Evaluating online arguments

Rick Clement (s3852903)
Remco Pronk (s2533081)
Jan Willem de Wit (s2616602)

October 5, 2018

Abstract

We have created a system that mines arguments from the internet.

1 Introduction

The internet contains several websites that offer people insights into many ongoing debates. They contain a summary of the issue, and a section of common arguments, both pro and con, making them perfect for using as a base for an argument mining implementation: the arguments are easy to find and already split into a groups for pro and con arguments, making testing the implementation a much simpler task.

1.1 Problem

We will try to find arguments that are for or against a certain proposition. Additionally, we will attempt to link points to their respective counterpoints.

1.2 State of the Art

There exist applications that can mine the internet in order to show points arguing for or against a certain position on an issue, like IBM's Project Debater.

1.3 New Idea

While applications exist that can mine the internet for arguments and employ sentiment analysis to classify them as pro or con arguments, none that we found can link arguments to their counterarguments, a gap we aim to fill.

2 Methods

2.1 Simulation Model

2.1.1 Crawlers

We created crawlers for the websites Procon.org and idebate's Debatabase. [1].

Procon.org compiles popular controversial issues, and the respective pros and cons for each issue. Procon.org uses two kinds of layouts to represent certain arguments. The first layout is dubbed *shortArguments*. Pro and con arguments are represented by approximately one sentence, summarizing its point. This point is then followed by text to expands on the argument. The crawler for this layout can be seen

[1] All code can be found at `https://github.com/Remco32/ArguingAgents_group5`

in the file `/Crawler/crawler_procon_org_noQuoteboxes.py`. The second layout is dubbed *longArguments*. Every argument has a preface in which the source of the argument is introduced, after which a quote containing the argument follows. Only the quote is mined. The crawler for this layout can be seen in `/Crawler/crawler_procon_org.py`.

The Debatabase provides points for and against a topic. Each point, both for and against, is then also accompanied by a counterpoint. The crawler for the Debatabase can be found in `/Crawler/crawler_debatabase_remco.`

The resulting web mined files can be seen in the directory `/Crawler/Crawled/`.

2.1.2 Doc2vec

We are using doc2vec [3] to represent arguments as vectors. This method is inspired by the more well known word2vec [5] where each word is represented by a vector. An algorithm learns these vectors by processing large amounts of texts, where the context of the word decides what the learnt feature vector will look like. This way, words that appear in similar contexts will have similar feature vectors. This way, the meaning of the word is learnt. We are using doc2vec with the purpose of filtering and linking arguments. Because we are using multiple sources, we want to filter out arguments that are too similar. Besides that, we would like to link pro arguments to similar con arguments. Using doc2vec we can represent these arguments as a vector, where two arguments that are very similar should also have a similar feature vector. Currently, our system can take arguments and create a model from this, which allows us to compute the similarity between arguments.

2.1.3 TF.IDF

TF.IDF is a method that determines the relevance of a words in a certain document in comparison to the rest of the documents in a corpus[6]. In the context of our application, the corpus is a folder of our mined arguments and the documents are text files containing one mined arguments each. The calculation for a term in a document is formatted as follows:

$$w_{t,c} = TF_{t,d} * IDF_{t,c}$$

where TF stands for the term frequency and IDF stands for inverse document frequency. Both terms and their respective calculations will be explained further below: The term frequency $TF_{t,d}$ is the frequency of the term t in its document d:

$$TF_{t,d} = \frac{count_{t,d}}{total - count_d}$$

where $count_{t,d}$ is the number of times the term t occurs in document d and $total - count_d$ is the total amount of terms in d. However, term frequencies in natural language tend to follow something called Zipf's law[2] where if we rank all terms in a document by their frequency, the following occurs:

$$\forall t \in d, count_{t_n,d} \approx 2 * count_{t_{n+1},d}$$

meaning that each term occurs approximately twice as often in the document as the next. This means that the calculated TF.IDF values will be biased towards more frequently occurring terms. There are two ways to help prevent this from happening: cleaning the text to remove stop words that tend to have high frequencies and little relevance and taking the logarithmic values of the original calculated frequencies.

The inverse document frequency is determined by finding the frequency of the term t in other documents in the corpus and taking the logarithm of its inverse. In mathematical notation, it looks like this:

$$IDF_{t,c} = log\left(\frac{N+1}{DF_{t,c}+1}\right) + 1$$

where N is the total number of documents in corpus c and $DF_{t,c}$ is the number of documents in c in which term t occurs.

Combining these two calculations, we can now complete the entire formula for the TF.IDF value of a word in a document:

$$w_{t,d} = log\left(\frac{count_{t,d}}{total - count_d}\right) * \left(log\left(\frac{N+1}{DF_{t,c}+1}\right) + 1\right)$$

These values can then be sorted, with higher values indicating a higher relevance. Currently, this is as far as the implementation of the method goes, but the plan is to compare these values to a set of lexicons that contain a significant number of commonly used positive and negative English words[1][4]. Once the arguments have been split into a positive and negative group, these values will once again be used to try and link the arguments together. The code for the implementation can be found in `/Code/TFIDF.py` on the GitHub repository linked earlier.

3 Results

3.1 Experiment Findings

3.2 Interpretation of Findings

4 Discussion

5 Role of team members

Rick:	TF.IDF implementation
Remco:	Crawler procon.org and debatabase; converter mined files; cleaner text
Jan Willem:	Converter mined files; doc2vec implementation

References

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

[2] R Ferrer i Cancho. The variation of zipfs law in human language. *The European Physical Journal B-Condensed Matter and Complex Systems*, 44(2):249–257, 2005.

[3] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196, 2014.

[4] Bing Liu, Minqing Hu, and Junsheng Cheng. Opinion observer: analyzing and comparing opinions on the web. In *Proceedings of the 14th international conference on World Wide Web*, pages 342–351. ACM, 2005.

[5] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[6] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 133–142, 2003.