Le Pham Minh Duc

Un-polarizing news in social media platform

Master’s thesis of mathematical information technology

February 20, 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, February 20, 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](#_Toc1507299)

[1.1 Problems overview 1](#_Toc1507300)

[1.2 Proposed solution and research questions 1](#_Toc1507301)

[1.3 Research method 3](#_Toc1507302)

[1.4 Thesis structure 3](#_Toc1507303)

[2 HYPOTHESIS AND TECHNOLOGIES 4](#_Toc1507304)

[2.1 Original hypothesis 4](#_Toc1507305)

[2.2 Evaluation criteria 4](#_Toc1507306)

[2.3 Required technologies 5](#_Toc1507307)

[2.3.1 Natural language processing and its sub-domains 5](#_Toc1507308)

[2.3.2 Open Information Extractor / Semantic triple 6](#_Toc1507309)

[2.4 Technologies used in this works 6](#_Toc1507310)

[2.4.1 Stanford CoreNLP 6](#_Toc1507311)

[2.4.2 Git and GitHub 7](#_Toc1507312)

[2.4.3 Node JS 8](#_Toc1507313)

[3 UN-POLARIZING ALGORITHM 9](#_Toc1507314)

[3.1 Overall solution/algorithm architecture 9](#_Toc1507315)

[3.1.1 Article annotation pipeline 9](#_Toc1507316)

[3.1.2 Article matching pipeline 11](#_Toc1507317)

[3.2 The first algorithm using Sentiment Analysis and Name Entity Recognition 11](#_Toc1507318)

[3.2.1 Web content processor 11](#_Toc1507319)

[3.2.2 Stanford Core NLP Annotator 12](#_Toc1507320)

[3.2.3 Sentiment analysis 13](#_Toc1507321)

[3.2.4 Named entity recognizer 14](#_Toc1507322)

[3.2.5 Articles similarity calculation using only Named Entity Recognition 17](#_Toc1507323)

[3.3 Un-polarizing algorithm using Semantic triples 18](#_Toc1507324)

[3.3.1 Open information extraction 18](#_Toc1507325)

[3.3.2 Triplet pairs, common entities and common statement counts 21](#_Toc1507326)

[3.3.3 Providing the information to the user 22](#_Toc1507327)

[3.3.4 Limitation of the current system. 24](#_Toc1507328)

[4 RESULTS EVALUATION 25](#_Toc1507329)

[5 FUTURE WORKS 26](#_Toc1507330)

[5.1 Ontologies based entity relevance 26](#_Toc1507331)

[5.2 Word-net verb contradiction 26](#_Toc1507332)

[5.3 Cloud service design 27](#_Toc1507333)

[5.4 User interface and user experience design 27](#_Toc1507334)

[6 CONCLUSION 28](#_Toc1507335)

[Bibliography 29](#_Toc1507336)

[Appendices 30](#_Toc1507337)

[A Title of the first appendix 30](#_Toc1507338)

[B Title of the second appendix 30](#_Toc1507339)

# 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).

## 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 relevance 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 relevance 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 relevance 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

## Original hypothesis

(*Write this to be longer, reference???*) Our original hypothesis is that: given two articles talking about one similar topic (*for example: The US’s involvement in Iraq*), if one article has a positive views on the situation (*ie: Saddam Hussein is a terrible man and the people living under his reign are suffering badly*) and the other has a negative views regarding the same situation (*ie: it furthermore destabilizes the region and the main intention of the war was because of oil*). With articles telling about the same story but with different sentiment value, it could be interesting for the reader to see from different kind of attitudes about the same topic, thus, bring him to different point of views about a problem.

This hypothesis, however, after some implementation and evaluation, was proven to be not good enough (**more in-depth in chapter 2.3.2 Sentiment analysis**). The final solution of this thesis utilizes more complexed calculations and processing techniques that was not originally planned from the start. However, the initial hypothesis did create a solid technology base to start working on: Named entity recognizer (*to understand the article topic*) and sentiment analysis (*to understand the positivity/negativity of the article*).

## Evaluation criteria

(*Longer, maybe no reference needed*) TDD – Test driven development -To test the rigidity of our algorithm, we gathered a small dataset of 78 articles (+ 8 non-readable by the web content parser) – *could be more, should update* – with 3 main themes: Muslim in Europe (24 articles), Muslim in Asia (39 articles) and Asians in Europe (17 articles) (*somehow, the number of articles in each categories and the total number doesn’t match. Something must be wrong with the* listing*, we need to re-check these things*).

With these hand-picked data, we can look through each article, to judge for ourselves which we think is the most relevant, and which is not, and then, compare our result to the result returned from the algorithm and judge the result for ourselves. With these three different categories, we can make sure that there we will know if the algorithm returns the relevance information or not, how close is the suggestion and in some case, if the return result can even surpass our hand-picked solution (because human is flawed and cannot read through all these data).

## Required technologies

### Natural language processing and its sub-domains

(*Consider rename the title*) (*Lots of references in this, this is like a wildly researched fields*) Named entity recognizer and sentiment analysis are two of the many sub-tasks covered by Natural language processing (NLP) technology, *which is an area of computer science and artificial intelligence concerned with the interactions between computers and human (natural) languages, how to program computers to process and analyze large amounts of natural language data. (Source from wiki- more information needed).*

For just named entity recognizer only and sentiment analysis only, there exists multiple tools that do not require NLP. For example, a simple named entity recognizer (*citation needed)* consists of a dictionary of many individual nouns and a lookup function to match the result from the document and the dictionary. Similarly, a basic sentiment analysis can be made by assigning a sentiment value to each of the adjectives in the sentence/paragraph and then calculate the sum of these values as the sentiment value for the sentence/paragraph. However, these methods are quite barebone and quite usually not really correct since their rules are quite flawed and there are many ways for them to misinterpret the true sentiment of the sentence. Sophisticated NLP tools with “sentiment analysis” supported have a more complex ways of defining the sentiment value

Named entity recognizer and Sentiment analysis can be easily created without NLP using a rule-based system. However, these basic tools are not good and sophisticated NLP tools are much better because they have a more complex system that take many other things into consideration.

### Open Information Extractor / Semantic triple

(*What is semantic triple, why it is important and a critical part of this thesis.*) – After our first implementation of the algorithm, we realized that just Sentiment analysis and NER was not enough. To make the result more interesting and more useful to the user, we have to use Open Information Extraction, which, unfortunately, was not as widely developed as the two previous features since they are much more complicated.

*https://nlp.stanford.edu/software/openie.html*

*https://nlp.stanford.edu/pubs/2015angeli-openie.pdf*

[*https://getd.libs.uga.edu/pdfs/hooge\_david\_c\_200705\_ms.pdf*](https://getd.libs.uga.edu/pdfs/hooge_david_c_200705_ms.pdf)

[*https://en.wikipedia.org/wiki/Semantic\_triple*](https://en.wikipedia.org/wiki/Semantic_triple)

*https://github.com/dair-iitd/OpenIE-standalone*

## Technologies used in this works

### Stanford CoreNLP

(*This could have a lot of comparison and references, or not, depends on my mood*) Many big companies offered NLP services like Microsoft, IBM, Google, each with their own technology and strong/weakness. These tools, however are 1. Expensive to run, 2. Close source and are subjective to change in any moment (like IBM) which make them not attractive to use.

Stanford CoreNLP is an NLP tool created by Stanford university, free and open source and quite easy to setup/use and very powerful as it supports many features.

After some testing some of these tools, we decided to go with Stanford CoreNLP because Microsoft is quite weak and in-accurate,

IBM is expensive and is a pain to setup and they change the name/domain quite often so not reliable, even though their named entity recognizer and sentiment analysis are really good.

Google is ok, looking and feeling very similar to Stanford CoreNLP (like they took the source code of NLP and improved it upon). They strip a lot of features from CoreNLP but the things they keep (like NER and Sentiment) are much better than the stock version in CoreNLP.

Still, I would like to have everything in one place, free and will guarantee to work no matter what, so, I went with Stanford CoreNLP. The important part of this thesis is the algorithm behind the news unpolarizer, not from how can I choose a better service to use.

Furthermore, if we want to include OpenIE in our tool, Google or IBM or Microsoft does not offer such feature, so Stanford NLP or that standalone Openie tool is our only option. Fortunately, both of them are free and opensouce.

### Git and GitHub

Software developer need to use Version control nowadays. Not only it acts as a good backup system in case something goes wrong, it’s also good for experimenting new features and going back to older version of the software to evaluate the result.

Git is I guess the most used version control system nowadays, originally developed to work with Linux and it’s awesome.

Github is I guess also the most widely used version control system in the world. So that’s what I used.

The whole thesis code and example and history and everything is saved in github, will remain there forever (at least until Microsoft close it) and link is provided here as well as in the end of the thesis.

<https://www.computer.org/csdl/mags/so/2012/03/mso2012030100-abs.html>

https://jyu.finna.fi/Record/jykdok.1501755

### Node JS

In this work, we choose NodeJS to do our coding, as it’s one of the easier and faster programming languages to code in, as well as the author expertise in it would make implementing the algorithm faster than other language. Maybe try some cool articles about NodeJS and backend development.

# UN-POLARIZING ALGORITHM

## Overall solution/algorithm architecture

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 relevance information with different point of views.

### Article annotation pipeline

Here is the overall architecture of the Article annotation pipeline.

Local

database

Web

content

processor

Stanford

Core

NLP

Sentiment

Value

Article

url

Semantic

Triples

Named

Entities

Sentiment analysis

NER

filtering

Semantic

triples

Filtering

Annotated

article

Article annotation pipeline

Figure 1. Test figure

Boxes and lines with different colors represent different parts of 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
* Dashed box and line: process and algorithm implemented in the second phase prototype, things that we didn’t plan to implement initially.

When we want to annotate an article, we use its url to read and then proceed to remove all the web <html> tag and other unnecessary information like advertisement and the website suggestion/meta data (more on 3.2.1). Then, with the pure article in text form, we parse it into our matching pipeline.

The first step to push the text into the Stanford Core NLP (more on part 3.2.2). From the information received from the engine, we use 3 sets of information: the sentiment analysis (more on part 3.2.3), the named entity recognition (more on part 3.2.4) and the detected semantic triples (more on part 3.3.1). Each dataset then will be furthermore analyzed/filtered using various equation we will described in later part of this thesis to make it ready for the articles matching solution. Then, all three processed results will be recombined into one annotated object and saved into our local database.

Note that the Semantic triples and other processes related to it are only utilized in the second prototype, where we weren’t satisfied with the results we have from the initial implementation.

### 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 relevance 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 relevance. 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.

## The first algorithm using Sentiment Analysis and Name Entity Recognition

### Web content processor

The article on the internet are presented inside a web page, with just not only the article itself, but with tons of other unnecessary things like html tag for formatting, images and captions, links to other things on their website and advertisement text.

To strip away all the unnecessary function, we have a web content parser module which utilize a similar technique to reader mode on Firefox or Safari which will automatically strip away all the non-article part in the web content. However, this is not enough since it only strips the advertisement and related news, the core article and the html tag around it still persists, which, to solve, I wrote a smaller module to automatically remove all the html tag as well as the image and the caption, which is not really accurate since each website have different layouts and ways to present content, make parsing out the content really not effective, since, for example, some website, when they end the sentences or the caption of the image, the don’t add the “.” (dot) or sometimes they make multiple dots, which make finding out the article contents with proper sentences are quite a problem in most of the article, which make the overall annotating result worse.

One effective way of ensuring that the sentences forwarded into the CoreNLP annotator are correct is to use a tools call SMMRY, an article summarization tools, which will read through the article and gives out the sentences that it thinks contains the most important information of the article. This tool is quite effective for our case as it first, strip away all of the garbage contents like html, tag and about this website … which make the article annotation work correctly. One downside of this is that it will not return the whole article, only part of it (the more important part) so there might be possible information that will be lost during the stripping of the content. Still, I think it’s better to use this SMMRY tool instead of my own web content parser because it’s not very good and accurate.

When all the data-collecting and parsing is completed, we push the parsed data into our annotation pipeline.

### Stanford Core NLP Annotator

(*What is the role of the Stanford Core NLP in our program and why we need the local db*) - After extracting the text part from the web article, we push the text into the Stanford Core NLP engine. Stanford Core NLP have many annotators, like NER, Sentiment, lemma … (*find more*), we can specify which annotator we want to run through to save processing power. After receiving the annotated result from the Core NLP, we will run the result through our special filter for sentiment, ner and openie to retain the only relevance information for our un-polarize algorithm.

The filtered annotation result (let’s call them core feature) will be saved into the local database for future comparison of the un-polarizing algorithm. The use of the local database for the core-results is necessary because the processing time of each article is quite long, around 30 seconds each, so, to compare one article to other hundred to find the most relevance one, we must process them beforehand so that our results are readily available for future accessing.

### Sentiment analysis

(*How do we use the sentiment analysis, how it’s not working and how we decided to just not use it*) – This is our very first idea upon figuring out a solution to this problem. Our initial hypothesis was simple, if two articles talking about one problem, for example: The U.S President, Donald Trump, if one article talk about him with a positive sentiment, and the other with a negative sentiment, then they are from the different point of view and we should suggest the other article to our user.

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.

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.

However, as this is a public research with source code fully available for people to use on their own, we don’t want to tie ourselves to private services that might either be unavailable someday, or restrict our access because of reasons, we decided to go with Stanford NLP, the open-sourced solution from other academia.

We calculate the sentiment value for article using Stanford NLP using the weighted average sentiment value of each sentence in the article. In which, we calculate the average value for the whole thing, but longer sentence (by characters count) will have a bigger weight than short sentence.

* **V** is the sentiment value of the article.
* is the sentiment value of sentence *i*
* is the length in number characters of sentence *i*

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.

In short, sentiment analysis proves to be almost useless.

### Named entity recognizer

(*How do we use the NER, the initial solution/algorithm use NER. Briefly the result of the one using only NER*) – Named entity recognizer (NER) is also presented in our original idea. With the sentiment value used to judge the \**difference in point of view*\* between two articles, we use named entity recognizer (NER) to find the relevance articles to the one the user is reading and then, compare the sentiment result to suggest it to him.

The default NER of Stanford NLP contains a lot of unnecessary information that bloat the return result for our algorithm so much. Many information (named entities) detected are quite un-relevance to our case, for example, he/she, year, number and some proposition text like Mister, Miss … We have to write a filter for the NER results to get the relevance information only. For the user convenience, we also split the result 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 |  |

Stanford Core NLP work on a single sentence basis only, which mean the meaning of the next or previous sentence in the paragraph doesn’t affect the annotation of one sentence in anyway, which is a bad thing for us since we want to work on a rather macro way, instead of micro based way like this. Thus, after filtering out the un-needed result, we perform the entities counting, in which we count the number of each time an entity has appeared in the article, as well as calculate the sentiment value or the average sentiment value of that entity, and finally, save all the result to the local database.

|  |  |
| --- | --- |
| (*Remove the table border later*) – Initially, the annotated value stored for one article in our database look like the part on the right:  All article will be stored as a JavaScript object, in one big single database.json file.  Each annotation object contains:   * Meta data about the articles: url, title * Annotated title, which contains sentiment value, length, and entities appearance * Annotated contents of the article, which contains information on all the entities, their appearance and their average sentiment value.   With these annotations’ information stored, we implemented a matching algorithm to suggest relevance articles to the one the use is viewing (more information on part **3.3.1 Articles similarity calculation using only NER**). 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. | Figure n: Example of an annotated article stored in our database |

With the article annotations stored in our database, we can now suggest news with different point of view to any given article with our un-polarizing algorithm.

### Articles similarity calculation using only Named Entity Recognition

(*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.

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.

## Un-polarizing algorithm using Semantic triples

### 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.
* 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.

## 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