An argumentation-based approach to summarizing discussions

Charlie Egan

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Bachelor of Science
of the
University of Aberdeen.

Department of Computing Science

2016

Declaration

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Signed:

Date: 2016

Abstract

With an ever increasing number of internet users, online communities are hosting an ever growing number of discussions. This user generated content contains millions of statements, opinions and ideas. Current systems for discovery of such information are playing catch-up, and in the meantime much of this resource is all but inaccessible.

In this paper an approach for summarizing such information is proposed. Our approach is based on the idea of 'point extraction', where a point is a verb and it's arguments. The approach uses both dependency parse information and verb case frames to identify and extract valid points.

We test the approach using online debates and evaluate our summaries against a high-performing, extractive summarizer. We found that we were able to improve significantly on this baseline by showing that points make better summary content units than sentences. We also show that analysis enabled by representing a discussion as points is valuable to the task of automatic summarization.

Acknowledgements

I would like to thank both Dr. Adam Wyner and Dr. Advaith Siddharthan for all their support in helping make the project a success as well as funding the project evaluation. I also thank my parents for their assistance designing my user evaluation. Finally, I'd like to acknowledge my support from the department over the course of my degree in preparation for this final project.

Contents

Chapter 1

Introduction

More people are online than ever before. Comment threads and discussion forums allow us to spend spare time participating as contributors - rather than just consuming content. From the latest blockbuster title to yesterday's celebrity misdemeanor, there's an online conversation already well underway. These discussions, where statements are encoded in natural language, represent a large untapped resource of ideas. A higher-level view of a discussion would be useful and interesting to many. Online retailers might analyze product reviews; social scientists could gain valuable insight on social media debate; and marketers could better understand the discussions of their customers.

This project explores an approach for summarizing such discussions. At the core of the approach is the notion of a 'point' - a short refined statement, extracted as a verb and its syntactic arguments. We model a user's post as a series of arguments, where an argument is a point made in their post relevant to a topic. These points are clustered and analyzed to give a summary of common arguments made in the discussion. To test our approach, we ran an evaluation using summaries generated from online political debates [36].

1.1 Motivation

Text summarization is a well established task in the field of Natural Language Processing. Summarization sub-tasks such as sentence extraction and compression, where text is selected and removed to arrive at a summary, have become commonplace due to the complexities introduced by abstractive methods. Extractive methods have become largely statistical using Naive-Bayes [16] approaches and, more recently even Neural Networks [34].

Argumentation mining, a newer area of study, has the aim of identifying argumentative discourse structure in text. Argumentation mining has been used in the processing of formal documents (where arguments are often stated more explicitly) such as parliamentary records [27] and legal documents [23]. However, argumentation mining has also more recently been applied to more informal texts [28]. Applications of argument mining on informal texts, coupled with summarization, encapsulates much of the novelty of this project.

Using 'points' as a data structure was adopted from a previous project in the department. This used a similar concept in a system with a focus on stance classification.

While this implemented point identification, it was limited by its point definition: a verb, subject and object. This lead to disfluencies caused by the absence of important syntactic arguments to the verb. While the tool was capable of linking contrasting points, this was based only on the presence of negation terminology.

Our project was based on this broad concept of a point, as well as the idea that they could be used to build a high-level summary of a discussion when considered to represent basic units of informal argumentation. Argumentation mining, as referenced as part of this project, focuses on how points, referencing topics, can be compared at the level of the whole discussion — rather than the intricacies of the arguments of individual posts. News article comment sections; forum threads; film & product reviews and even extended email conversations are all candidate applications for such analysis. However, the focus of this work is on political discussions.

1.2 Objectives

The overall objective for the project was to implement a means of summarizing large discussions; in particular, discussions that take place on social media with many participants. We wanted to find a means of representing the key concepts under discussion. Which ideas were related? Or contradictory? For this task, we chose to use points; and the relationships between them; to form a condensed representation of the discussion.

Continuing the work of a previous project that used a naive implementation of a point (subject, verb, object), we wanted to build on this and improve on the extraction of points from text. Fundamentally, this meant extracting more of the verb's arguments and related tokens - rather than only the subject and object. Take the following sentence:

"*I don't think so, an unborn child (however old) is not yet a human being.*"

From this, using the subject and object syntactic arguments of *is*, gives: `child.subject be.verb human.object` (discussed in detail later). This is useful, a structure like this can be linked with a point extracted from another sentence like: "*So you say: children are not complete humans until birth?*" (child is the lemma of children). However, information is lost in this abstraction and it is poorly suited to presentation. Our objective, with regards to better extracting points, was to *additionally* extract the sub-sentence context in which a point was made. Continuing this example, the following string would be a 'good' extract:

"*an unborn child is not yet a human being*"

This is almost half the length of the original sentence, but still represents the core idea expressed around being human — and is still easily readable. Using this representation as a means of compression appears to be a novel approach. As part of generating summaries, we wanted to go beyond just extracting points and investigate relationships between them. Counter points pairs — points that represent opposing ideas — and co-occurring points — points commonly raised together by multiple users — were of particular interest for use as summary content.

The project's success is evaluated with respect to these objectives.

Chapter 2

Background & Related Work

2.1 Background

This project builds on existing technologies and research. This section is intended to give an overview of past studies and other work relevant to the project.

2.1.1 Automatic Summarization

"A summary can be loosely defined as a text that is produced from one or more texts, that conveys important information in the original text(s), and that is no longer than half of the original text(s) and usually significantly less than that." [29]

Generally, approaches to automatic summarization could be categorized into either *extractive* or *abstractive* methods. Extractive summaries are constructed reusing content found in the source text while abstractive summaries are created by performing some form of analysis on the source text; before generating new content for use in the summary. One might also group summarization tasks based on the nature of the source text - approaches tend to summarize either single or multiple documents. This project is more closely related to the summarization of multiple documents and applies both elements of extractive and abstractive summarization.

Multi-document Summarization

Summarizing text made up of documents from different authors poses an interesting challenge as it typically introduces repetition and contradictions. This makes summarization selection tasks such as extraction more complex.

Earlier work on multi-document summarization suggested the task was going to require an abstractive approach; not being able to connect information between documents would lead to incoherent and repetitive summaries [21]. An alternative approach clustered paragraphs on the same topic from multiple documents and then use *Natural Language Generation* to combine phrases from paragraphs into a coherent summary [21]. A similar approach creates clusters by parsing sentences and using predicate-argument structures to link phrases [1] making it somewhat related to our approach. Interestingly, while both methods rely on *Natural Language Generation*, neither use a semantic representation (which is commonly used in NLG tasks).

More recently, MEAD, a topic-cluster centroid, multi-document summarization tool has been able to produce quality summaries without abstraction and generation [30]. Compared to previous abstractive systems such as SUMMONS that relied on templates

[20], MEAD was more generally applicable. Both these projects, and indeed most work on the summarization of multi-document sources, has focused on news articles. While there are parallels between multi-document news summarization and the summarization of discussions, fundamentally they are different tasks.

Sentence Compression & Content Unit Size

While sentences are the default content unit used in extractive summarization, often smaller units are preferable. Sentences found in real documents and discussions can make a number of statements; each of which can be useful independently when creating a summary. Sentences also often include surplus information such as references to other parts of the source text. In this case sentences can be compressed statistically, in a similar way to how documents are in extractive summarization. Both noisy channel and decision based models have been used on parsed sentences for the task of sentence compression [15]. However, our approach for sentence compression, where points are extracted around a verb, does not appear to have been previously documented.

Selecting the content unit for a summarization task is challenging. One might think that sentences could be relied upon as a coherent content unit; however, poor grammar and unresolved co-referring expressions make this assumption unreliable [38].

2.1.2 Argumentation Mining

Argumentation mining is a task that involves the identification of argument components in text — such as premises and conclusions. The task often involves fitting these components into a template or known pattern to enable some form of reasoning. [27]

The relatively new research area — combining ideas from natural language processing, information extraction and argumentation theory — argumentation mining is based largely on discourse theory. While argumentation mining can be applied to structured text, ultimately mining of free text is the goal. This means that argumentation tools often need to model concepts from linguistic research such as rhetoric structure theory [19] and local coherence centering [37] to identify structure.

Identifying Argumentative Structures in Text

When identifying argumentative structures in text, the first step is often to classify phrases that are part of an argument. One option is to define rules for argumentative sentences or clauses; however, this can also be approached as a classification task and statistical methods such naive Bayes have also been used [27]. The result of this stage is a collection of unrelated clauses potentially spanning many arguments present in the source text. The next stage in such analysis is often to group the identified units together into separate arguments. This grouping can be based on the similarity of the unit's content — such as the presence of keywords.

From this point, it is possible to perform more in depth analysis on argumentative units. This may include classifying premises & conclusions or finding relationships between the arguments made in different documents by reasoning on the argument's structure. However, this is fundamentally different from the work presented in this project. Starting from a collection of points, our representation of argumentative phrases, it is possible to analyze arguments made in the discussion as a whole. Rather than modeling

individual arguments, we look for common relationships in groups of points and use these to summarize many separate arguments made in the discussion.

Also related to this project is the topic of opinion mining. This is an approach often applied to user reviews, posts discussing a product or service. Early work in the area focused on the problem as a classification task, categorizing posts based on their sentiment bearing terms [35]. More recently there has been a greater focus on aspect-based approaches that attempt to isolate sentiment terminology to specific aspects referenced in the source text [13].

Our approach takes ideas from both areas. By looking for argument statements, and how these relate to topics, we build a summary of the discussion.

2.1.3 Prerequisite Work

Our approach builds on and uses prior research in several different areas.

Topic Modeling

Topic modeling covers a set of statistical modeling techniques used to identify and group topics in text. Fundamentally, this involves identifying words that are relatively more common in one document vs. a larger collection of documents. Documents often contain multiple topic words that need to be grouped into more conceptual topics. One approach to this is is *Latent Dirichlet Allocation* (LDA), this implements a "three-level hierarchical Bayesian model" [2] and is the implementation used for topic modeling in this project.

Dependency Parsers

Parsing text for dependencies involves building upon a phrasal parse to extract relations between tokens in text - dependencies are identified using rules applied to phrase structure trees. This project makes use of the dependency parser included as part of the Stanford CoreNLP framework [9]. This is based on their probabilistic, context-free grammar, phrasal parser [14]. The rules used in the Stanford dependency parser are based on *Universal Stanford Dependencies* [8].

Querying Dependency Parses

To make use of a dependency parse, one needs a means of querying the graph. *Semgrex* is one such approach. This defines a regular expression style syntax for formulating queries on tokens and relations in a parse [6]. However, more fundamentally, the task involves querying nodes and edges in a directional graph. For this project we opted to use *Cypher*[1], the graph query language implemented by the *Neo4j*[2] database. This was chosen for the querying of dependency parses as it allowed many separate graphs to be queried in parallel for the same pattern using a declarative syntax.

2.2 Related Work

Having covered the technologies and areas of research that underpin this project I will now discuss related work that comes closer to combining these two areas, as our project does.

[1] http://neo4j.com/docs/stable/cypher-introduction.html
[2] http://neo4j.com/

While most summarization research has been in the newswire domain, some work has been done to summarize more conversational texts. This differs from speech summarization as source content is typed rather than spoken, removing many additional difficulties. Documented business conversations such as email threads and meeting minutes are a common use case. In email threads (collective message summarization), a common approach is to identify question and answer pairs [33, 32, 5]. Questions can be identified using a classifier trained on a corpus that annotates speech acts [32]. The quotation formatting used in emails can also provide question-answer pairs - one approach used word overlap to link text to quoted segments [5]. The use of speech acts for summarization of email appears to have originally been used in the *SmartMail* project [7]. This technique marks an novel link between conversation summarization and argumentation.

More generally, approaches for the summarization of other discussion types have also been documented. Working on mailing list data, one approach clustered messages into subtopics and used centering to select sentences for an extractive summary [26]. The goal was to make what was a large collection of archived documents accessible for review - this is closely related to our task. The concept of recurring and related subtopics has been highlighted as being of greater importance to discussion summarization than the summarization of newswire data [39]. With topics changing more quickly in a question-response structure it is clear that the summarization of discussion requires a different approach to more traditional multi-document summarization techniques.

Moving the discussion towards more subjective texts, opinion mining - with a view to summarization - is also relevant to our project. While customer reviews are normally written in isolation of each other, unlike a discussion, research on their summarization offers some relevant ideas. One approach grouped sentences based on the feature discussed, then proceeded to use these groups to generate a summary of all the reviews for a product that minimized repetition [13]. This task could be grouped under 'opinion summarization', as introduced at the *Text Analysis Conference* in 2008. One participating paper by Lloret et al. [18] highlights the query-centered nature of the task. Opinion summarization work is typically limited to attaching polarity to features. While the TAC task does encapsulate the idea of 'justification' for a given opinion, argumentation features are not the focus.

More closely linked to our project is the idea of identifying agreement and disagreement, beyond positive and negative opinions. Galley et al. [11] used adjacency pairs to target utterances that had been classified as being an agreement or disagreement. This does rely on the identification of conversation participants (they use a conversation with two speakers), a prerequisite that is harder to satisfy in large discussion threads. Closer to our work is that of Boltuzic and Snajder [3] who set the challenging goal of identifying the prominent arguments an online discussions. They worked with argumentative sentences (rather than agreement/disagreement utterances) and used a hierarchical agglomerative clustering algorithm to group them into 'prominent arguments'. They reported scores of 0.15-0.3 on the V-measure cluster evaluation metric. Their results also highlighted some of the challenges in this approach. For example, that it is sometimes hard to find clear boundaries between argument clusters.

Work in the field of argumentation mining also bears similarities to our project. Argumentation schemes for discussions have been outlined [12], this work sets out a process for identifying and grouping *Argument Discourse Units* using a two-tiered approach with expert and novice annotators. A key difference between this and our task is the reliance on human annotators.

Cabrio and Villata [4] describe an approach based on *Textual Entailment* to give an overview of a debate. They cite that new participants to online discussions need to be updated on the history of accepted arguments made in the debate. Looking for accepted arguments in text goes beyond the argumentation features implemented as part of our project, however their goal of what might be described as debate summarization is highly related to ours.

Work has been done to investigate arguments raised in online discussion [3, 4, 12]. There has also been interest in the summarization of subjective content in discussions [13, 18, 11]. However, the use of argumentation as a basis for summarization does not appear to have been investigated. In this project we explore the intersection of these two areas by generating summaries of discussions based on the relations between points.

Chapter 3

Technologies

Our project relies on a number of software dependencies that form a foundation for the services we implemented.

3.1 CoreNLP

"Stanford CoreNLP provides a set of natural language analysis tools."[1]

CoreNLP provides the basis for our points extraction analysis and assists in selecting extracts during the generation of summaries. Running as a separate service, modules in our analysis pipeline make requests to get sentences, dependency parses and lemmas for text. While the CoreNLP frame work implements a number of different 'annotators', we only make use of `depparse` & `lemma` — which depend on `tokenize`, `ssplit` and `pos`.

3.2 Verb Case Frames

While not strictly a software dependency, we made use of verb case frames (standard representations of allowed verb argument patterns) in the definition of valid points. Using FrameNet frames included as part of the VerbNet index [31, 10], we were able to implement a mapping between these and the dependency parse information. This connection was implemented using queries on the Neo4j graph database.

3.3 Cypher and Neo4j

"Neo4j is a highly scalable native graph database that leverages data relationships as first-class entities, helping enterprises build intelligent applications to meet today's evolving data challenges."[2]

We use Neo4j to store and query parse information. Sentences are parsed using the CoreNLP dependency parser and the resulting graph structure is saved to a Neo4j instance. Tokens and dependencies are represented as nodes and edges respectively. Nodes are used to store the token in plaintext; the lemma, part-of-speech tag and its index in the source sentence. Edges are directional; connect governor to dependent tokens; and have a single attribute to store the dependency type.

"Cypher is a declarative graph query language that allows for expressive and efficient querying and updating of the graph store."[3]

---

[1] https://stanfordnlp.github.io/CoreNLP/

[2] http://neo4j.com/

[3] http://neo4j.com/docs/stable/cypher-introduction.html

Cypher is used to query dependency parses stored in the Neo4j database. After parsing the sentences from a user's post, all the dependency parses are saved. At this stage, a series of Cypher queries are executed to filter for allowed point patterns. These patterns, derived from case frame information, are represented in the Cypher syntax. Using Neo4j and Cypher in this way allows points to be extracted from many sentences at once. This was key to completing the analysis of large discussions in reasonable time.

3.4 Latent Dirichlet Allocation Implementation

We use a service to evaluate the core topics of a discussion. This is based on an implementation of the *Latent Dirichlet Allocation* generative model[4]. Discussion text is analyzed as a whole by a *Topic Analyzer*, a service that exposes this implementation — see Chapter 5. Topics are used in assessing the value of extracted points as well as guiding the extraction process.

3.5 ERB

"ERB provides an easy to use but powerful templating system for Ruby"[5]

We use ERB (Embedded RuBy) to generate HTML documents used as part of the evaluation (see Chapter 9). ERB templates were defined for each summary style being evaluated. The evaluation forms distributed on Mechanical Turk were also generated using ERB. Having templates for these documents allowed sets of summaries and surveys to be consistently re-generated as we iterated on the functionality of the summarization module and questionnaire format.

3.6 Ruby & Go

Almost all of the functionality of the tool is implemented in the Ruby programming language. Ruby was selected for its familiarity and suitability in connecting multiple networked services together. Once points had been extracted using a Ruby service, making use of CoreNLP and Neo4j, points are curated using a small program written in Go, this task required fast iterations and it allowed the results of changes to the point curation to be reviewed in a faster feedback cycle. Point curation is discussed in Chapter 7.

3.7 Docker

"Docker allows you to package an application with all of its dependencies into a standardized unit for software development."[6]

Docker is used to isolate the different environments of each service. CoreNLP and Neo4j are both run as containers using standard images. Our modules are run under the same system. All modules communicate using JSON requests. Docker Compose is used to codify the system's configuration and start services when they are required for use in development.

[4] https://github.com/ealdent/lda-ruby
[5] http://apidock.com/ruby/ERB
[6] https://www.docker.com/what-docker

Chapter 4

Research Goals

Having given an overview of the related work and background to the project, we now go on to outline our goals.

Building on points extraction, we wanted to use points as a new type of content unit in summarizing large discussions. Points allow sentences to be broken down into a number of atomic units that can be compared and matched to others. We wanted to use points as well as the analysis they enable to build a picture of the discussion.

This first required a robust means of identifying and extracting points from text to serve as a foundation for later analysis. We needed to investigate not only whether points were a viable content unit, but if they could succinctly present information better than extracted sentences. Given plain text from a discussion, we would need 1) a pattern or signature that could be used to link points — regardless of their exact phrasing — and 2) a short readable extract that could be used to present the point to readers.

We were also interested to build on these extracted points by linking them in different ways to model the discussion. Firstly, matching points and counter points using negation and antonyms was the main comparison of interest. Additionally, linking co-occurring points as a more generalized view on participant stances was also a goal. We wanted to use these relations between points; as well as point metadata extracted from text such as referenced topics, source post and negation; in formulating a structured summary that is useful to readers.

We also needed to evaluate the results of the tool as a whole. Testing the value of our analysis, and whether the results were useful, was also a goal for the project. We wanted to show that our approach to points extraction was suitable for the summarization of discussions. Aware that the base analysis of extracting points has utility beyond that of summarizing discussion, we also wanted to build our tools for analysis in a modular manner such that components may be reused in the future for further work on other tasks.

Chapter 5

System Architecture

This chapter is intended to give a technical overview of the system design and add context to the analyses detailed in the following chapters. The system has been implemented as a series of isolated modules that together form a processing pipeline. Each stage can take a significant time to complete, so the output is written to disk in standard format between each stage. In short, extracted points are aggregated for an entire debate, which are then revised before being grouped into a summary.

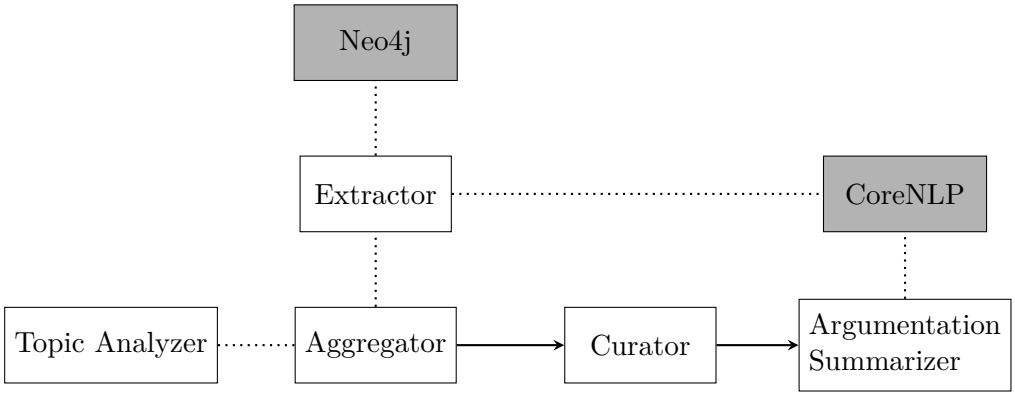


Figure 5.1:
Architecture Diagram. Arrows represent the pipeline data flow; dotted lines represent dependencies.

Figure 5.1 shows a high-level overview of the various modules and how they make up the analysis pipeline. Points for all posts in the source text are gathered by the *Aggregator*; these are filtered and cleaned by the *Curator*, and finally, curated points are used to generate a summary by the *Argumentation Summarizer*.

Before source text can be analyzed it must be cleaned for invalid characters. Posts in a discussion begin as single text files, these are loaded and transformed into JSON files that preserve their index and valid UTF-8 content. The *Aggregator* loads these JSON post files to extract the points made in each post.

The *Aggregator* loads all the JSON posts for the selected discussion. Before extracting points for each in turn it uses the text from all the posts (representing the discussion) to collect a list of topic words from the *Topic Analyzer*. For each post in turn the text and topics[1] are submitted to the *Extractor*. This carries out the point extraction process for

[1]Only sentences containing a topic word are analyzed for points to reduce processing time.

each sentence in the post — making use of both the CoreNLP dependency parser and Neo4j datastore (described in detail in Chapter 6). Then extracted points for all posts are saved to disk. A modern, mid-range computer can complete 150,000 words spread across 2300 posts in around 10 minutes, this is the slowest stage of the analysis pipeline.

While points must come from a sentence containing a topic word, at this stage points are still largely unfiltered. The *Curator*, a short program to filter and re-format points makes a series of transformations such as merging pronouns under a single `PERSON` label. Patterns for low-quality points are also encoded at this stage, for example: the `PERSON.nsubj be.verb sure.adj` point pattern that comes from every "*I'm sure*" is rejected. These transformations and rejections are applied to the list of points and a refined version saved for use at the next stage. This process is described in Chapter 7.

The *Argumentation Summarizer* is the final stage in the pipeline. Given a list of curated points, this module groups common points into the different summary sections — some based on argumentation analysis. This analysis (identifying counter points for example) is closely linked to our summarization task and is performed at the same stage in the pipeline. Summary sections are filled using points that have not been used earlier in the summary and are suitable for the current group. Once a point has been used in a summary section, it cannot be used again. This module must also choose an extract to display for each point. The point `abortion.nsubj be.verb right.dobj` occurs many times in the discussion and a single occurrence to represent this must be selected. This process makes use of the CoreNLP module again to request a dependency parse used in rating the quality of extracts. Selected extracts are formatted and used as text seen in summaries. Chapter 8 discusses the generation of summaries in detail.

The *Summarizer* also formats summaries of grouped extracts for selected point clusters as HTML files. This is the end of our pipeline - possibilities for further work are discussed in Chapter 10.

A monolithic architecture was avoided due to processing time required at each stage. Running the entire process, when making adjustments at the final stage, was not feasible. Currently stages in the pipeline must be run manually — however this is trivial to automate with a script. The process for using the pipeline has been outlined in the Maintenance Manual in Appendix B. The following chapters build on this technical overview and give more detail on the intricacies of the analysis performed by each module.

Chapter 6

Point Extraction

6.1 Point Concept Overview and Examples

A 'point' is a concept we created to describe a common object in our analysis — originally defined as: a verb and its arguments. Points encapsulate both a human-readable extract from the text as well as a pattern representing the core components that can be used to match and compare points to others. Extracts and patterns are stored as attributes in a key-value structure that represents a point. The following example walks through extracting such a structure from the following sentence:

"*I don't think so, an unborn* ***child*** *(however old)* ***is*** *not yet a* ***human*** *being.*"

This sentence puts forward the idea that, until a baby is born, it does not qualify as human. This idea is centered around the verb '*is*', "...an unborn child **is** not yet a human being". Another sentence in the discussion may also put forward this idea, for example: "*So you say: children are not complete humans until birth?*". Both of these sentences have words that express the same idea, only the context differs. At this first stage, negation and question tokens are ignored. The aim is to get to the core idea being discussed, regardless of negation or what might be implied by a question. We represent these core ideas as point patterns which list the key arguments and the verb. These two sentences both contain points with the pattern: `child.subject be.verb human.object` — the core idea they both reference.

This pattern loses some of the sentence context and is not suitable for use in human-readable text. To overcome this, when extracting points from sentences, points are represented as a pattern but *with an accompanying extract.* Extracts are strings made of one or more sentence substrings. The extracts for the points in these two sentences would be:

"*an unborn child is not yet a human being*"
"*children are not complete humans until birth?*"

Using this pattern/extract representation enables points expressed in different tenses and contexts to be grouped together using their patterns — while also retaining the relevant parts of the sentence to present the human-readable points. While points are the fundamental unit used in our analysis, Figure 6.1 shows how points make up part of a larger, hierarchical, discussion model. Points are stored in a key value representation with

Figure 6.1: Discussion model showing relationships between subcomponents

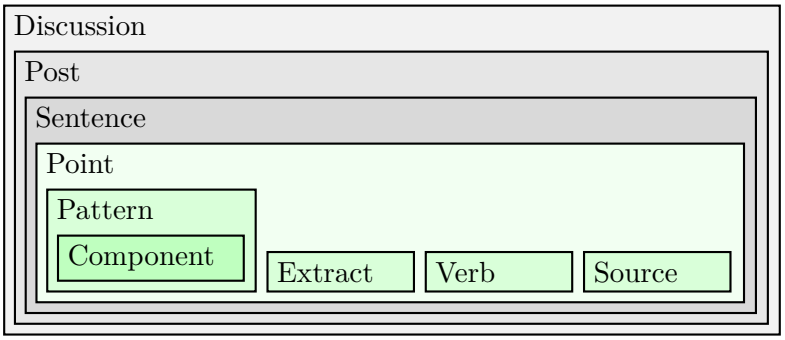


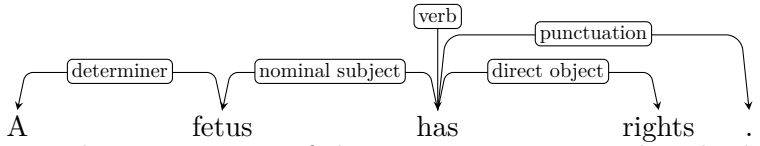
Table 6.1: Example point data structure

Attribute	Example Value
Pattern	`{child.nsubj be.verb human.dobj}` Single Component: `child.nsubj`
Extract	"*an unborn child is not yet a human being*"
Verb	to *be*
Source	*Post: #42, Debate: Abortion*

attributes for the pattern, extract, verb and point's original source post in the discussion, see Table 6.1.

The *Pattern* attribute consists of a list of two or more *Components*. Each component represents a dependency of the verb and the relation of that dependency. For example, if a child were to be the subject, then the component would be `child.nsubj` (nominal subject). Relations are defined using Universal Stanford Dependencies[1]. Patterns are broken down into components to enable certain comparisons between points, for example, to test that `abortion.nsubj be.verb legal.dobj` and `abortion.nsubj be.verb illegal.dobj` are 'counter points'. Such analysis is discussed in Chapter 8.

With a definition for a point, we can now explore the process of extracting these from text. A point, as described above, is built up using information from a dependency parse. A dependency parse is represented in a graph structure showing relationships between different tokens.

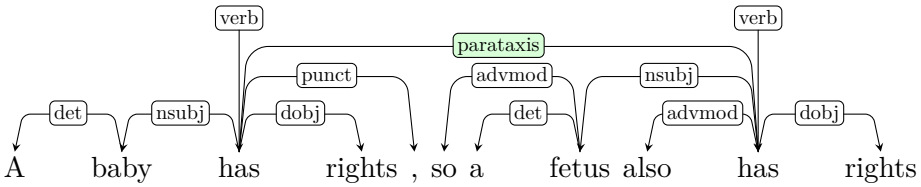


From this parsed representation of the sentence we can see how both an extract "*A fetus has rights.*" and pattern `fetus.nsubj have.verb right.dobj` could be constructed using the graph's relations. In this example the nominal subject and direct object relations form the pattern, while all relations are followed from the verb to generate the extract. In more complex graphs, not all relations are used to avoid excessively long extracts. Also, patterns must sometimes be derived from more than subject and object relations. Sentences often contain more than one point, the following (more complex) dependency parse shows a sentence that references two different points.

[1] http://universaldependencies.org/docs/en/dep/

Figure 6.2: VerbNet frame example for verb murder

```
<FRAME>
  <DESCRIPTION primary="NP V NP" secondary="Basic Transitive"/>
  <EXAMPLES><EXAMPLE>Brutus murdered Julius Cesar.</EXAMPLE></EXAMPLES>
  <SYNTAX>
    <NP value="Agent"><SYNRESTRS/></NP>
    <VERB/>
    <NP value="Patient"><SYNRESTRS/></NP>
  </SYNTAX>
  <SEMANTICS>...</SEMANTICS>
</FRAME>
```



Here the parse information helps isolate the two points: `baby.nsubj have.verb right.dobj` and `fetus.nsubj have.verb right.dobj`. A `parataxis` relationship connects the two — though this is one such relation that is not used to expand a point's extract.

6.2 Defining Points with Verb Frames

Not all of a verbs relations should be included in the point pattern but rather only those essential to the meaning. This typically means using only subject and object; however, this cannot be defined for all verbs with a single rule. Some additional dependencies, like adverbial modifiers and parataxis as seen in the previous example, which do not introduce information relevant to the point's pattern can be universally excluded. However, not all dependency types are easily classified. This is largely due to the different verb classes (transitive, intransitive and ditransitive) and how these are represented in parses by CoreNLP. In order to look for legal patterns in dependency parse graphs, we needed an index of verb transitivities that listed allowed patterns for a given verb. We started using FrameNet frames for this task and later extended these with a generic query as a heuristic to match patterns not present in FrameNet.

6.2.1 VerbNet and FrameNet

VerbNet is an XML verb lexicon that lists classes of verbs and accompanying FrameNet frames [31, 10]. Represented in VerbNet's 274 'classes' are 4402 member verbs. For each of these verbs classes, a wide range of attributes are listed. FrameNet frames are one such attribute, and the only one we make use of. These describe the verb's syntactic arguments and the semantic meaning of each in that frame. We can use the frames represented in the source class to determine the allowable forms for each verb. An example frame for the verb 'murder' is shown in Figure 6.2. Here we see that the verb takes two Noun Phrase arguments, an Agent (murderer) and Patient (victim).

We use this `SYNTAX` element information to determine the type of dependencies that can be included in the pattern for a given verb. To create a 'verb index' we parsed the VerbNet catalogue into a key-value structure where each verb was the key, and the list of allowed frames the value. Some verbs are listed in more than one class; for example, *feel* is listed in seven classes. In such cases the verb was allocated all frames for all classes it was a member of. Using this index, points can be extracted by querying the verb's dependency parse using information from each of its frames.

6.2.2 Matching Frames in Parses

With an index of verbs and their allowed frames, all the information required to identify points in dependency parses is available. However, frames are not inherently capable of querying the dependency graph structure and queries for each type of frame must be written. While frames in different categories encode additional semantic information, there are far fewer 'syntactically unique' frames than verb classes. Common frames such as `NounPhrase Verb NounPhrase` cover a high percentage of all frames in the index. We have manually implemented a means of querying dependency parses for 17 of the more common patterns covering 96% of all frames in the index.

FrameNet provides short example sentences for each frame. Using the dependency parses for these examples (returned from CoreNLP), we were able to manually map dependency relations to the syntax components in the frames. Mapping `nsubj` and `dobj` relations to noun phrases is relatively straight forward. However, more complex frames such as `NP VERB NP PREP NP PREP NP` require a mapping from a complex, recursive parse onto a flat frame structure. In these more complex cases, queries must search beyond the verb's immediate arguments for tokens with relations matching these additional frame components. For example, in the `NP VERB NP PREP NP PREP NP` frame, the first prepositional is connected with a case relation to the third noun phrase; which is in turn connected to the verb via a nominal modifier. Frames are always flat and nested structures must be expressed in the queries to flatten parses for these larger frames.

These queries, mapping frames to parses, must be written manually and was a time consuming exercise. Queries are written in *Cypher*, the declarative query language implemented by the Neo4j database that we used to store dependency parses during analysis. Figure 6.3 shows an example for a simple frame. Restrictions imposed commonly require subject and object relations. The query returns the nodes that match these relations. Should a verb *not* have the relations to complete the frame then nothing is returned from the query.

The remaining 4% of frames, as well as verb cases not covered by FrameNet, were matched using a 'Generic Frame' - discussed in the following section.

6.2.3 The Generic Frame

Originally our intent had been to use only queries derived from FrameNet frames to define allowed patterns for points. However, there were verbs missing frames for some patterns of interest. Usually this meant a simple frame for the verb was listed but it failed to adequately capture the point's idea. Take the following example:

Figure 6.3: Example Cypher query for the common `NP V NP` frame

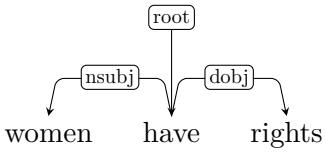
```
MATCH (verb:Node)
WHERE verb.uuid = UUID
MATCH (verb)-[rel_nsubj:REL]->(nsubj:Node)
MATCH (verb)-[rel_dobj:REL]->(dobj:Node)
WHERE rel_nsubj.label = "nsubj"
AND rel_dobj.label = "dobj"
RETURN nsubj, verb, dobj;
```

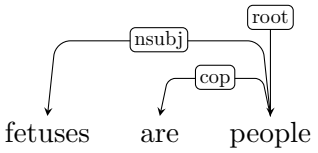
"*These images reflect the reality of abortions*"

This is an extract that only matched `image.nsubj reflect.verb`. Shorter points like this make extracts for points in a cluster considerably more varied, resulting in less cohesive clusters. To overcome this we introduced the idea of the 'Generic Frame'. This was a new query that could be run against any dependency graph to extract subjects, objects and open clausal complements. This was used to increase the number of complete points. For the extract above, the Generic Frame matched this more complete pattern: `image.nsubj reflect.verb reality.dobj`.

6.2.4 Copula Verbs

The dependency parse differs for copula verbs as the verb is no longer the root of the parse. Analyzing dependencies *from the verb* only works when the verb is the root. The following examples illustrate the inherently different structure.





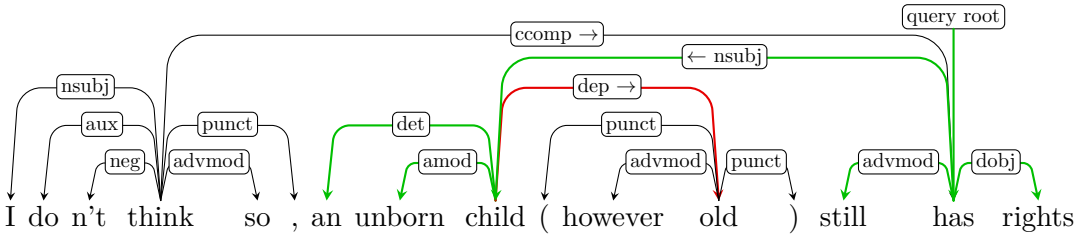
Rather than making the copula verb the head of the sentence, we opted to write adjusted queries that could be used for copula verbs when matching against frames. This allows the dependency parse to be left unaltered and makes the implementation more explicit. The example above becomes `fetus.nsubj be.verb people.dobj` so it matches queries like: *points where people are the 'object'*.

6.3 Human Readable Extracts for Points

So far we have covered the identification of points and how the dependency parse information has been interpreted to arrive at a point's pattern. This is separate from the point extract, which is a human readable sentence that can be presented in a summary. This is extracted using a second query on the information in the dependency parse. The task of selecting an extract for a point can be described as follows: recursively follow dependencies to nodes from the verb until the tree is fully explored.

This task was implemented as another Cypher query, similar to verb frames. However, rather than only detecting a small number of expected dependencies, this query follows

all dependencies recursively. Nodes in the graph that are related to the verb, or any of its descendants, are returned as part of the extract for the point. However, to keep points succinct the following relations are prevented from extending the extract tree: *adverbial clause modifiers, clausal subjects, clausal complement, generic dependencies and parataxis.* Generic dependencies occur when the parser was unable to determine the dependency type. These either arise from errors or long-distance dependencies and are rarely useful in extracts. However, the other blacklisted dependencies (csubj, ccomp etc.) were selected based on our results and should be seen as a starting point. While these often led to including unnecessary information, always excluding them is likely an oversimplification. The following dependency graph depicts the recursive extract expansion process.



The green paths represent a query expanding into the graph following permitted dependencies. This gives "*an unborn child still has rights*" as the extract for the point. Upon reaching `dep`, a blacklisted dependency, that route is no longer explored and so excluded from the extract. Should the query root have been for the other verb (*think*) the blacklisted clausal compliment would have isolated the extract to "*I don't think so,*".

This effect is not dissimilar from sentence compression, only there is a starting point 'query verb'. The end result is not so dissimilar to that reported by Knight and Marcu [15], though the approach was fundamentally different. As they do, we also strip trailing punctuation and capitalize where required. This is done at the summary generation stage in our approach, see Chapter 8.

6.4 Extraction Process

The previous sections give a theoretical description of our points extraction approach, whereas this section outlines the implementation. Making reference to Figure 5.1, the Aggregator first uses text from amalgamated posts to generate a series of topic words. Each post is then submitted in turn, along with the topics, to the Extractor. The Extractor requests dependency parses for the sentences in the post referencing topics from CoreNLP. Parses are then saved to the Neo4j graph database before a query is run to extract all the verbs; for each verb the relevant frames are looked up in the index and the matching Cypher queries run to extract point patterns. For each matched verb, a further query is run to retrieve the point's human readable extract. These are returned as a JSON array to the Aggregator, which writes them to disk in preparation for the Curator — the next module in the pipeline.

Chapter 7

Point Curation

Points extracted at the first stage must be refined to make them suitable for use in the following summarization module. These refinements focus on removing points generated from common phrases such as *'I'm sure'* and *'Ask yourself...'*. While adjustments could be made as points are extracted, this was implemented as a separate task during our implementation phase. Rejecting and editing point content is also conceptually different from identifying them in source text and much additional information is encoded at this stage. This chapter explains in detail the adjustments made to a list of extracted points in order to make them useful in the summarization task.

7.1 Minimum Requirements

Some basic restrictions are applied to points matched by the Generic Frame. The Generic Frame returns a large number of additional points (on top of other frame results), many of which are of little interest. To make sure that points from the Generic Frame are useful the following heuristics are applied as requirements.

- Contain a topic word - this is a good estimation of a point's informativeness.
- Have a subject, object and average word length of 5 or more characters - this restricts extremely short points from being matched. Points ideally make some form of connection between words of interest.

These requirements balance the benefits and problems of matching a greater number of patterns, some lower quality, using the Generic Frame. This means that shorter points, or points without a search topic word can only be returned by matching the query for a FrameNet frame.

7.2 Person Nominal Subjects

In an effort to better cluster extracted points into distinct statements used in the discussion, we opted to merge pronouns under a single Person subject. Points with person nominal subjects are very common and often segregated by different pronouns. Take the following example:

- `people.nsubj kill.verb people.dobj`
- `he.nsubj kill.verb people.dobj`
- `they.nsubj kill.verb people.dobj`

While there is a semantic difference between these points, we made the decision to amalgamate them under patterns like `PERSON.nsubj kill.verb people.dobj`. This means clusters for points are not dispersed among many pronouns. Reducing repetition between clusters means we can continue to rely on a cluster's size as a measure of salience in the summarization task.

7.3 Blacklists

To complete this module's function of cleaning and refining points, a number of 'blacklists' were established. These defined generic point patterns that were of little interest and should be removed from all topics of discussion. Blacklists were based on the patterns of points extracted from the Abortion debate corpus. All blacklisted patterns are listed in Appendix D.

7.3.1 Ambiguous Subjects

We identified a set of problematic referring expressions affecting the utility of extracted points. Patterns such as `it.nsubj have.verb rights.dobj` are ambiguous as the surrounding sentences are required to resolve the anaphor "it". One possible resolution was to merge such points with the most common pattern with a noun or proper noun nominal subject (`fetus.nsubj have.verb rights.dobj` for example). However, rather than merging points into other groups, we opted to exclude them in the interest of keeping the results verifiable.

The following ambiguous subjects were grounds for rejection of a point: *it(PRP), that(DT), this(DT), which(WDT), what(WDT).*

7.3.2 Person Verb Combinations

There are a series of verbs that are not permitted to form points when they have a `PERSON` subject. Certain phrases such as "*I think*" are very common; while important for comprehension, they do not make interesting points. *object, understand, realize, debate, speak, stand, refer, explain, support, feel* are some examples of 'actions' that participants write about which generate unsuitable points for use in summaries.

These restrictions are only applied to points that consist of two components, a verb and `PERSON` subject. This means that `PERSON.nsubj understand.verb issue.dobj` is a valid point, whereas `PERSON.nsubj understand.verb` is not.

7.3.3 Patterns

Common phrases create another class of poor quality points. Phrases such as *'I'm correct [in saying]', 'Something happened [to/that]', '[opposition] make the claim'* do not add valuable information. They could not be used to add useful content to the summary and are removed.

Chapter 8

Summary Generation

A process to extract and refine a list of points could be used in a number of ways. From this foundation, we investigate a potential use case for points in a summarization task. Extracted points are short and have a 'pattern' that enables new comparisons not possible with plain text extracts. One such comparison is the analysis of counter points. This argument analysis is performed within this module as it is closely linked to the task of generating summaries.

This chapter first explains a number of utility components used in building each summary section, then goes on to describe how each section is generated in detail.

8.1 Permitted Extracts

In a cluster of points with the same pattern there are a range of different extracts that could be selected. There is much variation in quality caused by additional words, bad parses and poor punctuation. Completely removing points with extracts that matched heuristics for poor quality would significantly reduce the cluster size — the key factor used in determining a cluster's salience. Such points are useful in analysis through aggregation but not for presentation in summaries.

Instead we implemented a set of rules that prevent a point's extract from being used in the summary, even while the point itself remains in the cluster to aid aggregation and sorting tasks. *This is fundamentally different from the curation task.* Points with valid patterns but poor quality extracts are still required in determining important clusters (cluster member counts are used). Points excluded at the curation stage are not at all useful.

Predominantly, points are prevented from being presented based on the presence of certain substring patterns tested with regular expressions. Exclusion patterns include: starts with a wh-question token or contains a question mark; has two or more consecutive words in block capitals or contains a mid-word case change. Since these patterns can be applied very quickly, extracts are tested using these first before using more expensive analysis to make further exclusions.

Following on from these, there are more complex exclusion patterns based on the dependencies from re-parsing the extract using CoreNLP. An extract may consist of many parts spread across the sentence; these do not always make sense. Extracts with clausal or generic dependencies (on re-parsing) are excluded. Such dependencies are characteristic

of overly long extracts or erroneous parses. An extract may also be excluded from display if it contains more than two conjunctive relations or nominal dependencies.

Finally some additional restrictions; not based on the dependencies or the strings; are applied. For example, an 'it' must be preceded by {on, and, but, whether} and repeated words are not allowed.

8.2 Extract Selection

Points are manipulated in clusters where members share a common pattern, for example, there might be a cluster where each point had the pattern `woman.nsubj make.verb choice.dobj`. This cluster would contain all the points with this pattern and in turn all the extracts available for use in a summary to express this cluster's point. Even after refining the list of points input to this component of the pipeline, there is much variation in the quality of the extracts available for selection. Take this example cluster of extracts for points about the Genesis creation narrative:

"The world was created in six days."
"The world was created in exactly 6 days."
"Is there that the world could have been created in six days."
"The world was created by God in seven days."
"The world was created in 6 days."
"But, was the world created in six days."
"How the world was created in six days."

All of these passed the 'Permitted Extracts' stage. The next task is to select the best extract to represent the cluster. In this instance our approach selected the fifth point, "The world was created in 6 days.". Selecting the best extract is performed every time a cluster (points with a pattern) has been selected for use in a summary. The best extract is used to represent the whole cluster in human readable text. Selections are made using a bag-of-words, length-weighted, bigram model of the extract words in the cluster. The goal was to select the most succinct extract that was most representative of the entire cluster. We tested this in our evaluation, see Chapter 9.

The bigrams for each extract in the point cluster were collected. Bigrams were given a value equal to the number of times they repeat in the cluster. Each extract was then given a value equal to the sum of the values for the bigrams it contained. This aggregated score value was then divided by the number of words in the extract to determine a final score. The extract with the highest score was selected to represent the cluster in the summary.

8.3 Extract Presentation & Formatting

Our points extraction approach works by selecting the relevant components in a string for a given point, using the dependency parse graph. While this has a key advantage in creating shorter content units, it also means that extracts are often poorly formatted for presentation when viewed in isolation (not capitalized, leading commas etc.). To overcome this we needed to implement a means of correcting extracts for presentation. Two examples

that illustrate the problem of poor punctuation and style are: "*that Scientifically , human life begins at this stage*" and "*.that a fetus is a person ,*".

To overcome such issues we have a function to ensure the extract meets the following properties: is capitalized; ends in a period; commas are not preceded by a space; contractions are applied where possible; and consecutive punctuation marks condensed or removed. Certain determiners, adverbs and conjunctions (because, that, therefore) are also removed from *the start* of extracts. The two extracts above are translated into the following cleaner versions: "*Scientifically, human life begins at this stage.*" and "*A fetus is a person.*".

This relatively simple process does not use parse information and predominantly operates on the opening and closing characters of an extract. The intention is to quickly tidy up the string for display as a short sentence. This feature still requires work. The task of correctly formatting a string has only been touched on here and is a candidate for further work.

8.4 Avoiding Repetition in Summaries

A cluster's inclusion in a particular summary section is a function of the number of points in the cluster. This is based on the idea that larger clusters are of greater importance and should be first to feature. Accounting for repetition using this metric is commonplace in other automatic summarizers (often aimed at news data). However, when generating summaries of social media discussion it raises the question: will this result in only showing the majoritarian view? While this is likely, it is also hard to account for. We have a number of sections such as 'points people disagree on' and 'longer form points' to bring up ideas from both sides and partially overcome this issue. However, without stance annotated posts, it is hard to know that a summary is truly balanced. Currently, cluster sizes are still the key factor in their selection for summary sections.

To avoid large clusters being repeatedly selected at each summary section, a list of used patterns and extracts is maintained. When an extract is used in a summary section it is 'spent' and added to a list of used patterns and extracts. The extract pattern, string, lemmas and subject-verb-object triple are added to this list. Any point that matches anything in this list of used identifiers cannot be used later in the summary. The same extract may be included in many clusters under different point patterns. For example, "Life begins at conception." is matched by both the `life.nsubj begins.verb` and `life.nsubj begins.verb at.prep conception.dobj` patterns. This means that when checking if a point is suitable it must be checked against the used extracts *and* patterns of those previously selected.

8.4.1 Negation Shorthand

As part of the argument analysis of negated points (discussed in section 8.5.2), a condensed point representation was developed for expressing extracts that shared many of the same words. For example, to include both "A fetus is a human" as well as "A fetus is **not** a human" is a highly repetitive presentation. Such a presentation does not make for good reading and uses surplus words that could be used to express additional information.

Based largely on a string diffing library[1], we developed an alternative, more condensed presentation for such pairs of points. Under this, the above examples could be written as "A fetus is **{**not**}** a human", saving five repeated words. However, more complex examples also occurred such as "Abortion **{**is not|should be**}** legal **{**after 12 weeks**}**" ("Abortion should be legal" and "abortion is not legal after 12 weeks"). While there are adjustments that can be made to improve this, such as requiring the two extracts to have a similar number of words, complex examples still surfaced and the formatting was eventually dropped from use in summaries.

This diffing functionality is however still used when matching similar negated points into pairs (see section 8.5.2).

8.4.2 Topic Words

Topic words are used in the point extraction stage are passed along to the summarization stage with the list of points (previously discussed in sections 3.4 & 6.4). Topic words are used in some summary sections to guide selection and avoid repetition. They are also used to highlight topic words in extracts in the formatted summary presentation (see Chapter 9).

8.5 Summary Sections

A summary could be generated by listing the common points in the discussion. However, we were interested to explore beyond this into basic argumentation relations, i.e. point and counter points. We grouped summaries into a series of sections where each was the result of a different analysis on the list of extracted points. This section details how summaries were built up from these various different components.

8.5.1 Counter Points

The identification of counter points was a goal for the project from the start. This analysis was intended to highlight areas of disagreement in the discussion. Counter points are matched on one of two possible criteria, either the presence of negation terminology (see Section 8.5.2) or an antonym for a pattern component.

Potential, antonym-derived counter points, for a given point, are generated using its pattern and a list of antonyms. Antonyms were sourced from WordNet [22] using the NLTK corpus reader interface[2]. Taking the pattern `woman.nsubj have.verb right.dobj` as an example, this generates the following counter points:

- `man.nsubj have.verb right.dobj`
- `woman.nsubj lack.verb right.dobj`
- `woman.nsubj have.verb left.dobj`

This shows that substitutions can be applied to any component that has an antonym, including verbs. If there are many words in the pattern with antonym matches then multiple potential counter points are generated for a single point.

Once a list of potential point vs. counter point patterns has been generated, these are filtered to ensure that the generated counter point pattern exists in the list of points. From

[1] https://rubygems.org/gems/differ
[2] http://www.nltk.org/howto/wordnet.html

the example above, only the first generated pattern: `man.nsubj have.verb right.dobj` appeared in our Abortion discussion. A check is also made to remove duplicate pairs - a point/counter point pair cannot also appear in reverse order. Counter points must be three or more components in length, e.g. the pair `life.nsubj begin.verb` and `life.nsubj end.verb` cannot occur. Points with shorter patterns tend to have more varied extracts and lead to 'counter points' that are not suitably matched. Pairs generated from the verbs 'come' and 'go' are discarded as they also lead to low-quality matches.

These pairs are selected for display based on the average cluster size for the point and counter point. This is intended to represent "the most common pairs of contradictory points". In the summary text, only the extract for the point is displayed, not the counter. The section is introduced as "points that people disagree on", this means that stating either presents it as a contested point. Previously the 'negation shorthand' was used to present points and counter points but variation in extracts often led to complex examples that were hard to read.

8.5.2 Negated Points

Negated points are displayed in the same summary section, "points that people disagree on", however they are identified differently. Negated points are selected from the largest clusters that have not already been used in the generation of previous sections. Negation terminology is not commonly part of the cluster pattern, for example, the `woman.nsubj have.verb right.dobj` cluster could include both "A woman has the right" and "The woman does not have the right" as extracts. Identifying these negated forms *within* clusters is how negation-derived counter points are collected.

First the cluster is split into two groups, extracts with negation terminology and those without. The Cartesian product of these two groups gives all pairs of negated and non-negated extracts. For each of these pairs a string difference is computed[3], the complexity of this difference pattern is used to identify a clear match. For example "{+that }a fetus is {-not }a person{- under the law of the eu}" is not as clear as "that a fetus is{n't }a person". By counting elements in the difference pattern we are able to select shorter and cleaner examples where possible.

However, similarly to antonym-derived counter points, clean examples do not always exist. Negated points are represented a single extract from the pair and listed under the same "*points that people disagree on*" section.

8.5.3 Co-occurring Points

As well as counter points we were also interested to present points that were commonly made by the same participant. These related pairs were used in a summary section intended to show points that were positively related, i.e. raised in conjunction with one another, rather than being negated or contradictory. Our corpus did not annotate participant's identities so the assumption was made that each post was from a unique participant.

To identify co-occurring points, each post in the discussion was first represented as a list of points it made. Taking all pairwise combinations of the points made in a post, for

[3] https://rubygems.org/gems/differ

all posts, generates a list of all co-occurring points. Using the point patterns, these pairs can be sorted based on the number of times they occur.

Co-occurring pairs are rejected if they are too similar — patterns must differ by more than one component. For example, `woman.nsubj have.verb choice.dobj` could not be displayed as related to `woman.nsubj have.verb rights.dobj` but could be with `fetus.nsubj have.verb rights.dobj`.

Pairs displayed in the summary can only use available points remaining after the counter point analysis. They must also have a suitable quality extract to display, if not the next most common pair is used instead.

8.5.4 Commonly Occurring Points

Before selecting the points for the remaining sections of the summary, a number of the remaining most common points are listed. There are no restrictions on points from this section other than that they must not have been used by the previous sections. The intention is to present common points important to the discussion that have no counter or co-occurring point.

8.5.5 Topic Points

There was also an opportunity to display points broken down by topic. To determine the salient topics the subjects and objects for all clusters were tallied. This gave a ranking of topic words used in points. This information is not available in the unordered list of topics returned from the topic model that is passed along the pipeline with the extracted points.

Using these common topics, points containing them can be selected and displayed in a dedicated section for that topic. Extracts for a given topic are selected based on the size of the cluster while also minimizing repetition with the extracts already selected for that topic. Similarly, only points not used in the previous sections can be used.

8.5.6 Longer Pattern Points

Not all patterns are three components long. While patterns of three components represent most large clusters, less common, longer form points offer more developed extracts. Example extracts from longer form patterns include: "*The human life cycle begins at conception.*" and "*A human being refers to a specific living organism of a specific kind.*". The patterns for these points also include the prepositional phrases which expand on the idea being discussed.

Longer points are selected based on the number of components in the pattern; such points must have more than three components. To avoid repetition, when longer form points are being listed, each consecutive point added to the section must introduce at least one new topic word. This requirement avoids common points, differing solely on the prepositional phrase, being listed in this section.

8.5.7 Topic Linking Points

An alternative to selecting points with a longer pattern is to instead select points that mention more than one topic word. This allows less common extracts, that are still relevant, to be displayed.

This section is based entirely on the point extracts, not the pattern clusters. Extracts are sorted on the number of topic words they include. From this sorted list, the top 100 extracts are taken to form a group. Next, the previously described 'Extract Selection' process is applied to repeatedly select the most presentable and representative extract. This is done until three 'Topic Linking Points' are extracted for the section.

8.5.8 Common Questions Points

As a final idea for an informative summary section we opted to include a list of questions that had been asked a number of times, repeated questions were far less common. To gather a list of common questions we started by selecting points with extracts containing question marks. These question points were then clustered using their patterns. While extracts from up to the top three repeated questions were displayed, it was uncommon for there to be three repeated questions to complete the section.

Chapter 9

Evaluation

This chapter will detail the evaluation metrics; how they were used to test the performance of the tool; and what the results of the evaluation were. Three separate studies were carried out in total. The first two compared six debates from the Internet Argument Corpus [36]: abortion, creation, gay rights, the existence of god, gun ownership and healthcare. This corpus was extracted from the online debate site *4forums*[1] and is a large collection of unscripted argumentative dialogs based on 390,000 posts. Our final study made use of the same debate summaries as well as two new summaries of 2014 Independence Referendum discussion on Twitter.

9.1 Research Questions

We wanted to evaluate the project as an automatic summarization tool for argumentative discussion. The end result of our analysis was a summary comprising of point extracts for a given debate. By assessing the quality of this output, we also can evaluate the underlying points extraction analysis and test its utility. The following research questions were set:

- Are our summaries more readable and informative than those produced by existing tools?
- Does introducing summary sections with an explanation improve readability?
- How well does our bigram-based extract selection perform?
- Which sections of generated summaries are most and least useful to readers?

To address these questions we compared different versions of our summaries against equal-length summaries generated by an implementation[2] of the approach described by Nenkova et al. [25]. We also gathered scores for groups of extracts scored by the bigram model for comparison with human scores. Additionally, summaries generated with randomly selected extracts (for the same point clusters) were compared against a summary based on our bigram model for selection. We also facilitated free textual feedback to help address the utility of individual sections.

9.2 Evaluating Automatically Generated Summaries

The evaluation of automatically generated summaries is an ongoing discussion. A common approach is to make a comparison against a model summary, written by a human. Various

[1] http://www.4forums.com/political/

[2] http://homepages.abdn.ac.uk/advaith/pages/teaching/NLP/practicals/Practical3.zip

metrics, such as ROUGE [17], are used to make the comparison between summaries statistically. These are however limited by the assumption that there is a single best model for a summary. Alternative methods have been proposed. The Pyramid Method is one such example, it models content units across the collection of summaries being evaluated to give each content unit a weight based on how commonly it occurs [24]. These weights are used to allocate scores to summaries without an ordering bias.

Automated methods are used primarily at the *Document Understanding Conferences* when many summaries must be evaluated for a given task. However, ideally summaries could be manually evaluated because it would allow extrinsic attributes such as the utility of a summary for a particular task to also be accounted for.

In this project we compare summaries against those generated using the approach outlined by Nenkova et al. [25] as a baseline. This paper outlines an approach for extractive, sentence summarization that implements 'Context Adjustment'. As sentences are selected, based on the value of their content words, the values of words in selected sentences are decreased. This shifts criteria for selecting the following sentence and results in selected sentences having a similar distribution of words to the input text. This implementation performed significantly better than more than half of the other submissions for the 2004 Document Understanding Conference. Only one system scored significantly better, this was a supervised Hidden Markov Model implementation. The system submitted by Nekova et al. was not supervised.

Comparing our summaries against those generated by this implementation; for the same input text; using scores from human judges rather than automated methods; formed the basis for our evaluation design.

9.3 Design

In response to our research questions we set the following null hypotheses for the evaluation:

- The readability and informativeness of our summaries do not improve on those of existing tools. (**H1**)
- Introducing summary sections does not improve overall readability. (**H2**)
- Our means of selecting extracts is not better than random selection. (**H3**)
- All summary sections are equally useful to readers. (**H4**)

To test these we ran three separate studies. These each consisted of a questionnaire to be completed by study participants. Please see Appendix E for an overview diagram of all questionnaire layouts and Appendices F & G for example questionnaire sections used.

9.3.1 Study Participants

To test these hypotheses, we prepared three comparative studies. These are described in Sections 9.3.3 onwards. Participants for studies 1 and 2 were recruited using Amazon Mechanical Turk. In both studies, workers were required to be 'Mechanical Turk Masters'. Study 1, comparing summaries, had 25 unique participants giving 54 responses to 6 different tasks. Study 2, rating extracts, had 27 unique participants giving 54 responses to 6

different tasks. There were 38 participants in total across both studies, each participant could not complete the same task more than once. The average approval rating for workers was 98%; this is based on an average of 3 previous tasks for the department.

In Study 3 we attended a workshop with social science researchers interested in using social media analysis in their work. As well as discussing the tool, we collected handwritten feedback from 9 of the 15 attendees.

9.3.2 Summary Styles

In the upcoming sections, reference is made to a range of summary styles used in evaluation tasks. These are described in the following list:

- **Stock**: A summary generated using an implementation of the approach described by Nenkova et al. [25]. These Stock summaries were used as a benchmark for those generated by our tool. There is no structure to these summaries. They are generated by selecting sentences that reference prominent words not already included in the summary. The resulting distribution of words in the summary is close to that of the input text.
- **Plain**: A collection of point extracts in the same style as a Stock summary. The extracts are still grouped into sections, only this is not made clear in the presentation. Plain summaries were designed to be as close as possible to the Stock summary presentation and were also of equal length to the benchmark Stock summaries.
- **Layout**: A summary adds explanatory text that introduces different sections of points. The extracts are the same as those in the Plain summary though the explanations increase the overall word count. For a fair evaluation, these are compared against longer Stock summaries.
- **Formatted**: A Layout summary with explanation keywords in bold and topic words in green. The content is the same as a Layout summary.

9.3.3 Study 1: Summary Comparison

In order to address **H1** and **H2**, we designed a questionnaire that compared various summary styles against equal-length Stock summaries. Each questionnaire was made up of three sections. Please see Appendix F for an example section comparing two summaries.

The first section compared a Plain and Stock summary on the same topic. Users were instructed to read both summaries and rate them relatively on the following factors:

- Content Interest / Informativeness (The summary presents varied and interesting content)
- Readability (The summary contents make sense; work without context; aren't repetitive; and are easy to read)
- Punctuation & Presentation (The summary contents are correctly formatted as sentences, punctuation, capital letters and have sentence case)
- Organization (Related points occur near one another)

Finally they were asked to give an overall rating and justify their response using free text. The following section asked the same questions but instead compared a Layout summary against a Stock one - this second comparison was for a different topic to ensure

the content was unseen. Layout summaries are longer because of the explanatory text. To account for this, they were compared against a longer Stock summary than the one used in the Plain comparison. The final section compared a Layout summary with a Formatted one. The Layout summary is reused from the second section, the Formatted summary has the same content — only different formatting. There was only an overall rating and justification for the comparison in this section.

There were six versions of the questionnaire set up in a Latin square to cover all six topics. Section 1 had a different topic from section 2 so as to not repeat summary content. Section 3 addresses the question of formatting only, and has the same content as the Layout summary in section 2 which makes the task shorter for participants.

Responses from the first section, comparing our Plain summary against a Stock one, can be used in addressing **H1**. The difference in the responses gathered in section 2 will allow us to address **H2**. Section 3 expands on this, to see if the formatting was a useful addition to the Layout summary.

9.3.4 Study 2: Extract Comparison

To investigate the performance of the tool in greater depth we ran a further study testing the quality of our extract selection mechanism. Responses from this questionnaire addressed **H3**. See Appendix G for an example section from a Study 2 survey.

The task given to participants had two sections. First were three sets of extracts for three different point patterns. Participants were asked to rate extracts accounting for their succinctness and the extent to which they made sense. This design was a balance between matching the bigram model and being easy to understand. The following section, similar to the first study, compared two summaries (see Appendix F). Both summaries were of the Formatted style, however this time their content differed. The extracts in one summary were selected using the bigram model, the extracts in the other were selected at random from the same clusters of points. Participants were asked to give these a relative rating and justify their response.

9.3.5 Study 3: Section Comparison

This study was completed as a paper exercise with workshop attendees in a practical session — before the approach was discussed. Participants were given side-by-side summary comparisons, similar to Study 1 (Appendix F). However, only qualitative feedback was requested. Given two comparisons of Layout and Stock summaries on two different topics, participants were prompted to:

> *Write your comments about the two summaries. Feel free to say anything that strikes you, but particularly address:*
>
> - *Which you prefer, and why?*
> - *Do you like the structuring into sections in one of the summaries?*
> - *Which sections are most interesting to you?*
> - *What other sections might be useful to you that are missing?*

Responses from this task were intended to answer **H4**. We also introduced two new summaries: ‘Before Referendum’ & ‘Referendum Day’ based on tweets referencing

the #indyref tag or mentioning one of the campaign accounts for those date ranges. One of these summaries made up the second comparison for each participant. The first comparison was selected from either: *Abortion*, *Gay Rights*, *Existence of God* or *Gun Laws*. These four were selected from the six available as they had all summary sections complete.

9.4 Results

9.4.1 Study 1: Summary Comparison

This first study made comparisons between three pairs of summary types: Plain vs. Stock; Layout vs. Stock; and Layout vs. Formatted. The results of each are reported below.

Plain vs. Stock Comparison

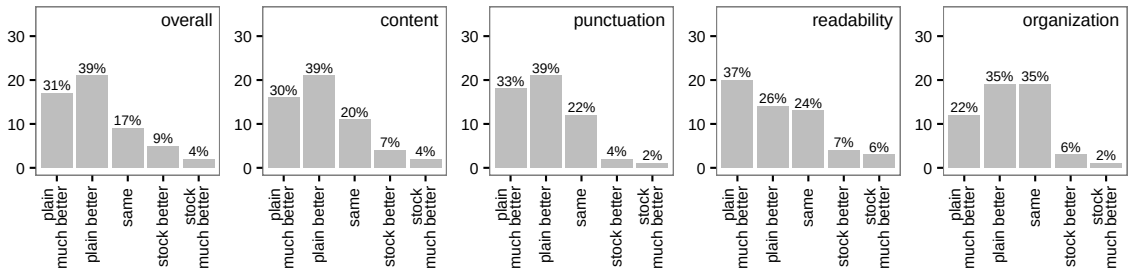


Figure 9.1: Counts of participant responses when comparing of Plain & Stock.

All of the five comparison factors presented in Figure 9.1 show a preference for our Plain summaries. These counts are aggregated from all Plain vs. Stock comparisons for all six debate topics. Each histogram represents 54 responses for a question comparing the two summaries on that factor. The results were tested using Sign tests for each comparison factor. 'better' and 'much better' were aggregated, 'same' results were excluded. The family significance level was set at $\alpha = 0.05$; with $m = 5$ hypotheses - using the Bonferroni Correction (α/m); gives an individual significance threshold as $0.05/5 = 0.01$. All factors, including both 'readability' and 'content', were found to show a significant difference, see Table 9.1. This enables us to reject **H1** — that our tool does not improve on the results of existing tools — with a good level of confidence.

Table 9.1: Plain vs. Stock factor comparison p values

Overall	Content	Punctuation	Readability	Organization
$p = 3.1 \times 10^{-6}$	$p = 1.6 \times 10^{-6}$	$p = 5.6 \times 10^{-9}$	$p = 2.5 \times 10^{-5}$	$p = 3.5 \times 10^{-6}$

Layout vs. Stock Comparison

Similarly, Layout was also compared against Stock on the same factors, the results of these comparisons are presented in Figure 9.2. We observed a stronger preference for Layout summaries.

To test this comparison against the Plain vs. Stock comparison, in order to address **H2**, we used a 'One-sided, Fisher's Exact Test'. Taking the 54 responses for the 'overall'

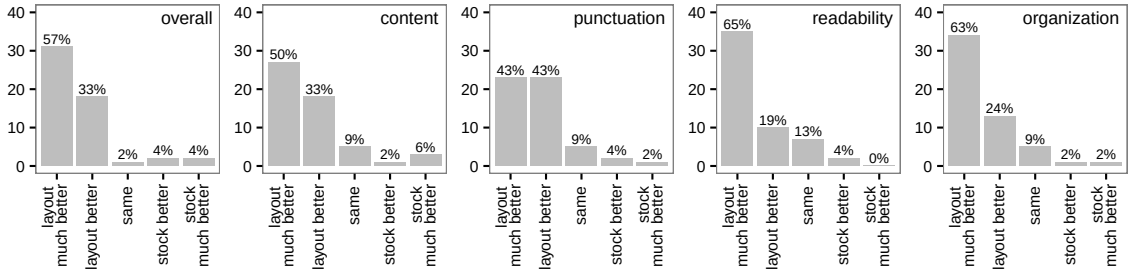


Figure 9.2: Counts of participant responses when comparing of Layout & Stock.

Table 9.2: Plain & Layout vs Stock responses contingency table

	Plain (A) vs. Stock (B)	**Layout (A) vs. Stock (B)**	*Row Total*
A much better	17	31	48
A better	21	18	39
A same	9	1	10
B better	5	2	7
B much better	2	2	4
Column Total	54	54	108

ratings from both comparisons, the test was performed on the following contingency table.

The p-value was found to be significant ($p = 0.011$). This gives cause to reject H2: that summary sections do not improve the overall readability.

Formatted vs. Layout Comparison

Finally, participants compared Formatted and Layout summaries — there was only a single 'overall' factor for this comparison. Results from the 54 responses, presented in Figure 9.3, show a preference for Formatted over Layout summaries with the same content. Comparing the results again using a Sign test (aggregating 'better' & 'much better' and excluding 'same'), showed a significant result ($p = 0.006$). This result builds on the previous comparison, showing that not only sections but other elements of presentation can improve

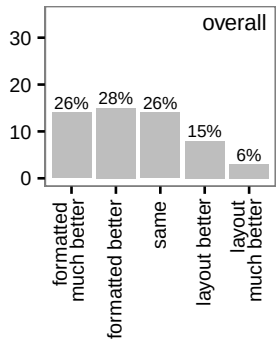


Figure 9.3: Counts of participant responses when comparing of Layout & Formatted.

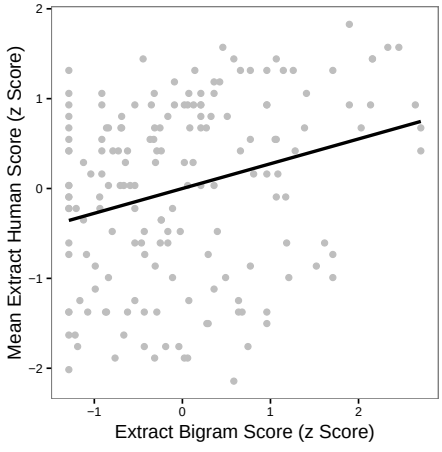


Figure 9.4: Correlation between human and bigram standard scores.

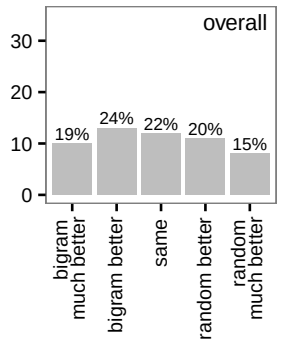


Figure 9.5: Counts of participant responses when comparing of Bigram & Random.

generated summaries.

9.4.2 Study 2: Extract Comparison

In this study participants completed two different tasks. First rating a series of extracts before comparing two summaries with different (bigram vs. random) extracts for the same point clusters.

Extract Rating

Participants scored extracts selected by our tool for display above average 73% of the time. Participants were instructed to score preferred extracts higher on a 5-point Likert scale. Extracts selected by the bigram model scored, on average, 3.93/5 while those not selected scored 3.11/5, 16% lower. Each extract was rated by 9 participants. To test the significance of this result we compared the participant scores with those allocated by the bigram model using Pearson's product moment correlation coefficient. The results of this test ($cor = 0.28, p = 0.0001$) show a significant correlation between the two scores. Figure 9.4 shows this relationship, when the bigram model assigns a good score it is also scored well by participants. While the bigram model score often conflicts with that of the study participants, there is agreement in high-scoring extracts. This is acceptable because the model is only required to select one good extract per cluster when producing summaries.

Bigram vs Random Extract Selection

In this study participants also compared two Layout summaries, one with extracts selected by the bigram model, the other with randomly selected extracts from the same clusters. This comparison showed an apparent preference for the bigram summary. 43% of participants preferred the bigram summary, 22% scored them the same, and 35% preferred the randomly generated summaries. Breaking this down by topic, the abortion bigram summary scored more than 5 times the random summary; for creation and healthcare both summaries scored equally. Interestingly, the random god summary scored higher than the accompanying bigram one. This method was intended to evaluate the bigram model at a summary level, however the results appear to vary greatly by topic and show no significant difference. To test these results we again used a one-sided Sign test. Grouping 'better' & 'much better' and excluding neutral ratings, the bigram model was selected by 23 of 42 participants, giving $p >> 0.05$. Based on this result were unable to reject **H3**, meaning that the bigram model for extract selection is not significantly better than random.

9.4.3 Study 3: Section Comparison

This final study was intended to answer questions on the utility of different summary sections. We received comments from 11 of 15 participants at the workshop, each comparing two pairs of summaries. 14 out of 22 comments, expressed a preference for a summary, 11 choosing the Layout summary and 3 choosing the Stock summary. Three different attendees marked the disagreement section as being the most interesting. Reference was not made to sections on questions or 'longer form points'. While the comments provided

input useful for future work, there were too few to use in testing **H4**.

9.5 Discussion

From our results, it is clear that both Plain & Layout point based summaries were preferred across all factors. Formatted summaries also improve on this overall. When selecting extracts, despite finding a correlation between the bigram and human scoring, bigram summaries did not score significantly better than those with randomly selected extracts for the same point clusters. While workshop comments suggested that the disagreement section was the most useful we were unable to test our final hypothesis.

In our questionnaires we solicited free textual feedback as justification for responses to rating questions. Participant's comments are referenced in the following sections discussing our results.

9.5.1 Plain vs. Stock

In the first study, Plain summaries were significantly preferred over Stock summaries. Participant's comments made reference to the comparison factors and comments align with scores allocated. Comparisons that were not captured by the factor ratings were also interesting. Multiple comments made reference to the following: Plain summaries have fewer questions, less surplus information and more content. Comments also made reference to Plain summaries being well written with "proper English" and "complete sentences". This is particularly interesting given that the points extraction process breaks down sentences into points before attempting to rebuild them as shorter sentences for presentation as 'point extracts'.

There were also five comments that suggested the participant believed the summaries had been written by a human. Participants were not explicitly told otherwise, however the concept of automated summarization is referenced in the opening question "*...could these online debates be summarized automatically?*". Such comments included: "As I read [STOCK] I felt that I knew the opinion of the writer. With [PLAIN] I could not tell." and "...clearer where the author is coming from". References were also made to higher level properties of summaries such as "logical flow", "relies on fallacy", "explains the reasoning" as well as factual correctness. Participants were however told that summaries are "*not intended to present a structured argument*".

In summary, based on their comments, participants acknowledge succinctness, variety and informativeness of the Plain summaries. This positive result, and feedback from the comparison, show the effectiveness of the points extraction approach. Points can form better summaries, even without section explanations and clear grouping.

9.5.2 Layout vs. Stock

Layout summaries were preferred significantly more than Plain summaries over Stock. References to organization doubled in comments for this comparison. Readability and the idea of assimilating information were also common factors cited in justifications for choosing Layout summaries.

Interestingly, only one comment made a direct reference to 'categories' (sections) of the summary. There was another comment that suggested the related points were

distracting. We had however expected a greater number of references to summary sections. There were fewer comments in this comparison that suggested the participants believed the summaries to be written by a human. This is perhaps because sections hint at a more structured and mechanized approach.

9.5.3 Layout vs. Formatted

While adding formatting to Layout summaries showed a significant improvement, this comparison was more divisive. The highlighting of topics and keywords received varied comments. Those who liked the formatting suggested it was easier to read quickly, those that disliked it claimed it was distracting and unnecessary. Positive comments such as "Highlighting important words makes the text easier to follow; more aesthetically pleasing; and simpler to read." and "Having keywords highlighted allows me to be able to quickly identify and distinguish between the aspects of the topic." suggest that the participants were able to make use of the formatting. However, comments like: "The keywords are bold too often and become a distraction to the reader" and "I don't think it is necessary to make the keywords and topic words bold" give reasons to remove the additional formatting.

Interestingly, there were also multiple comments that suggested participants did not notice that *only* formatting changed, despite this being communicated in the task instructions. Also of note was the multiple times participants state that their comment was their opinion, this was not common in feedback for other comparisons and shows this was a more subjective comparison.

9.5.4 Extract Rating

The results from bigram and participant extract scoring show a correlation (Figure 9.4). It was interesting to see the different patterns in the two styles of rating. The bigram model will assign a zero score if an extract does not feature any of the repeated bigrams. Participant scores were skewed towards the higher end of the scale with 5/5 being the most common score.

High scoring extracts (under the bigram model) were also scored well by the study participants. This is appropriate, and a positive result, as only one extract need be displayed in summaries for a chosen cluster.

9.5.5 Bigram vs. Random Extract Selection

Comments from this comparison referenced the fact that both summaries made the same points/reasoning/arguments. This is interesting as it means participants noticed the same points being made despite being represented by different extracts. Participants that preferred the bigram summary suggested that it was more concise while the random summary contained too much information. Longer extracts, often providing additional information, are made less likely by the length weighting in the bigram model. Participants that preferred the random summary cited that explanations were more complete and that the bigram summary was too simplistic.

As with previous comparisons, some comments gave the suggestion that the participants believed the summary to be written by a human. For example, "Side [RANDOM] uses the scientific method and doesn't involve a fake deity at all in the science section." and "[BIGRAM] is definitely college level.". Such comments suggest that there is a high

standard for summary depth.

In summary, the results and comments from this comparison appear to suggest that the detail in longer points is valuable if the point is well formed. This makes a case for reducing the effect of length weighting and including a readability factor when selecting extracts. Another possible reason for this result is that poor quality extracts are reliably rejected. This means that the group of extracts available for selection by either model (random or bigram) are already largely acceptable.

9.5.6 Section Comparison

Even with the workshop feedback — where we specifically asked for comments regarding the value of individual sections — we would like to have had a greater number of relevant comments to reference when discussing the value of individual sections. Comments from studies 1 and 2 tended to compare summaries as a whole and rarely made reference to any one section. In hindsight, this is something that we might have better encouraged.

The related points section was described as confusing. Even in large discussions, pairs of points that regularly co-occur are rare. This makes it hard for the approach to isolate points that 'make sense' together. This section may be made easier to understand if it was made clear that the related points were not adjacent in the source text.

Our counter points section was described as being the most useful overall. This is a positive result for argumentation as a factor in summarization. One comment also referenced a possible section that showed justification for voting choice — perhaps suggesting that additional argumentation-based summary sections would be useful. While this information gives a suggestion that the sections are not equally valuable, we were unable to gather enough targeted comments to test this against **H4**.

Other comments gathered from the workshop and other studies — not relevant to the performance of the current implementation — have given useful ideas for further work. These are discussed in the following chapter.

Chapter 10

Summary & Conclusion

10.1 Summary

In this project we have implemented a robust method for extracting points, meaningfully shorter content units than sentences. We made use of these in a summarization task by clustering points into cohesive sections. We then evaluated the effectiveness of our approach by comparing our summaries against those generated by a baseline statistical tool that uses sentence extraction.

We were to meet the project's initial goals of extracting points an formulating effective summaries. Using dependency parses, we have implemented a means of extracting more useful and complete points than those in the previous project. We also improved on the detection of counter points by expanding the approach to account for antonyms — rather than using negation terminology alone. We also implemented an approach for the identification of co-occurring points; however the results seem to be more variable and are highly dependent on large discussions where participants make many points.

We opted to present the results of the analysis as discussion summaries. Here we were able to go beyond our goals of counter/co-occurring points and include additional sections based on the points we had extracted. There are however areas that require further work. Results from our evaluation suggest that certain presentation decisions were not always popular. We also found that while scores allocated by our bigram model correlated with those of participants, this approach in selecting extracts from clusters was not significantly better than random selection (from the same clusters) when a compared at the summary level.

The results of the summary comparisons were very positive with all our summary types performing significantly better in comparisons against summaries of extracted sentences. We were able to reject our first two hypotheses. We were not able to reject **H3**, but have gathered useful information for future work on improving extract selection. We were unable to gather enough targeted feedback regarding sections to test **H4**, but again gathered much useful feedback for improving on the tool.

Comments from Study 3, at the social media workshop, suggested that the summaries fell between quantitative and qualitative analysis and that they could be made more useful to researchers by quoting the (already available) ratio of counter-point sides. Comments also suggested it would be useful to select the required summary length as well as provide

links back to the discussion source text. The disagreement section was referenced as being the most useful while participants found related points confusing.

10.2 Study Challenges & Limitations

Limitations in the approach stem from a number of factors. Firstly, we were limited by the availability of VerbNet frames. Despite being the largest collection of its kind, we found some verbs lacking the detailed frames required for fully formed points. The generic frame is a partial fix, but the (potentially useful) annotated semantics are lost for these points. Another key limitation is the required corpus size. To get sufficient matches for co-occurring points the discussion has to be very large. Corpus size also relates to the issue of performance. Currently, saving parses to the graph database adds a significant performance overhead. Ideally the Cypher queries could be run on parses stored in memory without the need for an additional service. Finally, Argumentation Mining techniques used are limited to identifying counter/co-occurring point patterns. Comments regarding counter points from the evaluation were positive; however, there is a lot more (supporting claims, counter arguments etc.) that might be done using Argument Mining on a points dataset.

While some technical decisions, such as choosing Docker to isolate services, worked well, others are less certain. It would have been interesting to test a dependency parser (CoreNLP or lighter alternative such as spaCy[1]) set up within the same service as the points extractor. Doing so would reduce the HTTP request/response overhead incurred when using a parser as an external service.

Extracting points with accompanying human-readable extracts was the most challenging task. This involved the mapping of verb frames to queries for dependency parses which took much longer than expected. Once this was in place, the point structure made the analysis used to generate the summaries relatively straightforward. Selecting negated pairs from a cluster that succinctly contradicted one another was the most challenging task in generating summaries. Designing the evaluation also took longer than planned and was conceptually very different. Supervisor input was invaluable in this exercise.

10.3 Further Work

The implementation is still the product of an experimental development process. Some components of the summary generation module have poor maintainability. Next steps include reducing the agglomerative complexity and repetition as well as establishing a level of unit testing. Clearer definitions for the re-formatting of extracts and extract selection (likely best expressed as a DSL) would also be beneficial for code readability. This ties into the corpus pre-processing and extract presentation, both areas that require a more integrated implementation.

The tool chain is currently closely coupled with the corpus. We would like to establish a less bespoke approach to make analysis of new corpora easier. One interesting direction would be a simple web application that would be capable of generating a summary for a

[1] spaCy, Multi-threaded Python NLP framework. Demo: https://spacy.io/demos/displacy

discussion of the user's choosing from *Reddit*, *Hacker News* or online forum. Using the tool in this way raises issue of the required discussion length. Currently a long discussion (upwards of 100,000 words) is required to extract a sufficient number of points to fill all our summary sections without repetition, and build a good sample of counter/co-occurring points. Shorter lists of extracted points can still be analyzed for counter-points; however, unless every counter point instance is of interest, a sufficient number of points are required to allow counts to reliably surface common pairs. Using shorter input texts would likely require the summary structure to be more flexible — generating shorter summaries for shorter discussions and contextually removing sections where information was lacking.

From the evaluation results we found that our bigram extract selection approach was not providing reliably better results than random — though it's scores correlated with those of participants. This appeared to come down to both readability and 'informativeness'. Currently extracts are weighted by the length. This was perhaps dominating the bigram score intended to represent informativeness. It would also be interesting to introduce readability as an additional factor, perhaps by incorporating an *Automated readability index*. Should the tool be made accessible to the public as a web application, there would also be the opportunity to use user rating of extracts to train a model for extract preference. The task of selecting readable, informative extracts that represent the cluster is an interesting task with various options for further work.

Currently the approach models discussions as a flat list of posts — without reply/response annotations. Using hierarchical discussion threads opens up interesting opportunities for Argument Mining using points extraction as a basis. A new summary section that listed points commonly made in response to other points in other posts would be a valuable addition. This would also be possible with discussions on *Twitter* where reply information is also available. Related to this, adding user identity information would make it possible to track a posts by the same user in discussions, and represents another interesting opportunity for further features.

Another direction to explore would be to build a graph-like representation of the discussion. Using patterns, a graph of nodes representing nouns connected by verbs as edges could be generated. Semantic annotations from the verb frames could also be included. As part of this project we experimented with a presentation of this type, an example is included in Appendix C. Related to this is the possibility of using the semantic annotations to investigate deeper abstractive summarization.

There is also potential for further work on the summary presentation. Feedback gathered in Study 3 suggested that it would be useful to present the counts for counter points. This would allow the section to not only show a difference in opinion, but to show how opinion is split. One comment suggested that the output was in between a quantitative and qualitative report, and that adding more quantitative information to the summaries would make them more useful. Workshop attendees also suggested that being able to refer back to the source text from the extracted points would also make the results more useful — again something that could be made possible with an online, interactive interface for summaries.

10.4 Conclusion

In this project we have shown that our approach is a viable foundation for summarization of discussion. We think this success can be generalized to tasks beyond summarization which also make use of text extracts. We have shown that existing tools struggle with the summarization of discursive texts, and that our summaries with sections that — designed to give a more complete overview — are preferred. However, summarization is just one use case for such analysis. Many varied applications enabling both search and discovery could be built using points extraction as a foundation.

We see this project as a step forward in the process of better understanding online discussion. Our hope is that this work can become the basis for future projects and applications. From political debate to product reviews, we see this as a starting point for the exploration of a vast wealth of ideas, arguments and information.

Bibliography

[1] Regina Barzilay, Kathleen R McKeown, and Michael Elhadad. Information fusion in the context of multi-document summarization. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 550–557. Association for Computational Linguistics, 1999.

[2] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[3] Filip Boltuzic and Jan Šnajder. Identifying prominent arguments in online debates using semantic textual similarity. In *Proceedings of the 2nd Workshop on Argumentation Mining*, pages 110–115, 2015.

[4] Elena Cabrio and Serena Villata. Combining textual entailment and argumentation theory for supporting online debates interactions. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*, pages 208–212. Association for Computational Linguistics, 2012.

[5] Giuseppe Carenini, Raymond T Ng, and Xiaodong Zhou. Summarizing email conversations with clue words. In *Proceedings of the 16th international conference on World Wide Web*, pages 91–100. ACM, 2007.

[6] Nathanael Chambers, Daniel Cer, Trond Grenager, David Hall, Chloe Kiddon, Bill MacCartney, Marie-Catherine de Marneffe, Daniel Ramage, Eric Yeh, and Christopher D. Manning. Learning alignments and leveraging natural logic. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, RTE '07, pages 165–170, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.

[7] Simon Corston-Oliver, Eric Ringger, Michael Gamon, and Richard Campbell. Task-focused summarization of email. In *ACL-04 Workshop: Text Summarization Branches Out*, pages 43–50, 2004.

[8] Marie-Catherine De Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D Manning. Universal stanford dependencies: A cross-linguistic typology. In *LREC*, volume 14, pages 4585–4592, 2014.

[9] Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. Generating typed dependency parses from phrase structure parses.

[10] Charles J Fillmore, Collin F Baker, and Hiroaki Sato. The framenet database and software tools. In *LREC*, 2002.

[11] Michel Galley, Kathleen McKeown, Julia Hirschberg, and Elizabeth Shriberg. Identifying agreement and disagreement in conversational speech: Use of bayesian networks

to model pragmatic dependencies. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 669. Association for Computational Linguistics, 2004.

[12] Debanjan Ghosh, Smaranda Muresan, Nina Wacholder, Mark Aakhus, and Matthew Mitsui. Analyzing argumentative discourse units in online interactions. In *Proceedings of the First Workshop on Argumentation Mining*, pages 39–48, 2014.

[13] Minqing Hu and Bing Liu. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 168–177. ACM, 2004.

[14] Dan Klein and Christopher D Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430. Association for Computational Linguistics, 2003.

[15] Kevin Knight and Daniel Marcu. Statistics-based summarization-step one: Sentence compression. 2000.

[16] Julian Kupiec, Jan Pedersen, and Francine Chen. A trainable document summarizer. In *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 68–73. ACM, 1995.

[17] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries.

[18] Elena Lloret, Alexandra Balahur, Manuel Palomar, and Andrés Montoyo. Towards building a competitive opinion summarization system: challenges and keys. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Student Research Workshop and Doctoral Consortium*, pages 72–77. Association for Computational Linguistics, 2009.

[19] William C Mann and Sandra A Thompson. Rhetorical structure theory: Toward a functional theory of text organization. *Text-Interdisciplinary Journal for the Study of Discourse*, 8(3):243–281, 1988.

[20] Kathleen McKeown and Dragomir R Radev. Generating summaries of multiple news articles. In *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 74–82. ACM, 1995.

[21] Kathleen R. McKeown, Judith L. Klavans, Vasileios Hatzivassiloglou, Regina Barzilay, and Eleazar Eskin. Towards multidocument summarization by reformulation: Progress and prospects. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*, AAAI '99/IAAI '99, pages 453–460, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.

[22] George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, November 1995.

[23] Simonetta Montemagni, Wim Peters, and Daniela Tiscornia. *Semantic Processing of Legal Texts*. Springer, 2010.

[24] Ani Nenkova and Rebecca Passonneau. Evaluating content selection in summarization: The pyramid method. 2004.

[25] Ani Nenkova, Lucy Vanderwende, and Kathleen McKeown. A compositional context sensitive multi-document summarizer: exploring the factors that influence summarization. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 573–580. ACM, 2006.

[26] Paula S Newman and John C Blitzer. Summarizing archived discussions: a beginning. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 273–276. ACM, 2003.

[27] Raquel Mochales Palau and Marie-Francine Moens. Argumentation mining: the detection, classification and structure of arguments in text. In *Proceedings of the 12th international conference on artificial intelligence and law*, pages 98–107. ACM, 2009.

[28] Joonsuk Park, Arzoo Katiyar, and Bishan Yang. Conditional random fields for identifying appropriate types of support for propositions in online user comments. *NAACL HLT 2015*, page 39, 2015.

[29] Dragomir R Radev, Eduard Hovy, and Kathleen McKeown. Introduction to the special issue on summarization. *Computational linguistics*, 28(4):399–408, 2002.

[30] 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 NAACL-ANLP Workshop on Automatic summarization*, pages 21–30. Association for Computational Linguistics, 2000.

[31] Karin Kipper Schuler. Verbnet: A broad-coverage, comprehensive verb lexicon. 2005.

[32] Lokesh Shrestha and Kathleen McKeown. Detection of question-answer pairs in email conversations. In *Proceedings of the 20th international conference on Computational Linguistics*, page 889. Association for Computational Linguistics, 2004.

[33] Lokesh Shrestha, Kathleen McKeown, and Owen Rambow. Using question-answer pairs in extractive summarization of email conversations. In *In Proceedings of CICLing, volume 4394 of Lecture Notes in Computer Science*. Citeseer, 2007.

[34] Krysta Marie Svore, Lucy Vanderwende, and Christopher JC Burges. Enhancing single-document summarization by combining ranknet and third-party sources. In *EMNLP-CoNLL*, pages 448–457, 2007.

[35] Peter D Turney. Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 417–424. Association for Computational Linguistics, 2002.

[36] Marilyn A Walker, Rob Abbott, and Joseph King. A corpus for research on deliberation and debate.

[37] Scott Weinstein. Centering: A framework for modeling the local coherence of discourse. *Computational Linguistics*, 21(2).

[38] Michael J Witbrock and Vibhu O Mittal. Ultra-summarization (poster abstract): a statistical approach to generating highly condensed non-extractive summaries. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research*

and development in information retrieval, pages 315–316. ACM, 1999.

[39] Liang Zhou and Eduard H Hovy. On the summarization of dynamically introduced information: Online discussions and blogs. In *AAAI Spring Symposium: Computational Approaches to Analyzing Weblogs*, page 237, 2006.

Appendices

Appendix A

User Manual

This document is also available online:
http://bit.ly/points_extraction_user_manual

This guide will list steps to complete the analysis for a single arbitrary discussion. The only software that you need to install on your computer is Docker and Docker Compose (unzip required to unzip sample corpus). Docker hosts lightweight virtual machines called containers that run each of our services.

These instructions have been tested on: OSX 10.11, Ubuntu 12 & 14 and on Digital Ocean (4GB Memory, 60GB Disk, LON1).

A.1 Installation

1. Install Docker and the Docker Compose interface. This varies depending on the host operating system.
2. Check that you can run `docker ps` and see output starting: `CONTAINER ID...`
3. Change directory into the project folder `cd project-folder-path`
4. Run `docker-compose build`, this will download all the dependencies for each of the project's services. This includes a series of operating system images and the CoreNLP framework and will take some time (allow 20-25 mins on a 20mbps connection, time also depends on the resources allocated to Docker).

A.2 Setting Up a Corpus

1. First you need to get a corpus in place to run the analysis on. This guide will talk you through using the Abortion corpus we used. Download the corpus to the `analysis_api` folder:
   `curl -L http://bit.ly/1QxG9i7 > analysis_api/abortion.zip`
2. Now unzip the downloaded corpus:
   ```
   unzip analysis_api/abortion.zip -d analysis_api/abortion &&
   mv analysis_api/abortion/**/* analysis_api/abortion/ &&
   rm -r analysis_api/abortion/5*
   ```
3. (OPTIONAL) Inspect a corpus file: `cat analysis_api/abortion/post_1`. Lines like `#key=value` are parsed into metadata. These are optional.
4. It is now time to start a console in the `analysis_api` service. To do this run:

`docker-compose run analysis_api /bin/bash`.

5. You will now have a new prompt `/app#`. The current directory is `analysis_api`, file changes are synced between this container and the host. Type `ls` and you will see the contents of the `analysis_api` folder.
6. Before extracting points from the corpus we need to clean the posts for invalid characters and parse any metadata. Run `ruby clean.rb abortion` to do this for all of the files in the raw abortion corpus we downloaded.

A.3 Extracting Points

1. You are now ready to extract a list of points from the corpus. To do this run `ruby collector.rb abortion`. This will take around 10-15 mins and is quite an intensive task. Output is written to `abortion_points.txt` in the `analysis_api` directory. This will be a large file (approx. 50mb), The first line is a list of topics and the following lines represent each point in JSON.
2. When you have finished running the points extraction process you can exit the analysis_api console with `exit`.
3. Note: the CoreNLP service will sometimes run out of memory or fail to respond. In this case you will need to run the task again adjusting the code to start from the failed post. You can stop all currently running containers with `docker stop $(docker ps -a -q)`. If your computer is low on RAM you may also wish to try reducing the 2gb allocated to Neo4j and CoreNLP in the docker-compose.yml file.

A.4 Cleaning/Curating Points

1. To prepare the list of points for use in summarization they must first be reformatted. First you need to move your extracted points file into the `curator` directory. From the project root run:
   `mv analysis_api/abortion_points.txt curator/abortion_points.txt`
2. From the project root directory run `docker-compose run curator /bin/bash` to get a console ready to run the curation task.
3. To clean the list for summarization run:
   `go run main.go abortion_points.txt > abortion_points_clean.txt`
4. Type `exit` to leave the curator service.

A.5 Generating Summaries

1. First get a clean list of points to use in generating the summary. From the project root run:
   `mv curator/abortion_points_clean.txt summarizer/abortion_points_clean.txt`.
2. Now run: `docker-compose run summarizer /bin/bash` to get a console prepared for generating summaries.
3. To generate a summary for your clean list of points run: `ruby summarizer.rb abortion_points_clean.txt`
4. This will save a file in the `summarizer` directory called `abortion_formatted.html`. This is the end result and should be viewed in a browser.

Appendix B

Maintenance Manual

This document is also available online at:
http://bit.ly/points_extraction_maintenance_manual

This document is intended to describe how the different components may be adjusted in future work. Some features and potential adjustments are described using references to key sections of code.

B.1 Points Extraction

Extracting points as part of generating a summary for a debate is described in the user manual. This however makes use of the `analysis_api` to collect points. This could be seen as an example of how one might use the `points_api` service. This section of code in the `collector.rb` file in the `analysis_api` folder is where points for text are extracted.

```
# specify where the points_api is running
uri = URI('http://points_api:4567/')
# create an http client to make the upcoming requests
http = Net::HTTP.new(uri.host, uri.port)
# for each of the posts in the discussion corpus, extract the points.
posts.each_with_index do |post, index|
  # prepare a request object
  # [post text, topics of interest and required point attribute keys]
  query = { text: post["content"], topics: topics, keys: %w(string
  ↪  pattern) }.to_json
  # make the request against the points api
  req = Net::HTTP::Post.new(uri)
  req.body = query
  # parse the response from the points api
  data = JSON.parse(http.request(req).body)
  # write each point in the response to the points file
  data.map { |p| out_file.write("#{p.merge(post).to_json},\n") }
end
```

This shows that extracting points with the points api is a simple request/response process. This excerpt from the points_api shows in part how requests for points are

handled. The input text is split into sentences (each with a dependency parse) and these are saved in turn to the Neo4j database.

```
# save each group of sentences
sentences.each_slice(5).each do |group|
    # Generate a Cypher query and execute it to save them all to the
    ↪  database
    query_string = neo4j_client.generate_create_query_for_sentences(group)
    neo4j_client.execute(query_string)
end
# find all the verbs that match one or more of their frames.
matches = PointsExtraction.matches_for_verbs(neo4j_client, frames,
↪  frame_queries)
# upgrade matches to points with extracts, collect points.
points += PointsExtraction.points_for_matches(neo4j_client, matches,
↪  data['topics'], data['keys'])
```

B.2 Summary Customization

It is also likely that the summary content will need to be adjusted. The key files used in generating summaries are `summary.rb` and `template_formatted.html.erb` in the summarizer folder.

Summaries are generated by calling `build`. This section below shows how this method generates each section of the summary in turn. Removing one of these lines will stop that section from being generated.

```
@counter_points = generate_counter_points; print "."
@related_points = generate_related_points; print "."
@negated_points = generate_negated_points; print "."
@common_points = generate_common_points; print "."
@longer_points = generate_longer_points; print "."
@commonly_discussed_topic_points =
↪  generate_commonly_discussed_topic_points; print "."
@multiple_topic_points = generate_multiple_topic_points; print "."
@question_points = generate_question_points; print ".\n"
```

If a section is removed it must also be removed from the summary template. Sections in the template look like this and should be removed if the summary content has been adjusted.

```
<p>Points about <strong>multiple topics</strong>:</p>
<% @summary.multiple_topic_points.each do |point| %>
  <blockquote>
    <p><%= Presenter.format(point["String"], @summary.topics) %></p>
  </blockquote>
<% end %>
```

Summaries can also be customised in other ways. When initializing a summary with `Summary.new(title, points, topics, point_count)` we can see that a title ('abortion'), list of points, topics, and a point_count is required. Increasing the `point_count` will create longer summaries.

B.3 Project Source Listing

This section lists all top level files and folders as well as sub folders containing source code. Some files are omitted in the interest of brevity.

analysis_api

This service makes used of the points api to collect a list of points for an entire discussion.

- `Dockerfile`: This is a file that contains the specification for an container to be used in collecting points for a corpus. This is used by Docker Compose to build the container image. It is built on the base Ruby docker image.
- `clean.rb`: This is a script for cleaning a corpus of posts. The key line: `content.chars.select(&:valid_encoding?).join` forcibly removed invalid characters. This file also parses post metadata lines (#key=value) and saves the input files as JSON.
- `collector.rb`: This should be seen as a client for the points_api. This script iterates the post files in the specified corpus directory and uses the points_api to collect points for each post in turn.

corenlp_server

This is a shell representing CoreNLP image as used as a service by other components.

- `Dockerfile`: This is a more complex, custom Dockerfile for the CoreNLP service. Used via docker-compose and based on the Java base image this file specifies how the CoreNLP server environment should be configured for use by other services.

curator

This single program service implements points curation.

- `Dockerfile`: This simple docker file specifies a Golang environment to run the points curation task.
- `main.go`: This file implements the entirety of the points curation task. It's process is described in detail in 7.

docker-compose.yml

This file is used as the basis for docker-compose in the project. The contents represent all local provisioning settings for services and how they are connected to one and other in YAML syntax. For example, we can see that the analysis_api is connected with `links:` to the points and topic API — as diagrammed in Chapter 5. This file also specifies the locations of all Dockerfiles for each service in the project.

docs

This folder contains the project's documentation source.

- `presentation`: This folder contains the slides used in presenting the tool at the social media workshop in Study 3.
- `report`: This folder contains all files used to generate the project report.

evaluation

This folder contains all resources used or generated as part of the evaluation.

- `evaluation_sheet`: This folder contains the evaluation forms used in Study 3.
- `extract_comparison (RESULTS)`: This folder contains the results from Study 2 in CSV format as exported from Amazon Mechanical turk.
- `summary_comparison (RESULTS)`: This folder contains the results from Study 1 in CSV format as exported from Amazon Mechanical turk.
- `extract_ranking.rb`: This is a script that contains the same bigram ranking code used in generating summaries. It can be used to get scores for a group of extracts.
- `process_extract_comparison.rb`: This is a general purpose script to process the results from Study 2. It prints the results in a variety of formats.
- `process_summary_comparison.rb`: This is a general purpose script to process the results from Study 1. It prints the results in a variety of formats.
- `r_analysis/`: This folder contains the r script files used to generate the graphs and significance figures used in the results section.
- `study3_comments.txt`: This file contains the typed comments from the Study 3 participants.
- `survey/`: This folder contains a series of files that were used to generated the questionnaires for Studies 1 & 2. These process summaries as exported by the summarizer into HTML that is ready for use on Mechanical Turk's HIT interface.

points_api

This is the core system service and is largely reusable. It is implemented as a simple web service that, for a request containing text, returns the results of points extraction analysis.

- `Dockerfile`: This defines the configuration required to run an instance of the points API. This includes installing the dependencies from the Gemfile.
- `Gemfile`: The Gemfile lists the requirements for this service. The points api is implemented as a basic Sinatra application and makes used of the Sinatra microframework. It also requires an XML parser (nokogiri) and the Ruby Neo4j wrapper.
- `app.rb`: This file implements the core of the points extraction service. It defines a root route that that accepts requests for points analysis. It splits the text into chunks and extracts sentences before triggering the points analysis.
- `frame_queries`: This folder contains all the frame queries in Cypher for each of the frame patterns (`NP VERB NP` etc.). The folder also contains variations of queries for copula verbs and the generic frame query.

- `groups.json`: This is a JSON representation of the VerbNet database. It is only used for inspecting verb classes when debugging.
- `lib`:
  - `corenlp_client.rb`: This is a wrapper used in requesting dependency parse information from the CoreNLP service.
  - `frame.rb`: This class represents a frame. Frames loaded from the index are used to initialize instances of this class. This exposes a number of helper methods used in processing frames and points.
  - `neo4j_client.rb`: This is a wrapper for the Neo4j services and is used to execute queries and fetch results using a simplified interface.
  - `node.rb`: This class extends the Neo4j::ActiveNode class and defines the structure that should be used to save tokens in the Neo4j database. This includes attributes and dependency relations to other nodes.
  - `points_extraction.rb`: This is the central file to extracting points from text. This is called from the application twice, first to get the verbs that have valid frame matches from in the database and then, for each of these, to get the extract information for the match as a complete point, ready to send in the response.
  - `relation.rb`: This class extends the Neo4j::ActiveRel class and defines the structure of a dependency graph edge. These objects only have a single label attribute and connect two token nodes in the graphs.
  - `utils.rb`: This implements a series of helper functions used in the extraction of points.
- `tasks`:
  - `parse_verb_net.rb`: This is a script used in parsing the XML information from VerbNet. VerbNet is not stored in its original form in the project source but is available for download here:
    http://verbs.colorado.edu/verb-index/vn/verbnet-3.2.tar.gz
- `verbs.json`: This is the verb index. It is loaded by the application and used to lookup frame each verb found in the input text.

stock_summarizers

This folder contains other summarizer implementations used in evaluation.

- `summarizer_topic.py`: This is the implementation of the 'Stock' summarizer used in the evaluation that was adjusted to run as part of the summary generation process. Originally sourced from
  http://homepages.abdn.ac.uk/advaith/pages/teaching/NLP/practicals/Practical3.zip

summarizer

This folder contains files for the summarizer service, the service that processes a list of extracted points into HTML summaries.

- `Dockerfile`: This defines the Ruby image required to run the summarization task. As part of the build process the requirements from the Gemfile are installed.
- `Gemfile`: This lists the requirements for the summarizer service. This service makes use of `differ` — a string diffing library — and `levenshtein`, a library for calculating the Levenshtein distance of two strings. The debugger Pry is also required when debugging the service.
- `antonyms.json`: This is a key value index of all antonyms used in the counter point analysis. This is sourced from the Wordnet NLTK corpus. Each word is a key, each value is a list of its antonyms.
- `condense.rb`: This module implements a means of merging strings that are similar. Making use of the differ gem, this can take a pair of strings e.g. "*Abortion is not a right*" and "*Abortion is a right*" and output: "*Abortion is* ***{****not****}*** *a right*".
- `counters.rb`: This file implements the search process for both counter and negated points.
- `curator.rb`: This implements the selection of the best extract for a list of points. This was tested in Study 2. `select_best`, given a list of points, will return the highest scoring one from the bigram model.
- `paragraphizer.rb`: This file is was used in the generation of the Plain summary style for the evaluation. It is not used in the current summarizer implementation to keep the basic analysis process as simple as possible.
- `presenter.rb`: This module implements the clean and format methods. Given a string such as "*, abortion is a right -* " it will output "*Abortion is a right.*". Every extract is cleaned using the methods defined here before being included as part of a summary.
- `related.rb`: This module implements the identification of related points.
- `summarizer.rb`: This script brings all the other modules in this service together and uses them to analyze points and produce a summary from them.
- `summary.rb`: This class implements the summary object. The summarizer uses an instance of this class to build a summary before passing it to an ERB template and writing an HTML summary to disk.
- `template_formatted.html.erb`: This is the template used specify the format a summary object should take when presented in HTML. Given a summary object, this fills a series of sections to complete the HTML summary.
- `utils.rb`: This file contains a utility function for collecting a key value representation of the duplicates in a list. This is used in the summary class.

topic_api

This folder contains the files that represent the topic api, a service that wraps the third-party LDA topic modeler.

- `Dockerfile`: This defines the docker image used to run an instance of the topic api. This sets up a base Ruby installation and installs the project gem dependencies.
- `Gemfile`: This lists the three service dependencies. Similar to the points api, this

service is also implemented as a Sinatra application. It also requires a Ruby wrapper to a C LDA (Latent Dirichlet allocation) implementation and the Pry debugger.

- `app.rb`: This file implements the topic api service. It exposes a route that can accept requests containing a body of text. From this it runs the LDA analysis and returns the topic words as a response.

utilities

This folder contained other experimental implementations. Currently it only includes the graph representation plotter.

- `plotter`:
  - `graph.html`: This is a template for the graph representation of points as seen in Appendix C.
  - `plot.rb`: This script takes a list of point patterns and uses the graph HTML template to build a resulting HTML file.

Appendix C

Discussion Graph Representation

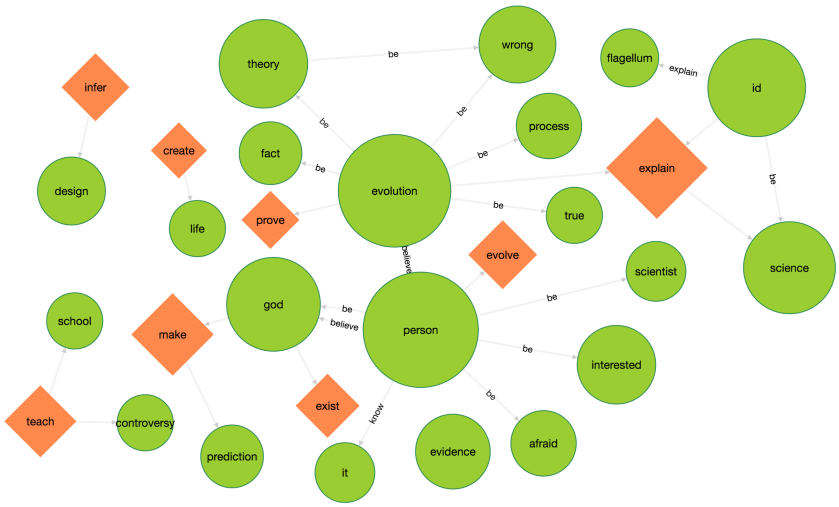


Figure C.1: A sample graph representation of the abortion discussion

Appendix D

Blacklists

D.1 Ambiguous subjects

it, that, this, which, what

D.2 Disallowed Person Actions

The following verbs are not allowed in a 2 component point with a `PERSON.nsubj`:

agree, argue, ask, begin, believe, believe, call, care, change, close, come, come, continue, debate, disagree, end, explain, fail, feel, feel, find, follow, get, go, go, guess, happen, hear, leave, live, lose, make, move, object, open, read, realize, refer, show, sit, speak, stand, start, support, take, talk, tell, think, try, understand, wonder, write

D.3 Disallowed Points

`PERSON.nsubj be.verb` cannot be completed by: *able, aware, correct, false, favor, glad, good, here, interested, likely, one, right, say, sorry, sure, true, willing, wrong*

`PERSON.nsubj want.verb` cannot be completed by: *have, what, what do*

`PERSON.nsubj` cannot be completed by: *say.verb what.dobj, mean.verb what.dobj, know.verb what.dobj, believe.verb what.dobj, see.verb what.dobj, see.verb argument.dobj, have.verb problem.dobj, tell.verb they.dobj, think.verb what.dobj, argue.verb in.prep fact.dobj, argue.verb with.prep you.dobj*

- debate.nsubj be.verb about.dobj
- question.nsubj be.verb
- make.verb claim.dobj
- ask.verb yourself.dobj
- thing.nsubj happen.verb
- something.nsubj happen.verb

Appendix E

Evaluation Questionnaire Structure

Study 1 Questionnaire (×6 variations in Latin Square)

Section 1 Plain vs. Stock (see Appendix F)

Order switches (3× Plain/Stock, 3× Stock/Plain)

Plain Summary (Topic A)	Stock Summary (Topic A)

4 factor ratings; overall rating & free text comment

Section 2 Layout vs. Stock

Order switches (3× Layout/Stock, 3× Stock/Layout)

Layout Summary (Topic **B**)	Stock Summary (Topic **B**)

4 factor ratings; overall rating & free text comment

Section 3 Layout vs. Formatted

Order switches (3× Layout/Formatted, 3× Form./Lay.)

Summaries share content, only differ in formatting

Layout summary reused from Section 2

Layout Summary (Topic **B**)	Formatted Summ. (Topic **B**)

Overall rating only & free text comment

Study 2 Questionnaire (×6 variations)

Section 1 Rating Extracts

Extract Set (Topic A) **(×3)** (see Appendix G)

Rating 1/5 per extract (approx. 10 extracts)

Section 2 Bigram vs. Random

Bigram Summary (Topic A)	Random Summary (Topic A)

Overall rating only & free text comment

Study 3 Questionnaire (×2 variations each, ×6 variations total)

Participants had **two sheets**, each with different topics.

Section 1 Layout vs. Stock

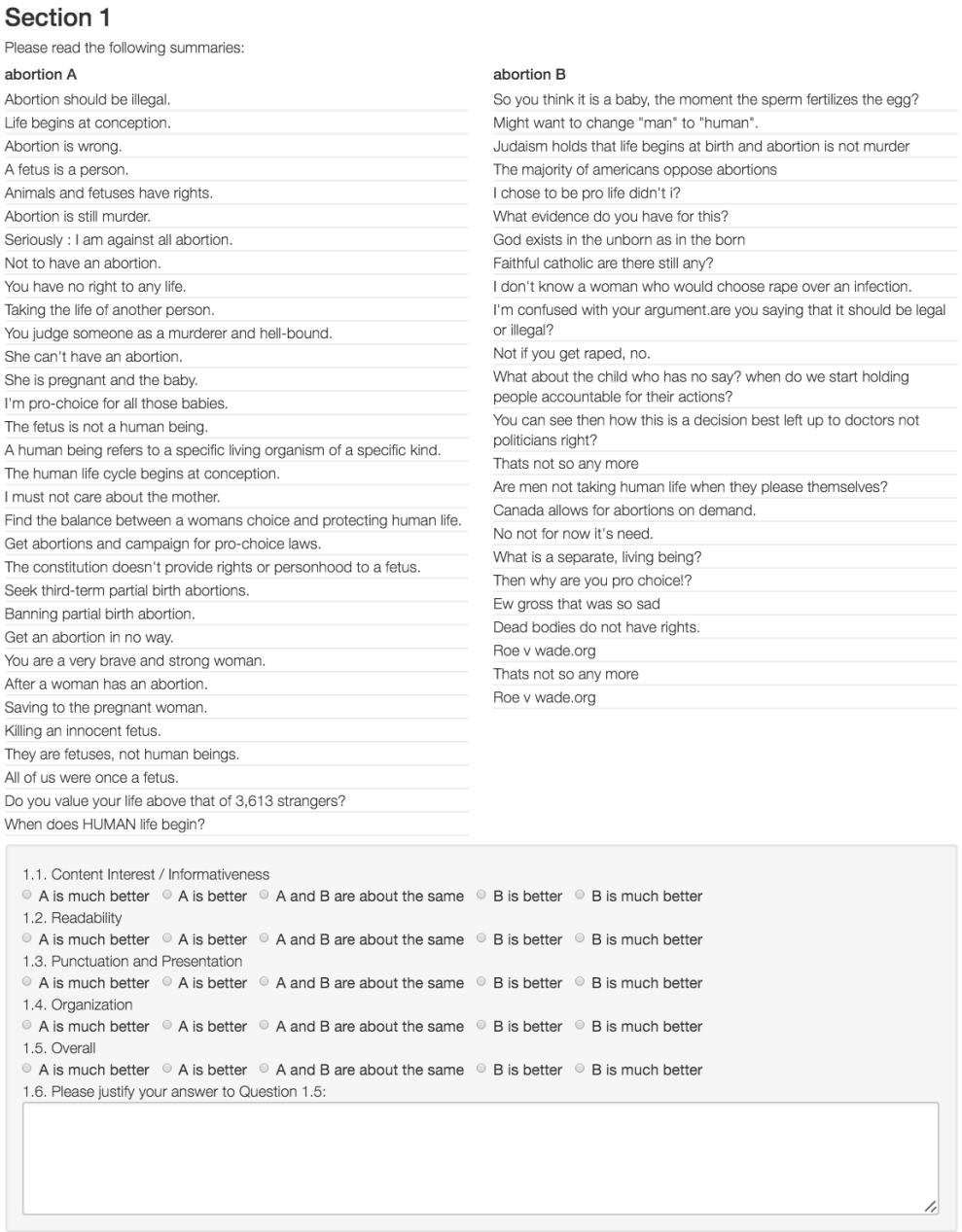
Layout Summary (Topic A)	Stock Summary (Topic A)

Free text comments only

Appendix F

Summary Comparison Survey Section

Figure F.1: An example study 1 survey section comparing a plain and stock survey.



Section 1

Please read the following summaries:

abortion A	abortion B
Abortion should be illegal.	So you think it is a baby, the moment the sperm fertilizes the egg?
Life begins at conception.	Might want to change "man" to "human".
Abortion is wrong.	Judaism holds that life begins at birth and abortion is not murder
A fetus is a person.	The majority of americans oppose abortions
Animals and fetuses have rights.	I chose to be pro life didn't i?
Abortion is still murder.	What evidence do you have for this?
Seriously : I am against all abortion.	God exists in the unborn as in the born
Not to have an abortion.	Faithful catholic are there still any?
You have no right to any life.	I don't know a woman who would choose rape over an infection.
Taking the life of another person.	I'm confused with your argument.are you saying that it should be legal or illegal?
You judge someone as a murderer and hell-bound.	Not if you get raped, no.
She can't have an abortion.	What about the child who has no say? when do we start holding people accountable for their actions?
She is pregnant and the baby.	You can see then how this is a decision best left up to doctors not politicians right?
I'm pro-choice for all those babies.	Thats not so any more
The fetus is not a human being.	Are men not taking human life when they please themselves?
A human being refers to a specific living organism of a specific kind.	Canada allows for abortions on demand.
The human life cycle begins at conception.	No not for now it's need.
I must not care about the mother.	What is a separate, living being?
Find the balance between a womans choice and protecting human life.	Then why are you pro choice!?
Get abortions and campaign for pro-choice laws.	Ew gross that was so sad
The constitution doesn't provide rights or personhood to a fetus.	Dead bodies do not have rights.
Seek third-term partial birth abortions.	Roe v wade.org
Banning partial birth abortion.	Thats not so any more
Get an abortion in no way.	Roe v wade.org
You are a very brave and strong woman.	
After a woman has an abortion.	
Saving to the pregnant woman.	
Killing an innocent fetus.	
They are fetuses, not human beings.	
All of us were once a fetus.	
Do you value your life above that of 3,613 strangers?	
When does HUMAN life begin?	

1.1. Content Interest / Informativeness
A is much better | A is better | A and B are about the same | B is better | B is much better

1.2. Readability
A is much better | A is better | A and B are about the same | B is better | B is much better

1.3. Punctuation and Presentation
A is much better | A is better | A and B are about the same | B is better | B is much better

1.4. Organization
A is much better | A is better | A and B are about the same | B is better | B is much better

1.5. Overall
A is much better | A is better | A and B are about the same | B is better | B is much better

1.6. Please justify your answer to Question 1.5:

Appendix G

Extract Survey Section

Figure G.1: An example study 2 survey section where participants rated extracts.