentences within the corpus into separate clusters. In order to cluster documents, they are first converted to vectors using a score known as Term Frequency - Inverse Document Frequency *tf-idf*. Term frequency measures how frequent a term occurs in a document, for example Equation (4.8) which denotes *tf* as the frequency of a term t in a document d, divided by the frequency of the most repeated word w in the document d.

$$tf(t, d) = \frac{frequency(t, d)}{\arg\max_w frequency(w, d)} \text{ for } w \in d \tag{4.8}$$

idf on the other hand normalizes for common words, which may have traditionally been stop words such as “the”, “and”, “or” etc. The *idf* score for a term t, and corpus C is shown in Equation (4.9) and decreases depending on how many documents contain term t.

$$idf(t, C) = \log \frac{|C|}{|d \in C : t \in C|} \tag{4.9}$$

These two scores are traditionally mixed using Equation (4.10) creating *tf-idf*.

$$tf\text{-}idf(t, d, C) = tf(t, d)idf(t, C) \tag{4.10}$$

Sentence S_{ij}, represent the i^{th} sentence in document j can now be represented as the vector

$$S_{ij} = [w_1 w_2 ... w_n] \tag{4.11}$$

where w_i is the *tf-idf* score of term i in sentence S. Once these vectors are created they can be clustered using traditional clustering algorithms, for example K-means. MEAD uses an algorithm called CIDR which uses the similarity score in Equation (4.12) to assign clusters to documents. A document D is assigned to a cluster Cl if this score is greater than an arbitrary threshold. If no cluster is assigned to a document then the document spawns a new cluster.

$$sim(d, Cl, C) = \frac{\sum_{t \in d} tf(t, d)tf(t, Cl)idf(t, C)}{\sqrt{\sum_{t \in d} tf(t, d)^2}\sqrt{\sum_{t \in Cl} tf(t, Cl)^2}} \tag{4.12}$$

In order to generate sentences for the summaries, sentences are selected based on their average distance from all the centroids. Thus, sentences that overlap all the topics within the document set are preferred. Sentences are also penalized for being too similar to selected sentences by eliminating sentences that cover the same or less topics than any other sentence. With the centroid system this can be estimated by sentences that fall into approximately the same scores for each cluster.

The centroid score of a term to a cluster Cl is defined as $CS(t, Cl, C) = tf(t, Cl)idf(t, C)$. The centroid score of sentence is as follows:

$$CS(s, Cl, C) = \sum_{t \in s} CS(t, Cl, C) \tag{4.13}$$

In order to select contrastive summaries, I used the MEAD maximal marginal relevance reranker[16]. This reranker selects sentences that maximize the following score:

$$score(s) = \lambda(CS(s, Cl, C)) - (1 - \lambda)(\max_{s_j \in S} Sim(s, s_j)) \tag{4.14}$$

Thus, sentences are selected that are similar to the cluster but sufficiently different from sentences already selected. The memory efficiency of CIDR is $O(N_{Cl} * N_t)$ and computational complexity is $O(N_{Cl} * N_s * N_{tav})$ where N_{Cl} is the number of clusters.

4.3.4 LexRank Summarization

The LexRank summarizer is based on the concept of prestige in social networks. Generally, the goal of LexRank is to find sentences with a high eigenvector centrality. The first task is building the adjacency matrix marking sentences that are close together. Cosine similarity in *tf-idf* space defined as:

$$sim(s_i, s_j) = \frac{\sum_{t \in s_i, s_j} tf(t, s_i) tf(t, s_j) idf(t, C)^2}{\sqrt{\sum_{t \in s_i} (tf(t, s_i) idf(t, C))^2} \sqrt{\sum_{t \in s_j} (tf(t, s_j) idf(t, C))^2}} \tag{4.15}$$

for sentences s_i, s_j where $tf(t, s)$ is the number of occurrences of t in s, and $idf(t, C)$ is the inverse document frequency of t in Corpus C. The adjacency matrix, M, can be constructed by following the rules in Equation (4.16).

$$M_{ij} = \begin{cases} 1 & \text{if } Sim(s_i, s_j) > \lambda \\ 0 & otherwise \end{cases} \tag{4.16}$$

Given this adjacency matrix, I can perform the well known PageRank algorithm [46] to extract the centrality of each sentence in the graph. For my work I let $\lambda = 0.2$, however I found that summaries did not change significantly for $\lambda = 0.1$ or $\lambda = 0.3$.

This algorithm is $O({N_s}^2 * N_{tav})$ in terms of computation, and $O(N_s + N_t)$ in terms of memory due to sparsity of graph. The continuous version of this algorithm, that uses a function on the similarity scores as edges, is not sparse. Thus it exceeded memory limits during my evaluation.

In order to obtain contrastive summaries, I perform k-means clustering on the data before I execute the LexRank algorithm. I then select sentences from each of the separate clusters and concatenate them. This approach was adopted inside the Dragon Toolkit[65].

4.3.5 Topic Model Based Summarization

Topic model based summarization starts by creating a document-term matrix. This matrix is then factored into two smaller matrices, a document-topic matrix, θ, and a topic-term matrix, ϕ. There are a variety of methods designed to decompose this matrix. However, recently Latent Dirichlet Allocation(LDA) has gained some traction for multidocument summarization [1] and email topic detection[12]. In order to estimate ϕ and θ I perform

Gibbs sampling to generate a set of topics from the Dirichlet distribution. Next I find sentences that maximize $p(s|T_j)$ for each topic T_j.

For my work I use the modified partially generative estimation of $p(s_i|T_j)$ shown in Equation (4.17)

$$p(s|T_j) = \frac{\sum_{t_i \in s} p(t_i|T_j) * p(T_j|c_s) * p(c_s)}{|s|} \tag{4.17}$$

where c_s is the comment containing s, $p(T_j|c_s)$ is the estimate of topic T_j being drawn from c_s, and $p(c_s)$ is the prior for generating comment c_s.

I generate a contrastive summary by sampling T_j from one of five topics drawn from a multinomial distribution and choosing unselected sentences that maximize $p(s|T_j)$. Thus, I assume that each comment cluster is composed of approximately five separate topics and attempt to select the most interesting sentences. I attempted to add background models to the LDA implementation, but results were drastically worsened, hence for this work I rely on stop words to eliminate unimportant terms.

This algorithm requires $O(N_C * N_{tav} * N_{sav})$ memory. The computation complexity is $O(I * |T| * N_C * N_{tav} * N_{sav})$, where $|T|$ represents the number of topics and I is the number of iterations of Gibbs Sampling requested. $I = 1000$ and $|T|$=5 for my work.

I also use SyntaxSum [10] which is an open sourced summarization system, created by Darling at the University of Guelph, which uses topic models to estimate both the topics within a corpus and the syntax class of words. This extension is used to eliminate low content function words within the English language.

4.3.6 Aspect Mining with Sentiment Methods

Sentiment analysis based summarization relies presenting sentences to the user that are for or against a given aspect of a story. In order to accomplish two pieces of information need to be determined: a sentiment lexicon and the aspects of a story.

The ideal sentiment lexicon maps each word in a corpus to either a positive or negative sentiment. In order to determine the sentiment of a term one can hand label a relatively small set of comments or sentences. Using this human labeled data I am able to then write a classifier that predicts the sentiment of each word within the sentence. Furthermore, once this classifier is written I can use term co-occurrence to determine words with similar sentiments. This is the general approach that is taken with product reviews, as most

Word Class	Positive	Negative	Description
JJ	0.75	0	characterized by quickness and ease in learning
JJ	0.5	0	showing mental alertness and resourcefulness
JJ	0.5	0	capable of independent and intelligent action
JJ	0	0.125	improperly forward or bold
JJ	0.5	0	elegant and stylish
JJ	0.5	0	quick and brisk
JJ	0	0.875	painfully severe
NN	0	0.625	a kind of pain such as that caused by a burn
VB	0	0.375	be the source of pain

Table 4.2: Sentiments for every word sense of the word 'Smart'

services provide a star rating, and this rating can be treated as a free source of human judgments.

For my comment system, due to lack of resources I instead rely on the well known SentiWord.net lexicon. This lexicon is built by walking the WordNet synonym graph and assigning either positive, negative, or objective scores to each word. The assigning of scores is done by three semi-supervised classifiers that are bootstrapped using manual training data. The original documentation can be consulted for more information on this process [3]

SentiWord.net also assigns sentiment based on word sense, which is the underlying meaning of a word. For example the term "smart" can mean either "elegant" or "bold". Furthermore, many words can fall into different parts of speech, for example "good" can refer to a commodity or be used as an adjective to describe a noun. By applying part of speech tagging one can perform word sense disambiguation which allows me to use these fine grained metrics to increase the success of sentiment based methods. SentiWord.net also has more complex word sense disambiguation, which provides the sentiment of every possible definition of a word, as exemplified for the word "smart" in Table 4.2. Determining which definition is used is a much more difficult task, thus I instead average the word sense for each part of speech (adverb, adjective, noun, and verb) and use part of speech tags to determine the sentiment of each word.

In reviews, aspect extraction is the task of extracting which aspects of a product users are generally discussing. Similarly, for comments aspect extraction is the task of extracting which aspect of a news topic that users are commenting on. Multiple approaches exist for aspect extraction, and it can be said that every summarization algorithm has some

form of aspect extraction built in. For example, KLSum+ uses K-L Divergence to determine interesting terms, and topic modeling uses word clusters determine various important aspects.

In sentiment analysis work however, more powerful natural language processing techniques are used to extract sequences of important noun phrases. The approach I took is similar to the work done by Blair-goldensohn et al[5]. In order to determine importance I executed the following steps.

Step 1
: Filter out sentences that do not contain any sentiment words.

Step 2
: Filter out noun sequences that do not occur in a syntactic pattern indicative of a discussed topic. This is explained further below.

Step 3
: Count all occurrences of each noun phrase, produce summaries for the top λ sentences, where $\lambda = 5$ for my work. For my work I also decided to use KL Divergence to determine how unique the noun phrase was to the current topic.

Step 4
: Optionally, run an association miner or topic modeling system to find correlations between noun phrases and create distinct subtopics.

Step 5
: For each sentiment S and each noun phrase NP, I select the highest scoring sentences containing NP using Algorithm 3.

Informally, this algorithm starts by dismissing sentences and noun phrases that are unimportant, generated either by flaws in the part of speech system, or the general use of nouns in the English language. Importance is then measured by frequency and occurrence in sentences that are classified as discussion using syntactic patterns. Sentimental sentences containing each noun-phrase are then selected based on their ability to maximize the score in Algorithm 4.2. This algorithm roughly estimates the use of negation in natural language, however does not deal with sarcasm or other flaws in natural language approaches.

The syntactic patterns I used are shown in Table 4.3. These patterns are discussed in more detail by Khan et. Baharudin[29]. Generally, these patterns are chosen because their high correlation with interesting aspects of a review.

Algorithm 4.2 Sentiment Scoring Algorithm

```
function GetSentiment(Sentence[1..n], Topic, TargetSentiment)
    NegationTerms = {"n't", "not", "never"}        ▷ List of terms that flip sentiment
    score = 0
    if Topic not in Sentence then
        return -1
    end if
    for i from 1..n do
        for j from i-3..i-1 do                     ▷ Detect negation before current word
            if Sentence[j] in NegationTerms then   ▷ Assume Sentence[-k] = "" ∀k ≥ 0
                TargetSentiment.reverse()
            end if
        end for
        score += sentiment(Sentence[i], TargetSentiment)
    end for
    return score / n
end function
```

Pattern	Type	Example
NN VB RB JJ	vBNP	camera/NN is/VBZ so/RB compact/JJ
NN VB RB JJ	vBNP	camera/NN is/VBZ so/RB light/JJ
NN VB RB JJ	vBNP	camera/NN produces/VBZ fantastically/RB good/JJ Pictures/NN
DT NN VB	dBNP	The/DT viewfinder/NN reflects/VB
DT JJ NN VB	dBNP	The/DT LCD/NN sees/VB
NN IN NN	iBNP	Quality/NN of/IN Photo/NN
JJ IN NN	iBNP	Range/NN of/IN Lenses/NN

Table 4.3: Syntactic Patterns Predicting Key Noun Phrases

A brief explanation is as follows: A base noun phrase (BNP) is a phrase containing 0 or more adjectives, followed by 1 or more nouns.

A linking verb based noun phrase, vBNP, is a BNP followed by a verb then followed by an adjective or adverb.

A definite base noun phrase, dBNP, is a BNP preceded by "the".

A preposition based noun phrase, iBNP, is a noun phrase with the preposition "of" between two BNPs.

4.3.7 Other Methods

I dismissed first sentence selection, which displays the first sentence of each article, for two reasons. First, despite its success in summarizing journals and blogs, it generally fails on comments. Its success on journals is likely due to the general attempt of news articles to convey the most important aspects of the story within the first sentence. However, untrained commenters have different motives, and thus this principal does not hold true. Second, with a large number of comments, choosing the first sentence still generates tens of thousands of sentences, which will still require summarization.

Temporal methods, which are successful on tweets have been dismissed because they are generally used to determine what articles in a larger corpus are related to an event. However, I solved this classification problem in an earlier step by filtering to a set of comments related to articles about my query.

4.4 Experimental Setup

In this section we discuss the results of executing all the algorithms discussed. The goal of this section is to describe the data we collected, and concretely show how our system is able to outperform every alternative in both efficiency and quality. However, due to lack of human judgment we are unable to formally evaluate our system using our full dataset and thus use the manually annotated DUC 2007 task for our quality evaluation.

4.4.1 Data

For this case study I use two datasets. The first collection is composed of the data from the DUC 2007 task. In this task competitors are given a set of topics, with each topic

Website	Number of Articles	Number of Comments
CNN.com	2665	297273
HuffingtonPost.com	16022	299812
RawStory.com	11307	372351
FoxNews.com	9257	529033
WashingtonPost.com	8413	905225

Table 4.4: Document Count for Top 5 Websites

Topic	Number of Articles	Number of Comments
Alternative Energy	141602	561529
Economy	139892	595959
Health Care	170355	708449
GOP Primary	219230	802635
Rick Santorum	223600	924523

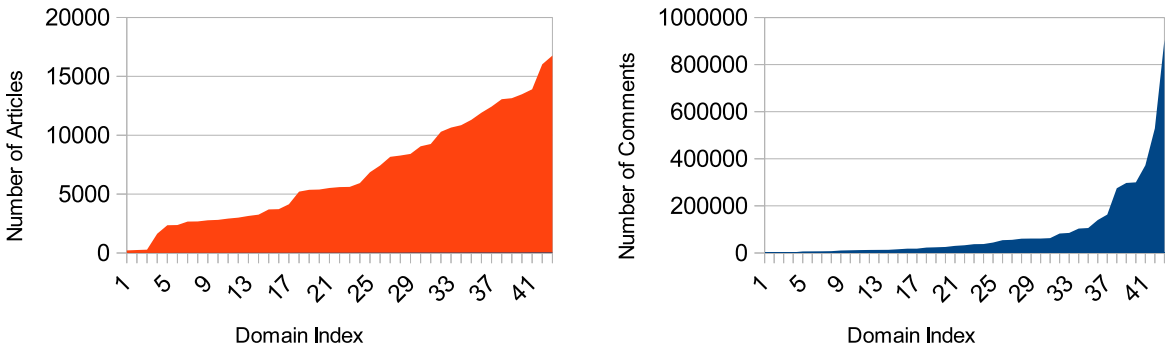
Table 4.5: Document Count for Top 5 Topics

containing up to 25 news articles. The task entails reading these documents and creating a summary of all the articles present for each topic. Each topic is also accompanied by summaries generated by at least four human judges, who attempt to create summaries that help answer a predetermined information need. During the DUC task these summaries are used as a gold standard, and I use these to create a baseline evaluation for the effectiveness of each system I implemented.

My second collection was harvested by using the crawler discussed in Chapter 3. The topics I selected were based on trending topics from the online community Reddit.com. In Table 4.4 I show the top five websites sorted by the number of comments on articles fetched. I also graph the overall distribution of comments over all the websites in Figure 4.1. From this distribution I notice that the majority of comments come from a few key websites, however the number of articles is not necessarily co-related with the number of comments. Thus creating a few high fidelity parsers for very popular websites can help grab the majority of comments. However, a generalized parsing solution is useful since different news websites may grab a different audience.

In Table 4.5 I show the five most commented on topics that I searched for. Figure 4.2 graphs the number of articles and number of comments for all the topics I searched for. This data shows that most of the topics achieved more than 100,000 comments.

Before summarization I perform no spam filtering, however I found that many irrelevant



(a) Distribution Per Website

(b) Distribution per Topic

Figure 4.1: Graph of Number of Left: Articles and Right: Comments per Domain ordered by Rank

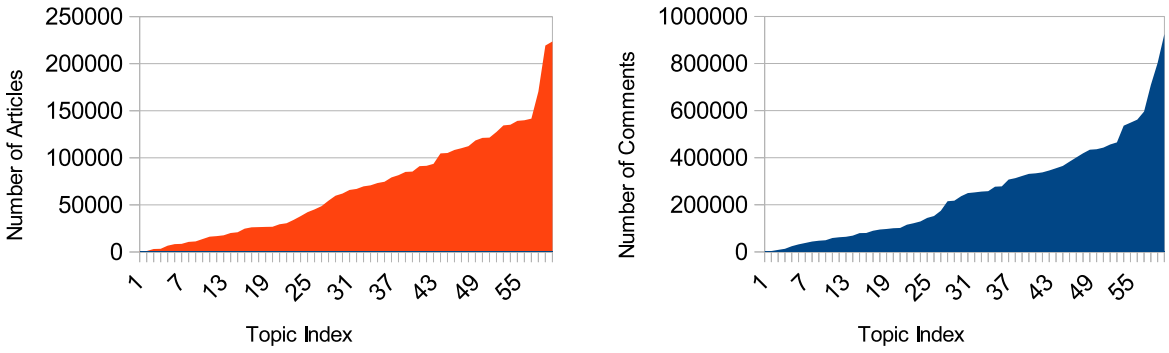


Figure 4.2: Graph of Number of Left: Articles and Right: Comments per Topic ordered by Rank

articles were returned by Google. This occurs due to the indexing of dynamic content such as related articles or current news. In order to improve relevance I filter away articles and related comments if they do not contain all the query terms.

It should also be noted that the comment data was extremely noisy, with the average comment containing 3.92 sentences and the average sentence containing 1.24 terms. It should be noted that these terms can include any stop words and punctuation characters, as well as artifacts produced from the failure of natural language based sentence tokenization when applied to noisy comments.

4.4.2 Efficiency

For my timing evaluation I executed my tests on a Quad Core AMD Phenom II X4 955 with 7918 MB of RAM and Ubuntu 12.1. Due to the fragile nature of timing information I briefly describe each implementation and my efforts to speed up each system. It should be noted that SyntaxSum and MEAD results are not presented, as both these systems were designed for smaller document sets and thus were unable to summarize my large corpus.

For sentence tokenizing, word tokenizing, and part of speech tagging I use the Stanford CoreNLP toolkit. All my systems are implemented in Java except for LDA, where I use the open source C++ implementation by Darling[1]. By using C++ and removing stop words/infrequent words before computing ϕ and θ I help ensure the algorithm terminates within a reasonable amount of time. This also requires the creation of inverted index of documents to the terms they contain. Given this doc-term matrix I use Gibbs Sampling with 1000 iterations to estimate ϕ and θ.

For LexRank I use the Java implementation by Drexel University implemented as part of the Dragon Toolkit[65]. This toolkit implemented both clustered LexRank and traditional LexRank. I modified the code slightly to use sparse matrices due to out of memory errors when attempting to allocate a dense N_s x N_s matrix. This library was selected due to its ability to handle the relatively large amount of sentences, and the amount of effort put into optimizing the system. Before executing KLSum+ I generate a background model by parsing all the terms in my full corpus of documents and tracking how often each term occurs. This operation only needs to happen once, and replaces the *tf-idf* file used in other summarization algorithms.

I start by analyzing the four fastest algorithms. In Figure 4.3 I show that Random, KL-Sum+, and SumBasic perform on par with each other. Surprisingly Random and SumBasic

[1] `www.uoguelph.ca/~wdarling/code/syntax_sum.tar.gz`

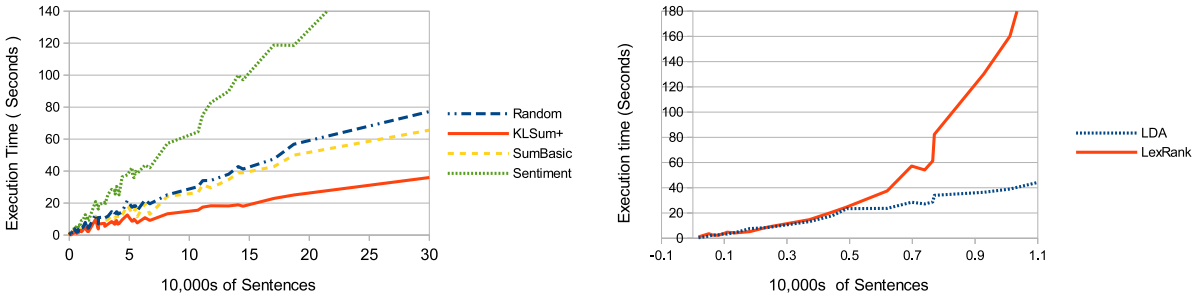


Figure 4.3: Summarization Time versus Number of Sentences

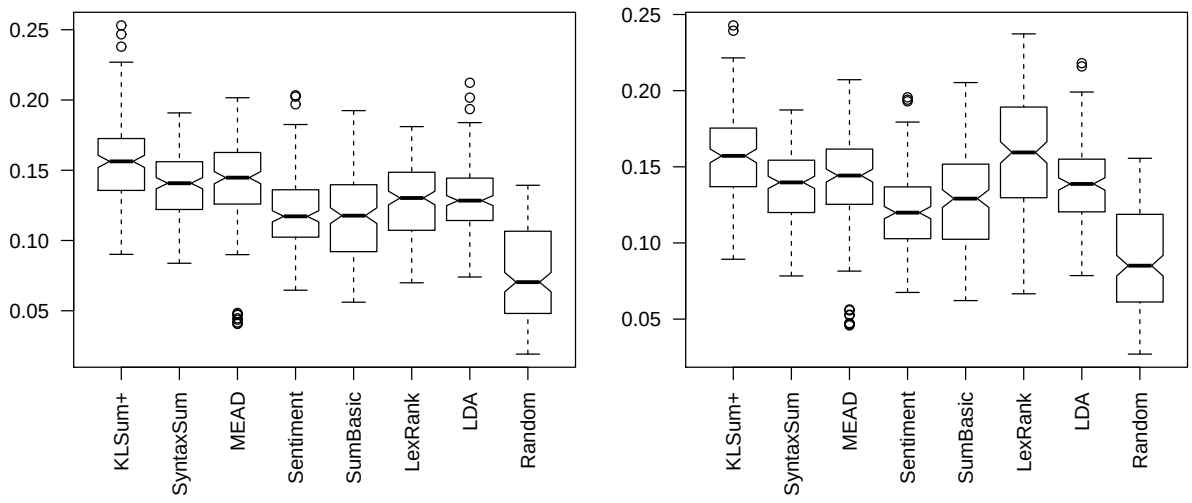
end up slower than KLSum+. This was likely due to the number of iterations that needed to be executed to achieve 250 word summaries. On average KLSum+ was required to score snippets $1/8^{th}$ as many times as SumBasic and $1/16^{th}$ as many times as the Random summarizer.

This figure also shows that that LDA and LexRank performed much worse. LexRank specifically was unable to summarize more than 30,000 sentences in under five hours. This failure occurs due to the inability to store the document matrix in memory and the need to constantly flush the matrix to disk.

4.4.3 ROUGE Scores

Due to the size of data, I found it difficult to generate human judgments. Summarizing even 3000 comments becomes difficult. Inouye and Kalita[25] accomplish this task by preclustering tweets and presenting them to annotators who then attempt to generate summaries. However, due to the biases introduced by this method, I decided to evaluate KLSum+ by comparison to the DUC 2007 task[7]. In this task, competitors attempted to create 250 word summaries of 45 document sets, each containing 25 news wire articles. The summaries generated are then evaluated using ground truth, which was created by 7 human judges manually.

When I evaluate my system I chose to use ROUGE SU4, since it is both generally accepted for evaluating summarizers[36] and designed to measure the coverage of automatic summaries over a set of gold standard summaries. ROUGE SU4 for given gold standard is formally defined in Equation (4.18):



(a) ROUGE SU4 scores

(b) SU4 Precision Scores

Figure 4.4: Scores for each Summarizer Against DUC 2007 data set

$$\text{ROUGE SU4}(s^*, \hat{S}) = \frac{\sum_{\hat{s} \in \hat{S}} \sum_{g \in s^*} \min(tf_{g,s^*}, tf_{g,\hat{s}})}{\sum_{\hat{s} \in \hat{S}} \sum_{g \in \hat{s}} tf_{g,\hat{s}}} \tag{4.18}$$

Where s^* represents the generated summary, $\hat{S}$ represents the set of gold standard summaries, and $tf_{g,S}$ represents the number of times the n-gram g occurs in summary s. In ROUGE SU4 the n-grams used are the union of the set of unigrams with the set of skip bigrams with at most 4 tokens skipped. I also run a Porter stemmer but do not remove stop words before calculating ROUGE scores as described by the DUC 2007 Task. Similarly to measure precision I use Equation (4.19):

$$\text{precision}(s^*, \hat{S}) = \frac{\sum_{\hat{s} \in \hat{S}} \sum_{g \in s^*} \min(tf_{g,s^*}, tf_{g,\hat{s}})}{|\hat{S}| \sum_{g \in s^*} tf_{g,s^*}} \tag{4.19}$$

For this comparison, I first implemented KLSum+ and the 6 algorithms described in Section 4.3. I then summarized the DUC 2007 data set using these algorithms along

with the two open source toolkits MEAD and SyntaxSum[10], which were chosen for their accessibility and published results relating to the DUC data set. When using background and syntax models in KLSum+ and SyntaxSum I generate the models from the full DUC 2007 corpus before creating summaries for the subtopics. The notched box plots of the recall and precision scores, as calculated by the ROUGE toolkit, are presented in Figure 4.4a and Figure 4.4b. These notched box plots display the following information:

1. The median, which is the central depression
2. The 95% confidence interval on the median, which is represented by the notches.
3. The interquartile range, i.e. the 25th to 75th percentile, represented by the boxes.
4. The spread, represented by the whiskers, which contains 99.3% of the data
5. Outliers excluded from the box plot, which are represented by circles below or above the box.

In order to reproduce these graphs the following code written in the R programming language can be executed on the output of the ROUGE score calculator provided with the DUC task.

```
1 data <- read.csv(inputfile, header=FALSE)
2 boxplot(data, notch=TRUE, las=2)
3 error.bars(data, add=TRUE)
```

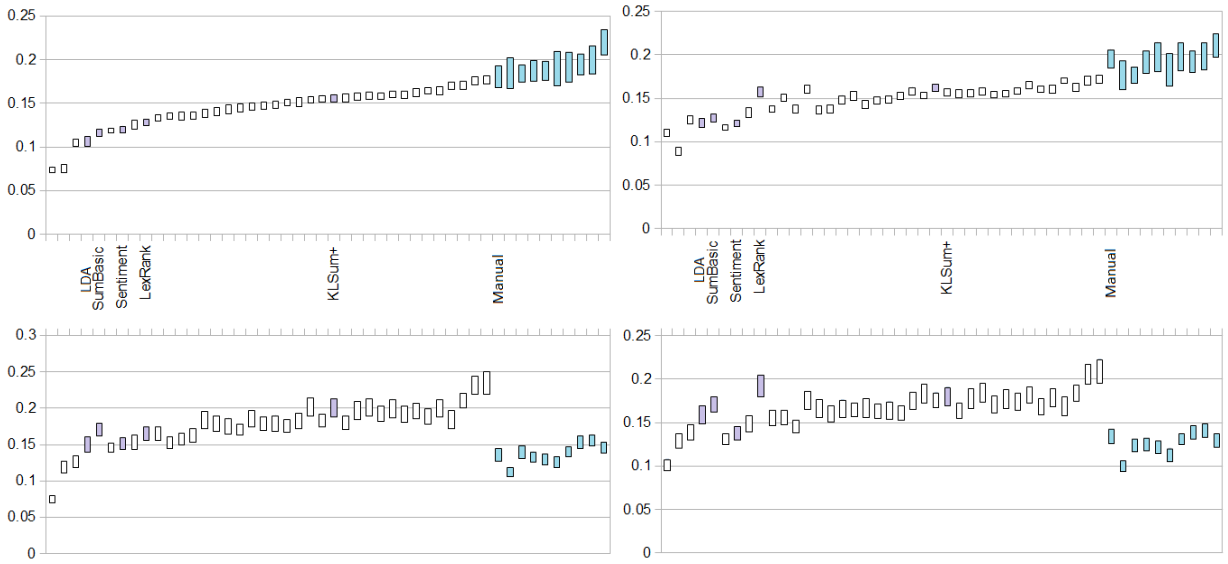
Listing 4.1: R code for generating notched box plots

These graphs show that KLSum+ matches or exceeds the performance of competing algorithms in both ROUGE SU4 and SU4 Precision scores.

Alternative Evaluation

As part of this work I also attempted to use the systems I discuss in Section 4.3 to automatically evaluate KLSum+. This theory was hypothesized because of a general belief that more complex and well known methods would perform better and serve as superior base lines. Furthermore I reasoned that the best summarizer would be able to concisely summarize all the summaries generated by its competitors.

Thus, my evaluation involved generating summaries with every system for the DUC 2007 task, then using the generated summaries as ground truth and evaluating all the



(a) ROUGE SU4 Scores

(b) SU4 Precision Scores

Figure 4.5: DUC Scores with **Top:** Human Judgements **Bottom:** Automatic Judgements

competitors. If I am able to show a high correlation between rank with human judgments and rank with automated judgments, I could reject the null hypothesis.

The results of this work is shown in Figure 4.5. From this work I notice that there is no real prediction power from using automatic judgments. Therefore I dismissed further work in this category. It should be noted that when using the full list of summarizers from the DUC task some outperform KLSum+. Unfortunately, they have closed implementations and thus we were unable to use them for our other evaluations. Furthermore, these systems may be complex machine learners that rely on signals that are strong in professional content but weak in comment data sets, such as first sentence selection,.

4.4.4 Example Summaries

In this section I present some summaries and highlight some obvious flaws when looking at the empirical data. Table 4.6 shows the first 100 words of the summaries produced by each algorithm, over comments about the controversial 2012 “SOPA” bill.

By glancing at these summaries I notice that SumBasic, Random, and LDA systems perform extremely poorly. These summaries approach bag of word models, that stitch

together sentence fragments in order to generate a summary. This failure occurs because all these systems attempt to select sentences with extremely important terms. However, since comments are extremely noisy, these systems are able to find extremely short fragments that match their selection criteria. The three other systems bypass this issue by using drastically different methods. Random specifically appears extremely poor, however performs as expected since the average sentence contains 1.24 terms.

Sentiment analysis systems require part of speech tags and sentiment laden words within each selected sentence. Thus sentences naturally have to be long enough to be understandable before they are selected. LexRank on the other hand avoids small sentences due to their low centrality; a sentence that is short has a very low cosine similarity to other sentences and thus is not selected. However, this can also lead to run on sentences being selected which can become abundant with larger data sets. KLSum+ is designed to solve this problem directly, penalizing short sentences for their lower information content, and longer sentences for their low information density.

The sentiment based system is unfortunately relatively fragile to the sentiment lexicon. Specifically sentences containing very strong positive or negative terms (such as good or bad) can often overwhelm the sentiment scores, and thus hide some of the discussion occurring. On the the other hand KLSum+ calculates the scores of each term adaptively much like SumBasic, and thus the lexicon is able to select terms that are highly relevant to the current topic. The aspect mining component of the sentiment summarizer does something similar, however it is used to filter out irrelevant material rather than select important material.

In Table 4.8 I present the top 25 terms as decided by the algorithms I implemented. For KLSum+ and SumBasic we chose these terms by running the algorithm five times and taking the top five terms each time. Thus, if a term is not selected it stays in the list in the next iteration. For LDA we generate five topic models, and select the top 5 terms from each model. Finally, from the aspect miner we just select the top 25 aspects by frequency in interesting sentences.

It can be noticed that generally there was some overlap, however overall they highlight different aspects as important. These differences highlight the variations in algorithms, and is some indication on the amount of coverage by each system. It is also noteworthy that SumBasic and KLSum+ are very similar, however KLSum+ diverges from the initial state much faster, boosting diversity.

In Tables 4.9 to 4.11 we present various summaries generated by KLSum+ over various other queries. We present these to allow the reader to judge the quality of summaries generated by our system. The topics were chosen due to their relatively polar discourse or

Algorithm	Summary
KLSum+	I disagree with Lamar's SOPA bill, This bill can destroy Internet and disrupt global economy even they stop the piracy. This will make it worse.
	How should we ensure that the pirate sites don't operate in America? The PIRATE SITE violate Copyright law and facilitate IP theft.
	If they don't shut up, we can take are all those: SOPA, PIPA and Hollywood. We know who sponsored those bills.
	Yes, Google makes enormous profits from websites that sell infringing content. And from users that search for images of feet being tickled with feathers.
SumBasic	Corporations are people! Obama opposed SOPA. Time is money. A free internet? Media companies !!! The government! Google condemns the bill. These bills won't actually STOP piracy. Abolish intellectual property law. VOTE RON PAUL!! Business is Business. foreign sites?) Occupy Congress! Bad, bad idea. Wrong thing, wrong time, wrong country. Tea Party Thesaurus. Ass real. President Nosferatu! Republicans AND Democrats. Like Palin she lives in her own little world. Can't debate on reason. Maybe Lamar Smith's website should be shut down for violating copyright laws.
Sentiment	(+) You were lucky this time. Well he is being truthful in a way. Abolish intellectual property law. The law is absurd. Fareed is good people. The internet works well. If you really want to protect free speech and you think SOPA is bad, then please suggest a viable alternative, instead of just criticizing and complaining.
	(-) It's time to kill SOPA totally. I in no way agree that intellectual property should be passed about freely. Good point– sane people should embrace these Tea Party children. Internet piracy is all for the common good. SOPA is about making copyright infringement criminal instead of tortious.
LexRank	If I can't do what I want to do with the internet, then I don't need it. I have been campaigning against Sopa pipa I even did the black out BUT I have copyright issues online ALL the time and sites like megaupload, filesonic and other sites are a real problem they KNOW what there users are doing and will NOT do anything to help I get 100's of files removed from sites like this every week but they keep letting the members do it again I have Asked them to BLOCK MY name so it would be hard for people

Table 4.6: First 100 words for summaries with the query "SOPA"

Algorithm	Summary
LDA	No government money please. Vote Obama! Do not control the internet with the SOPA bill. "rogue foreign websites" / "primarily dedicated to infringing copyright." Corporations are people! Smaller government? They must watch Fox News. SOPA is a horrible bill! The answer is many foreign sites simply ignore DMCA notices. Unions aren't people! Fareed is good people. Fox News ought to try it. The Internet should be FREE! The PIRATE SITE violate Copyright law and facilitate IP theft. http://www.youtube.com/watch?v... Important people. The money you pay into Social Security does not fund YOUR retirement. Why is story news ANYWHERE?? NO SOPA!
Random	. ? ??? ! Whoopee! ?!? OMG!!! 1. Ah. Baloney. OH! OMG ! Please!!! Dip-Shit! Stopped. Ever. WTF? Bwahahahahaha!! Yeah! McCain?!! What? Oops. please! Right. See. Teapartiers? Therefore. Nothing. Hah! but. Wow. indeed . Legally. Whatever. WHY? Why? Anti-Sopa? ¡rolls eyes=""¿. omg. False. 2. WUT? Ha. PS. Quick! nah. 4. Crazy. Sweet. 202. . GHASP!!! CHRISTIAN! Crap. GOOD! Yes? Geitner? Haha! Whoa. Scary? WHOOOPS. No. So? Wonderful. Hilarious! Almost. AWESOME!!!!! Rant? 201. WOOO! Creator? Bi-racial. Stole? Tee-hee-hee. 104. 102. Period. You? com . Arbitrary? Hmm? Huh? except. Shock! McCain? Thanx. Hmmm. No? ..... Seriously? Emails ? THIS. 3.

Table 4.7: Continued: First 100 words for summaries with the query "SOPA"

Algorithm	Top 5	Top 10	Top 15	Top 20	Top 25
KLSum+	sopa internet piracy copyright bill	copyright pipa sites hollywood content	pipa hollywood content smith bills	content smith censorship mpaa websites	hollywood censorship mpaa dodd trump
SumBasic	people internet sopa money bill	internet sopa money bill obama	internet money bill government time	internet bill government law country	bill government law country media
LDA	sopa internet bill sites content	obama president people party republican	people money tax government security	people internet money industry free	news time media fox read
Aspect Miner	people internet sopa money bill	government obama time way law	country meia piracy us years	something congress sites nothing thing	president copyright news content world

Table 4.8: Top 25 terms from each of algorithm

opinionated audience. From the summaries we can quickly spot this polarity, even from as few as two snippets.

For example in the "Evolution" summaries we are able to see two competing opinions on the argument of "Creationism versus Evolution". In other topics such as "BP" we are able to see various opinions all discussing the same stance, i.e. against oil drilling, however these arguments focus on different issues people have with oil drilling.

Query	Summary
Gun Control	Guns TAKE lives. Your gun is meant to KILL someone.
	Legal gun ownership, and this guy wasn't a legal gun owner, but a very sick person, doesn't cause mass murder. Get real! Did you know that 20 children were killed in an elementary school in China with a knife yesterday.
	There are however mentally ill people that should be institutionalized. The kid had a record of violence and mental illness that would have had him being properly treated thirty years ago. I have something constructive to say. Arm teachers and let law abiding citizens carry were they wish.
Gay Marriage	How does love equal freak? If you are anti-gay marriage then you don't have to marry a gay person.
	Married Heterosexual couples do have rights that homosexual couples do not have...such as hospital visitation, inheritance, immigration, etc. This IS a civil rights issue, and not a religious issue.
	Besides, isn't this the same old discredited "Bible says it's a sin" argument? I mean, why should gays want to unaquire homosexuality?
	No, a mixed race couple consisting of one woman and one man can actually be married. A union between two people of the same sex is something else.
Israel	Palestinians living in Israel are given the same rights as Israeli Jews. There is no apartheid in Israel.
	Before 1967, The West Bank and Gaza were annexed territories of Jordan and Egypt respectively. But no one considered them "Occupied" then, why?! The Arabs rejected the original 1948 partition of what was then known as Palestine into Arab and Jewish countries.
	Israelis are sitting fat and happy on their stolen land. On the other hand, the Arabs have long had a peace plan on the table endorsed in 2002 by Palestinian President Arafat and the entire Arab/Muslim world including Iran.
Nuclear	There are many studies about the safety of Nuclear Power Plants. In Japan, no one has died from radiation exposure.
	If Israel does send its military to attack Iran it could easily lead to World War III. Iran is not Iraq.
	Fukushima will teach us even more; and they will build safer reactors in the future. But I believe containment is more important even than the inherent reactor design. Chernobyl would have been fine with containment.
	We need to expand nuclear energy as a safe, clean electricity source. Further, that plant in Japan was built on a fault-line, near an ocean.

Table 4.9: Example Summaries from KLSum+

Query	Summary
Rick Santorum	Rick Santorum's father, Aldo Santorum, earned a doctorate degree in psychology and worked as a clinical psychologist for the VA. Aldo Santorum said that the greatest gift he received was the GI Bill, so that he could attend college after serving in WWII. Rick Santorum's mother, Catherine Dughi Santorum, worked as an administrative nurse. Rick Santorum's wife, Karen Garver Santorum, is a former nurse.
	Romney carpet bombs his opponents. Romney spends most of his money and time attacking his opponents rather than running on his record or his vision of a Romney Presidency. This is what the Republican establishment and Romney people don't seem to grasp.
	Ron Paul will win Iowa and South Carolina and Florida. Ron Paul will be our next President! Ron Paul does have the Hispanic vote and the vote of our armed forces! Go Ron Paul.
Evolution	Evolutionists have been very successful in posturing the issue as "Science vs. religion". The truth is that evolution is a religion and there is more real science supporting creation than evolution. The "evidence" that is offered to prove evolution is based on the unscientific ASSUMPTION that evolution has occurred.
	Remember how I told you the difference between a theory and a scientific theory? The "Creationism Theory" has no well-substantiated, well-supported, well-documented explanation for our observations. In fact, this Creationism theory is just a "guess". The normal every-day use of the word theory.
BP	BP ran the rig. BP was pulling up the oil. BP hired people to do work for them and failed to make sure the work was done right. BP stood to make all the big profits. BP is to blame.
	As bad as the Deepwater Horizon Spill was, there was another spill in 1979 that spewed oil into the Gulf off the Mexican coast for nine months and three years later, there was no residual evidence of that spill. The Ixtoc I oil spill in the Bay of Campeche didn't generate near the widespread hysteria that the Deepwater Horizon Spill did, mainly because it mostly affected Mexico.
	Of course after the accident his position on drilling reverted back to his original stance of no drilling. Well as we all know thats no drilling for American companies.

Table 4.10: Example Summaries from KLSum+

Query	Summary
Northern Gateway Pipeline	"Two litres of oil spill for every one million that are transported. In Canada, three million litres cross pipelines every day." The Canadian Energy Pipeline Association needs to check its math. That works out to 2190 L of oil spilled in Canada a year. The 2011 Rainbow pipeline leak north of Peace River was the largest oil spill in AB in more than 30 years: 4.5 million litres of oil.
	Why shouldn't this hearing be dominated by BC voices, and compelled to make a decision based on BC voices? The project is a BC project, crossing BC land, exposing BC people to any risks associated with it that would affect BC's lands and peoples.
Arab Spring	First in "The Arab Spring," Tunisia will be the first with the reality of The Arab WInter with Islamist leading. The rest of "The Arab SPring" will turn into a very cold Arab winter and President Obama helped that situation. He is either that Aprentice sorcerer bumbling in foreign policy or he prefer Arab to the USA and Israel.
	Egypt has Sharia Law in its constitution. Egypt is not a democracy. Egypt is a Dictatorship. The Muslim Brotherhood is paid by the Saudi Arabian Dictatorship to overthrow countries for Saudi Arabia. The Saudi Dictatorship is the bloodiest, most genocidal, Dictatorship in the Middle East.
Oil Prices	Oil prices jump because of Irish financial problems. Oil prices jump because of speculation. Oil prices jump because of over-supply. Oil prices jump because of bad weather in Gulf. Oil prices jump because of under-supply.
	We The People of the US own the vast majority of the natural gas in the US. Gas companies don't own the natural gas. They drill, complete and produce the natural gas for the own of the natural gas.

Table 4.11: Example Summaries from KLSum+

4.5 Discussion

In order to fully understand the byproducts of this research it is useful to summarize the results and implications of my evaluation. Overall KLSum+ was designed to be a simple and lightweight summarization system that acquired the advantages of faster algorithms such as SumBasic, while estimating cutting edge methods such as topic modeling. Specifically, SumBasic is generally known for being efficient and generating relatively good results that can be used as a baseline. On the other hand, LDA and topic modeling approaches are known for being slow but providing stronger summaries by dealing with complex natural language problems such as polysemy. Furthermore, LDA systems are in theory more robust, able to isolate general noise from key terms for the topic being discussed, by using background models. However, in my work I found both traditional frequency based work and topic modeling systems to perform poorly on noisy comment data. Complex part of speech based aspect extraction systems performed better, able to isolate both important and well formed sentences. However, these systems can lack diversity since they focus only on opinions that use sentimental keywords.

After taking these lessons into account I created the KLSum+ algorithm. In Section 4.4.2 I show that this system performs much faster than alternatives. Upon further analysis I find that SumBasic and Random are slower on news comment data as they select shorter sentences, which translates into more iterations before reaching the 250 word limit. Thus algorithms that choose a few high information content sentences will generally execute faster than algorithms that choose hundreds of high scoring sentences.

In Section 4.4.3 I attempt to evaluate KLSum+ by comparing it to other prevalent algorithms in the field. I generate summaries for the DUC 2007 data set and use the ROUGE toolkit to evaluate my summaries against human judgments provided as part of the DUC 2007 task. In this work I show that KLSum+ is able to select comparible summaries to much more complex summarization models. This work also shows that KLSum+ outperforms the highly related SumBasic algorithm, which shows the advantage of using a background model. I reason that systems that use generic *tf-idf* tables and stop words can be penalized, and thus domain specific background models are preferred. This result is generally applicable to other domains where machine learning is used.

However, even I found it surprising that KLSum+ performed on par with more advanced clustering approaches like MEAD, SyntaxSum, and LDA. This may indicate that the design of KLSum+ stumbled upon a quality of human readers that other algorithms have estimated indirectly. This may either be the human minds ability to quickly filter out content that is generic to all text, or a general preference for selecting multiple related

sentences before diversifying.

Finally, in Section 4.4.4 I quickly overview some generated summaries. From the SumBasic, LDA, and Random summaries I am able to realize that the noise within a comment corpus can quickly foil sentence extraction. LexRank and Sentiment based systems indirectly forced criteria that ensured longer sentences and greatly improved readability. However, these criteria also lead to the selection of run on sentences or summaries that do not contain a cohesive idea. Thus, by selecting multiple sentence snippets, and directly optimizing for snippet length, KLSum+ is able to approximate these advantages quickly while providing short information dense snippets.

The snippets generated by KLSum+ were also shown to become diverse relatively quickly. By tuning parameters the algorithm allows a quick divergence of aspects and allows readers to quickly get an overview of a corpus of noisy data. These summaries not only showed multiple facets of similar stories, but also showed aspects that were entirely independent of a sentiment lexicon. Instead, the opinions of users were highlighted as they related to different aspects of a discussed article. This separation from a static lexicon both increases relevance and decreases the overhead with applying KLSum+ to a new domain. However, it should be noted that one should not dismiss the power of merging algorithms. For example using KLSum+ on terms harvested from the sentiment based aspect miner may lead to interesting and opinionated summaries.

For more insight I deployed a web server at www.gobaan.com:8000 which allows users to explore KLSum+ summaries and context over all the topics I summarized. A snapshot from this tool is presented in Figure 4.6.

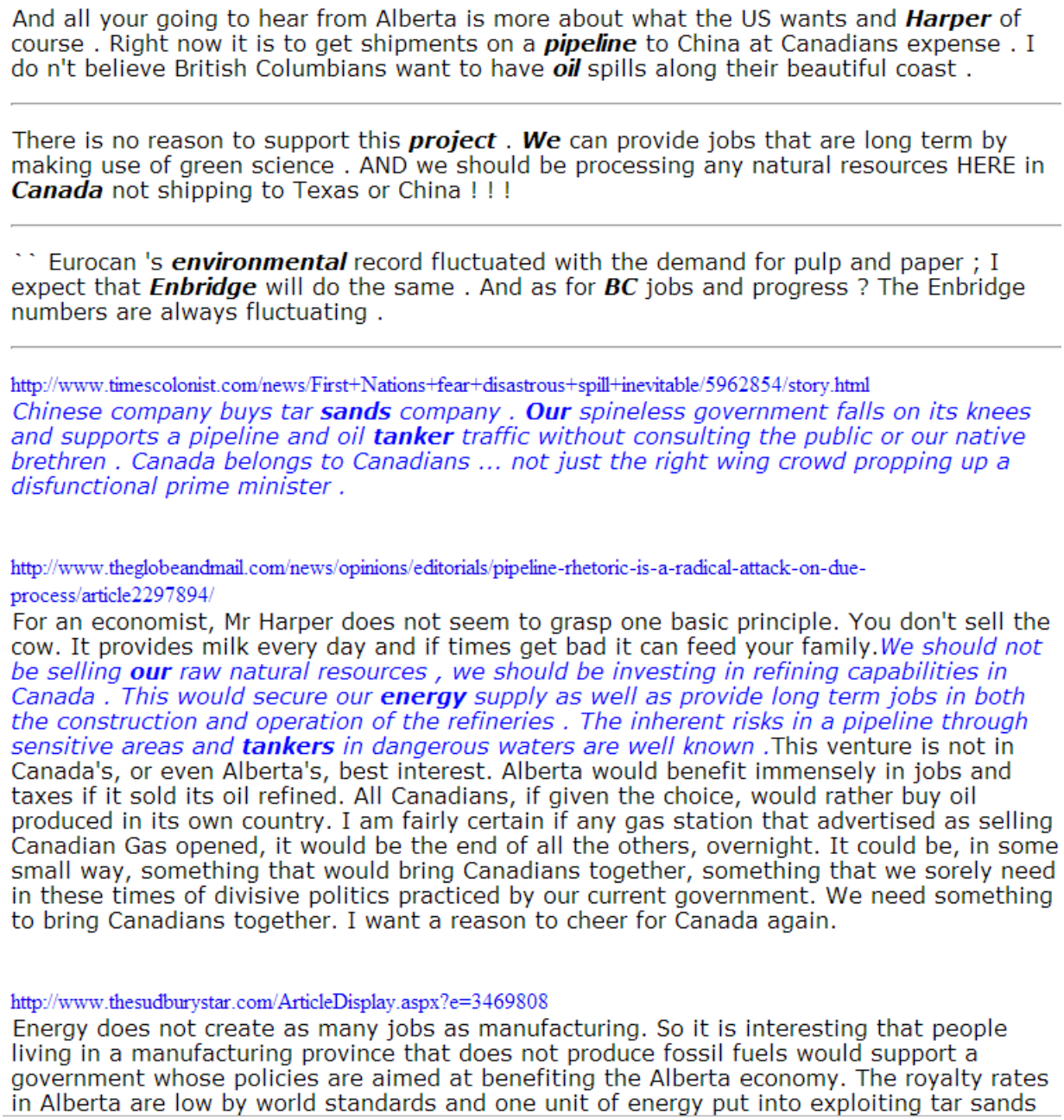


Figure 4.6: Snapshot from www.gobaan.com:8000, a tool I created to show context behind KLSum+ Summaries

5

Conclusion

We don't have better algorithms. We just have more data.

– Peter Norvig, *Google's Zeitgeist*

In this thesis I present my comment crawling framework designed to extract news comments from across the web. In the age of information I discuss the noise in news comments and decide that I can gain great insight into various communities by analyzing the discourse that they generate. I show that this task is not only interesting, but difficult and unsolved by current research.

In Chapter 3 I take steps towards a generalized solution that can be extended to extract comments from any domain across the web. Furthermore by using careful design I avoid the trap of becoming too dependent on with machine learning. Instead I create a system that works with users to quickly generate rules for crawling and parsing comments from various domains. By employing user feedback I am able to create high fidelity parsers, and by autogenerating code I reduce the difficulty such that only the most rudimentary knowledge of computer science is needed to generate parsers for new domains.

I employ my tool to rapidly create parsers for more than 40 domains, and extract a few million comments on various topics discussed in the last decade. Once we have this rich set of data, we attempt to harness some useful information from it. Using this data I not only evaluate current methods for automatic summarization, but generate a new system that is able to quickly extract diverse opinions from the large corpus of text. My system, KLSum+, is an algorithm that scales better than all the alternatives evaluated, while performing on par with more complicated approaches on both small and large data sets. By doing so I have set up a simple baseline for scalable multidocument summarization.

The end result of my work is two interesting open source projects that can stand independently as useful products. In the future researchers may use my comment extraction system to analyze other trends in social media. This corpus provides a noisy and rich stream of data that is publicly available and freely accessible unlike many other social media document sets. This is useful for academic circles where results must be reproducible. Furthermore, the ease of extension and isolation of parsers means my framework can be used for various other domains and website types. I hope that researchers will be able to work together to create larger collections of parsers for domains in various languages and target audiences.

As researchers focus more on using complicated semantic methods to force context into summaries I show that instead a reversion to simpler frequency based methods can still have strong results. Furthermore I show that these frequency based methods need not dismiss context, instead I can use domain specific knowledge to determine the importance of terms. By using domain specific knowledge I eliminate the need to attach myself to stop word that can be attached to a single language, and often become stale as language evolves.

However, my work is not without flaws. As websites change over time my system requires some maintenance in order to generate valid summaries. Furthermore some portals maintain dozens of versions of the site, with older articles using older versions. Future work on the crawler may wish to use some additional heuristics to discover when websites adapt and either notify users or automatically adapt with them. Alternatively, I currently rely on user navigation to determine the important components of websites. However, with some additional application of machine learning I may be able to provide the users options and then use a system such as Amazons Mechanical Turk to rapidly adapt with them.

On the other hand my summarization work, despite its strong results, was not formally evaluated on a large corpus. I briefly look at the summaries and decide that they are higher quality, however I lack any form of human judgment. Future work should likely focus on performing a direct pairwise comparison of summary systems on my large data set. Alternatively, there may be some interesting benefits of generating ground truth for the large corpus, despite the difficulty. The metrics I employed for evaluation also lacked any direct measure of linguistic quality. Thus as summarization research progresses, methods of automatic evaluation may become standardized and thus a stronger evaluation of KLSum+ can occur.

Future work can also focus on the uses of my data set. The data I harvest presents a rich collection and although I just performed a case study on summarization, it is easy to envision other possible uses. For example, some work into the political biases of individ-

uals mattering on news portals, or the effectiveness of marketing campaigns could be an interesting venue to pursue.

Appendix A: HTTP Request Header


```
{
  "statusCode":200,
  "method":"GET",
  "url":"http://widgets.outbrain.com/nanoWidget/3rd/comScore/comScore.htm",
  "bytes":445,
  "start":"2012-10-21T04:34:48.370-0400",
  "end":"2012-10-21T04:34:48.448-0400",
  "timeInMillis":78,
  "requestHeaders":[{
      "name":"Host",
      "value":"widgets.outbrain.com"
    },{
      "name":"User-Agent",
      "value":"Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:16.0) Gecko/20100101 Firefox/16.0"
    },{
      "name":"Accept",
      "value":"text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
    },{
      "name":"Accept-Language",
      "value":"en-US,en;q=0.5"
    },{
      "name":"Accept-Encoding",
      "value":"gzip, deflate"
    },{
      "name":"Proxy-Connection",
      "value":"keep-alive"
    },{
      "name":"Referer",
      "value":"http://abcnews.go.com/US/chicago-gang-life-gang-members-talks-life-streets/story?id=17499354"
    },{
```

```
        "name":"Cookie",
        "value":"obuid=7521bf81-1963-4f2e-8d93-92802b169405; tick=1350808488142; _lvs2=\"/+wQ+b2JMg4=\"; _lvd2=\"5qZP9A8rYKJV4ZyVLbsEwg==\"; _rcc2=\"53VdlhoVktWl+Ov6ordflA==\"; _fcap_CAM3=\"AEYAQwBBAFAAAAAFAAARdD+AAAAAABbkP4AAAAAADn0/gAAAAAAeVT+AAAAAABh5P4AAAA==\"; recs-977d59a416c537e8606e93fe3ece49ed=\"QIbgLMP7cX4wqtnU78cnOVchRXDrYLtUJjnmbTCfdojDpZ+o/HP6tV6SxhHqjHa2G9Pz6zrSMP0=\""
    }],
    "responseHeaders":[{
        "name":"Date",
        "value":"Sun, 21 Oct 2012 08:34:48 GMT"
      },{
        "name":"Server",
        "value":"Apache"
      },{
        "name":"Cache-Control",
        "value":"private, max-age=2592000"
      },{
        "name":"Expires",
        "value":"Tue, 20 Nov 2012 08:34:48 GMT"
      },{
        "name":"Content-Type",
        "value":"text/html; charset=UTF-8"
      },{
        "name":"Content-Length",
        "value":"445"
      },{
        "name":"Accept-Ranges",
        "value":"bytes"
      },{
        "name":"Via",
        "value":"1.1 (jetty)"
      },{
        "name":"Vary",
        "value":"Accept-Encoding"
      },{
        "name":"Content-Encoding",
        "value":"gzip"
      },{
        "name":"Age",
        "value":"0"
    }]
}
```


Listing A.1: Example header from capturing network traffic using Selenium

Appendix B: Data Description

Domain	Comments	Domain	Comments
americanthinker.com	162815	opinion.financialpost.com	6507
aptn.ca	389	politico.com	55807
arstechnica.com	2754	rawstory.com	372351
calgaryherald.com	10605	reuters.com	61157
canadianbusiness.com	1170	seattletimes.nwsource.com	62807
cbsnews.com	18308	sfgate.com	37271
chicagotribune.com	6766	straight.com	22767
cnn.com	297273	talkingpointsmemo.com	12993
csmonitor.com	82674	terracestandard.com	269
ctv.ca	103622	theatlantic.com	54364
economist.com	25405	theglobeandmail.com	105995
edmontonjournal.com	12203	thehill.com	274598
foxnews.com	529033	thestar.com	32694
fullcomment.nationalpost.com	85246	thesudburystar.com	15573
huffingtonpost.com	299812	thisislondon.co.uk	61234
latimes.com	24107	timescolonist.com	7630
news.cnet.com	139833	upi.com	11326
newsobserver.com	30089	vancouversun.com	12731
newsvine.com	37829	voices.yahoo.com	18194
npr.org	44243	washingtonpost.com	905225
nytimes.com	6990	winnipegsun.com	13381
online.wsj.com	60576		

Table B.1: Number of comments collected for each site

Topic	Comments	Topic	Comments
acta	3726	israel	535894
alternative energy	561529	israel bombing iran	115347
anonymous	49361	joe biden	144273
apple	257607	john kerry	31698
arab spring	322306	john mccain	401071
bankruptcy	256040	justin bieber	8499
barack obama	365130	larry page	236308
batman	64025	libya	418869
bp	129210	mark zuckerberg	89798
cars	3311	nasa	121650
catholic	276784	natural gas	307010
charlie sheen	80123	news	456334
climate change	252412	north korea	215140
creationism	38003	northern gateway pipeline	47368
dark knight rises	152404	nuclear	436181
economy	595959	nypd	58949
evolution	217302	occupy wall street	355310
facebook	278223	oil prices	383200
gay marriage	345867	olympics	94811
george bush	97100	piracy	174471
global warming	68928	polling accuracy	23927
gop primaries	100323	riaa	13207
gop primary	802635	rick santorum	924523
greece	43871	sarah palin	337506
gun control	333339	sopa	62136
health care	708449	steve jobs	331210
hillary clinton	313082	stock act	465279
intelligent design	79798	suicide	249513
iran	442938	the pirate bay	101537
iraq	433894	wall street	548803

Table B.2: Number of comments collected for each topic

References

[1] Rachit Arora and Balaraman Ravindran. Latent Dirichlet Allocation based Multi-document Summarization. In *Proceedings of the 2nd Workshop on Analytics for Noisy Unstructured Text Data*, pages 91–97, New York, NY, USA, 2008.

[2] Chandra Avinash. Lead of a news story. `http://journalism20.nuvvo.com/lesson/7587-lead-of-a-news-story`. Online; Accessed 08-Feb-2013.

[3] Stefano Baccianella, Andrea Esuli, and Fabrizio Sebastiani. SentiWordNet 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Bente Maegaard, Joseph Mariani, Jan Odijk, Stelios Piperidis, Mike Rosner, and Daniel Tapias, editors, *Proceedings of the 7th International Conference on Language Resources and Evaluation)*. European Language Resources Association, 2010.

[4] Alexandra Balahur, Elena Lloret, Ester Boldrini, Andrés Montoyo, Manuel Palomar, and Patricio Martínez-Barco. Summarizing Threads in Blogs using Opinion Polarity. In *Proceedings of the Workshop on Events in Emerging Text Types*, pages 23–31, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[5] Sasha Blair-goldensohn, Tyler Neylon, Kerry Hannan, George A. Reis, Ryan Mcdonald, and Jeff Reynar. Building a Sentiment Summarizer for Local Service Reviews. In *In NLP in the Information Explosion Era*, 2008.

[6] Dustin Boswell. Distributed High-performance Web Crawlers: A Survey of The State of the Art, 2003.

[7] Lori Buckland and Hoa Dang. DUC 2007: Documents, Tasks, and Measures. `http://duc.nist.gov/duc2007/tasks.html`. Accessed: 2013-05-14.

[8] Deepayan Chakrabarti and Kunal Punera. Event Summarization using Tweets. In *Proceedings of the 5th International Association for the Advancement of Artificial Intelligence Conference on Weblogs and Social Media*, 2011.

[9] Tim Cuthbertson. Python Readability. `https://github.com/gfxmonk/python-readability/blob/master/readability/readability.py`. Accessed: 2012-02-18.

[10] William M. Darling and Fei Song. Probabilistic Document Modeling for Syntax Removal in Text Summarization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers - Volume 2*, pages 642–647. Association for Computational Linguistics, 2011.

[11] Hal Daumé and D. Marcu. Bayesian Summarization at DUC and a Suggestion for Extrinsic Evaluation. In *Document Understanding Conference*, 2005.

[12] Mark Dredze, Hanna M. Wallach, Danny Puller, and Fernando Pereira. Generating Summary Keywords for Emails using Topics. In *Proceedings of the 13th International Conference on Intelligent User Interfaces*, pages 199–206, 2008.

[13] Cristian Duda, Gianni Frey, Donald Kossmann, Reto Matter, and Chong Zhou. AJAX Crawl: Making AJAX Applications Searchable. In *Proceedings of the 2009 Institute of Electrical and Electronics Engineers International Conference on Data Engineering*, pages 78–89, Washington, DC, USA, 2009. Institute of Electrical and Electronics Engineers Computer Society.

[14] Susan Dumais and Hao Chen. Hierarchical Classification of Web Content. In *Proceedings of the 23rd Annual International Association for Computing Machinery Special Interest Group on Information Retrieval Conference on Research and Development in Information Retrieval*, pages 256–263, New York, NY, USA, 2000.

[15] Günes Erkan and Dragomir R. Radev. LexRank: Graph-Based Lexical Centrality as Salience in Text Summarization. *Journal of Artificial Intelligence Research*, 22(1):457–479, 2004.

[16] JanFrederik Forst, Anastasios Tombros, and Thomas Roelleke. Less Is More: Maximal Marginal Relevance as a Summarisation Feature. In Leif Azzopardi, Gabriella Kazai, Stephen Robertson, Stefan Rger, Milad Shokouhi, Dawei Song, and Emine Yilmaz, editors, *Advances in Information Retrieval Theory*, volume 5766 of *Lecture Notes in Computer Science*, pages 350–353. Springer Berlin Heidelberg, 2009.

[17] Jesse James Garrett. Ajax: A New Approach to Web Applications. http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications. Accessed: 2013-01-09.

[18] Google. Making AJAX Applications Crawlable. https://developers.google.com/webmasters/ajax-crawling/docs/html-snapshot. Accessed: 2013-01-09.

[19] GreenBook. Study Finds Marketers Struggle with the Big Data and Digital Tools of Today. . Online; Accessed 08-Feb-2013.

[20] Yan Guo, Kui Li, Kai Zhang, and Gang Zhang. Board Forum Crawling: A Web Crawling Method for Web Forum. In *Proceedings of the 2006 Association for Computing Machinery International Conference on Web Intelligence*, pages 745–748, Washington, DC, USA, 2006. Institute of Electrical and Electronics Engineers Computer Society.

[21] Aria Haghighi and Lucy Vanderwende. Exploring Content Models for Multi-document Summarization. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 362–370. Association for Computational Linguistics, 2009.

[22] Donna Harman and Paul Over. The Effects of Human Variation in DUC Summarization Evaluation. In *Proceedings of the Association for Computational Linguistics-04 Workshop: Text Summarization Branches Out*, pages 10–17, 2004.

[23] Minqing Hu and Bing Liu. Mining and Summarizing Customer Reviews. In *Proceedings of the 10th Association of Computing Machinery Knowledge Discovery and Data Mining International Conference*, pages 168–177, 2004.

[24] Lei Huang and Yanxiang He. CorrRank: Update Summarization Based on Topic Correlation Analysis. In De-Shuang Huang, Xiang Zhang, CarlosAlberto Reyes Garca, and Lei Zhang, editors, *Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence*, volume 6216 of *Lecture Notes in Computer Science*, pages 641–648. Springer Berlin Heidelberg, 2010.

[25] D. Inouye and J.K. Kalita. Comparing Twitter Summarization Algorithms for Multiple Post Summaries. In *Privacy, Security, Risk and Trust, 2011 Institute of Electrical and Electronics Engineers 3rd International Conference on Social Computing (socialcom)*, pages 298–306, 2011.

[26] Michael Karlsson. Flourishing but restrained. *Journalism Practice*, 5(1):6884, 2011.

[27] Saleem Kassim. Twitter Revolution: How the Arab Spring Was Helped By Social Media. `www.policymic.com/articles/10642/twitter-revolution-how-the-arab-spring-was-helped-by-social-media/`. Online; Accessed 08-Feb-2013.

[28] Alistair Kennedy, Anna Kazantseva, Diana Inkpen, and Stan Szpakowicz. Getting Emotional about News Summarization. In Leila Kosseim and Diana Inkpen, editors, *Advances in Artificial Intelligence*, volume 7310 of *Lecture Notes in Computer Science*, pages 121–132. Springer Berlin Heidelberg, 2012.

[29] Khairullah Khan and Baharum B. Baharudin. Analysis of Syntactic patterns for Identification of Features from Unstructured Reviews. In *Intelligent and Advanced Systems*, volume 1, pages 165–169, 2012.

[30] Christian Kohlschütter, Peter Fankhauser, and Wolfgang Nejdl. Boilerplate Detection using Shallow Text Features. In *Proceedings of the 3rd Association for Computing Machinery International Conference on Web Search and Data Mining*, pages 441–450, New York, NY, USA, 2010.

[31] L. W. Ku, Y. T. Liang, and H. H. Chen. Opinion Extraction, Summarization and Tracking in News and Blog Corpora. In *Proceedings of Association for the Advancement of Artificial Intelligence 2006 Spring Symposium on Computational Approaches to Analyzing Weblogs*, 2006.

[32] Niraj Kumar, Kannan Srinathan, and Vasudeva Varma. An Effective Approach for AESOP and Guided Summarization Task.

[33] Sauce Labs. Selenium - Web Browser Automation. `http://docs.seleniumhq.org/`. Accessed: 2013-04-15.

[34] C. Y. Lin. Looking for a Few Good Metrics: Automatic Summarization Evaluation - How Many Samples are Enough? In *Proceedings of the NTCIR Workshop 4*, 2004.

[35] Chin-Yew Lin. ROUGE: A Package for Automatic Evaluation of Summaries. In Stan Szpakowicz Marie-Francine Moens, editor, *Text Summarization Branches Out: Proceedings of the Association for Computational Linguistics-04 Workshop*, pages 74–81. Association for Computational Linguistics, 2004.

[36] Chin-Yew Lin and Eduard Hovy. Automatic Evaluation of Summaries using N-gram Co-occurrence Statistics. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, pages 71–78. Association for Computational Linguistics, 2003.

[37] Jayant Madhavan, David Ko, Lucja Kot, Vignesh Ganapathy, Alex Rasmussen, and Alon Halevy. Google's Deep Web crawl. *Proceedings Very Large Data Bases Endowment*, 1(2):1241–1252, August 2008.

[38] MEAD. `http://www.summarization.com/mead/`, February 2013. Online; Accessed 08-Feb-2013.

[39] Rada Mihalcea and Paul Tarau. TextRank: Bringing Order into Texts. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2004.

[40] Ani Nenkova and Lucy Vanderwende. The Impact of Frequency on Summarization. *Microsoft Research, Redmond, Washington, Technical Report MSR-TR-2005-101*, 2005.

[41] Jeffrey Nichols, Jalal Mahmud, and Clemens Drews. Summarizing Sporting Events using Twitter. In *Proceedings of the 2012 Association for Computing Machinery International Conference on Intelligent User Interfaces*, pages 189–198, 2012.

[42] Alexandros Ntoulas, Petros Zerfos, and Junghoo Cho. Downloading Hidden Web Content. Technical report, UCLA Computer Science, 2004.

[43] Jolie O'Dell. For the First Time, More People Get News Online Than From Newspapers. `http://mashable.com/2011/03/14/online-versus-newspaper-news/`. Online; Accessed 08-Feb-2013.

[44] U.S. Department of Commerce. TAC 2011 AESOP Task Guidelines. `http://www.nist.gov/tac/2011/Summarization/AESOP.2011.guidelines.html`. Accessed: 2012-03-10.

[45] Christopher Olston and Marc Najork. Web Crawling. *Founding Trends Information Retrieval*, 4(3):175–246, March 2010.

[46] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford Digital Library Technologies Project, 1998.

[47] Sandeep Pandey and Christopher Olston. User-centric Web Crawling. In *Proceedings of the 14th International Conference on World Wide Web*, pages 401–411, New York, NY, USA, 2005.

[48] PearAnalytics. Twitter Study August 2009. `http://www.pearanalytics.com/wp-content/uploads/2012/12/Twitter-Study-August-2009.pdf`. Online; Accessed 08-Feb-2013.

[49] L. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the Institute of Electrical and Electronics Engineers*, 77(2):257–286, 1989.

[50] Dragomir R. Radev, Hongyan Jing, and Malgorzata Budzikowska. Centroid-Based Summarization of Multiple Documents: Sentence Extraction, Utility-Based Evaluation, and User Studies. In *Proceedings of the 2000 North American Chapter of the Association for Computational Linguistics Workshop on Automatic Summarization - Volume 4*, pages 21–30, 2000.

[51] Dragomir R. Radev, Hongyan Jing, Małgorzata Styś, and Daniel Tam. Centroid-based Summarization of Multiple Documents. *Information Processing Management*, 40(6):919–938, 2004.

[52] Readability. `http://www.readability.com`, February 2013. Online; Accessed 08-Feb-2013.

[53] Leonard Richardson. BeautifulSoup. `http://www.crummy.com/software/BeautifulSoup/`. Online; Accessed 08-Feb-2013.

[54] Sentiwordnet. `http://sentiwordnet.isti.cnr.it/`. Online; Accessed 08-Feb-2013.

[55] Beaux Sharifi, Mark-Anthony Hutton, and Jugal Kalita. Summarizing Microblogs Automatically. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 685–688. Association for Computational Linguistics, 2010.

[56] Stephen Smart. Why B.C., Alberta are Ending their Pipeline Standoff. `http://www.cbc.ca/news/canada/why-b-c-alberta-are-ending-their-pipeline-standoff-1.2126176`. Online; Accessed 03-Aug-2013.

[57] Hiroya Takamura, Hikaru Yokono, and Manabu Okumura. Summarizing a document stream. In Paul Clough, Colum Foley, Cathal Gurrin, GarethJ.F. Jones, Wessel Kraaij, Hyowon Lee, and Vanessa Mudoch, editors, *Advances in Information Retrieval*, volume 6611 of *Lecture Notes in Computer Science*, pages 177–188. Springer Berlin Heidelberg, 2011.

[58] Jie Tang, Limin Yao, and Dewei Chen. Multi-topic based Query-oriented Summarization. In *Society for Industrial and Applied Mathematics International Conference Data Mining*, 2009.

[59] Michael Tung. Diffbot: Identify and Extract from Any Web page. `www.diffbot.com`. Accessed: 2013-02-18.

[60] Junfeng Wang, Chun Chen, Can Wang, Jian Pei, Jiajun Bu, Ziyu Guan, and Wei Vivian Zhang. Can We Learn a Template-independent Wrapper for News Article Extraction from a Single Training Site? In *Proceedings of the 15th Association of Computing Machinery Knowledge Discovery and Data Mining International Conference*, pages 1345–1354, New York, NY, USA, 2009.

[61] Junfeng Wang, Xiaofei He, Can Wang, Jian Pei, Jiajun Bu, Chun Chen, Ziyu Guan, and Gang Lu. News Article Extraction with Template-independent Wrapper. In *Proceedings of the 18th International Conference on World Wide Web*, pages 1085–1086, New York, NY, USA, 2009.

[62] Zach Weiner. How Internet Arguments Work. `http://www.smbc-comics.com/?id=2939`. Online; Accessed 03-Aug-2013.

[63] Brian Wilson. MAMA: Key findings. `http://dev.opera.com/articles/view/mama-key-findings/`. Online; Accessed 08-Feb-2013.

[64] Wen-tau Yih, Joshua Goodman, Lucy Vanderwende, and Hisami Suzuki. Multi-document Summarization by Maximizing Informative Content-Words. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, pages 1776–1782. Morgan Kaufmann Publishers Inc., 2007.

[65] Xiaohua Zhou, Xiaodan Zhang, and Xiaohua Hu. Dragon Toolkit: Incorporating Auto-Learned Semantic Knowledge into Large-Scale Text Retrieval and Mining. In *Proceedings of the 19th Institute of Electrical and Electronics Engineers International Conference on Tools with Artificial Intelligence - Volume 02*, pages 197–201. Institute of Electrical and Electronics Engineers Computer Society, 2007.

[66] Rich Ziade. A Free Web & Mobile App for Reading Comfortably. www.readability.com. Accessed: 2013-02-18.