Le Pham Minh Duc

Un-polarizing news in social media platform

Master’s thesis of mathematical information technology

March 23, 2019

Author:Le Pham Minh Duc (Lê Phạm Minh Đức)

Contact information: [miduleph@student.jyu.fi](mailto:miduleph@student.jyu.fi), minhduc1993@yahoo.com

Supervisors: Oleksiy Khriyenko (oleksiy.o.khriyenko@jyu.fi)

Title: Un-polarizing news in social media platform

Työn nimi:

Project: Master’s thesis

Study line: Web Intelligence and Service Engineering

Page count: x+y (x = page count without appendices; y = page count of appendices)

Abstract: Abstract…

Keywords: Keywords…

Suomenkielinen tiivistelmä: Abstract in Finnish…

Avainsanat: Keywords in Finnish…

Preface

This is where you can write a preface for your thesis. The preface is optional so you don’t have to write it if you don’t want. Also, the glossary on the next page is optional. List of figures and tables are required if there is at least one figure or table in the thesis.

**The most important guideline** for the use of this template: look from the title list which style has been used to which section of the document and use similar styles also in the future when modifying the document. Use always styles, never do formatting outside styles.

If you notice bugs or mistakes in the template, please inform them to Antti Ekonoja ([antti.j.ekonoja@jyu.fi](mailto:antti.j.ekonoja@jyu.fi)).

Jyväskylä, March 23, 2019

Le Pham Minh Duc (Lê Phạm Minh Đức)

Glossary

NLP Natural language processing

DCOM Distributed Component Object Model  
More explanation…

C++ Shouldn’t need any explanation…

List of Figures

Figure 1. Service architecture **Error! Bookmark not defined.**

Figure 2. Test figure 25

List of Tables

Table 1. Example of the table 26

Contents

[1 INTRODUCTION 1](#_Toc4103520)

[1.1 Problems overview 1](#_Toc4103521)

[1.2 Proposed solution and research questions 2](#_Toc4103522)

[1.3 Research method 3](#_Toc4103523)

[1.4 Thesis structure 3](#_Toc4103524)

[2 HYPOTHESIS AND TECHNOLOGIES 4](#_Toc4103525)

[2.1 Our hypothesis 4](#_Toc4103526)

[2.2 Evaluation criteria 5](#_Toc4103527)

[2.3 Natural language processing 7](#_Toc4103528)

[2.4 Technologies used in this work 8](#_Toc4103529)

[2.4.1 Stanford CoreNLP 8](#_Toc4103530)

[2.4.2 Node.js 10](#_Toc4103531)

[2.4.3 Version control system, Git and GitHub 10](#_Toc4103532)

[3 UN-POLARIZING ALGORITHM 12](#_Toc4103533)

[3.1 Overall solution architecture overview 12](#_Toc4103534)

[3.1.1 Article annotation pipeline 12](#_Toc4103535)

[3.1.2 Article matching pipeline 14](#_Toc4103536)

[3.2 The first algorithm using Sentiment Analysis and Name Entity Recognition 14](#_Toc4103537)

[3.2.1 Web content processor 14](#_Toc4103538)

[3.2.2 Stanford Core NLP Annotator 18](#_Toc4103539)

[3.2.3 Sentiment analysis 20](#_Toc4103540)

[3.2.4 Named entity recognition 22](#_Toc4103541)

[3.2.5 Un-polarizing algorithm for the first prototype 25](#_Toc4103542)

[3.2.6 Weakness of the first prototype 26](#_Toc4103543)

[3.3 Second prototype with Semantic triples 27](#_Toc4103544)

[3.3.1 Open information extraction 27](#_Toc4103545)

[3.3.2 Triplet pairs, common entities and common statement counts 30](#_Toc4103546)

[3.3.3 Providing the information to the user 31](#_Toc4103547)

[3.3.4 Limitation of the current system. 34](#_Toc4103548)

[4 RESULTS EVALUATION 35](#_Toc4103549)

[5 FUTURE WORKS 36](#_Toc4103550)

[5.1 Ontologies based entity relevance 36](#_Toc4103551)

[5.2 Word-net verb contradiction 36](#_Toc4103552)

[5.3 AI based un-polarized algorithm 36](#_Toc4103553)

[5.4 Cloud service design 37](#_Toc4103554)

[5.5 User interface and user experience design 37](#_Toc4103555)

[6 CONCLUSION 38](#_Toc4103556)

[Bibliography 39](#_Toc4103557)

[Appendices 40](#_Toc4103558)

[A Title of the first appendix 40](#_Toc4103559)

[B Title of the second appendix 40](#_Toc4103560)

# INTRODUCTION

## Problems overview

(*Should write like a lot more, with more references as well*) Ideological polarization has been a problem in our society for quite a long time. (that reference from 1986). With the rise of social media (citation here or not), it’s estimated that 66% of US citizen use social media as one source of news (citation from that web page), the amplification of ideological polarization has been increasing much faster than previously due to social endorsement, and other social media techniques that is used to keeps its user engaged (Sporh. 2017). This creates the echo-chamber effects that, by the design of social networks that only show what the users want to see, make the user even furthermore polarized into his own belief and makes him see the world wrongly, which may turn the user into some extremists that might be harmful for the society.

Scholars have been researching about this problem and solutions are proposed (many citations needed) but these solutions are either too impractical (e.g: needing the giant media companies to change their entire business models) or just way out of reach of the scholar scope (e.g: needing of the government’s intervention on the issue or people to stop using the services).

https://en.wikipedia.org/wiki/Echo\_chamber\_(media)

*References:*

*Boczkowski, P., Mitchelstein, E., & Matassi, M. (2017, January). Incidental news: How young people consume news on social media. In Proceedings of the 50th Hawaii international conference on system sciences.*

[*https://scholarspace.manoa.hawaii.edu/handle/10125/41371*](https://scholarspace.manoa.hawaii.edu/handle/10125/41371)

*Beam, M. A., Hutchens, M. J., & Hmielowski, J. D. (2016). Clicking vs. sharing: The relationship between online news behaviors and political knowledge. Computers in Human Behavior, 59, 215-220.*

*https://www.sciencedirect.com/science/article/pii/S0747563216300656*

## Proposed solution and research questions

The main goal of this thesis is to find the way to break the people’s echo chamber that is mostly caused by the effect of social medias only show the user what he/she wants to see. To combat this, we need to show the user the news from the other side of view. If he/she reads about the opening of a new coal mines help creating a few hundreds of new jobs for the area, he should also know that the new coal mines will cause a great damage to the environment and might cause some local wild-life to disappear.

On top of that, the service must be accessible and easy to use, as the reason of many people using social media as their main source of news as it’s so convenience to have one place to go to and can see both your friend’s status as well as news.

With that goal in mind, the main research question of the thesis is:

* **How to find articles with alternative (different) points of view to a given article?**

We will only attempt to find the news that is relevant to the article but also provide oversight from different point of view that the first article misses. We will not check if the news is credible (but we will try to only provide news from credible sources) or if it is true, we simply provide the user different articles from many points of views about the relevant topic so that he/she can choose to interpret it whatever way he/she wants to.

With the first question answered, we will address two additional support questions on deploying the news un-polarizing service for the mass to use:

* **What is the most convenience way to deliver the service for the user?**

If the service is too complicated to use, or requires too many unnecessary steps, the user will rarely use the service, if at all, which reduces the effectiveness of the system. We need a method that can deliver the alternative points of view to the user that is most convenience for him, for example: a fully automatic system that whenever the user reads a news about a topic, he also has a snippet information of other relevant articles about the topic.

* **How to engineer the service so that it is autonomous, up-to-date and scalable?**

As a news service, it must always catch up with the latest news to be relevant to use. The service needs to read and analyze articles to from various news source all the time so it can serve the user with the latest and most relevant news. Also, as a cloud service, we need to prepare ahead of the service, so that when there are more users, the service will be able to handle that.

## Research method

(*Some good part from the book*) Abc, test text.

## Thesis structure

(*I should write this part last, because there is still more things to change*) – Including this introduction and the problem overview, this thesis will contain five parts. The next part contains our hypothesis based on theoretical research, as well as the state of the current technology and the tools we choose to go forward with the practical prototype.

The third and biggest part, named Un-polarizing algorithm will describe our step to step practical implementation, the problems we faced along the way and the reasons for our implementation decision.

The next part will test the result of our prototype using real world articles with cross human check/validation to see the effectiveness of the solution.

Finally, conclusions for the thesis as well as possible future work and extension are given in the last chapter.

# HYPOTHESIS AND TECHNOLOGIES

## Our hypothesis

Our main research questions and our hypotheses are based on this assumption:

* When a person read an article, it would be interesting and beneficial for him/her to also see other articles with the same topic(s) but from a different point of view. As having multiple view angles on a subject make the reader more informed about a problem/topic, he/she will be less likely to be affected by propaganda as well as reducing the effect of echo-chamber of social media platform, which is the news source of many people nowadays.

This assumption leads us to our main research question, which is:

* How to find articles with alternative (different) points of view to a given article?

However, the more interesting question would be:

* What does “different point of views” even means in our context, which are news and opinion?

As there are not any clear definition of what the term “different point of view” mean. To understand what it means in our context, and come up with a clear definition for it, consider this example:

* Topic: The US’s war in Iraq.
* First article main point: The US’s war in Iraq is good and justified because Saddam Hussein is a dictator and the people living under his reign are suffering.
* Other article main point: The US’s war in Iraq is bad because it furthermore destabilizes the region and the main intention of waging war was because of oil, not for humanitarian purpose.

From the example above, we came up with two different hypotheses that focuses on two main characteristics of the problem:

* Sentiment based hypothesis (more on chapter 3.2): Two articles are considered to have different point of views if two conditions are met: They both cover similar topics, and if one article has a positive view on the situation and the other has a negative view regarding the same subject.
* Statement based hypothesis (more on chapter 3.3): if two articles have contradictory or alternative facts or statements between them, they have different point of view and the reader should know about both.

However, even with these hypotheses, terms like “similar subject”, “positive/negative views”, or “alternative facts” are abstract terms and there is not any universally defined rule for finding these characteristics. Thus, we need to define our own rule for finding “Article similarity”, “Positive/negative views”, and “Alternative facts”. This leads us to our supporting hypotheses:

* Similar subject hypothesis: Two articles are considered to have similar topic if they both contains a good number of similar named entities. A named entity is defined as: a person, location, organization or a numerical expression (Grishman & Sundheim, 1996). For example, given three articles: A, B and C. Article B will be considered “more similar” to A than C to A if the number of similar named entities between B and A is bigger than the number between C and A, and vice versa. (more on chapter **3.2.4** and **3.2.5**).
* Positive/Negative views hypothesis: An article is considered to have a positive or negative view on a subject can be determined by either the sentiment value of such article or the average sentiment of all the sentences in the article, in which the subject/topic appear in (more on chapter **3.2.3**).
* Alternative facts hypothesis: if two articles have contradicting or alternative fact, they are considered to have different point of view. A fact or a statement can be defined as a semantic triple (a triplet) extracted from the article. A semantic triplet is a set of three parts that consists of [subject + predicate + object] (citation needed). Two semantic triples are considered to have contradiction or alternative information if they have two similar parts and one different part. For example, consider these two statements: “He goes to school” and “He leaves school”. Both have the same subject (He) and object (school), but different predicate (to go vs to leave), so, these two statements are considered to have alternative information. (more on chapter 3.3)

Finally, in case we could not find articles with different point of view using these hypotheses above, we came up with a term called “relevant article”, which defines news document that we think that would be interesting for the user to know and read about.

* If there is no contradiction information between the comparing article and our knowledge corpus, we suggest the most relevant articles to our user. “Article’s relevance” is calculated by both the similarity as well as the difference between the two articles (more on chapter **3.3.1** and **3.3.2**).

*Grishman, R., & Sundheim, B. (1996). Message understanding conference-6: A brief history. In COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics (Vol. 1).*

## Evaluation criteria

As we discussed in the previous chapter, term like “subject similarity” or “different point of views” are abstract term, so, there is no concrete way to evaluate these characters. We could not find any statistic or equation to evaluate the results of our algorithm. We could only judge the output by using our common sense and opinion’s survey.

To test the rigidity of our algorithm, we gathered a dataset of 78 articles (+ 8 non-readable by the web content parser), consist of three main themes:

* Muslim in Europe: 24 articles
* Muslim in Asia: 39 articles
* Asians in Europe: 17 articles

We then went through each of them and decided which set of articles are more relevant to each other and which pair of articles that contain opinion from different point of view. Once we finished annotating the database, we do the evaluation by comparing our annotation to the results from the algorithm.

With articles spanning in three different main categories that are also related to each other, for each them, there will be some positive hits (related articles) as well as false negatives: news/documents that share similar set of entity and keyword but convey different fields and are not related at all (for example: sports and politics). With these “traps”, we want to test if our algorithm can truly return the relevant information and how close the suggestion is to our annotation.

Finally, since our algorithm can read through the whole article thoroughly, it might discover interesting information that we missed as we only skimmed quickly through our database. Skimming is a realistic behavior though, since most people only read the title or consume through each news source quickly (Gabielkov, Ramachandran, Chaintreau & Legout, 2016).

*Ref:*

*Maksym Gabielkov, Arthi Ramachandran, Augustin Chaintreau, Arnaud Legout. Social Clicks: What and Who Gets Read on Twitter?. ACM SIGMETRICS / IFIP Performance 2016, Jun 2016, Antibes Juan-les-Pins, France. 2016*

## Natural language processing

**Need extra works!!! – This could be the first chapter, before the hypothesis, introduction to technology**

Peer (Liddy, 2001): “Natural Language Processing (NLP) is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human-like language processing for a range of tasks or applications”.

As our thesis require working with news document, which are usually written by human using natural language without any other statistics or properties to analyze, NLP provides a good foundation for us to proceed. Based on our hypotheses from the previous chapter, for this thesis, we will focus on three main sub-tasks covered by NLP:

* Named entity recognition (NER): is a task in Information Extraction consisting in identifying and classifying just some types of information elements, called Named Entities (Marrero et al, 2013). We use NER to identify the similarity between two articles, and then decide if we want to compare their difference in point of view or not. NER is generally considered as a solved problem as the best system entering MUC-7 scored 93.39%, compare to human 97.6% (Marsh & Perzanowski, 1998).
* Sentiment analysis:
* Open Information Extraction (OpenIE):

Making a computer fully able to understand human language have always been an interesting topic, with researches and application started as early as 1968 with SHRDLU (Terry Winograd, 1971). With are many tools, services, applications and researches for NLP that is fully available today, ranging from free open source platform to cloud service, … there are many options for us to consider. We will discuss these options and our choice in the next chapter: Technologies used for this work.

*Ref:*

*Liddy, E. D. (2001). Natural language processing.*

*Marrero, M., Urbano, J., Sánchez-Cuadrado, S., Morato, J., & Gómez-Berbís, J. M. (2013). Named entity recognition: fallacies, challenges and opportunities. Computer Standards & Interfaces, 35(5), 482-489.*

*Marsh, E., & Perzanowski, D. (1998). MUC-7 evaluation of IE technology: Overview of results. In Seventh Message Understanding Conference (MUC-7): Proceedings of a Conference Held in Fairfax, Virginia, April 29-May 1, 1998.*

*Terry Winograd, Procedures as a Representation for Data in a Computer Program for Understanding Natural Language. MIT AI Technical Report 235, February 1971*

## Technologies used in this work

### Stanford CoreNLP

Developed by the researchers at Stanford University from 2006, released as a free and open source software in 2010, with updates still being developed and released nowadays (Manning, Surdeanu, Bauer, Finkel, Bethard, & McClosky, 2014), Stanford CoreNLP is a Java (or JVM based) annotation pipeline framework for most of the common Natural Language Processing (NLP) steps like Named Entity Recognition (NER) (Finkel, Grenager & Manning. 2005), Sentiment Analysis (Socher, Perelygin, Wu, Chuang, Manning, Ng & Potts, 2013) and Open Information Extraction (OpenIE) (Angeli, Premkumar & Manning, 2015). We used Stanford CoreNLP to process raw web-based article text into annotated data and properties, ready for our “un-polarizing” algorithm. Detailed information on the role and usage of Stanford CoreNLP in our work will be presented in later chapters (chapter 3.1 and chapter 3.2.2) where we go in depth with our solution.

We chose Stanford CoreNLP as the foundation technology for our thesis because of two main reasons:

* It has all the services we needed integrated into one big package that will work well together. There are many tools that provide the necessary services (especially NER and sentiment analysis) for us, but each of them has different requirement for the input data as well as different output format. Using separated tools instead of just one require us to put time and effort into making them work together instead of focus on the main research question, which is the “un-polarizing” algorithm. We could argue that using a specialized tool for each of the task might provide better quality output, but our testing results does not show any significant different in the results outputted by these tools compared to Stanford NLP anyway (more on chapter 3.2.3, chapter 3.2.4 chapter 3.3.1).
* It is free and open-source, with full access to source code that can be installed and run locally. Having every cog in our machine (or solution) fully available is important, as the private and close-source service are subjected to changes or shut down at any moment, which, is problematic. Having our algorithm run well and not depending on services we do not control is important not only us, now, but also for when other researchers want to try or test or improve our solution, now, for 10 years from now.

We fully understand that Stanford CoreNLP is not perfect and there are better (and worse) performing tools for every NLP task we utilize in this thesis. Notable mentions are Google’s Cloud natural language[[1]](#footnote-1) or IBM’s Watson natural language understanding[[2]](#footnote-2). On later chapter where we focus on each specialized NLP task, we will provide comparison of results using other tools, and what is the hypothetical result/difference we could have for using other tools rather than using Stanford CoreNLP.

*Ref:*

*Manning, Christopher D., Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pp. 55-60.*

*Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL 2005), pp. 363-370*

*Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher Manning, Andrew Ng and Christopher Potts. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. Conference on Empirical Methods in Natural Language Processing (EMNLP 2013)*

*Gabor Angeli, Melvin Johnson Premkumar, and Christopher D. Manning. Leveraging Linguistic Structure For Open Domain Information Extraction. In Proceedings of the Association of Computational Linguistics (ACL), 2015.*

### Node.js

Even though most of the works done in this report are prototype code to demonstrate and test our hypothesis, we want to continue working on our “Un-polarizing algorithm” after this thesis work is completed. We our final goal is to produce a product for people all around the world to use and thus, help creating a better society. With that in mind, we want to choose a programming language that is capable producing quality and stable code base for longevity, performant and highly scalable, but also flexible enough for changes in our prototype development.

Node.js[[3]](#footnote-3) comes to mind as the perfect candidate for our requirements as its multi-paradigm nature and its giant ecosystem of libraries (Tilkov & Vinoski, 2010) allows quickly creation, testing and modification of our prototype with little overhead cost. Several benchmarks also prove the superior performance of a Nodejs web system when compare to other popular technologies like PHP and Python (Lei, Ma & Tan, 2014), which shows the potential of node.js for longevity and development of industrial application.

*Ref:*

*S. Tilkov and S. Vinoski, "Node.js: Using JavaScript to Build High-Performance Network Programs," in IEEE Internet Computing, vol. 14, no. 6, pp. 80-83, Nov.-Dec. 2010.*

*http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5617064&isnumber=5617049*

*K. Lei, Y. Ma and Z. Tan, "Performance Comparison and Evaluation of Web Development Technologies in PHP, Python, and Node.js," 2014 IEEE 17th International Conference on Computational Science and Engineering, Chengdu, 2014, pp. 661-668.*

*http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7023652&isnumber=7023510*

### Version control system, Git and GitHub

A version control system (VCS) is “a tool that tracks different versions of software or other content” (Loeliger, Jon, & McCullough, 2012). Using VCS is considered as one of software development best practices, even just for a personal project (Spinellis, 2005). As we are creating a software prototype to evaluate our hypothesis and algorithm, it is best to follow these principles and to use a VCS for our project. These principles are later on, proved to be quite helpful as, throughout the course of our prototype development, we found ourselves utilizing many features of VCS such as source-code backup, code synchronization between different computers, progression roll-back and, finally, through the commit messages: a diary/documentation system.

“Git” is a free “Decentralized version control system” that has a clean internal design, performs quickly and efficiently, enforces accountability (Loeliger, Jon, & McCullough, 2012), and is the VCS we chose to use for this thesis. “Git” was created in 2005 by Linus to help developing the Linux kernel as other VCSs system at that time had limitations and flaws that would make them not a viable solution. These reasons make “Git” not only a good solution to applied to our works, but also make it one of the mostly used VCS nowadays in both public and private sectors.

GitHub was chosen as our hosting service for the project as it was one of the biggest Git supporting services (hence, the name) and is free. All our software prototype, coding history, instructions and documentations are kept on GitHub and are freely available to view, access and execute at any moment from any computer anywhere in the world. An “url” to the project are provided at the end of this report.

*Ref:*

*D. Spinellis, "Version control systems," in IEEE Software, vol. 22, no. 5, pp. 108-109, Sept.-Oct. 2005,*

[*http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1504674&isnumber=32260*](http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1504674&isnumber=32260)

*Loeliger, Jon, and Matthew McCullough. Version Control with Git: Powerful tools and techniques for collaborative software development. " O'Reilly Media, Inc.", 2012.*

# UN-POLARIZING ALGORITHM

## Overall solution architecture overview

To answer the main research question of: **“How to find articles with alternative (different) points of view to a given article?”**, we developed a prototype called the “Un-polarizing algorithm”, containing two main parts:

* Article annotation pipeline: to process the natural language text from the article to machine readable format and save them to a local database for comparing later.
* Article matching pipeline: compare a given article to all the annotated articles in our database and find the most appropriate articles with relevant information with different point of views.

### Article annotation pipeline

Here is the overall architecture of the Article annotation pipeline.

Annotation’s

database

Web

content

processor

Stanford

Core

NLP

Filtering and

processing

module

Article annotation pipeline

Article’s url

Pre-processed text

CoreNLP’s annotation

Un-polarized annotation

Figure 1. Test figure

Boxes, dashed boxes and lines with different colors represent different parts and purpose of our process:

* Cyan **box and line**: open information, freely available on the internet
* Red **box and line**: Stanford Core NLP service.
* Green **box and line**: our own custom logic, implemented for this works
* Blue **box**: the article annotation process.
* Black **box**: the database to store the annotated articles

To annotate an article, we first read its content using the url, and then pre-process the website’s data so that it can be parsed for the NLP task (more on 3.2.1). We then push the news text into the Stanford Core NLP (more on part 3.2.2). From the information received from the Core NLP engine, we use 3 sets of data annotation: the sentiment analysis (more on part 3.2.3), the named entity recognition (more on part 3.2.4) and the semantic triples (more on part 3.3.1). Each dataset then will be furthermore analyzed/filtered using various equation to make it ready for the articles matching solution. Finally, the results will be recombined into one annotated object and saved into our local database

The Semantic triples and other processes related to it like semantic triples filtering are only utilized in the second prototype, where we need to go in-depth with the article and find the contradicting information.

### Article matching pipeline

Figure 1. Test figure

Local

database

Web

content

processor

Article

Annotation

pipeline

Articles scoring

and ranking based

on comparing article

Comparing

url

Most relevance articles

from different point of views

Articles matching pipeline

To find articles with different point of view to a given article, first, we need to run the article’s url through our annotation pipeline to extract the necessary information for the scoring and comparing tasks. After that, we calculate a relevant score for every article stored in our database and rank them based on this score (more on 3.3.2). Finally, we display the highest scored articles to the users as they are the most relevant. We weren’t able to implement the different point of view check in our prototype, but we have some good theoretical claims to do it.

## Web content processor and Stanford CoreNLP

### Web content processor

**TODO: FORMAT**

Our first step is to retrieve and read the news documents. However, articles on the internet are usually presented inside a web-page, with just not only the news itself, but with many related information for the web-page like html tag, images and captions, links to other news on their website and advertisement. As Stanford CoreNLP’s requirement for input is text paragraph only, we must pre-process the news content to remove the unnecessary information. We divided the data pre-processing task into two steps:



Figure 1. Left: Example news web page – Right: the web source code we received

* Strip away all other un-related content like advertisements, contact information, other stories from their network, etc. From the example above, we could see that the actual news content we want to see is presented in just half of the page (less if we also exclude the image). For this, we implemented a “web content parser” module which utilize a similar technique to reader mode on Firefox[[4]](#footnote-4) which can automatically strip away all the non-article part in the web content, using a NodeJS library called node-readability[[5]](#footnote-5). However, as this feature is intended for the user to read the news easier without all the bloated content, the html formatting tags, images and captions are still present, and this result will not work with the Stanford CoreNLP.
* Remove the HTML formatting tags and image captions. For this we wrote a small rule-based module to automatically remove the html tags, the image captions by removing any text appear inside a “< >” block, which is the standard for html tag. However, this approach will return a few faulty sentences for every article because each website will have a different layout and method to present their content, making our rule-based filtering ineffective.



Figure 1. Example with the html filtering. In this case, the word Media caption will not be filtered, but added to the next sentence. The result we have is an incorrect sentence: “Media captionPictures ….” parsed into the annotator

However, we found one other more effective way of ensuring that the sentences forwarded into the CoreNLP annotator are correct is to use a cloud service called SMMRY[[6]](#footnote-6): an article summarization tools, which can read through the article and return the sentences that it thinks contains the most important information of the article. SMMRY works by going through the whole documents, score each word based on their semantic roles and their appearance frequency in the text. It then returns the sentences that has the highest sum of all containing word’s scores.

This tool is quite effective for our case as it strip away all the unnecessary content like the html tags and sponsored contents, which provides the suitable data for the annotation pipeline. SMMRY also has a parameter to control how many percent of the news document should be reduce, so, when we set this value to 0 percent and get the full article in text form. For comparison, texts retrieve from SMMRY has a slightly higher content detection rate than node-readability and a much better <html> removal rate than our home-cooked solution.



Figure 1. SMMRY example

SMMRY, however, is not a perfect tool as there are two downsides for using:

* The sentences order in the paragraph is incorrect. As a document summarization tool, SMMRY’s main goal is to figure the most important sentences of the documents and recommend these to the user. As a result, the sentences retrieved by SMMRY are not in correct chronology order of the news article, but in the summarization order. This is, however, not a problem as Stanford CoreNLP works on a sentence basis only, and our features also do not rely on sentences index in the paragraph. We had tested the annotation on a sentence where it stands alone and when it is within a paragraph with other sentences and the results in both cases are the same, which means that CoreNLP does not considers the context in which the sentences appear in.
* This is a service from a private company, which, using it is against our arguments in chapter 2.4 for using open-source technologies only. However, as there is no good and easy to use open source alternative available, we decided to use this tool, but kept our “web content processor” module present in the code base, easily interchangeable with SMMRY for any future reference, in case SMMRY goes out of business.

### Stanford Core NLP Annotator

Glossary: API, Wrapper.

There are multiple ways to use the Stanford Core NLP as listed on their main website[[7]](#footnote-7), but it can be summed down to two main methods:

* Directly by the Java API: As Stanford Core NLP is created in Java (citation needed), we can import the whole CoreNLP as a Java library and call all the NLP function through their Java APIs.
* Indirectly through a wrapper: There are many wrappers for CoreNLP available for many common usages: command line wrapper, web-server wrapper, or many programming language wrapper libraries like C#, Python, Pearl, NodeJS …

As we are using NodeJS, here are the best two methods applicable to our usage:

* Using the webserver: this method creates a web service on a local host. This is quite useful as not only it provides all the annotating features, it also has a web interface for quick debugging and visualizing the results of the CoreNLP tool.
* Using the NodeJS wrapper: the NodeJS wrapper also has all the annotation features of the CoreNLP. However, it does not have the web interface for debugging.

We chose to use the Stanford CoreNLP as a webserver as it provides more feature but no significant there is no downside for our use case.

Continue from the previous step: pre-processing; after extracting the text document from the web article, we parse the text into the Stanford Core NLP local server to get the annotations from the article. Since Core NLP have support for many common NLP tasks, each with its own annotators (citation above), we can control which annotators to use, instead of all of them to save some processing power. Hence, for our needs, we only need three annotators: “sentiment”, “ner” and “openie”. However, as there are dependencies for our required annotators to work, here is the list of all annotators we use and their usages:

|  |  |
| --- | --- |
| tokenize | Split the text into a list token. A token could be a word, or a special character (dot “.”, comma “.”, etc). “tokenize” is required for all annotators below. |
| ssplit | Split sequence of tokens into sentences. First, the tokenize split the whole document into many smaller tokens, then, it will be combined back to sentences in this step. “ssplit” is required for all annotators below. |
| pos | Part-of-Speech (POS) tagger. This annotator assigns POS to each word in the text, such as noun, verb, adjective, etc. “pos” is required for all annotators below except “parse” |
| lemma | Generates the word lemmas (base form in dictionary) for all token in the document. “lemma” is required for “ner” and “natlog” |
| parse | Create a dependency tree for the sentence. “parse” is required for “sentiment” and “natlog” |
| natlog | Natural logic annotator: create a natural logic dependency between tokens in the texts, required for “openie” |
| ner | Named entity recognizer: recognize named entities. One of our main use cases for this thesis. |
| sentiment | Sentiment analysis: determine the sentiment value of each sentence. One of our main use cases for this thesis. |
| openie | Open information extraction: generate semantic triples from the texts. Used for the second prototype. |

With the Stanford CoreNLP running as a web server locally at port 9000 (or on the cloud), we request the annotations in json format by calling a GET request with this uri:

* *http://localhost:9000/?properties%3D%7B%22annotators%22%3A%22tokenize%2Cssplit%2Clemma%2Cner%2Copenie%2Csentiment%2Cnatlog%2Cparse%2Cpos%22%2C%22outputFormat%22%3A%22json%22%7D*

After receiving the results from the NLP engine, we apply our customized filter for all the annotations to remove all unnecessary information and reformat the result to fit with our un-polarize algorithm (more on next chapters). The filtered and reformatted results (let’s call them core feature) will be saved into the local database for future comparison calculation of the un-polarizing algorithm.

The use of the local database to store core-results is necessary, because when we try to un-polarize an article, we annotated it, then compare its core feature to every other documents’ core-feature in our knowledge corpus. Since the processing time for each article is quite long, around 10 seconds each[[8]](#footnote-8), so, it is not feasible to do all the annotation on the fly without the database.

## Sentiment based un-polarizing algorithm

### Sentiment analysis

Initially defined by (Nasukawa & Yi, 2003), the main task of “Sentiment analysis” is: “to identify how sentiments are expressed in texts and whether the expressions indicate positive (favorable) or negative (unfavorable) opinions toward the subject”. Since then, there have been a numerous improvement on implementing this task, from manually defined the sentiment value for each word (Nasukawa & Yi, 2003), to a classification model based using open database (citation needed), to using semantic relation and tree thing (Stanford citation). Even the industry sector is also interested in this field as the tech giant are also providing their own solution like Google (citation), IBM (citation), Microsoft (citation) and more …

However, with so many resources putting into them, sentiment analysis still is considered as an un-solved problem as recent benchmark show of only 40% succession rate even for the best tools out there (citation needed). Fortunately, this is not a problem for us, as even with such a low accuracy, we believed that sentiment analysis could be applicable to our use-case, as news and articles are usually conveyed in straight forward manner whereas most of the failed cases for sentiment test are from normal conversations with tricky word order like double negation (ie: This product is not bad) or sarcasm (Yeah, I love the Finnish weather!) (citation needed).

We developed the sentiment filtering module of our prototype based on second part of our initial hypothesis consist:

* An article is considered to have a positive or negative view on a subject can be determined by the sentiment value of such article.

This hypothesis has one flaw, however, as we learnt from doing this prototype, we saw that: an article usually does not have a single subject, but rather, have multiple topics that it conveys (more on next chapter). For example, with a news titled: “The US’s war in Vietnam”; there are many topics/categories that can be considered as the “main topic” that could be interested to different readers: US news, War news, Vietnam news, Historical news … Thus, with each news document contains many different subjects and topics, it is possible for the article to have an overall negative sentiment, but some subjects are viewed in a positive way.

With knowledge of these possible flaws, we implemented a filtering system that can analyze the sentiment of both the whole article as well as the opinion of each topics in it. Because the Stanford CoreNLP works on a single sentence basis (footnote: tested in chapter 3.2.1), each sentence has its own sentiment value, ranging from 1 (very negative) to 5 (very positive). With these single sentences value, we calculate the overall sentiment value of the article as well as the sentiment of each topic/subject using this equation:

In which:

* **V** is the overall sentiment value of the article/the subject.
* is the sentiment value of sentence *i*
* is the length in number characters of sentence *i*
* For the overall sentiment: *i* is every sentence of the text document
* For each topic/subject: *i* is every sentence contain the topic/subject

With this equation, we felt that we have sufficient data to evaluate if two articles are from different point of view. This leads us to the next question: how to know which pair of news should we take for comparison?

*References*

*Nasukawa, T., & Yi, J. (2003, October). Sentiment analysis: Capturing favorability using natural language processing. In Proceedings of the 2nd international conference on Knowledge capture (pp. 70-77). ACM.*

[*https://dl.acm.org/citation.cfm?id=945658*](https://dl.acm.org/citation.cfm?id=945658)

*SENTIMENT BENCHMARK*

*Ribeiro, F. N., Araújo, M., Gonçalves, P., Gonçalves, M. A., & Benevenuto, F. (2016). Sentibench-a benchmark comparison of state-of-the-practice sentiment analysis methods. EPJ Data Science, 5(1), 1-29.*

*https://link.springer.com/content/pdf/10.1140/epjds/s13688-016-0085-1.pdf*

### Named entity recognition

With the filtered sentiment value used to evaluate the difference in point of view between two articles, we then use named entity recognizer (NER) to find the relevant articles from our knowledge corpus to the one the user is reading and, determine which should we suggest to the user so he can see the subject from multiple viewpoints.

Unlike sentiment analysis, named entity recognition (NER) is generally considered as a solved problem since their benchmark reach a high score compare to human (citation needed). NER’s being a solved problem is always a good thing to hear since if the results turn out to be not what we expected it to be, we know that our hypothesis or implementation is wrong, not because of the technology.

By default, Stanford CoreNLP definition of “named entity” as a board term (citation needed, or even example), which make the NER annotator returns many unnecessary information that we do not needed like dates, times, , numbers, common words like “you/me/he/she …” , or proposition text like Mister, Miss … This information is too generalized and too broad, thus, does not provide any meaningful context for our algorithm and if left unchecked, will interfere with our article relevant calculation. As a result, it is important for us to we implement a system to filtered out these irrelevant entities. We also split the filtered results into two categories: abstract entities and discrete entities. The two groups contain:

|  |  |
| --- | --- |
| Discrete entities | Abstract entities |
| PERSON | RELIGION |
| LOCATION | NATIONALITY |
| ORGANIZATION | TITLE (job title) |
| MISC | IDEOLOGY |
| CITY | CAUSE\_OF\_DEATH |
| STATE\_OR\_PROVINCE |  |
| COUNTRY |  |

After receiving the filtered results, we created an entry object for every named entity in the article, which contain the appearance number of that entity, as well as the its sentiment value. All these entry objects, along with the annotated title and article overall sentiment value, are combined to created one article annotation data to save to our local database.

|  |  |
| --- | --- |
| On the right, is an example of a saved article annotation object. All annotations are stored as a JavaScript object (footnote: https://www.json.org/), in a single .json file.  From the example, we can see that each annotation contains:   * Meta data about the articles: url, title * Annotated title, which contains sentiment value, length, and entities appearance * List of every named entity entries object, which shows the named entity, is appearance, and its sentiment value (calculated using the equation on the previous chapter).   With these data stored, we now have the “annotation pipeline” ready and can proceed to the “article matching pipeline” to find news from another point of view to a given document. | Figure n: Example of an annotated article stored in our database |

### First version of the un-polarizing algorithm

The first step for our un-polarizing algorithm is to populate our knowledge corpus. For this prototype, we filled our database with annotation of news document listed in **Chapter 2.2 – Evaluation criteria**.

The goal here is to not find the most similar articles, but the most relevance one, in which, we defined the relevance <*do some research here!*> as articles sharing both similar and different contents (entities). Two articles, if deemed relevance, should have half of their contents talking about similar things, and the other half talk about different things. We calculate the relevance score between two articles using this equation:

In which:

* X is the relevance score
* is the number of unique similar entities. Unique means that each entity entrance is only count once, even if they appear multiple times in both articles.
* is the number of unique different entities, summed from both articles.

As shown in the equation, when the two articles are absolute relevance (by our definition above), X should be 0, and the less relevance the two articles are, the bigger the X value will be. So, to find the list of the most relevant articles, we find the article with the smallest X value (because X is the absolute value so 0 will be smallest).

In the code, we call the -<*similarityModule.findSimilarArticles*> - this is the function we called. The input is one single “url” and the output is the list of relevance articles with their annotated information displayed so that the user know why we suggest these.

(*End and hook for the next part*) - This equation provides good results, but however, we weren’t happy with the result, or more exactly, what we get from the solution. The result did give us some other articles to the solution, but what we have is just article’s titles, and some mentioned keywords. We felt that we could do better by analyzing the article furthermore and find different opinions/facts on a sentence basis, not just by keyword counting/sentiment analyzing.

However, this approach is not good enough since the sentiment analysis doesn’t work as well as expected, so this in effect just return bunch of articles talking about the same thing.

(*What a crap result we got with NER – something to cite here, I don’t know, some people must have probably researched about this*) – Our initial idea was really simple, find articles that talk about similar topic, and then sort them by the different in sentiment value, so that if two articles talk about similar topic but with vastly different sentiment value, they will have different point of view and it worth showing to the users. This approach however, as pointed out in part 3.2.2-Sentiment Analysis, is not effective since the Sentiment analysis results are all over the place and have no meaningful contribution to the algorithm at all. However, after finish writing the Similarity calculation, we found that even just suggesting the articles with relevance topics can provide lots of interesting information from many different points of view already, without considering the sentiment value.

### Weakness of the first prototype

Flaw of the sentiment analysis

(*Why it’s bad and why we don’t use it*) – The sentiment analysis hypothesis later on proves to be almost useless, as the sentiment value of the sentence/paragraph have very little correlation to the content in the articles. Which mean, even if the paragraph talks about the killing of Yemen people, it might still have a normal or positive sentiment (*TODO: find example*), because journalism is usually supposed to give provide information in the most neutral way, so it’s hard to find any correlation between them. Furthermore, it’s kind of easy to fool the system, using word like nice, good, or similar to that, to make it have a higher sentiment value.

(*Example of what other people do, properly write this in a formal way*) – Sentiment analysis services are offered by many big companies like IBM, Google, Microsoft, to some other smaller startups and a lot of open-source library exists as well. However, this method proves to be completely useless for our use-case (at least using the Stanford’s NLP lib). Because practically, sentiment analysis is only good for single sentences only, without taking into account the paragraph or the article. It is mostly used for analyzing customer reviews for products or customer feedbacks.

(*Stanford NLP bad*) – Furthermore, the Stanford NLP sentiment analysis result are quite limited, only 5 values, from very negative to normal to very positive, and work on the context of a single sentence only. Google’s service proves to be a bit more useful with sentiment scale from 0 to 100, but still work on a single sentence context. IBM’s service is the fanciest with scale of 100 for sentiment but also some other adjectives like “Anger”, “Nice”, “Happiness”, … which seems to be the most suitable for our case if we decided to go with this.

In short, sentiment analysis proves to be almost useless.

This is flaw since it can’t detect sarcasm and is flawed.

*http://blog.conceptnet.io/posts/2017/how-to-make-a-racist-ai-without-really-trying/*

*Sentiment from text analysis are bias.*

*https://researchportal.bath.ac.uk/en/publications/semantics-derived-automatically-from-language-corpora-necessarily*

## Semantic triple based un-polarizing algorithm

### Open information extraction

(*How openie improve our result, how do we use it and store it information*) – To furthermore improve the matching algorithm as to find even more relevance information for the user, we use Semantic Triples. The idea now is not only just match article that talks about the same things, but to read the sentences of the article and then see if there are any other articles that is also mention the same thing: ie: Trump hates brown people and Trump push brown people. This can in turn provide much more relevance information as two articles, at one point, talk about the same thing, so the user can not only see what each article says, but they can also have the power to see the content which the subject is talked about. To do such marvelous task, we use OpenIE from Core NLP.

Open information extraction from Core NLP works pretty good out of the box. Inputting one sentence into it and it gives you a lot of statements constructed from that sentence to work on. These statement however are too many and too noisy as some of the statement are quite useless information (he is president) or some are shortened version of other statement (USA hates muslim people vs USA hates muslim people all over the world).



Picture n: example of result from Core NLP

*TODO: Need example of this*

To make use of the OpenIE data, we do a three-step filtering.

First, triplets with the relation word that is not a verb and not the verb “be” is also removed. This make sure that all the non-meaningful statement (example needed!) will be removed from our result <*tripletMeaningfulProcessor.filterOpenieResult*>

Second, we remove all the triplets that are just shortened version of others, this remove quite a bit of them (needs some number value, to see the effect) *<tripletTrimmer.trimShorterTriplets>*

Last, combined with the named entities analyzed from 3.2.4, all the statement that doesn’t have an entity mentioned will also be removed, since the triplets without any meaningful entities mentioned will be useless in term of information for us anyway <*coreFeatureExtractor\_getContainingEntities*>

* Find example for each of these cases to see which result is removed and which is saved to see the impact, also, maybe some number on how much is removed for example.

After these three-step filtering, the annotated data of the article is saved to our local database as a JavaScript object in json format, same process as part 3.2.4, only with different data.

|  |  |
| --- | --- |
| This is a snippet of the annotation data stored in our database. Each entry contains:   * Meta data about the article: url and title * Array of annotated information about the content of the article, split down to a sentence level.   Each data-sentence contains:   * Full text content of the sentence * Triplets exists in the sentences and their information.   Each triplet in the sentence annotation contains:   * Subject, relation and object text. * Full text content of the triplet (combine subject, relation and object) * Containing entities.   We made the decision to store as much information as possible like the full sentence text as well as the triplet’s text so that when we return the un-polarized result to the user, they can see all the reason that leads to the decision to show them the results and can judge the comparisons for themselves. | Figure n: Example of an annotated article stored in our database (current version) |

### Triplet pairs, common entities and common statement counts

(*What are these 3 above stuffs and our sorting equation*) – With the semantic triples annotation implemented, we hoped to find some common or opposite statements from different articles that we normally could not normally find if we just read through the titles and skim through the contents. We use this equation to calculate the relevance of any given article to the source article:

In which:

* is the relevance score, the higher X is, the more relevant the article is to the comparing one.
* is the number of triplet pairs appear in both articles. Two triplets are considered a triplet pair when they share two common entities, on both the subject and object part. For example, if one statement says <*United Kingdom negotiates deal with the European Union*> and the other statement says <*European Union rejects the deals from United Kingdom*> they are considered a pair because they both talk about <*European Union*> and <*United Kingdom*> in theirs subject and object part.
* is the number common unique entities appear in both articles. If two news piece talk about an entity, it is count (but only once, even if it appears multiple times)
* is the number of common statements appear in two articles. If two statements (semantic triples) talk about a similar entity, this number goes up.
* are the constant weight value for each of these variables. In our prototype, we set =1. We use these values because entity pairs are quite rare, as most of the articles in our database, when compared to the rest of the other articles, doesn’t exist a pair at all. Maybe test with a bigger database?

General rating of the information here? I don’t know, maybe we add more data or stuffs like that?

### Providing the information to the user

Since the main purpose of our solution is to provide the user more information so that they can make a better judgement of themselves, we feel that it is important that the we should also provide as much information as possible. So, for our un-polarizing result, we will give the user the list of the most relevance articles to the one he wants to check, as well as other information that we use to come up with the conclusion, so that he can see the full picture himself, knows the reason we come up with the result, and now, being informed, can fully know the news about the situation or subjects.

|  |  |
| --- | --- |
| The return result for the user is a list of relevant articles and their annotated data, each contains:   * Meta data: general information about the two articles, containing their urls, titles and the number of entity pairs, common entities count and common statement count. * Entities pair data: we feel like this is the most interesting information so if is exist, it’s should be shown. * Entities data: all the rest information of the entities, which contains what the entity is as well as in what context it appears in, in both the source and target articles. | Figure n: Example of an annotated article stored in our database (current version) |

### Limitation of the current system.

* No verb processing in the Semantic Triple: Our current implementation only utilizes the “subject” and the “object” part of the semantic triples. The relation part, which, based on our filters, will always contains at least one action word. This information could be important as it can help us to find contradiction between different statement, thus, gives the user an even better view of the topic.
* No negation checking: current Stanford Core NLP system doesn’t detect negation in their Open Information Extraction yet, so we might miss some semantic triples from the articles. Especially, when the verb processing is implemented, missing the negative triples could lead to missing some of the contradict information. This is weird since they do have negation checking (in POS annotator), but I guess they can’t make it to work in “natlog” annotator.
* Computational drawback: processing an article on my computer (i5-6700HQ) took around 10 second to process one article. It’s not a big problem because 10 seconds isn’t too long but should be noted since it’s not instant.
* Needs huge database, to works. A single topic (ie immigrant) should have around 50 articles to be able to generate good ground truth of information, and the bigger the database is, the longer the computational time it takes. We can build the database to be bigger by automatically fetch news articles from source like Google news, but to solve the computational problem for big database, we need more research on how big companies like Google, Amazon or IBM deal with it.

# RESULTS EVALUATION

(*Detailed information from our article base, how many is close to the point, how many useful information can we get from that*) –

# FUTURE WORKS

## Ontologies based entity relevance

(*How using ontologies, can help finding similar words/entities, similar to the ontologies relevance in Chinh’s thesis*) – The current system finds entities pair base totally on their word-to-word similarity. Using ontologies, we could find and link together entities that are relevant to each other (ie, Muslim and Christian, as both are religion), thus, making the system smarter and able to find more information.

## Word-net verb contradiction

(*With the triplet pairs implemented, we could find contradiction between the triplets. Using word-net to find verb that have similar meaning/or opposite meaning*) – For processing the relation verbs, using wordnet (<https://wordnet.princeton.edu/>) or similar tool, so we can find words that have similar meaning, close meaning or opposite of each other, thus, making the system able to find more connection/contradiction between different statements within different articles.

## AI based un-polarized algorithm

It’s not that easy, you just can’t say AI and all the problem is solved. But traditionally, as NLP evolve, all the programming/method driven method for NLP has been changed for a better machine learning model for almost everything like sentiment, openie and more tasks (tons of reference needed here). We could try to apply the same for our stuffs. However, we realize that as there are so many possible inputs data (which is all of the Stanford NLP annotation stuffs), and the outcome is so limited, same POV, different POV ???, training these data would be really hard. Training these models would be an interesting task as well. But we think AI is the future, and we should aim for it.

## Cloud service design

Local

database

Web

content

processor

Article

annotator

Requested URL

News gatherer

News

suggestions

module

Suggested

articles

Figure 1. Test figure

With the algorithm ready, we need to automatically get our data somehow. Fortunately, there are a lot of news APIs available, for example, Google News or many other things, just one simple APIs and it can give you all the thing you want. It’s also good for evaluation later, because these APIs allow you to search for query by word, date and time, which will be useful to compare the results between our stuffs and theirs.

## User interface and user experience design

# CONCLUSION

Hope you enjoyed the text...

In the bibliography the recommendable style is Chicago. You can also use other styles: the main thing is that the styles of the bibliography and referring technique are **consistent** in the whole thesis.

|  |  |  |  |
| --- | --- | --- | --- |
| Word | Year | Magnitude | Example |
| example | [1700,2000] | [1,10] | example |
| example | [1950,2000] | [1,106] | example |
| example | [1995,2000] | [10–6,106] | example 1, example 2 |

Table 1. Example of the table

Bibliography

Spohr, Dominic. "Fake News and Ideological Polarization: Filter Bubbles and Selective Exposure On Social Media." *Business Information Review* 34.3 (2017): 150-160.

Meikäläinen, Matti. *Tietotekniikan perusteet.* Helsinki: WSOY, 2000.

Santanen, Jukka-Pekka. "Opinnäytteiden kirjoittaminen, lyhyt oppimäärä." 2000. http://users.jyu.fi/~santanen/info/kirjoittamisesta.html (accessed 5.10.2012).

Writer, John. "Name of the article." *Computer*, no. 2/2000 (2000): 31-32.

Appendices

1. Title of the first appendix
2. Title of the second appendix

1. <https://cloud.google.com/natural-language/> [↑](#footnote-ref-1)
2. <https://www.ibm.com/cloud/watson-natural-language-understanding> [↑](#footnote-ref-2)
3. https://nodejs.org/en/ [↑](#footnote-ref-3)
4. https://support.mozilla.org/en-US/kb/firefox-reader-view-clutter-free-web-pages [↑](#footnote-ref-4)
5. <https://www.npmjs.com/package/node-readability> [↑](#footnote-ref-5)
6. https://smmry.com/ [↑](#footnote-ref-6)
7. https://stanfordnlp.github.io/CoreNLP/download.html [↑](#footnote-ref-7)
8. Tested on average of 100 article annotations, using author’s computer: Dell inspiron 7559 with i5-6300HQ and 8GB of RAM [↑](#footnote-ref-8)