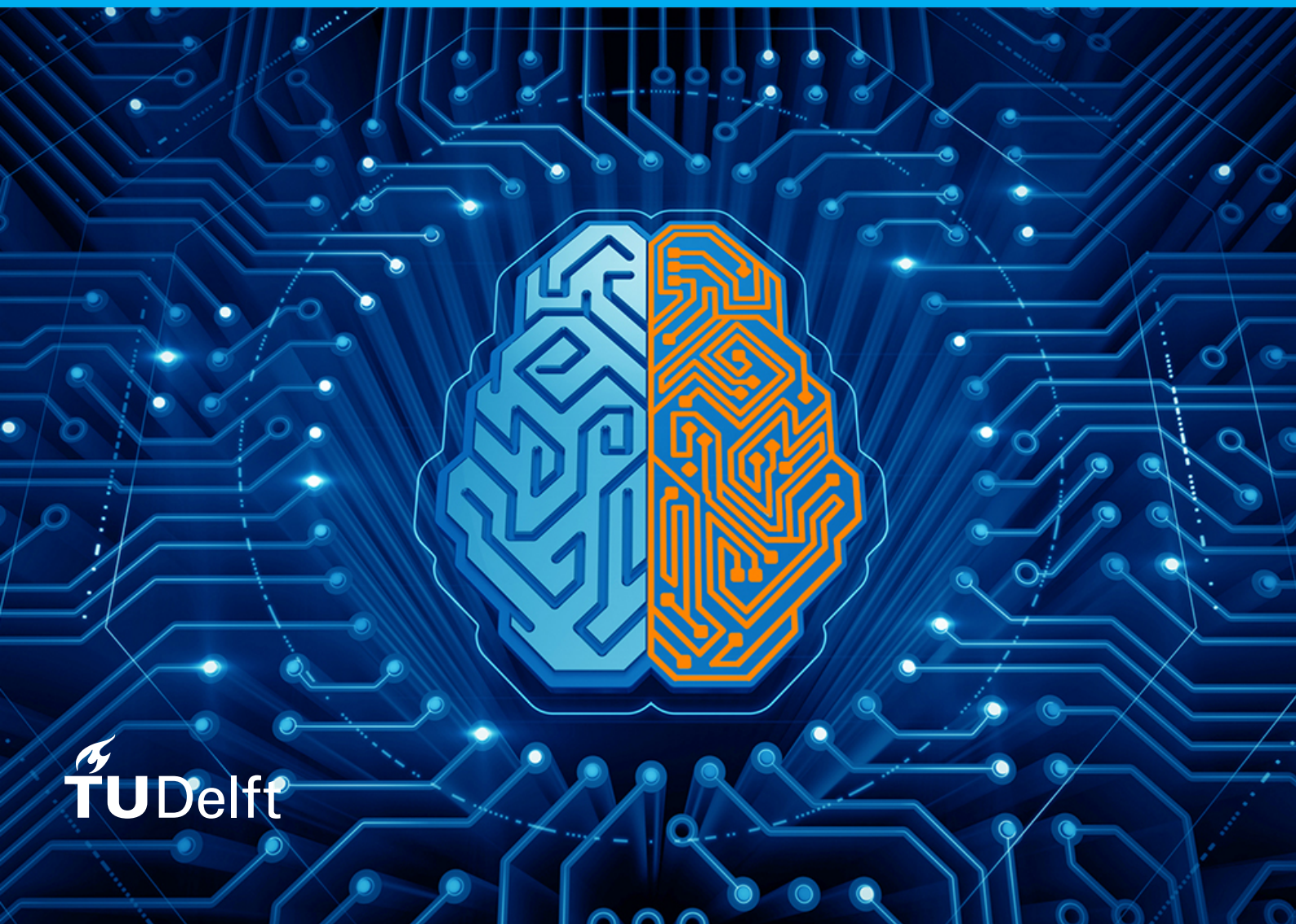
Politags

Breaking the filter bubble by using information extraction techniques to enable diverse personalization.

Joost Martijn Rothweiler

In collaboration with Max van Zoest
Supervised by Prof. Dr. Catholijn Jonker



TU Delft

Politags

Breaking the filter bubble by using information extraction techniques to enable diverse personalization.

by

Joost Martijn Rothweiler

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on June 25, 2018 at 9:00 AM.

Student number:	4246551	
Project duration:	June 26, 2017 – June 25, 2018	
Thesis committee:	Prof. Dr. Catholijn Jonker	TU Delft, supervisor
	Dr. Nava Tintarev	TU Delft, thesis committee
	Arjan El Fassed	Open State Foundation, thesis committee

This thesis is confidential and cannot be made public until June 25, 2018.

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Preface

After completing many different courses related to artificial intelligence and web science it was truly a pleasure to work on this project that is so well placed on the intersection of these research areas. However, what was even more unique about this project was that with all people involved we were able to combine our passion for technology with our genuine desire to make a positive contribution to the landscape of modern democracy.

It is well over a year ago when I first sat down with Professor Catholijn Jonker and Max van Zoest to discuss possible research directions for a thesis. I had always known from previous courses taught by Catholijn that she is extremely dedicated to using technology for the good. However, it was only after our first few meetings that I found how dedicated she is to make people get the most out of themselves. With her heart and endless amount of time and effort put into this project, Catholijn has always been able to enthuse me with her positive attitude and grand ideas. For this I am incredibly grateful.

Open State Foundation supported us in bringing our initial ideas to an entirely new level. Open State Foundation has been active in promoting open data and digital transparency for many years now. Their everlasting drive to make the government more aware and transparent has led to some great products over the years. I was very fortunate to cooperate with them so productively and to design Politags in collaboration with their latest product; Poliflw. For this, I would like to thank everyone at Open State Foundation and in particular Arjan el Fassed, director at the Open State Foundation. Arjan has helped me better understand how one can make a positive impact on society, even with a small team like the one at Open State Foundation.

We worked on a project so large and versatile that it allowed us to spread the work into two entirely separate theses but work together nonetheless. Together with Max van Zoest, I was able to set steps in this field of research that would never have been possible by a single student in the available time frame. Being able to discuss progress and challenges that we faced on a daily basis, as well as keeping each other motivated, has been a major factor in turning this project into the success it has become. Some say you are the product of the five people you spend the most time with on a daily basis. I am proud to say that Max has been one of those five people for me for the past two years.

I would like to thank Dr. Nava Tintarev for her critical contribution made towards the final stage of the project. By asking the right questions, she enabled me to report on crucial parts of the research that would otherwise have been missed.

Last but not least, I would like to thank my parents for their unconditional support. Not only for the period in which I conducted this particular research but for the last six years spent in university.

Democracy is something that we in the Netherlands often take for granted. However, advances in technology and an ever-increasing influence from abroad make our democracy change continuously. We need to stay on our toes to make sure that we allow it to change in the right direction. It is not only that one time we are asked to vote, but the daily arguments and critical thinking that make our democracy to the successful system of collaborative decision making that it is.

The fact that Poliflw was so widely covered in the media shows that with this research we are on the right track to make a positive impact. The Google Digital News Initiative was proud to present their investment in the Poliflw project, naming it as one of the three Dutch projects funded. The NOS (Dutch Broadcasting Foundation) covered a story on Poliflw in the weeks prior to the local Dutch elections. I hope that through similar innovations, we can make sure digitalization no longer poses a thread on democracy but instead enhances its capabilities and effectiveness.

Joost Martijn Rothweiler
Delft, June 2018

Contents

1 Introduction

With the advent of the Internet and social media, the number of opinions, perspectives, and ideas that can be found online is larger and more diverse than ever [6]. Nowadays, anyone who has an opinion can voice it online by typing it into a small box and pressing *submit*. While the aforementioned could be of great benefit for democracy, there is a problem associated with the business models of social media platforms and search giants. Social networks realize revenue by serving advertisements periodically in a user's *feed*. Search platforms realize revenue by serving advertisements as search results, which are monetized when a searcher clicks on the link.

To maximize earnings, social media platforms have to keep users scrolling through their feed for as long and often as possible, so they can be served many ads. Likewise, search platforms have to make search results as relevant as possible, so as to increase returning searchers and clicks.

Personalization algorithms are key to keep users scrolling and clicking. Therefore, they are designed specifically to serve users content that is as interesting or relevant as possible. Personalization has its benefits, when we search online for a cafe while being in Amsterdam, we want to see cafes in Amsterdam and not in New York. However, there is a backlash because people are served bits of information that are related to what they themselves, or their friends, have already interacted with [39].

In terms of exposure to political news, this results in a segregation of the Internet into small political groups with similar ideas to a degree that a narrow-minded approach to those with contradicting views may become inevitable. All of this happens without user consent and most people are unaware of these filters in social media and search [14].

Sunstein [54] argues that personalization algorithms result in a situation where people could easily and unknowingly cut themselves off from any information that might challenge their beliefs, and that this poses a negative effect on the democratic dialogue. Others such as Pariser [39] argue that the personalization algorithms used by Facebook and Google serve users with similar perspectives and ideas and remove opposing viewpoints without their consent. This may lead to a situation in which users receive biased information which, in the case of political information, means that contrasting viewpoints on a political or moral issue may never be presented. This issue is widely referred to as *filter bubbles* and its direct result is the decay of quality of information and diversity of perspectives to which citizens are exposed.

The scope of this problem is daunting. A study in the US in 2014 on how people access political news concluded that 48% of 10,000 panelists reported that they accessed news about politics and government exclusively on Facebook in the week before the survey [35]. A study on millennials noted that 86% of them usually turns to social media to receive diverse opinions [4]. Hence, the public views social media and search platforms as important news and opinion sources [6].

Different ideas and beliefs about the definition of democracy come with different ideas of the undesired consequences of filter bubbles [6]. These consequences range from the decay of quality of information to loss of autonomy. Before we can come up with solutions, it is essential to first elaborate on the differences in concepts and models of democracy and see how these democratic conditions can be influenced using modern technologies on the Internet.

1.1. Effects of filter bubbles on democracy

The concept of democracy can be summarized as a method of group decision-making, characterized by equality among participants at an essential stage of the collective decision-making process [6]. Different models of democracy exist, each with its own convictions of what should be at the heart of a democracy. While filter bubbles come as a concern to all models, the type of problems seen in these bubbles strongly depend on the way in which one understands the nature and value of democracy, and its requirements for objective and complete information.

Bozdag and van den Hoven [6] present the major models of democracy and their relevant conditions for success. They assess how software tools alleviate the disadvantages of the filter bubble in these models. The different models of democracy they describe are the liberal, deliberative, republican, and agonist democracy.

The classical liberal view of democracy attempts to uphold the values of freedom of choice and reason, and freedom from tyranny, absolutism, and religious intolerance [6]. Filter bubbles, according to the liberal view, pose a problem as the non-transparent filters employed by the algorithms limit the freedom of choice. Moreover, citizens are not made aware of different opinions and possible options required to make a reasonable decision, violating their autonomy and interfering with their ability to be the judge of their own interest. Finally, separation of powers and freedom of the media are in danger if algorithms are designed to serve the interests of selected groups or individuals.

Deliberative democracy can be characterized as decision-making by discussion among free and equal citizens [6]. Deliberative democrats propose that societal problems of public concern are addressed through cooperative reasoning about how to best solve them. The goal is to use common reason of equal citizens who are affected by decisions, policies or laws, instead of representing them by means of aggregation of their individual preferences. Deliberative democrats claim that creative, more optimal solutions can be found through true deliberation. Filter bubbles pose a problem for deliberative democrats because they cause low quality of information and lack of information diversity. Both of these make it more difficult to discover new perspectives, ideas, and facts.

Republicanism focuses on political liberty, and is defined as a democracy where civilians are not subject to arbitrary or uncontrolled power. Its most important implication is that people can always contest decisions made by the ruling government [6]. To be able to raise critical questions, one must be aware of something that is a candidate for contestation. Filter bubbles block incoming and outgoing information channels and thus increases the risk that citizens are unaware of important news. They also decrease the awareness of both the items people disagree with, as well as the information used for justification and reasoning.

While deliberative democracies aim for consensus, agonists see politics as a realm of conflict and argue that disagreement is inevitable even in a well-structured deliberative democratic setting. Filter bubbles form a problem for agonists as algorithms focus on relevance and may hide or remove channels that offer opinions opposing our own. Furthermore, larger audiences for unpopular viewpoints may still be reached through paid advertising. This violates the inclusion norm of agonist democracies as only a select group of people can afford to have their opinion communicated in this manner.

Overall, we can conclude that there are two main problems associated with filter bubbles for the mentioned theories of democracy. The decrease of autonomy that the public experiences, making people no longer able to judge their own interest, and the blocking of channels, making people unable to communicate their opinions and decreasing their exposure to opposing views and arguments communicated by others.

1.2. Breaking the bubble

Bubbles caused by selective exposure to news are not new. In the first half of the twentieth century, many European countries were affected by selective exposure through party press. In the Netherlands, this phenomenon was called pillarization (Dutch: Verzuiling). Pillarization stemmed from a separation into groups based on religious beliefs, but it manifested itself in all aspects of society such as schools, sports clubs, and newspapers [56]. These newspapers wrote congenial news articles, and in some occasions, politicians even functioned as journalists.

Two historical developments decreased the effect of pillarization, one technological, and the other societal. After World War II, Europe experienced a period of strong secularization. The influence of religion on society became smaller, diminishing pillarization as a side effect. Furthermore, television became more publicly available and a majority of the public started consuming their news through this new medium. The number of available television channels at this time was limited, meaning that nearly all viewers were exposed to the same news [56]. Hence, one could argue that this selective exposure problem solved itself without cal-

culated intervention.

The outlook for digital filter bubbles does not seem to be equal. When we compare the diminished autonomy and blocked channels of these historical bubbles to the current digital bubbles, we find some key differences. First, the current diminished autonomy problem is far less transparent. People were aware of their pillar in society and deliberately chose their religious beliefs. People using search engines or social media platforms often do not know that the information they see is selective [39]. The same holds for the blocked channel problem. People knew which group of people they were targeting when writing a news article for a certain newspaper, whereas now someone posting anything online has no clue of the filters affecting their audience. Current bubbles are created through small and unconscious clicks, and serve business models, not belief systems. They originate from computer science developments, and as personalization is developing into more elements of our lives, their impact will increase.

For these reasons, we believe calculated intervention is necessary. As computer science students, we believe the responsibility lies in the hands of computer scientists, as theyr are also responsible for creating the current situation.

In our opinion, today's problems of diminished autonomy and blocked channels as discussed in Section 1.1 can be alleviated through the modification of online personalization algorithms to present diverse, instead of one-sided perspectives. Computers do not semantically understand text the way humans do, so a set of article metadata (information about articles) is required to enable a computer to present diverse perspectives.

The goal of this research is to build a system that automatically collects article metadata on political content and thereby enable personalization algorithms to present users with different viewpoints on a topic.

We first discuss which metadata on political articles is necessary to be able to algorithmically select information that can be served to a reader to expose him/her to alternative viewpoints on a given subject.

After establishing this set of metadata, we employ state of the art computer science techniques to extract the data. The two categories of techniques we use are artificial intelligence and human computation techniques.

Artificial intelligence and machine learning have come a long way in the past decade. As computing power keeps developing, we are able to deploy increasingly complex algorithms that target increasingly complex problems. One such problem of high complexity is information extraction. This research investigates state of the art information extraction techniques to extract valuable information from political texts. Due to the statistical nature of such techniques, however, we are uncertain of the correctness of extracted metadata.

Fortunately, there is another kind of intelligence that we can utilize to attain a higher accuracy: human intelligence. The discipline that utilizes human intelligence for problems that machines cannot solve flawlessly is called human computation. As political articles are read by users, we are in a position where we can ask readers to help us out when our models are uncertain. Human computation is vital for checking and improving predictions, as the platforms we target usually do not publish content written by employees. Content creation is performed by millions of volunteering users. Checking and improving the data should be as democratic as its creation, and should therefore be done by citizens or 'the crowd'. The additional benefit of human computation is that it promotes critical thinking among citizens, thereby stimulating the democratic process. Finally, checking the initial machine learning predictions through human computation provides us with new training data that can be used to improve the machine learning models, creating a loop of continuous improvement. Combined, these technologies form a system that can be implemented on any content-serving platform, be it social media, search, or news.

A challenge not addressed in this thesis but nonetheless important is re-engineering the personalization algorithms in such a way that they incorporate alternative viewpoints automatically. The scope of this work is the metadata model and its collection. We aim to provide the data infrastructure that enables personalization algorithms to present alternative viewpoints, but we do not create the personalization algorithms ourselves.

We build a metadata enrichment system that can be incorporated into any platform and used to improve personalization algorithms. The aim is to show companies around the world that they can take responsibility and focus both their resources and their people on developing such systems and build personalization algorithms that benefit democracy. This way, exploring alternative viewpoints could become as simple as checking your feed.

To build a proof-of-concept system, we cooperate with the Open State Foundation on the Poliflw[1] project. Poliflw is a news platform that scrapes local (municipal) Dutch political news from websites spread all over the Internet. Collectively it hosts over 500.000 articles sourced mainly from Facebook and local party websites. The news is served through a search engine with the aim to provide transparency in municipal politics.

[1] https://poliflw.nl/

The project was funded[2] by Google through the Google Digital News Initiative fund. After launching Poliflw, the website was covered by the main Dutch news provider NOS in an article[3], showing how the website increased transparency and information flow. Poliflw is discussed further in Chapter 3.

1.3. Thesis outline

As this research is conducted in the form of a collaboration between Joost Rothweiler and Max van Zoest, the outline of this thesis report is unconventional. Chapters 1, 2, 3, 4 are shared chapters written collaboratively. Chapter 2 discusses existing solutions and introduces the novel approach taken to address the filter bubble. Chapter 3 introduces the content-serving platform on which the prototype was built, and Chapter 4 introduces the prototype itself. Hereafter, each of us devote several chapters to zoom in on our own individual part of the research; information extraction and human computation.

Chapter 5 reviews the literature on information extraction, and more specifically on the tasks of named entity extraction and topic classification. Chapter 6 discusses the design of the information extraction system implemented in Politags. Chapter 7 discusses the results of the design in terms of performance and Chapter 8 concludes the work on information extraction and its main contributions.

Finally, Chapter 9 is shared to conclude the overall work and discuss future implications.

[2] https://openstate.eu/nl/2017/07/english-poliflw-selected-for-support-by-digital-news-initiative/
[3] https://nos.nl/googleamp/artikel/2223324-online-verkiezingscampagnes-wat-staat-er-in-de-facebookberichten.html

2

Breaking the bubble: A novel approach

Chapter 1 describes the negative impact of filter bubbles on the information foundations of modern democracy. As filter bubbles became more extensively studied throughout the last decade, different researchers have worked on methods for breaking the bubble through a range of tools designed to target different problems associated with the bubbles [6]. These tools can be divided in those that aim to increase autonomy and control for the individual citizen and those that aim to improve the flow of information through diverse statements and arguments [6].

This chapter first reviews these existing solutions proposed to battle filter bubbles in Section 2.1. Next, Section 2.2 discusses the challenges these solutions still face and outlines a novel approach that aims to solve the problem at its core. Section 2.3 discusses the criteria taken into account when designing the system. Finally, Section 2.4 discusses the implications of the proposed solution.

2.1. Existing solutions

To identify state of the art approaches among recently proposed solutions, we looked at different tools discussed in a survey by Bozdag and van den Hoven in 2015 [6]. Their survey provides an overview of the different tools developed up until 2014. We elaborate on the tools deemed relevant to this work and those that provide an integral overview of the proposed solutions. Besides this overview, we have searched for additional published papers and tools for which no scientific papers were written. Search was conducted using terms such as "filter bubble", "deliberation", "biased news", "tools democracy", and "one-sided information". This section highlights the different solutions we found.

Balancer is a browser extension that shows an approximate histogram of the user's liberal and conservative page visits [36]. The aim is to provide feedback and thereby stimulate the user to balance its reading behavior. News articles are scored on a scale from conservative (-1) to liberal (1). To calculate a score, Balancer uses the article's source. Balancer employs a whitelist of sources that is human-labeled with a score on the conservative to liberal scale. The focus of Balancer is to stimulate readers' autonomy by making them aware of the balance in their reading behavior. Balancer does not provide them with new material to read. Munsen notes that the score could have been generated by a machine learning model and that when proper training data would be available, classifiers could also be trained to identify topics, parties and political figures in the articles [36].

Scoopinion is a browser extension that provides visualizations to the user to depict a summary of one's reading behavior [6]. The extension analyzes the different sources from which one reads articles and then recommends other articles to encourage the reader to read more diversely. This tool is aimed at both stimulating autonomy, as well as to diversify the flow of information to the reader. Article sources are a good indicator of one's reading habits, but as with *Balancer*, the tool could be further extended with data on topics, parties and political figures.

ConsiderIt is an online deliberation platform that aims to help people learn about political topics and trade-offs [27]. It reflects on considerations made by other voters and enables users to see how others consider trade-offs. The interface allows for users to create pro/con lists of arguments and to back existing arguments. Rather than trying to increase reader autonomy by reflecting on reading behavior, *ConsiderIt* mainly targets the increase of quality information flow by showing alternative perspectives. All data on the platform

is generated by the platform's users.

AllSides[1] is an online platform that aims to present readers of online news with three alternative perspectives on a broad range of issues and news stories. *Allsides* uses a bias-rating ranging from left to right for different news sources, which is scored according to a survey and can be corrected by users. The goal of the platform is to support the flow of quality information to readers, presenting them with different views on an issue or topic. In our opinion, the platform could further benefit from a data model richer than the left to right rating scale in order to provide readers with even more control on their reading behavior.

Echo Chamber Club[2] provides a completely different approach to the concept of providing a diverse, high quality information flow to the user. Through subscription to its newsletter, users are informed with different voices and contrary opinions to provide them with the perspectives they might otherwise miss out on. Topics and articles are carefully curated from different sources by the owner of the newsletter, and although this provides a large amount of flexibility, it does make the information sensitive to personal bias. Furthermore, users have no control over the different topics or opinions they are served.

EscapeYourBubble[3] is a browser extension that asks users whether they would like to better understand either republicans or democrats in the United States. Based on your preference, *EscapeYourBubble* inserts curated positive articles into your Facebook *feed*. Its goal is to provide people who are well aware of their political preferences with alternative opinions and arguments, thereby helping them to gain understanding. The articles presented are not topic-specific, but contain highly positive messages on activities of the different parties. *EscapeYourBubble* does not share how they find the different articles.

Overall, these solutions can be categorized based on a number of features. These include the type of platform, the type of information they aim to provide, and the data type they rely on. An overview of these features is provided in Table 2.1.

Name	Type of platform	Main goal	Data needs	Data generation
Balancer	Browser extension	Autonomy	Classification of conservative/republican web pages	Human and automatically generated
Scoopinion	Browser extension	Autonomy	Page views and whitelisted news websites	Automatically generated
Escape your Bubble	Browser extension	Autonomy / Deliverative	User preference and articles with highly positive sentiment from conservative/replublican sources	*unknown*
Echo Chamber Club	Newletter	Autonomy	Different articles that describe events from different political angles	Human generated
ConsiderIt	Online platform	Deliberative	Topics, pros, cons, and arguments	Human generated
AllSides	Online platform	Autonomy / Deliverative	Classification of sources to be left, center, or right-oriented as well as classification of articles to topics	Human and automatically generated

Figure 2.1: Overview of existing solutions.

2.2. Fighting the problem at its core

The solutions discussed in the previous section still face challenges in battling the filter bubble. First and foremost, these solutions only function for people that actually download, visit, or install them on their computer. One could therefore argue that the audience that uses these tools is already conscious of their filter bubble, and is trying to break out. This small group of people has either actively searched for tools to decrease their filter bubble problem, or has been pushed by others to do so. Through these tools, people have the opportunity to alleviate the problem. However, more is to be won with readers that are unaware of their filter bubble. These unaware readers might assume that they are basing their opinions on balanced news, while being exposed to only one side of the story.

Second, the tools discussed only *patch* the problem of uniform news delivered by social media platforms and search engines. *Scoopinion* for instance, tracks reading history, creates a fingerprint and then bases recommendations on this fingerprint. This allows for exploration outside one's reading history, but only after the user has read from sources such as social media or search. *Allsides* presents US news from the left, center, and right on the same topic, so a user gets the full range of perspectives, but only after this user has made the conscious decision to browse to *Allsides'* website. Only *EscapeYourBubble* creates changes to the actual news feed in Facebook, which is where the problem originates.

Instead of providing aware users with *patches* to the problem, new approaches to breaking the bubble should be aimed at the heart of the problem; personalization algorithms. As far back as in the Jacksonian

[1] https://www.allsides.com/
[2] https://echochamber.club/
[3] https://www.escapeyourbubble.com/

Era[4] of the early 1800s, modern newspapers in the US were given the responsibility to provide readers with objective and broad perspectives on events. This responsibility should now be incorporated into the personalization algorithms used by companies that recommend politically oriented text to their readers. However, as the content that is recommended by these companies was not written by their employees, assigning this responsibility is a challenge.

2.3. Data enrichment

We propose a system that automatically gathers the metadata required to solve the problem at its core. We argue that by enriching politically oriented text with the right metadata, we can allow for personalization algorithms to present diverse perspectives to the reader. We aim to find a method to enrich articles that can be implemented in a diverse array of platforms serving politically oriented text, be it search engines, social media platforms, or otherwise. We present a proof of concept on a test platform serving political news, further discussed in Chapter 3.

2.3.1. Enrichment criteria

To our knowledge, the idea of enabling personalization algorithms to present diverse perspectives based on political metadata is new and previous research on the required metadata is lacking. One of the contributions of this research is such a set of data enrichments. In order to define the required metadata, we propose to first define a set of criteria to which our set of metadata should uphold. Our metadata should minimally meet all these criteria, they help us define the scope of this work.

The first criterion is *algorithmic extraction*: it should be possible to automatically extract the enrichments from raw text using computer science techniques. This enables the final system to work automatically without moderation.

The second criterion is *human extraction*: humans should be able to extract the enrichments from text and the link between the text and the enrichments should be easily verifiable for a human. This ensures that human computation can be employed for verification purposes.

The final criterion is *diverse personalization enablement*: the set of enrichments should be sufficient to enable a computer algorithm to show alternative perspectives for a topic at hand. This criterion specifies the domain to battling the filter bubble by enabling diversity in personalization algorithms.

To summarize, the criteria defined are as followed:

1. A computer science technique exists to automatically extract the enrichments from text.
2. The enrichments can be extracted from text by a human.
3. The set of enrichments should enable a computer to show diverse perspectives.

2.3.2. Proposed metadata

As previously mentioned, Munson et al. [36] hint that finding political figures, positions, and topics would be informative enrichments. We agree with that statement and propose the following set of data enrichments for any piece of politically oriented text:

1. The topic of the text
2. The entities in the text:
   (a) Political parties
   (b) Politicians
3. The sentiment of the text
4. The stances with regards to the topic in the text

Our reasoning behind these enrichments is as follows: when you want to present a reader with alternative perspectives purely based on text, you want to present texts that have the same topic, but different perspectives. These alternative perspectives can be found in texts that mention other politicians and parties. For

[4] https://sites.google.com/site/jacksoniandeomacracy/political-culture-of-the-jacksonian-age

an even better assessment of the content we think that text sentiment and specific stances with regards to the topic would be beneficial. However, due to limited resources, time constraints, and the current state of artificial intelligence techniques, the scope of this research is focused on *topics* and *entities*, as we consider these to be most essential.

2.4. Implications

Based on these enrichments, a personalization algorithm could easily be modified to present texts that allow for serving of alternative viewpoints. Figure 2.2 shows how this works based on two articles and their enrichments. In this example, we see that someone is reading an article on gun control, in which Donald Trump and the Republican party are mentioned. In the database, we have articles on the same topic, mentioning a different politician and party. In this case: Hillary Clinton and the Democratic Party. Based on this data infrastructure, a personalization algorithm can easily be modified to present *article 2* when *article 1* is being read or presented.

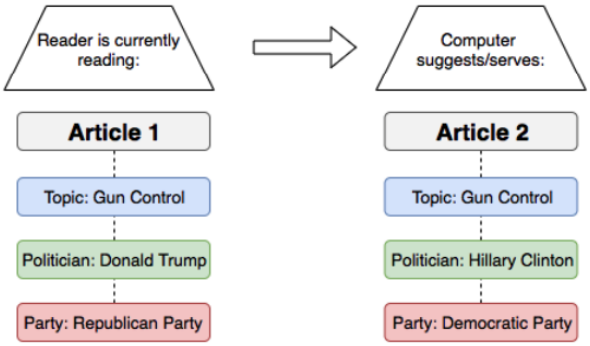


Figure 2.2: A reader reads the article on the left, the computer suggests the article on the right.

Our proof of concept creates this data infrastructure for politically oriented text on a content-serving platform. As it does so using artificial intelligence and human computation techniques, there is no need for employees to curate the data. This is essential, as curating enrichments by platform employees would be too costly and could cause selective bias. Implementing the actual personalization algorithms to serve alternative viewpoints is beyond the scope of this research.

We present a system and show that it automatically collects and verifies enrichments that enable diverse personalization. The system can be tailored to any content-serving platform, including Google or Facebook. We hope that this motivates such platforms to investigate whether they could assign the resources and time to create such systems and then modify their personalization algorithms to diversify its content. Doing so would improve the functioning of democracies worldwide by unlocking the potential that the Internet offers.

3

Use case: Poliflw

The goal of this research is to build a proof of concept system that extracts from politically oriented text the data necessary for a personalization algorithm in order to show diverse perspectives. To be able to build this system, our collaboration with the Open State foundation[1] is of key importance. More specifically, their project Poliflw[2], a search platform for local (municipal) political news in the Netherlands, is the perfect platform to build our proof of concept system.

This section introduces the Open State Foundation, highlights the key goals and ideas behind Poliflw, and provides more detail on why this project is the perfect fit for us to integrate and research our proof of concept.

3.1. Open State Foundation

The Open State Foundation is an advocate of Open Data. The following quote is part of their mission and is available on the organization website:

> Open data is information in an open, machine readable format, with no restrictions on re-use and available on the Internet. By publishing public information as open data, barriers for (re)-use are reduced and new possibilities emerge. The Open State Foundation works on digital transparency by opening up public information as open data and making it accessible for re-users. We believe that it will strengthen democracy and create substantial civic and economic value.

This mission statement aligns closely with the goals we pursue in our research.

3.2. Poliflw Goals

Political monitoring websites allow journalists and the public to control the people in power. These websites make it easy to find who represents who, what topics are being debated, and how politicians have used their power. In the past, statements and electoral programs could mainly be found in mainstream press and party websites, but nowadays, as reported by Bozdag et al. [6], social media has become an important medium for voicing political statements. This means that information is more widely scattered over the web, especially for local municipal politics, who each have their own social media page and party website. No platform exists today that brings all these sources together into one easily searchable news platform.

Poliflw solves this problem by scraping political statements from social media and news websites and serving them through a search-based news platform. This platform employs filters for users to sift through over 500,000 articles. The project was funded [3] by Google through their Google Digital News Initiative fund. Poliflw enables journalists and citizens to be the watchdog over local municipal politicians. The effectiveness of the platform was showcased in an article by the main Dutch news provider NOS[4], showing how the website increased transparency and information flow.

[1] https://openstate.eu/nl/
[2] https://poliflw.nl/
[3] https://openstate.eu/nl/2017/07/english-poliflw-selected-for-support-by-digital-news-initiative/
[4] https://nos.nl/googleamp/artikel/2223324-online-verkiezingscampagnes-wat-staat-er-in-de-facebookberichten.html

3.3. Synergy

Poliflw is ideal for us to build a proof-of-concept system. It holds a massive database of politically oriented texts and is tested by professional journalists from renowned newspapers in the Netherlands. The platform itself may alleviate filter bubbles and increase transparency, with our system increasing its effectiveness. This means that through this interaction, both systems receive an upgrade. From our point of view, we are able to work with real data that showcases a broad and unique level of debate, applicable to the lowest level of deliberation in the Netherlands. Namely that of local politicians and political parties in different municipalities throughout the country. Extracting the right data at this level of politics is far more difficult than doing it at a higher level of aggregation such as national politics. From their point of view, the Open State Foundation can use the newly created data infrastructure to create the first ever open dataset that conveys local political opinions and debate and can be very specifically searched and filtered.

4

Prototype: Politags

The previous chapter discussed Poliflw, the website on which our proof of concept is built. From now on, we refer to the actual proof of concept that consists of the information extraction and human computation engine built for this research as *Politags*. Politags is designed in a platform agnostic way so that it can easily be modified to run on any other content-serving platform and thus runs separately.

Figure 4.1 provides a high-level overview of the communication between Poliflw and Politags. Poliflw uses its own web scrapers to retrieve political news articles from the web. Using an API call to the Politags server, it sends the raw document text and expects the metadata including topics, political parties, and politicians as a response. A complete overview of the API response formats can be found in Appendix B. The metadata is extracted from text using the Politags information extraction engine that has access to a knowledge base of possible named entities and topics. The engine then stores the metadata for a given article in the Politags database. Next, when a Poliflw user reads any given article on the Poliflw website, the client-side JavaScript plugin of the human computation engine sends an API request to the server that tells the engine who is reading what article on Poliflw. The human computation engine then uses the Politags database to identify uncertain metadata identified for this article and generates questions for the reader in the user interface. Answers to these questions (verifications) are sent back through an API call and stored in the Politags database to be further processed by the human computation and information extraction engines.

This chapter discusses the main components that together make up Politags. We first elaborate on Poliflw news articles in Section 4.1 to gain an understanding of the data we work with. We then elaborate on the Politags server, database, and knowledge base in Sections 4.2, 4.3 and 4.4 respectively. After this, we note on the user interface in Section 4.5 and user data in Section 4.6.

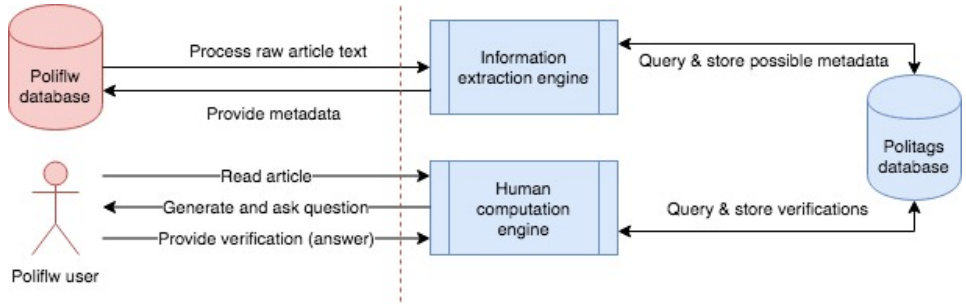


Figure 4.1: Poliflw and Politags high-level system overview.

4.1. Poliflw Articles

At the moment of writing, Poliflw hosts slightly over 550.000 political news articles stemming from different sources. These sources include news pages of local political party websites (e.g. VVD Arnhem[1], GroenLinks

[1] www.arnhem.vvd.nl

Amsterdam[2]) and local political party Facebook pages (e.g. VVD Arnhem[3]).

Articles from Poliflw are sent to Politags in HTML format, parsed to normal text, and processed. The length of the articles on Poliflw varies widely. While some articles only include a link to a web page, and therefore contain only a brief summary of the content, others include entire party programs and consist of multiple paragraphs. Based on 10.000 articles randomly drawn from the entire collection in April 2018, we concluded that the average length of an article is 1250 characters and 193 words. In terms of language, 98.8% of these articles are written in Dutch. Almost all of the remaining articles are written in English.

4.2. Server

The server of Politags is built as a RESTful API server, which means that all communication with Politags is done through web requests. This allows for Poliflw and Politags to function as completely independent systems. The programming language used is Python. The decision to use Python was mainly based on the already extensive use of this language among the Open State Foundation projects.

4.3. Database

The Politags database is used to store the knowledge base, linkings made between articles and the knowledge base (enrichments), and verifications provided to the human computation engine by Poliflw users. An overview of the Politags database design can be found in Appendix A. The database is designed as a relational database and runs on PostgreSQL.

4.4. Knowledge base

The knowledge base contains different named entities and topics we aim to identify from Dutch political articles. Named entities in the knowledge base allow us to link text mentions of named entities in articles to real-world entities. For our particular use case, we are interested in two types of named entities; Dutch (local) *politicians* and political *parties*. The *topics* we are interested in stem from a specific source later explained in this section.

Politicians

Our aim is to identify all politicians currently active in the Dutch political domain. We were able to retrieve a collection of over 11,000 politicians from the Dutch political archive available from two government open data websites[4] [5]. The information in these archives is kept up-to-date on behalf of the Ministry of the Interior and Kingdom Relations. Both archives include names, political roles, municipalities, and working addresses from politicians active in different Dutch government organizations. To our knowledge, this is the most complete dataset available on Dutch politicians.

A challenge, however, is that it contains only limited information on the politicians. For example, names only include initials rather than given names and some entries even lack initials, the municipality where the politician is active, or political roles. To address this problem, we have combined the collection retrieved from the archive with a list of over 50,000 local politicians running for the municipal elections in 2018. This list was constructed as an initiative of a number of organizations including the Open State Foundation and the Dutch Broadcast Foundation (*Nederlandse Omroep Stichting, NOS*). Through this process, we were able to identify given names of over 45% of the politicians in our original collection. Appendix C provides a more elaborate overview of the attributes that represent politicians in our knowledge base.

Parties

Due to challenges associated with crawling news articles from different sources, the Poliflw prototype focuses on a subset of the political parties active in the Netherlands. This subset of parties includes the 10 largest parties currently active across all of the Netherlands. An overview of these parties can be found in Appendix C. For each party, our knowledge base contains the full name of the party as well as the abbreviation often used in text. No distinction is made between the same parties active in one city or another.

[2] www.amsterdam.groenlinks.nl
[3] www.facebook.com/Arnhem.VVD/
[4] https://almanak.overheid.nl/
[5] https://gegevensmagazijn.tweedekamer.nl/

Topics

The topics we aim to identify are those used to label the parliamentary questions published as open data[6] and maintained by the Dutch Ministry of the Interior and Kingdom Relations. This collection is particularly interesting for our research as it is a list of topics that politicians at a local level are dealing with. Moreover, the list is kept well updated and specific to Dutch politics.

4.5. User interface

An important element of the human computation engine is its user interface. To keep Poliflw and Politags separate, we took the approach of creating a JavaScript plugin that renders all user interface elements into the original Poliflw web page. As with the data enrichment processing, the rendering occurs the moment a user opens and starts reading an article.

4.6. User data

We conduct anonymized user behavior research and serve users with relevant questions every time they read an article. Poliflw does not require readers to login, as this would be a barrier for people to start using the website. Thus, in consultation with the Open State Foundation, we decided to store unique cookie identifiers that uniquely identify a user's browser, but contain no other user information. A user could have multiple unique identifiers when he uses the website on multiple browsers, devices, or when he deletes cookies and returns to the website. Hence, this method is not perfect, but it does provide us with more information without the need for a user registration system.

[6] https://officielebekendmakingen.nl/

5

Background: Information extraction

The explosive growth and popularity of the World Wide Web has resulted in a huge amount of data publicly available. However, due to the heterogeneity and lack of structure for these sources, access to this huge collection of information is often limited to browsing and searching [10]. The field of research concerned with automated translation from raw text to structured data is called information extraction (IE) [10]. This task is performed using a form of natural language processing (NLP), which takes raw textual content as input and extracts fixed-type, unambiguous snippets as output. The extracted data may be used directly for display to users or for improving information access tasks [13, 51]. IE is different from the task of information retrieval, which aims to retrieve relevant documents from a collection, as IE is concerned with generating structured output that can be used for further post-processing [10].

This chapter focuses on two IE tasks relevant to the problem this work aims to address. The first task is the extraction of named entities from text and linking these entities to real-world entities through disambiguation. This is discussed in Section 5.1. The second task is the classification of text into topics which is discussed in Section 5.2.

5.1. Named entities

Named entities are real-world objects such as people, organizations or locations that can be viewed as entity instances (e.g. Amsterdam is an instance of a location). The task of algorithmically extracting named entities from textual data is called Named Entity Recognition (NER) [37]. NER is broken down into two main phases: entity detection and entity typing (also called classification). A common follow-up step to NER is Named Entity Disambiguation (NED). NED aims to link mentions of entities in text to known target entities in the real world. This is also referred to as entity linking or entity resolution [13]. This section elaborates on the two steps by discussing state of the art NER and NED techniques, evaluation methods, and existing challenges.

5.1.1. Named entity recognition

The aim of NER is to identify boundaries of entity mentions in documents, as well as to classify these entities into predefined categories such as personal names, locations, and organizations [51]. Research on NER gained substantial popularity with the shared task introduced for the Conference on Computational Natural Language Learning (CoNLL) in 2003, where the challenge was to classify different named entities from news articles [16]. To illustrate how NER can be a challenging task, we consider the following sentence:

> **Apple** chief **Steve Jobs** was born in **SF**.

This sentence contains three named entities that demonstrate some of the complications associated with NER. First, "Apple" is short for the computer company Apple Computer Inc., which is an organization. Secondly, the mention "Steve Jobs" consists of two words which together make up a single entity mention. Thirdly, "SF" in this case, is an acronym for the location San Fransisco.

Approaches

Algorithmic approaches towards NER can be divided into *rule-based*, *supervised*, and *semi-supervised* NER. The remainder of this section goes further in depth on these approaches.

Rule-based NER One of the initial algorithmic methods to identify named entities from text is based on a set of rules, usually written in the form of regular expressions for different categories of entities [17]. Rules, in this case, are written by language experts. To exemplify how such rules are constructed we consider a number of basic rules to identify names of people and organizations.

To identify names of people, one can imagine that there are properties in the text that can help an algorithm do so. These include, for example, mentions of titles such as "Mr.", "Ms.", and "Dr.", initials such as "A." and "D.", common first and last names, and capitalized words. Knowing what are possible titles and common names we can use a combination of the following set of rules to identify names of people:

- `title [ capitalized-word+ ]`
- `[ capitalized-word initial capitalized-word ]`
- `[ common-first-name capitalized-token ]`

Where `[...]` identifies the boundaries of the names and `capitalized-word+` indicates that multiple consecutive capitalized words are considered the same entity mention. Common names and titles are matched against an externally constructed dictionary. In NLP, such dictionaries used to support initial tagging are commonly referred to as *gazetteers*.

A similar set of rules can be written to identify mentions of organizations in text. Mentions of companies in English text can often be recognized based on final tokens and suffixes such as "Company", "Associates", "Inc.", and "Corp.". Most mentions of companies, however, usually appear without contextual clues and therefore a list of known company names is required to achieve good performance. The same holds to identify mentions of locations [17].

Although it is possible to build effective NER systems that are primarily rule-based, problems that arise are that new names may pop up over time and that large lists may include entries which are simply capitalized forms of common words. For example, Kissing and Hell may refer to towns in Germany and Norway respectively. Furthermore, some degree of skill is required to construct good rules and a large amount of labeled training data is necessary to effectively evaluate the addition of new rules [17].

Supervised NER As discussed in the previous paragraph, rule-based approaches require annotated data for effective evaluation. A natural question to ask is whether we could automatically learn rules based on such annotated data [17]. Current effective NER approaches are based on this idea and are commonly referred to as supervised NER systems [29].

Nymble [5] was one of the first supervised NER systems built on Hidden Markov Models, with a single state for each entity type and an additional state called "other". To capture contextual information, conditional probabilities were used. Maximum entropy methods were later introduced by Borthwick and the Edinburgh group [17].

Later, local features on words were incorporated into the model to gain the benefit of both machine learned and hand-coded features. These include lexical features, dictionary features, and shape features [17]. Lexical features state whether the current word has a specific value. Dictionary features capture whether the current word is in a particular dictionary (as discussed in rule-based approaches). Shape features capture the form of the token in terms of capitalization or numeric values.

Most names represent the same type wherever they appear. Therefore, *context* in text encoded in the form of long-range features can help to solve mentions found difficult to classify [17]. One simple approach to capture the *context* is to apply a two-phase strategy. On the first pass over a sentence, document, or entire corpus, the system builds a cache that records each name and its type assigned on this pass. On the second pass, these types assigned are incorporated as features to represent the dominant name type [17].

Semi-supervised NER Problems that arise with rule-based and supervised learning NER approaches are the lack of adaptability to changing and alternative forms of text. Therefore, much of the recent work has been targeted towards semi-supervised learning techniques. These techniques learn word representations from large unlabeled corpora in an unsupervised manner and use these representations as input features for supervised training. As a result, these systems become much more adaptive to different corpora as they do not rely on labeled data to learn representations from.

One can categorize techniques for learning word representations into the following categories: clustering-based, word embeddings, and distributional word representations. These categories form the basis for semi-supervised NER and representing words in a more semantic feature space. However, research is still being conducted on combining these techniques with character-level feature representations of words [29, 45].

Clustering-based Clustering-based techniques are applied in a variety of research [8, 52]. One clustering algorithm that has shown to achieve high performance is the Brown algorithm. This is a hierarchical clustering algorithm that clusters words to maximize the mutual information of bigrams [52]. A downside of the algorithm is that it is based only on bigrams, not considering word usage in a wider context.

Word embeddings Word embeddings are among the most popular techniques and work by transforming words into feature vectors in such a way that the embedding captures morphological, syntactic, and semantic information [45, 55]. The type of word embedding created depends on the size of the context window used. A larger context window creates topic-oriented embeddings whereas a smaller context window creates syntax-oriented embeddings [16]. A large part of the research conducted on word embeddings is based on the popular and fast to train word embedding (word2vec) method proposed by Mikolov et al. [33]. This approach is based on the Skip-gram model which uses the current word to predict the surrounding window of context words, weighting nearby context words more heavily than more distant ones. In doing so, the model trains vectors that seek to approximate a function that computes the probability of seeing an outside context word vector given a center word vector [18]. The use of word vectors has become more popular and successful than more traditional representations. Currently, the most widely used representation is that of GloVe vectors [40].

Distributional word representations Distributional word representations are constructed in the form of a co-occurrence matrix, where each row represents a word in the vocabulary. This word in the matrix is represented using a feature vector composed of the different columns, each representing some context. In constructing the co-occurrence matrix there is a handful of possible design decisions to be made, such as the choice of context type and type of frequency count. As for the context type, one has to make a decision whether to use a left or right window and what window size to use. As for the frequency count, one could, for instance, choose to use a raw, binary, or *tf-idf* count [52]. Because it is not well understood what settings are appropriate to use in different NER systems, and because clustering-based and distributed word representations have shown to be more successful, distributional word representations are not often used [52].

Evaluation

NER systems are expected to be applied to a diverse range of documents like historical texts, news articles, patent applications, etc., and therefore should perform robustly across multiple domains [42]. In order to evaluate the quality of NER systems, several measures have been defined over time.

While accuracy on the token level is one possibility of measuring performance, this measure comes with two major problems. First, because the larger majority of tokens in a text is not part of any named entity, the baseline accuracy achieved as a result of always predicting token as 'not an entity' is already over 90%. This is extremely high while no named entities are actually extracted. Second, not extracting the entire named entity, but only part of it, is not properly penalized as only extracting a person's first name, but not its last name, results in an accuracy score of 50%.

Alternative measures are required to more properly score NER models. This resulted in variants of well-known performance metric to be proposed by academic conferences such as CoNLL [44]. Their variant of the F1-score, $F_{\beta=1}$, is defined in Equation 5.1. Precision is measured as the percentage of named entities found by the system that is correct. Recall is the percentage of named entities present in the evaluation corpus and found by the system. Named entities are only considered correct if the system finds an exact match with the entry in the evaluation corpus. The result of this is that any prediction that misses a single token, includes one that is not part of the entry in the evaluation data, or has the wrong class (e.g. 'location' instead of 'person') is a hard error and does not contribute to either precision or recall performance.

$$F_\beta = \frac{(\beta^2 + 1) * precision * recall}{(\beta^2 * precision + recall)} \tag{5.1}$$

Other evaluation models proposed are based on a token-by-token matching [15]. These models enable the scoring of partially overlapping matches but reward only fully exact matches. Through this, they allow for a more fine-grained evaluation, taking into account the degree of mismatch in non-exact predictions.

Challenges

While research on NER has come a long way and has continued to be heavily studied for over a decade now, there are scenarios in which it remains difficult to apply these NER techniques. These scenarios include the extraction from social media content and the extraction of long-tail entities.

Social media content Classical NER systems have shown to dramatically under-perform on social media content due to the nature of its short and noisy form [13, 16]. For example, Liu et al. [30] report an F1-score of 45.8% when applying Stanford NER[1] to their constructed Twitter corpus, and Ritter et al. [43] even report an F1-score of 29%. Therefore, researchers have proposed new systems specifically designed to extract named entities from short and noisy content [16]. Most of these systems rely on hand-crafted features such as gazetteers. Godin et al. [16] propose a method solely based on word embeddings but note that adding gazetteers would likely increase their F1-score performance. Strauss et al. [50] provide an overview of the 2016 WNUT shared task[2] where five out of eight proposed solutions make use of gazetteers.

Long-tail entities Another area in which NER systems show their limits is when trying to classify domain-specific and long-tail entities [32]. We define long-tail entities as the entities that occur in the text on a low-frequency and have a low probability of occurrence. Recognizing these type of entities from text remains an open challenge for two reasons. First, the long-tail nature of the entities in existing knowledge bases and targeted document collections makes it difficult to apply statistical models and external knowledge. Second, there is a high cost associated with the creation of hand-crafted rules or human-labeled training datasets for supervised machine learning models [32]. Mesbah et al. [32] hint that using external knowledge in the form of gazetteers can help in recognizing long-tail entities from text.

5.1.2. Named entity disambiguation

After determining the expressions in text that are mentions of entities using NER, a follow-up IE task is concerned with mapping ambiguous entity mentions in a text (simply *mentions* for short) onto a set of known target entities. This task, which is commonly referred to as Named Entity Linking (NEL) or Disambiguation (NED) [13, 38] has received a noticeable amount of attention since the early 2000s [21]. The former terminology (NEL) stresses the importance of linking a mention to an actual instance in a given knowledge base. Some authors [9], however, prefer the latter term (NED), as it focuses more on the potential ambiguity among several possible instances. For this same reason, this work also refers to the task as NED. Ambiguity, in this sense, refers to the number of referents (instances for NED) that a target word has [9]. To illustrate how NED can be challenging, we consider the following sentence:

> The **Python** of **Delphi** was a creature with the body of a snake. This creature dwelled on **Mount Parnassus**, in central Greece.

To translate this data to a structured knowledge base, we must be able to determine that *Python* earth-dragon in Greece mythology and not the popular programming language, *Delphi* is not the auto parts supplier, and *Mount Parnassus* is in Greece, not in Colorado [21]. To solve this problem, researchers make use of different knowledge bases as a disambiguation target. Commonly used knowledge bases include entity databases such as Wikipedia and Linked Open Data resources such as DBpedia, YAGO, and Freebase [13].

Approaches

This section discusses different approaches proposed in literature. In general there are two main categories of dealing with the NED problem; *single named entity disambiguation (single-NED)*, focused on disambiguating one entity at a time without considering the effect of other named entities, and *collective named entity disambiguation (collective-NED)*, where all mentions of entities in the document are disambiguated jointly [3].

Single-NED Single-NED approaches focus on the use of local contextual features of the entity mention to compare the mentions and its context with the features of candidate entities in the knowledge base [3]. The entity with the highest certainty match is the one linked to. Two of the first to tackle the problem of NED this way were Bunescu and Pasca [7], who measured the similarity between the textual context of the mention

[1] https://nlp.stanford.edu/software/CRF-NER.html
[2] https://noisy-text.github.io/2016/

and Wikipedia categories associated with the candidate. Common measures used to compute relatedness of context and entity include the dot product, cosine similarity, Kullback-Leibler divergence or Jaccard distance [21]. However, these measures are often found to be duplicate or over-specific [21]. As an extension to this, He et al. [21] proposed a method to learn context entity association using a complex neural network architecture. A more straightforward extension of the basic context-similarity approach is proposed by Milne [34] who uses richer features for the similarity comparison. In particular, instead of using the similarity function directly, Milne introduced a supervised learning step to better estimate the weights of the underlying feature labeled training data. Rather than learning context entity association at word level, Cucerzan [12] took advantage of the topical coherence between a candidate entity and other entities in the context. Researchers have further built upon this idea [20]. However, the *one topic per entity* assumption makes it difficult to scale to large knowledge bases [21].

Collective-NED The second line of work focuses on resolving ambiguous mentions within the same context simultaneously [21, 38]. One of the main arguments for following this approach is that different named entities in a document can help to disambiguate each other [3]. However, because of the high combinatorial cost for the mapping of all entities, finding an exact solution becomes expensive. To solve this problem, different researchers [3, 38] have proposed to work with graphs where edges represent mention-entity pairs and entity-entity pairs. Finding the joint probability distribution of all mappings in a probabilistic interpretation remains an NP-hard problem, but relaxations of the problem through approximations has shown promising results [3]. Looking at the two different approaches of single-NED and collective-NED, Ratinov et al. [42] show that even though collective approaches can be improved, local methods based on only similarity of context and entity are hard to beat.

Evaluation

Commonly used evaluation datasets to use for NED tasks are the *TAC* datasets[3]. *TAC* queries consist of a mention string, as well as the source it was drawn from. The label expected to be returned is a reference to a unique *TAC* knowledge base node or `NIL` when there is no corresponding node in the knowledge base. Documents in this dataset are drawn from news and blog collections [19]. Key metrics for evaluating NED systems include accuracy as the official *TAC* measure to evaluate end-to-end systems, as well as knowledge base accuracy and `NIL` accuracy. Other metrics of interest include candidate count, candidate precision, candidate recall, `NIL` precision, and `NIL` recall [19]. Comparing different NED approaches remains a challenge due to the fact that NED systems rely on another algorithm to first extract named entity mentions from text [19].

Challenges

There are a number of challenges that make the disambiguation of named entities in text difficult. Two of the major challenges are considered trade-offs between accuracy and runtime efficiency and the disambiguation of long-tail entities.

Accuracy vs. runtime efficiency One of the major challenges still faced in state-of-the-art NED systems is the trade-off between accuracy, runtime efficiency, and scalability. Using relatively simple contextual features in a single-NED system we can avoid the use of combinatorially expensive algorithms that use collective inference for jointly mapping all mentions in the text.

Long-tail entities Long-tail entities are difficult to disambiguate due to the probabilistic nature of the algorithms proposed [23, 38]. Hoffart et al. [23] for example, propose a method that takes into consideration prior probabilities of entities but they are only able to estimate these probabilities well for popular entities based on Wikipedia anchor links. This means that the initial ranking of the prior is accurate, but drops sharply, making it difficult to correctly identify the less prominent long-tail entities. Thus resulting in low accuracy for the long tail of less prominent and more sparsely connected entities [23].

A challenge not similar but related to that of recognizing long-tail entities is the knowledge base used to identify the actual entities behind the mentions. The system's performance is dependent on the type of entities that one aims to recognize but also on the information that is available on those entities. The quality of the knowledge base, and how one deals with limited information, is an important challenge for our research

[3] https://tac.nist.gov/data/index.html

in particular. Literature available is on recognizing Wikipedia entities for which there is a 'rich' set of information available. If no such rich information is available on the disambiguated entities, a different approach must be considered.

5.2. Topic classification

Automatically classifying the semantic content or topic of text, commonly referred to as text classification, is one of the critical problems in IE and NLP more generally [1, 26]. Its aim is to assign a document to some predefined category based on its content [1]. Implementations are, among others, used for automatic email filtering, medical diagnosis, newsgroup filtering, documents organization, indexing for document retrieval, word sense disambiguation, and news recommendation [1, 2]. This last use case is particularly interesting for us, also since newspaper articles have shown to provide a particularly good opportunity to learn text classification models. This is due to the relatively coherent semantic content, and the enormous amount of labeled data that is publicly available [26].

The focus of this survey is mainly targeted towards machine learning models that make use of supervised or semi-supervised learning techniques where the target labels are known. Supervised models aim to leverage standard machine learning techniques commonly used for classification tasks in general. Using a set of training records, where each record is labeled with a class value drawn from a set of predefined labels, a classification model is built to relate features in the underlying record to one of the class labels [1].

Different fields of research related to this, but not reviewed in this survey, are unsupervised techniques aimed at topic modeling and mapping. Topic modeling is the practice of statistically discovering the abstract topics that occur in a collection of documents [41, 48]. Frequently used methods for this are Latent Dirichlet Allocation (LDA) which aims to split a corpus into clusters of words that should together represent the abstract representation of a topic [48]. New documents can then be classified as a combination of such topics. This method is particularly interesting when training data is not available or when labels are uncertain or unknown. This approach is not further discussed in this review because we aim to classify documents to be part of specific predefined categories rather than a more abstract concept of a topic.

This section first discusses different approaches to represent raw text in a more computer-readable format in Section 5.2.1. Next, it discusses commonly used classifiers in Section 5.2.2, domain adaptation techniques in Section 5.2.3, and evaluation methods in Section 5.2.4.

5.2.1. Text representation

In order to classify the content of our documents, these documents need to be represented in a format that allows for machine learning algorithms to be applied. These representations can be constructed in a number of ways. The two main lines of work can be divided into those that rely on term frequencies and those that rely on word embeddings.

Term frequency

One approach to transform text into term frequency feature vectors is to convert a collection of text documents into a matrix of term counts. The terms used in this matrix are the ones extracted from the complete vocabulary of the training corpus after performing preprocessing and removing duplicates. The result is that documents are represented as exact frequencies at which different terms across the entire vocabulary are used in the document. As an example, one can consider a training vocabulary of `{dog, fox, jumped, lazy, quick, zebra}`. The sentence "the zebra jumped" would then be encoded as `[0,0,1,0,0,1]`. Drawbacks of this approach, however, include the fact that common words are not normalized according to occurrence across the entire training corpus, and that long documents can reach high counts across a larger number of terms.

In order to fight these drawbacks, term counts are commonly replaced by Term-frequency inverse document frequency (*tf-idf*), which is a numerical statistic that reveals the importance of the term to a document in the collection [53]. The value of *tf-idf* increases proportionally to the number of times a term appears in a document but is counteracted by the frequency of the term in the corpus. The main assumption is that the more a term occurs in a document, the more relevant it is to the document and the more the word occurs throughout all documents in the collection, the more poorly it discriminates between documents [1].

n-grams By encoding the term-frequency representations discussed, documents are represented as a form commonly referred to as *bag-of-words*. This means that the internal structure of the document, and informa-

tion on the order in which words are combined throughout the text, are not used in the model. In order to retain some of the structure that is present in the document, researchers have proposed a similar encoding that, rather than using single words, uses n-grams.

The n-gram model can be used to store this spatial information within the text. A bigram (2-gram) model parses the text into different units and stores the term frequency of each unit as before. For example, the sentence "John likes new movies" is parsed into the following units: ["John likes", "likes new", "new movies"]. Conceptually, we can view *bag-of-words* models as a special case of the n-gram model, with $n = 1$. n-grams with $n > 1$ can be treated in the same way in our term frequency model as representations where $n = 1$.

Preprocessing Preprocessing is an important step in the text classification pipeline, as a large portion of the words in the training data do not hold information that can help us classify the text as they contain no topical content. These low information words include for instance dates, numbers, stop words, etc. The goal of preprocessing is to reduce the level of noise in the training data. Popular preprocessing methods include stop word removal and stemming.

Stop words can be described as the most common words in a text that give no particular meaning to the text [53]. Examples of stop words are "the", "in", "a", and "with". The motivation for removing stopwords as a preprocessing step is that stop words make the text look heavier while these words contain limited importance for analytics [53]. By removing stop-words, one can reduce the dimensionality of the analytics problem while keeping almost all key information intact. Different methods used for stop word removal include methods where words from precompiled lists are searched for and removed, Z-methods which remove most frequent words (*tf-high*) or singleton words (*tf-1*), and more complex learning methods such as the Mutual Information (MI) method [53].

Stemming methods are used to identify the root (or stem) of the words in the corpus. As an example, the words "connected", "connect", "connecting", and "connection" can all be stemmed to the word "connect" [53]. The motivation for stemming words in a corpus is to capture the same meaning of words into a single term and thereby have accurately matching stems and to reduce the number of words and thereby reducing time and memory space [53]. A wide variety of stemming algorithms is available that can be categorized as truncating, statistical, and mixed methods. Truncating methods aim to remove the suffixes or prefixes (commonly known as affixes) of a word and are usually rule-based. Examples include the Lovins, Porters, Paice/Husk and Dawson stemmers. Statistical methods also remove affixes but are based on an implementation of a statistical algorithm. Examples include stemmers based on N-grams, Hidden Markov Models, and clustering techniques (YASS Stemmer) [53]. Mixed models apply a combination of rule-based and statistical approaches.

Feature reduction After applying preprocessing algorithms, the number of features in term-frequency representations usually remains excessive as the training datasets become larger. Therefore, feature reduction methods have been proposed to remove features that contain low levels of information. The aim here is to reduce the number of features without affecting classifier accuracy [1].

The two methods for for feature reduction include *feature extraction* and *feature selection* [1]. Feature extraction aims to reduce dimensionality through a transformation of the original feature space into subset features. It maps high-dimensional data on lower-dimensional data. Popular algorithms include *Principal Component Analysis* and *Singular Value Decomposition*. Feature selection techniques, on the other hand, aim to select the most important features from the original feature space based on criteria such as *information gain* [1].

Word embeddings

While term frequency models have shown to perform well, these systems do not store any semantic value of the terms stored in the model, thereby throwing away a lot of the available information. In order to compensate for this, just like named entity recognition as discussed in Section 5.1, text classification tasks can benefit from using methods based on word embeddings to represent documents.

To use word embeddings to represent sentence-level, and eventually document-level representations, researchers have come up with different approaches [33]. One straightforward approach is to use the weighted average of the feature vectors of all word embeddings in the document. The problem, however, with this approach is that word order is lost in the same way as it is lost in any standard *bag-of-words* model. A more sophisticated approach is to combine the word vectors in an order given by a parse tree of a sentence [47]. This method, however, has shown to only work for sentences as it relies on parsing.

To better represent documents using word embedding techniques, Mikolov [28] came up with the *Paragraph Vector* model (also commonly referred to as Doc2Vec) which is capable of constructing representations from text of variable length and thereby is able to model sentences, paragraphs and documents. Furthermore, it does not require task-specific tuning of word weighting functions and does not rely on the parsing of complete sentences. Mikolov presents promising results on text classification tasks, convincingly beating *bag-of-words* models with a relative improvement close to 30%. Another line of work by Chen et al. [11] is focused on extending the *Paragraph Vector* model by directly learning embeddings from raw documents together with their tags (topics) in the learning phase. A new document is first embedded using a Doc2Vec component and tags are assigned by searching the nearest tags embedded around the document.

5.2.2. Classifiers

There is a number of classifiers commonly used for text classification tasks. These include Naive Bayes, Support Vector Machine (SVM), and Decision tree classifiers. The Naive Bayes classifier assumes independence among features. The model is simple and requires little training time. However, performance is low in scenarios with few features. SVM performs well on text classification tasks and has shown to generally outperform the Naive Bayes classifier for text classification tasks. Decision tree classifiers do not assume independence among features and perform well on text classification tasks. However, these classifiers become more difficult to apply when feature vectors contain a large number of features [1].

5.2.3. Domain adaptation

Often in text classification tasks, we do not have training data available from the domain which we aim to apply the model to. Domain adaptation considers this setting in which the training and testing data are sampled from different distributions [24]. The domains are assumed to be related but not identical. Directly applying models trained on the source domain to the target domain often results in poor performance.

Despite the importance of domain adaptation in NLP more generally, there are no standard methods for solving this problem [25]. This section further elaborates on commonly used domain adaptation approaches used in text classification tasks and highlights the scenarios in which they are expected to perform best.

Instance weighting One general approach to solving the problem of domain adaptation is to apply instance weighting techniques, assigning different weights to instances in both domains and thereby minimizing the expected loss over the distribution of the data in the target domain [24]. Alternatively, one can also resample training instances from the source domain so that they better represent the same class distribution as the target domain [24]. However, in order to effectively make use of both approaches, one needs to know the class distributions in the target domain. In some studies, it is assumed that this information is known, which is rarely the case in reality. To solve this, one can apply expectation maximization techniques to estimate the class distribution in the target domain [24].

Hein [22] argues that instance weighting is not useful in high-dimensional feature spaces or when applying complex model families. Storkey [49] argues that in general, instance weighting provides only a computational benefit rather than a modeling benefit, as it allows one to select a model family with low complexity [24]. This makes us believe that applying such instance weighting techniques on a text classification model that relies on *tf-idf* document representation is not an effective approach, as feature vectors quickly become thousands of features.

Semi-supervised learning Another approach is to ignore the domain difference and treat labeled data in the source domain as the same data in the target domain. Doing so allows us to treat the problem as a semi-supervised learning problem and apply common semi-supervised learning techniques. The subtle difference between domain adaptation and semi-supervised learning is that the amount of labeled data in semi-supervised learning is expected to be small and can be large in domain adaptation. Also, the labeled data may be noisy in domain adaptation and is expected to be reliable in semi-supervised learning [24].

A common approach in semi-supervised learning is to make use of self-labeling [31]. Self-labeling is an iterative method that trains an initial classification based on labeled data in the source domain, uses this model to classify instances in the target domain, and then uses the estimated labels for building the final model [31]. This method has been applied in both supervised and unsupervised domain adaptation settings. Variations exist as to how the samples in the target domain are added and used. For example, one can either use hard labels or label probabilities, train for a fixed number of rounds or until convergence and incorporate

all new labels on each round or only the most certain predictions [31]. In the scenario where the classification model trained on the source domain already performs well on the target domain, this is a common approach.

Feature representation So far we have assumed the representation of features in both domains to be the same. However, these can be different across the two domains. A change of representation can affect both the marginal distribution and the conditional distribution. One can assume that under a transformation of the representation, the two domains can become more alike in terms of distributions. One transformation to achieve this is to apply feature subset selection. Satpal and Sarawagi [46] propose a method specifically targeted towards domain adaptation, where features are selected to minimize an approximated distance function between the distributions in the two domains. This approach does require a large amount of labeled data from the target domain [24].

5.2.4. Evaluation

A common measure for text classification models is accuracy. This measure is appropriate if the percentage of documents in each class is high (e.g. > 10%). However, this is not a good measure for "small" classes because classifying everything as samples from most common classes can lead to the highest performance and at the same time defeat the purpose of building a classifier. Therefore, when dealing with small classes, precision, recall, and F_1 are more appropriate measures to consider.

When processing a collection with several two-class classifiers, as often done in a multi-label text classification problem, we want to eventually compute a single measure that combines the performance of the individual classifiers. Macro- and micro-averaging are two common methods of doing so. Macro-averaging computes a simple average over classes and provides a good sense of effectiveness on small classes in the test collection. Micro-averaging pools per-document decisions across classes and therefore provides a good measure of effectiveness on the large classes in a test collection.

5.3. Research

From Chapter 2, we learned that the main goal for this research is to enable simple algorithmic presentation of alternative perspectives to news readers and that the first steps into achieving this goal are to extract named entities and topics from a politically oriented text. This research is focused on how one can build an information extraction engine that automatically extracts these enrichments from text, taking into consideration the main challenges presented in this review. We state the following main research question:

> *How can we enrich politically oriented text using information extraction techniques to enable algorithmic presentation of alternative perspectives?*

The main challenges stated in this literature review are summarized as well as how these challenges apply to our specific domain. Based on these challenges, we pose a number of sub-questions that must be answered first in order to provide answers to the main research question.

Named entities We see a challenge in the extraction and disambiguation of long-tail entities, especially since our aim is to apply the research to Dutch local political content, which is diverse in terms of the number of active politicians and political parties. The problem is that single-NED methods perform poorly when disambiguating long-tail entities. Collective-NED methods, on the other hand, are compute-intensive and rely on a lot of data about linkings and information such as prior probabilities. Furthermore, a rich knowledge base is key to developing strong disambiguation systems. We state the following sub-question:

> *How can we apply named entity recognition and disambiguation techniques to identify politicians and political parties in the Netherlands from Dutch political text?*

Topics The challenge in recognizing topics comes from the fact that we do not have labeled data for our target domain. Therefore, it is a challenge in itself to find labeled data that is representative for our target domain in terms of both topics and type of text. After finding such labeled data, we need to consider domain adaptation techniques to see whether we can successfully adapt our model to fit the target domain data which we aim to classify. We state the following sub-question:

> *How can we apply text classification techniques to extract political topics from Dutch political text?*

Continuous improvement The aforementioned challenges come with the fact that there is a lack of proper training data to build statistical models on. The system that we design has access to a *human computation system* that can verify predictions and return verifications of our predictions back to the system. We can use these verifications to further improve our initial classification results. The question that remains is:

> *How can we design the system to enable learning from a given human computation module and thereby support continuous improvement?*

6

Design: Information extraction in Politags

This chapter builds on the literature reviewed in Chapter 5 to create the Politags information extraction system designed to extract the enrichments discussed in Chapter 2. These enrichments include named entities and topics. The information extraction system was built to allow for the automatic extraction of these enrichments. This enables Poliflw to retrieve the following structured information based on the content of a news article:

1. Disambiguated Dutch politicians mentioned in the article and present in the knowledge base.

2. Disambiguated Dutch political parties named in the article and present in the knowledge base.

3. Political topics the news article concerns.

This chapter discusses the different parts of the design. We describe design decisions based on the literature discussed in Chapter 5 and iteratively go over the different steps for each information extraction task. In each iteration, we argue how different design choices impact performance and analyze the results before moving on to a next step. The goal of this approach is to make the design understandable and reproducible. It also allows for people who conduct future research to understand how different design choices impact performance.

Section 6.1 discusses the design of the named entity recognition and disambiguation model to extract politicians and parties. Section 6.2 discusses the topic classification model. Section 6.3 discusses how we can enable continuous improvement based on the Politags human computation system.

6.1. Named entities

To extract named entities from text, we took the approach of first identifying named entities in text using NER techniques and then linking these named entities to entries in our knowledge base using NED techniques.

An alternative method is to skip the NER step and to identify entries in our knowledge base in the text based on a direct comparison between names in the knowledge base and mentions in the text. However, we took the approach of using NER because our knowledge base is not complete in terms of attributes, with first names of politicians missing. Therefore, this method would match against incomplete mentions (e.g. "Churchill" when the text mentions "Winston Churchill") and thereby make the disambiguation process more difficult.

This section discusses the implementation of the NER model in Section 6.1.1 and the implementation of the NED model in Section 6.1.2.

6.1.1. Named entity recognition

The aim for the design of the NER model was to have a system specifically tailored and top-performing for the Dutch political domain. We discuss the NER implementation used, as well as our method of tailoring the implementation to perform better on our target domain.

NER implementation

Different open-source NLP libraries implement state of the art NER functionality. Instead of reinventing the wheel, we chose to make use of these existing libraries to perform NER in our system.

NLP libraries considered include NLTK, TextBlob, Stanford CoreNLP, and spaCy. NLTK is the most widely used Python NLP library, but in our opinion does not have an interface as intuitive as other libraries. TextBlob is built on top of NLTK and provides an intuitive interface to its functionality, but is considered more appropriate for quick prototyping rather than serving production NLP applications. Stanford CoreNLP is considered a production-ready NLP suite but is written in Java instead of Python. As stated in Section 4.2, the preferred server-side programming language is Python. There are Python wrappers available for CoreNLP but these are written by the community and are not officially supported.

spaCy has achieved state of the art results on performing NER on Dutch text with a precision score of 86.59%, a recall score of 87.19%, and thereby an F1-score of 86.89%[1]. Also, it is fast compared to other frameworks [2] and has a straightforward and powerful Python API interface. The implementation is trained on the WikiNER[3] corpus, a free annotated training corpus based on Wikipedia articles. Entity types supported include 'person', 'location', 'organization' and 'miscellany'. We see no downside to using spaCy's NER model in our system and therefore choose to use it as our NER model.

Extending gazetteers

As stated in 5.1.1, current challenges in NER are commonly addressed by incorporating external knowledge into existing models in the form of gazetteers. spaCy implements a *phrase matcher* that can be used to incorporate this external knowledge. It can be used in combination with the existing spaCy NER pipeline and allows us to perform exact searches for named entities in the text. By implementing an additional spaCy *phrase matcher*, we extend the gazetteers already present in the model and expect a gain in performance by injecting domain knowledge.

Implementation The *phrase matcher* is implemented by adding full names of politicians in the knowledge base from which we know the first name and as well as names and abbreviations of parties in the knowledge base. We only incorporated names of politicians from whom we know the full names (including first names) and not last names on their own because adding last names on their own would result in a decrease in quality of extracted information. To illustrate this, we again consider the following sentence:

> **Apple** chief **Steve Jobs** was born in **SF**.

Solely looking at the named entity of the person *Steve Jobs*, we expect for the initial spaCy model to successfully identify the boundaries of this named entity mention (`[S...s]`), as well as the type (`Person`). If we would insert our knowledge base record (`([Steve Jobs]: Person)`) in the spaCy *phrase matcher* we can be certain that we correctly extract this entity from this sentence. However, if we would not know from our knowledge base that *Steve Job*'s first name is *Steve*, we would only be able to insert (`([Jobs]: Person)`) in the spaCy *phrase matcher*. This would result in only the partial extraction of the named entity, as *"Jobs"* will be extracted and considered a *complete* named entity on its own. In that case, the extracted named entity is wrong and our entity disambiguation process becomes more difficult as we miss part of the information on the named entity.

Evaluation A common approach to evaluate NER performance is to use a hand-labeled NER evaluation dataset. Entries in such a dataset include start and end position of named entity mentions (mention refers to the phrase in the text that represents the named entity) and their entity type. As an example, the sentence "Who is Winston Churchill?" is labeled as `["Winston Churchill", 7, 23, 'person']`. However, to follow this approach one would have to hand-label a large collection of articles. This is a time intensive task and a lot of data would have to be labeled to draw reliable conclusions in terms of performance.

Instead, to test the performance of the *phrase matcher*, we propose a different approach that avoids the work of hand-labeling large amounts of data. This approach consists of the following steps:

1. Extract 10,000 articles recently published on Poliflw.

[1] https://spacy.io/models/nl
[2] https://spacy.io/usage/facts-figures
[3] https://github.com/dice-group/FOX/tree/master/input/Wikiner

2. Apply the initial NER system (without additional gazetteers).
3. Count the number of extracted entities that exactly match entries in our knowledge base based on entity type ('person' for a politician or 'organization' for a political party) and full names of politicians or names of parties. We consider these matches as successfully extracted named entities.
4. Extend the initial NER system by adding the gazetteers to the system. These gazetteers include the names of the parties and politicians such that the NER model can match text based on these new gazetteers.
5. To test performance, recount the number of exact matches found in the evaluation articles.

Through this approach, we can determine the difference in the number of politicians and political parties correctly extracted for the entries in the knowledge base of which we know the full names. An improvement in the number of matches found is assumed to represent an improvement in NER performance.

It is unique to our domain and knowledge base that this approach provides reliable results. For other domains, the knowledge base may contain common words that would be recognized as named entities. However, the combination of first and last names is assumed to never represent a phrase of such common words. For political parties, the same holds as political parties are given unambiguous and uniquely identifiable names.

Recently published articles are preferred over older articles because our knowledge base contains the currently active Dutch politicians. This makes us believe that using recent data gives us the most representative change in performance.

Table 6.1 summarizes the results of implementing the spaCy *phrase matcher*. From these results, we can conclude that, by implementing the *phrase matcher*, the model becomes better at recognizing the specific named entities we aim to extract as the number of exact matches in every case increases after adding the *phrase matcher*.

Type of match	**NER**	**NER + *phrase matcher***	**Increase**
Exact politician name matches	2,579	2,985	15.74%
Exact party name matches (abbreviations)	16,598	17,158	3.37%
Exact party name matches (full name)	4,944	5,090	2.95%
Named entities labeled as "Person"	31,343	32,845	4.79%
Named entities labeled as "Organization"	40,615	42,127	3.72%

Table 6.1: Results of implementing the spaCy phrase matcher

6.1.2. Named entity disambiguation

The NED model expects as input the output from the NER model. This output of an article includes the different entities, their positions, and their labels. Labels predicted by the NER model are mutually exclusive and include 'person', 'organization', 'location', and 'miscellany'. Since we are interested to disambiguate politicians and political parties, the focus is on the disambiguation of entities labeled as 'person' (politician) or 'organization' (political party). Only entities labeled with one of these two labels are used as input for the disambiguation model.

As discussed in Section 5.1.2, single-NED techniques based on similarity of context and entity are hard to beat by collective-NED techniques. Therefore, we implement a single-NED approach to disambiguate and link recognized entities to entries in our knowledge base. The implementation of the disambiguation system is divided into three parts. These parts include candidate generation, candidate disambiguation, and candidate linking. The remainder of this section discusses each part separately.

The part on candidate linking elaborates on the performance of the end-to-end system based on a handcrafted evaluation dataset.

Candidate generation

For a recognized named entity labeled as 'person', the first step in the disambiguation process is to define a list of candidate politicians. This is done for performance reasons. The knowledge base consists of over 11,000 politicians. Common last names of politicians in our knowledge base occur as often as 52 times (See Table 6.2 for an overview of the most common names). This means that even for the most common last names

recognized in articles, 99.99% of the politicians in our knowledge base are unlikely matches. The candidate generation process aims to remove this 99.99% to improve the run-time efficiency of the disambiguation model.

The process of narrowing down the list of possible candidates must not miss out on the actual candidate. Missing the actual candidate would result in the system not being able to find the right entity linking. This section discusses the design decisions made and evaluates the final candidate generation design.

Last name	Count
van Dijk	52
de Jong	46
Jansen	44
de Vries	38
Janssen	37

Table 6.2: Most common last names of politicians in the Politags knowledge base

Implementation The candidate generation process for parties is straightforward. There are only 10 political parties in our knowledge base and candidate generation is considered a method for improving run-time efficiency. Because this number is so small, we do not necessarily have to lower the number of candidates and can simply compare them all. Therefore, we return all political parties in the knowledge base as candidates for each entity labeled as 'organization'.

The candidate generation process for politicians is tricky as we do not know which parts of a named entity labeled as 'person' (e.g. "Gert-Jan van der Boom") represents a first name, middle name, or last name. Parsing names to extract such information is especially difficult for Dutch names that span hyphenated or multiple words like "Gert-Jan", "van der Boom" or "Veld-Fonk". To avoid the need for such information in the candidate generation process, we propose the following algorithm to match parts of the mention to last names in the database:

1. Split the mention into a list using spaces as delimiter.
   - `"J. van der Boom"` → `["J.", "van", "der", "Boom"]`
   - `"Dinie Veld-Fonk"` → `["Dinie", "Veld-Fonk"]`
2. Check if any candidates are found for this list. Someone is considered a candidate if either:
   (a) The concatenated words in the array are *equal* to the candidate's last name.
   (b) The concatenated words in the array are *part* of the candidate's last name and the candidate's last name contains a hyphen ('-').
3. If no candidate is found and the list is not empty: remove the first entry from the list and repeat step 2.
4. If at least one candidate is found, return all candidates found for this part of the name.
5. If list is empty, return an empty list of candidates and stop the disambiguation process.

The assumption for the first matching rule (2a) is that politicians are mentioned using the last name and that no last name is misspelled in the text. When not making this assumption, a different strategy would be to generate a larger list of candidates based on partial (or fuzzy) matches in last names. However, we found that such an approach showed to be less effective in generating the right candidates. This is because the spaCy NER model still produces errors, classifying capitalized common words (e.g. "Huis", Dutch for "House") as being names of people. Through our approach, we can already filter out these mistakes while maintaining high-quality candidates for correctly recognized named entities.

The second matching rule (2b) is included because it is common for Dutch women to hyphenate their spouse's name with their own birth name. As a result, entries in our knowledge base contain such hyphenations. Articles usually only mention either of the two names separately, which this rule successfully accounts for.

Evaluation At this point, we can identify a list of all possible candidates for mentions in each article. We evaluate the performance of the politician candidate generation implementation on the dataset also used to evaluate our end-to-end system. This dataset contains all the target identifiers of politicians mentioned in the articles in the dataset. There is no need to evaluate the party candidate generation process as all parties are considered candidates.

One could argue that using the same dataset in the design as well as in the evaluation process may introduce bias and result in overfitting of the final model. However, because our candidate generation process is in no way a statistical model and because the defined rules are in no way specific to the dataset used, we do not see a problem in using the same data.

By generating a list of possible candidates for each article, we can evaluate the NER and candidate generation process based on the metric of recall. This metric tells us the number of candidates in the evaluation data that was actually found through the candidate generation process. It therefore tells us how well we are able to extract all *possible* candidates from the text at this point.

We summarize the evaluation process as follows:

1. For each article in the evaluation dataset, extract *all possible candidates* through NER and candidate generation.
2. Based on the extracted candidates, calculate the recall of candidates in the evaluation dataset actually found through this method.

Through this process, we found that the current implementation achieves a recall of score of 95.08% for candidate politicians. Furthermore, it generates an average of 10.14 candidate politicians per recognized named entity. After analyzing the results we observe the following mistakes:

- **"Burgemeester Kolff"** not recognized as person
- **"Mady** `(new line)` **van Hemel"** not recognized as person

Mentions of named entities not recognized by the NER model result in no candidate generation for these mentions. Therefore, we cannot solve this problem by adjusting the implementation of the candidate generation system. Taking into account these errors in the NER model, we can conclude that the candidate generation system is successful in generating the right candidates.

Candidate disambiguation

The next step is to disambiguate the named entities text based on article context and information on candidates in the knowledge base. We built the candidate disambiguation system in such a way that it outputs certainty scores for each named entity-candidate pair. This section discusses the way in which we compute these certainty scores. The next section on candidate linking discusses the certainty threshold that determines whether or not a named entity is actually linked to a candidate in the knowledge base.

For political parties, the process of disambiguation is straightforward as there is no ambiguity in the names of the parties. Certainty $C(m,c)$ is computed as the maximum string similarity S between the mention in the text m and the candidate party names. These names include the full party name (c_{full}) and the abbreviation (c_{abb}). String similarity is computed using the *SequenceMatcher* implemented in the Python *difflib* library[4]. The equation is defined in Equation 6.1.

$$C(m,c) = max(S(m,c_{full}), S(m,c_{abb})) \tag{6.1}$$

For politicians, the candidate disambiguation process is less straightforward. The remainder of this section discusses the candidate disambiguation model built for politician candidate disambiguation.

Similarity Features As mentioned at the beginning of this section, we make use of single-NED techniques that rely on the similarity between the named entity and candidate. The initial model that we implement has no access to large amounts of training data and is therefore limited in the type of similarity measures available. We discuss additional measures that can be implemented as part of the continuous improvement model in Section 6.3. The initial disambiguation model implements the following similarity measures between a named entity mention and a candidate:

[4]https://docs.python.org/2.4/lib/sequence-matcher.html

1. **Name:** Maximum string similarity ratio between different parts of mention and candidate properties.
2. **Who name:** Similarity ratio between mention and candidate name based on Python library *whoswho*[5]
3. **First name:** Whether or not the first name of the candidate is part of the mention.
4. **Initials name:** Whether or not the first initial of the candidate equals the first letter of the mention.
5. **Role:** Whether or not the role of the candidate is mentioned in the document.
6. **Party:** Whether or not the party of the candidate is the same as the source party of the document.
7. **Location:** Whether or not the municipality of the candidate is same as source location of the document.
8. **Gender:** Whether or not the gender of the candidate is same as guessed gender of mention's first word based on Python library *gender-guesser*[6].
9. **Context:** Jaccard similarity between a list of named entities recognized in the document, source location of the document, and source party of the document and a list of all known properties of the candidate.

The implementation of how we calculate values for the different features can be found in Appendix D. We designed the similarity measures in such a way that all knowledge that we have on candidates from our knowledge base, as well as metadata that we already have on the articles, is incorporated.

Classification model The similarity measures are combined in a feature vector that is used as input for the classifier. The benefit of abstracting information on named entity-candidate pairs into feature vectors is that we can leverage every 'correct' or 'incorrect' linking made to train a classifier without having to worry about underlying named entities, candidates, and articles. To see which classifier best fits our use case, we implement a number of commonly used models that can output label probabilities (certainties) and evaluate their performance in terms of accuracy.

Evaluation To train and evaluate the different models, we randomly draw 500 articles published on Poliflw in the period of January 2018 until April 2018. We choose to draw articles from this particular period because our knowledge base contains politicians active in this particular period. We take the following steps to create a labeled dataset used for training and evaluation:

1. Apply the NER model on these 500 articles.
2. For each of the named entities recognized and labeled as 'person':
   (a) Generate a list of candidates using the candidate generation system.
   (b) If any candidate is found:
      i. Hand-pick the correct candidate that the named entity actually refers to.
      ii. If more candidates are found, pick the highest probability candidate from the list of candidates based on the calculated value of the *Who name* similarity feature.
      iii. For each picked candidate, insert the text of the named entity, the full name of the candidate, feature vector (based on the similarity features), and the label ('correct' for correct linkings, 'incorrect' for incorrect linkings) into the dataset.

To prevent overfitting and protect against biased results, we remove any duplicate named entity-candidate pairs for which the similarity feature vector is equal. The result is a dataset of 780 named entity-candidate pairs with a similarity-based feature vector and the label 'correct' for correct linkings and 'incorrect' for incorrect linkings. An example of the first 5 entries in this dataset is given in Table 6.3.

To gain insight into the distribution of the data across the 9-dimensional (9 features in feature vector) space, we transform the dataset to two dimensions through *Principal Component Analysis* (PCA). PCA is a

[5]https://github.com/rliebz/whoswho
[6]https://pypi.org/project/gender-guesser/

Named entity (mention)	Candidate full name	Similarity Feature vector	Label
Maarten van Ooijen	M. van Ooijen	1, 1, 1, 1, 1, 0, 1, 0.25, 1	correct
Maarten van Ooijen	M.A. van Ooijen	0.77, 1, 0, 1, 0, 0, 0, 0, 1	incorrect
Joëlle Gooijer	J.K. Gooijer-Medema	0.59, 1, 1, 0.79, 1, 0, 1, 0.22, 1	correct
Mark Rutte	M.H.J. Janssen-Rutten	0.71, 1, 0, 0.44, 0, 0, 0, 0, 0	incorrect
Mark Rutte	M. Rutte	1, 1, 1, 1, 0, 1, 0, 0, 0	correct

Table 6.3: Example of training and evaluation data disambiguation classifiers

statistical procedure that uses an orthogonal transformation to convert the dataset into a set of values of linearly uncorrelated variables called principal components. By transforming the high-dimensional space into two principal components, we can visualize the data in a 2-dimensional scatter plot. The result is depicted in Figure 6.1. The 'correct' class is depicted as diamonds and the 'incorrect' class as red crosses. This plot shows us that the data is relatively separable in the reduced feature space. This means that the data is equally or more separable in the original 9-dimensional feature space. The remainder of the design is based on the original 9-dimensional feature space.

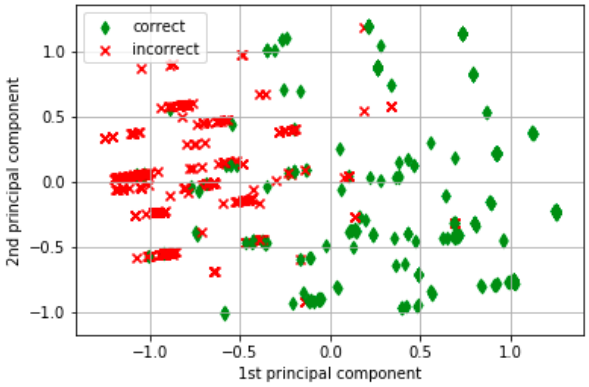


Figure 6.1: Dataset visualized in first two principal components.

Classifiers On the original 9-dimensional dataset, we train a number of different classifiers (implemented using scikit-learn[7]). The classifiers considered include a K-nearest neighbor (KNN) , support vector machine (SVM), and Decision tree classifier. Each classifier has its own specific way of separating the data and together they provide us with insight into the method most appropriate for the samples in this dataset.

Cross-validated learning curves with accuracy as scoring function provide us with a better understanding of how well the different classifiers perform for different amounts of training data. An overview of these learning curves is depicted in Figure 6.2.

From Figure 6.2, we conclude that a similar accuracy score is achieved by both KNN models and the SVM. KNN-7 achieves a cross-validated accuracy of 93.64% with 20% of the labeled samples used as evaluation data. KNN-3 scores 93.40% and the SVM scores 93.14% using the same evaluation setting.

Because of the similar performance among these two types of classifiers, we can choose the type we feel most suited for the problem. We believe KNN models to be best fit for our use case as they are flexible and able to correctly classify outliers associated with particular entries in our knowledge base when plenty of training data is available. Furthermore, a KNN-7 classifier is preferred over a KNN-3 classifier because we want our model to output a broad range of probability estimates. KNN models can estimate probabilities based on the number of 'correct' and 'incorrect' K nearest neighbors. The larger the number of nearest neighbors used, the larger the set of possible probabilities becomes (fraction with whole numbers and value for K as denominator). For these reasons, we continue our design and analysis implementing the KNN-7 as our disambiguation classifier.

[7] http://scikit-learn.org/stable/modules/svm.html

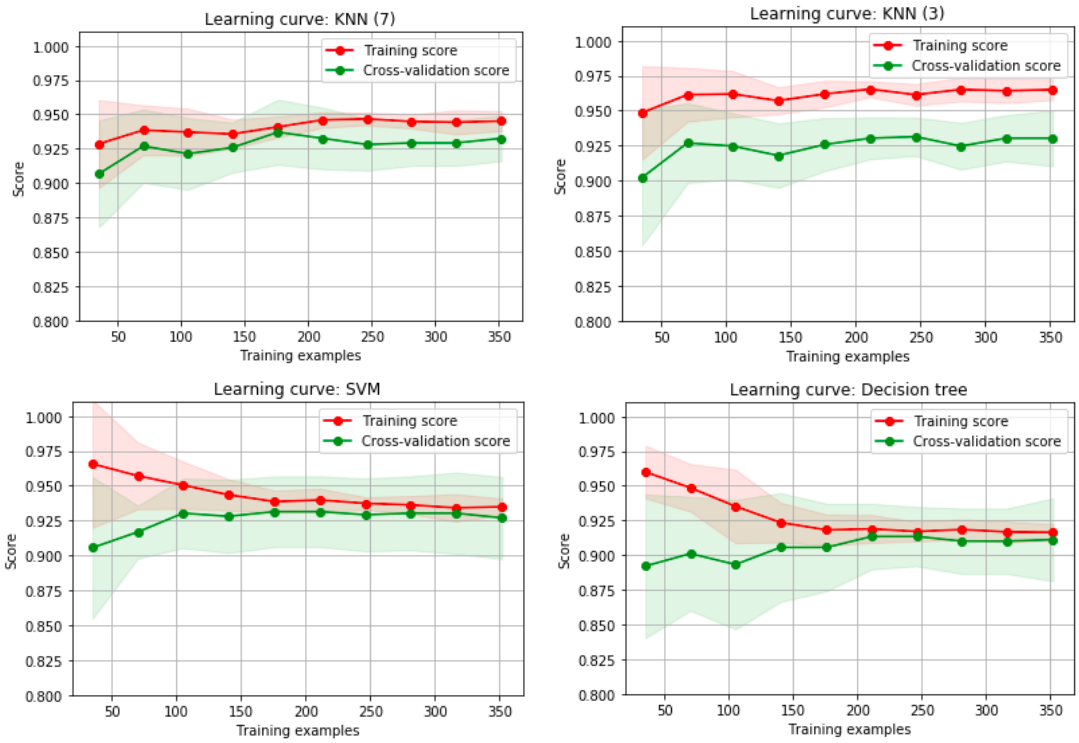


Figure 6.2: Learning curves for different disambiguation classifiers.

Feature importance It is important to evaluate the different features in terms of positive contribution to the classification process as some features may only introduce noise. To gain this insight, we build and evaluate 9 different classifiers based on one single feature at a time. We train the same KNN-7 classifiers and evaluate the results in terms of accuracy. A higher accuracy achieved on the single feature suggests a larger impact on the classification performance. The cross-validated accuracy for the different features is depicted in Figure 6.3.

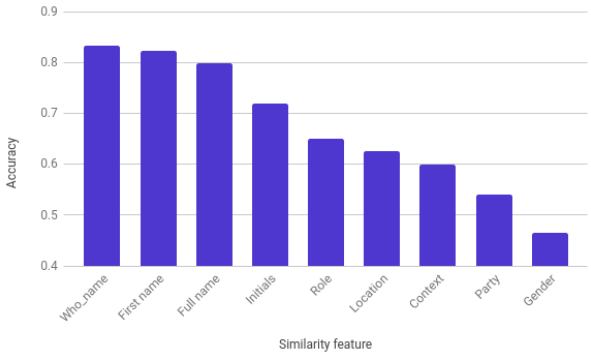


Figure 6.3: Per-feature classifier accuracy with KNN-7 classifier

The dataset contains a total of 780 samples, of which 449 are labeled as 'correct' and 331 as 'incorrect'. This means that a classifier that would always predict 'correct' would already achieve an accuracy of 57.56% (449/780). Features that score lower or close to 57.56% should be further analyzed to make sure they do not add more noise than value.

Figure 6.3 depicts an accuracy value lower than 57.56% for one similarity feature; *gender*-based similarity. We evaluate the system with and without this feature to see whether the feature actually contributes to classification performance. The results are summarized in Table 6.4.

From these results we conclude that, although the *gender* feature is not able to separate the two classes on

its own, that combined with the other features in the feature space it is able to contribute to the performance in a positive manner. We therefore choose to include the *gender* feature in the feature vector.

The remaining features all achieve accuracy scores higher than 57.56% on their own and are therefore included in the final model.

Feature vector	**Accuracy**
Gender feature included	93.64%
Gender feature excluded	92.25%

Table 6.4: Difference in performance after excluding gender feature.

Stress test Another metric used to evaluate the system is the accuracy of the classifier in situations where the number of possible candidates is larger. The assumption here is that when there are more candidates, there is a higher level of uncertainty among the most probable candidates in our knowledge base.

To test this, we add another value to the samples in our dataset, which describes the number of candidates found for the particular named entity during the candidate generation process. In our knowledge base, this number ranges from 1 to 62. We separate all samples in the training data into bins for the different number of candidates. These bins are constructed in steps of 10. This means that we have the following bins: [1:10], [11:20], [21:30], [31:40], [41:50], [51:60], [61:70]. Each of these bins is separated into a training and test dataset. Next, the KNN-7 classifier is trained on the training dataset. For every bin, we calculate the cross-validated mean accuracy of the classifier, as well as the average number of candidates for samples in that bin. We plot the average number of candidates against the cross-validated accuracy together with a regression line over the accuracies of all bins. The area around the plotted accuracy line depicts the standard deviation. The results are depicted in Figure 6.4.

From this figure, we see that although the performance decreases as the number of candidates increases, the performance remains high. The performance remains high because the number of possible candidates does not directly affects the disambiguation classifier. The samples in our dataset for which there is a higher number of candidates and are labeled as 'incorrect' are expected to be more similar to the actual 'correct' candidate because there are more options to pick the most similar candidate from. However, the model decides on a single named entity-candidate pair basis whether or not the named entity refers to the candidate at hand.

Another observation to be made is that the standard deviation is high when the number of candidates per named entity is around 33 or 52. High standard deviation may be the result of overfitting. However, to draw conclusions as to whether the classifier is actually overfitting we can refer back to the learning curves depicted in Figure 6.2. The KNN-7 learning curve shows low average standard deviation near the right end of the graph and depicts a training and test error close to each other. Based on this, we conclude that we do not have to worry about overfitting and that the average standard deviation across the entire training set is well within acceptable boundaries.

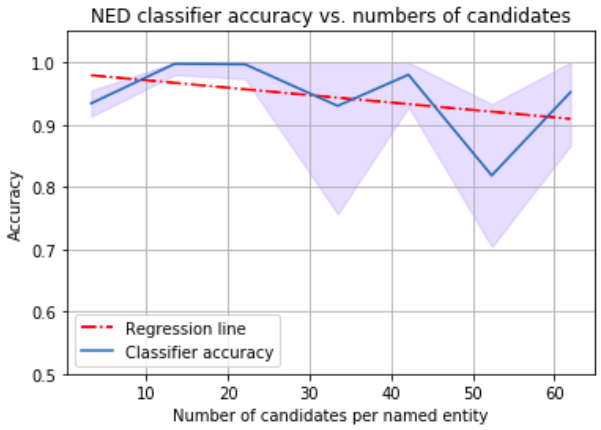


Figure 6.4: Stress test results based on different number of candidates per named entity.

Candidate linking

With candidate generation and disambiguation covered, this section discusses the actual linking of named entities to candidates. The candidate linking system is implemented with two main goals in mind.

The first is that candidate linking is the final step in our named entity enrichment model and therefore should conclude which candidates parties and politicians are actually the ones named in the document. We do so by returning the candidate with the highest disambiguation certainty for each named entity in the text, in combination with a certainty threshold.

The second is to link an additional number of candidates that are found to be less probable. This is done so that when the candidate with the highest certainty is rejected through Politag's human computation system, we have an additional candidate to ask confirmation on.

The remainder of this section elaborates on the optimal certainty threshold for both parties and politician linkings separately. The performance is evaluated using the same data as the candidate generation system but this time in an end-to-end fashion where we compare all disambiguated predictions from the complete system against true expected values.

In consultation with the Open State Foundation, we decided that precision and recall of the named entities system are equally important measures. This is because we do not want to insert incorrect data, but we also do not want to miss a larger part of the named entities in the text. To this end, it is necessary to choose a certainty threshold that takes both into consideration. The F1-score, representing the harmonic average of the two measures serves as the appropriate final measure.

Parties Figure 6.5 summarizes the party candidate linking performance for different certainty thresholds as well as the point at which an optimal F1-score is achieved. The graph depicts the trade-off between precision and recall.

The highest recall achieved by the system is is 86.52% when no threshold is applied. In this setting, we return the most probable party for every named entity recognized as 'organization'. However, the precision in this setting is low as all named entities labeled by the NER module as 'organization' are linked to the a party in the knowledge base. As the threshold increases, we observe a slight decrease in recall but a larger gain in precision.

The optimal F1-score is found at a certainty threshold of 70%. The precision, recall and F1-score for this threshold are 96.25%, 86.52%, and 91.15% respectively.

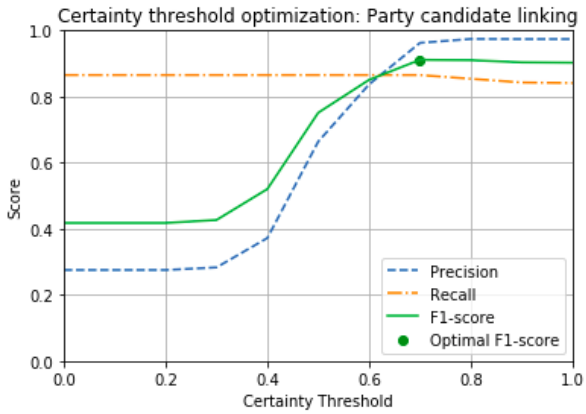


Figure 6.5: Certainty threshold optimization for party candidate linking.

Politicians Figure 6.6 summarizes the politician candidate linking performance for different certainty threshold as well as the point at which an optimal F1-score is achieved. It depicts the same trade-off between precision and recall as for the party candidate linking. We make use of a KNN-7 classifier which estimates candidate certainty in terms of the number of nearest neighbors labeled as 'correct' candidates. Because of this we can set 7 different certainty threshold values ranging from 14.29% (1/7 neighbors) all the way up to 100% (7/7 neighbors).

The highest recall achieved at the lowest threshold is 85.94%. This percentage is lower than the recall found in the section on candidate generation because we now only return the most certain candidate per

named entity. An optimal F1-score of 89.57% is found for a certainty threshold of 57% (4/7). This is also the boundary at which our classifier itself would classify a candidate as 'correct' as the majority of 7 nearest neighbors is labeled as 'correct'. We therefore choose to use this same threshold as the final value for our candidate linking system. The precision and recall for this certainty threshold are 95.45% and 84.38% respectively.

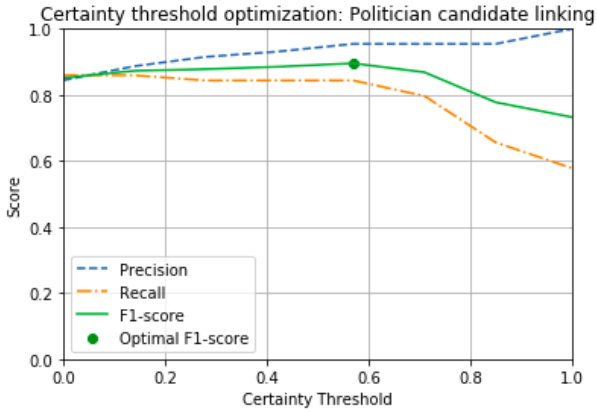


Figure 6.6: Certainty threshold optimization for politician candidate linking.

6.1.3. Summary

The final design consists of two consecutive tasks to extract named entities from text. The first task is named entity recognition which is implemented using the NER functionality incorporated in the spaCy NLP library. To improve the performance of this extraction, we implemented a *phrase matcher* to incorporate domain knowledge and search for politicians based on full names and political parties based on name and abbreviation in the text.

The second task is named entity disambiguation. This task links named entities labeled as 'person' or 'organization' by the NER model to entries in the knowledge base. This task is further divided into the subtasks of candidate generation, candidate disambiguation and candidate linking.

Candidate generation for politicians is done by checking whether different parts of recognized named entities labeled as 'person' match a last name in the knowledge base. For named entities labeled as 'organization', all parties in the knowledge base are considered a candidate.

Candidate disambiguation for politicians is done based on similarity measures incorporated as features in a KNN classifier. For political parties, the disambiguation process is based on a string similarity ratio between named entity mention and candidate name in the knowledge base.

The output of the candidate disambiguation subtask is a certainty score between the named entity mentioned in the text and a candidate in the knowledge base. The subtask of candidate linking is concerned with deciding which candidates to return as actual 'correct' linkings. This is done based on a threshold determined for both politicians and parties separately.

6.2. Topic classification

This section discusses the design implemented to extract topics from Poliflw articles (target domain). No large amount of labeled training data from the target domain is available. Therefore, we followed the approach of training a text classification model on a source domain as discussed in Section 6.2.1 and applying it to the target domain as discussed in Section 6.2.2.

6.2.1. Source domain

As a source domain, we chose to make use of labeled training data that is publicly available from the House of Representatives (Dutch: Tweede Kamer der Staten-Generaal). This data consists of parliamentary questions, published as open data[8] and maintained by the Dutch Ministry of the Interior and Kingdom Relations. The parliamentary questions in the dataset are questions asked by members of the House of Representatives to the government and cover a wide range of political topics. This set of topics consists of 111 unique topics.

[8]https://officielebekendmakingen.nl/

Motivation We used this particular dataset as our source domain for a number of reasons. First, the articles are expert-labeled using different topics specific to the domain of Dutch politics. This means that there is a coherent style in labeling the documents, making the data more reliable than for instance data for which labels are user-generated.

Second, both the content of the articles as well as the labels assigned are similar to the articles we aim to classify in the target domain in terms of writing style and wider theme. All articles in the source domain are concerned with politics and are written by politicians just like our target domain. The topics are also specific to this domain, describing a broad range of political topics but nothing that is not related to politics.

Third, articles in the dataset originate back to as far as the year 2000 and over 10,000 articles can easily be extracted. As we discuss classifier performance, it becomes evident that the size of the training dataset is crucial in order to achieve high performance.

One major difference between the two domains is that articles in the source domain contain more words than articles in the target domain. Articles in the source domain contain an average of 315 words and 2,058 characters, whereas articles in the target domain contain an average of 193 words and 1,250 characters.

Extraction The documents are extracted using a web scraper implemented by the Open State Foundation. This web scraper allows the user to provide a start and end date and extracts all questions published within this period as well as answers to the questions. Because documents that contain answers are a duplicate of the question with short one-sentence answers inserted we choose to remove the answers from the dataset used. The final dataset used in our design contains 13,945 documents published between January 2012 and April 2018.

Alternatives To our knowledge, there are no alternative datasets more suitable to use as our source domain. One alternative considered is a dataset of articles published on the VNG (Association of Dutch municipalities) website [9]. The VNG represents the interests of Dutch municipalities and publishes news articles concerned with a wide range of topics on a more local level. The published articles go back from as far as 2016 and consist of 4,100 labeled articles. We could use a web scraper to retrieve this data in a similar fashion as we do with the parliamentary questions. However, because the number of articles published by the VNG is much smaller and the number of different topics is larger, we consider this dataset to be less suitable for us to train a classification model on.

Classification model

This section first discusses the text representation format that is implemented to represent articles in the source domain. Next, it discusses the classifiers that are implemented to classify the articles.

Text representation We reviewed different ways of representing documents as feature vectors in Section 5.2.1. The two lines of work discussed include term frequency models and other representations based on word embeddings. For our scenario, we believe a term frequency model to be more suitable. The reason for this is that the frequency model represents the text in terms of keywords, and thereby is expected to be able to generalize better among a large variety of sources. Better generalization capabilities also make term frequency models better suited for applying a trained classifier in a different domain.

We have analyzed the performance for both *count* and *tf-idf* vectorization algorithms and optimized performance based on n-grams, preprocessing, and feature reduction techniques used. The methods used for optimization, as well as an analysis of how different implementations affect performance can be found in Appendix E. The final implementation of the term frequency model can be summarized as followed:

- **Vectorization algorithm**: *tf-idf*
- **n-grams**: $n = 1$
- **Preprocessing**: capitalization of all words in the dictionary
- **Feature reduction**: none

[9] https://vng.nl/onderwerpenindex

The result of the vectorization algorithm is depicted in Table 6.5. This table summarizes the highest *tf-idf* value words on a per-topic basis for three different topics. From this table, we see that the highest ranking terms per topic are very representative of the different topics and that there are no unwanted terms like numbers and stop words in the dictionary.

Tf-idf term rank	Bestuur | Nederlandse Antillen	Bestuur | Gemeenten	Cultuur en recreatie | Kunst
1	CURAÇAO	GEMEENTEN	MONUMENT
2	ARUBA	GEMEENTE	SUBSIDIE
3	SINT	VNG	SCHILDERIJ
4	BONAIRE	RAADSLEDEN	KUNSTWERELD
5	MAARTEN	GEMEENTELIJKE	SPOTPRENT
6	EILANDEN	BURGEMEESTER	DETENTIECENTRA
7	RAFFINADERIJ	BRP	PRIVÉBEZIT
8	SABA	PERSOONSGEGEVENS	OPHEF
9	EUSTATIUS	MONDELINGE	ROCKBAND
10	BES	BRINKMAN	RIJKSMUSEUM

Table 6.5: Top tf-idf ranks for different topics in the dataset.

Classifiers We evaluated different classifiers commonly used in text classification tasks and discussed in Section 5.2.2. These include Naive Bayes, SVM, and Decision tree classifiers.

The different classifiers are trained in a *one-vs-rest* fashion, where one classifier is trained for each class to distinguish this class from the *rest* of the classes. As training data, we use 80% of the extracted documents in the source domain. 20% of the articles are used for evaluation. The parameters of each classifier have been optimized using an exhaustive grid search across a number of parameter settings.

Metrics used to evaluate the classification model are the standard text classification measures of precision, recall, and F1-score. These can be either macro- or micro-averaged among class labels. Micro-averaging is preferred over macro-averaging because there is a significant imbalance in class labels. Macro-averaging would compute the metrics independently for each class and take the average, treating all classes as equal, whereas Micro-averaging aggregates the contributions of all classes to compute the average metric.

Initial evaluation showed that the Naive Bayes classifier performs poorly. This classifier is therefore excluded for further analysis. Performance achieved by the other classifiers in terms of F1-score is depicted in the form of a learning curve in Figure 6.7. Cross-validated precision, recall and F1 scores for the different classifiers are depicted in Table 6.6. Based on Figure 6.7 and Table 6.6, we conclude that the SVM is the top performing classifier compared to the other models.

One interesting property of the learning curves depicted is that the training performance remains close to 100% (99.95%) for both the SVM and decision tree classifiers. The reason for this is that in the enormous number of features that we have, the classes in the training dataset are nearly perfectly separable. When the training and testing error are as far apart as they are in our case, we speak of overfitting on the training data. However, we were unable to compensate for this by adding additional training data as the number of features remains too large.

We could close this gap by reducing the number of features through feature reduction techniques but this showed to decrease performance (see Appendix E). A different strategy would be to increase the degree of regularization used in our classifiers. However, we found such an approach to also decrease cross-validated classifier performance and not affect the training error. We therefore chose to continue without further adjusting the classifier.

Topics classifier	Precision	Recall	F1-score
SVM	79.27%	53.35%	63.78%
Decision tree	75.68%	38.35%	50.90%
Random forest	71.32%	34.78%	46.76%

Table 6.6: Cross-validated topic classification performance.

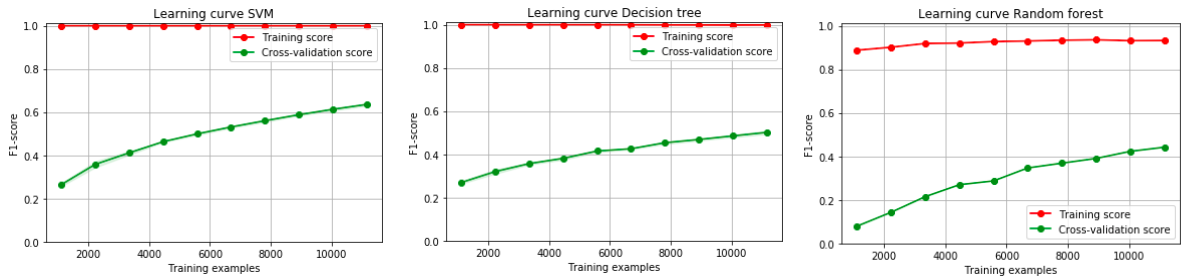


Figure 6.7: Learning curves topic classification for different classifiers.

Ensemble learning The last step performed to possibly improve performance is to test whether multiple classifiers together can improve classifier accuracy. To this end, we tested multiple majority vote ensemble learning setups. The SVM performs best so far and is therefore always used with the highest or same weight as the decision tree (DT) and random forest (RF) classifiers. The results of using different weights for the different classifier is depicted in Table 6.7. We observed no increase in performance as a result of combining the different classifiers. The remainder of the design is based on the SVM classifier.

SVM weight	DT weight	RF weight	Precision	Recall	F1-score
1	1	1	81.73%	45.67%	58.60%
2	1	1	83.62%	50.88%	63.27%
3	1	1	81.46%	51.98%	63.46%
4	1	1	81.04%	51.91%	63.28%
3	2	1	82.09%	51.98%	63.65%
3	1	0	80.26%	52.36%	63.38%

Table 6.7: Results for majority voting classifiers using different weights for each classifier.

6.2.2. Target domain

So far the focus of the implementation has been to implement a classifier that performs well on the source domain. However, the goal of implementing the model is to classify documents in the target domain, which are the ones published on Poliflw. This section discusses the evaluation dataset used to represent the target domain, the evaluation of the initial classifier applied to the target domain, and the efforts of adapting the model to perform better in this domain.

Evaluation dataset

We use an evaluation dataset constructed from the target domain. This dataset contains 150 articles published last year, randomly drawn from the target domain to obtain a distribution of topics that is representative for this domain. These articles are hand-labeled with topics by at least two different people involved in the research project. A consensus on the topics for each article has been reached on each of the articles before any further performance analysis was done.

A drawback to this approach is that the evaluation data is constructed by the authors of this work and may therefore be considered to contain bias in terms of topics chosen that perform well. However, by labeling the data together with multiple people and by constructing the evaluation corpus before analyzing any results we aim to remove such bias.

Evaluation initial classification model

The initial classification model refers to the model specifically tuned towards the source domain as described in Section 6.2.1. We evaluated this model on the target domain using the exact same metrics as we did for the source domain. The precision, recall, and F1-score are summarized in Table 6.8.

Domain adaptation

The goal of applying domain adaptation techniques is to increase the performance of the classification model trained on the source domain and applied to the target domain. For our problem, we considered the three

Domain	Precision	Recall	F1-score
Source domain	79.27%	53.35%	63.78%
Target domain	50.93%	25.11%	33.64%

Table 6.8: Classifier performance on both source and target domain.

domain adaptation techniques commonly used in text classification tasks as discussed in Section 5.2.3. These include approaches based on instance weighting, semi-supervised learning, and change in feature representation.

Instance weighting Instance weighting is not expected to provide us with any gain in performance because, as discussed in Section 5.2.3, it is not considered a suitable approach when dealing with high-dimensional features spaces.

Semi-supervised learning Semi-supervised learning ignores the domain differences and treats labeled data in the source domain as the same data in the target domain. From Section 5.2.3, we know that for such an approach to work the performance of the initial classifier applied to the target domain must already perform well. The precision of our initial classification model applied to the target domain is 50.52%. Although this precision score is not particularly high for a semi-supervised learning approach, we can attempt to incorporate only the N most certain predictions on each iteration as labeled data and retrain the model for as long as performance on the evaluation dataset increases.

The result for different values of N is summarized in Figure 6.8, where the number of samples used as labeled data starts at 13,945 as this is the number of samples in our source domain. Different steps of 500 and 2,000 are used as a difference in step size results in different performance. Therefore, using both a smaller and large step size provides a more complete overview.

The semi-supervised learning approach would be successful if we would observe a new optimal F1-score after incorporating an additional number of samples as labeled data. This is not the case for any step size. Performance first drops and then increases back to the original level after incorporating around 20,000 samples as labeled data. However, the F1-score for this number of samples is still lower than the original F1-score achieved without applying semi-supervised learning.

Figure 6.8: Learning curves for semi-supervised learning in the target domain.

Feature representation Another approach to improve performance in the target domain is to change the feature representation used for the classifier. Traditional ways of doing so include the use of features only present in both domains. However, because we make use of a *tf-idf* vectorization transformer, we can also build the dictionary and transformer on both domains or solely on the target domain. We consider the advantages and disadvantages of both approaches.

***Tf-idf* on both corpora** The advantage of training fitting the vectorization algorithm on both corpora is that we can better estimate *tf-idf* scores based on the occurrence of words in documents in both domains.

This means that we can leverage the knowledge of the source domain to train a good classifier in the source domain while being able to better estimate *tf-idf* values for articles in the target domain as well.

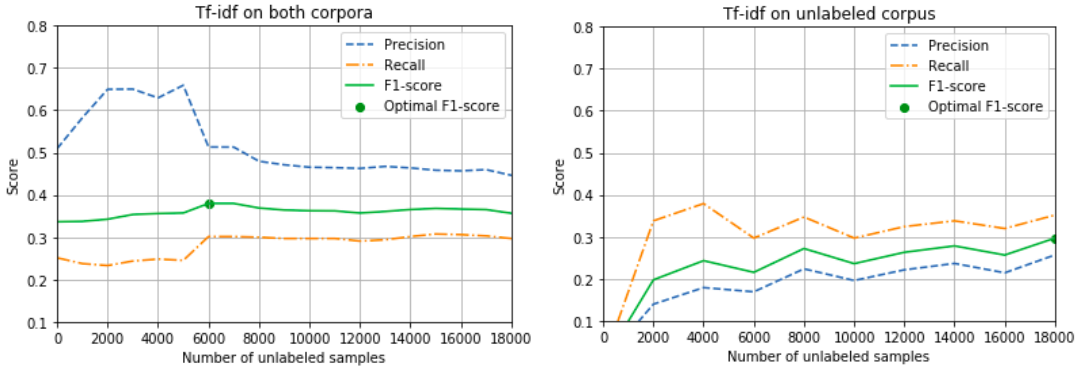
A disadvantage to this approach is that we remain highly dependent on the source domain because as we take the source domain as our basis.

***Tf-idf* solely on target domain** A different approach is to fit the vectorization algorithm solely on the target domain. The advantage of this approach is that we adapt our model more towards the target domain and rely less on the source domain.

A disadvantage to this approach is that our vectorization algorithm is not able to well determine *tf-idf* values in the source domain well and therefore makes us expect a drop in performance in terms of training a classifier on the source domain.

Results The results of fitting the vectorization algorithm on both domains in either setting are depicted in Figure 6.9. From the two graphs, we concluded that fitting the vectorization algorithm on both corpora leads to the best performance and actually increases performance as we incorporate more data from the target domain. Fitting the vectorization algorithm solely on the target domain does not show a high performance for any of the number of articles used to fit the vectorization model.

Although the optimal F1-score is achieved after incorporating 6,000 unlabeled samples from the target domain, we prefer to incorporate additional data as performance remains relatively constant as we fit the vectorization algorithm on additional data and we argue that *tf-idf* values can only become more reliable as additional data is used. We therefore incorporate all 20,000 unlabeled samples randomly drawn from the target domain when fitting the *tf-idf* vectorization model. Performance after using both the source and target domain in construction of the *tf-idf* corpus is summarized in Table 6.9.



(a) Tf-idf on both corpora

(b) Tf-idf solely on target domain

Figure 6.9: Learning curves for tf-idf vectorization fitted on both source and target domain, and only the target domain.

Tf-idf corpus	Precision	Recall	F1-score
Source domain	50.93%	25.11%	33.64%
Source+Target domain	44.52%	31.07%	36.90%

Table 6.9: Summary of performance when constructing the tf-idf corpus based on two different corpora.

Other adaptation We do not want to make changes to classifier parameters because this is likely to result in overfitting on evaluation data, especially as the size of the evaluation data is small. One minor change we can make is to set a different cutoff threshold for the classifier at which it predicts a class label. Based on Figure 6.10 we analyzed the influence of setting the threshold to different values between 0% and 95%. We observed an optimal F1-score of 38.53% for a threshold of 55%. The precision and recall at this threshold are 50.75% and 31.05% respectively.

The changes in F1-score are minimal for thresholds close to 50%. This shows that the classifier is not sensitive to modification of the threshold to other values close to the original threshold, suggesting that we

can choose such a value without having to worry too much about overfitting. The remainder of the design implements a probability threshold of 55%.

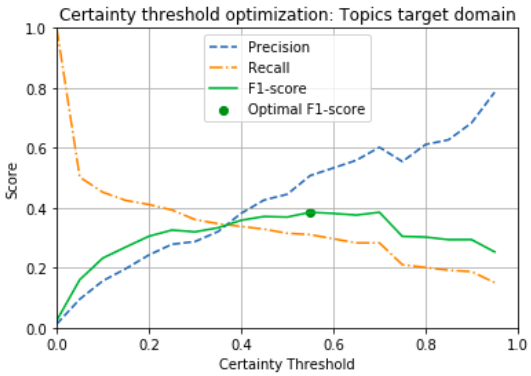


Figure 6.10: Classifier performance based on different classifier thresholds.

6.2.3. Summary

The final design consists of two tasks. First, we designed a topic classification model that performs well on the source domain we argue to be best suitable for our application. We chose to represent articles as feature vectors based on *tf-idf* vectorization. The vectorization algorithm is optimized based on techniques of n-grams, preprocessing, and feature reduction.

Next, we considered different domain adaptation techniques to improve classifier performance in the target domain. After considering different approaches and concluded that techniques based on feature representation provide the highest performance.

6.3. Continuous improvement

This section discusses the learning approach implemented to build a system that continues to improve based on input generated by the Politags human computation system. The human computation system enables the verification and rejection of candidate linkings and topics. It updates the certainty score of the predictions made and can set certainties to `0.0` for denied predictions and `1.0` for verified predictions. This information is used to update and continuously improve our classification models.

6.3.1. Named entities

In terms of extracting, disambiguating, and linking named entities, the additional labeled data is expected to be particularly useful for the disambiguation process. As we collect more labeled data, not only do we expect an increase in performance of our initial KNN disambiguation classifier, but we can also incorporate additional more complex features.

The remainder of this section first discusses how we can compute and incorporate additional features based on the output of the human computation system. Next, it discusses how we can use the output to retrain the disambiguation classifier.

Additional features

Based on candidate linkings verified by the human computation system, we can compute additional features that rely on information gained from these verified linkings. This section first elaborates on the implementation of two of such features implemented and the evaluation, as well as their evaluation in terms of performance gain.

Candidate mention probabilities Previous linkings made between named entities and candidates can be utilized as features to incorporate knowledge on different ways of mentioning the same candidate. There is only a fixed number of ways in which one can write a person's name. We hypothesize that the lack of quality of the names in the knowledge base is one of the major factors holding back the performance of the disambiguation system. Therefore, if we were able to identify the different ways in which candidates are mentioned throughout the text, we could use this information to compare mentions of named entities with

previous linkings of candidates. We propose a similarity measure that is calculated as the number of times a mention phrase m is linked (l, considering all linkings L) to a candidate c with updated certainty (l_u) 1.0, divided by the number of times this mention is linked to any other candidate with updated certainty 1.0:

$$F_{mention}(m,c) = \frac{|\{l \in L \mid l_c = c, l_m = m, l_u = 1.0\}|}{|\{l \in L \mid l_m = m, l_u = 1.0\}|} \tag{6.2}$$

This feature is expected to be as discriminative as the initial name-based features combined. However, we were not able to use this feature in the initial model due to the lack of data to approximate the feature value.

Candidate prior probability Similar to the previous feature, additional labeled data allows us to incorporate information on the probability of candidates mentioned in the articles. We propose an additional feature that does not depend on the way in which candidates are mentioned in text, but rather use their prior probability to determine whether or not a candidate should be linked to. The calculation for this feature is rather straightforward for candidate c and every linking l for which the updated certainty l_u equals 1.0 and the candidate l_c equals c:

$$F_{prior}(c) = \frac{|\{l \in L \mid l_c = c, l_u = 1.0\}|}{|\{l \in L \mid l_u = 1.0\}|} \tag{6.3}$$

The prior probability of candidates is expected to be effective when dealing with highly ambiguous candidates. In that case, we do not have to solely rely on similarity in names and parties to decide which candidate has the highest certainty of linking to. Instead, we can leverage knowledge of how likely it is that this person is actually mentioned in articles.

Evaluation We compute the two similarity features for each of the named entity-candidate pairs in our disambiguation evaluation data used in Section 6.1.2. At the time of writing, only 200 of the named entities in our evaluation dataset contain candidate linkings for which the updated certainty equals 1.0. We retrain the classifier and evaluate the performance. The result is depicted in Figure 6.11.

We see no immediate increase in terms of accuracy. This does not come as a surprise since there is a small number of linkings in the database that have actually been fully processed by the human computation system. Therefore, the two properties cannot yet be computed and provide trustworthy results.

To get a better understanding of the features added, we also analyze the features the same way we did in Section 6.1.2 by training and evaluating the classifier on single features. The result is depicted in Figure 6.12. From these results, we conclude that the two features are actually relatively good at classifying the data compared to other features used.

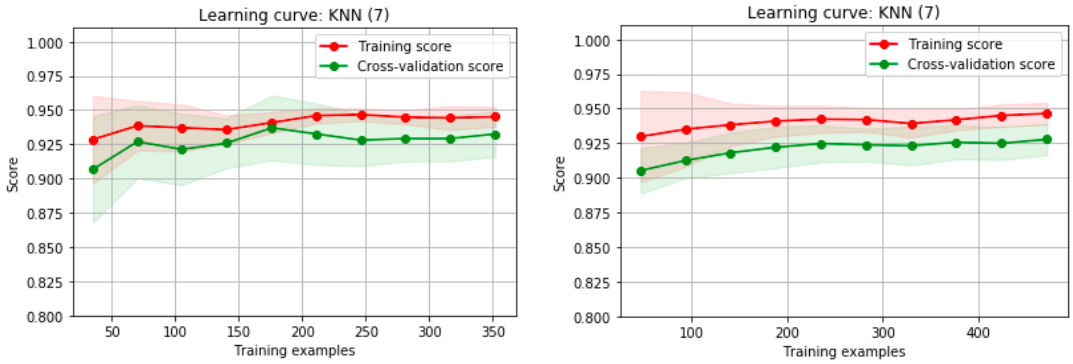


Figure 6.11: Learning curves based on initial features and extended set of features.

Retraining disambiguation classifier

The implementation of the current classifier still has its limitations as it is based on limited amounts of data and the candidates in the dataset labeled as 'incorrect' are not necessarily the candidates with which the current classifier has difficulty classifying. To keep improving the classification model, we retrain the model based on verifications provided by the human computation system.

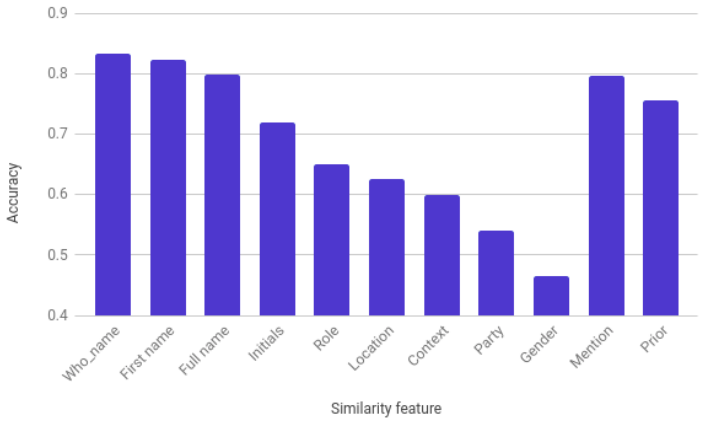


Figure 6.12: Per-feature classifier accuracy for additional features with KNN-7 classifier

The classifier is retrained on all candidate linkings for which the updated certainty is either set to `1.0` or `0.0` by the human computation system. The classifier is trained and evaluated in a cross-validation setting and if performance is higher than 93.63% (highest performance initial classifier) saved and deployed as the new disambiguation classifier.

6.3.2. Topic classification

To implement a learning algorithm to improve our topic classification model, we take a similar approach to the one we implemented with unlabeled data. However, we can now use labels from the human computation system as additional labels.

We apply the following strategy:

1. Retrieve all articles in the target domain for which the updated certainty of a topic is 1.0 to the labeled training data. Incorporate only the topics that have this certainty as labels.
2. Split these articles using cross-validation
3. In each cross-validation iteration:
   (a) Augment the corpus from the source domain used for *tf-idf* vectorization with 20,000 unlabeled articles from the target domain.
   (b) Train the classifier as described in Section 6.2 on the labeled data from the source and target domain.
   (c) Score the classifier on the target domain
4. If the cross-validated F1-score is higher than 38.53%, save and deploy this classifier as the new topic classification model.

This strategy could be evaluated by plotting a learning curve similar to the one plotted on semi-supervised learning in Section 6.2.2. However, as we saw in Section 6.2.2, the number of training samples required to achieve any significant change in performance is in the thousands. At this point, we have only 250 verified topic predictions available from the human computation system. We therefore cannot yet draw any conclusions in terms of how our strategy for continuous topic classifier improvement works.

What we do know is that based on additional training data, especially when considering the source and target domains, a classifier by definition becomes better at estimating predictions. Therefore, including verified predictions in this way will eventually improve performance.

6.3.3. Summary

The design for continuous improvement consists of two separate parts that enable future improvements for the extraction of named entities and classification of topics.

For the extraction of named entities, we incorporated two additional similarity features on politicians that complement the similarity measures already defined in Section 6.1.2. Furthermore, we described how we can retrain the classification model based on feedback provided by the human computation system.

For the classification of topics, we discuss how we can eventually train a model based on the target domain without relying on a source domain by consistently retraining the classifier based on feedback provided.

7

Results

This chapter summarizes the results of the design, discussing the three different design parts of named entities, topic classification, and continuous improvement.

7.1. Named entities

The results for extracting named entity are divided into those on named entity recognition and those on named entity disambiguation.

7.1.1. Named entity recognition

We have shown that we can increase NER performance by extending the gazetteers of an existing NER implementation, tailoring the implementation to this specific domain.

We evaluated the difference in performance between the original NER implementation and the one that includes extended gazetteers. This is done by counting the number of exact matches between extracted named entity mentions and names of entries in our knowledge base. An increase in this number describes the number of entries in our knowledge base that was previously not correctly recognized is now correctly recognized.

In terms of political parties, we were able to successfully recognize an additional 3.37% based on party abbreviations and an additional 2.95% based on full party names as stated in our knowledge base. In terms of politicians, we were able to successfully recognize an additional 15.74% based on full name matches. From these results, we learn that the largest gain in performance is achieved for recognizing politicians.

Correctly recognizing named entities in text is the first vital step in the extraction of named entities. Not recognizing a named entity in the text results in the fact that this political party or politician is not processed in the task of named entity disambiguation and can therefore not be extracted as enrichment. Not properly recognizing the boundaries of a named entity results in a more difficult disambiguation process based on incomplete information. The improvement found by extending the gazetteers, therefore, has a direct impact on final named entity extraction performance.

7.1.2. Named entity disambiguation

The next step is to disambiguate different named entities recognized in the text. This section summarizes the results of the named entity disambiguation implementation.

Candidate generation

To increase runtime efficiency, we combine the NER implementation with a candidate generation process before disambiguating politicians. By applying the steps of NER and candidate generation consecutively, we were able to identify 95.08% of the politicians mentioned in texts. Problems with identifying the remaining candidates originate from the NER module not correctly recognizing all mentions of people as named entities. As a result of this, these mentions are not incorporated into the candidate generation process and no correct candidates can be generated.

Candidate disambiguation

The task of candidate disambiguation is to compute a reliable certainty estimate between pairs of named entities and candidates. For political parties, certainties are estimated based on the similarity between the named entity mention and name of the party in the knowledge base. As discussed in the next section on candidate linking, this is an effective measure for computing the most probable linking.

Politician candidate certainties are estimated using a combination of similarity features that describe name, role, party, location, and gender, and a KNN classifier. We have shown that the two classes of 'correct' and 'incorrect' linkings between named entities and candidate politicians are separable based on these features. As a result, the KNN classifier achieves an accuracy score of 93.64% when predicting the class labels. Analyzing the performance of the different similarity features, we have shown that all features used positively contribute to classifier accuracy.

We analyzed classifier performance when dealing with named entities for which a higher number of candidates is generated in the candidate generation process. We observe a small decrease in cross-validated performance as the number of generated candidates increases. However, average performance remains over 90% after testing on the most ambiguous named entities.

Candidate linking

The final step of the named entity disambiguation system is to link named entities to candidates and to return this information as an enrichment. We have shown that by returning the most probable candidate and by setting a cutoff threshold to determine when a named entity is actually linked to a candidate, we can achieve high precision linking for both political parties and politicians. For political parties we set the linking certainty threshold to 70%, resulting in a precision and recall score of 96.25% and 86.25% respectively. For politicians we set the certainty threshold to 57% (4 out of 7 nearest neighbors of the KNN classifier), resulting in a precision and recall score of 95.45% and 84.38% respectively.

This means that we are not yet able to make perfect predictions as to what candidates are mentioned in a document. However, through the continuous improvement system we are confident that the performance for more difficult candidate linkings will become better over time.

7.2. Topic classification

We have made use of the most representative source domain available to us to classify articles in the target domain. Based on this domain, we tuned the *tf-idf* vectorization algorithm to perform better at the task of topic classification. From our analysis, we conclude that incorporating *n-grams* with $n > 1$ do not improve performance, terms that occur in over 70% of the documents must be removed from the vectorization dictionary, digits must be removed from the vectorization dictionary, and all words must be capitalized as a preprocessing step. We have also shown that reducing the number of features using feature extraction or selection techniques does not improve classifier performance.

In terms of classifiers, we have found that an SVM classifier performs best on this classification task, achieving a precision of 79.27% and a recall of 53.35%. We have also shown that ensemble learning through majority vote among random forest and decision tree classifiers does not improve performance.

By applying this same classifier to the target domain of Politags articles, we achieved a precision of 50.93% and recall of 25.11%. This translates into an F1-score of 33.64%. To improve performance in the target domain, we have considered three different strategies of domain adaptation. These include instance weighing, semi-supervised learning, and feature representation based techniques. From these techniques, an adaptation of feature representation resulted in the highest increase in performance.

By applying domain adaptation techniques we were able to achieve a precision of 50.75% and a recall of 31.05%. This translates into an F1-score of 38.53%. By increasing the F1-score from 33.64% to 38.53%, we realized a gain in performance of 14.54%.

Taking into consideration the fact that we aim to classify a document based on 111 different topics, we can conclude that the performance is definitely usable. 50.75% of the topics assigned to documents are considered 'correct'. However, what is 'correct' is debatable and different topics that are not in the evaluation dataset but classified nonetheless can be considered 'correct'.

7.3. Continuous improvement

One challenge we encountered in designing the system is the lack of correct training data. To compensate for this, we designed the system in such a way that it is able to continuously improve its models based on

input provided by the Politags human computation system. This section discusses the results in terms of continuous improvement for extracting of named entities and classification of topics.

7.3.1. Named entities

For named entities, verifications from the human computation system are used to incorporate more complex features. These include estimates of the *mention probability* and *prior probability*. Incorporating the features does not yet improve candidate disambiguation accuracy of the original classifier. However, analyzing the features did result in the insight that the two features are better at distinguishing 'correct' from 'incorrect' linkings than 6 out of 9 of the initial features.

Furthermore, retraining a classifier on additional labeled data by definition improves classifier accuracy. By retraining the candidate disambiguation classifier on additional data we can continuously improve the quality of extracted politicians.

7.3.2. Topic classification

The topic classification model is retrained on additional labeled data from the target domain. Because the number of articles labeled with topics is still relatively small, we cannot yet draw conclusions as to how much additional data will impact classifier performance. However, as with the candidate disambiguation classifier, additional labeled data by definition improves classifier performance. By continuously incorporating more data from the target domain we can continue to improve the topic classification model based on verified data.

8

Conclusion on information extraction

This work targets the problem of political filter bubbles by suggesting a novel approach based on the use of information extraction techniques to enable diverse personalization. The following research question were stated in Section 5.3:

How can we enrich politically oriented text using information extraction techniques to enable algorithmic presentation of alternative perspectives?

1. *How can we apply named entity recognition and disambiguation techniques to identify politicians and political parties in the Netherlands from Dutch political text?*

2. *How can we apply text classification techniques to extract political topics from Dutch political text?*

3. *How can we design the system to enable learning from a given human computation module and thereby support continuous improvement?*

The chapter discusses the different sub-questions in Section 8.1, 8.2, and 8.3. Section 8.4 discusses the main research question related to the task of information extraction in Politags.

8.1. Named entities

In our design, we show that we can greatly improve upon NER performance of existing solutions by incorporating external knowledge from the knowledge base used, thereby overcoming the challenge of dealing with long-tail entities. Furthermore, we have shown that although the knowledge base is not always complete in terms of politician attributes, we can effectively apply single-NED techniques based on hand-crafted features.

Disambiguation techniques described in literature focus on the disambiguation of named entities present in a 'rich' knowledge base such as Wikipedia for which a rich set of information is available in the form of Wikipedia article content. Because there is no such information available on the named entities we aim to disambiguate, we had to come up with our own methods of disambiguation. By engineering our own set of features that abstracts the politician and article into a feature vector, we were able to train a classifier that is able to classify every politician in our knowledge base without the need for separate classifiers per politician. This enabled us to identify highly-ambiguous entities in the text making use of limited labeled training data, decreasing the efforts of hand-labeling a lot of data.

Literature shows that when dealing with highly-ambiguous entities, collective-NED approaches are more effective in terms of performance but require a lot of computing power. Our single-NED approach does not require lots of computing power and has shown to be effective in disambiguating ambiguous entities.

The design enables the effective linking of disambiguated politicians and political parties to news articles. Enriching articles with such structured information allows for a personalization algorithm to take advantage of knowledge on key political figures mentioned in an article.

8.2. Topic classification

We have identified a source domain effective in classifying a broad range of articles originating from different sources. We have shown how one can design the source domain classifier in such a way that it can also be applied to the target domain, without relying too much on the characteristics of the source domain. To this end, we have used *tf-idf* to represent text in both domains.

The main challenge has been to adapt the classification model to fit the target domain. Different techniques have been tested and the approach based on adapting the feature representation has shown to be most effective. Through this, we have shown how one can build a topic classification model for different content-serving platforms without access to labeled data from the target domain. This means that whenever there is a source domain fit to use for text classification and domain adaptation, one can apply the same steps as described in our design to automatically enrich political content on a platform.

8.3. Continuous improvement

Key to enabling continuous improvement is that we were able to make use of a human computation system that allows for the verification and rejection of predictions made by the information extraction system.

In terms of named entities, we rely on hand-crafted similarity features that describe to what extend properties of politicians correspond with the named entity recognized in an article. However, as we deal with thousands of politicians, we know that some of the politicians are mentioned a lot more often than others. Verified linkings allow us to compute additional features like prior probabilities that have shown to perform well in named entity disambiguation. This is where we expect a major gain in performance in terms of continuous improvement.

As for topics, we rely on a classifier trained on a different domain than the one it is applied to. Therefore, by implementing a continuous improvement system we can gradually shift the classifier training data to the actual target domain as the number of verifications of topics grows. By definition, a growing number of labeled training samples improves classifier performance. This means that as the human computation provides more and more input, we can successfully enable learning to support continuous improvement.

Another important aspect of the continuous improvement system is that it allows the public to be in control over the data enrichments generated. The fact that our system can work without professional moderation and thereby democratizes the data extracted enables it to be applied to a broad range of content-serving platforms.

8.4. Enabling algorithmic presentation of alternative perspectives

By enriching political content with topics, we believe that a vital step was made into the understanding of political content by computers. Together with structured data on named entities, we can now already let a computer 'reason' about the topics and political entities involved in the story. As humans, the questions of 'who' and 'what' are some of the most obvious questions to ask when we discuss politics. Through our research, we enable computers to better engage in this discussion.

To conclude, we have shown how one can effectively enrich politically oriented text using information extraction techniques to enable algorithmic presentation of alternative perspectives. While our prototype is specifically tailored towards Dutch politics, it could easily be extended to any language and country. This means that any news platform could be extended in this way, making it possible to use simple and straightforward algorithms that serve more diverse perspectives to readers.

9

Final conclusion

Online political filter bubbles pose an existential threat to democracies worldwide. They negatively impact nescient citizens' ability to form opinions based on complete information. As personalization algorithms make their way into our lives, we should be conscious of their effects on individuals and on society as a whole.

This work targets online filter bubbles and the two main problems they bring to democracy. First, the decreased autonomy that citizens experience because they are insufficiently informed to be the judge of their own interest. Second, the blocking of information channels as a result of information being filtered from producer to consumer. Many solutions have been proposed to battle either of these problems. However, these solutions only target the select groups of people that are aware of their filter bubbles. Furthermore, existing solutions only provide a patch to the problem of one-sided news found on social media platforms and search engines. We believe that instead of providing a select group of aware users with patches to the problem, new approaches to breaking the bubble should be aimed at the core of the problem; personalization algorithms.

We argue that by enriching politically oriented articles with the right set of metadata, we allow for new personalization algorithms to present diverse political perspectives to the reader. We defined this set of data enrichments to consist of the article topic, and the politicians and political parties that are mentioned in the article. A personalization algorithm that has access to this metadata on articles can be modified to serve diverse perspectives on a certain topic by diversifying the politicians and parties mentioned in the articles while keeping the topic as a constant. Through this approach, we enable designers of personalization engines to focus on leveraging this rich information without having to worry about curating the information themselves.

Our work shows that through artificial intelligence and human computation techniques we can create a system (called Politags) that generates the necessary data enrichments on a content-serving platform, without the need for expert or company employee reviews. The lack of need for expert review is vital because a democracy by definition is moderated by its citizens and selective moderation could lead to bias problems. Politags can be re-engineered to function on top of any content-serving platform and includes continuous improvement mechanisms that increase its effectiveness over time. To illustrate how this works, a complete and fully working prototype was built as part of the design, where our ideas were applied to such a content-serving platform: www.poliflw.nl. We have shown that our ideas can be effectively implemented in an operational system. This was affirmed by the main Dutch news provider NOS who used the system to write an article [1] showing how the website increased transparency and information flow.

Our intention was not to solve this problem ourselves but to send a strong message. If two graduate students can build a system like Politags during a master's thesis project, what would happen if companies behind search engines and social media platforms would dedicate even a small portion of their resources to this challenge? As with newspapers, responsibility lies at the company that serves the content. These companies belong to the most valuable and influential in the world, have access to ample resources, and to some of the brightest minds. Their personalization algorithms were never expected or intended to cause such societal problems, but they did. Now it is up to these companies to reverse the damage. We are optimistic that our work shows that instead of harming democracy, personalization could foster it, and make people more informed than ever before.

[1] https://nos.nl/googleamp/artikel/2223324-online-verkiezingscampagnes-wat-staat-er-in-de-facebookberichten.html

9.1. Future work

A lot of work still remains to be done to be able to truly free people from their filter bubbles. In this section, we provide a brief overview of some of the research directions that could be a great follow-up on the fundamentals we built in this research.

Personalization system

An evident next step is to build the actual personalization systems to present people with alternative viewpoints on topics. The aim of this work has been to build a foundation that enables this based on metadata enrichments, making the actual personalization a matter of serving content using these enrichments. However, the actual implementation of such a personalization system still presents many challenges, even when the enrichments are accurate. Some of the challenges we foresee in implementing the personalization systems include defining its performance indicators and using a user's reading history to personalize future content.

Extension of the set of enrichments

Due to time constraints, this research focuses on the two most fundamental enrichments of named entities and topics. However, we believe that additional, more complex enrichments could even further improve the quality of the set of enrichments. Such complex enrichments may include the stance taken on a particular issue. This, in its turn, is expected to improve the personalization algorithm, by giving the computer a better semantic "understanding" of the contents of an article.

Free-form data input

Free-form data input could result in more valuable responses and offer more task freedom. For entities, research could be directed at looking up the right match directly in the knowledge base using a search query. For topics, free entry tagging and its effects on database noise would be interesting. The quality control mechanisms that come with free entry topics pose a great challenge.

Evolving knowledge base

The knowledge base is considered static in our research. However, in reality, it evolves. Our current implementation is built to disambiguate politicians and political parties currently active in Dutch politics. These entities change over time as politicians become inactive and new politicians become more prominent figures. Challenges associated with these changes are updating the knowledge base and adjusting prior probabilities associated with entries in the knowledge base. In terms of named entities, we designed the system in such a way that it does not matter what politician or political party is extracted as the properties of these entities are abstracted into feature vectors. This means that it would be straightforward to add new politicians. However, politicians could also change party or municipality. These type of changes are considered out of scope for this work.

Other applications

Both the information extraction and the human computation engine could be modified for other domains. The information extraction engine is able to identify entities and topics in the text. When trained on a different training set outside of the political domain, it could extract entities and topics that belong to this new domain.

The human computation engine can generally be used to verify entities and topics. Some of its mechanisms are now tailored for the political domain, but these could be adapted to be more applicable to topics and entities of other domains.

A

Database design

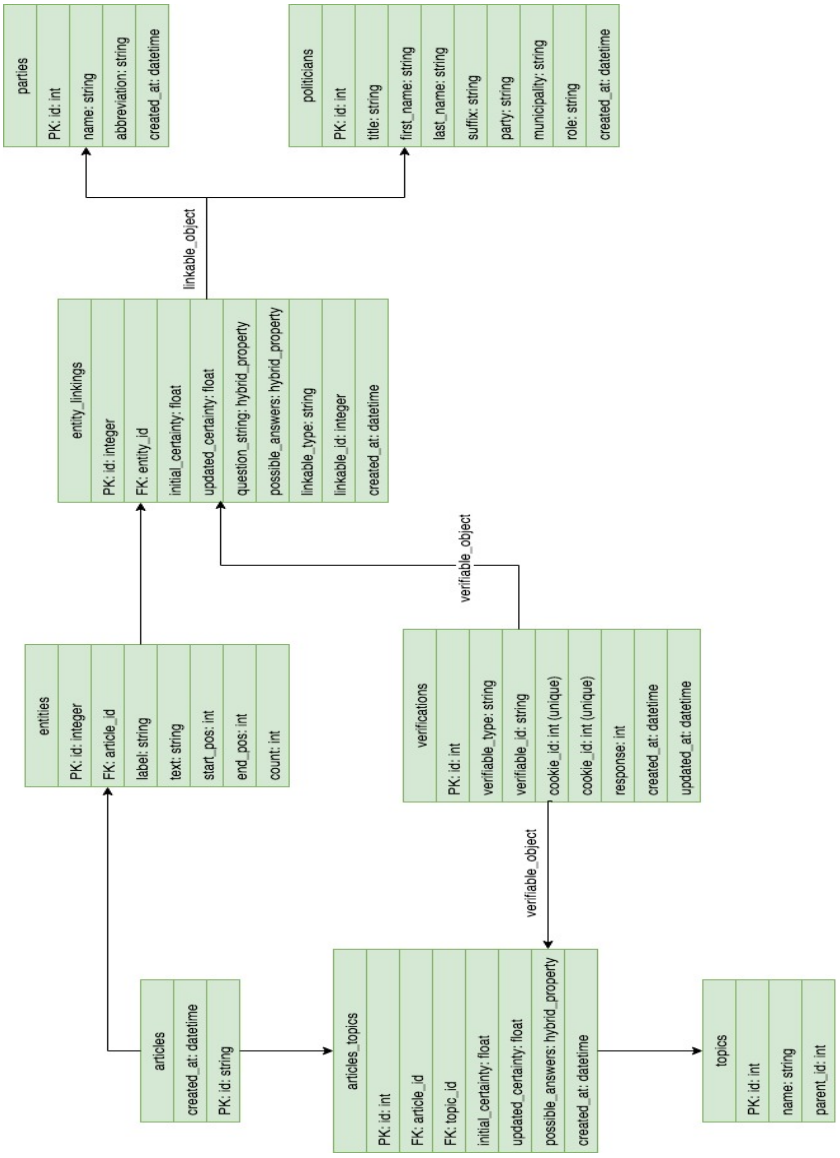


Figure A.1: Politags database design overview

B

API Response format

```
{
  "article_id": "7ebb3e39e73ed87ef1b1038e726290d4bf348526",
  "parties": [
    {
      'id': 123,
      'name': 'Democraten 66',
      'abbreviation': 'D66'
    },
    {
      'id': 124,
      'name': 'GroenLinks',
      'abbreviation': 'GL'
    }
  ],
  "politicians": [
    {
      "full_name": "J.P. Breur",
      "full_name_long": "Dhr. J.P. (Jan) Breur",
      "id": 4474,
      "initials": "J.P.",
      "last_name": "Breur",
      "municipality": "Veenendaal",
      "party": "SP",
      "role": "Fractievoorzitter",
      "system_id": 81576,
    },
    {
      "full_name": "J.R. van Geijtenbeek",
      "full_name_long": "Dhr. J.R. van Geijtenbeek",
      "id": 4436,
      "initials": "J.R.",
      "last_name": "van Geijtenbeek",
      "municipality": "Utrechtse Heuvelrug",
      "party": "SP",
      "role": "Fractievoorzitter",
      "system_id": 73276,
    },
  ],
  "topics": [
    {
      "id": 32,
      "name": "Bestuur | Gemeenten"
    }
  ]
}
```

Listing B.1: Example API response illustrating the format

C

Knowledge base

id	name	abbreviation	created_at
1	Christen-Democratisch Appèl	CDA	2018-03-15 13:33:18
2	ChristenUnie	CU	2018-03-15 13:33:18
3	Democraten 66	D66	2018-03-15 13:33:18
4	GroenLinks	GL	2018-03-15 13:33:18
5	Partij van de Arbeid	PvdA	2018-03-15 13:33:18
6	Partij voor de Dieren	PvdD	2018-03-15 13:33:18
7	Partij voor de Vrijheid	PVV	2018-03-15 13:33:18
8	Staatkundig Gereformeerde Partij	SGP	2018-03-15 13:33:18
9	Socialistische Partij	SP	2018-03-15 13:33:18
10	Volkspartij voor Vrijheid en Democratie	VVD	2018-03-15 13:33:18

Figure C.1: Parties in knowledge base

id	system_id	title	initials	first_name	last_name	party	municipality	role	created_at	gender
1	134414	Dhr.	R.T.A.		Korteland	VVD	Meppel	Burgemeester	2018-03-15 13:32:18	male
2	127993	Mw.	S.		Wolvekamp-Bloemert	CDA	Meppel	Fractievoorzitter	2018-03-15 13:32:18	female
3	127994	Dhr.	L.	Bert	Kunnen	ChristenUnie	Meppel	Fractievoorzitter	2018-03-15 13:32:18	male
4	85638	Dhr.	V.		Veldhorst	D66	Meppel	Fractievoorzitter	2018-03-15 13:32:18	male
5	131310	Mw.	P.		van Eerden-Hein	GroenLinks	Meppel	Fractievoorzitter	2018-03-15 13:32:18	female
6	135441	Dhr.	F.		Hummel	PvdA	Meppel	Fractievoorzitter	2018-03-15 13:32:18	male
7	121001	Dhr.	X.		Topma	SP	Meppel	Fractievoorzitter	2018-03-15 13:32:18	male
8	135442	Mw.	E.		Bakkenes-Van Hese	Sterk Meppel	Meppel	Fractievoorzitter	2018-03-15 13:32:18	female
9	127995	Dhr. drs.	F.J.	Frank	Perquin	VVD	Meppel	Fractievoorzitter	2018-03-15 13:32:18	male
10	129368	Dhr.	R.P.	Roelof Pieter	Koning	VVD	Meppel	Locoburgemeester	2018-03-15 13:32:18	male
11	20274285	Dhr. drs.	G.J.		Fokkema		Meppel	Raadsgriffier	2018-03-15 13:32:18	male

Figure C.2: First 11 politician entries in knowledge base

D

Disambiguation similarity features

```
1
2  from app.models.models import EntityLinking, Entity
3  from app.modules.common.utils import string_similarity
4  from whoswho import who
5  from gender_guesser import detector
6  import logging
7
8  gender_detector = detector.Detector()
9
10
11 def f_mention_prior(mention, candidate):
12     # number of times mention is linked to candidate with certainty 1.0 /
13     # number of times mention linked to any candidate with certainty 1.0.
14     certain_candidate_linkings = EntityLinking.query.filter(EntityLinking.linkable_object == candidate) \
15         .filter(EntityLinking.updated_certainty == 1).all()
16     certain_mention_candidate_linkings = 0
17     for entity_linking in certain_candidate_linkings:
18         if entity_linking.entity.text == mention:
19             certain_mention_candidate_linkings += 1
20
21
22     mention_entities = Entity.query.filter(Entity.text == mention).all()
23     mention_entity_ids = [x.id for x in mention_entities]
24
25     certain_mention_linkings = EntityLinking.query.filter(EntityLinking.entity_id.in_(mention_entity_ids))
       .filter(
26         EntityLinking.updated_certainty == 1).count()
27
28     if certain_mention_linkings  > 0:
29         return certain_mention_candidate_linkings / certain_mention_linkings
30     else:
31         return 0
32
33
34 def f_candidate_prior(candidate):
35     candidate_certain_linkings = EntityLinking.query.filter(EntityLinking.linkable_object == candidate).
       filter(
36         EntityLinking.updated_certainty == 1).count()
37
38     total_certain_linkings = EntityLinking.query.filter(EntityLinking.linkable_type == 'Politician').
       filter(EntityLinking.updated_certainty == 1).count()
39
40     if total_certain_linkings > 0:
41         return candidate_certain_linkings / total_certain_linkings
42     else:
43         return 0
44
45
46 def f_name_similarity(mention, candidate):
47     last_names = candidate.last_name_array
```

```
    score = 0

    for last_name in last_names:
        sim_last = string_similarity(last_name, mention)
        sim_first = string_similarity((candidate.first_name + ' ' + last_name), mention)
        sim_full = string_similarity(candidate.full_name, mention)
        score = max(sim_first, sim_last, sim_full)

    return score


def f_who_name_similarity(mention, candidate):
    sim_first = who.ratio(mention, (candidate.first_name + ' ' + candidate.last_name)) / 100
    sim_initials = who.ratio(mention, candidate.full_name) / 100
    return max(sim_first, sim_initials)


def f_first_name_similarity(mention, candidate):
    mention_names = mention.lower().split(' ')
    candidate_first_names = candidate.first_name.lower().split(' ')

    sim = 0
    for name in mention_names:
        if name in candidate_first_names:
            sim = 1
    return sim


def f_initials_similarity(mention, candidate):
    first_letter_mention = mention[0].lower()
    first_letter_candidate = candidate.initials[0].lower()

    if first_letter_mention == first_letter_candidate:
        return 1
    else:
        return 0


def f_role_in_document(document, candidate):
    role_splitted = candidate.role.lower().split(' ')
    sim = 0

    if len(role_splitted) > 0:
        for role in role_splitted:
            if role in document['text_description'].lower():
                sim = 1
    else:
        sim = 0
    return sim


def f_party_similarity(document, candidate):
    candidate_parties = [x.lower().strip() for x in candidate.party.split('/')]
    sim = 0
    if len(document['parties']) > 0:
        document_parties = [x.lower().strip() for x in document['parties']]
        for candidate_party in candidate_parties:
            if candidate_party in document_parties:
                sim = 1.0
    return sim


def f_location_similarity(document, candidate):
    if document['location'].lower() == candidate.municipality.lower():
        return 1.0
    else:
        return 0.0


def f_gender_similarity(mention, candidate):
    mention_first_name = mention.split(' ')[0]
```

```
    mention_gender = gender_detector.get_gender(mention_first_name)
    candidate_gender = candidate.gender

    if mention_gender == 'unknown' or candidate_gender == 'unknown':
        return 0
    elif mention_gender == 'andy':
        return 0.15
    elif candidate_gender == 'female' and (mention_gender == 'female' or mention_gender == 'mostly_female'
    ):
        return 1.0
    elif candidate_gender == 'male' and (mention_gender == 'male' or mention_gender == 'mostly_male'):
        return 1.0
    else:
        return 0


def f_context_similarity(document, entities, candidate):
    # Fill document entries for comparison
    document_entries = []
    for entity in entities:
        document_entries.append(entity.text)
    for party in document['parties']:
        document_entries.append(party)
    document_entries.append(document['collection'])
    document_entries.append(document['location'])

    candidate_array = [candidate.last_name,
                       candidate.party,
                       candidate.municipality.split(' ')[-1]]

    a = [x.lower() for x in document_entries]
    b = [x.lower() for x in candidate_array]

    sim = jaccard_distance(a, b)
    return sim


def jaccard_distance(list1, list2):
    intersection = len(list(set(list1).intersection(list2)))
    union = (len(list1) + len(list2)) - intersection
    return float(intersection / union)
```

Listing D.1: Python implementation politician disambiguation similarity features

E

Text representation optimization

This appendix discusses the process of improving the performance of the topic classification model by optimizing the construction of the *tf-idf* text representation. It discusses how the use of different term vectorization algorithms, n-grams, preprocessing steps, and feature reduction techniques impact performance.

To evaluate different term vectorization methods, we implemented a basic Support Vector Machine (SVM) classifier also implemented in scikit-learn [1]. The classifier is trained in a *one-vs-rest* fashion where one classifier is trained for each class to distinguish this class from the *rest* of the classes. As training data, we use 80% of the extracted documents in the source domain. 20% of the articles are used for evaluation.

Metrics used to evaluate the classifier are standard text classification measures of precision, recall, and F1-score. These can be either macro- or micro-averaged among class labels. Micro-averaging is preferred over macro-averaging because there is a significant imbalance in class labels. Macro-averaging would compute the metrics independently for each class and take the average, treating all classes as equal, whereas a Micro-averaging aggregates the contributions of all classes to compute the average metric.

***Count* vs. *tf-idf* vectorization** Two algorithms for term frequency vectorization considered are *count* vectorization and *tf-idf* vectorization as discussed in Section 5.2.1. We used the implementations of the Python library scikit-learn[2].

The initial vectorization algorithms both construct a dictionary containing an average of 79,000 words (features) for our dataset. The evaluation results of our initial classifier are depicted in Table E.1. We see a clear benefit from using *tf-idf* vectorization in terms of performance, especially when looking at the difference in precision. Therefore, we choose to further optimize our vectorization method for *tf-idf*.

Figure E.1 depicts the performance of the *tf-idf* vectorization algorithm in terms of F1-score. From this graph, we see that the increase in test performance stagnates after incorporating all training samples. We also observe a training score of roughly 100%. This indicates that the different classes are perfectly separable among the different features. This does not come as a surprise as at this point the dictionary contains a total of 79,000 words.

Vectorization method	Precision	Recall	F1-score
Count vectorization	63.16%	51.26%	56.59%
Tf-idf vectorization	80.23%	52.68%	63.60%

Table E.1: Result of initial evaluation of count and tf-idf vectorization algorithms.

N-grams The standard implementation of *tf-idf* vectorization in scikit-learn represents documents as a *bag-of-words* using *1-grams*. This means that any internal structure of the document and information regarding the way in which words are combined throughout the text is lost. As discussed in Section 5.2.1, we can use *n-grams* with $n > 1$ to retain part of this structure.

[1] http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
[2] http://scikit-learn.org/stable/modules/classes.html#module-sklearn.feature_extraction.text

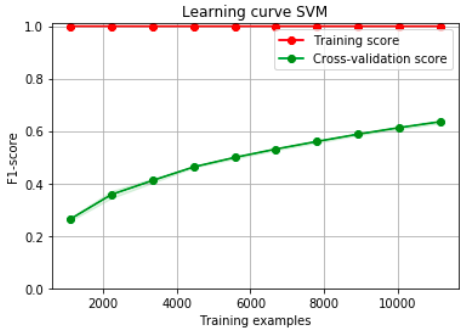


Figure E.1: Learning curve initial SVM classifier.

We evaluate the performance of using a number minimum and maximum of values for n. Minimum values describe the shortest *n-grams* extracted, whereas the maximum values describe the longest *n-grams* extracted.

The result in terms of precision, recall, and F1-score is depicted in Table E.2. We observe an increase in terms of precision when moving towards larger maximum values for the extracted *n-grams*. However, at the same time, we see a drop in recall, resulting in an overall F1-score that is lower when using maximum values of $n > 1$. Also, we observe an increase in the number of features extracted per document. The reason for this is that from a sentence the vectorization algorithm extracts both the single words as features, as well as the different phrases for $n > 1$. This increase in the number of features per document makes the feature vectors difficult to work with as we will suffer more from the *curse of dimensionality*. The *curse of dimensionality* refers to various problematic phenomena that arise when analyzing data in high-dimensional spaces that do not occur in low-dimensional settings.

Based on our findings, we conclude that it is better to stick to the text vectorization model that relies on the traditional *bag-of-words* representation with $n = 1$.

(min_n, max_n)	Number of features	Precision	Recall	F1-score
1,1	79006	80.23%	52.68%	63.60%
1,2	730877	87.28%	48.63%	62.46%
1,3	2158304	89.03%	46.39%	61.00%
1,4	4101337	89.67%	45.29%	60.18%

Table E.2: Results for different ranges of n-grams.

Preprocessing To see which preprocessing steps provide the highest gain in performance, we analyzed the words currently present in our vectorization dictionary. We first extracted a random article from the dataset and analyzed the top 15 features for which the *tf-idf* score is the highest.

The results are visualized in Table E.3. From this table, we observed a number of words in the top 15 that can result in unwanted effects. First, *"De"* (Dutch for "the") is considered a stop word and does not contain any value for when classifying topics from text. Second, "1" is a digit which is not considered a stop word but also contains no value in terms of the topics we aim to classify. Third, the words *"Het"* and *"het"* (also Dutch for "the") are problematic in the sense that the same word in is the dictionary twice but in a capitalized and lowercase form. Using both forms as separate features can have negative effects on our classifier. One goal of preprocessing is to remove these unwanted words from our dictionary.

We first considered removing stop words. Scikit-learn's implementation of *tf-idf* vectorization has a built-in parameter to remove corpus-specific stop words. This parameter, called `max_df`, lets the algorithm ignore terms that have a document frequency strictly higher than the given threshold when building the vocabulary. The parameter (as a decimal in the range of [0.7, 1.0]) represents the portion of documents that contain these terms. We analyze the performance for parameter different settings. The results are depicted in Table E.4. From these results, we see that the highest performance is achieved when setting `max_df` to 0.7.

Next, we considered removing digits from our vectorization dictionary. To this end, we removed all words considered a digit in Python based on the Python function `isdigit()`. The results of applying this prepro-

Rank	Feature	tf-idf
1	studenten	0.274761
2	dupeert	0.263465
3	poort	0.239084
4	allochtonen	0.233354
5	onderwijs	0.231408
6	tweedeling	0.155569
7	mohandis	0.147581
8	hoger	0.146072
9	Het	0.145827
10	westerse	0.133972
11	opleidingen	0.125419
12	de	0.120568
13	leerlingen	0.113446
14	het	0.109686
15	1	0.107531

Table E.3: Top 15 tf-idf scores for a single document.

max_df	Precision	Recall	F1-score
Not set	80.23%	52.68%	63.60%
0.7	80.32%	53.28%	64.06%
0.8	79.62%	53.53%	64.02%
0.9	79.80%	53.05%	63.73%

Table E.4: Classifier performance for different values of max_df

cessing step are summarized in Table E.5.

We observed no direct gain in performance from applying this preprocessing step. However, after analyzing 13,948 articles from the source domain, and 20,000 articles from the target domain, we found that the average number of digits in an article in the source domain is 40.90 whereas the average number of digits in an article in the target domain is 10.11. Therefore, because we do not want the classifier to model features not present in the target domain, we applied the preprocessing step of removing all digits nonetheless.

Preprocessing	Precision	Recall	F1-score
Leave digits	80.32%	53.28%	64.06%
Remove digits	79.28%	53.10%	63.61%

Table E.5: Classifier performance after removing digits.

To prevent the same capitalized and lowercase words to be considered as different features, we can transform all words in the vocabulary to lower or uppercase. The effect (for both lower or uppercase) is that words that are used in both senses are considered the same for the vectorization algorithm. Table E.6 summarizes the change in performance from performing such transformation to uppercase. We observe a gain in performance in terms of recall and F1-score.

We found other preprocessing steps such as stemming to result in no increase in performance and a drastic slow down in terms of run-time. Therefore, we chose to perform no such preprocessing to our final *tf-idf* vectorization algorithm.

Feature reduction By applying the different preprocessing steps of `max_df` and digit removal, we have lowered the number of features from 79,006 to 68,527. We applied feature reduction techniques mentioned in Section 5.2.1 to see whether these can have an impact on classifier performance. The techniques mentioned in Section 5.2.1 include feature extraction and feature selection.

As a popular form of feature extraction, we applied the method of PCA to see whether reducing the high-dimensional space to a lower number of orthogonal dimensions can improve performance. We tried a different number of principal components to project the 68,527 features on, ranging from 1,000 to 50,000 with steps

Preprocessing	Precision	Recall	F1-score
No capitalization	79.28%	53.10%	63.61%
Capitalize all words	79.27%	53.35%	63.78%

Table E.6: Classifier performance after capitalizing all words.

of 1,000. However, we found that no number of projections could actually improve cross-validated classifier performance.

Next, we implemented a variance-based feature selection approach to select a number of features as a form of feature selection. Variance-based feature selection works by removing all features whose variance does not meet a threshold. A variance threshold of $3 * 10^{-6}$ in our case removes half the feature from the dataset, resulting in 30,531 features. We used variance-based feature reduction to reduce the number of feature to between 1,000 and 50,000 features. However, no number of selected features other than the original 68,527 resulted in an increase in cross-validated performance.

Bibliography

[1] Basant Agarwal and Namita Mittal. Text Classification Using Machine Learning Methods-A Survey. pages 701–709. 2014. doi: 10.1007/978-81-322-1602-5{_}75. URL https://link.springer.com/content/pdf/10.1007%2F978-81-322-1602-5_75.pdfhttp://link.springer.com/10.1007/978-81-322-1602-5_75.

[2] Charu C Aggarwal and Cheng Xiang Zhai. A survey of text classification algorithms. In *Mining Text Data*, volume 9781461432, pages 163–222. 2012. ISBN 9781461432234. doi: 10.1007/978-1-4614-3223-4{_}6. URL https://link.springer.com/content/pdf/10.1007%2F978-1-4614-3223-4_6.pdf.

[3] Ayman Alhelbawy and Robert Gaizauskas. Graph Ranking for Collective Named Entity Disambiguation. *Acl*, pages 75–80, 2014. URL http://www.aclweb.org/anthology/P14-2013.

[4] American Press Institute. How Millennials Get News: Inside the Habits of America's First Digital Generation. (March):57, 2015. URL http://www.americanpressinstitute.org/wp-content/uploads/2015/08/Media-Insight-Millennials-Report-March-2015.pdf.

[5] Daniel M Bikei, Scott Miller, Richard Schwartz, and Ralph Weischedel. Nymble: a High-Performance Learning Name-finder. pages 194–201, 1997. URL http://www.aclweb.org/anthology/A/A97/A97-1029.pdf.

[6] Engin Bozdag and Jeroen van den Hoven. Breaking the filter bubble: democracy and design. *Ethics and Information Technology*, 17(4):249–265, 12 2015. ISSN 15728439. doi: 10.1007/s10676-015-9380-y. URL http://link.springer.com/10.1007/s10676-015-9380-y.

[7] Razvan Bunescu and Marius Pasca. Using Encyclopedic Knowledge for Named Entity Disambiguation. *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics*, (April):3–7, 2006. URL http://www.cs.utexas.edu/~ml/papers/encyc-eacl-06.pdf.

[8] Miriam Cha, Youngjune Gwon, and H T Kung. Language Modeling by Clustering with Word Embeddings for Text Readability Assessment. doi: 10.1145/3132847.3133104. URL https://arxiv.org/pdf/1709.01888.pdf.

[9] Angel X Chang, Valentin I Spitkovsky, Christopher D Manning, and Eneko Agirre. A comparison of Named-Entity Disambiguation and Word Sense Disambiguation. *Lrec*, pages 860–867, 2016. URL https://nlp.stanford.edu/pubs/chang2016entity.pdf.

[10] Chia-Hui Chang, Mohammed Kayed, Moheb Ramzy Girgis, and Khaled F Shaalan. A Survey of Web Information Extraction Systems. *IEEE Transactions on Knowledge and Data Engineering*, 18(10):1411–1428, 2006. ISSN 1041-4347. doi: 10.1109/TKDE.2006.152. URL http://doi.ieeecomputersociety.org/10.1109/TKDE.2006.152.

[11] Sheng Chen, Akshay Soni, Aasish Pappu, and Yashar Mehdad. DocTag2Vec: An Embedding Based Multi-label Learning Approach for Document Tagging. pages 111–120, 2017. URL http://www.aclweb.org/anthology/W17-2614http://arxiv.org/abs/1707.04596.

[12] Silviu Cucerzan. Large-Scale Named Entity Disambiguation Based on Wikipedia Data. In *EMNLP-CoNLL 2007*, page 708–716, 2007. ISBN 9788890354175. URL http://www.aclweb.org/anthology/D07-1074http://www.aclweb.org/anthology/D/D07/D07-1074.

[13] Leon Derczynski, Diana Maynard, Giuseppe Rizzo, Marieke Van Erp, Genevieve Gorrell, Raphaël Troncy, Johann Petrak, and Kalina Bontcheva. Analysis of named entity recognition and linking for tweets. *Information Processing and Management*, 51(2):32–49, 2015. ISSN 03064573. doi: 10.1016/j.ipm.2014.10.006. URL https://ac.els-cdn.com/S0306457314001034/1-s2.0-S0306457314001034-main.pdf?_tid=05b27eb4-ae69-11e7-a196-00000aacb35e&acdnat=1507715355_c45a8f3eb023de1791ac39220768249a.

[14] Motahhare Eslami, Aimee Rickman, Kristen Vaccaro, Amirhossein Aleyasen, Andy Vuong, Karrie Karahalios, Kevin Hamilton, and Christian Sandvig. "I always assumed that I wasn't really that close to [her]". *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI '15*, pages 153–162, 2015. doi: 10.1145/2702123.2702556. URL http://dl.acm.org/citation.cfm?doid=2702123.2702556.

[15] Andrea Esuli and Fabrizio Sebastiani. Evaluating Information Extraction. In *Proceedings of the 11th International Conference of the Cross-Language Evaluation Forum*, pages 100–111, 2010. doi: 10.1007/978-3-642-15998-5{_}12. URL http://nmis.isti.cnr.it/sebastiani/Publications/CLEF10.pdf.

[16] Fréderic Godin, Baptist Vandersmissen, Wesley De Neve, and Rik Van de Walle. Multimedia Lab @ ACL W-NUT NER Shared Task: Named Entity Recognition for Twitter Microposts using Distributed Word Representations. In *Workshop on Noisy User-generated Text, ACL 2015*, pages 146–153, 2015. ISBN 9788578110796. doi: 10.1126/science.1247727. URL http://fredericgodin.com/papers/NamedEntityRecognitionforTwitterMicropostsusingDistributedWordRepresentations.pdf.

[17] Ralph Grishman. Information Extraction: Capabilities and Challenges. *International Winter School in Language and Speech Technologies*, page 377, 2012. ISSN 1541-1672. doi: 10.1561/1500000003. URL https://cs.nyu.edu/grishman/tarragona.pdf.

[18] Shrey Gupta, Armin Namavari, and Tyler Otha Smith. Word Sense Disambiguation Using Skip-Gram and LSTM Models. Technical report, 2017. URL http://web.stanford.edu/class/cs224n/reports/2762042.pdfhttps://web.stanford.edu/class/cs224n/reports/2762042.pdf.

[19] Ben Hachey, Will Radford, Joel Nothman, Matthew Honnibal, and James R Curran. Evaluating entity linking with wikipedia. In *Artificial Intelligence*, volume 194, pages 130–150, 2013. ISBN 0004-3702. doi: 10.1016/j.artint.2012.04.005. URL www.elsevier.com/locate/artint.

[20] Xianpei Han and Le Sun. An entity-topic model for entity linking. *EMNLP-CoNLL 2012*, (July):105–115, 2012. URL http://www.aclweb.org/anthology/D12-1010http://dl.acm.org/citation.cfm?id=2390948.2390962.

[21] Zhengyan He, Shujie Liu, Mu Li, Ming Zhou, Houfeng Wang, and Longkai Zhang. Learning Entity Representation for Entity Disambiguation. *ACL Short Papers '13*, pages 30–34, 2013. ISSN 03029743. doi: 10.1007/978-3-642-41491-6. URL http://www.aclweb.org/anthology/P13-2006.

[22] Matthias Hein. Binary Classification Under Sample Selection Bias. In *Dataset Shift in Machine Learning*, volume 135, pages 41–64. The MIT Press, 12 2008. doi: 10.1039/c005526n. URL http://mitpress.universitypressscholarship.com/view/10.7551/mitpress/9780262170055.001.0001/upso-9780262170055-chapter-3http://www.ncbi.nlm.nih.gov/pubmed/20857222.

[23] Johannes Hoffart, Mohamed Amir Yosef, Ilaria Bordino, Hagen F\"{u}rstenau, Manfred Pinkal, Marc Spaniol, Bilyana Taneva, Stefan Thater, and Gerhard Weikum. Robust Disambiguation of Named Entities in Text. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, page 782–792, 2011. ISBN 978-1-937284-11-4. URL http://delivery.acm.org/10.1145/2150000/2145521/p782-hoffart.pdf?ip=131.180.140.93&id=2145521&acc=OPEN&key=0C390721DC3021FF.512956D6C5F075DE.4D4702B0C3E38B35.6D218144511F3437&CFID=1014449622&CFTOKEN=33224711&__acm__=1512727483_164ce54c567a84600c30647db062.

[24] Jing Jiang. A literature survey on domain adaptation of statistical classifiers. *://Sifaka. Cs. Uiuc. Edu/Jiang4/Domainadaptation/Survey*, (March):1–12, 2008. URL http://sifaka.cs.uiuc.edu/jiang4/domain_adaptation/survey/da_survey.pdf.

[25] Jing Jiang and Chengxiang Zhai. Instance Weighting for Domain Adaptation in NLP. *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, (October):264–271, 2007. ISSN 0736-587X. doi: 10.1145/1273496.1273558. URL http://www.aclweb.org/anthology/P07-1034http://aclanthology.info/papers/instance-weighting-for-domain-adaptation-in-nlp.

[26] Orren Karniol-tambour. Learning Multi-Label Topic Classification of News Articles. pages 1–6. URL http://cs229.stanford.edu/proj2013/ChaseGenainKarniolTambour-LearningMulti-LabelTopicClassificationofNewsArticles.pdf.

[27] Travis Kriplean, Jonathan Morgan, Deen Freelon, Alan Borning, and Lance Bennett. Supporting Reflective Public Thought with ConsiderIt. URL https://homes.cs.washington.edu/~borning/papers/kriplean-cscw2012.pdf.

[28] Quoc V. Le and Tomas Mikolov. Distributed Representations of Sentences and Documents. 2014. ISSN 10495258. doi: 10.1145/2740908.2742760. URL https://cs.stanford.edu/~quocle/paragraph_vector.pdfhttp://arxiv.org/abs/1405.4053.

[29] Nut Limsopatham and Nigel Collier. Bidirectional LSTM for Named Entity Recognition in Twitter Messages. In *2nd Workshop on Noisy User-generated Text*, pages 145–152, 2016. doi: 10.17863/CAM.7201M4-Citavi. URL https://noisy-text.github.io/2016/pdf/WNUT20.pdfhttps://www.repository.cam.ac.uk/bitstream/handle/1810/261962/Limsopatham_and_Collier-2016-WNUT2016-VoR.pdf?sequence=1&isAllowed=y.

[30] Xiaohua Liu, Shaodian Zhang, Furu Wei, and Ming Zhou. Recognizing Named Entities in Tweets. pages 359–367, 2011. URL http://anthology.aclweb.org/P/P11/P11-1037.pdf.

[31] Anna Margolis. A literature review of domain adaptation with unlabeled data. *Tec. Report*, (March 2008):1–42, 2011. URL https://pdfs.semanticscholar.org/0afb/0f4a89845a29bb2c1d955cb1fa4770bc7770.pdf.

[32] Sepideh Mesbah, Alessandro Bozzon, Christoph Lofi, and Geert-Jan Houben. Long-Tail Entity Extraction With Low-Cost Supervision. 2018. URL https://2018.eswc-conferences.org/wp-content/uploads/2018/02/ESWC2018_paper_8.pdf.

[33] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. 2013. URL https://arxiv.org/pdf/1310.4546.pdf.

[34] David Milne and Ian H Witten. Learning to link with wikipedia. In *Proceeding of the 17th ACM conference on Information and knowledge mining - CIKM '08*, page 509, 2008. ISBN 9781595939913. doi: 10.1145/1458082.1458150. URL https://www.cs.waikato.ac.nz/~ihw/papers/08-DNM-IHW-LearningToLinkWithWikipedia.pdfhttp://portal.acm.org/citation.cfm?doid=1458082.1458150.

[35] A Mitchell and Rachel Weisel. Political Polarization and Media Habits. *Pew Research Center*, (October), 2014. doi: 202.419.4372. URL https://scholar.google.com/scholar?q=political+polarization+and+media+habits&btnG=&hl=en&as_sdt=0,45#1.

[36] SA Munson, SY Lee, and Paul Resnick. Encouraging Reading of Diverse Political Viewpoints with a Browser Widget. *Icwsm*, (Festinger 1957):419–428, 2013. URL http://www.aaai.org/ocs/index.php/ICWSM/ICWSM13/paper/viewFile/6119/6381.

[37] David Nadeau. *Semi-supervised named entity recognition: Learning to recognize 100 entity types with little supervision*. PhD thesis, 2007. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.109.4327&rep=rep1&type=pdfhttp://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.109.4327%7B&%7Damp;rep=rep1%7B&%7Damp;type=pdf%5Cnhttp://cogprints.org/5859.

[38] Dat Ba Nguyen, Johannes Hoffart, Martin Theobald, and Gerhard Weikum. AIDA-light: High-Throughput Named-Entity Disambiguation. 2014. URL http://ceur-ws.org/Vol-1184/ldow2014_paper_03.pdf.

[39] Eli Pariser. *The Filter Bubble: What the Internet Is Hiding from You*. Viking, 2011. ISBN 1594203008. doi: 10.1353/pla.2011.0036. URL http://www.amazon.com/dp/1594203008.

[40] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. ISBN 9781937284961. doi: 10.3115/v1/D14-1162. URL https://nlp.stanford.edu/pubs/glove.pdfhttp://aclweb.org/anthology/D14-1162.

[41] Daniel Ramage, Susan Dumais, and Dan Liebling. Characterizing Microblogs with Topic Models. *Icwsm*, pages 1–8, 2010. doi: 10.1.1.309.2194.

[42] Lev Ratinov and Dan Roth. *Design challenges and misconceptions in named entity recognition*. 2009. ISBN 978-1-932432-29-9. doi: 10.3115/1596374.1596399. URL http://delivery.acm.org/10.1145/1600000/1596399/p147-ratinov.pdf?ip=145.94.154.13&id=1596399&acc=OPEN&key=0C390721DC3021FF.512956D6C5F075DE.4D4702B0C3E38B35.6D218144511F3437&CFID=810714595&CFTOKEN=61126706&__acm__=1505743477_ecdc62bf767687359f3cb666http:.

[43] Alan Ritter, Sam Clark, Mausam, and Oren Etzioni. Named entity recognition in tweets: an experimental study. In *Proceedings of {EMNLP}*, pages 1524–1534, 2011. ISBN 978-1-937284-11-4. URL http://http://portal.acm.org/citation.cfm?id=2145595.

[44] Erik F. Tjong Kim Sang and Fien De Meulder. Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. 2003. doi: 10.3115/1119176.1119195. URL http://www.aclweb.org/anthology/W03-0419http://arxiv.org/abs/cs/0306050.

[45] Cicero Nogueira dos Santos and Victor Guimarães. Boosting Named Entity Recognition with Neural Character Embeddings. 2015. ISSN 10797114. doi: 10.18653/v1/W15-3904. URL https://arxiv.org/pdf/1505.05008.pdfhttp://arxiv.org/abs/1505.05008.

[46] Sandeepkumar Satpal and Sunita Sarawagi. Domain adaptation of conditional probability models via feature subsetting. *Lecture Notes in Computer Science*, 4702:224–235, 2007. ISSN 03029743. doi: http://dx.doi.org/10.1007/978-3-540-74976-9{\textbackslash}{_}23. URL https://pdfs.semanticscholar.org/1e15/42a0f9ce1e3807aad47eb387f24ab47def9c.pdfhttp://www.scopus.com/scopus/inward/record.url?eid=2-s2.0-38049120269&partnerID=40.

[47] Richard Socher, C C Lin, Andrew Y Ng, and Christopher D Manning. Parsing natural scenes and natural language with recursive neural networks. *Proceedings of the 26th International Conference on Machine Learning (ICML)*, 2:7, 2011. ISSN <null>. doi: 10.1007/978-3-540-87479-9. URL https://nlp.stanford.edu/pubs/SocherLinNgManning_ICML2011.pdfhttp://fukushima.55-works.com/index/nlp.stanford.edu/pubs/SocherLinNgManning_ICML2011.pdf%5Cnpapers2://publication/uuid/3109E241-AC5E-4BD3-B147-2262D8A2A859.

[48] Yangqiu Song, Shimei Pan, Shixia Liu, Michelle X. Zhou, and Weihong Qian. Topic and keyword re-ranking for LDA-based topic modeling. In *Proceeding of the 18th ACM conference on Information and knowledge management - CIKM '09*, page 1757, New York, New York, USA, 2009. ACM Press. ISBN 9781605585123. doi: 10.1145/1645953.1646223. URL http://portal.acm.org/citation.cfm?doid=1645953.1646223.

[49] Amos J Storkey and Masashi Sugiyama. Mixture Regression for Covariate Shift. *Advances in Neural Information Processing Systems 19*, pages 1337–1344, 2007. ISSN 10495258. URL http://homepages.inf.ed.ac.uk/amos/publications/StorkeySugiyama2007MixtureRegressionForCovariateShift.pdf.

[50] Benjamin Strauss, Bethany Toma, Alan Ritter, Marie-Catherine de Marneffe, and Wei Xu. Results of the WNUT16 Named Entity Recognition Shared Task. *Proceedings of the 2nd Workshop on Noisy User-generated Text (WNUT)*, pages 138–144, 2016. URL http://aclweb.org/anthology/W16-3919.

[51] Van Cuong Tran, Dinh Tuyen Hoang, Ngoc Thanh Nguyen, and Dosam Hwang. A named entity recognition approach for tweet streams using active learning. *Journal of Intelligent & Fuzzy Systems*, 32:1277–1287, 2017. doi: 10.3233/JIFS-169126. URL http://content.iospress.com/download/journal-of-intelligent-and-fuzzy-systems/ifs169126?id=journal-of-intelligent-and-fuzzy-systems%2Fifs169126.

[52] Joseph Turian, Lev Arie Ratinov, and Yoshua Bengio. Word Representations: A Simple and General Method for Semi-Supervised Learning. pages 384–394, 2010. URL http://metaoptimize.http://www.aclweb.org/anthology-new/P/P10/P10-1040.bib%5Cnhttp://www.aclweb.org/anthology-new/P/P10/P10-1040.pdf.

[53] S Vijayarani, Ms J Ilamathi, Ms Nithya, Assistant Professor, and M Phil Research Scholar. Preprocessing Techniques for Text Mining -An Overview. URL http://www.ijcscn.com/Documents/Volumes/vol5issue1/ijcscn2015050102.pdf.

[54] Jürgen Von Hagen. *Money growth targeting by the bundesbank**, volume 43. Princeton University Press, 1999. ISBN 9780874216561. doi: 10.1016/S0304-3932(99)00009-4. URL http://www.scopus.com/inward/record.url?eid=2-s2.0-84923995958&partnerID=40&md5=3915029dc0ef5a3fa62ab7b68d694dfe.

[55] Yonghui Wu, Min Jiang, Jianbo Lei, and Hua Xu. Named Entity Recognition in Chinese Clinical Text Using Deep Neural Network. In *Studies in Health Technology and Informatics*, volume 216, pages 624–628, 2015. ISBN 9781614995630. doi: 10.3233/978-1-61499-564-7-624.

[56] Frederik J Zuiderveen Borgesius, Damian Trilling, Judith Möller, Sarah Eskens, Balázs Bodó, Claes H. de Vreese, and Natali Helberger. Algoritmische verzuiling en filter bubbles: een bedreiging voor de democratie? *Computerrecht*, 173(5):255–262, 2016.