Summarization of Statements in Argumentation Graphs

Master's Thesis

by

Jan Julius Jessewitsch

born in

Haan

submitted to

Professorship for Computer Networks
Prof. Dr. Martin Mauve
Heinrich-Heine-University Düsseldorf

March 2020

Supervisor:
Markus Brenneis, M. Sc.

Abstract

Text contributions written by users who participate in an online discussion are systematically connected to form an argumentation graph. In this thesis, a procedure has been developed to summarize the statements inside these graphs to keep track of the generated textual data. With the aid of algorithms from natural language processing, keywords and keyphrases are extracted from and assigned to the statements. The performance of four different statement summarizers is measured against handmade keyword and keyphrase data sets.

The results have shown, the implemented algorithms for statement summarization identify most of the predefined words and phrases, but there is still much space for improvements concerning the manual annotation rules or the automatic summarization procedure.

Acknowledgments

In the first place, I want to thank my supervisor, Markus Brenneis, for supporting me during the time of writing this thesis. The feedback and criticism he provided on the many different steps of progression has always been reasonable and constructive.

Secondly, many thanks to Prof. Dr. Martin Mauve for giving me the opportunity to write this thesis. The really well organized courses which are offered by the research group of computer networks have been exceptionally inspirational for me.

Thirdly, many thanks to Prof. Dr. Stefan Conrad for sharing his knowledge and tremendously extending my experience about the application of databases and about the existing solutions of knowledge discovery.

Finally, I want to thank Prof. Dr. Egon Wanke for allowing me to start the masters degree course in computer science at the HHU. I appreciate the helpful advice he gave me during the time of transition.

Contents

List of Figures

List of Tables

Chapter 1.

Introduction

Online communities accomplish the purpose of entertainment, communication of knowledge or collaboration. One of the first online discussion systems was Usenet [LF03] which was invented in 1979, a decade before the WWW was born. In Usenet, users subscribe to newsgroups where articles news or postings are published, to which questions, answers or comments can be written. These early approaches have been largely replaced by blogs, forums, wikis and online news media. On the one hand, the tremendous amount of data that is daily accumulating as a result of user generated content serves as a valuable and almost inexhaustible resource for optimization, classification, analysis or business processes. On the other hand, do established platforms for online discussions have significant drawbacks [Spa+14] like redundancy of contributions, lack of argumentative structure or limited scalability. A relatively new area of research that is not only dealing with, but also overcoming the known problems of online discussion is online argumentation.

With an increasing number of contributions and participants grows the complexity of a discussion and the more difficult it gets to follow the different chains of reasoning. Methods of summarization are necessary to simplify the analysis of text contributions and to improve the applicability of online argumentation systems.

1.1. Dialog-based Online Argumentation

The main idea of structured dialog-based online argumentation was originally proposed by Krauthoff et al. [Kra+16] who explained the fundamental concepts for a dialog-based online argumentation system (D-BAS)[1] . It was designed to be usable by any participant without the necessity of previous training. With D-BAS [Kra+18], users enter into a time-shifted dialog with each other. The participant is guided through the discussion by the system.

First, the user selects a topic or position he is interested in. Then he is asked to comment on it, argue for it or argue against it. In this response step, he is allowed to reuse arguments that are already available in the system. Afterwards, based on his previous participation, a counter argument of another user is presented to him and he can respond to it. The System assists the participant in providing new text input within the dialog by offering sentence openers and previous contributions of similar content. To tackle the problem of incorrect or inappropriate user input, D-BAS applies a decentralised moderation system. A valid text contribution is also called a *statement*. A *position* is a prescriptive statement. It contains a judgement of value or an instruction. Starting from the topic or the *issue* of a discussion, the argumentation of participants establishes a web of reasons (WoR). An *argument* is constituted by its *premise(s)*, its *conclusion* and a *reason-relation* between the premise(s) and the conclusion. Premises are formed by statements, and a conclusion is a statement or a reason-relation of another argument. The result of a structured online discussion is called an "argumentation map" or an "*argumentation graph*".

Possible areas of application for dialog-based online argumentation are e-participation [Mac04], to improve the processes of decision making in e-democracy, or embedding into websites [MKM17], to connect their contents or the discussions in comment sections.

1.2. Problem Description

The text contributions of users participating in an online discussion are usually not written in formally standardized but in *natural language*. From a user perspective it may be acceptable that they are written and connected in this raw state. However, even if obvious communi-

[1] `https://dbas.cs.uni-duesseldorf.de`

cation problems caused by spam, bad choice of words, spelling errors or language barriers would not occur, the results of a discussion can be unclear or inappropriate for further processing or evaluation, because the discussion itself, the arguments and statements being used are complex entities.

The results of online argumentation should not become less useful the more users participate and the more text contributions are submitted, especially it is applied in an online decision making process.

1.3. Motivation and Goals

For improved applicability of online argumentation and automatic or manual utilization of its results, further processing steps of simplification and summarization of text contributions are necessary. These additional steps facilitate navigation within argumentations or support the moderation of discussions. The highlighted display of important segments, a more precise search or the classification of statements, arguments or subgraphs are possible application examples.

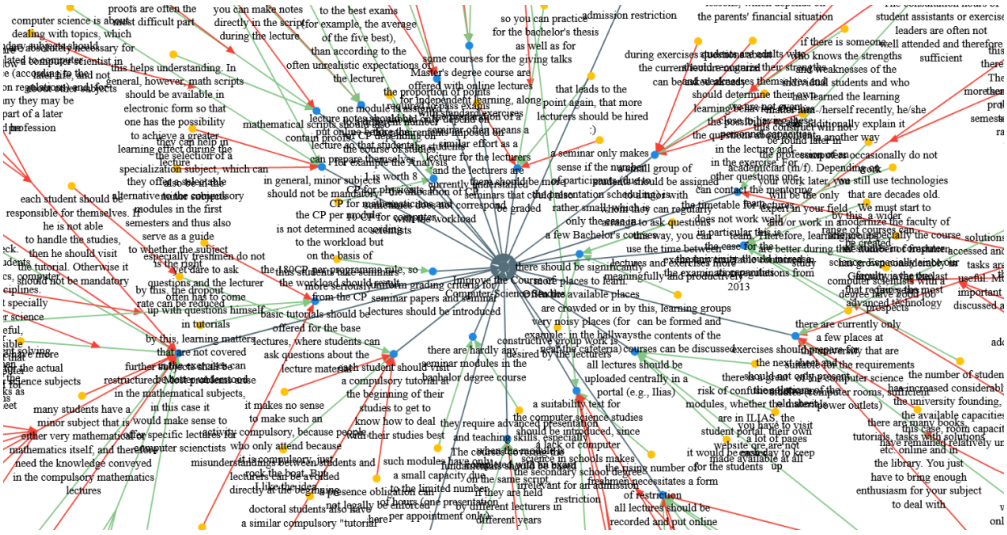


Figure 1.1.: The graph view of a discussion on the D-BAS website[2]

Currently, the graphical representation of an argumentation graph and its nodes' texts on the D-BAS website looks like Figure 1.1. The representation is unclear, there is no highlighting or search function for statements.

The main goal of this work is to develop a processing unit that automatically reduces statements of argumentation graphs to the essential. Capturing the content of statements is primarily achieved by applying different *natural language processing* (NLP) techniques and extracting keywords from them. Achieving this goal will set the stage for the necessary improvements.

[2]The picture was taken from `https://dbas.cs.uni-duesseldorf.de/discuss/improve-the-course-of-computer-science-studies#graph`

Chapter 2.

Basic Terms and Related Work

The development of D-BAS and the goals of this thesis were explained in the first chapter. On that point, it is important to emphasize that this work aims to improve the results of structured online discussion. All concepts from *natural language processing* (*NLP*) that are presented here are being sufficiently studied and already used in many different areas of research such as speech recognition or argument mining, but have not yet been applied to dialog-based online argumentation. In order to comprehend the explanations in the this chapter and the following chapters, it is necessary to define some of the basic terms from natural language processing.

2.1. A Selection of important Terms from NLP

If appropriate tools or modules solving different tasks from NLP are concatenated, the emerging execution procedure is called a *NPL pipeline*. In fact, the whole process of text summarization can be described as a NLP pipeline. An application for text summarization is the summarization of statements in argumentation graphs. This application is the venture of this thesis. The different resources and algorithms, which have been utilized to solve this task, are discussed in the subsequent chapter (see Sections 3.2.3 and Section 3.3 in particular).

2.1.1. Cleaning a Text from Noise

Text documents, before being processed by a pipeline, often contain unnecessary or problematic characters or character sequences, which can induce the pipeline to produce incorrect results or to show unwanted side effects. If a text is written in the natural language L and V_L is the vocabulary of L, character sequences or words which are not included in V_L are not translatable into corresponding words of V_L or can not be transformed into words of V_L (*normalization*) i.e. by spelling error correction methods, can be defined as *noise*. Whereas, some authors of scientific papers equate noise with stop words (see 2.2). There is no general definition of noise, since this preprocessing step depends on the domain of the texts and on the topic the author is elaborating. Text data from several online resources like forums, blogs, wikis or comment sections of websites commonly contains noise, especially if it is obtained by a crawler without appropriate parsing. In this case, HTML tags, additional whitespaces or even exploits can be contained in the text data. To avoid incorrect results or unintended behavior of the program, noise should be removed or suitably replaced before initiating further processing steps. The occurrence of noise which has been identified inside the statements of the studied argumentation graphs is addressed in Section 3.1.5.

2.1.2. Tokenization

An algorithm which is splitting a text into continuous character sequences based on a predefined delimiter or a set of rules is a *tokenizer*. A tokenizer can be applied to subdivide a text into sentences or into words. The output of a word tokenizer are called *tokens*. If a word tokenizer only splits on whitespaces, it also produces tokens containing non-alphabetic characters like the apostrophe or the hyphen. To improve the performance of a tokenizer, additional rules or exceptions can be defined and taught to it. An example of a common tokenizer exception for the English language is to replace the word “don’t” with the tokens “do” and “not”. Texts from different domains often include uncommon abbreviations. This implies an additional challenge which needs to be addressed before tokenization. Abbreviations like “D-BAS” or “M.Sc.” have to be previously identified as tokenizer exceptions to avoid incorrect tokens or sentence boundaries.

2.1.3. Part of Speech Tags

The grammatical type of a word which can be concluded from its syntactic role is its word class or its *part of speech* (*POS*). English parts of speech being commonly used are noun, verb, adjective, adverb, adposition, pronoun, conjunction, numeral, interjection and determiner. A label which is assigned to a word by a person or by a program to indicate the word's part of speech is called a *POS tag*. Successfully POS tagging text data can not be achieved by simple lookup techniques, because the syntactical context of words needs to be considered. The English word "mean" can either be a noun, a verb or an adjective depending on syntactical relations.

For the purpose of more accurate POS tagging and for extracting additional information from syntactical properties of text data, scientific institutions often apply a more fine-grained *POS tagset*. An enhanced set of POS tags can allow to distinguish between grammatical forms of words like singular and plural for nouns, or the different tenses for verbs. For the English language, the Department for Computer Linguistics from the University of Pennsylvania is offering the Penn Treebank tagset [1], which includes 36 different labels for POS tagging. For German, the Institute for Natural Language Processing from the University of Stuttgart provides the Stuttgart-Tübingen TagSet (STTS) [2], which contains 54 different labels for POS tagging.

2.1.4. Stemming and Lemmatization

The sequence of alphabetic characters which can be derived by stripping off distinctive suffixes from specific grammatical forms of a word (*inflections*) is the *stem*. An algorithm which determines the stems of words from a natural language based on a predefined set of rules is called a *stemmer*. The stem itself is not necessarily a word from the natural language's vocabulary. The English words "compute", "computer", "computing" and "computability" share the common stem "comput". A well-known algorithm for suffix stripping is the Porter Stemmer described in [Por80].

[1] `https://www.ling.upenn.edu/courses/Fall_2018/ling001/penn_treebank_pos.html`

[2] `https://www.ims.uni-stuttgart.de/forschung/ressourcen/lexika/germantagsets/`

If the determined root of inflected words from a natural language is part of its vocabulary and equals the dictionary form, it is called a *lemma*. A *lemmatizer* determines the lemma for inflected forms of a word. The processing step of lemmatization does not only require the word but its POS tag to correctly infer its lemma, because homonymous words exist in many natural languages [3]. For instance, the word "bearing" could be the inflected form of the verb "to bear" or it could be the adjective or the noun "bearing". A huge lexical database for English, which is popular in scientific research, can be used for lemmatization and has been developed at the Princeton University, is the WordNet [4].

2.1.5. Grouping of Words

In order to derive additional information from processed text data, words are not only individually regarded but captured in context or in semantical groups. The term *n-gram* either designates a continuous sequence of n characters or a continuous sequence of n words. The latter definition is usually applied in the context of text summarization. A useful measure to describe relations between different words is the *co-occurrence*. This measure can be determined by counting how often words occur together in a particular order. Some algorithms (see 2.3.2) extend the definition of this measure to include words having a maximum distance d from each other (for $d > 1$). A special case of co-occurring words are *collocations*. They occur so frequently in a specific domain or generally in texts of a specific language that they form a semantic entity, and none of the words in the collocation can be replaced by a synonym without them loosing that quality. An example for an English collocation is "applied science".

A sequence of words which contains a noun, specifies that noun and serves either as an object or as the subject inside the grammatical structure of a sentence is a *noun phrase* (*NP*). An algorithm performing the segmentation of text data into noun phrases is an NP chunker and its results are *NP chunks*. An example of a noun phrase is the segment "the segmentation of text data" in the previous sentence. NPs can be nested inside each other forming more complex noun phrases.

Some of the words, segments or phrases inside of text data refer to real-world objects, in

[3] Words with different meanings having the same sequence of characters and the same pronunciation.
[4] `https://wordnet.princeton.edu/`

the context of NLP also known as *named entities* (*NE*s). For instance, the acronym "HHU" represents a named entity. The procedure or the challenge of assigning meaningful tags like "organization", "person" or "location" to named entities is called *named entity recognition* (*NER*).

2.2. Criteria for Stop Words and Keywords

Stop words are words that belong to a natural language and are most frequently used, independently from context or domain of a conversation or of a text. For each natural language exist several stop word lists in open source libraries, academic projects or scientific papers. Luhn, who was a German computer scientist and a pioneer in the research area NLP, referred to stop words as "common words" which glue the words together that represent the content of a text document. Good examples for stop words of the English language are the words 'for', 'of', 'the', 'are', 'is' or 'to'. A suitable but not sufficient indicator is the length of a word in characters. Another reliable indicator is the word's part of speech. Nouns, adjectives or full verbs are rarely considered good stop words, but prepositions, pronouns, conjunctions or auxiliary verbs are commonly. Most important for the creation of a useful stop word list is the context independence. Although it might be convenient to create a list of selected terms for a certain task or area of application. For instance, the word 'humans' is frequently used in news articles and it may be a good stop word in this domain, but it is not applicable for general use.

Keywords are rather bound to a single document or a collection of documents than to a natural language in general. They are usually not stop words but are characterized by a high frequency of occurrence. Whereas the manual assignment of keywords from authors of scientific articles or websites constitutes an exception, if words are chosen which occur marginally or not at all in their document. The relation between the keyword and the document is not only of syntactical but of semantical quality. Keywords serve to identify documents or certain parts of documents since they reflect their contents. The content is a mixture of facts, decisions, intentions and ideas, which can belong to or stem from the most different impressions, experiences, points of view and bases of knowledge. Keywords can also serve as index terms to retrieve documents in an information system.

2.3. History of Keyword Extraction

Luhn was the first who described a relation between a word's significance inside a text document and it's frequency of occurrence [Luh58]. He stated that some words are "common" words and other words are *key words*, which represent the content of a document. Initially, his goal was to use keywords to score sentences and extract the significant ones to automatically create literature abstracts. Later, he points out their importance to serve as index terms in information retrieval [Luh60]. Another game-changing contribution came from Spärck Jones, who proposed considering a terms *collection frequency* for identifying keywords and distinguishing documents [Jon72].

To describe the task of automatic identification of significant terms for documents, the authors of scientific papers use different terminology [Bel14]. "Key words", "key terms", "key phrases" or "key segments" are often used interchangeably to characterize the content of a document. For the purpose of clarification during the progression of this work, "*keyword*" will denote a single representative term and "*keyphrase*" will designate a representative n-gram.

Over the years, many keyword or keyphrase extraction (KE) algorithms have been developed, but only a few of them are commonly used as a reference for comparison and are regarded as state-of-the-art. In order to categorize existing keyword extraction methods without elaborating too many details and considering mixed variants, it is adequate to use the coarse differentiation of supervised and unsupervised approaches.

2.3.1. Supervised Methods

Supervised keyword extraction methods require an annotated set of training data. The manual annotation is usually done by the originator of the utilized data set or by the author of the proposed strategy. It is also possible that the annotation is accomplished through an agreement of several annotators. For supervised methods, keyword extraction is a binary classification task: Either a word in a tokenized sentence is a keyword or it is not a keyword. The input text documents are split and transformed into a multidimensional feature vector space. "Learning" is achieved by deriving probabilities or rules for classification from the input vectors or by minimizing the error rate of assigning keywords by weighting and readjusting them.

The learning process induces a model that mimics the annotation pattern of the training data. A major drawback of supervised approaches is the domain and language dependency. As a consequence, a new model has to be trained each time the language or domain of the input documents changes.

The keyphrase extraction algorithm (KEA) proposed by Witten et al. [Wit+99] applies the *Naive Bayes classifier* to fulfill its purpose. The classification of candidate phrases being identified in preprocessing steps relies on two features: TFxIDF and first occurrence, which is a measure of distance from the document's start to the phrase's first appearance. The trained model predicts the probability of candidates being keyphrases. During postprocessing, keyphrases that are subphrases of higher-ranked keyphrases are dropped.

GenEx, which has been proposed by Turney [Tur99], consists of two components: Whitley's genetic algorithm (Genitor) and a keyphrase extraction algorithm (Extractor). The Extractor is a modified version of Quinlan's C4.5 decision tree algorithm, which takes twelve features as input and is capable of producing keyphrases from text documents. The Genitor is used to optimize the Extractor's performance by adjusting its parameters based on the training data.

Two significant drawbacks in GenEx and KEA are diagnosed by Hulth [Hul03]. The number of tokens for keyphrase candidates is limited to three although manually assigned keyphrases may consist of more than three terms. The number of extracted keyphrases needs to be defined by the user. She suggests that the extraction system itself should determine a threshold for the probability of candidates being classified as keyphrases. Her own approach includes linguistical knowledge for the task of keyword extraction. For candidate creation or term selection are not only n-grams considered, but noun phrase chunks detected and POS tag patterns applied. Used features are TF, IDF, first occurrence and the sequence of POS tags. The installed machine learning algorithm is the Compumine Rule Discovery System, therefore the induced model is constituted by a set of learned rules.

2.3.2. Unsupervised Methods

Unsupervised keyword extraction methods do not require annotated training data. They rely on statistical or structural qualities of the input text documents. Except for preprocessing

steps like stop word elimination, unsupervised methods are characterized by language and domain independence.

A relatively simple approach has been pursued by HaCohen-Kerner [HK03] that consisted in extracting keyphrases from abstracts and titles of scientific papers. He used n-grams as keyphrase candidates and saved them in word weight matrices. The weight was determined by counting full and partial appearances of those words.

In TextRank, which was proposed by Mihalcea and Tarau [MT04], words are mapped to graph vertices. Syntactic filters can be applied to allow only words with certain POS tags. An edge between two vertices is created if the words represented by those vertices co-occur within a window of maximally N words, where $N \in [2, 10]$. The score of a vertex is iteratively calculated based on its in- and out-degree, on the scores of its neighboring vertices and optionally on edge weights. Starting from arbitrary values, the score calculation for each node is repeated until convergence is reached. This procedure is derived from the PageRank algorithm [BP98], but instead of determining the significance of websites, it allows to extract important words or sentences from text documents.

The Rapid Algorithm for Keyword Extraction (RAKE) was developed by Rose et al. [Ros+10]. Candidate keywords are generated by splitting the input document's text into an array at positions of specified stop words and delimiters. From the remaining word groups, a matrix of word co-occurrences is derived. The degree and the frequency of words are taken into consideration for words scoring. The score of candidates that are composed of several words is formed by the sum of these words. RAKE is capable of extracting keyphrases that contain interior stop words by adjoining detected keywords. A new keyphrase is created if two keywords appear together in the document in the same order at least twice.

Another unsupervised graph-based keyphrase extractor is DegExt [Lit+11], which differs from TextRank in simplicity and reduced computational complexity. For each distinct non-stop word, a vertex with a unique label is added to the graph. An undirected edge from vertex A' to vertex B' represents an order-relationship between the corresponding words A and B, i.e. if word B immediately succeeds word A in any sentence of the document, the graph includes an undirected edge from A' to B'. Edges are labeled with sentence IDs for sentences containing both words in correct order. Only words of most connected nodes are selected as keywords. During postprocessing, keyphrases are identified by joining sequences of adjacent keywords.

An entirely different strategy was proposed by Timonen et al. The Informativeness-based Keyword Extraction (IKE) [Tim+12] aims at short documents, which usually have not more than 100 words. This kind of documents imply the "*TF=1 challenge*" [Tim12], that is each word occurs only once in the document. After cleaning the text of the documents from noise, they are clustered by means of Agglomerative (CompleteLink) clustering. The word informativeness evaluation happens on the different levels: Corpus level, cluster level and document level. The score of the last level is calculated by taking the weighted average of the two levels above. The authors conclude that compared methods show poor performance because they are not accessing or including the corpus.

Chapter 3.

A Natural Language Processing Unit for Summarization of Argumentation Graphs

To describe the development of a natural language processing unit for the summarization of statements in argumentation graphs, the next steps are: To discuss the requirements, to illustrate the desired functionalities, to explain the followed design decisions, and finally, to outline the actual implementation of it.

3.1. An Uncommon Type of Dataset: Argumentation Graphs

Argumentation graphs are the results of structured online discussions in D-BAS. They are rooted graphs with directed edges that are pointing towards the root. The basic building blocks of argumentation graphs are issues, arguments and statements.

The central instance or root node is constituted by the issue of the discussion. Valid text contributions from participants form the statements. A position is a statement that is directly connected to the issue. Arguments are represented by nodes that, on the one hand, precede and are adjacent to one other node, a statement or another argument forming the conclusion

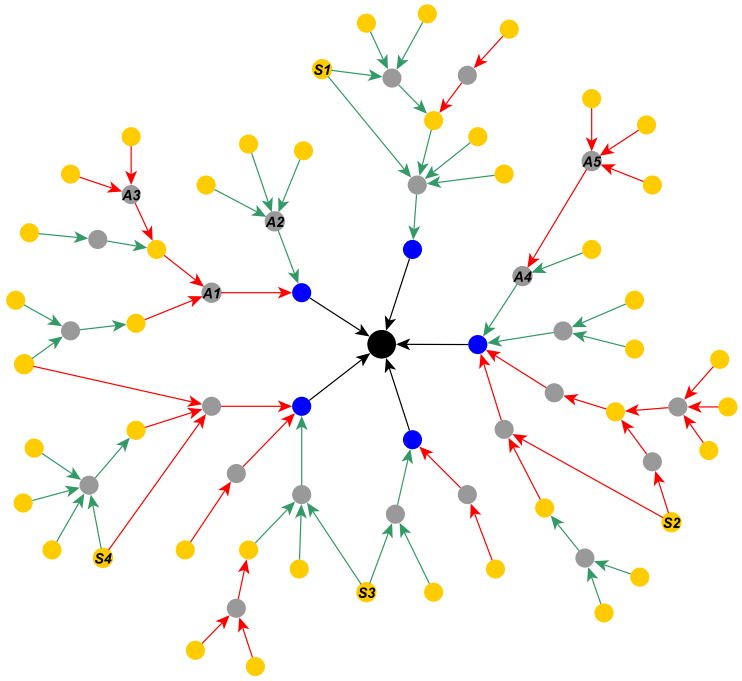


Figure 3.1.: An exemplary argumentation graph. The black dot represents the issue of the discussion. The blue dots represent positions, the gray dots represent arguments and the yellow dots represent statements.

which is either being attacked or supported. On the other hand, they succeed and are adjacent to at least one or to a group of statements which constitutes the premises. Branches or subgraphs of the argumentation graph are argumentations and may be interconnected, i.e. a statement of an argumentation can be reused and included in another one.

From the linguistic perspective, statements are text fragments written by the participants and framed by the system with sentence openers (see Section 1.1). For instance, the text of a statement could be "The road is wet" =: s_1 or "it is raining" =: s_2. An argument, if it is not undercutting (more about this in the following subsection), is connecting exactly one statement with another one or more other statements by constituting a reason-relation between them. For example, "s_1 because s_2" is a valid argument. The linguistic structure of statements is addressed in Section 3.1.5 in more detail.

3.1.1. Supporting vs. Attacking Arguments

The differentiation between supporting and attacking arguments is significant to understand the the basic structural properties of the studied datasets and to clarify statements can generally not be analysed or interpreted independently from each other.

An argument is a supportive argument if it contains a set of statements (the premises) **speaking for** the validity of another statement (the conclusion). The node A2 in Figure 3.1 shows this kind of relation. Supportive arguments express the users approval of the conclusion and of the reason-relation towards the proposed premises.

An argument is an attacking argument if it contains a set of statements **speaking against** the validity of another statement or the validity of the reason-relation indicated by another argument. The nodes A1 and A5 in Figure 3.1 depict these different kinds of attacks. The users disapproval is targeting one of the three components of an argument:

- If the conclusion of an argument A is the negation of a premise of an argument B, A *undermines* B. In Figure 3.1, $A3$ is undermining $A1$.
- If A's conclusion is the negation of B's conclusion, A is a *rebuttal* of B. In Figure 3.1, $A1$ rebuts $A2$.
- If A implies the negation of B's reason-relation between its premises and its conclusion, A *undercuts* B. In Figure 3.1, $A5$ is undercutting $A4$.

The option to reuse statements within the dialog establishes some structural features of the argumentation graph that will be further discussed in Section 3.1.3. The nodes S1, S2, S3 and S4 in Figure 3.1 show reused statements and set examples for "special edges".

3.1.2. Basic Properties of Currently Available Argumentation Graphs in D-BAS

During the elaboration of this thesis, nineteen argumentation graphs were available in D-BAS. The texts in ten of them are written in German and the texts in nine of them are written in English. Most of them have been compiled by the originators of D-BAS themselves or by

uid	title	language	nodes	edges	arguments	statements	positions
1	Verbesserung des Informatik-Studiengangs	de	507	518	240	266	24
4	Improve the Course of Computer-Science Studies	en	500	511	237	262	23
12	Verteilung von Qualitätsverbesserungsmitteln	de	228	231	111	116	8
5	Pferdehuhn	de	127	129	62	64	5
13	Dummyverteilung	de	76	76	36	39	3
2	Town has to cut spending	en	67	66	30	36	6
3	Improvements of D-BAS	en	46	45	19	26	7
8	Archivierung von Abschlussarbeiten	de	46	45	19	26	7
20	Summary discussion	en	43	43	20	22	2
19	Test Discussion	en	41	40	19	21	1
21	Animal Testing	en	30	29	14	15	1
7	Is the Hololens the best? (Debate 2)	en	24	23	10	13	3
6	Hololens vs other AR devices	en	20	19	8	11	3
10	Pre-Test Feldexperiment	de	17	16	7	9	2
15	Impfpflicht	de	10	9	4	5	1
18	Anwesenheitspflicht an der Uni	de	8	7	3	4	1
14	Kohlekraftwerke	de	4	3	1	2	1
16	Testtopic	en	4	3	1	2	1
9	Energy transition in Twente	en	1	0	0	0	0

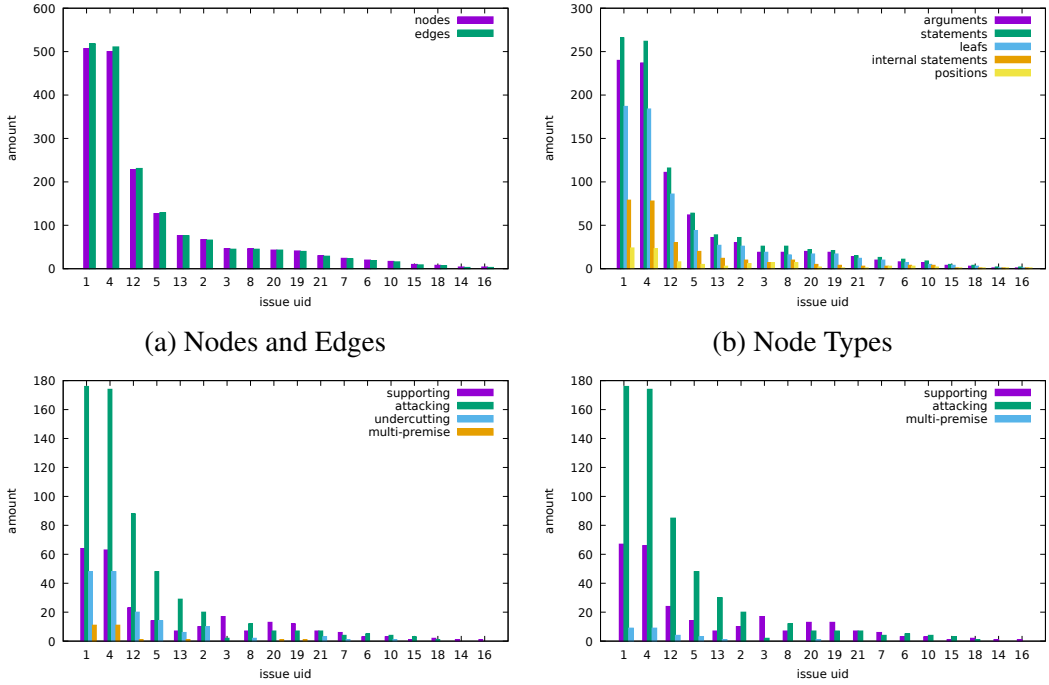
Table 3.1.: Argumentation Graphs – issue UID, Title, Language and Size data

subsequent co-workers and -developers for the purpose of evaluation of the systems functionalities and its performance. Only a few of these argumentation graphs have become bigger than those from the test discussions, since they are the result of actual field studies, which took place at the HHU, in which all students and members of the computer science department have been in invited via e-mail to participate in a real-world online discussion. The findings have been summarized and evaluated in [KMM17], respectively in [Ebb19].

Listed in Table 3.1 is the unique ID, the title and the language of the issues which have been requested from the D-BAS' database. Each line represents the basic attributes of one issue and different size data of the corresponding argumentation graph.

One of the existing argumentation graphs is not included in Figure 3.2a and in the following figures in this chapter, because it does only contain one node, that is the issue itself. In the remaining eighteen graphs, the amount of nodes reaches from four up to 507. Aside from that, the amount of edges reaches from 3 up to 518. The graphs' sizes shown in Figure 3.2a are assigned to the respective issue's unique ID, which this graph has in the system. In the following explanations, the expression *graph i* names and refers to the argumentation graph of the issue with ID i, where $i \in \mathbb{N}$.

Twelve of these argumentation graphs have exactly $n-1$ edges, because they are trees, and n denotes their particular number of nodes, with $n \in \mathbb{N}_0$. From the remaining six graphs, two of



(a) Nodes and Edges

(b) Node Types

(c) Argument Types

(d) Statement Types

Figure 3.2.: Argumentation Graphs' Basic Properties

them have an equal amount of nodes and edges, and four of them have more edges than nodes, namely the graphs 1, 4, 12, and 5. If a graph's number of edges is equal to or greater than its number of nodes, the graph is not a tree, since it contains additional edges interconnecting different subgraphs, which can be obtained by removing the root node. Apart from the root, an argumentation graph consists of statements and arguments using these statements (as premises or conclusions) to attack or to support each other or the positions. Although they account for a relatively small amount of the graphs' nodes, between 3.5% and 4.73% for the five biggest graphs, positions are indicating directions of discussions, offering possible solutions regarding the issue and they are the only nodes being directly connected to it.

As Figure 3.2b shows, statements make up the largest part of nodes for all available argumentation graphs. However, if the number of positions is subtracted from the number of statements, it is equal to the number of arguments for most of the argumentation graphs. This is reasonable because the contribution of a participant usually involves adding exactly one statement and exactly one argument to the graph. Variances may occur if "multi-premise arguments", i.e. arguments having a premise group that includes more than one premise, are formed, or if available statements are reused in new arguments.

Another differentiation of statements is *leafs* and internal nodes. Leafs mean nodes with in-degree 0 or, in other words, statements being neither supported nor attacked. An internal statement is a node with in-degree greater than 0, that is a statement being supported or attacked. For all graphs, except the test discussions with less than four statements, the proportion of leafs to internal statements varies between $3/2$ and $4/1$ showing an average of approximately $7/3$.

According to Figure 3.2c, the amount of attacking arguments outweigh the amount of supporting arguments more than twice in the five biggest graphs. Therein and in graph two, the number of undercuts is similar or up to $1/4$ less than the number of supports and it constitutes about 29% of the attacking statements on average. Aside from that, some graphs include single instances of multi-premise arguments, graph 1 and 4 do even feature eleven of them.

The collected values for statements being used in supportive or in attacking arguments, what is illustrated by Figure 3.2d, resemble the results for arguments in Figure 3.2c. This outcome is plausible, since only some of the statements are being used in more than one argument. They can be called "multi-premise statements" but that means something entirely different than a multi-premise argument, since it indicates the reusability character of statements has successfully been applied. The first five argumentation graphs contain one to nine statements which have been reused at least once.

3.1.3. Implications of the Reusability of Statements

Special edges inside or between subgraphs imply that argumentation graphs are generally not *in-trees*. An in-tree or *anti-arborescence* is a directed (acyclic) graph (DAG), whose underlying (undirected) graph is a tree, with a designated root and all edges converging to it. Argumentation graphs are rooted DAGs because they represent the logical structure of a structured online discussion. They contain logical chains of arguments and counter-arguments but do never include circular argumentation, because it is an error in reasoning and it would reduce the reusability of statements to absurdity. However, circular argumentation may appear in other argumentation graphs than those which have been studied.

For the purpose of correct usage of graph theory terms, if an argumentation graph is not a tree but a DAG, the nodes with in-degree 0 are not called leafs but *sources*, and the root can

also be referred to as *sink*.

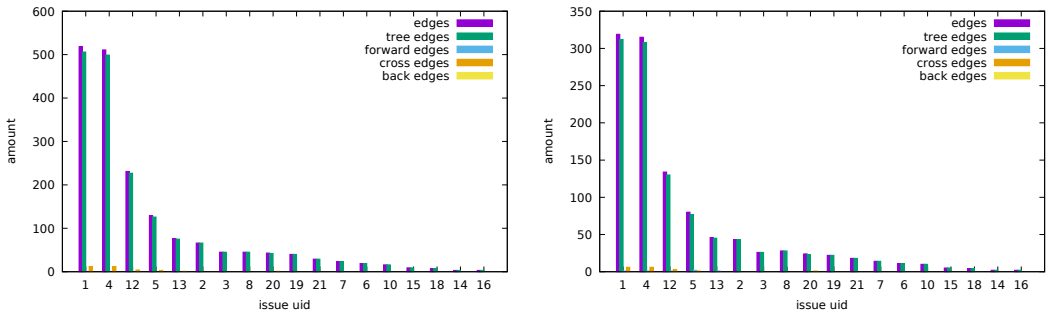
To answer the question, if and and how statements have actually been reused in real-world online discussions managed by D-BAS and to proof that they do not contain any circular argumentation, the depth-first search algorithm has been applied on the available argumentation graphs.

3.1.4. Excursion: Depth-First Search Ordering

The depth-first search (DFS) algorithm is useful to explore the structure of a graph and to differentiate between *tree edges* and additional edges. If the DFS is applied to a graph and the DFS is numbering the graph's nodes in the same order of visitation, the result is a *DFS ordering*. The DFS start index (*DFS-Index*) is assigned to a node and incremented if it is visited for the first time. The DFS end index (*DFE-Index*) is assigned to a node and incremented if all of its successors (child nodes) are visited and the DFS is backtracking the path it came from.

- *Tree edges* are edges the DFS follows, i.e. an edge $(u,v) \in E'$ with $E' \subseteq E$, if $T = (V,E')$ is the spanning tree induced by applying DFS to graph $G = (V,E)$.
- *Forward edges* are edges $(u,v) \notin E'$ with DFS-Index[v] > DFS-Index[u].
- *Cross edges* are edges $(u,v) \notin E'$ with DFS-Index[v] < DFS-Index[u] and DFE-Index[v] < DFE-Index[u].
- *Back edges* are edges $(u,v) \notin E'$ with DFS-Index[v] < DFS-Index[u] and DFE-Index[v] > DFE-Index[u].

Figure 3.3a shows that all edges in all argumentation graphs are tree edges, except for the graphs 1, 4, 12, 5, 13 and 20. They contain additional edges, i.e. edges that are not part of the spanning tree induced by the DFS. These edges are indicating that statements have been reused in real-world online discussions that were managed by D-BAS. However, the additional edges found by the DFS algorithm are apparently all cross edges. A closer look at the available data reveals that this is true because reused statements are separated from the referenced branch by the argument. The approach to potentially identify existing forward (or



(a) DFS – all arguments included

(b) DFS – arguments reduced

Figure 3.3.: Argumentation Graphs' Structural Properties

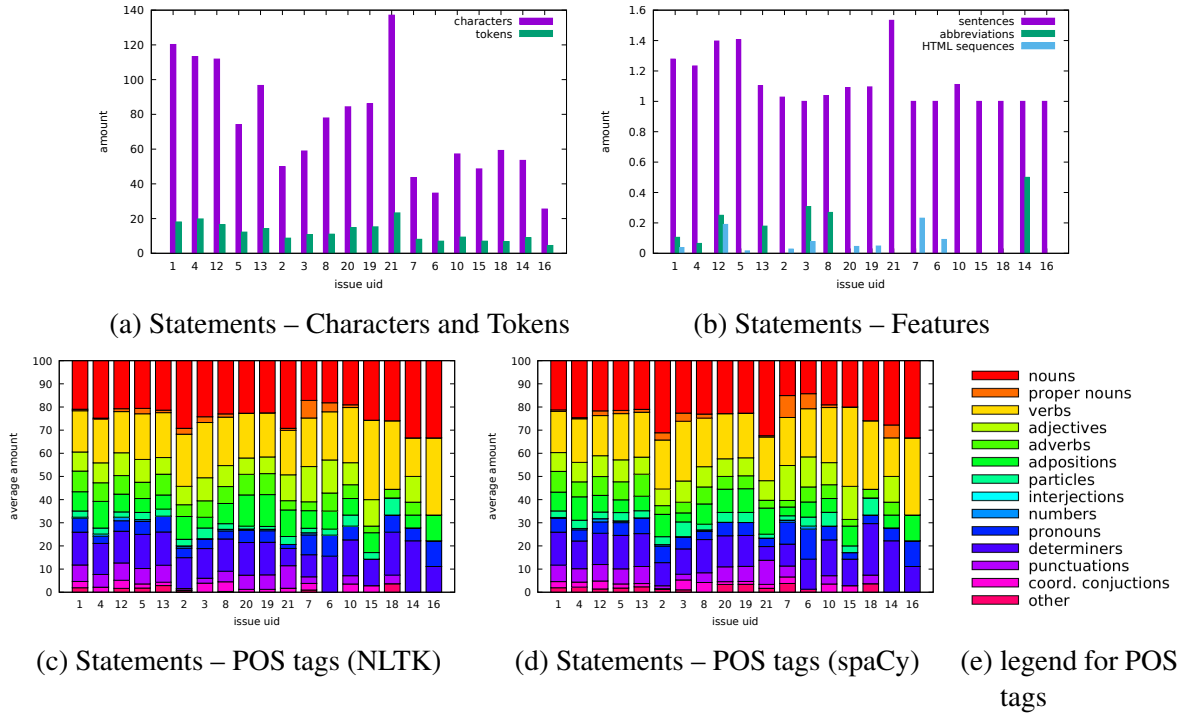
back) edges is to reduce the set of arguments by the following two rules:

- If an argument v is not undercut, remove it (and its edges) from graph G.
- For all statements being used as premises by v, directly connect the statement with the conclusion of v.

Exploring the resulting graph G' using DFS exposes additional structural properties of G. The results for the available argumentation graphs are shown in Figure 3.3b. The number of edges is more than half of the number of edges in the original graph, because edges between the issue and positions, and the edges of arguments being undercut, were kept. Each of graphs 1, 4, 12 and 13 includes exactly one forward edge and graph 5 contains two forward edges. They are indicating that arguments exist which are reusing statements, from a second argument, as premises to refer to third argument (or a position) from the same branch, which is closer to the root. In comparison to the original graphs, the number of cross edges in graph 1 and 4 is additionally reduced by five, meaning these graphs include exactly five redundant arguments. The fact that none of the available graphs featured a back edge proofs that they are indeed directed acyclic graphs.

3.1.5. Composition of Statements

Not only the examination of the argumentation graph's structure, but the examination of the (linguistic) structure of its statements reveal characteristic properties which are significant to get an overview of the textual data. Since the studied datasets are not previously annotated,



(a) Statements – Characters and Tokens

(b) Statements – Features

(c) Statements – POS tags (NLTK)

(d) Statements – POS tags (spaCy)

(e) legend for POS tags

Figure 3.4.: Argumentation Graphs' Linguistic Properties

some of those linguistic properties can not be discussed without anticipating the application of (open-source) NLP libraries for automatic annotation. The required description about their application for the project of this thesis is included in Section 3.3. The arithmetic mean of the length of statements in the different argumentation graphs is shown in Figure 3.4a. The first observation, based on this data, is that the average length of a statement does generally not depend on the language in which they are written, since the average length of statements in English discussions varies between 25.5 and 137.1 characters or 4.5 and 23.2 tokens, whereas the average length of statements in German discussions varies between 48.6 and 120.2 characters or 7 to 18 tokens. It rather depends on the range of a discussion and the number of participants. The argumentation graphs which are the results of actual field studies (referred to as "AGFS" for the rest of this section), namely the graphs 1, 4 and 12, contain statements with an average length from 111.8 to 120.2 characters or 16.5 to 19.7 tokens. Without considering the graphs 13, 20 and 19 because they are compilations of excerpts from the AGFS, the length of statements in test discussions averages lower than 78 characters or 12.3 tokens. An exception to this observation is formed by graph 21 with the topic "animal testing", which is not the result of a field study but its statements have an average length of 137.1 characters or 23.2 tokens. So, it may be that the length of statements in a discussion is topic-dependent, but this question can not be resolved based on this relatively small amount

of studied data.

The assumption "if the statements of graph *a* have more characters than the statements of graph *b*, the statements of graph *a* have more tokens than the statements of graph *b*" does generally not hold. On average, the statements of graph 1 have more characters but less tokens than the statements of graph 4. This phenomenon is comprehensible due to the different (average) characters per token ratios of languages. Based on the data of the available argumentation graphs, the English statements have 5.6 characters per token, while the German statements have 6.6 characters per token.

Another feature of the statements in the AGFS and in the graphs 5 and 21 is an increased ratio of sentences per statement (greater than 1.2) as Figure 3.4b shows. For the other argumentation graphs the ratio is equal to or less than 1.1 sentences per statement. Every fourth to every third statement in the graphs 12, 3 and 8 contains an abbreviation, whereas in graphs 1 and 4 only every tenth to every fifteenth statement has one. For example, the argumentation graph 3 is the result of a discussion about D-BAS, where six of the eight detected abbreviations in statements are "D-BAS"[1]. The value of 0.5 in graph 14 is a bit higher but not as informative, because the graph only has two statements and one of them contains an abbreviation. Since D-BAS escapes HTML special characters in statements on database insertion, some of the statements contain HTML escape sequences, e.g. "'" for the apostrophe in "town's"[2]. The statements of graphs 12 and 7 include about 0.2 of those character sequences on average, whereas the statements from the other graphs include less than 0.1 of those sequences. Most of the escaped characters are quotation marks and apostrophes.

In order to derive the distribution of POS tags for the statements of the examined argumentation graphs, shown in Figure 3.4c and Figure 3.4d, the POS taggers from NLP libraries are utilized for automatic annotation. Since they are trained taggers, the assignment of POS tags is based on probabilistic guesses. The results from the two different POS taggers are similar but not equal. It has been equally identified that slightly more than 20% of the statements from German discussions are nouns. An exception are the statements of graph 15, depending on the applied tagger, they either have more nouns or more adjectives and pronouns. The statements in some of the English discussions (graph 4, 2 and 21) tend to have more than 24% and up to 32% nouns. It is noticeable that the POS taggers slightly deviate from each

[1] One abbreviation is "UI" and another one is "AI".

[2] `https://dbas.cs.uni-duesseldorf.de/graphiql?query={statement(uid: 43){text}}`

other in the determination of nouns for many of the English discussions (graphs 2, 21, 7, and 6). Up to 10% of the statements in argumentation graphs are proper nouns. The amount of verbs is between 20% and 25% independent of language or graph with the exception of the graphs 15, 18 and 16 where the amount of verbs is between 30% and 35%. Adjectives represent between 7% and 15%, adverbs between 3% and 10%, adpositions between 4% and 15%, and particles between 2% and 8% of a statement. For the statements of English discussions, the second POS tagger estimates the amount of adverbs and adpositions lower and the amount of particles higher than the first tagger. Interjections appear in hardly any argumentation graph. Only two statements from graph 4 contain an interjection [3]. Up to 3% of the tokens in statements are identified as numbers, 3% to 14% as pronouns, 5% to 22% as determiners. Most of the graphs have statements with an average amount of about 15% determiners. The statements in all graphs consist of up to 10% of punctuations, up to 4% of coordinating conjunctions, and up to 4% of other POS tags. Remarkable is, on the one hand, that the second tagger classifies some of the statements' tokens, which have been classified as adverbs, adpositions or determiners by the first tagger, as particles, pronouns, nouns or other tags. On the other hand, statements from German discussions, especially in the AGFS, do feature a quite similar linguistic structure independently from the applied POS tagger.

3.2. Design

In this section, incorporated functionalities and the important design decisions are described that determined the development of the natural language processing unit for summarization of argumentation graphs. Its main purpose is the automatic capturing of statements' contents in argumentation graphs by extracting and assigning the correct keywords or keyphrases from/to them.

3.2.1. Acquiring the Argumentation Graphs

The first and basic challenge consisted in requesting the necessary data from D-BAS for independent storage and further processing. Because D-BAS supports the graph query lan-

[3] They contain the words "oh" and "please".

guage [4] since the release of its APIv2 [5], the required datasets are easily accessible through the provided web interface GraphiQL [6]. If the unique ID of a discussion issue is known, the corresponding argumentation graph can be requested with the following query:

```
query {
  issue(uid: ID) {
    completeGraph
  }
}
```

The response contains a JSON object which holds all the necessary information about the nodes and edges of the graph.

For the purpose of quick offline access and processing of the data, the argumentation graphs are locally stored in addition to the opportunity of requesting them from the remote database. The negative aspect of this decision is the need for methods that detect changes and support keeping track of them to ensure the local datasets are up-to-date. The positive aspect consists in independent processing of the acquired data and establishing a second control instance of validation for it. A positive side effect of this decision was, for example, the identification of inconsistent entries in two tables of the remote database concerning the mapping from statement (or argument) to issue.

3.2.2. Structuring the Data

To prepare the data of an argumentation graph for summarization, its components and the graph itself are first transformed into objects, so that they can be simply passed around in the following processing steps without the need to requery information from the local or from the remote data pool. The purpose of this step is not to mirror the data structures already defined in D-BAS, but to create a similar environment that allows for additional functionality.

The major advantage of this transformation is, on the one hand, the straightforward implementation or application of algorithms for graph data structures, and on the other hand, the

[4] GraphQl `https://graphql.org/`
[5] `https://dbas.cs.uni-duesseldorf.de/docs/api/v2.html`
[6] `https://dbas.cs.uni-duesseldorf.de/graphiql`

opportunity to attach linguistic annotations or attributes to the nodes of the argumentation graph.

3.2.3. The Different Steps of Summarization

Before discussing the actual process of summarization, it is important to define which parts of the argumentation graph are used as input data for it. The statements contain the raw text that has been contributed by participants. The dataset of the issue which can be separately requested via GraphiQL includes relevant text fields for the title, a short description and a longer description about it. The latter may be composed of several sentences. Altogether, these texts are utilized for the summarization process. A statement summarizer is an algorithm which takes an argumentation graph as input, processes its textual data and assigns keywords and keyphrases to each statement inside the graph. The different steps of summarization are discussed in the following sections.

Preprocessing

During preprocessing, the relevant syntactical tasks are solved considering the input texts. An important step before tokenization is to clean the texts from additional whitespace and escape characters. Otherwise, they might be treated as valid tokens by the tokenizer. For example, some of the statements in the actual argumentation graphs contained the token "\\n", which is discarded. Another valuable step before tokenization is the detection of (uncommon) abbreviations or acronyms. An abbreviation which may serve as a reasonable keyword candidate like "D-BAS", would in other ways be treated as two separate tokens by the tokenizer. If an abbreviation is written in extracted form they are contracted as well ("e. g." is contracted to "e.g."). The detected abbreviations are set as tokenizer exceptions. The next steps in preprocessing require that the language in which the texts are written has been previously identified. Otherwise, the results were most probably incorrect. Helpfully, the above-mentioned dataset of an issue provided by the API entails this information.

The tokenizer splits the statements' and the issue's texts into sentences and into words, also called tokens. Afterwards, the tokens are used as the input for the POS tagger. The resulting tagged tokens are fed into the lemmatizer to acquire the lemma for each pair of known word

and known POS tag. The process of lemmatization is important to reduce the amount of possible keyword candidates and to allow adding up different forms of the same word. The final essential step in preprocessing is the identification of stop words to exclude them in the process of keyword extraction. It is useful to incorporate the opportunity to select the list of stop words independently from the applied KE algorithms and NLP libraries used for implementation, since this selection determines the quality of the results.

Two additional optional steps, which are not relevant for keyword extraction, are: For one thing, to identify the negation words inside a statement, because they are usually stop words and excluding them for representation of a statement could invert the original meaning of it. For the other thing, to identify linking words to more accurately represent the different clauses of a statement.

Candidate Creation and Scoring

A probably unconventional approach is chosen for keyword or keyphrase candidate creation and scoring. Since the regarded type of documents is significantly short (see *TF=1 challenge*), the approach to directly apply the KE algorithm on them results in keywords simply being equal to not-stop words. Instead, the followed approach consists in applying the KE algorithm to the whole corpus, which is the concatenation of the preprocessed texts.

The selection of suitable candidates and the subsequent scoring of them primarily depends on the applied KE algorithm. Different KE algorithms can be utilized for this part of the summarization. A fairly simple method entails in selecting single alphabetic non-stop-word tokens and combining them in their order of occurrence after scoring, if they immediately succeed each other in any sentence, to produce phrases. More advanced KE methods use n-grams, skip-grams, noun phrase chunks, POS tag patterns or co-occurring words inside a sliding window of n tokens to determine the candidates. Most KE algorithms exclude stop words, though some include internal stop words for keyphrase candidates. The texts are processed in lemmatized form, so that the KE algorithm considers different versions of words with the same lemma as equal.

The scoring mainly consists in calculating the frequency of occurrence for candidates. The most basic approach is to determine the word frequency distribution and assigning the weighted or normalized frequency as the score. Other KE methods make use of a word's degree, which

is the sum of co-occurrences with other candidates, or build up a co-occurrence graph scoring the nodes based on their degree or based on edge weights (see Section 2.3.2), or incorporate the words pointwise mutual information. Usually, a KE algorithm, that has successfully been applied to a single text document, returns the top n keyphrases, where $n \in \mathbb{N}_0$. Whereas, in case of the followed approach, another step is necessary to correctly assign the results to the individual statements.

Keyword and Keyphrase Assignment

For each statement, only the keyphrases (or keywords) the statement contains are considered as relevant for it. A statement contains a keyphrase, if it contains not only the same sequence of characters but the same sequence of (lemmatized) tokens as the keyphrase does. This restrictive definition serves to avoid incorrect assignments like: “less net” in “wireless network”.

Three different methods are applied to filter potentially assigned keyphrases:

- A keyphrase is dropped if it is a subsequence of another keyphrase which has already been selected for the statement.
- The score of a keyphrase, which is assigned to the issue, is adjusted by a weighting factor $w_1 \in [0,1]$ for statements that include more than just the issue’s keyphrases.
- The score of a keyphrase, which is assigned to one of the statement’s immediate successors, is adjusted by a weighting factor $w_2 \in [0,1]$.

For comparison of different strategies of selection or assignment, at least three different methods are supported: *top n*, *top r* and *threshold t*.

Top n is an approach which is commonly used in literature. For a predefined absolute value $n \in \mathbb{N}_0$, the top n candidates for a text document are selected as keyphrases (or keywords) for it. The value for n is usually set between five and twenty (see [Tur99] for instance). However, a fixed value of n does not necessarily take the text documents length into consideration.

Top r is a rather flexible approach which considers the length of the text document. The number n of selected keyphrases is relatively defined by $n = r \times |C|$ with predefined fraction

$r \in [0,1]$ and $|C|$ being the number of candidates. For example, Mihalcea and Tarau argue for this approach in [MT04].

Threshold t is a rather unconventional and naive approach. Each statement is "filled" with the candidates being representative for it. Beginning with the candidate which has the highest score, the relevant candidates C_a for statement a are marked as representative (marked as keywords or keyphrases respectively) until the sum of their score reaches or exceeds a threshold of $t \times \sum_{c \in C_a} s_c$, where $t \in [0,1]$ and s_c is the score of c. For $0 \leq t \ll 1$, this procedure of assignment ensures that candidates with relatively low scores do not represent statements, if they include candidates with higher scores. For $t = 1$ are candidates not selected which have a score of zero.

Postprocessing

The final step of postprocessing is not necessary for the purpose of keyphrase extraction and assignment. Nevertheless, the output format for the summarization results can be adjusted, they can be augmented with additional non-keywords, or further analysis steps can be performed.

Two reasonable improvements for more accurate representation of statements are, for instance, to reestablish the order of selected keyphrases in which they appear in the original statement, or to add occurring negations to the appropriate keyphrases to indicate their original function inside the statement.

3.2.4. Data Collection for Statistics

During the whole process of summarization, an analyser is gathering derived information about the inspected argumentation graph, the composition of its statements and the quality of the results. The summarizer utilizes simple nameable counters and series which are traced by the analyser. Measured values are for example the length of a statement in characters or in tokens, or the amount of each assigned POS tag inside the statement.

At the end of summarization, the analyser evaluates the results by applying different metrics, like *precision* or *recall*. A more detailed study of the requirements and of the suitable metrics

(see 4.3) for this purpose is undertaken in Chapter 4. Finally, the analyser creates a condensed form for all the collected data, meaning that statistical measures like arithmetic mean or standard deviation are calculated and output to represent the collected series of data.

3.3. Implementation

The processing unit for the summarization of statements in argumentation graphs was implemented in Python. This programming language was chosen because it features a comprehensive standard library and an easy access to many open source libraries, especially libraries that offer efficient solutions for the various tasks of natural language processing.

3.3.1. The Natural Language Toolkit

The Natural Language Toolkit[7] is a popular open-source NLP library which was initially released in 2001 and is since then being frequently used in academic teaching and research. NLTK offers modules for simple statistics, syntactical and semantical analysis, clustering and classification tasks. Its biggest advantage are the well documented and easy-to-use interfaces for over 50 different corpora and lexical resources.

To implement the functionalities described in the previous section, NLTK has been applied for tokenization, POS tagging, lemmatization and stop word identification. NLTK does not include any algorithms for keyword extraction. However, the package *rake-nltk*[8] contains an implementation of RAKE (explained in Section 2.3.2) which is based on NLTK. This implementation of RAKE and another simple approach, which consists in extracting, scoring by collection frequency[9] and combining single (not-stop) words, are used to solve the tasks of keyword or keyphrase candidate creation and scoring for the process of summarization.

The texts of available argumentation graphs are either written in English or in German. NLTK supports POS tagging for English and for Russian texts. Lemmatization in NLTK is provided

[7] `https://www.nltk.org`

[8] `https://pypi.org/project/rake-nltk/`

[9] In this context, "collection" refers to the aggregate of textual content from the different nodes of the argumentation graph (see 3.2.3).

using WordNet, a large lexical database for English. Currently, NLTK does not support POS tagging or lemmatization of German texts. Therefore, it was necessary to find working solutions for these tasks.

Training a POS Tagger for German Texts

Different trainable taggers are offered by NLTK in the "taggers" package, which can be utilized to facilitate POS tagging of German texts. A valuable resource for this task is the TIGER corpus[10], a collection of approximately 900.000 annotated tokens from German newspaper articles. The TIGER corpus has been developed in collaboration of the universities of Potsdam, Stuttgart and Saarbrücken.

The first approach consisted in training a Bayes classifier based tagger. Varying selections of different features have produced results between 94% and 96.5% of accuracy. The standard NLTK POS tagger for English texts is the "averaged perceptron tagger"[11]. After training the perceptron tagger on the TIGER corpus, its accuracy is about 97.1%. Incorporating two additional features, the length of a token in characters and the pre-predecessors' POS tag plus the token itself, improved the accuracy by approximately 0.1%. The thereby induced model of this slightly modified version of the perceptron tagger is used to determine the POS tags of tokens for the German texts of argumentation graphs.

Lemmatization of German Texts

To enable the lemmatization of German texts for the process of summarization, the Inverse Wiktionary for Natural Language Processing (IWNLP) lemmatizer [12], described in [LC15], is applied.

Apart from the requested word, the IWNLP lemmatizer takes the word's universal POS tag[13] as an argument. Since the model of the modified perceptron tagger was trained on the TIGER corpus, it's output tags are from the Stuttgart-Tübingen Tagset. A tagset map, which has been

[10] https://www.ims.uni-stuttgart.de/forschung/ressourcen/korpora/tiger/
[11] https://www.nltk.org/_modules/nltk/tag/perceptron.html
[12] https://dbs.cs.uni-duesseldorf.de/research/IWNLP/
[13] https://universaldependencies.org/u/pos/

parsed from the universal dependencies website[14], is used to infer the correct universal POS tags.

3.3.2. spaCy

Another comprehensive open-source NLP library is spaCy which is considerably newer than NLTK because its initial release was in 2015. The major differences in comparison to NLTK are:

- The design of spaCy is rather minimal and not as flexible as NLTK with regard to the possible solutions for NLP tasks.
- The basic approach to NLP is object-oriented instead of NLTK's "strings-to-strings".
- It does support word vectors, dependency parsing, and features pretrained statistical models for 11 languages.

For the implementation of the functionalities, described in Section 3.2, spaCy has been applied for the same preprocessing tasks as NLTK. Algorithms for keyword extraction are not included in spaCy, but the package *textacy*[15], which is based on spaCy, is offering solutions to NLP tasks "before and after spaCy", and contains an implementation of the TextRank algorithm (explained in Section 2.3.2). This implementation of the TextRank algorithm and another simple approach, which consists in extracting (not-stop-word-containing) n-grams and scoring them using (weighted) pointwise mutual information [16], are applied to enable keyword or keyphrase extraction for the process of summarization with spaCy.

During the implementation, a minor problem occurred in the process of German stop word identification. Another problem arose from the design decision to separately preprocess the texts from statements and issue, but to concatenate them for candidate creation and scoring. Both problems and their solutions are addressed in the following subsection.

[14] `https://universaldependencies.org/tagset-conversion/de-stts-uposf.html`
[15] `https://chartbeat-labs.github.io/textacy/build/html/index.html`
[16] The inspiration for this approach was taken from [Bou09].

3.3.3. Excursion: Contributing to spaCy

Addressing stop word identification, the German noun “Menschen” which appears in some of the text contributions in the available argumentation graphs was identified as a stop word by spaCy, whereas the word “einige” was not. The explanation for this unusual decision is, that spaCy utilizes a list of stop words for each language and the German stop word list contained the token “Menschen” but not “einige”. The reasons for this minor bug are that the separating whitespace between “einige” and “einigen” was missing, and the German stop word list was created by means of the TIGER corpus in which “Menschen” is the 115th of most common words. So “Menschen” may be a good stop word in the context of news articles but is not applicable for general use. The git merge request including the proposed changes was directly accepted by the maintainers of spaCy.

The other problem existed because spaCy supports parallel processing for large volumes of text with the “Language.pipe” method, but does not support merging of the resulting multiple “Doc” objects into one. To keep the results from preprocessing it was necessary to implement a method to solve this task. Basically, the implemented method takes a list of “Doc” objects (docs), an optional list of attributes and a single flag to indicate if single spaces should be inserted between the concatenated docs. It raises an error if the docs do not all share the same “Vocab” object and it returns the new “Doc” object that is containing the other docs or None, if docs is empty or None. This working solution has successfully been applied in this thesis’ project and has been proposed to the maintainers of spaCy, but has not been accepted yet.

3.3.4. Summary of Implemented Statement Summarizers

Four different statement summarizers have been implemented during the progression of this thesis. Two of them are based on NLTK and two of them are based on spaCy. For the purpose of simple identification and improved readability, the naming scheme “<KE algorithm> summarizer” is employed to describe and reference the statement summarizer which uses a certain KE algorithm for candidate creation and scoring. The *single-word summarizer* is based on NLTK, determines single word candidates, scores them on collection frequency and combines them to form keyphrase candidates. The *RAKE summarizer* is also based on NLTK, creates candidates just like the single-word summarizer, but scores them using word’s degree in proportion to its frequency. The *n-gram summarizer* is based on spaCy, creates candidates

for $n > 0$ until no longer sequences of valid tokens are found, and scores them by means of self-information (for $n = 1$) and pointwise mutual information (for $n > 1$). The *TextRank summarizer*, is based on spaCy as well, determines single word candidates, builds a word co-occurrence graph, applies the PageRank algorithm for scoring and forms keyphrase candidates by joining words which are linked inside the graph (considering order and distance in which they occur together).

Both, the single-word summarizer and the n-gram summarizer, are rather naive approaches which have been implemented to provide simple solutions to the problem of statement summarization in argumentation graphs, and to demonstrate their performance in comparison to summarizers which use the well known unsupervised KE algorithms RAKE and TextRank.

Chapter 4.

Evaluation

The application of the evaluation measures, described in Section 4.3 of this chapter, is not possible without having annotated data sets. In the context of keyphrase or keyword extraction from statements that means it is necessary to have a set of previously defined keyphrases and a set of previously defined keywords for each statement.

4.1. Manual Annotation

In order to perform the evaluation of the applied and the implemented methods for summarization, a total of 939 statements have been manually annotated by the author of this thesis. In sum, the number of the different keywords which have been assigned to statements is 2743 and the number of different keyphrases is 2719. On average, each statement is represented by 6.07 keywords and 3.97 keyphrases.

Since summarization of statements in argumentation graphs constitutes a new challenge for the application of NLP algorithms, there are no guidelines of how to properly annotate the statements or how to extract their contents. However, during the process of manual annotation of the studied data sets, the author resorted to certain heuristic annotation principles which served to generate the respective results. These rules of how to determine the keywords and the keyphrases of a statement can be described as follows:

1. A keyword candidate is a word which occurs inside the statement, but is not very

frequently used in the language in which the statement is written.

2. The keyword candidate's part of speech is either noun, adjective, full verb or adverb.

3. A keyphrase candidate is a continuous sequence of keyword candidates (including single words which do not have immediate neighboring candidates).

4. Start from the center of the particular argumentation graph and arbitrarily choose a position which has not been chosen before, until all positions have been selected once.

5. Identify the potential keyword and keyphrase candidates for this statement.

6. Form single candidates from different inflected forms of the same candidate and use the lemmatized form of each candidate for the following rules. This step is necessary to correctly "match" (see 4.2) selected candidates when applying the evaluation measures.

7. Drop candidates which are duplicates or subsequences of other candidates.

8. For the two following rules: Do not exclude keyword candidates which are part of a keyphrase candidate which is not selected for the regarded foreign node (the issue or a successor). In other words, keep a keyword candidate for a statement if it occurs inside a new keyphrase candidate.

9. Exclude candidates which also occur in the description of the argumentation graph's issue, if the statement is not a position and the statement has more than those candidates. Otherwise, if the keywords or keyphrases are applied to retrieve relevant statements using a search query, each statement which references the issue by containing some of its candidates (if those index terms are queried) would be considered as a "hit".

10. Leave out candidates which are already selected for a statement s_i which is directly "above" this statement s_j (i.e. $\text{depth}(s_i) < \text{depth}(s_j)$), if it has more than those candidates. Otherwise, if the keywords or keyphrases are applied to produce a highlighted or reduced view of the argumentation graph, for each branch, the same keywords or keyphrases would be repeatedly shown.

11. The remaining candidates are respectively selected as keywords or as keyphrases for this statement.

12. Continue with the statements s_k "below" this statement s_j, that is, continue with statements which are linked to this statement and depth(s_j) < depth(s_k), until all statements in this branch (or subgraph) are annotated.

The resulting data set annotations are by no means irrefutable, since they have been produced by only one author and not by many authors or annotators who agreed on them. Changing the set of rules which is utilized to generate the keywords and keyphrases for statements, could produce different results as well. Nevertheless, the manually extracted data sets of keywords and keyphrases can be used to evaluate the different algorithms and approaches which have been implemented or applied in the natural language processing unit for summarization of statements.

4.2. Matching of Words or Phrases

To measure the performance of the different algorithms and approaches which have been utilized for keyword and keyphrase extraction from statements, some of the basic metrics which are commonly used in information retrieval can be employed. The principle of these measurements is to compare the output of the summarizer, i.e. the automatically generated sets of keywords and keyphrases, with the manually generated results. However, the validation of the consistency of results can not be achieved by simply counting the "hits" in terms of equality of character sequences. If the summarizer selects the keyphrase "computer network" for a statement, but the manually created set of keyphrases for the statement contains the phrase "computer networks", those keyphrases only differ in the form of their grammatical number. Different inflected forms of the same keyword or keyphrase correspond with, or *match* each other. Two phrases, which consist of the same or "nearly" the same words, but the words occur in a different order, do not match, because they could have an entirely different meaning. For instance, the phrase "snowflake pattern" does not match the phrase "pattern snowflake". A keyphrase *a*, which consists of a subsequence of words of another keyphrase *b*, does not match *b*, because it is not capturing the particular meaning of it. Turney proposes in [Tur97] that "a handmade keyphrase matches a machine-generated keyphrase when they correspond to the same *sequence of stems*". His approach is followed and applied in this thesis to evaluate the algorithms being implemented. In the following section, the word *keyphrase* is replaceable by the word *keyword* because the same principles are applicable.

	K_h	$\overline{K_h}$
K_c	a	b
$\overline{K_c}$	c	d

Table 4.1.: Possible outcomes of binary keyphrase classification. The different sets K_h, K_c and different variables $a,b,c,d \in \mathbb{N}_0$ are explained in Section 4.3.

4.3. Suitable Metrics

As aforementioned in 2.3.1, keyphrase extraction can be described as a binary classification task. Either a phrase is a keyphrase or it is not a keyphrase. According to Turney, the scheme for comparison between the computer-generated and the anthropogenic results can be displayed in a confusion matrix.

The possible outcomes of binary keyphrase classification are shown in Table 4.1. In this overview, K_h denotes the (actual) set of keyphrases which has been previously defined by a human (usually the annotator of the text data) or resolved in consensus of many. K_c is the (predicted) set of keyphrases which has been generated by the classificator, i.e. the algorithm for keyphrase extraction which is executed by a computer. $\overline{K_h}$ and $\overline{K_c}$ are the sets of phrases which are not classified as keyphrases by a human or respectively by a KE algorithm. The different variables $a,b,c,d \in \mathbb{N}_0$ inside the confusion matrix denote the sizes of the intersections between the different sets, with $0 \leq a \leq min(|K_h|,|K_c|)$, $0 \leq b \leq min(|\overline{K_h}|,|K_c|)$, $0 \leq c \leq min(|K_h|,|\overline{K_c}|)$ and $0 \leq c \leq min(|\overline{K_h}|,|\overline{K_c}|)$.

A commonly used metric in information retrieval for measuring the performance of an algorithm is accuracy.

$$\text{accuracy} = \frac{a+b}{a+b+c+d} \tag{4.1}$$

This metric can generally not be applied to evaluate a KE algorithm, because “the space of keyphrase candidates is not well defined” as stated by Turney. According to him, a text document can be segmented into a set of phrases in many different ways depending on the selected approach for candidate creation. Moreover, even on the condition that selected terms

have to occur at least once in the document, it is common that people who write a set of keyphrases to describe a document, do not provide a set of non-key phrases. For this reason, a concrete or an approximate value of d can usually neither be determined nor estimated. The number of correctly classified, matching keyphrases is represented by a. The value of b is the number of redundant keyphrases being additionally generated by the KE algorithm ($b = |K_c \setminus K_h|$). The value of c is the number of keyphrases which have been selected by the annotator but have not been identified by the KE algorithm ($c = |K_h \setminus K_c|$).

Metrics being used in information retrieval, which can be applied to evaluate KE algorithms and only depend on a, b and c, are *precision*, *recall* and F_1 *score* (also known as *F-measure*).

$$\text{precision} = \frac{a}{a+b} \tag{4.2}$$

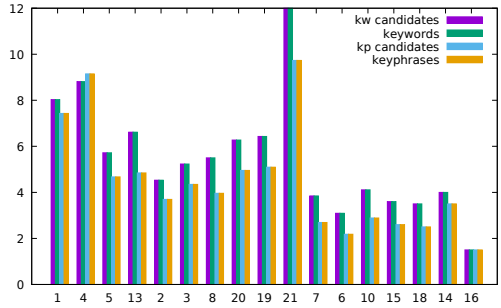
$$\text{recall} = \frac{a}{a+c} \tag{4.3}$$

$$F_1 \text{ score} = \frac{2a}{2a+b+c} = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{4.4}$$

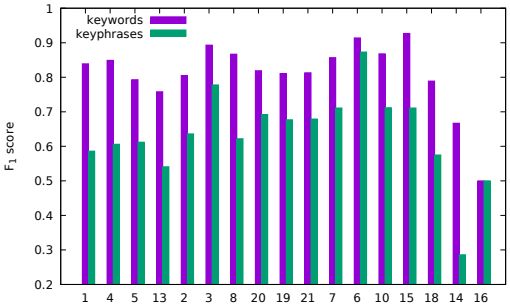
Precision, which is declared in 4.2, is a measure which represents number of correctly classified keyphrases a in proportion to the number of totally extracted (or generated) keyphrases $a+b$. Recall, which is declared in 4.3, is a measure which represents number of correctly classified keyphrases a in proportion to the number of predefined (and thus expected) keyphrases $a+c$. The F_1 score, which is declared in 4.4, is a measure of balance between precision and recall, which is always less or equal the average of them. In order to evaluate the performance of the statement summarizers which have been developed during the progression of this thesis, these metrics are utilized.

4.4. Setup

Eighteen different argumentation graphs which have been discussed in detail in Section 3.1 can be used for evaluation. Seventeen of them serve to find the optimal values for the parameters, the best keyword or keyphrase selection strategy, and to determine the best working statement summarizer. One of the argumentation graphs is not included in the optimization



(a) The average number of assigned candidates, selected keywords and keyphrases (per statement)



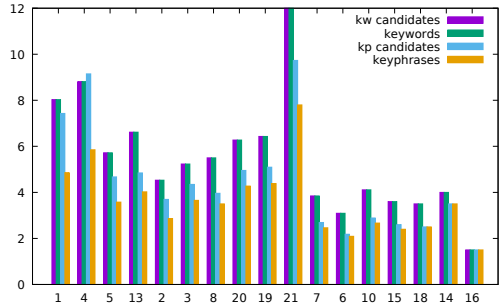
(b) The average F_1 scores

Figure 4.1.: Results of the single-word summarizer, run without applying any weights or filters

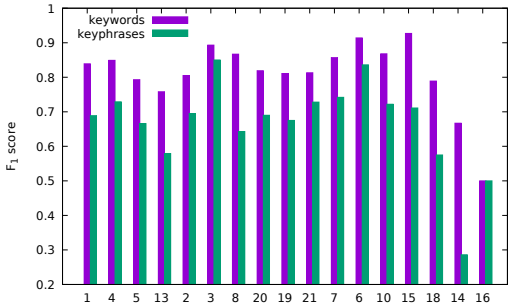
procedure but it is reserved and used as a validation dataset. Graph 1[1] contains the most statements and was constructed by many different participants during a field study of D-BAS, but it is ineligible because graph 4 is the translated, English version of graph 1. None of them could be used to draw final results without previously excluding the other one, because otherwise the evaluation would be biased, if it refers to (nearly[2]) the same graph and linguistic structure that was used for optimization. Graph 12[3] is selected as the validation dataset because it is preferably suitable[4]. From the eighteen argumentation graphs, it contains the third most statements and is the result of a real world online discussion.

As Figure 4.1a illustrates for each of the argumentation graphs, the average number of assigned candidates per statement equals the average number of determined keywords and keyphrases per statement, respectively, if no filter or weight is applied and the selection method allows all of them (e.g. for top r with $r = 1.0$). As soon as subsequences are filtered, the average number of determined keyphrases per statement decreases, like Figure 4.2a depicts. The average number of keywords per statement is not effected by this filter since a candidate words are in lemmatized form and a word is not considered to be a subsequence of another word. This impact of candidate subsequence filtering is shown exemplarily for the single-word summarizer but holds true for the other summarizers.

[1] The topic of graph 1 is “Verbesserung des Informatik-Studiengangs”.
[2] See Section 3.1.2 and Section 3.1.5 for the explanation of differences between them.
[3] The topic of graph 12 is “Verteilung von Qualitätsverbesserungsmitteln”.
[4] Actually, graph 12 was reserved right from the start of the study. It was never looked at (except for the manual annotation) and its linguistic characteristics never influenced any design or implementation decisions.



(a) The average number of assigned candidates, selected keywords and keyphrases (per statement)



(b) The average F_1 scores

Figure 4.2.: Results of the single-word summarizer, run with just filtering subsequence candidates

For each of the argumentation graphs, the average F_1 score (of keyword and keyphrase selection) which is reached when using the single-word summarizer without any restrictions is listed in Figure 4.1b. It is noticeable, that with these non-restrictive settings, the average F_1 score concerning the keywords is already above 0.8 for most of the graphs. An improvement for the average F_1 score of selected keyphrases, which eventuates when applying the subsequence filtering, can be observed in Figure 4.2b.

For the purpose of using aggregated statistical information and to capture the performance of a particular summarizer in general (over all argumentation graphs), the weighted arithmetic mean of the average values for each different metric can be calculated. This evaluation measurement for F_1 score, precision and recall, is formed by normalizing the measured average values for the different argumentation graphs based on their number of statements. To improve the readability of the following sections, the author refers to these meta metrics as $\overline{f_1}$ score, $\overline{precision}$ and $\overline{recall}$, respectively.

4.5. Determination of Optimal Summarizer and Settings

Several runs of the summarizers have been performed with many different configurations to find the optimal values for the parameters, to determine the best selection method, and to decide which statement summarizer exhibits the best performance. Some of the intermediate results can be found in Appendix A. The configurations which cause high $\overline{f_1}$ scores for

	top n		top r		threshold t	
	(iw, sw, n),	$\overline{f_1}$	(iw, sw, r),	$\overline{f_1}$	(iw, sw, t),	$\overline{f_1}$
single-word	(0.0, 0.0, 20),	0.83	(0.0, 0.0, 1.0),	0.83	(0.0, 0.0, 1.0),	0.86
RAKE	(0.0, 0.0, 20),	0.83	(0.0, 0.0, 1.0),	0.83	(0.0, 0.0, 1.0),	0.86
n-gram	(0.0, 0.0, 20),	0.85	(0.0, 0.0, 0.95),	0.85	(0.0, 0.0, 1.0),	**0.89**
TextRank	(0.0, 0.0, 20),	0.85	(0.0, 0.0, 0.95),	0.85	(0.0, 0.0, 1.0),	0.89

Table 4.2.: Summarizer comparison for keywords on optimal values for parameters. ($iw :=$ weight for issue's candidates, $sw :=$ weight for successors' candidates and $\overline{f_1} :=$ weighted arithmetic mean of F_1 scores, with $n \in [0,20]$ and $iw, sw, r, t, \overline{f_1} \in [0.0, 1.0]$)

assigned keywords are listed in Table 4.2. For $n = 20$ the selection strategy top n yields a higher $\overline{f_1}$ score than it does for lower values of n. Most statements have significantly less keywords associated (in the manually created datasets for keywords) and do not contain that many candidates. Although the $\overline{precision}$ is slowly decreasing for $n > 4$, the recall is rising because in most argumentation graphs exist statements[5] with more than four associated keywords which are eventually identified for higher values of n. The most promising results can be observed for the naive approach threshold t with parameter values $iw = 0.0$, $sw = 0.0$ and $t = 1.0$. It is superior to the other selection strategies because it never choses a candidate whose score is reduced to zero.

	top n		top r		threshold t	
	(iw, sw, n),	$\overline{f_1}$	(iw, sw, r),	$\overline{f_1}$	(iw, sw, t),	$\overline{f_1}$
single-word	(0.0, 0.0, 8),	0.63	(0.0, 0.0, 0.65),	0.65	(0.0, 0.0, 1.0),	0.73
RAKE	(0.0, 0.0, 8),	0.63	(0.0, 0.0, 0.75),	0.65	(0.0, 0.0, 1.0),	0.73
n-gram	(0.0, 0.0, 4),	0.64	(0.0, 0.0, 0.55),	0.66	(0.0, 0.0, 1.0),	**0.80**
TextRank	(0.0, 0.0, 10),	0.63	(0.0, 0.0, 1.0),	0.63	(0.0, 0.0, 1.0),	0.65

Table 4.3.: Summarizer comparison for keyphrases on optimal values for parameters (with $iw, sw, n, r, t, \overline{f_1}$ equally defined as in Table 4.2)

A similar behavior is observable for the different summarizer configurations which cause high $\overline{f_1}$ scores for assigned keyphrases. As Table 4.3 shows, the summarizers perform well with top n for $n \in [4,10]$, but top r is preferable to it. However, the best selection method for keyphrases is threshold t because it yields the highest $\overline{f_1}$ scores. The relatively poor performance of the TextRank summarizer is explainable due to the fact that the utilized implementation of the TextRank algorithm, which is mentioned in Section 3.3.2, does pre-filter

[5] A very few of these statements do not have the characteristics of a proper statement but consist of multiple sentences or more than 50 words.

similar terms. Because of this, the TextRank summarizer is missing keyphrase candidates which are relevant for some statements[6]. Over several runs with different configurations the n-gram summarizer, in combination with selection method threshold t, has produced the best $\overline{f_1}$ score.

4.6. Final Results

For the purpose of validation, argumentation graph 12 has been reserved as explained in Section 4.4. In Table 4.4 are the different metrics listed to show the performance of the different summarizers. Concerning the keywords, the single-word and the RAKE summarizer exhibit lower precision than recall while the n-gram and the TextRank summarizer have a slightly increased precision in comparison to recall. The F_1 score is accordingly of similar height. The major advantage of the spaCy based summarizers in comparison to the other which are based on NLTK, is spaCy's comprehensive stop word list which helps to avoid most of the unwanted words to be considered as candidates. This condition might be the main reason for this differences.

	keywords			keyphrases		
	precision	recall	F_1 score	precision	recall	F_1 score
single-word	0.84	0.90	0.86	0.69	0.71	0.70
RAKE	0.80	0.90	0.84	0.68	0.67	0.67
n-gram	0.90	0.88	**0.88**	0.72	0.73	**0.72**
TextRank	0.92	0.85	0.87	0.66	0.59	0.61

Table 4.4.: The statement summarizers performance measured against argumentation graph 12

For the keyphrases, precision and recall for each of the summarizers are quite balanced (difference ≤ 0.02), except for the TextRank summarizer which does not identify as much keyphrases correctly as the other summarizers. A reason for this characteristic was given at the end of the previous section. It is the lowest performing summarizer regarding the extraction and correct assignment of keyphrases. The minor difference in performance between the single-word and the RAKE summarizer exists because the latter assigns more candidates

[6] A more suitable implementation of the TextRank algorithm could yield better results.

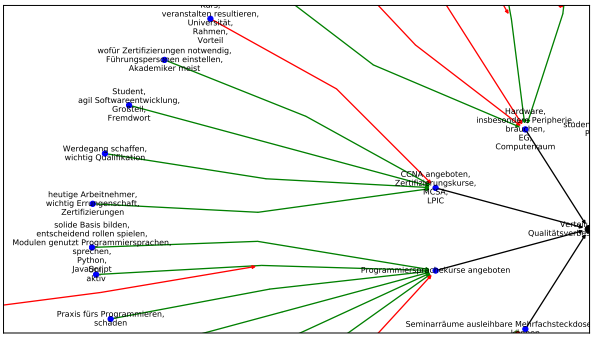


Figure 4.3.: Subgraph of argumentation graph 12 with automatically extracted and assigned keyphrases for statements.

to the statements. This is because the utilized RAKE implementation does not (successfully) avoid non-alphabetic characters and sets candidate words to lowercase, which is not necessary since they are previously lemmatized during preprocessing[7].

The best performance, considering the extraction of candidates from argumentation graphs and correctly assigning them as keywords and keyphrases to statements, is presented by the n-gram summarizer. The F_1 score of 0.88 for the assignment of keywords is slightly lower than the previously determined $\overline{f_1}$ score of 0.89 which was documented in Table 4.2. The F_1 score of 0.72 for assigned keyphrases is lower than the previously observed $\overline{f_1}$ score of 0.80 which can be found in Table 4.2. The reason for this may be some incorrectly identified keyphrases containing stop words or non-keyword candidates which have not been considered. A more accurate or topic dependent stop word list, or the incorporation of more linguistic knowledge as proposed by Hulth (see Section 2.3.1), could improve the performance of the statement summarizer. Decisive for the quality of the results of a statement summarizer is a well structured preprocessing pipeline, a restrictive algorithm for candidate creation and scoring[8], distinctive parameters or weights which can be applied to those candidates, and finally a suitable selection method which may refer to previously generated knowledge to decide if a keyword candidate is a keyword or not (for keyphrases respectively).

In Figure 4.3, a subgraph from graph 12 is shown with keyphrases as node labels instead of whole text contributions.

[7] An appropriate implementation of RAKE would be preferable.
[8] An algorithm which creates and scores candidates but does not pre-filter possible candidates.

Chapter 5.

Conclusion and Future Work

During the progression of this thesis, four different statement summarizers have been implemented which share a common base: The natural language processing unit which solves the task of summarization of statements in argumentation graphs. The performance and the drawbacks for each of the summarizers was determined and discussed in Section 4.6. Their common basic functionalities suffice to identify most of the keywords and keyphrases which have been previously selected during manual annotation. Although, not many argumentation graphs have been available, they were studied in detail to implement appropriate and sufficiently well performing algorithms. An issue for the automatic assignment of keywords are malformed statements, since no preference for correct candidate selection could be inferred.

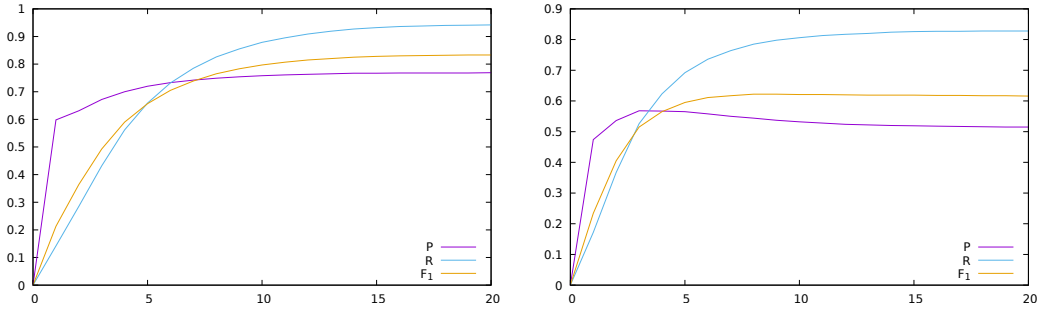
In future work, the creation of different context or topic related stop word lists and their impact on statement summarization could be researched. Since the described rules for manual annotation neither consider malformed statements, nor do they heavily restrict keyword and keyphrase selection, an extended set of annotation rules, probably compiled in consensus of two or more annotators, could provide superior results.

Incorporating more linguistic knowledge like syntactical dependency or named entities into the procedure of candidate creation, scoring or selection, could increase the performance of summarizers. Additionally, the application of different machine learning approaches could significantly benefit the summarization of statements in argumentation graphs.

Appendix A.

Optimization

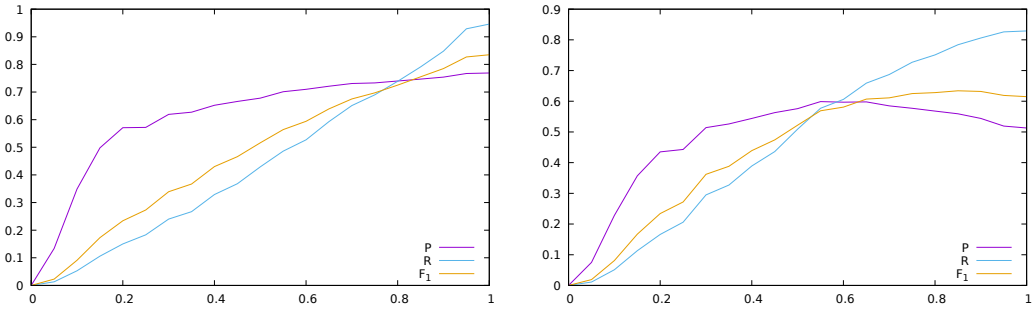
A small selection of intermediate results from the determination of optimal values for parameters, the best selection method and the best statement summarizer. The metrics $\overline{precision}$, $\overline{recall}$ and $\overline{f_1}$ are explained in Section 4.4 (in the following diagrams they are denoted with P, R and F_1).



(a) metrics for keywords

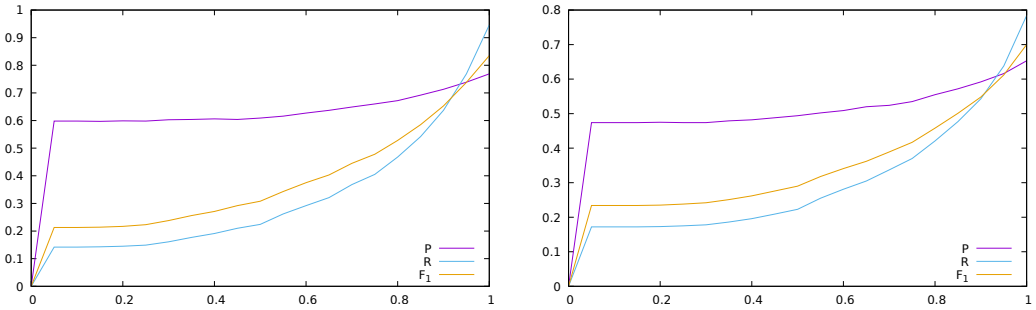
(b) metrics for keyphrases

Figure A.1.: single-word summarizer with selection method top n (for $n \in [0,20]$ and subsequence filter only)



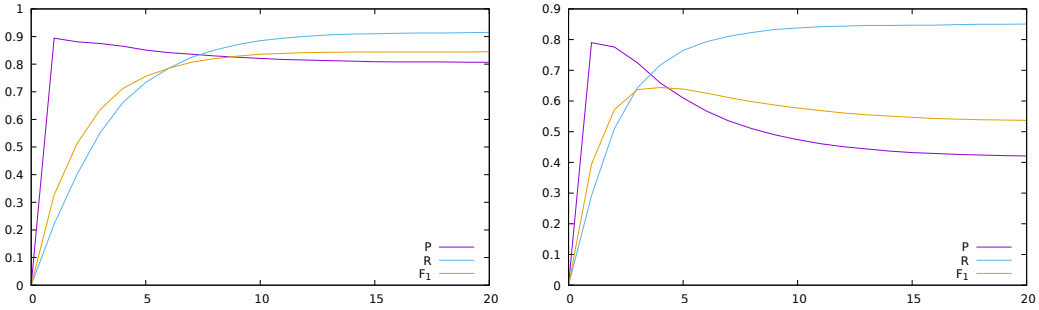
(a) metrics for keywords (b) metrics for keyphrases

Figure A.2.: single-word summarizer with selection method top r (for $r \in [0,1]$ and subsequence filter only)



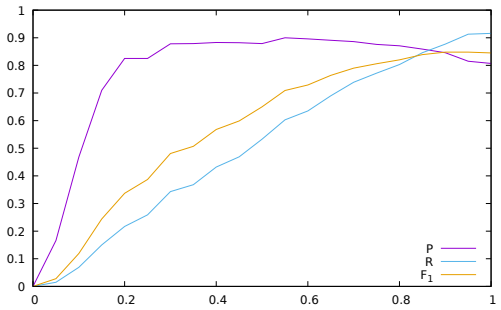
(a) metrics for keywords (b) metrics for keyphrases

Figure A.3.: single-word summarizer with selection method threshold t (for $t \in [0,1]$ and subsequence filter only)

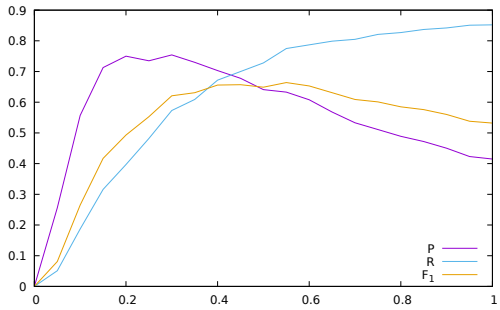


(a) metrics for keywords (b) metrics for keyphrases

Figure A.4.: n-gram summarizer with selection method top n (for $n \in [0,20]$ with subsequences, issues, successors candidates filtered)

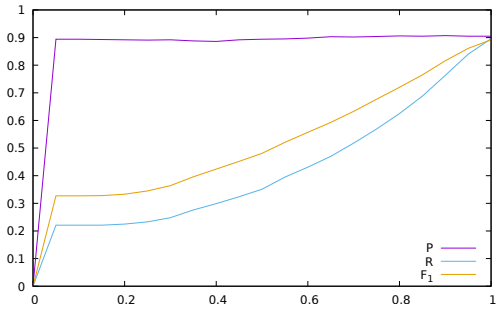


(a) metrics for keywords

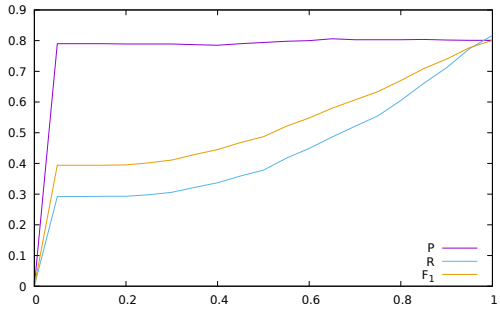


(b) metrics for keyphrases

Figure A.5.: n-gram summarizer with selection method top r (for $r \in [0,1]$ with subsequences, issues, successors candidates filtered)



(a) metrics for keywords



(b) metrics for keyphrases

Figure A.6.: n-gram summarizer with selection method threshold t (for $t \in [0,1]$ with subsequences, issues, successors candidates filtered)

Bibliography

[Bel14] Slobodan Beliga. “Keyword extraction : a review of methods and approaches”. In: 2014.

[Bou09] G. Bouma. “Normalized (pointwise) mutual information in collocation extraction”. In: *From Form to Meaning: Processing Texts Automatically, Proceedings of the Biennial GSCL Conference 2009*. Vol. Normalized. Tübingen, 2009, pp. 31–40.

[BP98] S. Brin and L. Page. “The Anatomy of a Large-Scale Hypertextual Web Search Engine”. In: *Seventh International World-Wide Web Conference (WWW 1998)*. 1998. URL: http://ilpubs.stanford.edu:8090/361/.

[Ebb19] Björn Ebbinghaus. “Decision Making with Argumentation Graphs”. Master’s Thesis. Department of Computer Science, Heinrich-Heine-University Düsseldorf, 2019. DOI: 10.13140/RG.2.2.12515.09760.

[HK03] Yaakov HaCohen-Kerner. “Automatic Extraction of Keywords from Abstracts”. In: *Knowledge-Based Intelligent Information and Engineering Systems*. Ed. by Vasile Palade, Robert J. Howlett, and Lakhmi Jain. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 843–849. ISBN: 978-3-540-45224-9. DOI: 10.1007/978-3-540-45224-9_112.

[Hul03] Anette Hulth. “Improved Automatic Keyword Extraction Given More Linguistic Knowledge”. In: *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*. EMNLP ’03. USA: Association for Computational Linguistics, June 2003, 216–223. DOI: 10.3115/1119355.1119383.

[Jon72] Karen Spärck Jones. “A statistical interpretation of term specificity and its application in retrieval”. In: *Journal of Documentation* 28.1 (1972), pp. 11–21. DOI: 10.1108/eb026526.

[KMM17] Tobias Krauthoff, Christian Meter, and Martin Mauve. “Dialog-Based Online Argumentation: Findings from a Field Experiment”. In: *Proceedings of the 1st Workshop on Advances in Argumentation in Artificial Intelligence*. Bari, 2017, pp. 85–99.

[Kra+16] Tobias Krauthoff et al. “Dialog-Based Online Argumentation”. In: *Computational Models of Argument*. Potsdam, Sept. 2016, pp. 33–40.

[Kra+18] Tobias Krauthoff et al. “D-BAS – A Dialog-Based Online Argumentation System”. In: *Computational Models of Argument*. Warsaw, Sept. 2018, pp. 325–336. DOI: 10.3233/978-1-61499-906-5-325.

[LC15] Matthias Liebeck and Stefan Conrad. “IWNLP: Inverse Wiktionary for Natural Language Processing”. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Association for Computational Linguistics, 2015, pp. 414–418. URL: http://www.aclweb.org/anthology/P15-2068.

[LF03] Christopher Lueg and Danyel Fisher. *From Usenet to CoWebs: Interacting with Social Information Spaces*. Jan. 2003. ISBN: 978-1-85233-532-8. DOI: 10.1007/978-1-4471-0057-7.

[Lit+11] Marina Litvak et al. “DegExt — A Language-Independent Graph-Based Keyphrase Extractor”. In: *Advances in Intelligent Web Mastering – 3*. Ed. by Elena Mugellini et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 121–130. DOI: 10.1007/978-3-642-18029-3_13.

[Luh58] Hans Peter Luhn. “The Automatic Creation of Literature Abstracts”. In: *IBM J. Res. Dev.* 2.2 (Apr. 1958), 159–165. DOI: 10.1147/rd.22.0159.

[Luh60] Hans Peter Luhn. “Keyword-in-Context Index for Technical Literature (KWIC index)”. In: *American Documentation* 11.4 (1960), pp. 288–295. DOI: 10.1002/asi.5090110403.

[Mac04] Ann Macintosh. “Characterizing E-Participation in Policy-Making”. In: vol. 2004. USA: IEEE Computer Society, Jan. 2004. DOI: 10.1109/HICSS.2004.1265300.

[MKM17] Christian Meter, Tobias Krauthoff, and Martin Mauve. “discuss: Embedding Dialog-Based Discussions into Websites”. In: *Learning and Collaboration Technologies. Technology in Education*. Vancouver, Canada, 2017, pp. 449–460.

[MT04] Rada Mihalcea and Paul Tarau. "TextRank: Bringing Order into Text". In: *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 404–411. URL: https://www.aclweb.org/anthology/W04-3252.

[Por80] Martin F Porter. "An algorithm for suffix stripping". In: *Program* 14.3 (1980), pp. 130–137.

[Ros+10] Stuart Rose et al. "Automatic Keyword Extraction from Individual Documents". In: Mar. 2010, pp. 1 –20. DOI: 10.1002/9780470689646.ch1.

[Spa+14] Paolo Spada et al. "A First Step toward Scaling-up Deliberation: Optimizing Large Group E-Deliberation using Argument Maps". In: (2014). DOI: 10.13140/RG.2.1.3863.5688.

[Tim12] Mika Timonen. "Categorization of Very Short Documents". In: *KDIR 2012 - Proceedings of the International Conference on Knowledge Discovery and Information Retrieval* (Jan. 2012).

[Tim+12] Mika Timonen et al. "Informativeness-based keyword extraction from short documents". In: *Proceedings of the 4th International Conference on Knowledge Discovery and Information Retrieval*. 2012, pp. 411–421. DOI: 10.5220/0004130704110421.

[Tur97] Peter D. Turney. "Extraction of Keyphrases from Text: Evaluation of Four Algorithms". In: (1997). DOI: 10.4224/5765105.

[Tur99] Peter D. Turney. "Learning to Extract Keyphrases from Text". In: *CoRR* cs.LG/0212013 (1999). URL: https://arxiv.org/abs/cs/0212013.

[Wit+99] Ian H. Witten et al. "KEA: Practical Automatic Keyphrase Extraction". In: *Proceedings of the Fourth ACM Conference on Digital Libraries*. Berkeley, California, USA: Association for Computing Machinery, Aug. 1999, pp. 254–255. DOI: 10.1145/313238.313437.

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, 31. März 2020 Jan Julius Jessewitsch