UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Faculty for Computer Science, Electrical Engineering and Mathematics
Department of Computer Science
Research Group Computational Social Science

Master's Thesis

Submitted to the Computational Social Science Research Group
in Partial Fullfilment of the Requirements for the Degree of

Master of Science

Stance Classification in Argument Search

by
Philipp Heinisch

Thesis Supervisor:
Jun.-Prof. Henning Wachsmuth

Paderborn, July 19, 2019

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen worden ist. Alle Ausführungen, die wörtlich oder sinngemäß übernommen worden sind, sind als solche gekennzeichnet.

Ort, Datum	Unterschrift

Danksagung

Ich danke allen, die mich bei meiner Arbeit beigestanden haben. Zuerst und vor allen Dingen möchte ich Jun. Prof. Dr. Henning Wachsmuth danken, der meine Masterarbeit betreute und sich freundschaftlich bei jedem unserer Treffen ausführlich Zeit genommen hat. Ich danke herzlich für seine Anregungen, Nachfragen und Begleitung meiner Denkprozesse und für etliche Paper, die er mir zur Verfügung stellte. Somit konnte zuerst ein Verständnis für die Thematik und Frameworks entstehen und darauf aufbauend viele Ideen. Ich blicke sehr positiv auf die Masterarbeitszeit mit Henning Wachsmuth zurück und freue mich, so viel dazugelernt zu haben.
Ich möchte mich jedoch auch bei den drei Personen bedanken, die Zeit investierten um den Auszug aus den Datensatz von `args.me` zu annotieren[1]: Shahbaz Syed (Promovierender an der Universität Leipzig), Milad Alshomary (Promovierender an der Universität Paderborn) und meiner Verlobten Anna Xenia Stephan. Allen drei danke ich für ihren Beitrag zur Forschung. Dadurch wurden zentrale Evaluierungsaussagen dieser Masterarbeit erst möglich gemacht. Besonders hervorheben möchte ich Anna Xenia Stephan für all ihre weitere Unterstützung, um mir einen guten Forschungs- und Schreibprozess zu ermöglichen.
Hinzu kamen noch viele hilfreiche Feedbacks von Henning Wachsmuth und weiteren Personen, durch die ich diese Arbeit weiter optimieren konnte.
Letztlich möchte ich auch all meinen Freunden inklusive einer studentischen Initiative, der SMD-Paderborn, danken, die mir viel Mut zugesprochen und für mich gebetet haben.
Insgesamt bin ich dankbar für die Zeit der Masterarbeit und freue mich auf meine weitere akademische Laufbahn.

[1]Abschnitt 6.1.1 berichtet über diesen Auszug

Abstract.
Stance classification is an essential step in computational argumentation and the argument search. For example, an argument search engine should present arguments on the correct side – *PRO* or *CON*. This classification is a difficult problem because it must deal with the diversity in natural language in an open domain context. Also, the stance prediction must be quickly executable on demand. There exist already promising approaches in stance classification. Our approach is based on a modular approach which divides the problem into three subtasks: the target identification, the sentiment classification towards the targets and the contrast classification of the topic towards the claim which the stance classifier processes. This thesis answers whether this idea is an applicable approach for stance classification in the argument search, too. This thesis discusses and evaluates adaptations to reach sound effectiveness and efficiency. Our approach outperforms a heuristic which an argument search engine uses. Hence, this thesis contributes insights to improve the stance classification in argument search and suggests further ideas to improve on the classification like using contextual features.

Contents

1 Introduction

There is overwhelming user-generated content on the Internet in these modern days. Millions of people share their opinions, likes, rates and have discussions in social networks like Facebook and Twitter. This behaviour results in a significant possibility to extract knowledge from the Internet.
Hence, we are living in a remarkable time where we have a massive amount of data freely available. A considerable part of this data is unstructured and in natural language. Hence, it is initially not possible for a computer to understand and structure this data. Humans can do this – but there is too much data to process it all manually. Hence, an automatism would be helpful to extract knowledge and make the data more usable. This automatism is called natural language processing based on machine learning. The massive amount of data including preprocessed datasets and the computational power gives us the possibility to retrieve more and more information mechanically.
One exciting part of this data contains arguments – often written by humans in natural language. Persons need arguments in political discussions, finances, science, decisions, in daily life and more areas. Arguments help people to evaluate things or to convince somebody. The extracted knowledge can support, for example, debates, which is necessary to build persuasive arguments. During the upraising research area of natural language processing, machines can help with good *PRO* and *CON* arguments (Bar-Haim et al., 2017a). Already now, it is possible that the computer is supporting a human in debating or making decisions. One practical example is the IBM Debater® in the debating support (Bar-Haim et al., 2017a). The company has run this project to provide artificial intelligence that will be able to argue with a human about a random complex topic like climate policy.
Until now, the general Internet user does not notice much about computational argumentation. Still, the use cases of computational argumentation for daily life are numerous: automated decision making, opinion summarization, deep opinion mining and even on-demand argument construction (Stede and Schneider, 2018). However, standard search engines like Google do not offer that. However, "Computational argumentation is expected to play a critical role in the future of web search." (Wachsmuth et al., 2017) Let us consider a computational argumentation search engine like `http://www.args.me/`. In contrast to other conventional search engines which are indexing web pages, it indexes and retrieves arguments (Wachsmuth et al., 2017). In the search on arguments against a search query, which is the specified topic, it is necessary to know whether an argument is for or against the topic. Consider the significant question: "Does God exist?". The related search query can be *the existence of God*. If the user of the search engine wants to debate *for* God's existence, he is primarily interested in the *PRO* arguments. In the

perspective of an atheist, he would probably be primarily interested in the *CON* arguments.
The stance of an argumentative text is the overall position toward an idea, object or proposition (Somasundaran and Wiebe, 2010). It is specified by the standpoint which the author argues to be true. An argument is a rational construct which justifies a standpoint. An argument contains one or more propositions and the *claim*, the standpoint. The propositions are related to the claim and support it (Stede and Schneider, 2018). Often there is a relation to the sentiment of a text or the polarity. An example (*PRO*) for the sentiment correlation is: "Hallelujah, I experienced that God exists and I am happy about it!" However, a stance can be expressed without polarity, too. In the topic of the existence of God, a *PRO* example would be: "Kurt Gödel introduced the ontological proof of God's existence."
Stance classification is, given a topic (for example a search query) and a set of claims, to determine which of these claims argues for and which against the topic – so sorting them into the *PRO* class and the *CON* class (Bar-Haim et al., 2017a). The task is to do that automatically. Stance classification does *not* measure the relevance of a claim regarding the topic. However, stance classification is an essential and required step in several natural language processing pipelines which are related to the field of computational argumentation.

Automatic stance classification is not trivial – there are several challenges. First, stance classification is based on natural language. It is difficult for a computer to deal with natural language. The use of language is not consistent but depends on the general topic. Words with ambiguous meanings exist like "letter". A second challenge with natural language is the implicit point, for example the existence of enthymemes (Rajendran et al., 2016). It is like an iceberg (Wojatzki and Zesch, 2016): the written text shows only the small part above the water, but the major part is below that. Often people presuppose lots of background knowledge and do not express such facts or relations. The problem becomes apparent in argumentation: Consider the text: "The cosmological argument implies God's existence". This sentence assumes knowledge of the cosmological argument. Alternatively, suppose the argument: "The pain in the world implies the non-existence of an almighty, loving God" contains the hidden claim (therefore an enthymeme) that almighty love is determined by always preventing pain. The research area of argument mining treats implicit argumentation (Wojatzki and Zesch, 2016). The determination of implicit argumentation is no necessary requirement for stance classification, but enthymemes can increase the difficulty of the classification. Another critical point, especially in argumentation, is the different usage of natural language. The language differs in the theme of the debate, and for example, the date of the entry in political discussions influences the meaning (Vilares and He, 2017). It is a big difference whether the debate is in public, face to face, or (anonymous) on the Internet. Because of the lack of generalisation, different research papers exist in the area of computational argumentation like for discussion on forums (Hasan and Ng, 2013) or Twitter (Addawood et al., 2017). Research has shown that the "debates in public forums differ from congressional debates and company-internal discussions in *terms of language* use" (Hasan and Ng, 2013). Besides that, even in a similar case like a specific forum, the research has shown that the writing style differs from person to person (Khan et al., 2016). So, it is hard or even impossible to train the one general classifier which always performs well – independently of the domain (Vilares and He, 2017).
However, there are more problems and challenges in stance classification like negation. A single word like "not" does not invert necessarily the whole context/stance like in logics. Besides, argumentation can contain sarcastic and comparative sentences and more (Khan et al., 2016). Moreover, there are even more challenges like the problem with dual meaning words and incomplete (opinion) lexicons (Khan et al., 2016).

The challenges show the difficulty of the stance classification task. Hence, lots of research takes place, and still, it is not finished yet. If there is stance information from debate forums where a crawler collects claims a simple heuristic can be applied (Wachsmuth et al., 2017). However, if such information is missing, obviously, machine learning is required to come up with satisfying results. Hasan and Ng (2013) pointed out that unigrams (the occurrences of single words) as features is a relatively strong baseline. For spontaneous speech, unigrams are even the best choice. Other approaches take sentiment and arguing features into account, sometimes also with unigrams. Somasundaran and Wiebe (2010) trained a classifier with features like aspect-based polarity, discourse relations, subjectivity and arguing lexicons. Ranade et al. (2013) exploited other texts of the same author to predict the stance. Faulkner (2014) strongly considered linguistic research with linguistic subtleties. Hasan and Ng (2013) combined N-gram features (the occurrences of n consecutive words), syntactic features, semantic features, Internet query search and author constraints to large feature vectors. Finally, Somasundaran and Wiebe (2009) converted the stance classification problem in an integer linear programming problem.
All these approaches suffer, e.g. in the drawback of topic-specific features (Faulkner, 2014) or the requirement of many posts from the same user (Ranade et al., 2013). Sometimes the existing approaches are very limited to a specific domain. Moreover, all these approaches are not totally reliable: they accuracy is between 61% and 82%. Only the approach of Somasundaran and Wiebe (2009) claimed accuracy of 100%, but the authors admit that they used only a small test set.
Stance classification in argument search is to classify the stance of a (crawled) text snipped regarding a search query on-demand. This new problem requires several critical points like a high value of generalisation. The stance predictor can neither assume anything about the search query nor about the argumentative texts (claims), but they are related to the topic of the search query. A user should request any topic. Hence, the predictor must deal with unknown topics – in any domain. The algorithm cannot expect lots of contextual information like previous approaches do. Furthermore, if the algorithm wants to classify a claim, probably only a short sentence, an extended text part for extensive annotations is missing. Also, to avoid dissatisfaction, the predictor should have high accuracy and rapid computation time for each claim-topic-pair. Caching already requested results is a nice feature, but not a general solution for this problem.
IN 2017, IBM presented a novel approach of a stance classification which faced issues about natural language without lots of contextual information and generalisation in argument search (Bar-Haim et al., 2017a). The novel idea is to predict the stance not directly but with a modular approach (Bar-Haim et al., 2017a). It divides the classification into the task of Claim Target Identification, Claim Sentiment Classification and Contrast Classification. The step of the claim target identification tags the target phrase of the claim. The claim sentiment classification determines whether a claim has a positive or negative polarity towards its target. Contrast Classification answers the question of whether a target of a claim is in contrast to the topic or search query or not (Bar-Haim et al., 2017a). For example, if we consider the claim “Believing that God exists is just stupid” and the topic “God does not exist”, then the claim supports the topic of the search query, even if the sentiment regarding the target of the claim is negative. However, because of the contrasting targets “God exists” and “God does not exist”, the contrary claim converts to a *PRO* argument. This was an easy task for the modular approach, but a non-modular approach would probably fail.
The problem with this approach is that IBM did not develop it for the case of argument search. IBM calculates the stance of an (often complex) claim regarding another claim, the topic. This paper will deal with the problem of stance classification in argument search and will apply the result in the argumentation search engine `args.me`. The algorithm must calculate on-demand

the stance of a claim regarding a search query in the case of this search engine. However, claims often contain only one word (Ajjour et al., 2019). The claim is the conclusion of a relevant argument from a debating portal in natural language (Wachsmuth et al., 2017). The arguments were written by many different users in many different contexts. Hence, this paper will adapt this approach and will show which improvements are possible with the modular approach in comparison to a simple heuristic which is currently used in `args.me` (Wachsmuth et al., 2017). For the adaptation, the paper will present possible steps regarding the preprocessing of the search query regarding the text snippets which should be classified. A further research task is to find out and describe which process steps are computable on-demand and show up possible alternatives. The paper will present further improvements and improvements which IBM suggested like the automatic expansion of the underlying initial sentiment lexicon (Bar-Haim et al., 2017b).
A further problem with the approach of IBM is the commercial attitude – it is a commercial application (Wachsmuth et al., 2017), so the source code or implementation details are not public. In return, this thesis will present an adapted public implementation of the IBM-Approach (Bar-Haim et al., 2017a). Additionally, this thesis will present adaptations of the approach for the case of argument search by trying and verifying different parameters and new features with different machine learning models.
To avoid a lousy generalisation in this open-domain task and to present comparisons with the IBM approach (Bar-Haim et al., 2017b), the new algorithm uses for training and testing the argumentation corpus by IBM. The training, validation and test set combined contains 55 completely different controversial topics. Each topic has a large set of claims. The 2,394 claims were crawled from Wikipedia articles, so this thesis deals with various writing styles from different users (Bar-Haim et al., 2017a).

We tested our adapted implementation additionally on a sample of an argument search engine. Our modular approach outperforms a simple heuristic which sets the stance of an argument towards the topic to the *PRO*/*CON*-flag of argument's premises towards the conclusion. Also, we evaluate various text analysis pipelines and configurations of our approach to present a solution which succeeded with satisfying accuracy and runtime in the argument search.

This thesis contains Chapter 2 of foundations firstly to define and introduce concepts which are used. The same chapter contains an extensive view of the related work. Chapter 3 describes the core approach of IBM in detail and asks about the adaptability for argument search. We will notice that some adaptation steps are mandatory. Some other steps will improve accuracy. Chapter 4 discusses the mandatory and optional extensions of the adaptation. The thesis will pin down the concepts to an overview of the written code and machine learning classifiers and regressors in the following chapter. Also, the chapter provides an insight into the user interface. The evaluation takes place in Chapter 6. It describes the experimental setup and the results. Additionally, the chapter analyses the results and compares them with other approaches like the baseline approach of unigrams and approaches based on machine learning. At the end, this thesis contains the conclusion chapter (Chapter 7) and will give an outlook and possible future work.

2 Foundations and related work

2.1 Foundations and definitions

This section will give an overview of the foundations and definition. We make use of the base concept of machine learning (Section 2.1.1). This section explains the used learning models and why machine learning is feasible. Feasibility is the underlying assumption of the whole work. Section 2.1.1 gives an introduction to validation and testing, too.
Because stance classification works with natural language, Section 2.1.2 introduces natural language processing. It describes the central steps and presents a brief overview of the syntax of natural language.
The last part (Section 2.1.3) of this chapter defines some words which the paper will use later.

2.1.1 The machine learning approach

Machine learning is a strategy to learn by machine from data (Abu-Mostafa et al., 2017). For example, a child learns from many pictures (data) what a cat is. Hence, if we show an animal to an adult, he can classify very precisely whether the animal is a cat or not. That is an easy task, but no adult can describe in a formal analytic way or with a mathematical formula how he distinguishes a cat from any other animal.
To come back to machines, machine learning can apply sensibly wherever we do not have an analytic solution but lots of data with enable us to construct an empirical solution (Abu-Mostafa et al., 2017). In the optimal case, the machine learning algorithm finds a pattern in the data itself and predicts the right output for every instance. In other words, in the optimal case the machine learning algorithm finds the best fitting function to the problem in a given hypotheses set. A *hypotheses set* is a set of functions (mostly infinite) which are applicable for the problem (Abu-Mostafa et al., 2017). A more detailed description of a machine learning setup is shown in Figure 2.1. So, in general, machine learning contains two components: the training and the validation/ testing. If a machine learning algorithm is trained, we can use it for predicting the outputs for new instances. It is uncommon to train an algorithm which always predicts correctly for all new instances. Nevertheless, the research explores statistical bounds like Hoeffding inequality. This bounds show, that, if an algorithm performs well in a large independently generated training data set, then it will probably perform well for new instances. Learning is feasible (Abu-Mostafa et al., 2017).

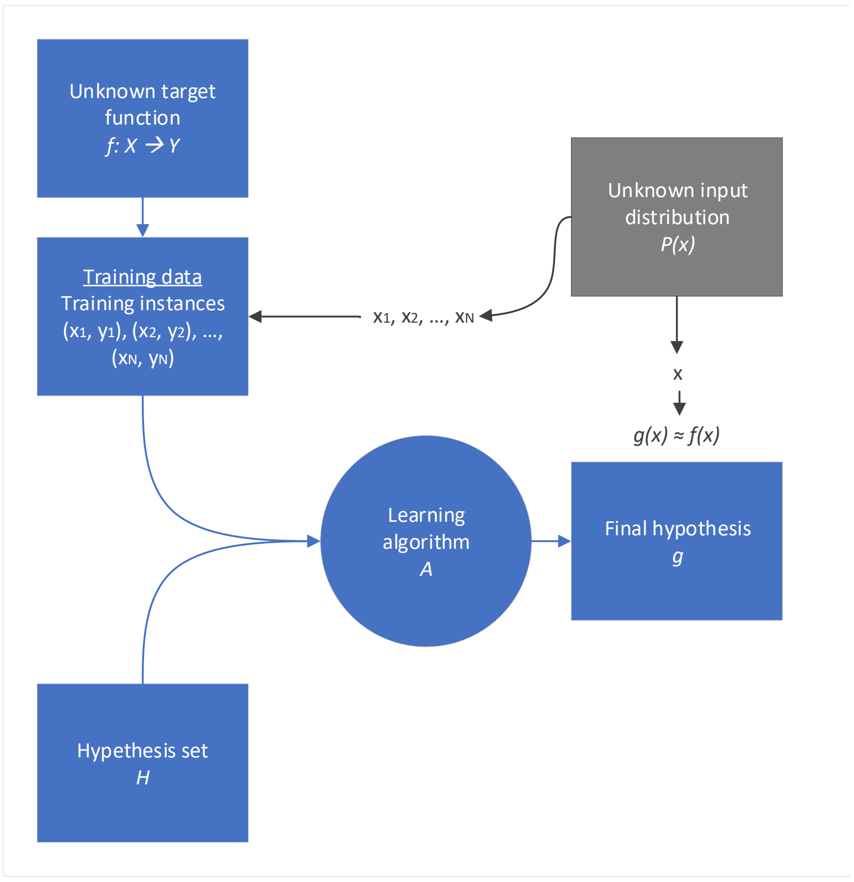


Figure 2.1: The figure models the basic setup of machine learning by Abu-Mostafa et al. (2017). Firstly we have an unknown target function $f : \mathcal{X} \rightarrow \mathcal{Y}$ which we want to approximate. The function maps each instance $\mathbf{x}$ to its right output y (fuzzy outputs are possible, too). Furthermore, we have training instances – specific instances of unknown input distribution. Additionally, we can define a hypotheses set $\mathcal{H}$ which contains hopefully f or at least a good approximation of f. The learning algorithm $\mathcal{A}$ is a user-defined algorithm which selects from the hypotheses set with the help of the training instances in a sometimes very complex process the best fitting hypothesis. This learning algorithm returns the hypothesis function g. Function g can be used to predict the output-values (y) for new instances.

There are several learning strategies. Each strategy has its own requirements to the training data. The machine learning algorithm is trained with the training data. The *supervised* learning delivers the learning algorithm the training instances, including the right output. For example, if the problem is a classification problem (the output should be a class), each instance is labelled with the right class (Khan et al., 2016). This thesis deals only with supervised learning like in Figure 2.1, but there are other learning strategies in use.
A second strategy is the *semi-supervised* learning. In this strategy, the algorithm does not get the right output directly, but indirectly by manual tuning by domain experts (Khan et al., 2016).
A third option is the *unsupervised* learning. In this case, the machine learning algorithm gets no clue what the right output should be. This strategy is useful for making use of statistical analyses (building clusters) on a large volume of data. Hence, this strategy does not require training (Khan et al., 2016). One practical example is a machine, which distinguishes coins, where the producer does not know the values of each coin batch.

Data for training or predicting outputs contain many *instances*, known as instance set or data set $\mathcal{X}$ (Frank et al., 2016). These instances $\mathbf{x}$ are called data points, too. One main challenge is to make an instance understandable for a computer which is basically a calculator. For example, the natural language expression "Sun is shining." and "The sun shines." looks similar to humans. However, both expressions are for a computer just two unequal `String` objects which are encoded in bytes. Hence, each instance needs a proper representation, called *features* or *attributes* (Frank et al., 2016). In practice, most machine learning algorithms are based on vector space models (Wachsmuth, 2015). Each number (value) of a vector represents a feature. A bundle of features are called *feature types*, if they belong together conceptually or are considered in combination (Wachsmuth, 2015). An example of a feature type in case of a string is the relative frequency of each letter. All features in combination (the vector) are called a *feature set* (Khan et al., 2016). It is crucial for machine learning that each feature set of $\mathcal{X}$ has the same number of features, so the same vector length, also called dimensionality. This property results in the *feature space* (called instance space, too).

There is the possibility to edit the feature space and the instance representations. One opportunity is to transform the feature sets. It is possible to apply *feature transformation* functions $\Phi : \mathbf{x} \mapsto \phi(\mathbf{x})$ (Abu-Mostafa et al., 2017). This transformation maps the instances to a new feature space $\mathcal{Z}$ which can have another dimensionality than the original $\mathcal{X}$ space. Transformations can help to combine or extend single features to a relevant concept. This can help to solve the problem of predicting wrong outputs. Even it can help to make problems with a high complex output behaviour solvable for a limited hypothesis set in a satisfying manner (Abu-Mostafa et al., 2017). It is possible to apply a feature transformation before the machine learning algorithm. Sometimes it is a fix component of a learning algorithm. In such a case the transformation is often done by a *kernel* like in supported vector machines (Meyer, 2018). A kernel is a function $K(\mathbf{x}_1, \mathbf{x}_2) \mapsto z, z \in \mathbb{R}$ which often measures the similarity of two instances.
A effective and common case of transformation before or during applying machine learning is the *feature selection*. This step filters the features to reduce dimensionality and get rid of disturbing features (Frank et al., 2016). The goal is to have only relevant features at the end of this step which results in a better generalisation. Another reason for feature selection is the reduction of computation time in general. One practical step to determine the usefulness of each feature type

is to train and test a machine learning algorithm isolatelly with each feature type, then with all feature types except one feature type – for each feature type. In the end, the training and testing uses all feature types. The last step is the comparison of all results (Blum and Langley, 1996).
Figure 2.2 shows a feature transformation.

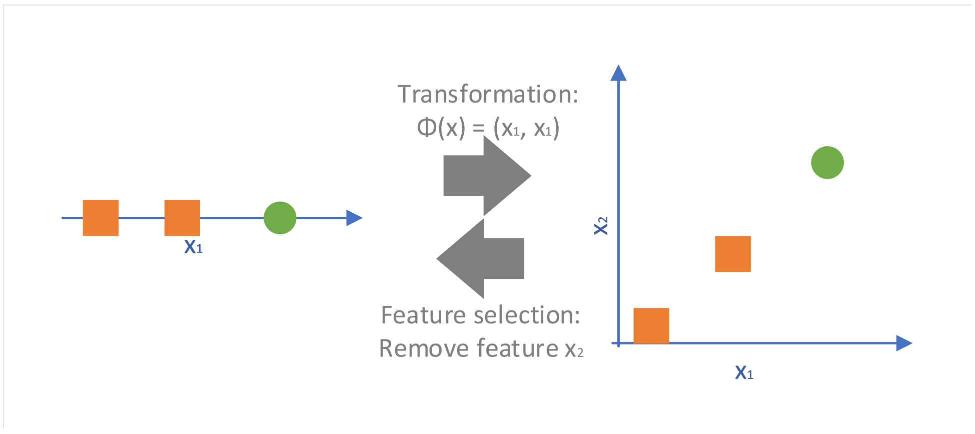


Figure 2.2: This figure is a symbolization of transforming the feature space. The squares are negative instances (means an output $y = -1$) and the circle is a positive one ($y = +1$). The transformation function $\phi(\mathbf{x}) = (\mathbf{x}_1, \mathbf{x}_1)$ (where $\mathbf{x}$ is an instance and $\mathbf{x}_1$ the only one feature) adds a dimension. The result is a two-dimensional feature space. It is obvious that the feature $\mathbf{x}_2$ is redundant. An applied filter would remove for example feature $\mathbf{x}_2$ in the feature selection step. The result is again the original one-dimensional feature space.

Learning models

A *learning model* is the combination of the hypotheses set and the learning algorithm (Abu-Mostafa et al., 2017). Each learning model has a defined set of hypotheses and provides a procedure to select a hypothesis function from the set. Hence, the returned hypothesis function will differ from different learning models. Lots of learning models exist – the procedure to find the best fitting one for a problem is called model selection (Thornton et al., 2013).

There are two types of learning models. One type is the *classifier* type. Classifiers predict classes. These learning models learn from instances with a predefined set of output classes and will label a new instance with a class of this set. A common base case is *binary classifiers* (Abu-Mostafa et al., 2017). Binary classifiers classify their instances into negative and positive ($\mathcal{Y} = \{-1, +1\}$).
Another type is the *regression* type. Regressors predict continuous values (real-valued outputs). One use case is to predict the value of an object. A popular special case of regression is the *probability estimation*. The learner does not predict a class in the probability estimation, but how a confidence score which describes how sure the learner is that an instance belongs to a certain class. In probability estimation often the $\mathcal{Y} = [0, 1]$ (Abu-Mostafa et al., 2017).

A learning model can have hyperparameters. A hyperparameter is a parameter whose value the user must set before the learning phase. These parameters change the way how the learning algorithm works. A hyperparameter can change the hypothesis set or can determine how the learning algorithm should treat or transform the input data. Hyperparameters can influence the selection procedure of a hypothesis, too. The learning model defines its hyperparameters individually by itself. Because of the changing behaviour of a learning model with different hyperparameters, it is essential to choose beneficial hyperparameters for the problem. Although it is often not trivial which hyperparameters are beneficial it is useful to try and test different hyperparameters[1]. This procedure is known as *hyperparameter optimization* or hyperparameter tuning (Thornton et al., 2013).

Also, metamethods extend the variance of learning models. A *metamethod* takes one base learning model and additional parameters as input. For example, such parameters pre-process the instances or aggregates predictions from different trained models of the same type. Another opportunity is to *ensemble* learning models and aggregates the predictions afterwards (Thornton et al., 2013).

The linear model and linear regression A very common learning model class is the linear model. It is based on vector space models. Let $\mathbf{x}$ be an instance. Hence, it is a vector, suppose the dimensionality is d. Furthermore, $\mathbf{w}$ is a given weight vector of dimensionality $d+1$. The weight vector can be a result from a learning pre-process (training). Additionally, Θ is a self defined link function. The general formula is given in Equation 2.1 (Abu-Mostafa et al., 2017).

$$h(\mathbf{x}) = \Theta\left(\mathbf{w}_0 + \sum_{i=1}^{d} \mathbf{w}_i \cdot \mathbf{x}_i\right) \tag{2.1}$$

w_0 is the bias term, because it is independent of the instance (Abu-Mostafa et al., 2017).
For binary classification Θ in Equation 2.1 is the sign function: $\Theta(s) = sign(s)$. For regression, Θ can be the identity function $\Theta(s) = s$. And for probability estimation, Θ can be the logistic function (2.2).

$$\Theta(s) = \frac{e^s}{1+e^s} \tag{2.2}$$

The linear model is called linear, because the prediction depends linearly on $\mathbf{x}$. A linear hyperplane (depends on $\mathbf{w}$) divides the positive and negative class in the case of binary classification. Figure 2.3 shows this.

Support Vector Machines (SVM) are one of the most essential classifiers (regressors) (Meyer, 2018) of a linear model. SVM base on the idea to maximize the margin of the hyperplane – the best possible separation among the classes. That means to find a hyperplane which has the largest distance to those instances which are closest to instances of another other class (2.3) (Witten et al., 2017).

$$\max_{\mathbf{w}} \left(\frac{1}{||\mathbf{w}||} \min_{((\mathbf{x},y)|\mathbf{x}\in\mathcal{X},y\in\mathcal{Y})} \left(y\left(\mathbf{w}_0 + \sum_{i=1}^{d} \mathbf{w}_i\mathbf{x}_i\right)\right)\right) \tag{2.3}$$

Instances with the smallest distance to the hyperplane are called support vectors. Only those instances determine the location of the hyperplane – figure 2.4 shows that fact.

[1] this can be done by trying manually, for example with the grid search or by tools like Auto-Weka (Thornton et al., 2013)

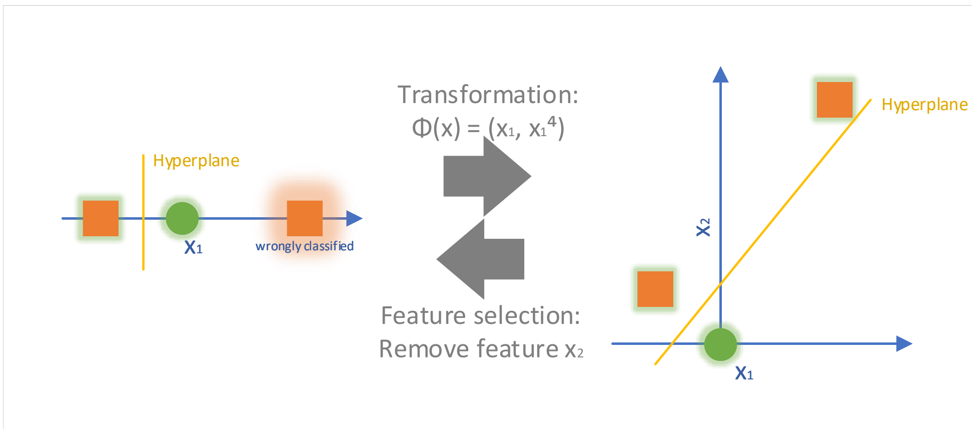


Figure 2.3: This figure represents the linear model in the case of binary classification. Not all instance sets are linearly separable like in the left diagram. There is no possibility to set the line such that all squares are on one side and all circles on the other side. The feature transformation can help to convert the instance set into a linearly separable one. The transformation allows "bending" the line in the plot before the transformation (left).

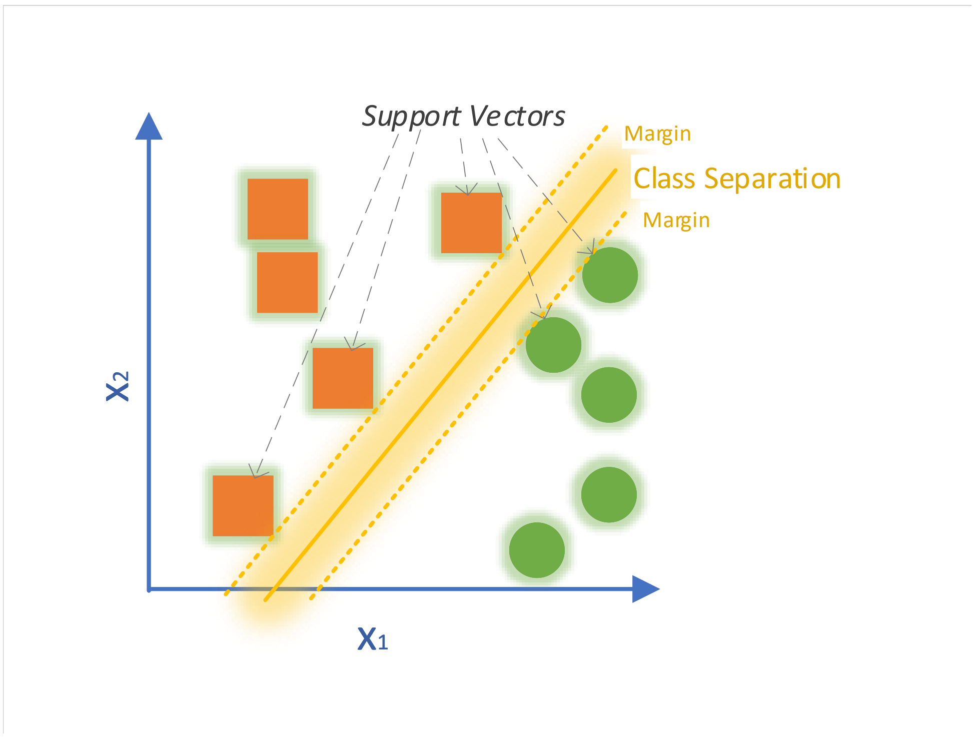


Figure 2.4: This figure shows an SVM in its instance space. Squares are instances of the first class, circles or instances of the second class. The closest instances to the hyperplane (support vectors) are at the border of the margin.

Additionally, it is possible to choose the soft margin approach. In this approach, it is possible to have instances inside the margin. Instances which violate the margin (inside the margin or wrongly labelled) are considered in a penalization term. The hyperparameter c controls the weight of the penalization term.
On the one hand, SVM consumes many resources at the training stage (especially the hyperparameter tuning) (Meyer, 2018). On the other hand, SVM have linear complexity and are efficient with a large feature set. Additionally, the user can use his kernel function in SVM. However, the practise shows that this tends to be domain specific (Witten et al., 2017).

Selected classifiers There are several further classifiers, some of them are presented in this paragraph.

Majority voting predicts that class which occurs most often in the training instance set. The prediction is independent of the instance itself, which should be classified. Obviously, this approach is trivial and has, in general, no use except as a baseline.

K-nearest neighbour (Khan et al., 2016) (k-NN) is a lazy learning model. That means that there is no computation at all during the training step except to store the instances in a set. The computation takes place in the step where the class of a new instance should be predicted. Firstly, the k nearest instances, called neighbours, are selected. The determination requires a distance function which is a hyperparameter like the k, too. An aggregation function aggregates the known classes of the neighbours in the next step. The aggregation function is a hyperparameter, too. The return value of the aggregation function is the prediction. Some implementations weight the influence of a neighbour with its distance. K-nearest neighbour delivers promising results, but it suffers from imbalanced instance sets. If a class is only rarely represented in the training instance set, the model will rarely or never predict this class, no matter which instance the input is. K-nearest neighbour can be used for regression problems, too.
A similar learning model is *K-Star* (Cleary and Trigg, 1995)

Decision Trees (Quinlan, 1986) are trees which contain a rule in every node except the leaf nodes. A leaf node contains a class prediction. A rule is a feature-based test with a branch for each possible outcome. An example is a threshold test like $x_1 > 0$. There are two outcomes (true or false), so two branches. A new instance starts with the first rule in the root, and according to the result, the instance is redirected to a specific node in the next level of the tree. The procedure runs until the instance reaches a leaf. Not every single feature is requested during a prediction process. It can happen that a decision tree never requests a certain feature after training. Moreover, a rule of thumb is to keep a decision tree simple – avoid many nodes. If two decision trees fit the training set with the same accuracy, the developer should take the simpler one. Decision trees are fast and very accurate. Features with a low impact are pruned out. This behaviour is conducive for natural language processing. However, the drawback is that only feature-value combinations (paths in the tree) decide over the prediction and not, for example, feature interactions directly.
Random forest (Liaw and Wiener, 2002) is a metamethod which aggregates decision trees. The algorithm trains many decision trees based on a randomly chosen feature subset (across feature types, too). During a prediction of the class of a new instance, every single decision tree calculates its prediction. The final prediction is the majority vote[2]. Random forest outperforms well, in

[2]Other aggregation methods can be used, too. Random Forest is available for regression problems, too (Thornton et al., 2013)

general, but it performs worse if some features are dominant. It is robust against overfitting. Overfitting means to be very good in the training data but worse in predicting non-training-data instances.

Evaluation and validation

The instance set is often only a small portion of all possible instances. Nevertheless, a single instance can be a very complex vector which has been computed over several steps. Machine learning, especially in combination with natural language processing, needs evaluation and validation like in nearly all disciplines in computer science.

Effectiveness gives information about the correctness of an output by an algorithm (evaluation) (Wachsmuth, 2015). In the case of binary classification like stance classification there are positive instances and negative instances. In this case all possible instances can be divided into four classes (Wachsmuth, 2015):

TP True positives: positives instances classified positive by the algorithm

TN True negatives: negative instances classified negative by the algorithm

FP False positives: negative instances classified positive by the algorithm

FN False negatives: positives instances classified negative by the algorithm

One measurement for effectiveness is the *accuracy* a. The accuracy is the ratio of correct decisions (Equation 2.4) (Wachsmuth, 2015).

$$a = \frac{|TP| + |TN|}{|TP| + |TN| + |FP| + |FN|} \tag{2.4}$$

Another measurement is the *precision* and *recall*. Precision p measures the ratio of output information which is inferred correctly. Recall r calculates the ratio of all correct information which is inferred (Wachsmuth, 2015). The requirements often ask for high precision and high recall. Therefore, the f_1-score defines the harmonic mean of precision and recall. An overview gives Equation 2.5.

$$\begin{aligned} p = & \tfrac{|TP|}{|TP|+|FP|} \\ r = & \tfrac{|TP|}{|TP|+|FN|} \\ f_1 = & \tfrac{2pr}{p+r} \end{aligned} \tag{2.5}$$

In classification with more than two classes the accuracy and f_1-measure can be used, too. The measurement computes for every class the scores in that case that the selected class is the first one and all other classes are combined in the second class. Now, there is the binary classification problem again. The scores can be aggregated, for example, by averaging.
An algorithm in natural language processing often contains one or more machine learning models. We define the error function $e(y, \hat{y})$ to measure the effectiveness of a learning model for a single instance $\mathbf{x}$. y is the ground truth output, $\hat{y}$ the predicted one. A common error function for regression problems is the squared error function (Equation 2.6) (Abu-Mostafa et al., 2017). The mean squared error defines the error for an instance set by averaging the squared error function over all instances. This error is called E_{in}.

$$e(y, \hat{y}) = (y - \hat{y})^2 \tag{2.6}$$

The problem is that the instance set is only a small portion of all possible instances. Often, the error inside the labelled instance set is different from the error outside this set (E_{out}).

One problem is *overfitting.* Overfitting occurs by producing a classifier or regressor, which fits the training data too tightly (Witten et al., 2017). In other words: overfitting fits the noise (difference of the label to the ground truth target function), too (Abu-Mostafa et al., 2017). A complex target function or noisy data increases the danger of overfitting. Many degrees of freedom for the machine learning algorithm increase the danger of overfitting, too. A huge number of instances decrease the difference between E_{in} and E_{out} (Abu-Mostafa et al., 2017). Hence, to avoid overfitting, the machine learner must gather lots of instances or – as an additional option – must regularize the machine learning algorithm, for example with a small c-hyperparameter in support vector machines.
Validation determines E_{in} and gives a clue about E_{out}. To present a reliable result, a machine learner should divide its instance set into the training set, validation set and testing set. The testing set measures the error (or accuracy / f_1-score of the whole algorithm) *after* training and freezing the learning model. The training set is to train a learning model. The validation set is to validate the learning model after training and can be used for a good hyperparameter estimation and additional training iterations. Therefore, the validation set influences indirectly the learning model (Abu-Mostafa et al., 2017).
A common method for giving a clue on E_{out} is the n-fold-cross-validation. Often, n equals ten. In the n-fold-cross-validation, the instance set is divided into n sets of equal size. Then, in n iterations, n learning models are trained with $n-1$ of these sets. The nth set estimates the error. In the end, every set is validated once. In the end, the method combines all sets for the last training step. The final error is the average of the n estimates (Abu-Mostafa et al., 2017).

Efficiency is a second important measurement. It is defined by the *run-time* (also called running time) of an algorithm to calculate its output (prediction) (Wachsmuth, 2015). The run-time in this thesis is always the average time to process a single claim.

2.1.2 Text analysis and natural language processing

The approach of this thesis presents different ways pf dealing with natural language. Hence, natural language, text/ strings, are not useful for computations for the computer. Humans can understand their meaning, but for computers, it is just a stream of characters. Hence, algorithms must analyse the text first. Often, the text analysis is organized in a pipeline: information extraction, text classification, and natural language processing (NLP) (Wachsmuth, 2015). We assume an already classified text. We assume arguments, separated in the conclusion (claim) and the remaining part of the argument. Hence, this thesis only takes a view to NLP. NLP analyses the input text, identifies and structures relevant information (Wachsmuth, 2015). NLP faces different problems in natural language, for example, the problem of ambiguity (Wachsmuth, 2015).
Often, NLP is organized in pipelines, too. In term of reusability, there are a bunch of algorithms (single executable steps) which analyses the text regarding a specific entity. Hence, for example, there are algorithms which detect sentences, algorithms which detect words or dates. Every algorithm (primitive analysis engine) has specific prerequisites and a specific result. An application engineer now has the possibility to combine primitive analysis engines in a pipeline to get an aggregate analysis engine. Figure 2.5 sketches an aggregate analysis engine. One popular framework which supports this approach is Apache UIMA[3] (Ferrucci and Lally, 2003).
The revealed information from analysis engines is stored in annotations. An annotation represents an instance of a particular type of information. An annotation covers the whole text or a specific span of text (Wachsmuth, 2015).

[3]`https://uima.apache.org/`

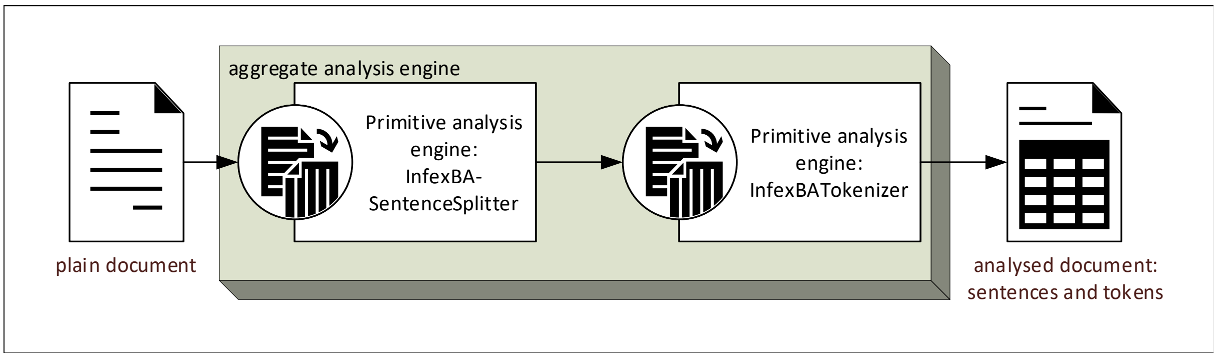

aggregate analysis engine
plain document
Primitive analysis engine: InfexBA-SentenceSplitter
Primitive analysis engine: InfexBATokenizer
analysed document: sentences and tokens

Figure 2.5: This figure shows an example of a pipeline in natural language processing. The pipeline also called aggregate analysis engine has in this example two primitive analysis engines[4]. One of them detects sentences. The other detects tokens which are mainly single words.

NLP and Machine Learning

Some NLP tasks follow strict rules. For example, the task of splitting a text into words is basically to split whenever a white space occurs. However, there are other NLP tasks which are more complicated. One example is to find out the sentiment of a sentence or a word. For some problems exist some well know and proper functions, but for other problems, a machine learning approach might be helpful. There is the possibility to use machine learning approaches in NLP. Hence, a primitive analysis engine can use a machine learning approach (Wachsmuth, 2015).
The creation of a new primitive analysis engine which is based on machine learning requires a text corpus. A text corpus is a collection of texts. The corpus should include the ground truth which an analysis engine should detect or predict. It is recommended to split the corpus into training and test data. After that, already existing analysis engines derive required annotations. This derivation is the part of text analysis. With the help of the annotations follows the next step: the feature engineering. The computed features are the input for a specified learning model. This model is saved after training and testing. The saved model can become a part of a new primitive analysis engine now. This analysis engine includes feature computation as well.

Word embeddings describe the technique to transfer words (or phrases) into a vector representation. Therefore, related models are called *Word2Vec*-models. The vector representation is an output of machine learning procedures and is helpful for machine learning which uses that vector representation. The aim of the training is to group similar words. The vector representation should be similar. The research chooses neuronal network approach with skip-gram-models for that (Mikolov et al., 2013).
Collecting all words from the training text and encoding every distinct word with the one-hot-encoding is one fundamental way to put it into practice. Assuming that our input text contains x distinct words, then every word is encoded with a vector of x dimensions, which consists only zeros and one times the number one. Now, the learning procedure feeds a neuronal network with y layers with the encoded text. The goal is that the network learns to predict the right probability that two words appear in a specific window together in the training text. Each output neuron encodes the probability to that word which is encoded with the one-hot-encoding on the position of the output neuron. The output vector for each word with y dimensions consists of the weights of the y layers of the trained neuronal network.
Besides to the semantic similarity-functionality, word embeddings have further advantages like encoding many linguistic regularities and patterns. For example, the trained Word2Vec-model

likely fulfils approximations like Equation 2.7 (Mikolov et al., 2013). Hence, this thesis uses the described technique, too.

$$vec(\text{Berlin}) - vec(\text{Germany}) + vec(\text{France}) \approx vec(\text{Paris}) \tag{2.7}$$

Brief introduction into the structure of natural languages

Figure 2.6 represents the different zoom levels of natural language. The highest level is the text level. A natural language process analyses one or more documents. When the analysis engine zooms in, then the paragraphs are in focus. Every text contains one or more paragraphs. The next zoom level contains sentences, which are often used in analysis pipelines. One sentence contains one or more clauses. Clauses are the smallest grammatical unit that can express a complete proposition. Every clause contains one or more phrases. "A phrase is a contiguous sequence of related words, functioning as a single meaning unit." (Jurafsky and Martin, 2008). Phrases can be nested. Analysis engines which detect phrases face the problem of structural ambiguity (Jurafsky and Martin, 2008). There are five types of phrases: the noun phrase, adjectival phrase, verb phrase, adverbial phrase and prepositional phrase. Phrases are called constituents, too. The next zoom level is the token level which contains mainly single words. Additionally, punctuation marks are tokens, too. Analysis engines do not annotate white spaces as tokens. There is the possibility to zoom into words. Basically, words are a chain of syllables. The deepest level is the character level. Every token contains one or more characters/ symbols. One of the most impressive level is the token level. A token annotation can be extended with additional analysis information. Every word contains single characters. A word contains units, the so-called syllables. For example, the syllables of the word *speaker* are *spea* and *ker*. To come back to the token level, every word contains exactly one morpheme. A morpheme is the smallest linguistic unit with a meaning or grammatical function (Jurafsky and Martin, 2008). For example, the morpheme of the word *speaker* is *speak*. Another linguistic unit of a word is its stem. A stem is the part of a wordform that never changes. The stem with its affixes (suffixes, prefixes, infixes, circumfixes) results in the whole word. For example, the stem of *rewording* is *word*. The next additional information can be the lemma. Lemma is called lexem, too. The lemma is the dictionary form of a word. For example, the lemma of *was* is *be*.
Important additional information for a token is the part of speech (POS) tag. The POS tag marks the word class (lexical category) or syntactic category of a word in a sentence, respectively (Jurafsky and Martin, 2008). POS tags reveal a lot about a word and its neighbour. Furthermore, it is a crucial aspect of parsing to detect and classify, for example, phrases or to build dependency trees of the words in a sentence (Jurafsky and Martin, 2008). Analysis engines which add POS tags often have a closed word classes set like the Penn Treebank tagset[5]. The amount is fixed as in opposite to open word classes sets. Closed word classes sets are not adaptable to changes in the language. A typical abstract word class is, for example, the nouns. This class has, for example, the specific classes "single noun", "plural noun" and "proper noun". A problem which POS tagger faces is the ambiguity. Most word types are unambiguous (80-86%). The issue is that ambiguous words are often widespread words. In the end, the POS tagging is not trivial for 55-67% of the tokens in running texts (Jurafsky and Martin, 2008). There are different solutions for that problem: the simplest one is the most frequent class tagging (accuracy: 92%). Better are sequence classifiers. These classifiers are based on hidden Markov models, for example. A third possibility is to use machine learning.

[5]`https://www.sketchengine.eu/penn-treebank-tagset/`

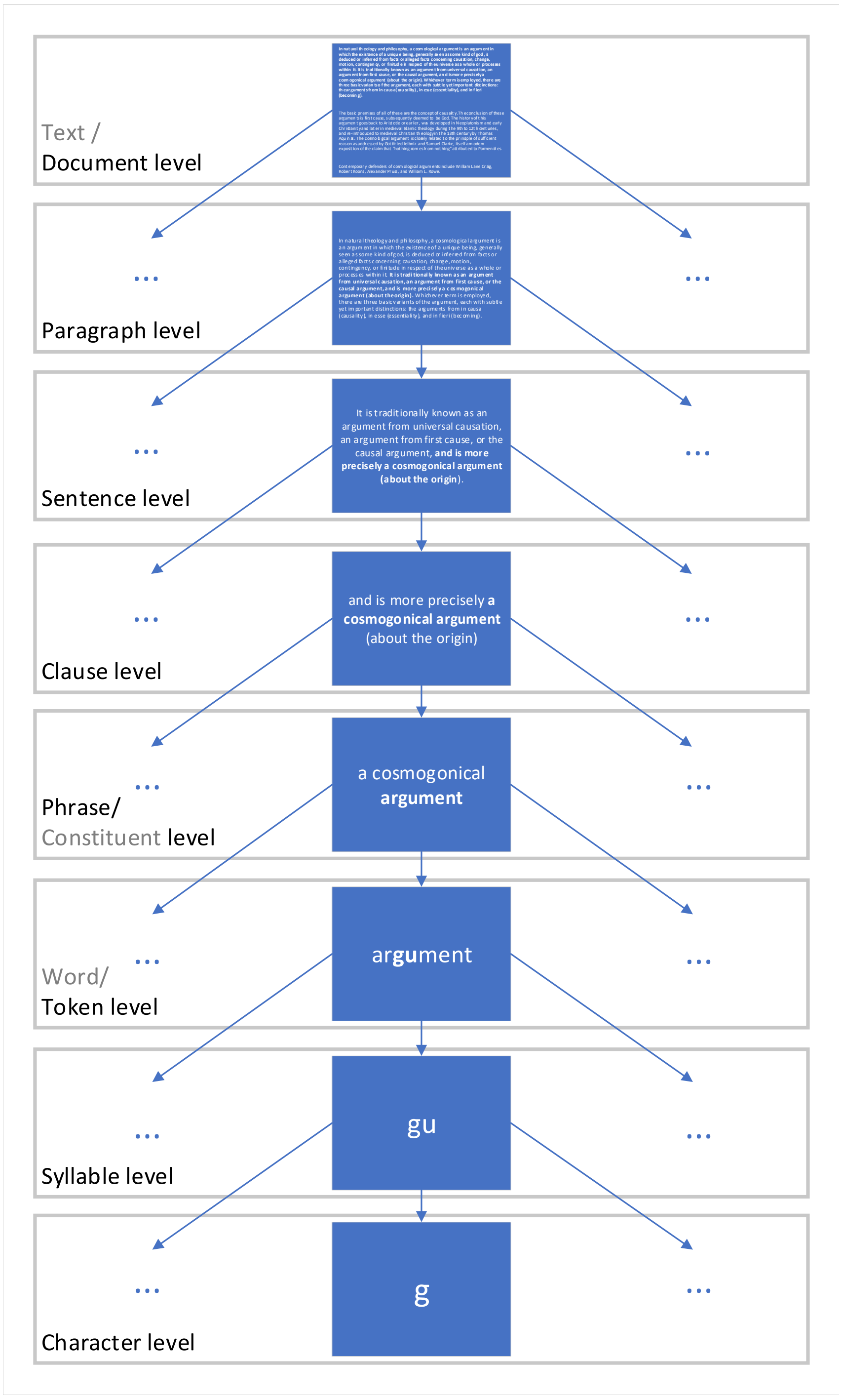

Text /
Document level
Paragraph level
In natural theology and philosophy, a cosmological argument is an argument in which the existence of a unique being, generally seen as some kind of god, is deduced or inferred from facts or alleged facts concerning causation, change, motion, contingency, or finitude in respect of the universe as a whole or processes within it. It is traditionally known as an argument from universal causation, an argument from first cause, or the causal argument, and is more precisely a cosmogonical argument (about the origin). Whichever term is employed, there are three basic variants of the argument, each with subtle yet important distinctions: the arguments from in causa (causality), in esse (essentiality), and in fieri (becoming).
Sentence level
It is traditionally known as an argument from universal causation, an argument from first cause, or the causal argument, and is more precisely a cosmogonical argument (about the origin).
Clause level
and is more precisely a cosmogonical argument (about the origin)
Phrase/
Constituent level
a cosmogonical argument
Word/
Token level
argument
Syllable level
gu
Character level
g
...

Figure 2.6: This figure shows the structure of natural language.

Word (phrases) relations: the WordNet: Fellbaum (1998) presented the WordNet® of the Princeton University[6]. The WordNet is an extensive lexical database for the English language. We will use this freely available lexicon. It is based on sets of cognitive synonyms that express a concept uniquely. These 117,000 sets are called *synset*. For example, the synset {God, Supreme Being} represents the concept of a "*supernatural being conceived as the perfect and omnipotent and omniscient originator and ruler of the universe; the object of worship in monotheistic religions*". Each synset is like a node in a graph. Semantic relations between synsets represent edges. Therefore, the database is called WordNet. There are several relation types: hyperonymy/instance (superordinated synsets including specific persons, countries and geographic entities), hyponymy / has_instance (subordination), meronymy (the part-whole relation) and antonymy (direct contrastive synsets). For example, the synset {God, Supreme Being} has as an instance which describes the name for the God of the Old Testament (Judaism, Christianity). This synset contains amongst others the word "Yahweh". Figure A.1 shows examples of semantic relations which we retrieved from WordNet.
The WordNet takes some general POS-tags of words into account. Also, the lexicon provides some additional information to each synset, like example sentences and frequency counts. However, the ability to search through the WordNet-graph and retrieving synonyms are enough for this thesis.

2.1.3 Computational argumentation and stances

Computational argumentation is basically the computational analysis and synthesis of arguments in natural language and argumentation. Often computational argumentation is based on empirical data (Wachsmuth, 2015). Computational argumentation includes detecting, structuring and indexing arguments. Another task of computational argumentation is to match and rank arguments regarding a specific search query like in `args.me` (Wachsmuth et al., 2017). However, this thesis does not dive into argument mining and argument ranking.

Computational argumentation deals with argumentative texts. Such a text contains one or more arguments. Each argument contains one or more premises and the conclusion – the central *claim*. The claim is the most important part of an argument (Stede and Schneider, 2018). However, its position can be at the beginning or at the end or somewhere else. The premises argue all in all for the conclusion. However, not all premises must directly support the conclusion. An argument can include attacking claims, too. The goal of including such claims is to refute or weaken them (the relation label is "undercut") (Stede and Schneider, 2018). Arguments can be nested. Additionally, it is possible to build a chain of arguments. The stance of an argument represents a two-sided position of the writer regarding a given topic (Stede and Schneider, 2018). A topic is a central phrase or statement in a debate or an argumentative text like a title. If an argument argues for the topic, then it is a *PRO* argument. Otherwise, if an argument argues against the topic, then it is a *CON* argument. Hence, stance classification is a for/against classification (Stede and Schneider, 2018).
Relating to stance classification, it can be helpful to take a view on the sentiment and opinion term and the target term. IBM defined the *target* of a claim of an argument as the central noun phrase, about which the claim or argument, respectively, makes a positive or negative statement (Bar-Haim et al., 2017a). For example, the target of the claim *"Therefore, there are many hints for God's existence"* is *God's existence*. The *sentiment* "is a type of linguistic subjectivity, specifically positive and negative expressions of emotions, judgements, and evaluations" (Somasundaran and Wiebe, 2010). The sentiment sketches the polarity, the personal

[6]`https://wordnet.princeton.edu/`: it is usable online, too, at `http://wordnetweb.princeton.edu/perl/webwn`

opinion on a topic. Therefore, the sentiment is a clue for the stance, but the sentiment alone is not sufficient for stance classification (Mohammad et al., 2017). Consider the claim *"The belief in God's existence is an outdated idea."* The sentiment is negative. The personal opinion of the author is contrastive to the topic *"God's existence"* and in that case, this claim is the conclusion of a *CON* argument. If the topic is *atheism*, then the sentiment is still negative, but now the argument argues *for* the topic. Therefore, the stance is *PRO* and not *CON*. Furthermore, an author can express his stance without sentiment.

This thesis defines *stance classification* as a task to classify the stance of an argument (this paper has an inside view on the claim of the argument) towards a topic. We assume that the argument is related to the topic. Therefore, it is a binary classification problem. An algorithm should compute this classification. This NLP task is called *stance detection* if there is no assumption that the argument is related to the topic (Stede and Schneider, 2018).
Stance classification is not a trivial task. Therefore, many types of research exist about the topic. This thesis sketches some approaches in the next Section (Section 2.2.2). The approach of this paper is described in Chapter 3.

2.2 Related work

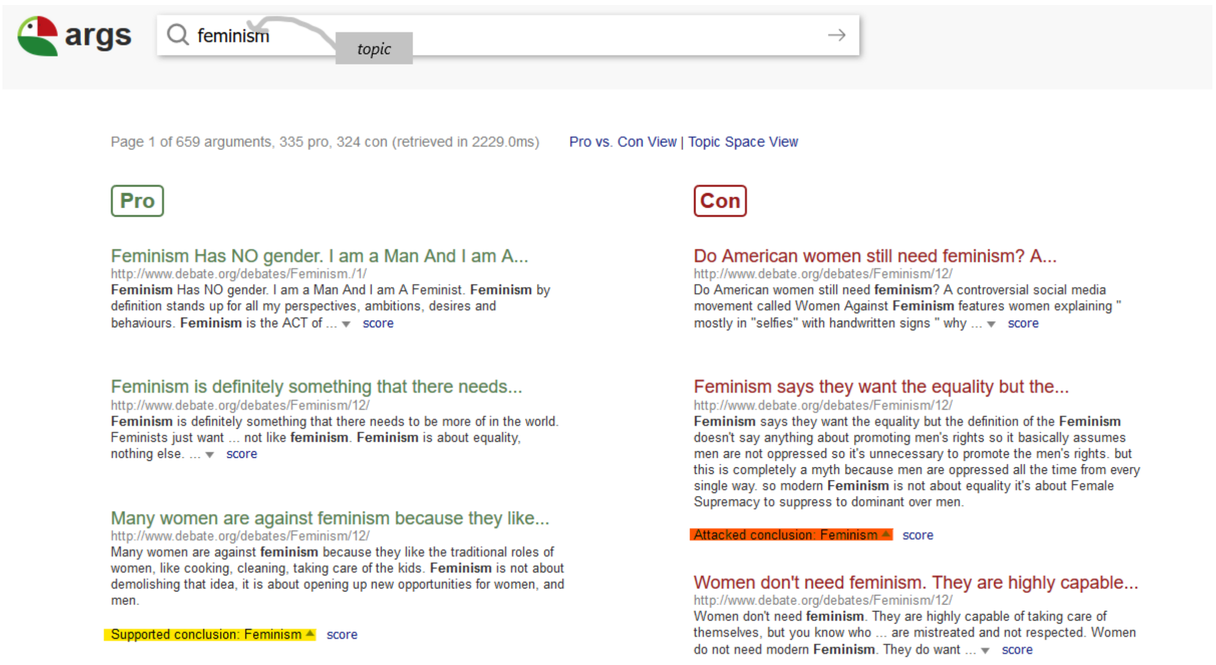
The argument search needs stance classification in order to sort each argument to the correct side: *PRO* and *CON*. This thesis is about stance classification in argument search. Therefore, Section 2.2.1 gives an overview of the argument search (specifically for the used search engine `args.me`). Section 2.2.2 sketches a part of already existing approaches for stance classification.

2.2.1 The argument search

Many questions do not have only one single correct answer. There are lots of controversial topics like the existence of God, feminism and more. Traditional search engines do not offer explicitly an overview of existing opinions likewise arguments for and against the controversial topic. Computational argumentation improves the search experience for opinion summarisation and decision making (Wachsmuth et al., 2017).
The argument search fills this gap. Wachsmuth et al. (2017) developed a search engine `args.me`. The user can enter a query. The engine retrieves arguments which match the query (the topic). Furthermore, the engine ranks the arguments and presents them. The presentation includes a labelling in *PRO* and *CON* arguments. The user can switch between the *Pro vs Con View* and the *Topic Space View*. Figure 2.7 shows a screenshot of the current implementation.
The contributions of Wachsmuth et al. (2017) are various. They present an extensible argument search framework. This framework is related to the research on argument search on the web. It includes a common argument model. Each argument has an ID, a conclusion (the claim), and a list of premises. Each premise has a stance label towards the conclusion. Additionally, each argument has a context. One example is the URL of the discussion which includes the argument. Model extensions are allowed but not necessary in this thesis. Another contribution of Wachsmuth et al. (2017) is an argument search index. The index provides 291,440 arguments. Wachsmuth et al. (2017) claims that this index is the largest argument resource. The arguments were crawled from online debate portals[7]. The final contribution of Wachsmuth et al. (2017) is

[7]The debate portals are `idebate.org`, `debatepedia.org`, `debatewise.org`, `debate.org` and `forandagainst.com`. The conclusion is the debate title in the first four web sites. In the last case, it is the claim. The crawler can directly retrieve the stance label of each premise to the conclusion from the discussion in the debate portal (Wachsmuth et al., 2017)

a prototype of an argument search engine `args.me`. This thesis refers to that search engine and provides a new stance classification algorithm for this search engine.
A user can query a word/topic standalone like *God's existence*. In that case, *PRO* arguments argue for the topic, *CON* arguments against it. Furthermore, a user can enter a query which includes already a stance like *God exist*, or *God does not exist*. The second query should invert the two sides. A third option is a request for comparison like *theism vs atheism* (Wachsmuth et al., 2017). However, this thesis focuses on the first two query types.
The implementation of stance classification of the retrieved arguments in `args.me` is only a simple heuristic until yet. Given a premise and a conclusion which is related to the search query, the stance label is copied from the stance label from the premise to the conclusion.



Page 1 of 659 arguments, 335 pro, 324 con (retrieved in 2229.0ms) Pro vs. Con View | Topic Space View

Pro

Feminism Has NO gender. I am a Man And I am A...
http://www.debate.org/debates/Feminism./1/
Feminism Has NO gender. I am a Man And I am A Feminist. **Feminism** by definition stands up for all my perspectives, ambitions, desires and behaviours. **Feminism** is the ACT of ... ▾ score

Feminism is definitely something that there needs...
http://www.debate.org/debates/Feminism/12/
Feminism is definitely something that there needs to be more of in the world. Feminists just want ... not like **feminism**. **Feminism** is about equality, nothing else. ... ▾ score

Many women are against feminism because they like...
http://www.debate.org/debates/Feminism/12/
Many women are against **feminism** because they like the traditional roles of women, like cooking, cleaning, taking care of the kids. **Feminism** is not about demolishing that idea, it is about opening up new opportunities for women, and men.

Supported conclusion: Feminism ▴ score

Con

Do American women still need feminism? A...
http://www.debate.org/debates/Feminism/12/
Do American women still need **feminism**? A controversial social media movement called Women Against **Feminism** features women explaining " mostly in "selfies" with handwritten signs " why ... ▾ score

Feminism says they want the equality but the...
http://www.debate.org/debates/Feminism/12/
Feminism says they want the equality but the definition of the **Feminism** doesn't say anything about promoting men's rights so it basically assumes men are not oppressed so it's unnecessary to promote the men's rights. but this is completely a myth because men are oppressed all the time from every single way. so modern **Feminism** is not about equality it's about Female Supremacy to suppress to dominant over men.

Attacked conclusion: Feminism ▴ score

Women don't need feminism. They are highly capable...
http://www.debate.org/debates/Feminism/12/
Women don't need **feminism**. They are highly capable of taking care of themselves, but you know who ... are mistreated and not respected. Women do not need modern **Feminism**. They do want ... ▾ score

Figure 2.7: This figure is a screenshot of the search engine `args.me`. We added three marks into the picture. This view appears after entering a search query (the topic). The search engine matches arguments to the topic and presents *PRO* and *CON* arguments.

2.2.2 Existing approaches for the classification of stances

An important example in the early work in stance classification comes from Somasundaran and Wiebe (2009). They focussed on online debates. More precisely, their corpus is restricted to two-sided debates for products and competitors, for example, *Blackberry vs iPhone*. They mined the web to learn associations and concession relations. Additionally, they extracted topic and subjectivity clues. In the end, they formulated the classification task as an Integer Linear Programming problem.
Other researchers work with the same data source. The interactions of posts in a discussion thread come into play. For example, some researchers emphasised the role of rebuttals in the interactions. Other researches built a complete network of for and against groups of users (Stede and Schneider, 2018).
Hasan and Ng (2013) took a view on ideological debates from another debating portal and investigated the interactions between posts. Each user has a particular stance towards a topic. Hence, there are author constraints. This knowledge was used with internet query search,

additional to n-gram-features, syntactic features like punctuation cues and semantic query. In the end, Hasan and Ng (2013) presented two helpful patterns which they combined in an Integer Linear Programming inference problem:

- Interaction patterns: mostly a post is followed by a reply with the opposite stance in a debate thread
- Ideology patterns: The opinion of an author is not independent of his/her opinions to other topics. For example, Hasan and Ng (2013) found out that the probability is high, that, if an author argues for gay rights, he has a positive stance towards marijuana.

Hasan and Ng (2013) reaches an accuracy of 71-76% in predicting the right stance in online ideological debates.
A further example of the stance classification of online debates is the approach by Sridhar et al. (2014). This approach combines linguistic features with features based on the network structure. Therefore, the underlying relationships between posts, including stance labels, and users help to classify the stance. This approach uses the probabilistic soft logic model for classification. An advantage of this model over some machine learning approaches, which are like a "black box", is the production of weights of symbolic rules. This behaviour is useful for error analysis. Sridhar et al. (2014) reach an average f_1-score of 0.74.

Faulkner (2014) analysed the stance of student essays and developed an automatically essay-stance-classifier. He claimed that the relationship between linguistics and NLP is an on-again-off-again affair and presented a linguistic approach which makes use of linguistic subtleties. In detail, his approach has two key ingredients:

- Part-of-speech-generalised dependency subtrees: the approach captures the relationship between a stance or attitudinal expression (lexicon based – words or phrases) and the target of the expression. A standard syntactic parser is used to reach that goal. The extracted dependency subtrees symbolise stance-attitude profiles
- Prompt-topic-features: motivated by related work in aspect-based sentiment analysis, the approach considers the relation of the prompt of an essay to its content. For that, the approach generates a set of essay words which are semantically related to words in the prompt. A Wikipedia link-based measure is used for choosing and scoring such words.

Faulkner (2014) trained an SVM and evaluated his topic-word model. For training and testing he uses a set of 1135 stance-annotated essays. The result is an overall accuracy of 82%.

A further auspicious approach is presented in Section 3.1. This approach is the base for the strategy which is presented in this thesis.

3 Adapting Stance Classification to Argument Search

This chapter gives an overview of the approach of this thesis. The approach is inspired by the approach of Bar-Haim et al. (2017a), which will be presented in Section 3.1. This approach is labelled as the approach of IBM. The modular idea of this approach, including the derivation of the central stance classification formula is the core idea if this thesis, too. Therefore, Section 3.1 describes the base.
Hence, the scenario of IBM is not the same as the scenario in argument search. Section 3.2 summarizes the differences and gives an insight into the importance of handling the differences. Section 3.2 is the reason for Section 3.3. This section explains the concepts of the necessary adaptations. Hence, this is the point where new research ideas come into play, and the discovery of stance classification in argument search starts profoundly.

3.1 Description of the approach of IBM

The section summarizes the paper of Bar-Haim et al. (2017a). They presented a novel modular approach of stance classification.
The authors decomposed the stance classification into three tasks:

1. Claim Target Identification
2. Claim Sentiment Classification
3. Contrast Classification

The approach of IBM calculates the stance of a claim towards another claim (the topic). Consider the claim c = “faith in God is beneficial” and the topic t = “atheism is not the truth”. IBM assumes a claim target x_c and a topic target x_t. A target is a phrase the claim contains. Furthermore, the text which contains the target makes a positive or negative assertion about it. On the base of this assumption, IBM defined the claim sentiment $s_c \in \{-1, 1\}$ and the topic sentiment $s_t \in \{-1, 1\}$ to its target. The value -1 indicates a negative sentiment towards the target and 1 a positive one. In the case of the example, x_c = faith in God and therefore $s_c = 1$. For the topic, x_t = atheism and hence $s_t = -1$.
Additionally, IBM defined a contrast relation between x_c and x_t, expressed with $\mathcal{R}(x_c, x_t) \in \{-1, 1\}$. A contrast relation of -1 means that the two targets are semantically contrastive to each other. This is the case in the example: *faith in God* implies the opposite stance towards *atheism*. Therefore, $\mathcal{R}(x_c, x_t) = -1$ holds in our example. Otherwise, a contrast relation of 1

means consistent targets. A consistent topic target of *faith in God* would be *theism* and, in the extended sense, for example, *God exists*, *religion* and *Christianity.* The contrast relation is the most challenging subtask.
The calculated values produce the final output, the stance. The output follows a simple formula (Equation 3.1).

$$Stance(c, t) = s_c \cdot \mathcal{R}(x_c, x_t) \cdot s_t \tag{3.1}$$

In our example the formula concludes that the claim *faith in God is beneficial* is the conclusion of a *PRO* argument towards the topic *atheism is not the truth* ($1 \cdot -1 \cdot -1 = 1$). This inference is true and was not a difficult task for the modular approach. This assertion holds in contrast to some other stance classification approaches which would face a difficult task in this case.
Until now, all introduced number values, including the output, were presented as binaries: the values are exclusive -1 or 1. IBM presented a continuous variant, too. In that model, $Stance(c, t), \mathcal{R}(x_c, x_t), s_c, s_t \in [-1, 1]$. Real values have several advantages: the continuous model allows the scientist to include the confidence of a value prediction. If a value is 0 or nearly 0, then it means that the algorithm is very unsure about its prediction. Therefore, the output is not reliable. An example is the calculation of the contrast relation between *faith in God* and *Buddhism.* If the absolute value is 1 or nearly 1, then the algorithm is very confident about its prediction. Another example is the calculation of the contrast relation between *theism* and *atheism.* This gradation is useful for a ranking afterwards. Another use case is to introduce a threshold $\tau \in [0, 1]$. In this case, only such claims are outputted which fulfils $|Stance(c, t)| \geq \tau$. This filter can increase accuracy enormously.
Figure 3.1 sketches the continuous model which this paper adapts for the argument search.

3.1.1 The three subtasks in detail

Each subtask has its own challenges and solutions for these. This subsection presents mainly the approaches of Bar-Haim et al. (2017a) to these subtask. Improvements of Bar-Haim et al. (2017b) are already included.

Claim Target Identification

Claim Target Identification is a known problem in natural language. Ranade et al. (2013) extracted the targets by analysing the full dependency parse of utterances in online debates. Wojatzki and Zesch (2016) proposed to model implicit argumentation and covered the topic of target identification. They presented a semi-automated, bottom-up approach by using association words between topic and utterance. They claimed that selecting the most explicit target is not the best choice for overall stance analyses.
IBM decided to model the identification problem as a machine learning problem. They used a regularised logistic regression classifier for the task for labelling phrases with a confidence value for being the target. To do that, they first extracted noun phrases with the help of a constituent parser[1]. Each noun phrase was a candidate for being the target. In the next step, IBM extracted features for the classifier from each candidate. They used syntactic and positional features. Also, they searched the candidate in Wikipedia to find out whether it appears as a title or not. They included dependency relations connecting the candidate to any sentiment phrase in the rest of the claim. The last feature type was the topic relatedness. Three different values were calculated: morphological similarity, paths in WordNet and cosine similarity of Word2Vec embeddings.

[1]more precisely, the ESG parser

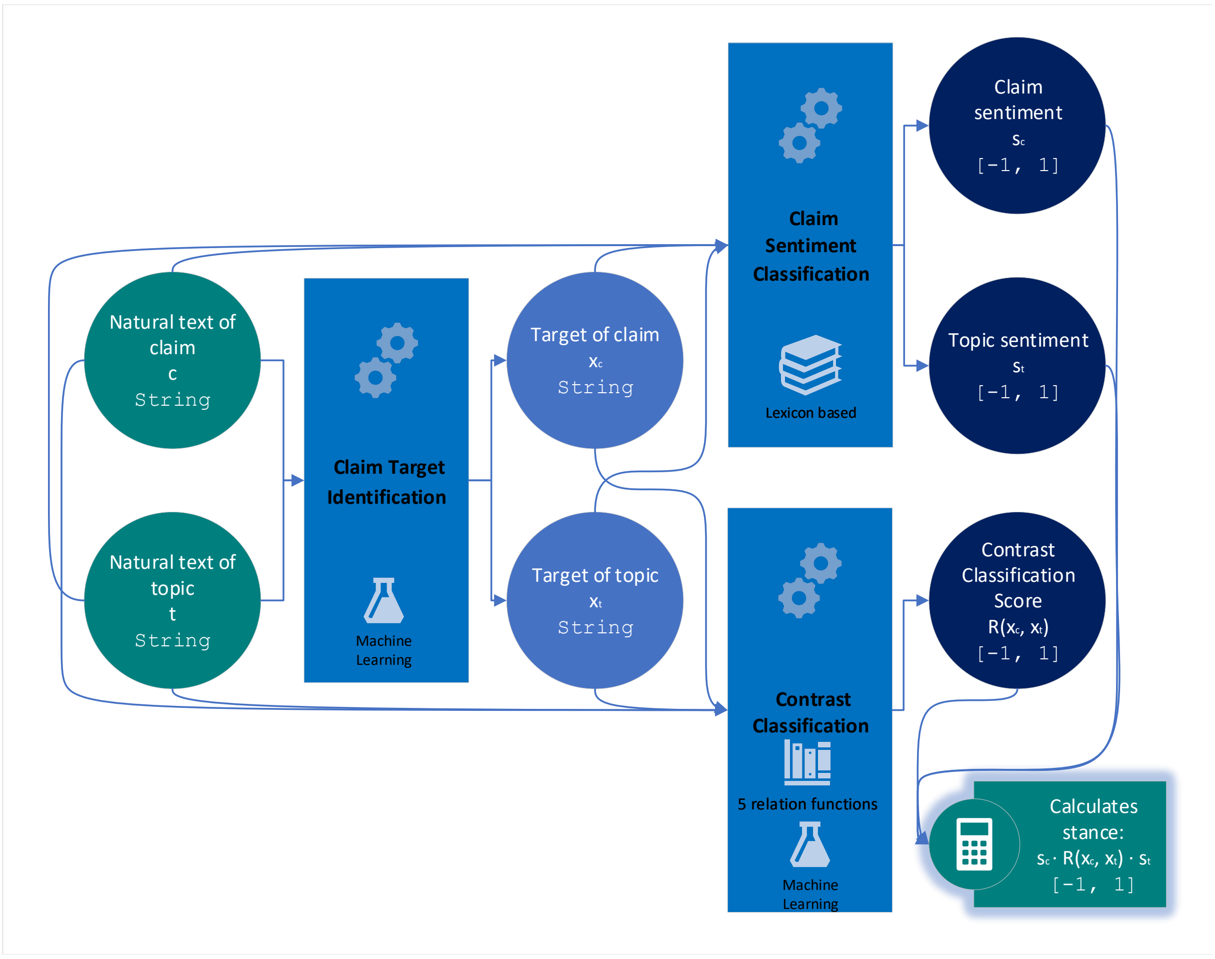


Figure 3.1: This figure represents a broad overview of the stance classification approach of IBM (Bar-Haim et al., 2017a). The stance classifier gets two claims as input in natural language. One of them is a conclusion of an argument, the other is the topic claim. Three main components infer knowledge out of the strings which other steps use later on. The general flow is to extract the target and then to determine the sentiment towards the target and to calculate the contrast relation between claim and topic. The last step is to compute the Equation 3.1.

Claim Sentiment Classification

Sentiment Classification is a very known problem in natural language. There are many approaches. Some of them use a bag-of-word-learning-strategy (Pang et al., 2002); other approaches use already existing sentiment lexicons. A famous lexicon is the opinion lexicon of Hu and Liu (2004) (Bar-Haim et al., 2017a). This lexicon contains 6789 words and separates words in a list with positive sentiment and a list with negative sentiment. Another example is the *sentiWordNet* which stores for each word and for each sense of it a sentiment-positive-score and a sentiment-negative-score in WordNet (Ranade et al., 2013). All in all, the advantage of lexicon-based sentiment analysis is that they prevent overfitting. The disadvantage is that they do not consider the context of words with a sentiment. Hence, because of dependencies and ambiguous words, this can lead to wrong results (Khan et al., 2016).
Wang et al. (2010) introduced the aspect-based sentiment. Socher et al. (2013) introduced recursive neural networks for semantic linguistic compositions. Further ideas analysed the discourse structure, used argument-related models or the sentiment flow or have a semantic constitutionality view. Many approaches use supervised machine learning (Khan et al., 2016).

The main difference between the existing approaches and the approach of IBM is IBM calculates the sentiment towards the target of the claim. The previous approaches calculate the overall sentiment, for example, of a claim. In most cases both sentiments are consistent. However, in the following claim, the sentiments are not equal: “Right persons love nature and therefore avoid aeroplanes” has a positive sentiment. Consider the target “aeroplanes” now. This claim has now a negative sentiment towards its target.
The approach of IBM is lexicon-based. First, they match words with sentiment[2] from the opinion lexicon of Hu and Liu. In the next step, they consider the context of sentiment words with the sentiment shifter application. If a word from a sentiment shifters list is in front of a sentiment word, the application inverts its sentiment. The last step points out the target: a distance function weights each sentiment value regarding its distance to the target. If d is the distance in tokens, then the final sentiment s of a word is $s \mapsto s \cdot d^{-0.5}$. For the final sentiment prediction, the Equation 3.2 was used. p_c is the weighted sum of positive sentiments in the claim and n_c the same for negative sentiments. Analogously, p_t and n_t are defined for the topic.

$$s_c = \frac{p_c - |n_c|}{p_c + |n_c| + 1} \qquad s_t = \frac{p_t - |n_t|}{p_t + |n_t| + 1} \tag{3.2}$$

Later IBM presented an improvement in the sentiment matching step. The aim was to extend the opinion lexicon to cover more sentiment words. They trained an SVM classifier to predict the sentiment polarity for obscure words using word embeddings. The resulting accuracy was 90.5% and led to a sentiment lexicon with 938,559 words. Because of the large size, they considered to compress it by filtering using relations on a WordNet.

Until now, the approach only takes into account the isolated claim. Therefore, Bar-Haim et al. (2017b) presents an extension for the claim sentiment classification which suggested contextual features. The assumption behind the features is that the neighbouring texts tend to agree on the sentiment. Therefore they trained an SVM classifier on these contextual features. They suggested the following contextual features types:

[2]The sentiment value is -1 or 1

Wikipedia Header Features: Often, claims appear in Wikipedia articles. The headers of an article which includes the claim can reveal something regarding the sentiment of a claim. If the claim appears in the *Criticism*-section, the sentiment is probably negative. In the opposite, if the claim appears in the *Advantages*-section, the sentiment is probably positive. Therefore, IBM takes the sentiment of the header of the section, the subsection and the sub-subsection as a feature. Also, they add the difference between positive sentiment words and negative sentiment words in each header as features.

Neighbouring claims and sentences: IBM stores the surrounded text to each claim. This is intuitive in an argumentative text: the central claim is the conclusion, and the surrounded text contains the premises, which are themselves claims, too. 88% of the claims shared the majority polarity. They cluster successive claims until a potential polarity flip term was found like *however*, *although* and *in contrast*. They add the sum of the sentiment scores over all other claims in the cluster of the predicted claim as a further feature.

Contrast Classification

Contrast Classification in the sense of two targets is not a common problem in computational linguistics until now. IBM suggests the following approach: first, they generate anchor candidates from the claim and the topic. An anchor candidate is every word. Additionally, they consider phrases as anchor candidates. The aim was to have a strong association to debate topic and, in addition, semantic relation between paired anchors candidates, to find all consistent and contrastive cues. Hence, in the next step, they selected the main anchor pair. An anchor pair a combination of an anchor candidate from the claim a_c and from the topic a_t. The Equation 3.3 determines the choice.

$$(a_c, a_t) = \arg \max_{\forall i \in \{1,\cdots,n\}, j \in \{1,\cdots,m\}} w(a_{c_i}) \cdot |r(a_{c_i}, a_{t_j})| \cdot w(a_{t_j}) \tag{3.3}$$

In Equation 3.3, $w(x)$ represents the ratio of the frequency of x in articles that were identified as relevant to the topic in the dataset of IBM and the overall frequency in Wikipedia. r is a relatedness function which outputs a value in the range between -1 (contrastive relation) and 1 (consistent relation). IBM presented several relatedness functions which will be presented later on in this thesis. The next component is the computation of the contrast score of an anchor pair. A polarity function is used for that purpose which is lexicon-based. The formula for the contrast score is in Equation 3.4.

$$\text{contrast score}(a_c, a_t) = \textit{polarity}(x_c, a_c) \cdot r(a_c, a_t) \cdot \textit{polarity}(x_t, a_t) \tag{3.4}$$

Now we have all parts for the final algorithm of contrast classification. First, the foremost K anchor pairs are determined with the use of Equation 3.3. Then, the algorithm separately calculates the contrast score for each contrast relation function r (Equation 3.4) for the main anchor pair. The algorithm calculates the value of the currently used relation functions of the other $K-1$ anchor pairs, too. These values, converted with the sign-function, are multiplied with the contrast score of the main anchor pair. As the result, the algorithm calculates an extended contrast score $cs_r \in [-1, 1]$ for each contrast relation function. IBM treats each contrast score as a feature. Then, IBM uses random forest classifier to get the final likelihood prediction that x_t and x_c being consistent.
The following example helps to get a better understanding of the extended contrast score computation. Consider the claim "*Practical theory* is useful" to the topic "*Theoretical practise* is helpful". Hence, assume that the main anchor pair is $(\textit{theory}, \textit{practise})$. Furthermore, consider $K = 2$ and the next best anchor pair is $(\textit{Practical}, \textit{Theoretical})$. Hence, assume that the

contrast score of the main anchor pair is $1 \cdot -0.9 \cdot 1 = -0.9$ – theory and practice are mainly contrastive. The relation function is for example $r(Practical, Theoretical) = -0.9$. In the aggregation step, the result is $-0.9 \cdot sgn(-0.9) = -0.9 \cdot -1 = 0.9$. This result meets the example because the targets are consistent.

IBM introduced the following contrast relation functions $r(u, v) \in [-1, 1]$[3] which are used several times in the contrast classification step:

- methods for single tokens
  - morphological similarity
  - cosine similarity in word2vec embeddings
  - reachability in WordNet via synonym-antonym chains
  - thesaurus-based synonym-antonym relations using polarity-inducing LSA
- method for phrases: this method is based on co-occurrence of the anchor pair with consistent and contrastive cue-phrases in Wikipedia headers and query logs

3.1.2 Further contributions and experiments

Bar-Haim et al. (2017a) presented not only a novel approach but a meaningful annotated argument corpus, too. The researchers updated an earlier version, the IBM argumentative structure dataset. The updated version contains 55 controversial topics, randomly selected from the debate motions database at the International Debate Education Association (IDEA) website. Each topic follows the tradition of British Parliamentary debates. Therefore, each topic starts with "This house". Each topic has a list of related claims. The 2,394 claims come from hundreds of Wikipedia articles. Then, human annotators added information to the topics and claims: they decided the stance giving a vote for the sentiment and the target. Besides that, the structured dataset contains more information which is not necessary for this thesis.

Also, they did experiments with the help of their dataset. In the case of classifying all claims (hence, the threshold $\Theta = 0$ for predicting the stance) they presented improvements in accuracy to some strong stance classification baselines. The full approach which Section 3.1.1 describes without the extensions results in an accuracy of 0.632. This accuracy is equal to the baseline, which uses unigrams and a sentiment SVM. However, a reduced variant of the approach is slightly better: the reduced variant assumes that $\mathcal{R}(x_c, x_t) = 1$ holds for all claim-topic-pairs. Additionally, the sentiment of claim and topic is computed with a traditional sentiment algorithm which does not take the target into account. However, by adding contextual features and the idea of extending the opinion lexicon for Sentiment Classification, the accuracy of the whole approach increases to 0.691. For coverage of predicting 60% or less of the stances (therefore there is a noticeable threshold Θ which filters out predictions where the classifier is very unsure), even without the extensions they reached the best accuracy with the full implementation of their approach: an accuracy of 0.74 to 0.85. The accuracy values, which include the extensions, reach 0.78 to 0.95. Figure 3.2 plots the accuracy values.

[3]In the set of relation function are similarity measures, too. Such "relation functions" outputs values in the range between 0 and 1

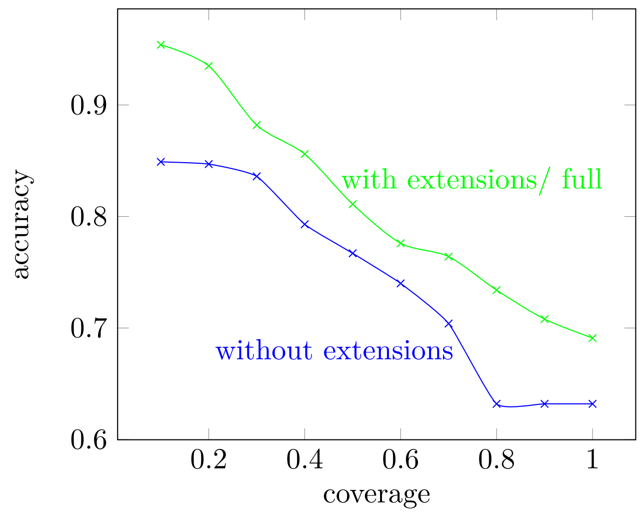


Figure 3.2: This plot, which we created, represents the stance classification results of the experiments by Bar-Haim et al. (2017a) and Bar-Haim et al. (2017b). The green lines shows the effectiveness the full implementation of IBM. The blue line shows the effectiveness of the original approach without the sentiment lexicon extensions and the contextual features. The coverage infers the fraction of claims which the algorithm is forced to classify.

3.2 Differences in the case of argument search

The approach of IBM provides a solution for Stance Classification – this is the same task for the argument search. An algorithm should correctly sort the crawled arguments into the *PRO* or *CON* side. However, the database is not the same in both cases. The claims of IBM belong to Wikipedia articles (Bar-Haim et al., 2017a) and are always sentences. Many of them are linguistically complex and accurate because of the standards of Wikipedia. This is not the case in `args.me`. The claims are conclusions of arguments from debate portals in this case. That means: in most cases claims are titles of forum threads/ debates (Wachsmuth et al., 2017). This source has in many cases an effect in that sense that the claim is just a keyword like *feminism* instead of full sentences. Furthermore, claims can be questions or two phrases with *versus* in between in rare cases. Additional challenges are spelling and punctuation mistakes in many cases. Challenging conclusions from `args.me` are “Men Have Big Problems Too” or “Abortion=Good!!!!!!”. Missing punctuations can lead to false parsing results[4], for example. Spelling mistakes can lead to cases where an algorithm cannot find a particular word. A third challenge is a critical degree of missing context in some cases – more than in the IBM dataset case. For example, `args.me` contains the conclusion “For And Against Is BETTER Than Google.”. This conclusion seems strange without background knowledge. However, with the knowledge, that this conclusion was on the web site `http://www.forandagainst.com`, the conclusion is sensible. The user wants to debate whether this web site is better than the known search engine for controversial topics. There is an attenuation for these challenges: in most cases the claims of `args.me` are linguistically simpler than the claims of the IBM database. Half of all claims in `args.me` have six to ten tokens (Ajjour et al., 2019).
Furthermore, not only the claims are different; the topics are different, too. IBM has a fixed

[4]For example constituency parsing: robust parsers can handle spelling and punctuation mistakes to a certain degree. For example, the Stanford constituency parser delivers the same result for “Men Have Big Problems Too”, and the corrected variant “Men have big problems, too.”

set of topics. Each topic is a claim itself. In `args.me` the topics are user-defined search queries. 58% of all search queries contain only one word, and only 2% of all search queries contain more than four tokens (Ajjour et al., 2019). Hence, the minority of search queries are sentences. The majority are topic titles like "feminism". The *PRO* side should argue for the topic and the *CON* side against it. Also, the stance classifier in argument search should handle queries with a comparison or/and an indicated stance in the end (Wachsmuth et al., 2017).
Furthermore, the argument search requires an extension of the central formula which IBM presented. The algorithm must consider to classify the stance of an argument the stance of its premises towards its conclusion, too. An opposite is the case of IBM: only the claim is decisive for the stance. Hence, let $\mathcal{S}(a, c)$ the stance of the argument's premises a towards its conclusion, the claim c. If the author labels his text as a set of premises which support the conclusion, we set $\mathcal{S}(a, c) = 1$. Otherwise if there is an attacked relationship, we set $\mathcal{S}(a, c) = -1$. Hence, the final formula for the stance prediction of a certain argument's premise is Equation 3.5.

$$\mathit{Stance}(c, t) = \mathcal{S}(a, c) \cdot s_c \cdot \mathcal{R}(x_c, x_t) \cdot s_t \tag{3.5}$$

Besides, it is not possible to copy the contextual features which IBM suggested (Section 3.1.1). Each claim in the IBM corpus comes from a Wikipedia article. Therefore, the claim itself is directly inside the argumentative text. This context is not given in `args.me` (Wachsmuth et al., 2017). The conclusion is not a part word by word of the argumentative text in many cases.

Another critical difference is the importance of efficiency. IBM have never written something about runtimes or related terms in their two papers about stance classification. Hence, it becomes clear that efficiency was second-rate in some points. One example is the contrast classification (Section 3.1.1). IBM did complex computations not only for one anchor pair but for many pairs. Consider the claim "Exposure to violent video games causes at least a temporary increase in aggression and this exposure correlates with aggression in the real world" and the topic "This house believes that the sale of violent video games to minors should be banned" from their corpus. The claim contains 23 tokens and the topic 15 tokens. The researchers suggested to generate an anchor pair for every tuple of two tokens. Therefore, the algorithm must handle $23 \cdot 15 = 345$ anchor pairs. Also, IBM suggested considering phrases as anchor candidates. Hence, in the end, there is a workload of over 350 pairs instead of one pair: the pair of claim target and topic target. A second example is that IBM considered morphemes. Wachsmuth (2015) shows that the Mate Tools, which can extract morphemes, takes notably more time than some other basic primitive analysis engines. For example, the TT4J-TreeTagger needs approximately one millisecond for tagging the lemma and POS of all tokens in a sentence. In opposite, the Mate Tools needs approximately 21 milliseconds per sentences.
The challenge is to deliver results on-demand in argument search. A response time of more than 1000ms may interrupt the flow of thought of a user (Brutlag et al., 2008). If the response time exceeds 3000ms, then the majority of users tend to choose another faster search engine (Brutlag et al., 2008). All processing steps, including the stance classification, should meet that time limitation. One request loads 20 arguments. Therefore, the stance classification of an argument's premise should not exceed 150 milliseconds, if the server uses no parallelisation for this task. A computation time of 50ms or less can lead to a fluent workflow by using the search engine.

3.3 Concepts of adaptation

This thesis presents solutions for the differences which Section 3.2 describes. Hence, there are several differences, so there are many single concepts to solve the problem. Table 3.1 gives an overview over all adaptations which will explained by the following subsections.

3.3.1 Concepts of handling claims and topics which are not linguistically correct sentences

The conclusions of arguments are the classified claims in the argument search. Internet users wrote these arguments including conclusions and forum thread titles. Therefore, Section 3.2 already describes that linguistic mistakes can occur. There are two approaches to face the problem. One approach is to correct the claims. There exist several spelling and grammar checkers which can be potentially applied. These checkers can correct the query, too. Hence, this approach is not sensible for the stance classification step. If checkers correct claims which fits to the query for every request, then every single claim will be corrected many times. This is a waste of computation time. More applicable is to use checkers in the indexing process[5]. Argument search is a process with two main steps in the case of `args.me`: the indexing and retrieval process (Wachsmuth et al., 2017). The indexing process includes the text acquisition (e.g. with web crawling), the mining and assessment process (here checkers can correct the relevant text parts) and the indexing process. Hence, in the indexing process, every argument is processed only once. Because of the independence of spelling and punctuation correction and the retrieval of arguments related to a search query the stance classifier should not check the right spelling or punctuation.
Hence, even by using such checkers, there will always be incorrect spelling/ punctuation mistakes. Hence, the solution is to seek for is choosing robust algorithms. Therefore, in addition to the linguistically correct sentences as claims and topics, the stance classifier must handle the following common cases:

Single word/ single phrases claims or queries, respectively: This input is an easy case if we classify, for example by using a heuristic, an input in this class. An algorithm chooses the whole string in the target identification step. In the sentiment classification step, the algorithm considers a positive sentiment. The reason is in the human reasoning: if a user argues in a debate portal in a certain thread regarding the topic t_F with a supported or *PRO* relation, respectively, he argues *for* the topic. Therefore the sentiment of the claim, which is the topic t_F must be positive. For example, if a user supports the debate with a title/ claim *God*, the user is likely to give a reason for God's existence or why it is reasonable to believe in God, respectively. The same argumentation applies for a query. If a user searches in the argument search case for *God*, he expects arguments for God on the *PRO* side and arguments against God, God's existence or faith in God, respectively, on the *CON* side. The Equation 3.5 shows the validity of this concept. If an authors supports the single word/ single phrase topic t_F and somebody searches for a related single word/ single phrase topic t'_F in the argument search engine, the stance classifier predicts for this argument's premise a: $Stance(t_F, t'_F) = \mathcal{S}(a, c) \cdot 1 \cdot \mathcal{R}(t_F, t'_F) \cdot 1 = 1 \cdot 1 \cdot 1 \cdot 1 = 1$. Hence, the algorithm sorts the argument's premise a to the *PRO* side.

versus-queries: versus-queries are valid queries for `args.me` (Wachsmuth et al., 2017). Consider the query *Christianity vs Islam.* If the argument search engine retrieves arguments which contain Christianity or Islam, then the search engine retrieves probably too many arguments. Consider the conclusion *Religions, for example Islam, are bad* and an argument which supports that claim. The argument is neither a *PRO* argument nor a *CON* argument because it argues against Christianity *and* Islam. A concept suggestion to solve this problem is to have two retrieving processes: one process searches for claims, which include *Christianity.* Then, the stance

[5]A remark to applying checkers automatically without asking the author: checkers correct texts in a wrong direction, too. For example, consider a conclusion from `args.me` "*For And Against* Is BETTER Than Google.". Maybe this was a misleading result of a checker because what the author meant is the debate portal *ForAndAgainst* and not the three words.

classifier would discard all arguments with a negative stance towards the topic. The second process does the same with *Islam*. Hence, there is a *PRO-Christianity* column and a *PRO-Islam* column in the final presentation.

Questions as claims or queries, respectively: The concept is similar to a claim case. The concept of a question is to convert it to a claim. For example, if a user queries “Does God exist?” it is similar to the claim “God exists”. Therefore, such a user expects on the *PRO*-side arguments which support God’s existence and on the *CON*-side arguments which deny God’s existence.

Spelling mistakes in the argument index (therefore, in the claim): There is no efficient way to solve this problem in the retrieval process. If the mistake is in the class of upper and lower case mistakes, then the problem is not a critical problem in some languages like English by lowering all characters on the token level. Other languages are a little bit more critical, but not highly critical — like German. “Weg” is a noun and means way, path, journey or method, respectively. In opposite to “weg” – this word is an adverb and means something like away or gone, respectively. Hence, spelling mistakes constitutes a more critical problem. A spelling mistake may lead to empty results in lexicon-based processes like the sentiment labelling of tokens or the finding of paths in WordNet. Also, analysis engines on the token level can stumble by spelling mistakes, for example by retrieving the lemma of a word.
However, the argument index and the query log of `args.me` do not show many shreds of evidence of spelling mistakes except upper and lower case mistakes.

Punctuation mistakes in claims or queries, respectively: There is a valid assumption that claims or queries are only single sentences if they are not single words or single phrases. Hence, inserting or assuming a missing punctuation mark at the end of the string is a trivial solution that solves many punctuation mistakes. Although this solution exists, it does not solve the problem of missing commas.
Hence, it is essential to have a robust parser. There are considerable differences in robustness or tolerance to linguistically incorrect inputs, respectively. For example, the constituency parser of Stanford[6] is very robust but potentially slow. In opposite, the TT4J-TreeTagger/PhraseChunker[7] is very efficient but not robust against linguistic violations (Figure 3.3). Moreover, the TT4J-PhraseChunker executes a shallow constituency parsing by the using decision trees. A shallow constituency parsing tags the top-level phrases from a constituency parse tree. That means that phrases are ignored, which are included by a broader phrase. A second disadvantage of shallow parring is that phrases do not get references to parent entities. Parent entities are phrases which includes the tagged phrases or, if such phrases do not exist, the whole clause or sentences. However, a shallow parsing mostly is more effective than a complete tree-parsing. The TT4J-TreeTagger-Tools are actually very efficient: the average computation time of a shallow constituency parsing takes lesser than a millisecond for each sentence (Wachsmuth, 2015). However, a further disadvantage is the effectiveness with mistakes in the use of upper and lower case and the punctuation mistakes. It is often an effectiveness-efficiency-trade-off at the end. One approach is to test different parsers and decide later for one of them.
Although, even for humans, it is sometimes hard to infer the intended meaning out of linguistically incorrect sentences. Consider the claim “Religion will increase feed and heal” and “Religion will increase, feed and heal”. Only one comma has changed the message. However, claims in

[6]Stanford CoreNLP 3.9.2 which can be tested here: `http://corenlp.run/`
[7]`https://www.cis.uni-muenchen.de/ schmid/tools/TreeTagger/`

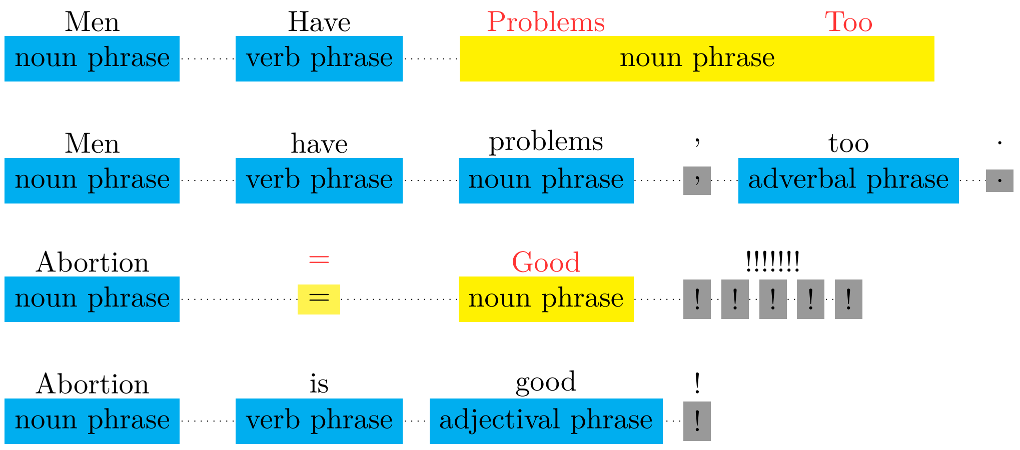


Figure 3.3: This pictorial representation shows the results of the shallow constituency parsing by the TT4J-TreeTagger/PhraseChunker. This figure contains two claims from the argument index of `args.me` (Ajjour et al., 2019): "Men Have Problems Too" and "Abortion=Good!!!!!" (Section 3.2). The TT4J-TreeTagger/PhraseChunker parses these two examples and does mistakes several times – in the figure marked by a red font (wrong POS-tag or lemma-tag) or a yellow node (wrong phrase label). The claims were manually corrected in the next step of the experiment. Now, the TT4J-TreeTagger/PhraseChunker was able to tag all entities correctly.

`args.me` are often not so complex linguistically (Ajjour et al., 2019). Also, the authors pay attention to such punctuation failures that lead to another interpretation than the author meant.

3.3.2 Concepts of handling not public parts

The application of stance classification of IBM and some related methods or references are commercial or internal and therefore not accessible for the public (Wachsmuth et al., 2017). Hence, the application which this thesis presents must reimplement approaches or is forced to find engines which are public for some part similar.
One important example is the ESG-Parser. IBM uses the ESG-Parser to find noun phrases in the claim target identification step and to generate anchor candidates in the contrast classification step (Bar-Haim et al., 2017a). Another example is a contrast relation function *thesaurus-based synonym-antonym relations using polarity-inducing LSA* (Bar-Haim et al., 2017a). There is a reference to a paper which describes the method in a sufficient way, but the code or a library is missing. Because of limited manpower it is not possible to reimplement every single method reference. Also, Bar-Haim et al. (2017a) mention parts of the approach where a detailed description is missing. One example is in the claim target identification step the suggestion of including *paths in WordNet* as a feature representation of a phrase. The Appendix A.1 of this thesis describes an algorithm for *paths in WordNet*.
Apart from this cases there are public references, too. For example the feature of *cosine similarity of word2vec embeddings* was easy to derive: there exists a library `deeplearning4j`[8] for Java.
Though, not only methods and functions are sometimes not public. The problem includes missing word lists and lexicons, too. For example, IBM uses a manually composed list with approximately 160 sentiment shifter words for the sentiment shifters application in the claim sentiment classification (Bar-Haim et al., 2017a). This list is not public and therefore we com-

[8]`https://deeplearning4j.org/`

posed a list which is not as comprehensive as the original list. Other missing lists are the cue phrases lists for the relation function of IBM in the contrast classification step. Furthermore, IBM uses the query logs of the Blekko^{TM} search engine which are not public, too. Instead, this thesis uses the query logs of AOL [9] from the year 2006. The search queries differ from the search queries of our days, but they are still useful to get clues about consistent and contrastive anchor pairs. Additionally, this thesis refers to the query logs of the `args.me`-argument-search-engine. This log fits perfectly to the task to find consistent and contrastive anchor pairs. The reason is the special type of querying. The log potentially contains many hints because of valid search patterns like “*Heavenly Father* and *Christianity*” (consistent) and “*summertime* vs *wintertime*” (contrastive). The problem is the small size of the log so far (Ajjour et al., 2019). But there is a mutual effect: if more users use `args.me`, then the contrast classification becomes more accurate and as result the stance classification. Hence, `args.me` is becoming more attractive and potentially more users use this argument search engine.

3.3.3 Concepts of efficiency

In this subsection, this thesis presents adaptations in favour of efficiency to succeed with an on-demand-solution on argument search. There are many points for saving computation time. One approach is to exchange primitive analysis engines in the analysis pipeline. This pipeline includes machine learning processes to predict, for example, the target of a claim. There are some steps in the pipeline in which more than one primitive analysis engine is available. One example is the step to determine phrases in a sentence. There exist several constituency parsers, like the parser of Stanford or the TT4J-PhraseChunker. In some cases, the runtime and effectiveness differs a lot (Figure 3.3) (Wachsmuth, 2015). One approach to increase efficiency is to evaluate different combinations and to observe the different side-effects of using different sets of analysis engines.
Following steps are necessary for a full approach-implementation of stance classification, including their considered primitive analysis engines as subitems in the list (we implemented the cursive printed analysis engines):

1. segmentation of sentences
    - InfexBASentenceSplitter/ WachsPottSentenceSplitter: rule-based sentence splitter (runtime[10] per sentences: < 1ms)
    - ICU4jSentenceSplitter: break-iterator algorithm (average runtime per sentences: < 1ms)
    - UIMATokenAndSentenceSplitter: simple whitespace segmentation (average runtime per sentences: < 1ms)
2. segmentations of tokens
    - UIMATokenAndSentenceSplitter: simple whitespace segmentation (average runtime per sentences: < 1ms)
    - InfexBATokenizer: rule-based token splitter (average runtime per sentences: < 1ms)
3. segmentation of syllables
    - TextHyphJSyllableDetector: TeXHyphenator-J[11] (average runtime per token: < 1ms)

[9] `http://www.cim.mcgill.ca/ dudek/206/Logs/AOL-user-ct-collection/`
[10] This and following runtimes refer to a successively execution on a Intel-Dual-Core-CPU i3 2GHz / 8GB RAM
[11] `http://www.davidashen.net/texhyphj.html`

4. tagging each token with its lemma:
   - TT4jLemmaAndPartOfSpeechTagger: uses TT4J-TreeTagger (average runtime per token: < 1ms)
   - MateLemmatizer: classifies lemmas based on a large margin model (average runtime per token: 11ms)
5. tagging each token with its POS-Tag
   - TT4jLemmaAndPartOfSpeechTagger: uses TT4J-TreeTagger (average runtime per token < 1ms)
   - MatePartOfSpeechTagger: classifies lemmas based on a large margin model (average runtime per token: 11ms)
6. tagging each token with a sentiment
   - *TokenSentimentLexiconAnnotator*: lexicon-based annotator
7. parsing of phrases (average runtime per sentence: 1ms)
   - StanfordParser[12] (runtime depends strongly on the complexity of the sentence)
   - TT4jPhraseChunker: TT4J-PhraseChunker (average runtime per sentence: 1ms)
8. filtering the phrases
   - *RelevantConstituentsTagger*: simple Phrase-Label-Filter (average runtime per phrase: < 1ms)
9. labelling each phrase with a score whether it exists as header in Wikipedia or not
   - *WikipediaTitleRecognizer*: uses the API of Wikipedia and a local cache (runtime depends on cache strongly, but initial runtimes of approximately a half second are common per phrase)
10. tagging each phrase with a confidence value that this phrase is the target
    - *ConstituentTargetRegressorTagger*: machine-leaning approach (runtime depends on its hyperparameters strongly)
    - *ConstituentTargetGroundTruthTagger*: a simple rule-based approach which gives that phrase a high value which fits to the ground truth target – useful for training and testing (average runtime per sentence: 1ms)
11. take a final decision with phrase is the unique target
    - *TargetTagger*: simple max-value-search (average runtime per sentence: < 1ms)
12. compute a sentiment score for the text
    - *TextSentimentLexiconAnnotator*: implements the approach of Section 3.1.1 without an extended opinion lexicon and contextual features (averageruntime per text, for example, a claim: 5ms)
    - ArguAnaGlobalSentimentScoreClassifier: SVM-based classifier

[12] `http://corenlp.run/`

- *TextSentimentStanfordAnnotator*[12]

13. compute a contrast score for the text
    - *ClaimContrastClassificationRegressor*: implements the approach of Section 3.1.1 (runtime depends strongly on its hyperparameters)

This list reveals that a second factor is essential for efficiency, too: the hyperparameters. Nearly all primitive analysis engines have hyperparameters like learning models. These parameters influence the behaviour of the engine. For example, in the ConstituentTargetRegressorTagger and ClaimContrastClassificationRegressor hyperparameters control the feature computation, which features will be activated and which learning model is used. This control results in enormous effectiveness-efficiency-deviations. One example is in the ConstituentTargetRegressorTagger: one feature is the paths in WordNet (A.1). The algorithm executes a depth-first search in the graph of WordNet. Each node (a word) has many connections to other related words. The degree of a node differs much, but let us assume, each node has a degree of n. Hence, we assume allowing to search deeply will increase the quality of the approach: the algorithm will find relations of two words which have nothing to do with each other on the first view. Such cases will be punished with a high distance value, but they will be recognized. This attitude is not the case if the algorithm quits a search too early. However, this tenacity costs lots of computation time: it is $\mathcal{O}(n^i)$ if we allow a maximum depth of i. With a degree of 50 and a maximum depth of five and two relevant tokens in the claim target candidate and the topic target, the algorithm will look up approximately $50^5 \cdot (2 \cdot 2) = 1.25$ billion words. Even with a maximum search depth of two, the algorithm will look up approximately 10.000 words in this example. There are some ideas to save computation time like saving each result of a search in a map. Because of symmetry, we store a result for two pairs. However, Table A.1 shows that this feature is very costly. Hence, it is worth to consider disabling the whole feature type.
Storing already computed results of deterministic complex time-independent functions is, in general, a desirable approach for increasing efficiency. Another example where this idea is useful is the Wikipedia-feature-type in the target identification step. Instead of asking Wikipedia for every target candidate for every request, which can result in congestion, we store every answer (Wikipedia title or not) in a map. Before shutting down the analysis engine we save it in a file. The complexity of this analysis engine will converge quickly to $\mathcal{O}(1)$ by using this strategy. Wikipedia headers tend to be stable over time. Nevertheless, in the case of Wikipedia this solution should use expiry timestamps if the system which uses the stance classifier runs over a long time (months or years).
Another critical part is the relation function, which was presented by Bar-Haim et al. (2017a) in the contrast relation step. This relation function searches for cue phrases in log files of search engines. Even with preprocessing (filtering all queries which contain only one token[13]) it is costly to filter a list of millions of strings. For example, it takes approximately a quarter second to process a list with 1.6 Million strings. It turns out that avoidance of regular expression reduces the computation time a lot. The approach which uses regular expressions needs two seconds per claim-topic-pair. The reason is the creation of the `Pattern`-Objects in Java, which must be done on demand because the pattern of the regular expression requires the search query. The replacement by an approach which uses branched `String::indexOf` can additionally be combined with further preprocessing of the log files and a more intelligent storage structure like tree structures – although, it turns out that even strongly suggested relation functions can be considered to be disabled.

[13]The cause is that the approach looks for queries which contain the *two* sides of an anchor pair, hence, at least two words

The third concept of efficiency is to replace parts – primitive analysis engines or feature types. In this case, the paper suggested already an adaptation in the previous section: instead of using morphological similarity features, which requires the slow MateMorphologicalTagger, the algorithm can choose the syllable similarity. Consider the example *syllable* and *syllableness*. The morpheme of both is *syllable*. Hence the morphological similarity is 1. The syllables are $\{syl, la, ble\}$ and $\{syl, la, ble, ness\}$ and the Jaccard similarity coefficient $\frac{|\{syl,la,ble\}|}{|\{syl,la,ble,ness\}|} = \frac{3}{4}$ which is an approximation of 1. This approximation succeeds not in all cases, for example *speak* and *spea-ker*. Although, it is a suitable similarity measure for two phrases because a high similarity coefficient means a significant overlap of the two phrases.

An overview of all adaptations of the stance classification approach resulting from this section is given in Table 3.1.

Suggested methods	**Used methods in argument search**
Claim identification step of the IBM approach (Section 3.1.1)	
1) target candidate extraction	
ESG-Praser	Stanford-ConstituencyParser or TT4J-PhraseChunker
2)Features for claim candidates	
syntactic and positional features	syntactic and positional features
Wikipedia	Wikipedia
sentiment	sentiment
morphological similarity to topic	syllable similarity to topic (*Jaccard similarity*)
paths in WordNet to topic	(paths in WordNet to topic)
cosine similarity of word2vec embeddings to topic	cosine similarity of word2vec embeddings to topic + Word Mover's Distance to topic (Kusner et al., 2015)
Claim sentiment classification step of the IBM approach (Section 3.1.1)	
1. Sentiment matching	1. Sentiment matching
2. Sentiment shifters application	2. Sentiment shifters application
3. Sentiment weighting and score computation	3. Sentiment weighting and score computation
Sentiment-lexicon-extension	no extension
+ contextual features	not used
Contrast classification step of the IBM approach (Section 3.1.1)	
1) anchor pairs	
all single tokens + phrases ⇒ max-computation returns the main anchor pair + further anchor pairs	determine exact one anchor (x_c, x_t)
2) contrast relation functions	
morphological similarity	syllable similarity (Jaccard similarity)
cosine similarity using word2vec embeddings	cosine similarity using word2vec embeddings
reachability in WordNet via synonym-antonym chains	reachability in WordNet via synonym-antonym chains
thesaurus-based synonymantonym relations using polarity-inducing LSA	–
relation function of IBM	(relation function of IBM)
3) output of the machine learning process	
Random Forest classifier: likelihood-score $[0, 1]$	K-Star/ K-nearest neighbours classifier ($K = 5$): $[-1, 1]$ for contrastive / consistent relation

Table 3.1: The table presents all considered adaptations between the approach of Bar-Haim et al. (2017a), including the extensions which Bar-Haim et al. (2017b) suggested and the approach of this thesis. The reasons for adaptations are a better fit for text peculiarities in the argument search case (cyan), a forced reimplementation or the absence of the source code, respectively, (gray), purposes of efficiency (blue) or own ideas for a extension (green).

4 Extensions of the adapted approach

Chapter 3 introduces the necessary adaptations for argument search. Hence, not only adaptations will improve effectiveness and efficiency, but extensions of the approach will, too. Some ideas need an argument search context; other ideas are helpful for stance classification in general. This chapter presents only implemented extensions. Table 3.1 shows already a part of the implemented extensions. Nevertheless, this thesis had another focus than to implement and evaluate all ideas. Hence, Section 7.2 will provide additional ideas for further work.
The most essential implemented extension refers to word embeddings (Section 4.1). Also, there are some minor extensions, most of them on the technical level. Section 4.2 lists the additional minor changes.

4.1 Word embeddings

Word embeddings are able to capture a large number of precise syntactic and semantic word relationships. For that purpose, they map each word into a vector. There exist solutions to consider idiomatic phrases, too (Mikolov et al., 2013). Therefore, the so called Word2Vec-Models gives a great opportunity to linguistic research. The thesis gives an additional weight to word embeddings in this chapter. The claim identification step and the contrast classification step (Section 3.1.1) use word embeddings.

4.1.1 Creation and use of a word embedding model

Each word embedding model was trained on a specific text corpus to learn many linguistic regularities and patterns (Mikolov et al., 2013). Effectiveness and efficiency strongly depend on the dimensionality of the word vectors. High dimensional Word2Vec-models have many possibilities to capture and model various nuances of the natural language. Hence, the downside is a very storage intensive model file with a long loading time of the model, if the majority of words should be shattered. Small models are not as effective as big models, but they are handy. Common Word2Vec-models have a vector dimensionality of 50-300.
There are two possibilities to get a Word2Vec-model. The approach of this paper provides both:

- Using already trained Word2Vec-models: there are lots of Word2Vec-models freely available. Kusner et al. (2015) used a model which was trained on the Google news and provides an embedding for 3 million words/ phrases[1].The approach of this thesis prefers

[1] `https://code.google.com/p/word2vec/`

GloVe (`https://nlp.stanford.edu/projects/glove/`). The Global Vectors for Word Representation combines the advantages of the two major model families: local context window methods and global matrix factorization (Pennington et al., 2014). However, the implementation of the approach of this thesis allows any Word2Vec model. The way to use it is to download the preferred model and to input the file path as a parameter for the class `RelationFunctions`.

- Training a Word2Vec-model: the implementation provides a Word2Vec-model-Generation-class for the case that a Word2Vec-model-reference does not point to an existing file. In that case, the implementation collects the claims that should be classified, including their context or argument's premises. This text is the training text corpus for the Word2Vec-model. Hence, the word embedding approach uses neural networks (skip-gram model (Mikolov et al., 2013)); the implementation can define a layer size. The layer size determines the dimensionality of the word vectors later. The implementation sets the minimum word frequency to consider a word in the model in dependence to the length of the text corpus. Also, the implementation provides a sentences iterator and a tokenizer. Training on the IBM corpus results in 22,019 embedded words. This size is a miniature amount of Word2Vec-models which can be found on the Internet.

4.1.2 Cosine Similarity and Word Mover's Distance

Bar-Haim et al. (2017a) uses the cosine similarity in the claim target identification step and the contrast classification step. In these cases, the algorithm processes phrases. Hence, there is a need for determining the cosine similarity over a group of words. However, the cosine similarity is originally only defined for pairs of single words in the context of word embeddings (Mikolov et al., 2013). Hence, the algorithm must aggregate the results for each word combination of the claim and the topic. A heuristic sorts the normalized similarities decently. Then, the heuristic calculates the maximal token length of claim and topic, defined as m. It takes the first m cosine similarity scores from the result list and returns the average of them. For example, the two phrases "the honour of Jesus" and "God's glory" would receive a high score because of a high cosine similarity of $(Jesus, God)$ and $(honour, glory)$. Consider the phrases "Heavenly Father" and "God". Christians call God "Heavenly Father". Hence, the cosine similarity should be high. However, because of measuring word by word, so $(Heavenly, God)$ and $(Father, God)$, the return value unfortunately is not so high.
Kusner et al. (2015) presented a novel measurement which is based on word embeddings, too. It handles the distance between text documents. The approach is useful to measure the distance (and therefore, the similarity, too) of phrases. It is not a novel problem to measure the differences/ similarity of two texts – there are already solutions like the bag of words or the term frequency-inverse document frequency. However, these measurements have drawbacks. For example, they do not capture the distance between individual words and they do not use a high semantic representation of words like word embeddings. The *Word Mover's Distance (WMD)* uses word embeddings and a similar idea like the approach using cosine distances in this thesis. The idea is words from the one text (source) travel to words to the other text (target) – the travel costs are measured by the Euclidean distance between the vector representations of the words. The distance output is the minimal summed travel costs to transfer all words from the source text (excluding stop words like "to" and "the") to the target text. Figure 4.1 gives an example. The author solved the minimization problem with a linear program. The consideration of phrase entities like "Heavenly Father" depends on the underlying word embedding model. The result of WMD is a positive real value which may be above 1. Because of that, the distance value must be normalized by the largest distance value in the training set, pruned to 1.

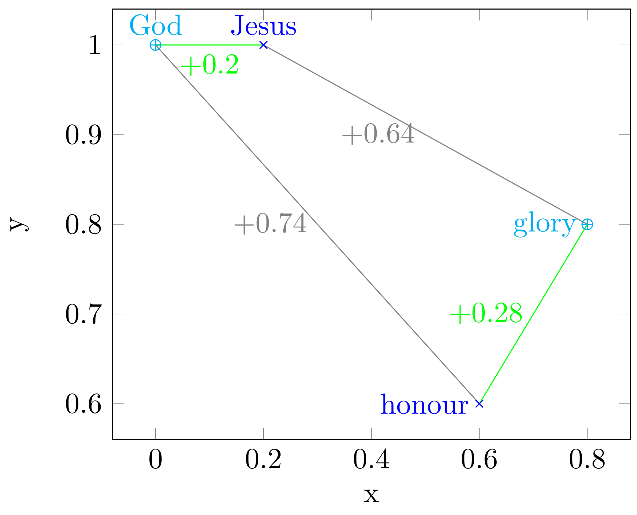


Figure 4.1: This plot, created for the paper's example in 4.1, symbolizes the Word Mover's Distance between the phrases "the honour of Jesus" and "God's glory". The algorithm filters the stop words in the first step. Therefore we consider only "honour", "Jesus", "God" and "glory". Suppose a two-dimensional word embedding; therefore, every word is represented by two coordinates in the plot. The blue words are from the source phrase, the cyan words from the target phrase. Now the source words "move" to the target words – the algorithm computes the Euclidean distance and decides to map "Jesus" to "God" and "honour" to "glory". The costs are approximately $0.2 + 0.28 = 0.48$, which is the return value.

The WMD makes 58% fewer mistakes in the text documents distance measure concerning the k-nearest-neighbourhood results. in nearly all cases WMD is more accurate than other state-of-the-art text distance measuring methods (Kusner et al., 2015).

4.2 Other minor extensions

The first minor extension is to decouple the classification of the claim target candidates and determination of the target for later use. The target has its isolated annotation, and therefore, it is possible to exchange the classification analysis engine without any side effects to the other analysis engines. There are at least two possibilities to classify a claim target candidate: with machine learning or with a heuristic by knowing the ground truth or by inferring it directly from the topic. Beside to the suggest machine learning approach the stance classification module contains an analysis engine for classifying target candidates concerning their Jaccard similarity coefficient to the ground truth or the topic. Hence, it is possible that not exactly one candidate is labelled as the true target or with a high confidence value – in this case, the analysis engine which determines the target generates or selects the final target. This two-step-approach makes the (claim/ topic) target identification step robust and flexible.
A second minor extension is mainly in the sentiment matching of the (claim/ topic) sentiment classification step. Bar-Haim et al. (2017a) suggested determining positive and negative terms with one sentiment lexicon. Later they suggested an expensive way to extend the lexicon. The approach of this thesis follows another idea. Because of the popularity of the sentiment research, there exist many sentiment lexicons in these days. The approach is to combine a set of sentiment lexicons. The implementation provides a primitive analysis engine which can consume a set of sentiment lexicons (list of words with positive and negative sentiments). Because of having a

set, a word can occur in several sentiment word lists. Because of the ambiguity and context dependence, a word may occur in some lexicons as a word with positive sentiment and in other lexicons as a word with negative sentiment. In that case, this primitive analysis engine provides four aggregation methods: the average, the majority voting, and the max- and min-function. Due to the average aggregation method, it is possible to result with real-valued sentiment values in the range of -1 to 1.
The last minor extension helps for purposes of debugging and the measurement of efficiency. Every primitive analysis engine which we implement to solve the stance classification task inherits from an abstract intermediate class. This abstract class provides an effective logger and traces the computation time of this analysis engine. This class calculates and outputs the average computation time per processed element in the end.

5 Technical overview of the implementation

This chapter gives a technical overview of the implementation of this thesis – the stance classifier. The implementation contains thousands of lines of code. We mainly used the programming language Java, but there are several configuration files (mostly in XML), too. Also, the implementation contains small lexicons which we composed manually for some subtasks. Hence, this chapter gives just an overview and a more in-depth insight into the implementation only on some points. Section 5.1 gives a structural overview from the perspective of the stance classification application. Also, Section 5.3 describes how to use the application and it gives further implementation insights.
A central part of stance classification uses machine learning. Section 5.2 gives an overview from the perspective of the underlying machine learning parts.

5.1 General overview of the structure of the implementation

The implementation uses an interlocked framework as Figure 5.1 shows. The stance classifier is embedded in the AITools4. This tool provides integration to Apache UIMA (Ferrucci and Lally, 2003). This framework supports type-systems of annotations and interfaces for creating own analysis engines. There are five used categories for primitive analysis engines. A basic category is the segmentation. Analysis engines divides the input text into useful units like the `InfexBATokenizer` in this category. Also, the following category is the tagging category. Taggers add additional information to existing annotations like tokens. An example of a primitive analysis engine is the `TT4jLemmaAndPartOfSpeechTagger`. We extended the tagging category by primitive analysis engines which create annotations for the stance classification on the base of existing annotations. One example is the `RelevantConstituentsTagger`. This tagger marks specific phrases as relevant for further processes, for example, as target candidates for the target identification step. Because of the implementation habit not to extend core types like the phrase annotation, we created a new annotation type which refers to its phrase annotation. The parsers interleave with the taggers. The parser-category contains primitive analysis engines like the `TT4jPhraseChunker`. Then, there is the categorization category. Analysis engines in this category classify units of texts or the whole input text. Examples are the `TextSentimentLexiconAnnotator` and `ClaimContrastClassificationRegressor`. The last presented category of entity recognition. Analysis engines in this category detect entities in the text like names, dates or wikipedia article headers in the case of `WikipediaTitleRecognizer`
Primitive analysis engines can be composed to aggregate analysis engines. With these engines,

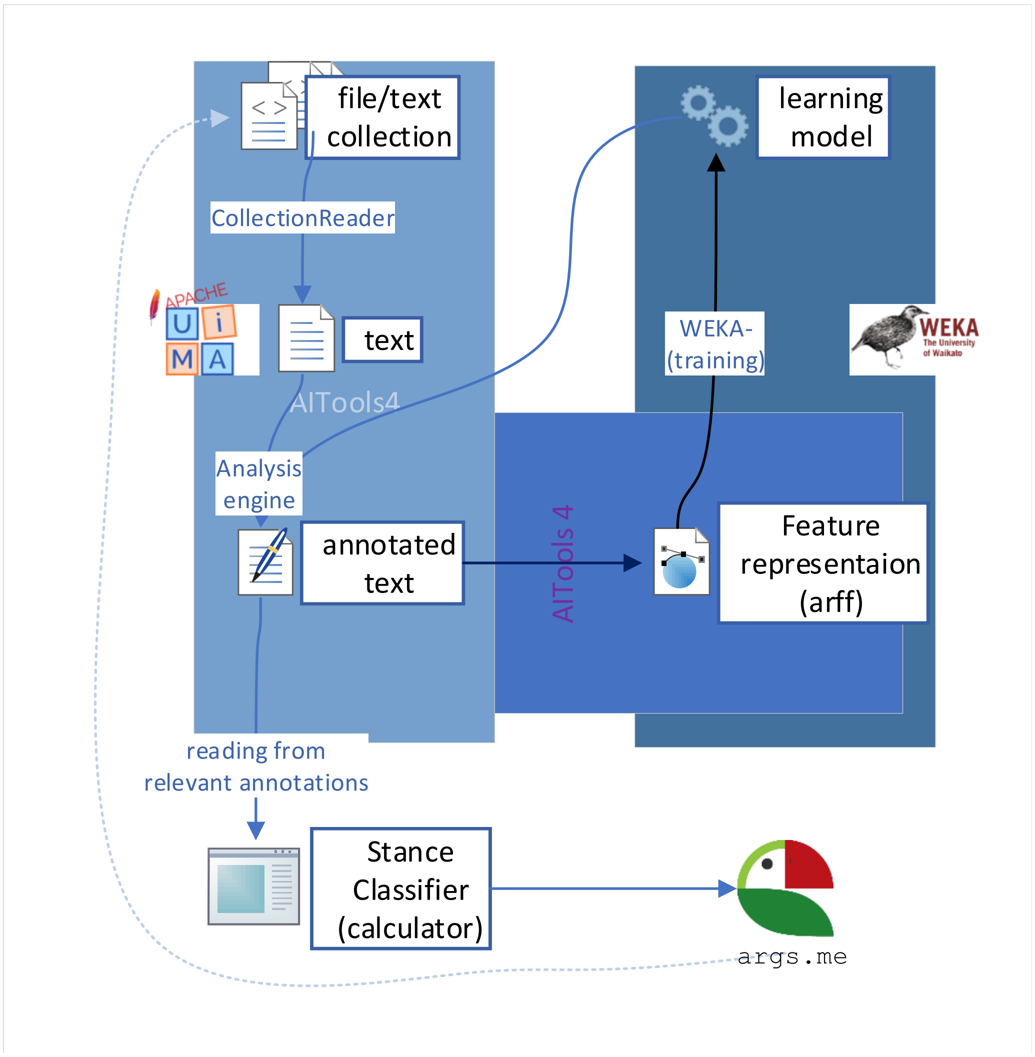


Figure 5.1: This figure shows the general structure of our implementation. Apache UIMA supports the main flow in combination with the AITools4 which contains several primitive analysis engines. Also, the feature-framework of the AITools4 helps to integrate Weka, a machine learning software. Trained learning models can be used as well as analysis engines.

there is the ability to process a raw input text to a richly annotated text including high-level annotation types which are based on each other. Section 3.3.3 lists the relevant primitive analysis engines for our approach. The standard analysis pipeline of our implementation contains the following primitive analysis engines:

1. `segmentation/InfexBASentenceSplitter`
2. `segmentation/InfexBATokenizer.xml`
3. `segmentation/TextHyphJSyllableDetector`
4. `tagging/TT4jLemmaAndPartOfSpeechTagger`
5. `parsing/StanfordParser`
6. `tagging/RelevantConstituentsTagger`
7. `entityrecognition/WikipediaTitleRecognizer`
8. `categorization/TokenSentimentLexiconAnnotator`
9. `tagging/ConstituentTargetRegressorTagger`
10. `tagging/TargetTagger`
11. `categorization/TextSentimentLexiconAnnotator`
12. `categorization/ClaimContrastClassificationRegressor`

We will adapt this aggregate analysis engine for a validation in effectiveness and efficiency in Chapter 6 later on.
The used annotation types come from the core-type-system, metadata-types, which are inspired by the approach of Bar-Haim et al. (2017a), and a type-system which contains specific annotation types for the stance classification. Examples for stance specific types are the `ConstituentAdditionalInformation`-type which stores feature-related values for the target identification, `Contrast`-type and the `ClaimInformation`-type which stores the sentiment of a claim (or the topic – each topic is a claim in itself, too). The mentioned metadata-types are `Topic` and `Claim`. These types are important for the usage of the stance classifier. Hence, Section 5.3 explains them.

Apache UIMA is not the only integrated framework as Figure 5.1 shows. The `ConstituentTargetRegressorTagger` and `ClaimContrastClassificationRegressor` analysis engines use machine learning. There are two prerequisites for machine learning: an algorithm which converts an instance, for example, a target candidate or a claim, into a vector representation. Vectors consist of feature types (Section 2.1.2). The second prerequisite is a machine learning algorithm. Weka fulfils the second prerequisite. This tool contains a collection of machine learning algorithms for data mining tasks (Frank et al., 2016). The used AITools4 provides a framework for the first prerequisite. This framework, which we extended slightly, contains the following steps:

1. feature determination step: this step introduces the feature types with their features. Every feature, that is every value in the vector representation, is labelled with a name. Hence, this step defines the number of dimensions for each vector. Also, this step searches for the most extreme value for each desired feature to have a normalization value later on. This step is only executed in the training phase.

   (a) `initializeFeatureDetermination(Properties configurationProps)` $\rightarrow$ `initializeParams(Properties configurationProps)`: this method initializes the feature engine. For example, in this step reads the input properties and loads dictionaries.
   (b) `updateCandidateFeatures(JCas uimaContainer, int start, int end)`: this method is called for every instance in the training set and updates the feature set and optionally the value bounds for normalization.
   (c) `determineFeatures(Properties configurationProps, Properties normalizationProps)`: this method finishes the feature determination step and writes the normalization values into a property file.

2. feature computation step: this step calculates the feature representation of an instance.
   (a) `initializeFeatureComputation(List<String> features, Properties configurationProps, Properties normalizationProps)` $\rightarrow$ `initializeParams(Properties configurationProps)` + `initializeNormalizationParams(Propertiesnormalizationprops)`: this step loads the properties and prepares the following step.
   (b) `computeNormalizedFeatureValues(JCas uimaContainer, int start, int end)` $\rightarrow$ `computeNormalizedFeatureValuesStep(Cas uimaContainer, int start, int end)`: this steps calculates and outputs the feature representation of an instance. The extension of the framework logs the computation time and collects statistics which can be requested.

The primitive feature engines can be aggregated like primitive analysis engines in Apache UIMA. Every primitive feature engine is responsible for one specific feature type.

5.2 Selected insights to the machine learning parts

Machine learning is an essential part of the target identification step and contrast classification step. Machine learning consumes the features which have been created before as a real-valued vector and returns a real-valued prediction. All input values are in the range of 0 to 1. Exceptions are the inputs from the relation function in the contrast classification step, which can detect contrasts directly. These functions return values in the range of -1 to 1. These limitations require an intermediate normalization step. The normalization step divides each feature value by the highest value of the feature, which has been observed in the training phase. If a higher value occurs in the testing phase than the highest value in the training phase, then the value is pruned to 1 (or -1, analogously). This situation can occur by features types which measures a distance.
Model selection and hyperparameter tuning play a critical role in machine learning. One hint to find the best fitting model, including its hyperparameters, was using AutoWeka (Thornton et al., 2013). The problem with that approach was that AutoWeka is only available to a Weka-version of 3.7.13 or higher. Because of dependencies to the used framework, the version 3.7.4 was only sensible. Hence, the version 3.7.13 enabled many models for regression tasks. So, AutoWeka suggested the most times a RandomForest-model which is not available as a regression model in 3.7.4. Nevertheless, it turns out that other available regression models are sensible, too.

The chosen model for the target identification task is a Regressor-SVM – a confidence value predictor that a particular target candidate is the right target. Hence, an output of 0 means that the model is sure that the candidate is not the target. An output of 1 expresses the confidence that the candidate is the target. All real values (degrees of uncertainty) are possible. The reason for choosing an SVM for this task is that most features are directly linearly related to the predicted confidence value. An interesting point is the weighting of the one-hot-feature set regarding the relation to sentiment tokens. Some encodings are rare, and hence, this can become a weak point for overfitting. High absolute weights gained, for example, an auxiliary relation from target candidate to a sentiment token (-1.2), a clausal complement relation ($+1$) and a nominal subject relation ($+0.57$). Another interesting fact of trained SVM was that the Wikipedia-feature-type has no or nearly no weight. Nevertheless, an SVM is a reasonable choice, here. Also, the right choice for the complexity parameter c is critical. Due to an exhaustive search it turns out that $0.1 \leq c \leq 0.6$ fits well. Another hidden hyperparameter was the threshold for the Jaccard similarity of classifying a phrase as a ground truth target phrase. Bar-Haim et al. (2017a) suggested classifying a target candidate as ground truth target for the leaning model if the word-based Jaccard similarity with the ground truth target is ≥ 0.6. Hence, the learning model learns false targets, too. However, due to this threshold, the small ratio of positive instances increases. In the validation it turns out that a higher threshold increases the mean squared error of the trained regressor even with oversampling the positive instances. The reason is the overfitting problem if the variety of positive instances becomes too small. Hence, we fix the threshold at 0.6.

The favourite learning model for the contrast classification problem was not an SVM. The reason is that two out of four relation function (in a setup where all feature types are enabled) are similarity functions and therefore not directly related to a classification in contrast and consistency. The relation function which IBM presented works accurately with a rich log base. Until yet, the used log base is not extensive, especially in respect to logged queries which contain comparative conjunctions. This relation function performs more to a step function than a smooth function. Because of this circumstance, an SVM does not outperform decision trees, the K-Nearest Neighbourhood or the K-Star. The latter two learning models archive the best effectiveness. It was essential for good effectiveness to activate weighting for the K-Nearest Neighbourhood. The inverse-distance-weight aggregation function outperforms other configurations. They output real values in the range of -1 (contrast) to 1 (consistency). However, these models are lazy learning models – this decreases efficiency. Nevertheless, the results were promising. Hence, we choose a Regression-K-Star model.

5.3 Usage of Stance Classifier

The purpose of the stance classifier is the use in the argument search engine `args.me`, like 5.1 figures. It calculates a set of stances regarding a set of claims (the conclusion of arguments) and a topic, the query. Nevertheless, the stance classifier works standalone, too. An essential part of Apache UIMA is the collection reader step which uses collection analysis engine(s) (Ferrucci and Lally, 2003). The defined collection reader gathers the content of files or a text collection and wraps them into the general Apache-UIMA-object (called *Common Analysis System* – `CAS`/ `jCAS`). A UIMA-CAS provides an efficient general object-based representation of the annotations, based on the loaded type-system. It functions as a container which holds and ships the results of the collection and text analysis engines. It provides serialization methods, too (Ferrucci and Lally, 2003).
Hence, it is essential to use collection analysis engine(s) for the usage of the stance classifier. We implemented a robust and adaptable collection analysis engine directly for this purpose: the `UIMAJSONIBMDebaterReader`. This collection reader takes JSON-files and converts them into

Claim-Topic-pairs. To archive that, a JSON-file must have the following structure:

- the text of the topic (`topicText`). Optional for topics are: the target, the ID, the sentiment and a marker whether this topic is for training (`train`) or testing (`test`) of the machine learning parts
- an array of `claims` which belong to the topic and where the stance classifier should determine a stance to the topic. Each claim should contain:
  - the text of the claim (`claimText`) *or* the following two name-value-pairs: `claimOriginalText` and `claimCorrectedText`
  - the context of the claim, if there is a context. This can be for example the article where the claim was found or the argument's premise towards the conclusion which is the claim. There are two possibilities for encoding that:
    - the plain text with `text`
    - an `article`-object which contains file references for the original file (`cleanFile`, `cleanSpan`) and the corrected file (`rawFile`, `rawSpan`)
  - the following attributes are optional: the ID, the relation to the target (-1 (contrastive) or 1 (consistent)), the sentiment of the claim towards its target (-1 negative or 1 positive), the author's stance towards the claim (*PRO* or *CON*) and the stance towards the topic (*PRO* or *CON*)

The collection reader creates a UIMA-CAS for each claim. The text of the UIMA-CAS is the text of the claim. The `Claim`-type stores all additional information like the context or information for validation like the stance. This type acts like a metainformation-container. Each `Claim`-type has a reference to a `Topic`-type. This type holds the information to the related topic. A `Topic` has a topic text, and analysis engines can process it like the claim due to an implemented handler, the `TopicAnnotationStorage`. This class replaces the preprocessing of the collection analysis engine and creates a `Claim` annotation for the topic, including a referred `Topic` annotation. We implemented it as a reflexive relation.
Hence, the current working way to use the stance classifier is to write one or more topic-claim-JSON-files[1] and inputs them into the provided utility class for stance classification.

[1]the JSON-representation of the IBM Debater dataset, which Bar-Haim et al. (2017a) presented, is working, too. The download page is `https://www.research.ibm.com/haifa/dept/vst/debating_data.shtml`

6 Evaluation

This chapter contains the evaluation of the approach of this thesis. Section 6.1 describes the experimental setup and is the base for the results which Section 6.2 presents. Section 6.3 analyses, discusses and compares the results with other approaches. This Section compares the stance classification approach of this chapter with the simple heuristic in `args.me`, for example. Also, this section analyses the results with respect to the results of the base approach (Bar-Haim et al., 2017a).

6.1 Experimental setup

Our goal is to validate the effectiveness and efficiency of the adapted approach, based on Bar-Haim et al. (2017a) (Section 3.1), in the argument search. To this end, we present two corpora in Section 6.1.1. One of them bases directly on the corpus of the argument search engine `args.me`. A further goal is to understand the influence of the components, adoptions and ideas in respect to effectiveness and efficiency. In the end the final aim is to determine a composition of the components which succeeds with the stance classification in argument search.
Hence, this section explains the complex experimental setup. The implementation offers many degrees of freedom. Every one of the three steps for stance classification, which 3.1.1 describes, offers different configuration options. Also, the analysis pipeline is flexible, too. These possibilities lead to a massive amount of specific setups, which 6.1.2 describes.

6.1.1 Two corpora for the evaluation

This chapter presents two corpora. We validate the approach of this thesis against both of them. Although, the corpus of IBM is the central one. We used a part of that corpus for training the machine learning parts.

The corpus of IBM

IBM Research presented a public corpus (Bar-Haim et al., 2017a). This corpus is split into a training set with 25 topics and 1,039 claims for machine learning purposes. The test set of the corpus contains 30 topics and 1,355 claims (Bar-Haim et al., 2017a). Although, Section 3.2 shows that there are significant differences between the setup of the IBM corpus and the argument search case. For example, we introduce a dummy value in the IBM corpus which always returns 1 to project an author's stance towards the claim. The reason is to overcome the differences that Bar-Haim et al. (2017a) classifies the stance of claims directly and not argument's premises like the argument search. Therefore, we assume that each claim is always supported, which is equal to the original task to determine the stance of a claim towards a topic. Nevertheless, the other differences can not project to the IBM dataset. Hence, this thesis introduces an argument corpus for the argument search in the next subsection.

The corpus of `args.me` – an annotated sample

The argument search engine `args.me` uses an underlying corpus with 387,606 arguments (Ajjour et al., 2019). We used this large argument resource to create a manually annotated sample of this corpus. This subset was used for testing in addition to the IBM corpus. The goal is to prove the effectiveness and efficiency of the approach of this thesis in an actual argument search case. We composed the annotated subset as follows:

Gathering of claims: Claims were gathered in the first step. In the process we first collected topics. The process results in ten controversial topics, that are the search queries for `args.me`. Five of the ten topics are search queries of the top ten search queries (Ajjour et al., 2019): "Climate Change", "Feminism", "Google", "Vegan" and "Donald trump". The other five topics were relevant, too, in that sense that users of `args.me` search at least once for each of them. The five topics are: "abolish death penalty", "abortion is bad", "refugees", "God" and "Hitler". The claim collection follows on choosing the topics. We collected ten different supported conclusions and ten different attacked conclusions, in sum 20 claims per topic, for each query.

The annotation process: Three annotators which were not involved in this thesis assessed each claim towards its topic. The task was set as follows:

> "The task is to classify the stance of 200 pairs of argument conclusion and stance towards ten different topics. For each topic, the appended Excel sheet has one separated index tab.
> In each tab, you will find the topic in row 2. column B contains the conclusions and column C the stance of someone towards each conclusion. In column D, you shall now fill in the resulting stance (*PRO* or *CON*) towards the given topic. So, you can read each row as follows: If someone has a <Column C> stance towards the conclusion <Column B>, that person is likely to have a <Column D> stance towards <topic>.
> In case the conclusion is a question, interpret the given conclusion-stance pair as you think is reasonable."

The annotators agreed with each other independently in most cases. However, there were 31 conclusion-topic-combinations where there was a disagreement. Hence, 85% of all claims have unambiguous ratings[1]. The majority opinion wins in the ambiguous case. An example of such a disagreement is the stance of the conclusion "If there is a God, then it is most definitely the Christian God." towards the topic "God". Nearly all 31 disagreements resulted from cases like the example in which it is not initially clear which stance fits. However, there were a few disagreements in unexpected cases, too. For example, the annotators voted differently for the stance of an argument's premise which supports the conclusion "Feminism is bad" towards the topic "Feminism". The majority voted for a *CON* stance. This manual annotation process shows that even for a human, it is not easy to succeed with the stance classification task. Nevertheless,

[1]Therefore, a lower bound for the human baseline is, in this case, an accuracy of 0.85, if we assume, that two annotators always agree to each other and the third one always disagrees in the ambiguous cases.

the annotated sample contains a substantial agreement – the Fleiss' kappa score is 0.79 (Brennan and Prediger, 1981).
After the manual annotation process each conclusion got a corresponding argument's premise from `args.me`. Hence, the argument's premise did not influence the decision of the annotators at all. Besides the contextual features, the annotators assessed only on the conclusion of the topic like the stance classifier.

The conversion step: Until then the Excel sheet contained the annotated data. The Excel format is not optimal for machine reading processes. Hence, a conversion to the JSON-format was the next step. The JSON-format bases on the format of the IBM format. Each topic is an upper-level node with the name-value-pairs `topicId: Number` `topicTarget: String`, `topicText: String` and `topicSentiment: Number`. Also, each topic contains an array with all its claims. A claim-JSON-object consists of `claimId: Number`, `claimText: String`, `claimTarget: String`, `claimSentiment: Number`, `targetsRelation: Number`, `text: String`, `authorsStanceTowardsClaim: String` and `stance: String`. All name-value-pairs were set automatically, including the majority voting of the annotators for the stance. The sentiment of the topic, the sentiment of the claim and the contrast relation to the target of a claim established an exception. The annotators assessed only on the basis of the author's stance towards the conclusion, the conclusion itself and the topic like the stance classifier. The stance classifier can use contextual features additionally.

6.1.2 The experiment

The experiment measures the effectiveness and efficiency on the two corpora which Section 6.1.1 describes. The efficiency is measured by the runtime on an Intel-Dual-Core CPU of the third generation with 2.0GHz and 8 GB RAM. Furthermore, we define the terms *accuracy* and *coverage* accordingly to Bar-Haim et al. (2017a). Let $\mathcal{A}$ be a set of claim-topic-pairs. Also, let $|right(\mathcal{A})|$ be the number of claims for which the stance classifier predicts the ground-truth stance. $|wrong(\mathcal{A})|$ is the number of wrong predictions. If $|\mathcal{A}|$ is the total number of claims and we assume that the stance classifier has the possibility to deny a prediction – hence, it is neither a right nor a wrong prediction then – Equation 6.1 defines accuracy and coverage.

$$\begin{aligned} accuracy(\mathcal{A}) &= \tfrac{|right(\mathcal{A})|}{|right(\mathcal{A})|+|wrong(\mathcal{A})|} \\ coverage(\mathcal{A}) &= \tfrac{|right(\mathcal{A})|+|wrong(\mathcal{A})|}{|\mathcal{A}|} \end{aligned} \tag{6.1}$$

Also, if $|positive(\mathcal{A})|$ is the number of those claim-topic-pairs which the stance classifier labels with a *PRO*-stance and $|negative(\mathcal{A})|$ those with a *CON*-stance-prediction, we define the PRO-CON-ratio as 6.2.

$$PROCONratio(\mathcal{A}) = \frac{|positive(\mathcal{A})| - |negative(\mathcal{A})|}{|\mathcal{A}|} \tag{6.2}$$

Hence, a PRO-CON-ration of 0 indicates a balanced stance classifier. A value in the range of $(0, 1]$ indicates a stance classifier which tends to predict *PRO* stances and a value in the range of $[-1, 0)$ a stance classifier which tends to predict *CON* stances.
Furthermore, all methods which use word embeddings utilize a Word2Vec model with 200 dimensions.
We will analyse the following hypotheses:

The approach which this thesis presents outperforms the current heuristic in `args.me`. To this end, we determine a suitable configuration of the stance classifier and compare the results of the stance classifier and the simple heuristic in `args.me`. For both we utilize our sample of `args.me`. Hence, the heuristic is a baseline. If we apply this heuristic to the main stance classification formula (Equation 3.5), the heuristic would always set the value 1 for the sentiment of the claim and the topic and contrast classification score.

The approach of Bar-Haim et al. (2017a) outperforms our adaptation on the IBM-corpus. However, our adaptation reaches better results on the `args.me`-corpus. The reason for this hypotheses is that some details of the approach of Bar-Haim et al. (2017a) were optimized for the IBM-corpus like the Wikipedia features. Also, the research team of the IBM-approach had more human resources and used non-public lexicons or algorithms. Nevertheless, we suppose that our adapted approach outperforms better in the argument search case. To this end, we compare the accuracy values of Bar-Haim et al. (2017a) and Bar-Haim et al. (2017b) with our accuracy values on the IBM-corpus and our sample of the `args.me`-corpus.

If we do not force the stance classifier to classify the stance of each claim-topic-pair, then the accuracy increases notably. There were some claim-topic-pairs for which the stance classification was hard even for humans. Therefore, if we discard predictions where our continuous model was unsure, we probably gain accuracy. To this end, the experiments include tests for some threshold values τ to prove what effectiveness improvement would be possible if the stance classifier has the possibility to deny the stance prediction. However, an increasing threshold will decrease coverage.

Neither in terms of effectiveness nor in terms of efficiency it is a recommendation to use all available features and subcomponents. Occam's razor describes the principle not to make a model complicated without a reason, for example in machine learning – or in other words: the simplest solution is most likely the right one (Abu-Mostafa et al., 2017). In terms of efficiency, it is trivial not to use all available features and subcomponents which consume computation time. We want to prove that only a specific subset of features and subcomponents is necessary for good predictions. Hence, we measure the effectiveness and efficiency for different configurations which disable subsets of features and subcomponents. A basic strategy for measuring the use of a single feature type in machine learning is to use the particular feature type stand-alone, then using all feature types except the certain one and finally all feature types. We consider that some feature types and subcomponents have parameters and hyperparameters by an exhaustive search. We will compare the results afterwards.
An intriguing hypothesis regarding this central hypothesis is that it is not a good idea to save computation time in the early stages of the analysis process. To prove this hypothesis, we evaluate a configuration which uses the TT4J-PhraseChunker instead of the StandfordParser.

Further hypothesis: We presented three central hypotheses in this section. Although there are lots of small research question besides or some peculiarities, respectively. We present them in Section 6.3, too.

6.2 Results

This section visualizes a small portion of the results to give a statistical insight into the answer in the next section. Values which we label with "IBM" are from this central paper by Bar-Haim et al. (2017a). Values which we labelled with "IBM+" result from the improvements which the

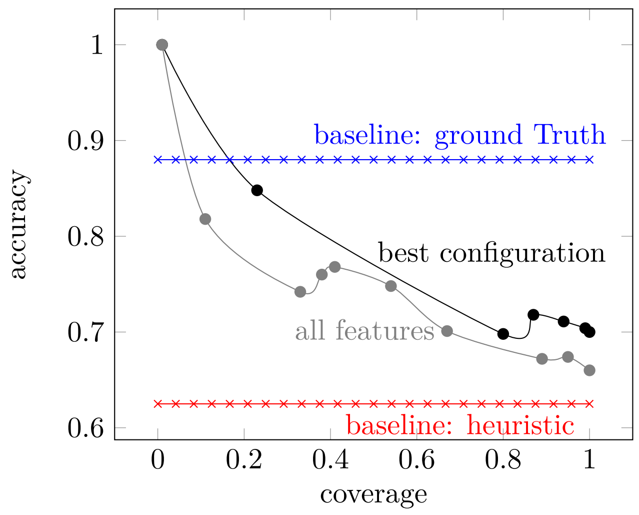


Figure 6.1: This plot compares the accuracy for different coverage levels: the preferred configuration is black. The configuration which uses all features is grey. That means all required text analysis engines including the Stanford Parser, the target identification with all features types including the WordNet-Feature with a maximum search depth of 3, and all features types for the contrast classification. Also, this plot contains both baselines: the simple heuristic in red. The upper baseline is blue. This baseline uses the value which we annotated for $s_c, R(x_t, s_t), s_t$. The used dataset is our sample of `args.me`

successor paper suggests (Bar-Haim et al., 2017b). Section 6.3 analyses these results regarding the hypothesis which Section 6.1.2 presents. For that purpose we use the best configuration which we observed in the experiments. The choice mainly considers the f_1-score and accuracy, but also the average runtime. Hence, we searched for a configuration which satisfied good effectiveness and good efficiency. The chosen configuration uses the standard analysis engine, including the Stanford-Parser. However, it does not include the contrast classification but a fixed prediction value $\mathcal{R}(x_c, x_t) = 1$. Furthermore, it uses all features-types for target identification. However, we excluded the sentiment feature type and the Wikipedia feature type. The configuration calculates the sentiment as Section 3.1.1 describes. If this suggested method does not find any sentiment token then we return a default sentiment. The default sentiment is $s_c = 0.5$ and $s_t = 0.75$.

Figure 6.2, 6.1 and 6.3 represent the results for our stance classifier. Figure 6.4 takes into account one of the three subtasks, the claim target identification. Appendix A.2 contains results for experiences with the sentiment classification module and also the contrast classification module. In general, all detailed results are listed in the appendix.

6.3 Analysis and discussion of the results and comparison with other approaches

Now, we will analyse the results concerning our hypotheses. The analysis is based on the figures in Section 6.2, but uses the results in Appendix A.2, too.

The approach which this thesis presents outperforms the current heuristic in `args.me`. Figure 6.1 shows the proof of this hypothesis. Nevertheless, the current heuristic is not a low

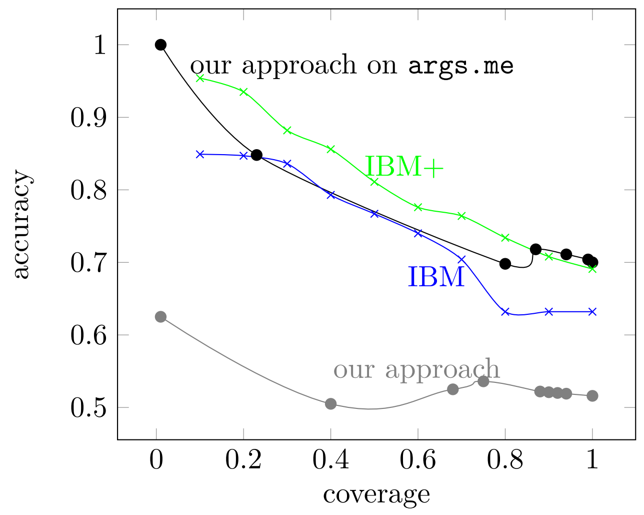


Figure 6.2: This plot compares the accuracy of the approach of Bar-Haim et al. (2017a) (including its extension in green) with the best configuration of our approach observed in our experiments. The marked dots are explicitly measured values. All other values of the graphs are approximations. The experiments used the corpus of IBM except for the graph which we labelled with “my approach on `args.me`”

baseline. The heuristic classifies 63% correctly of all argument’s premises in our sample of `args.me`, our preferred configuration classifies 70% correctly. However, there are several configurations which do not perform well to the heuristic. For example, one configuration achieves only 56%. This configuration uses our full approach but the TT4J-PhraseChunker and does not use the relation function which Bar-Haim et al. (2017a) presents in the contrast classification. Hence, the right choice of feature types and learning models is essential to beat the heuristic. Moreover, even the knowledge of the ground truth of the target does not guarantee better results than the heuristic. Nevertheless, an accuracy increase of 0.075 and an f_1-score increase of 0.08 is notable. If we apply further improvement ideas which we have not implemented until now then the recommendation to replace the heuristic with the stance classifier is convenient. Furthermore, if the argument search engine considers the confidence value of the stance classifier into the argument ranking then the advantage of the stance classifier becomes even more notable: our approach achieves an accuracy of approximately 85% for a coverage of 23% of the claim-topic-pairs. This accuracy is only 3%-points worse than our approach, which uses the ground truth knowledge in sentiment and contrast classification. An additional remark is that the simple heuristic works for the corpus of IBM, too. It achieves 52% accuracy. Our approach outperforms this in same specific configurations, but we noticed a specialisation of our approach to argument search. Moreover, the simple heuristic outperforms our approach inside the corpus of IBM in respect to the f_1-score. It always predicts a *PRO*-stance. Hence, the recall is 1. However, this is not an argument search case. Moreover, to predict always *PRO* would not be helpful for an argument search engine.

The approach of Bar-Haim et al. (2017a) outperforms our adaptation on the IBM-corpus. However, our adaptation reaches better results on the `args.me`-corpus. Figure 6.2 answers this hypothesis. The approach of Bar-Haim et al. (2017a) outperforms our approach clearly on their dataset. This observation is reasonable since Bar-Haim et al. (2017a) do not train their approach for the argument search in contrast to our approach. An interesting

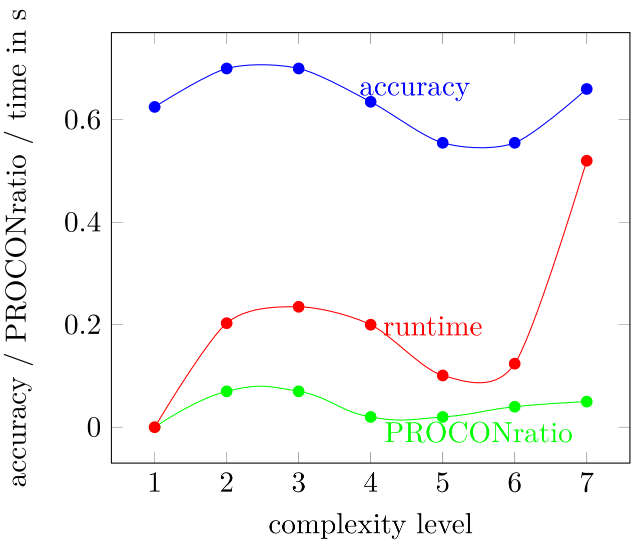


Figure 6.3: This plot visualizes the development of our approach with respect to our sample of `args.me` by adding more and more features and sub-methods. The value for effectiveness is the accuracy in blue. Besides that, this plot draws the PRO-CON-ration in green. Also, this plot adds a further red graph for the efficiency – the average runtime of the configurations in seconds. The feature additions are sorted as follows:

1. no features activated – always use default values (heuristic-baseline)
2. adding target identification without WordNet and sentiment feature type and Wikipedia feature type, no contrast classification
3. adding Wikipedia feature type to target identification
4. adding contrast classification, but without WordNet relation functions and the own approach of IBM
5. using all features for Target Identification, but only with the TT4J-PhraseChunker and add the WordNet-relation function to contrast classification
6. including verb phrases for the TT4J-PhraseChunker
7. using StanfordParser instead + using all features in target identification and contrast relation

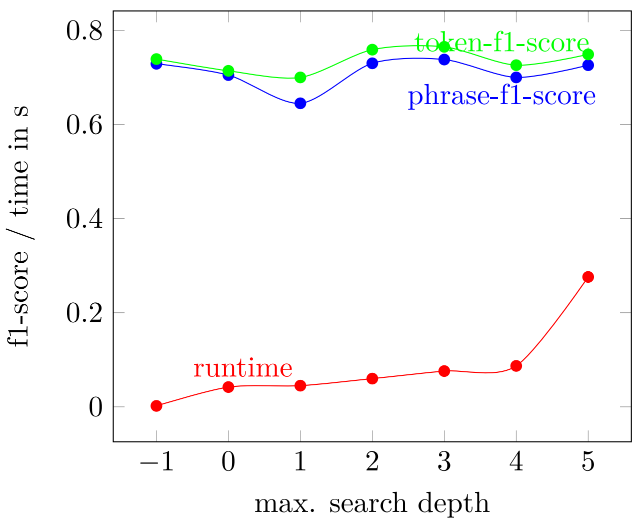


Figure 6.4: This diagram visualizes the development of the claim target identification module of our approach on the IBM corpus by adding more and more complexity. The value for effectiveness is the f_1-score on the phrase-level in blue. Besides that, this plot draws the f_1-score on the token-level in green. Also, this plot adds a further red graph for the efficiency – the average runtime of determining the target of a claim. The baseline (random guess) reaches an f_1-score of 0.2 on the phrase level. The feature additions are sorted as follows:

-1: no features which calculate the similarity/distance to the topic, no sentiment feature type

0: using all features, but the WordNet-feature-type is still disabled

1-5: the WordNet-feature-type is enabled with the specified value of the maximum search depth

$\geq$ 6: is too complex for execution on a typical laptop remark is that the accuracy of our approach does not increase notably over a decreasing coverage on the IBM-corpus. In this case our approach wrongly predicts stances in the low-confidence-area and the high-confidence-area. If the portion of remaining claim-topic-pairs is too low (for example three), then the probability of having rough fractions and outliers is high. Therefore, Bar-Haim et al. (2017a) stopped their experiment output at a coverage of 10%, which are 136 claim-topic-pairs in the case of the test set of the IBM-corpus. The part of the hypotheses which claims that our adaptation outperforms on the `args.me`-corpus is hard to falsify since we can not apply the approach of Bar-Haim et al. (2017a) directly on our sample. The implantation is not public. We know that claims and topics of the IBM-corpus are in general more linguistically complex than our sample. For example, the IBM-corpus contains claims like "Exposure to violent video games causes at least a temporary increase in aggression, and this exposure correlates with aggression in the real world" – in opposite to our sample in which many claims are like the example "Feminism is bad". Nevertheless, a stance classifier on our sample must be robust against other challenges (Section 3.3). However, we assess that the extended approach of IBM would beat our approach at least in the low coverage areas, too, although it is possible that the aspect of efficiency can be a pitfall for the approach of Bar-Haim et al. (2017a).

If we do not force the stance classifier to classify the stance of each claim-topic-pair, then the accuracy increases notably. Figure 6.1 shows hints for this hypothesis for our sample of `args.me` and Figure 6.2 shows it for the IBM-corpus. The increase of accuracy is not so notable as in the approach of Bar-Haim et al. (2017a), especially on the IBM-corpus. Nevertheless, there is an overall increase – at least a stepwise one. Both skips in the two curves in Figure 6.1 are in the threshold area of $[0.35, 0.4]$. We have set a default sentiment of our sentiment classification module to 0.5 and 0.75. The product of both is 0.375. Hence, the ratio of default sentiment is approximately in the threshold area of $[0.35, 0.375]$ on its highest point. So, the influence of default sentiment predictions affects the accuracy value strongly. A threshold higher than 0.375 discards all the default cases – a higher threshold results in lower coverage. Hence, the accuracy with a lower coverage is free from the influence of default ratings. The accuracy increases with decreasing coverage. Nevertheless, having an enabled default sentiment is usually recommended to avoid zero-predictions because of multiplying $s_c = 0$ or $s_t = 0$.

Neither in terms of effectiveness nor in terms of efficiency it is a recommendation to use all available features and subcomponents. Recommendable figures for this hypotheses are 6.3 and 6.4. Hence, even the best-observed configuration which we use does not use all available features and subcomponents. Using the full approach with all features including maximum search depth of three in the WordNet-feature-type decreases the accuracy by 0.04 and it increases the computation time by 156% in comparison to our best configuration in our experiments on our `args.me`-sample.

In terms of effectiveness: As we supposed, a complex model does not always lead to better results. The target identification step does not show this obviously. However, the best f_1-score on the IBM-corpus achieves the feature type selection which excludes the WordNet-Feature and the sentiment feature type. In general, the one-hot-encoding of the sentiment feature type often leads to loose generalizations. The reason is the variety of encodings in therefore a weak point for overfitting. In our training the dimensionality increases by 24, if the sentiment feature type is activated. Hence, there exist encodings for which only one positive or negative instance exists in the training instance set. That is not a promising representation of the general ground truth function. For example, adding the sentiment feature type decreases the f_1-score by 0.05 (and on the token level by 0.06). The use of the Wikipedia feature type is interesting, too. The

enabling of this feature type slightly increases the f_1-score on the IBM-corpus but can slightly decreases exclusivity f_1-score on our `args.me`-sample.
Another impressive result of the experiments is that enabling a feature type only to a small degree (like a small maximum search depth in WordNet) can be rather harmful than helpful (Figure 6.4). Also, the same effect can happen if a feature type gets too many degrees of freedom. A maximum search of 4 in the WordNet-feature-type harms the prediction because a distance of 4 can predict similarity although there is no similarity. For example, it is possible to get from "feminism" to "accuracy" via "libber" and "Truth". However, the trade-off of complexity becomes evident in the contrast classification. It is more effective to disable the contrast classification and always predict consistency than having a complex module for that. Even the approach of Bar-Haim et al. (2017a) shows better results for the majority voting than for their contrast classification approach in respect to full coverage. The argument search engine `args.me` increases the effectiveness of the majority baseline. The full-text-search matches primarily consistent conclusions. Hence, our sample of `args.me` contains only four contrastive cases (2%). Moreover, the annotation of a contrastive targets relation is doubtable in two out of four cases, for example "Obama" to "Abortion". Therefore, Figure 6.3 shows a decrease in accuracy at the point where we added the contrast classification. This figure shows a hint for the hypothesis that saving computation time in the early stages can lead to notable accuracy losses in the final result, too. Using the TT4J-PhraseChunker instead of the Stanford-Parser decreases the accuracy by 2-8%-points, depending on which corpus is in use.

In terms of efficiency: The results are surprising at this point. First, the runtime measurements show fluctuation. For example, the processing time is 229ms to create a feature representation of one claim-target-pair, given all required text analysis annotations, in the contrast classification if we consider all feature types. However, excluding the WordNet-features-type increases the measured time by 16ms. This measurement is unexpected. A reason for that is probably the workload of the CPU. The stance classifier is not the only running process, and even if we try not to change the process diversity it is hard to get an environment without fluctuations. Moreover, used services have different response times, too, for example in the case of the Wikipedia-Feature-type. Nevertheless, our runtime measurements are reliable on average. For example, they shows that an increase in the maximum search depth in WordNet disproportionately increases the runtime per claim. In general, we proved our hypothesis as a rule of thumb: a more complex model requires more computation time. In Figure 6.3 the valley in the graph in the complexity level five to six is interesting. We used the TT4J-PhraseChunker instead of the StanfordParser in these complexity levels. The reason is that parsing the claim or the topic is one of the most time-consuming steps in the text analysis preprocessing and the TT4J-PhraseChunker is way more efficient. In general, in the most configurations the stance classification with its three modules is not the most time-consuming part, but the text analysis preprocessing steps like tokenizing, parsing and general linguistic tagging. For example, the preprocessing of a linguistically complex claim takes approximately 200ms and even less complex claims can cost 100ms.

Further hypothesis and analysis: The results and logging outputs reveal way more insights into the processing and understanding of stance classification. We will now present a few further points:

Target Identification: The target identification is not a trivial task in linguistically complex sentences. For example, the Stanford-Parser finds eleven noun phrases (target candidates) in the claim "Exposure to violent video games causes at least a temporary increase in aggression and this exposure correlates with aggression in the real world" in the IBM-corpus. Hence, the probability of getting the right phrase by chance is 9%. The high number of target candidates results in requiring of having a very strong target identification module. However, conclusions and queries often are linguistically not very complex in the argument search case. For example, "Feminism is bad" contains only one target candidate which is the true target. Hence, we can save computation time in this module. For example, the f_1-score of taking only syntactic and positional features is only by 0.02 worse than taking the best configuration in this case. Nevertheless, we can save 94% of computation time for calculating the feature representation.

Sentiment Classification: This step is the weak point of our approach. Two out of four variables in the central stance formula (Equation 3.5) are sentiment values. Just one mistake in the sentiment classification can lead to a wrong stance prediction. For example, if we predict all values correctly while miscalculate the sentiment of the claim (negative instead of positive and vice a Versa) then the stance prediction is wrong. Unfortunately this central module does not achieves outstanding accuracy values. The best configuration calculates the right sentiment in only 68% of the cases on the IBM-corpus. The majority baseline is 56.2%. Hence, (Bar-Haim et al., 2017a) concentrate on the following paper to improve that module (Bar-Haim et al., 2017b). The fact that the promising sentiment approach of Stanford does not reach the accuracy of the majority baseline is fascinating. One problem was the handling of neutral predictions and the issue that Stanford predicts the overall sentiment, but not the sentiment regarding the target. A further interesting observation is that our approach tends to label a claim/ topic with a positive sentiment because of the default sentiment. This tendency becomes evident by looking at the PRO-CON-ratio of 0.266 on the IBM-corpus of the configuration which uses default values for all steps except the sentiment calculation of the claim.

Contrast Classification: The relation function which Bar-Haim et al. (2017a) presented depends strongly on a good database. This relation function with our database does not outperform well. Hence, the use of this stand-alone relation function results in an f_1-score of 7% on the IBM dataset. However, the precision of our approach is very promising, achieving a precision pf 92% by using the best configuration on the IBM-corpus.

Whole approach: It seems that linguistically complex databases lead to more specific behaviour of the stance classifier after training the machine learning parts. For example, the PRO-CON-ratio is far more variable in the case of the IBM-corpus than of our sample of the `args.me`-corpus.
Another interesting observation is that our approach has outliers with respect to efficiency. For example, our approach proceeded stance predictions for a claim-topic-pair in 14ms, another pair in 9.6s on our sample of `args.me`. One reason is technical fluctuation. Another reason is the varying linguistic complexity. For example, one claim-topic-pair in our sample is "Feminism"-"Feminism".
Also, it is interesting to observe that the initial stance predictions take more time than stance predictions after a longer runtime of the stance classifier. For example, discarding the first 10% of runtime-measurements of claim-topic-pairs decreases the average computation time by approximately 100ms. We expected such an effect. Our implementation mainly causes this increase in efficiency over time because of the principle of storing already computed results of deterministic complex time-independent functions. We have already presented the example of our `WikipediaTitleRecognizer`-analysis-engine: the engine uses a map-structure as cache. Hence, phrases which have appeared beforehand will not in general start a new HTTP-request but the engine can label them in $\mathcal{O}(1)$. This behaviour is especially helpful for common noun

phrases like “we”. Another example of an efficient implementation in long-time-runs is our implementation of the `TopicAnnotationStorage` which stores, also in an efficient HashMap, already processed topics with their annotations. Hence, our implementation only calculates the annotations for a query once. Moreover, if a user enters a topic which has was been already queried then the stance classifier has the annotations in $\mathcal{O}(1)$.
Another interesting observation is the slight increase of the f_1-score and of the accuracy in one configuration series where we have modified the target identification module. We have modified it by using the ground truth first and afterwards predicting it without feature types of sentiment and WordNet. This unexpected behaviour appears while using our sample of `args.me` and a default value for the contrast classification. Because of this, only the sentiment classification requires target identification. By analysing the logs, it turns out, that the target prediction marks in rare cases the whole claim, phrases like “we” or a possibly long other phrases. Our sentiment classification module ignores sentiment tokens inside the target. Therefore the target sometimes hides misleading sentiment tokens and leads to a correct stance classification. However, we suppose that this anomaly might be a kind of statistical noise and might disappear with a broader sample – or further improvements of the sentiment classification module.
Furthermore, we observed that our approach mostly had difficulties with claim-topic-pairs in which the claim compares things like “Hitler was Worse than Stalin” towards ”Hitler” (example of our sample of `args.me`). Also, our approach had difficulties in predicting the stance in cases which are hard for humans, too. That refers especially to claim-topic-pairs, where the annotators disagree with each other.

7 Conclusions

in this final chapter our work will be summarized and reflected. We want to answer the question, whether we succeeded in building a suitable stance classifier for the argument search. Is that stance classifier able to sort the fetched argument's premises to the correct side – *PRO* or *CON*? To that end, we look again on the results of the previous chapter. We summarize our work on this base in Section 3.3. This section presents some additional remarks to improve efficiency.
The long working process on this thesis resulted in a complex application. However, we have some additional ideas which were, however, too time-consuming to integrate or not in the scope of stance classification alone. Section 7.2 gathers all noteworthy ideas we have to improve effectiveness for future work.
There is the Section 7.3 at the end. That section rounds off this thesis and comes back to a broad view on the stance classification problem. It contains the final remarks and reflects the whole work.

7.1 Summary

In the previous chapter we evaluated our approach to present an intelligent solution for stance classification in the argument search. We notice that in the argument search engine `args.me` the current heuristic is a very efficient approach for stance classification, which results in an accuracy of 63% on our sample of `args.me`. Our approach increased accuracy by 0.075. The simple heuristic also reaches 52% on the IBM-corpus. This is 0.11 worse than the accuracy of the approach of Bar-Haim et al. (2017a). In the end, it is the decision of the provider of an argument search engine to use either a simple, very efficient heuristic or our approach, which is more effective. On the one hand our approach would increase the satisfaction of users who search separately for *PRO* or *CON* arguments. If a user is interested, for example, in *PRO* arguments and reads a long argument which is labelled as *PRO,* just to realize in the end that the argument is a *CON*-argument then the user can be quickly annoyed. On the other hand the search engine must deliver a reply on demand. Hence, the stance classifier must calculate all requested claim-topic-pairs efficiently, too. There is a problem exactly at this point. Brutlag et al. (2008) shows that latency of three seconds is critical and more than five seconds fatal for daily use. Hence, we inferred that we should not exceed 150ms for each claim-topic-pair. Our configuration exceeded this time (203ms). There were only a few configurations which were below this time boundary. Some of them partly use the ground truth data (our manual annotations). Hence, they are not practical for use in the running system later. Also, the other

fast configurations use the TT4J-PhraseChunker, which leads to smaller accuracy and f_1-score-values than the simple heuristic. Hence, the simple heuristic is more effective and efficient in this case. However, the computation time of 203ms in our approach is not as critical as it seems. We have already discussed the effect of efficiency converge. Because of the efficient storage of intermediate results our approach becomes faster by time. Commonly processes run a long time in servers. Moreover, a common modern server mostly provides notably more computation power than the laptop on which we evaluated our approach. Also, we notice that most of computation time is not consumed by the stance classifier itself in the many configurations but by the engines preprocessing in text analysis, especially the Stanford-Parser for long argument's conclusions. If preprocessing happens in the indexing process, which does not have to compute on-demand, then we can save computation time, too. Apache UIMA provides possibilities to store and load annotated and therefore preprocessed objects (Ferrucci and Lally, 2003). On top of that, we observed that in the majority of cases in `args.me` the stance classifier does not get 20 different conclusion-query-pair, but only a few. Conclusions overlap, and there are mostly several argument's premises for one conclusion. For example, the top-20-results for the query "feminism" contain only two different conclusions: "Feminism" and "Feminism is not sexist."[1]. Also, the server has the opportunity to parallelise the different stance computations. Hence, the efficiency problem of our approach is not as critical as it seems at first sight and is furthermore a solvable problem in the end.
Hence, we presented a suitable solution for the stance classification in argument search. We looked at the differences between the approach of Bar-Haim et al. (2017a) and the argument search case and integrated the approaches to face the differences. We used the novel idea to separate the stance classification into three subproblems, too: the claim target identification (which we extended to the topic, too), the claim sentiment classification (which we also extended to the topic) and the contrast classification. In the process of adaptation and evaluation we discovered a weak point in this novel idea: this idea only works if all submodules work correctly. If only one of the three modules is weak then the whole stance classification process is affected. For example, if the target identification module does not work well then it tags a false phrase as the target. The two other modules work with the false target and will probably deliver wrong results. For example, if an argument concludes "Clinton is better than Trump", and the ground truth target is "Clinton" but the first module tags "Trump" as the target, the claim sentiment classification process will return -1 instead of 1, which would be the right answer. The same problem holds for the sentiment classification and the contrast classification module. Also, if only the last two modules are very unsure with their prediction and output a value which is almost 0 then the complete result will be very unreliable. Moreover, even if all modules are relatively sure, for example, they output the value 0.8 then the final result is not absolutely speaking: $0.8^3 = 0.512$. Nevertheless, the multiplication of the single values is sensible in the end – it is an excellent method to reduce return values to a single stance variable, which includes a confidence vote.
Nevertheless, our approach, which bases on this novel idea, is promising in the case of argument search. We presented a solution which is compatible with the common argument model (Wachsmuth et al., 2017) with our adapted Equation 3.5 and with useful extensions, for example, in the word embeddings area: the Word Mover's Distance. Our approach uses lots of methods and approaches like machine learning (mainly SVM – in the target identification and contrast classification), lexicon-based algorithms (in the sentiment classification), deep learning with word embeddings and lots of non-public methods which we reimplement and make suitable for the argument search. We faced the issue with claims and topics which are not correct sentences linguistically. We extended primitive analysis engines to prevent exceptions. We in-

[1] `https://www.args.me/?q=Feminism` (15.07.2019)

troduced default routines and values and handled punctuation mistakes in claims or queries, respectively, to a certain degree. Furthermore, we shows that the robust Stanford-Parser, which is slower in general than the TT4J-PhraseChunker, is an excellent choice for the argument search case. It builds a good base for the target identification module. Hence, we tried to face lots of challenges in natural language processing. We faced the challenge of negation in natural language in the sentiment classification step, too. Bar-Haim et al. (2017a) introduced the idea of a sentiment shifters lexicon. We used this idea to detect the various expressions of negations in terms of sentiment. For example, the negation word "not" inverts the sentiment of "bad" in "Clinton is not bad" and therefore the sentiment towards the target "Clinton" is turned into positive.

To summarize all these insights into the research: we succeeded in building a suitable adaptation of the approach of Bar-Haim et al. (2017a) concerning the variance of conclusions and queries on the Internet and concerning efficiency. Our non-commercial implementation imports lots of public methods and analysis engines and also implements lots of other recommended procedures. We considered adaptation and extensions for improving effectiveness and efficiency.

7.2 Outlook and future work

Stance classification will play a significant role in future, too. In the case of `args.me` the stance classification implementation described in this thesis is meant to replace the simple heuristic which is used currently. Hence, we programmed the implementation related to this thesis in such a way that the integration is only minor work.
Chapter 4 presents some extensions to the approach of Bar-Haim et al. (2017a). The goal is to optimize the stance classification for the use in argument search. This section discusses some additional ideas to potentially improve the algorithm. These ideas are not implemented or validated. However, they are based on the related work and the experiences with the argument search.

7.2.1 Including contextual features

Bar-Haim et al. (2017b) have already suggested to include contextual features for the claim sentiment classification. However, the presented approach is not applicable to the argument search case because of the lack of a syntactically direct textual context of the claim in the argument's premises. Although the web crawler of the debating portal (Wachsmuth et al., 2017) stores other contextual features like the discussion which contains the processed argument
On the one hand the keywords of the request appear in many cases inside the argument's premises. One idea is to consider the sentiment of the sentences which contain a central keyword from the query. For example, if a user searches for "feminism" and there is a post of second user who attacks and denigrates feminism ($s_c = -1$), then the second user is likely to have a negative sentiment in the sentences he writes about feminism.
On the other hand we can make use of the other contextual sources. 87%[2] of all arguments `args.me` uses come from a debate portal where every user who has created a profile has writing access (Ajjour et al., 2019). Every profile is public and contains information about other debate topics the user was involved. This overview is valuable information: based on the idea of Hasan and Ng (2013) there is the possibility to apply author constraints, too. A practical implementation approach is to request the author and his related debates including his stance

[2]This percentage based on the upcoming update of the `args.me` database. The current argument index (July 2019) percentage is 72% (Wachsmuth et al., 2017)

towards the debate topics for each *PRO/CON* argument (which is in the common argument model of Wachsmuth et al. (2017) a *PRO/CON* premise of an argument)) in the indexing process. If the crawled website does not offer that then we suggest to store an empty list of the author's debates. Hence, every argument's premise in the common argument model has, in addition, an opinion vector containing further debates topics. When the stance of an argument's premise is requested the algorithm can search through the opinion vector. If there is an entry which matches exactly the query or is very similar to it, then the stance to the entry can be considered in addition to the output of the main approach (Chapter 3). The similarity can be measured with the Word Mover's Distance, for example. If an entry has a Word Mover's Distance below a certain threshold this entry has a (weighted) stance vote. Consider the query "feminism is an illness". Now we have a claim (debate title) "women's quota is necessary" with a PRO argument which contains several times the word "feminism". Hence, it is retrieved. However, it is hard to resolve the claim toward the query. Consider that the author wrote a CON argument to another debate with the title "feminism is cancer". Word Mover's Distance would recognize a small distance because of two identical words and a word pair which is related. Hence, the user argues against "feminism is cancer" and therefore probably against "feminism is an illness" in the debate with the claim. We detected the right stance without processing the claim "women's quota is necessary". Also, we can infer all other stances of the argument's premises (PRO/ CON arguments) of this particular debate with that claim now.
A practical realisation of this approach based on the author's constraints can be to predict the real-valued stance of the claim-topic-pair as the first feature for a machine learning process. The second feature is the number of entries in the opinion vector, for which the Word Mover's Distance is below a certain threshold. A third feature can be the average of all stances of those entries. To receive a more realistic value we suggest to inverse-weight the entries with their distances to the topic and the time difference to the argument's premise the stance classifier currently processes. This second weighting has its reason in the behaviour that the probability of an opinion change increases with the time difference. This implementation is compatible with claims from debate portals which do not provide such user-debate-history-references. In this case, the second and third feature would be 0.
On top of that, the indexing process can try to generate a complete network of for and against groups of users as Stede and Schneider (2018) explain.

7.2.2 Additional features types in the three single subtask of the basic approach

Each step in the original approach of Bar-Haim et al. (2017a), which is presented in Section 3.1, may be improved by adding additional concepts. However, each additional feature may decrease efficiency. Therefore it is essential in the future work to find out which effectiveness improvements are possible without decreasing efficiency notably.

Additional features types for the target identification step: Most arguments `args.me` use come from `https://www.debate.org` (Ajjour et al., 2019). Hence, that debate portal may be helpful in the target identification subtask because it fits directly into the argument search case. Bar-Haim et al. (2017a) already suggested taking into account the Wikipedia header occurrences as a feature type by predicting the target confidence value of a claim candidate. Based on that there is the idea to use the search service of `https://www.debate.org`, too. The algorithm can enter the candidate phrase via the advanced search and can restrict the search range to topics. We use topics as claims. If the algorithm inputs a controversial phrase like "feminism", he receives lots of debates (217). Therefore, it is an indicator that this phrase (containing one word) is the target. If the algorithm inputs another phrase like "an advantage", it receives only 18 results. This number is reasonable because this phrase is probably not the target of a claim or topic.

Additional features types for the sentiment classification step: The problem with the approach of Bar-Haim et al. (2017a) is, that this approach needs an extensive token sentiment tagger. If the tagger labels no token in a claim with a sentiment, all following steps inside the sentiment classification are useless. Bar-Haim et al. (2017b) already suggested a solution to extend the sentiment lexicon the token sentiment tagger uses. Also, Section 4.2 provides an alternative way to attenuate the problem. Nevertheless, there is a third option to face the problem. The approach is to calculate the sentiment prediction with the sentiment classification of our approach. Then, in addition, other already existing sentiment classifiers which get the whole claim/ topic as input for predicting a sentiment value. On the base of the evaluation of Bar-Haim et al. (2017a) we can assume that ignoring the unique role of the target to predict the sentiment score is not that critical. If we use x additional sentiment classifiers then we have $x + 1$ predictions. Hence, if we train an SVM, for example on these $x + 1$ predictions then we may have a more accurate and robust sentiment classification.

Additional features types for the contrast classification step : Apart the full implementation of this step including the full anchor pair generation we can try to increase the effectiveness and efficiency in the detection of the contrastive word. It is more difficult to detect contrastive pairs than consistent pairs because there exist many similarity functions for consistency. Until to now, we use a WordNet and the cue phrases feature for directly contrast detection. However, the WordNet do not contain all contrastive relations and quickly become inefficient. The cue phrases feature type needs a rich query log database and tends to be inefficient, too. There is potential to increase efficiency by applying more intelligent data structures. However, an interesting attempt can be the use of word embeddings. Hence, vector operations on word vectors often preserve semantic relationships (Kusner et al., 2015) like $vec(\text{healthy}) - vec(\text{cancer}) + vec(\text{good}) \approx vec(\text{bad})$. This example of contrastive relationships leads to the attempt proposal to use word embeddings for detecting contrastive words or even contrastive phrases for future work. Moreover, Trask et al. (2015) presented an accurate and fast method for solving the problem of word sense disambiguation. The sense of a word strongly depends on its context, for example, the POS tag. Therefore it determines whether a word with its sense is consistent, contrastive or none of both to another word. Hence, a fitting Word2Vec model and such an approach can improve the effectiveness and efficiency of the contrast classification step notably.

7.3 Reflection and conclusion

This thesis is a contribution to the research on stance classification. This research topic is not novel. Section 2.2 has already reported about the related work. The tendency is an increase of the accuracy / f_1-score of the approaches over time. However, it is hard to compare the accuracy / f_1-scores of the many different approaches. Stance classifiers would suffer from a lack of portability if the classifier were trained on a specific domain (Vilares and He, 2017). The writings and expression styles differ from author to author and from group/domain to group/domain (Khan et al., 2016). For example, congressional debates differ from debates in public forums (Hasan and Ng, 2013) which are relevant for the case of argument search. Even our approach shows notable differences in applying it to the case of our central paper (Bar-Haim et al., 2017a) and the argument search case, the `args.me` search engine.
On the one hand, we did not have to face issues which other areas of application have to. For

example, implicit argumentation was out of our scope. On the other hand, we faced challenges which other stance-classification-approaches do not have like incorrect natural language and the on-demand-requirement. We presented a combination of effectiveness and efficiency explicitly for the first time in the stance classification research so far. We analysed trade-offs and discovered the side-effects of replacing parts in the text-analysis-pipeline, which includes machine learning. We applied different learning models and configuration to determine a suitable result. Hence, we contribute a bunch of ideas to stance classifications in argument search. We expect that our approach will improve accuracy of the stance labels in argument search engine `args.me`. This improvement will lead to increased user satisfaction. Furthermore, the implementation of the other ideas may improve accuracy even more. We trained our stance classifier on the English language, but we can adjust it for even more languages, too. All these improvements may increase the popularity of this search engine. The typical Internet user does not visit argument search engines until now. We hope that `args.me` (or/ and other argument search engines) will change that in the future. This change is probably the base for a new experience with computational argumentation for the everyday user. The aim is that computers learn step by step to help the standard user in his inspections of persuasions or decision makings. It becomes more and more relevant to provide a balanced picture of controversial topics with *PRO* and *CON* arguments. Argumentation stays important. This relevance especially yields in our days with used phrases like “fake news” and “filter bubbles”. Stance classification in argument search is an essential part of this vision. Besides that, a good stance classifier in argument search can be the base for applications on top of that. An application which generates argumentative texts becomes possible, for example.
All in all, the user-generated content, including arguments, increases day by day. Stance classification helps to structure and sort that content and partly brings a system into reality which provides more and more information with increasing accuracy. This thesis is a contribution to that research process, which may help in further research.

Bibliography

Yaser S. Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. 2017. *Learning from data.* AMLBook.

Aseel Addawood, Jodi Schneider, and Masooda Bashir. 2017. Stance Classification of Twitter Debates. In *Proceedings of the 8th International Conference on Social Media & Society - #SMSociety17.* ACM Press.

Yamen Ajjour, Henning Wachsmuth, Johannes Kiesel, Martin Potthast, Matthias Hagen, and Benno Stein. 2019. Data Acquisition for Argument Search: The args.me Corpus. In *KI 2019*, Kassel, Germany.

Roy Bar-Haim, Indrajit Bhattacharya, Francesco Dinuzzo, Amrita Saha, and Noam Slonim. 2017a. Stance Classification of Context-Dependent Claims. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 251–261, Valencia, Spain. IBM, Association for Computational Linguistics.

Roy Bar-Haim, Lilach Edelstein, Charles Jochim, and Noam Slonim. 2017b. Improving Claim Stance Classification with Lexical Knowledge Expansion and Context Utilization . In *Proceedings of the 4th Workshop on Argument Mining.* Association for Computational Linguistics.

Avrim L. Blum and Pat Langley. 1996. Selection of relevant features and examples in machine learning. Artificial Intelligence 1997, 245-271. Received October 1995.

Robert L. Brennan and Dale J. Prediger. 1981. Coefficient Kappa: Some Uses, Misuses, and Alternatives. *Educational and Psychological Measurement*, 41(3):687–699.

Jake D. Brutlag, Hilary Hutchinson, and Maria Stone. 2008. User Preference and Search Engine Latency. resreport, Google, Inc.

John G. Cleary and Leonard E. Trigg. 1995. K-Star: An Instance-based Learner Using an Entropic Distance Measure. In *Machine Learning Proceedings 1995*, pages 108–114. Elsevier.

Adam Robert Faulkner. 2014. *Automated Classification of Argument Stance in Student Essays: A Linguistically Motivated Approach with an Application for Supporting Argument Summarization.* phdthesis, City University of New York.

Christiane Fellbaum. 1998. *WordNet – An Electronic Lexical Database.* 2. The MIT Press – A Bradford Book, Cambridge.

David Ferrucci and Adam Lally. 2003. Accelerating corporate research in the development, application and deployment of human language technologies. In *Proceedings of the HLT-NAACL*

2003 workshop on Software engineering and architecture of language technology systems. Association for Computational Linguistics.

Eibe Frank, Mark A. Hall, and Ian H. Witten. 2016. *Data Mining: Practical Machine Learning Tools and Techniques*, fourth edition, chapter The WEKA Workbench. Morgan Kaufmann. Online Appendix.

Kazi Saidul Hasan and Vincent Ng. 2013. Stance Classification of Ideological Debates: Data, Models, Features, and Constraints. In *International Joint Conference on Natural Language Processing*, pages 1348–1356, Richardson, TX 75083-0688. Human Language Technology Research Institute, University of Texas at Dallas.

Daniel Jurafsky and James H. Martin. 2008. *Speech and Language Processing.* PRENTICE HALL.

Muhammad Taimoor Khan, Mehr Durrani, Armughan Ali, Irum Inayat, Shehzad Khalid, and Kamran Habib Khan. 2016. Sentiment analysis and the complex natural language. *Complex Adaptive Systems Modeling*, 4(1).

Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, and Kilian Q. Weinberger. 2015. From Word Embeddings To Document Distances. In *Proceedings of the 32. international conference on machine learning*, volume 37, Lille, France. Washington University in St. Louis, JMLR: W&CP.

Andy Liaw and Matthew Wiener. 2002. *Classification and Regression by randomForest.*

David Meyer. 2018. *Support Vector Machines – The Interface to libsvm in package e1071.* FH Technikum Wien.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. *Computation and Language*, 1.

Saif Mohammad, Parinaz Sobhani, and Svetlana Kiritchenko. 2017. Stance and sentiment in tweets. *ACM Transactions on Internet Technology (TOIT)*, 17(3):26.

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up? Sentiment Classification using Machine Learning Techniques. In *ACL-02*, pages 79–86.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

J. R. Quinlan. 1986. Induction of decision trees. *Machine Learning*, 1(1):81–106.

Pavithra Rajendran, Danushka Bollegala, and Simon Parsons. 2016. Contextual stance classification of opinions: A step towards enthymeme reconstruction in online reviews . In *Proceedings of the Third Workshop on Argument Mining (ArgMining2016).* Association for Computational Linguistics.

Sarvesh Ranade, Rajeev Sangal, and Radhika Mamidi. 2013. Stance Classification in Online Debates by Recognizing Users Intentions. Language Technologies Research Centre International Institute of Information Technology Hyderabad, India.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher Manning, Andrew Ng, and Christopher Potts. 2013. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Conference on Empirical Methods in Natural Language Processing*, Stanford, CA 94305, USA. Stanford University.

Swapna Somasundaran and Janyce Wiebe. 2009. Recognizing Stances in Online Debates. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - Volume 1*, ACL '09, pages 226–234, Stroudsburg, PA, USA. Association for Computational Linguistics.

Swapna Somasundaran and Janyce Wiebe. 2010. Recognizing Stances in Ideological On-line Debates. In *Proceedings of the NAACL HLT 2010 Workshop on Computational Approaches to Analysis and Generation of Emotion in Text*, CAAGET '10, pages 116–124, Stroudsburg, PA, USA. Association for Computational Linguistics.

Dhanya Sridhar, Lise Getoor, and Marilyn Walker. 2014. Collective Stance Classification of Posts in Online Debate Forums. In *Proceedings of the Joint Workshop on Social Dynamics and Personal Attributes in Social Media*. Association for Computational Linguistics.

Manfred Stede and Jodi Schneider. 2018. *Argumentation Mining*. Morgan & Claypool Publishers.

C. Thornton, F. Hutter, H. Hoos, and K. Leyton-Brown. 2013. Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. In *Proc. of KDD-2013*, pages 847–855.

Andrew Trask, Phil Michalak, and John Liu. 2015. sense2vec - A Fast and Accurate Method for Word Sense Disambiguation In Neural Word Embeddings. *Computation and Language*, 1.

David Vilares and Yulan He. 2017. Detecting Perspectives in Political Debates. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Henning Wachsmuth. 2015. *Pipelines for Ad-hoc Large-scale Text Mining*. phdthesis, University of Paderborn.

Henning Wachsmuth, Martin Potthast, Khalid Al Khatib, Yamen Ajjour, Jana Puschmann, Jiani Qu, Jonas Dorsch, Viorel Morari, Janek Bevendorff, and Benno Stein. 2017. Building an Argument Search Engine for the Web. In *Proceedings of the 4th Workshop on Argument Mining*. Association for Computational Linguistics.

Hongning Wang, Yue Lu, and Chengxiang Zhai. 2010. Latent aspect rating analysis on review text data. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 10*, pages 783–792. ACM Press.

Ian Witten, Eibe Frank, Mark A. Hall, and Christopher J. Pal. 2017. *Data Mining: Practical Machine Learning Tools and Techniques*, volume 4. Elsevier LTD, Oxford.

Michael Wojatzki and Torsten Zesch. 2016. Stance-based Argument Mining – Modeling Implicit Argumentation Using Stance. In *KONVENS 2016*.

A Appendix

A.1 The feature of paths in WordNet

Bar-Haim et al. (2017a) suggested “path in WordNet” as a feature type for the target identification without further comment. A feature type for machine learning must be one or more real-valued numbers. Hence, we implemented an algorithm to achieve that. The algorithm works as follows:

1. Previous analysis engines segmented the claim and the topic already into tokens. Tokens which are in the candidate phrase and the topic target are extracted in the first step. In the case of argument search the algorithm does not know the target of the topic in the target identification step. Therefore all tokens must be considered. However, this is in most cases no disadvantage because of the query behaviour of the user. The majority of queries contain only one or two tokens and already represent the target.

2. The algorithm filters the tokens by their POS-tags. Not all tokens are worth considering in this distance feature calculation, for example, determiners. One example is “the”. There should be no difference in the semantic distance if the input for a claim target candidate is “creator’s existence of *the* universe” or “creator’s existence of universe”. Tokens like “’s” or punctuations (on the phrase level) are not helpful either.

3. The algorithm calculates the distance for each pair of a relevant claim token and topic token. The sub-method and works as follows:

   (a) If the two words of the pair are equal, return 0.

   (b) Else: A collector searches for all WordNet-Relations for the word on the left side of the pair – the source. The sink, a token of the topic, is fixed. Relations are another words in the same sense group, called the synset. For example, “universe” and “creation” are synonyms in the sense of everything that exists anywhere. There are other relations, too. Each synset contains pointers to antonyms, hypernyms, words regarding the domain or instances and further pointer types. For example, “God” is an instance of the term “creator”.

   (c) The algorithms calculate the distance of each word, which the collector lists to the sink and returns the minimal distance increased by one.

4. The collector of the main method fetches the distance. For example, the distance of the pair $(existence, existence) = 0$, they are equal. Hence, a target candidate containing “existence” seems to be related to the topic containing “existence”. Also, the algorithm does not ignore the other cues and sorts the pairs. Although, we do not have to take into account all distances. How many distances we should consider depends on the number of relevant tokens. The reason is that every relevant token modifies the semantic meaning of a phrase. If we have two long texts, a token match hardly means anything, but if both texts contain exactly one token then a match is significant. Another example: if the input is “existence” and “existence” then the distance should be zero. If the input is “existence of Trump” and “God’s existence” then the algorithm has still $(existence, existence) = 0$, but both phrases are not significantly related to each other any more. Therefore, the algorithm uses the following efficient heuristic: it considers the number of pairs which are equal to the maximum number of the claim target’s relevant tokens and the topic target’s relevant tokens. The algorithm weights the distances to mention the following fact: two phrases tend to be more related to each other if a huge number of tokens match but there is only one exception than there are only one match and one exception. We suggest an inverted-modified logistic regression function as a weight function.

5. Distances above 1 are possible. Therefore, we suggest a normalization of the outputted distance value.

An example related to the description figures A.1.

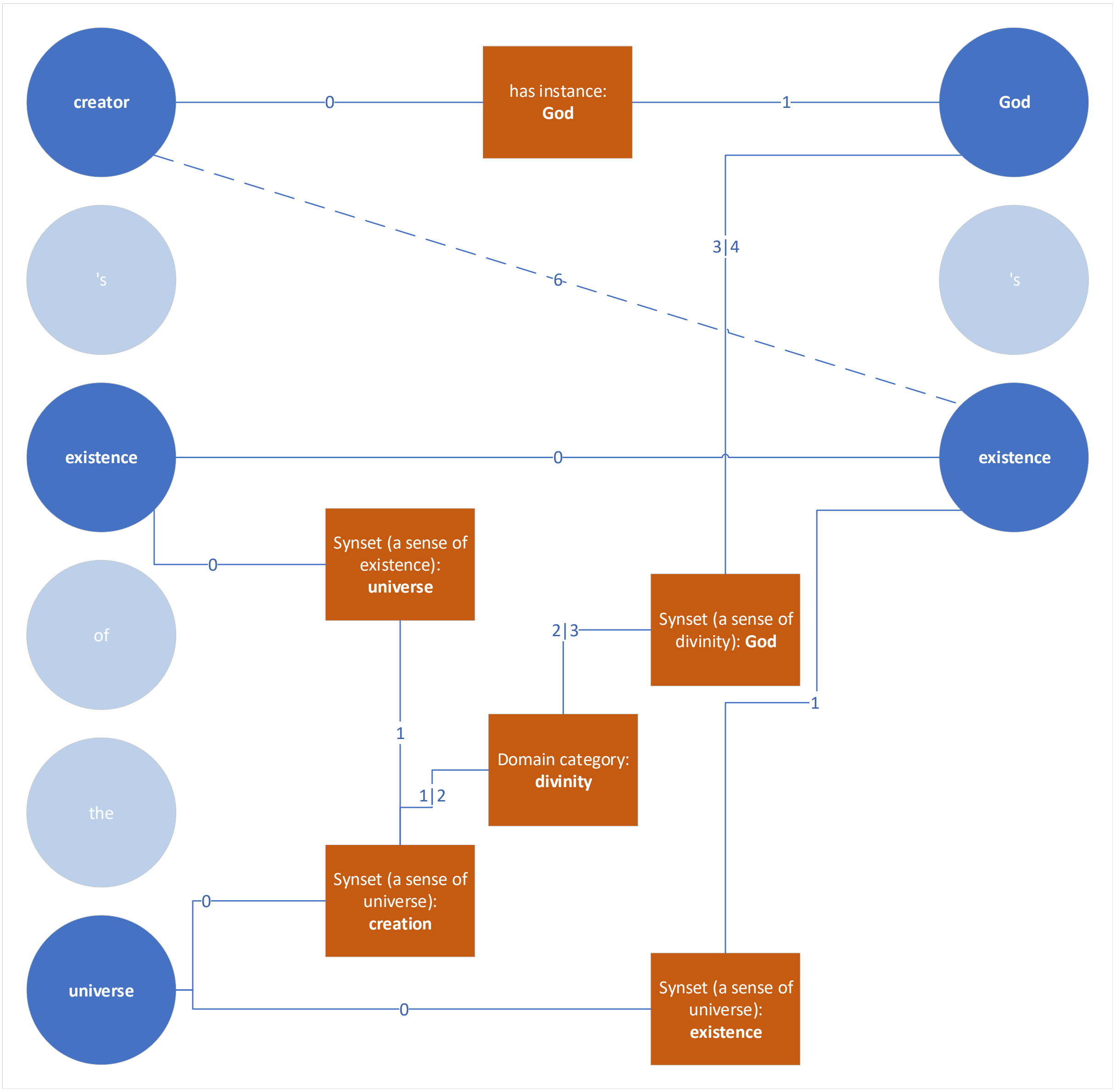

creator
has instance: God
God
's
's
existence
existence
Synset (a sense of existence): universe
Synset (a sense of divinity): God
of
the
Domain category: divinity
Synset (a sense of universe): creation
universe
Synset (a sense of universe): existence
0
1
6
3|4
0
0
2|3
1
1
1|2
0
0

Figure A.1: This figure presents the self-composed algorithm of the WordNet-Path-Feature. Here as an example, the claim target candidate is the noun phrase “creator’s existence of the universe” (left) and the topic target “God’s existence” (right). Each blue node is a token. However, only three tokens on the claim’s side and two tokens on the target’s side are relevant – a filter disables the other tokens, marked in azure. The connections between the left and the right side are the minimal path through the WordNet (brown nodes) to the other word of a word pair. In the example we would have the set $\{(existence, existence) = 0, (creator, God) = 1, (universe, existence) = 1, (universe, God) = 3, (existence, God) = 4, (creator, existence) = 6\}$. Hence, the algorithm would consider the subset $\{(existence, existence) = 0, (creator, God) = 1, (universe, existence) = 1\}$ in the example because of three relevant tokens on the claim’s side. Hence, the final distance for the example is $0 + 1 \cdot \frac{2}{2+e} + 1 \cdot \frac{2}{2+e^2} \approx 0.64$

A.2 All results from the evaluation (effectiveness + efficiency)

This section only lists the results of this thesis. Section 6.1 describes the experimental setup. Section 6.2 visualizes these results graphically and describes them. The analysis is shown in Section 6.3. This section contains Table A.1, A.2 and A.3.

Moreover, this appendix presents all results of the validation of the whole stance classifier. The possibilities of combinations are various. Hence, we give an overview and introduce abbreviations for the later use in the tables:

1. The text analysis preprocessing pipeline (abbreviation: ***Pre***)
   - The standard pipeline which Chapter 5 describes
   - the standard pipeline, but with the TT4J-PhraseChunker instead of the Stanford Parser (abbreviation: ***TT4J***)
   - the standard pipeline with the TT4J-PhraseChunker + tagging verb phrases as relevant phrases for the target identification, too (abbreviation: ***TT4J+VP***)
2. The target identification step (abbreviation: ***Targ***)
   - the approach which Bar-Haim et al. (2017a) suggested – Section 3.1.1 (abbreviation: ***IBM***)
     - ... with all features including the path in WordNetFeature with the maximum search depth of x (abbreviation: ***+W:***x)
     - ... with all features excluding the Wikipedia header feature type (abbreviation: ***-Wiki***)
     - ... with all features excluding the sentiment relation feature type (abbreviation: ***-S***)
   - taking the ground truth (abbreviation: ***ground***)
3. The sentiment classification step – once for the claim (abbreviation: ***Sent-C***) and a second time for the topic (abbreviation: ***Sent-T***)
   - the approach which Bar-Haim et al. (2017a) suggested with a default sentiment of 0.5 – Section 3.1.1 (abbreviation: ***IBM***)
   - the approach of Stanford (Socher et al., 2013) (abbreviation: ***Stan***)
   - taking the ground truth (abbreviation: ***ground***)
   - taking a default value. The majority have a positive sentiment, especially the topic. The search queries often consist of only one to two tokens. Hence, we can assume a positive sentiment. Hence, the default value is 1 (abbreviation: ***1***).
4. The contrast classification step (abbreviation: ***Cont***)
   - the approach which Bar-Haim et al. (2017a) suggested – Section 3.1.1 (abbreviation: ***IBM***)
     - ... with all features excluding the feature type of reachability in WordNet via synonym-antonym chains (abbreviation: ***-W***)
     - ... with all features excluding the relation function which Bar-Haim et al. (2017a) suggested (abbreviation: ***-IBM***)
   - taking the ground truth (abbreviation: ***ground***)

- taking a default value. Especially in the argument search case only a negligibly small part of claim-topic-combinations has a contrastive relation. The reason is in the retrieval step in the argument search engine which is based on a full-text search (Wachsmuth et al., 2017): contrastive targets are hard to recognize with that method. Therefore, the default value is 1 (abbreviation: ***1***).

All abbreviations in combination result in a specific configuration. For example, a setup with the abbreviation "Pre|Targ:ground|Sent-C:IBM|Sent-T:1|Cont:IBM-W-IBM" is an algorithm with the standard text preprocessing pipeline. Furthermore, it uses the ground truth target and a default value of 1 for the sentiment of the topic. However, it uses the approach which Bar-Haim et al. (2017a) suggested for the sentiment prediction of the claim. Also, the algorithm uses the suggested approach for the contrast classification. However, it does not use the feature type of reachability in WordNet via synonym-antonym chains and does not use the relation function which Bar-Haim et al. (2017a) suggested.
Table A.4 contains the results from experiments without a threshold. The stance classifier is forced to predict a stance for every claim-topic-pair.
Table A.5 then presents a test with an activated threshold.

Enabled features	**chosen learning model**	**⊘ squared error**	**f_1-score**	**precision**	**⊘ time**
None (Baseline)	Random	-	0.201 (0.2)	0.201 (0.2)	< 1ms
Only syntactic / positional features	SVM (c = 0.1)	0.293	0.729 (0.739)	0.729 (0.757)	2ms
Only Wikipedia feature	SVM (c = 0.1)	0.460	0.296 (0.335)	0.296 (0.538)	3ms
Only topic relatedness features, no WordNet	SVM (c = 0.1)	0.261	0.531 (0.608)	0.531 (0.675)	30ms
syntactic / positional features + topic relatedness features	SVM (c = 0.1)	0.264	0.745 (0.777)	0.745 (0.806)	38ms
All except Wikipedia feature, no WordNet	SVM (c = 0.01)	0.227	0.751 (0.771)	0.751 (0.793)	36ms
All except sentiment features, no WordNet	SVM (c = 0.01)	0.240	0.752 (0.77)	0.752 (0.802)	34ms
All except WordNet	SVM (c = 0.01)	0.297	0.705 (0.714)	0.705 (0.767)	42ms
All (WordNet: max-depth: 2) except Wikipedia feature + sentiment features	SVM (c = 0.01)	0.31	0.743 (0.771)	0.743 (0.791)	56ms
All (WordNet: max-depth: 2) except Wikipedia feature	SVM (c = 0.75)	0.271	0.726 (0.757)	0.726 (0.762)	53ms
All (WordNet: max-depth: 2) except sentiment feature	SVM (c = 0.01)	0.313	0.737 (0.762)	0.737 (0.779)	59ms
All (WordNet: max-depth: 1)	Decision tree	0.233	0.645 (0.7)	0.645 (0.753)	45ms
All (WordNet: max-depth: 2)	SVM (c = 0.5)	0.27	0.73 (0.759)	0.73 (0.762)	60ms
All (WordNet: max-depth: 3)	SVM (c = 0.1)	0.26	0.738 (0.765)	0.738 (0.764)	76ms
All (WordNet: max-depth: 4)	SVM (c = 0.5)	0.271	0.7 (0.726)	0.7 (0.735)	87ms
All (WordNet: max-depth: 5)	SVM (c = 0.75)	0.329	0.726 (0.749)	0.726 (0.756)	276ms

Table A.1: This table lists the results of the isolated **claim target identification** step on the test set of the dataset of Bar-Haim et al. (2017a). The descriptions for the single feature types are shown in Section 3.1.1. We trained several learning models for each feature configuration. Hence, some learning models outperform other learning models which we prefer in the case of activation of all features in different feature configurations. The second column shows the best learning model out of twelve for every configuration. As can be seen all learning models are regressors. Therefore, the third column shows the mean squared error of the real-valued predictions. All remaining values assume conversion of the target confidence values to a fix prediction, which marks precisely one target candidate as the target. A target is a phrase – the first number represents the score on the phrase level, the number in brackets on the token level. The last column represents efficiency. The listed times are the average processing time for each claim to calculate the feature representation for all containing claim candidates, given all text analysis annotations. A claim contains five claim candidates in average. The maximum depth of five of the WordNet-feature-type is caused by an overload of the computer for higher values.

chosen method	⊘ **squared error**	**accuracy**	⊘ **time**
Random guess (baseline)	1	0.5	< 1ms
Approach of Bar-Haim et al. (2017a) without default sentiment	0.815	0.505	9ms
Approach of Bar-Haim et al. (2017a) with default sentiment -1	1.172	0.666	9ms
Approach of Bar-Haim et al. (2017a) with default sentiment 0.5	0.888	0.677	9ms
Approach of Bar-Haim et al. (2017a) with default sentiment 1	1.128	0.677	9ms
Stanford	0.931	0.446	53ms

Table A.2: This table lists the results of the isolated **sentiment classification** step on the test set of the dataset of Bar-Haim et al. (2017a). The descriptions for the approach of Bar-Haim et al. (2017a) is in Section 3.1.1. "Stanford" means the sentiment approach of Socher et al. (2013). Stanford does not calculate the sentiment towards the target but it calculates the overall sentiment. The time column represents the whole averaged processing time for determining the sentiment of a claim. All listed methods output real values in the range of -1 to 1. Hence, the averaged squared error based an the real values. The accuracy relates to a discrete sentiment prediction, that is a mapping to -1 or 1

Enabled features	**chosen learning model**	⊘ **squared error**	f_1**-score**	**precision**	⊘ **time**
None	Random	-	0.5	0.5	<1ms
None (Baseline)	Majority voting	-	0.886	0.796	<1ms
Only similarity features	Decision Tree	0.941	0.667	0.929	8ms
Only WordNetFeature	K-Star	0.865	0.626	0.939	5ms
Only IBM feature	SVM (c = 10)	1.056	0.074	0.16	177ms
All except WordNetFeature	SVM (c = 0.5)	1.737	0.596	0.957	245ms
All except IBM feature	K-Star	0.942	0.697	0.914	11ms
All	K-Star	0.93	0.689	0.92	229ms

Table A.3: This table lists the results of the isolated **contrast classification** step on the test set of the dataset of Bar-Haim et al. (2017a). The descriptions for the single feature types are shown in Section 3.1.1. We trained several learning models for each feature configuration. Hence, some learning models outperform other learning models which we prefer in the case of activation of all features in different feature configurations. The second column shows the best learning model out of twelve for every configuration. As can be seen all learning models are regressors. Therefore, the third column shows the mean squared error of the real-valued predictions. All remaining values assume conversion of the prediction to a fix label: *contrastive* or *consistent*. The last column represents efficiency. The listed times are the average processing time for each claim-target-pair to determine the contrast, given all text analysis annotations.

Configuration	f_1**-score**	**accuracy**	**PRO-CON-ratio**	⊘ **time**
-|-|Sent-C:1|Sent-T:1|Cont:1 (heuristic)	**0.684**; 0.627	0.519; 0.625	1; 0	<1ms
-|-|Sent-C:ground|Sent-T:ground|Cont:ground	0.998; 0.88	0.998; 0.88	0.04; -0.01	<1ms
Pre|Targ:IBM+W:3|Sent-C:IBM|Sent-T:IBM| Cont:IBM	0.492; 0.67	0.503; 0.66	-0.083; 0.05	0.64s; 0.52s
Pre|Targ:IBM+W:3|Sent-C:IBM|Sent-T:IBM| Cont:IBM-W-IBM	0.502; 0.65	0.503; 0.645	-0.044; 0.02	421ms; 206ms
Pre|Targ:IBM+W:3|Sent-C:IBM|Sent-T:IBM| Cont:1	0.57; 0.712	0.519; 0.7	0.199; 0.07	419ms; 212ms
Pre:TT4J|Targ:IBM+W:2|Sent-C:IBM|Sent-T:IBM| Cont:IBM-IBM	0.446; 0.562	0.489; 0.555	-0.177; 0.02	159ms; 101ms
Pre:TT4J+VP|Targ:IBM+W:2|Sent-C:IBM|Sent-T:IBM| Cont:IBM-IBM	0.456; 0.567	0.491; 0.555	-0.171; 0.04	145ms; 124ms
Pre|Targ:ground|Sent-C:IBM|Sent-T:IBM| Cont:1	0.562; 0.683	0.52; 0.675	0.152; 0.04	209ms; 101ms
Pre|Targ:IBM-S|Sent-C:IBM|Sent-T:IBM| Cont:1	0.562; 0.712	**0.519**; 0.7	0.157; 0.07	355ms; 235ms
Pre|Targ:IBM-S-Wiki|Sent-C:IBM|Sent-T:IBM| Cont:1	0.561; **0.712**	0.516; **0.7**	0.163; 0.07	256ms; 203ms
Pre|Targ:ground|Sent-C:Stan|Sent-T:IBM| Cont:1	0.559; 0.593	0.543; 0.575	0.034; 0.08	291ms; 147ms
Pre|Targ:ground|Sent-C:IBM|Sent-T:Stan| Cont:1	0.579; 0.693	0.540; 0.683	0.2; 0.04	220ms; 104ms
Pre|Targ:ground|Sent-C:Stan|Sent-T:Stan| Cont:1	0.534; 0.615	0.516; 0.606	0.115; 0.04	287ms; 134ms
Pre|Targ:ground|Sent-C:1|Sent-T:IBM| Cont:1	0.661; 0.61	0.533; 0.61	0.7; 0	212ms; 105ms
Pre|Targ:ground|Sent-C:IBM|Sent-T:1| Cont:1	0.555; 0.638	0.504; 0.623	0.266; 0.06	212ms; 105ms
Pre|Targ:IBM-S+W:2|Sent-C:1|Sent-T:1| Cont:IBM	0.454; 0.589	0.447; 0.575	**-0.037**; 0.06	478ms; 284ms
Pre|Targ:IBM-S+W:2|Sent-C:1|Sent-T:1| Cont:IBM-IBM	0.454; 0.592	0.453; 0.578	-0.044; 0.05	380ms; 278ms
Pre|Targ:IBM-S+W:2|Sent-C:1|Sent-T:1| Cont:IBM-IBM-W	0.442; 0.556	0.46; 0.545	-0.096; 0.04	348ms; 232ms

Table A.4: This table lists the results of the full approach. This chapter contains The descriptions for the configuration encodings. The remaining columns always have two values. The first value results from a run on the test set of the dataset of Bar-Haim et al. (2017a). The second value results from a run on our sample of `args.me`. We do not allow the stance classifier to deny a prediction. Hence, the coverage is 1 (Section 6.1.2). The last column represents efficiency. The listed times are the average processing time for each claim-target-pair to determine the stance. We ignore the first 10% of the claim-topic-pairs in the time average because the first claim-topic-pairs sometimes have an extraordinary time consumption because of the amortization process of the initialization.
The table formats the best values in each category bold whose configuration does not contain any use of the ground truth. The ideal PRO-CON-ratio is 0.039 in the case of the dataset of Bar-Haim et al. (2017a) and is 0.01 regarding the case of our sample of `args.me`.

Threshold	f_1-score	accuracy	PRO-CON-ratio	coverage
0.0	0.561; 0.712	0.516; 0.7	0.163; 0.07	100%; 100%
0.05	0.548; 0.712	0.519; 0.704	0.231; 0.07	94%; 100%
0.1	0.543; 0.712	0.52; 0.704	0.184; 0.08	92%; 100%
0.15	0.543; 0.712	0.525; 0.704	0.182; 0.08	90%; 100%
0.2	0.531; 0.712	0.521; 0.704	0.18; 0.08	90%; 100%
0.25	0.532; 0.713	0.522; 0.711	0.174; 0.1	88%; 94%
0.3	0.519; 0.687	0.536; 0.718	0.228; 0.08	75%; 87%
0.35	0.492; 0.631	0.525; 0.698	0.525; 0.082	68%; 80%
0.4	0.264; 0.306	0.505; 0.848	0.268; 0	40%; 23%
0.6	0.006; 0.02	0.625; 1.0	0.333; 0	1%; 1%
1.0	0.0; 0.0	–; –	0; 0	0%; 0%

Threshold	f_1-score	accuracy	PRO-CON-ratio	coverage
0.0	0.492; 0.67	0.503; 0.66	-0.072; 0.05	100%; 100%
0.05	0.46; 0.667	0.512; 0.674	-0.072; 0.058	84%; 95%
0.1	0.422; 0.653	0.518; 0.672	-0.086; 0.08	73%; 89%
0.15	0.337; 0.586	0.517; 0.701	-0.01; 0.1	50%; 67%
0.2	0.265; 0.528	0.523; 0.748	-0.062; 0.103	32%; 54%
0.25	0.192; 0.435	0.505; 0.768	0.058; 0.073	23%; 41%
0.3	0.149; 0.411	0.531; 0.76	0.019; 0.08	16%; 38%
0.35	0.112; 0.368	0.544; 0.742	0.046; 0.075	12%; 33%
0.4	0.046; 0.127	0.544; 0.818	-0.244; -0.13	6%; 11%
0.6	0.0; 0.02	–; 1.0	0; 0	0%; 1%
1.0	0.0; 0.0	–; –	0; 0	0%; 0%

Table A.5: These both tables show the results of a threshold test: we allow the stance classifier to deny predictions. For example, if the threshold is 1.0, then the stance classifier predicts only the stance if the algorithm is 100% sure.
We selected the two representative configurations for the threshold test, which did not need the ground truth (see A.4):

1. Pre|Targ:IBM-S-Wiki|Sent-C:IBM|Sent-T:IBM| Cont:1
2. Pre|Targ:IBM+W:3|Sent-C:IBM|Sent-T:IBM| Const:IBM

Each configuration has a separate table. Therefore, the first table is related to the first configuration, the second table is related to the second configuration.
Each cell except the cells of the first column contains two values. The first value belongs to the experiment based on the test set of the dataset of Bar-Haim et al. (2017a). The second value results from a run on our sample of `args.me`. The f_1-score takes claim-value-pairs with an unpredicted stance into account, too.