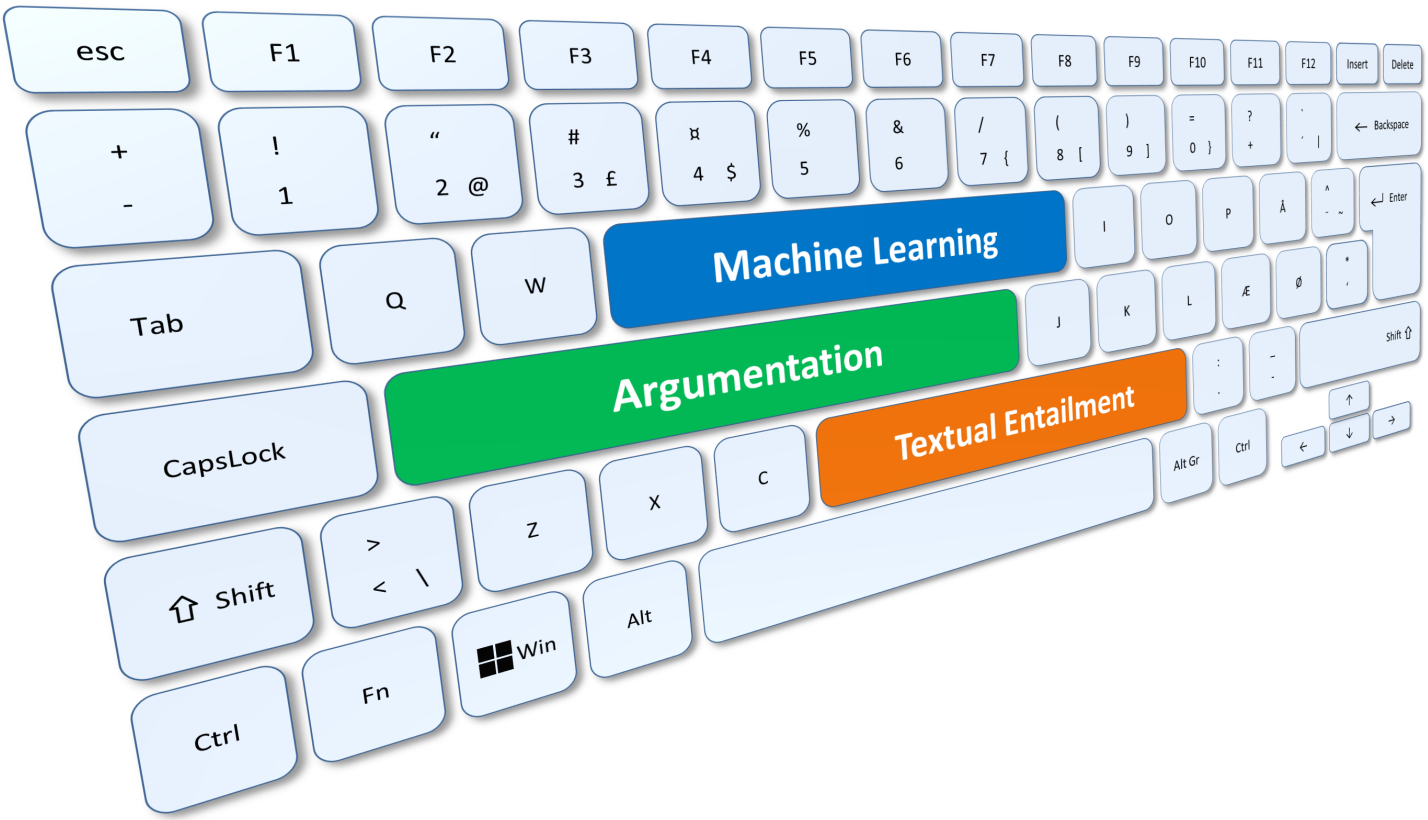
Technical University
of Denmark

Master Thesis

Machine Learning for Argumentation Mining

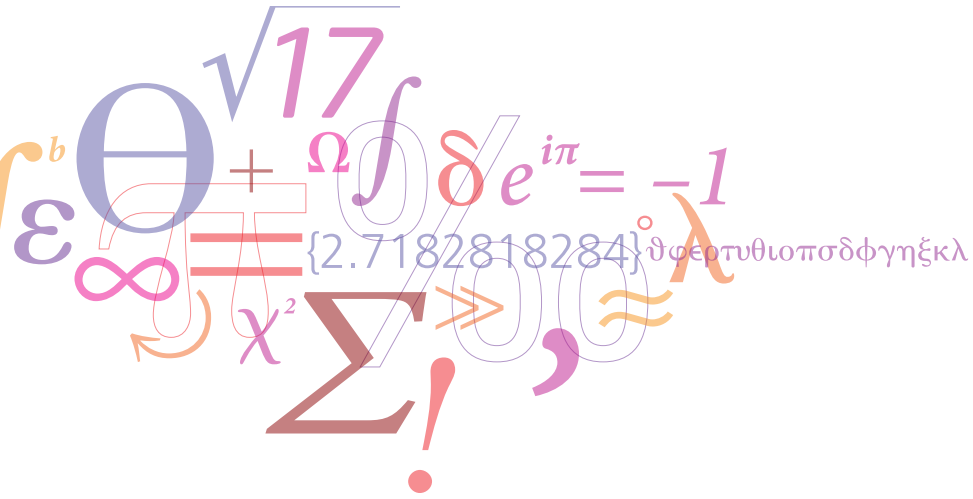
Jeppe Nørregaard, s103020

July 10, 2016



Supervisor: Lars Kai Hansen

DTU Compute
Department of Applied Mathematics and Computer Science

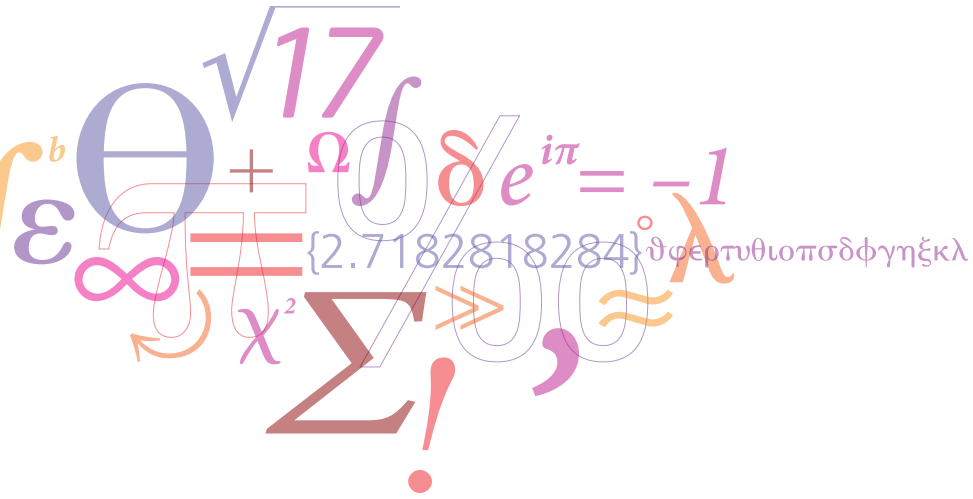


Abstract

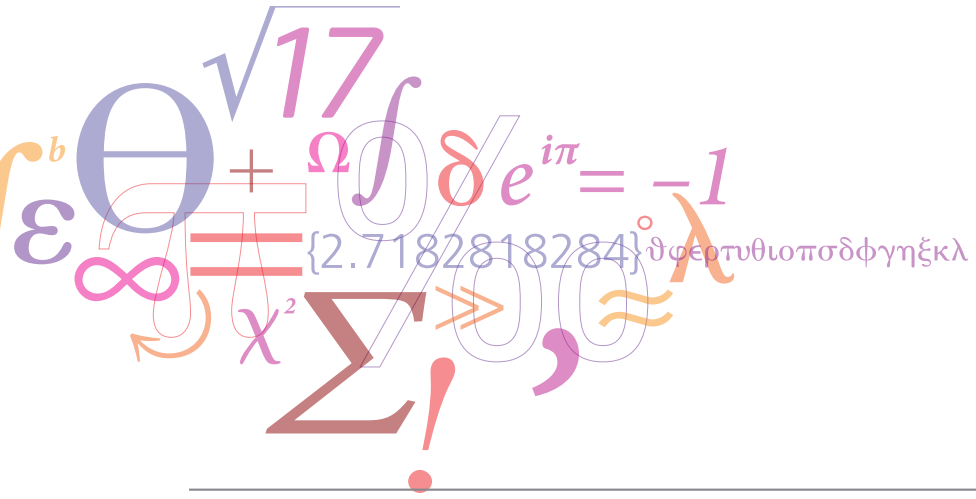
This project surveys the current field of argumentation mining and investigates the methods proposed by Raquel Mochales Palau and Marie-Francine Moens in [53]. The survey describes many of the modern methods used, together with models and representations of arguments in computers and some philosophical aspects of argumentation theory.

In [53] they use a machine learning system to detect argumentative propositions in texts. The motivation for this method is questioned, as most propositions seems to be argumentative in some context. The method is implemented and tested, although the method of testing is different from that of [53] as the original data is unavailable, making the exact experiments of [53] impossible to reproduce. The method was capable of selecting similar samples from unlabelled text-sources, and could potentially help increase the size of the used database.

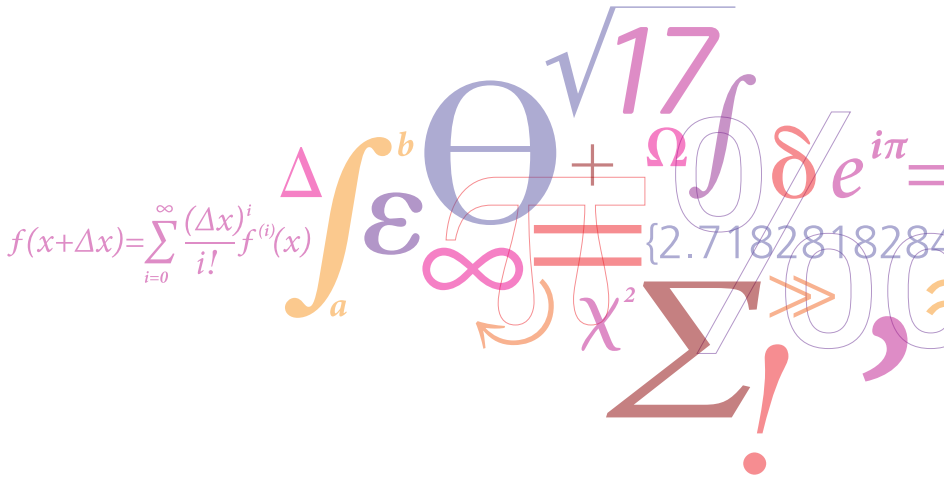
A different system created in [53] detects argumentative structures by using a context-free grammar. An alternative to this system is created here, where a textual entailment approach is used to detect entailment between claims and premises. It uses machine learning methods together with the same features as the argumentative proposition detection system mentioned above. The learning schemes tested were logistic regression, linear SVM and radial basis function SVM. For improving performance, sample-normalization, feature-scaling, outlier detection and removal, and feature reduction were tested as preprocessing. Furthermore the well-known Word2Vec and Doc2Vec schemes were used to create an alternative feature-space. The best-performing system found was a stacking ensemble, which used a logistic regression classifier to post-process the outputs of 11 other classifiers. This ensemble was capable of detecting premise-claim links with an F1-score of 68% and accuracy of 73%. This performance is comparable to the one from [53].



Contents



Index



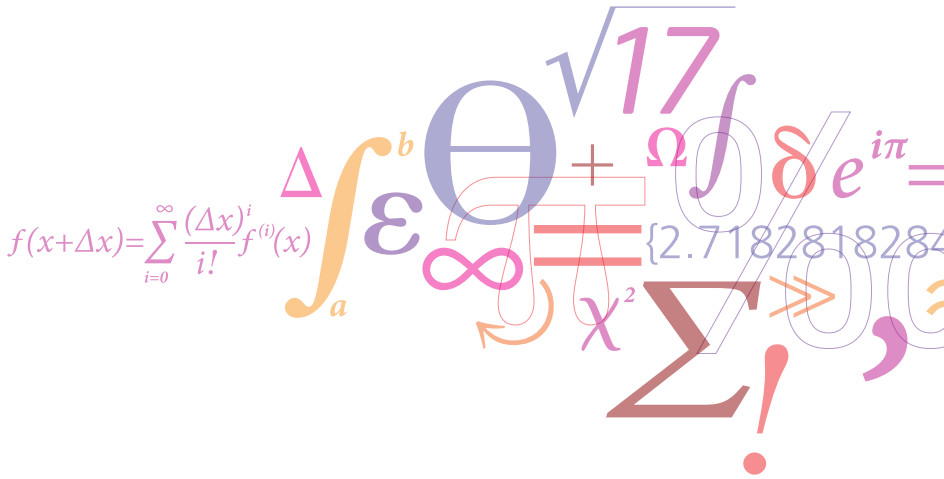
Chapter 1

Introduction

The intentions with this project was to survey the current field of argument mining, and investigate the use and possible improvements of one of the systems used today. Argument mining is the use of computers for automatically detecting arguments in text. The field encompass elements from natural language processing and machine learning, together with some more philosophical aspects in argument theory, which analyses arguments and their use. Chapter 2 describes the related work done within argumentation mining as well as describing some theories on argumentation.

The investigative part of the project focus of the work of [53]. This article takes on several tasks, two of which this report will also analyse. The first task is to detect argumentative elements in pre-segmented text. For this system, [53] suggests some features derived from paragraphs, which machine learning algorithms use to detect argumentative elements. This method is discussed and tested here, although, due to changes made in the related database, the method is investigated slightly differently. The other task is the detection of argumentative structures, which is handled by detecting links between argumentative premises and conclusions in a bipartite graph. For this task [53] uses a grammar. Instead of the grammar, this project investigates the use of the same features as used for the previous task to detect the links with machine learning techniques. Various methods will be used to improve the classification task, including normalizations of the data, feature reduction, outlier removal and a simple ensemble method. Furthermore the Word2Vec (vector representations of words[42]) and Doc2Vec (vector representations of texts [37]) methodologies are tested for making alternative features for detecting arguments.

The data used by [53] is the Araucaria Database. This database does not exist in the same state as used previously. The database was found with only the arguments (not their origins). This limits the possibility of reproducing the results of [53], as the origins of the arguments are needed to test detection of arguments. In order to handle this the problems are defined differently. Also some additional data was gathered from online sites, where some of the sites are believed to be the origins of some the the Araucaria arguments.



Chapter 2

Argumentation Mining Survey

2.1 Related Work

Below is a summary of the many approaches and efforts related to the field of automatic argumentation analysis or computational argumentation, which argumentation mining (AM) is part of. The first part is a description of the different fields that argumentation mining is currently being applied to (or attempted to be applied to), followed by a section on the progress made within argumentation mining.

2.1.1 Domains

Legal Domain

Argumentation mining has had a lot of scientific attention in the last two decades. Much of the attention have focussed on analysing arguments in legal texts. There are several reasons for using legal texts: they are typically publicly accessible, they are well structured and consistent in language and phrasing, they are sorted within topics and cases, and there is a large commercial interest. The usages for argumentation mining in the legal sector includes filtering and searching through legal documents. As legal reasoning is complicated and often requires digging into similar cases of the past, an information retrieval system that can detect specific arguments could be very valuable. [46] made some important progress in detecting argumentative sentences and classifying these sentences into being premises or claims. They use the Araucaria Database [65], which was intended for the development of automatic argument analysis, and later ([53]) use documents from the European Court of Human Rights [25]. [4] further motivates the use of argumentation mining in legal texts to improve information retrieval. [3] wants to use argumentation analysis to detect Almost Identical Expressions (AIE) in legal texts, which is important for the understanding and correct interpretation of laws.

Online Community

Another domain for argumentation mining is the vast amount of online data, such as social networks, debate forums, new sites etc. The amount of textual data online is increasing and can

at times be overwhelming. A very useful purpose for AM would be to summarize debates with all arguments for and against some claim, filter out irrelevant comments and search through sites to find specific arguments one is interested in. Such systems could heavily improve the online debating environment and the data is readily available, although a big obstacle with such data is the inconsistency in language and grammar, which is a big challenge for the field. [16] has worked with online data from *Procon.org*[54] and *Idebate.org*[6]. From these two sites they gathered arguments and comments for and against two topics: the legalization of gay marriage and the inclusion of God in the American Pledge of Allegiance. They stored these in the ComArg database, which is available online [15]. The choice of debates was motivated by *"...[they have a] larger-than-average number of comments (above 300) and are well-balanced between pro and con stances"*. [19] used *Debatepedia*[5] to collect a database which they also made available online through [20]. Debatepedia lets users place pro- and con-comments, and thus creates an user-tagged dataset. [22] motivates the use of AM on online content, by addressing its usefulness with *Quaestio-it*[41], which is an online platform for visualizing argumentation graphs. Automatic extraction of arguments could convert plain text into such graphs, using systems such as Quaestio-it, to give users an overview of the subject. [29] and [30] have made a lot of work on the argumentation-theoretical perspective. They determine that different argumentation theories and representations should be applied to different situations and domains on the web, as no theory or representation is sufficiently efficient and broad to cover them all. Furthermore they create a dataset from various sources within the educational domain, which is available online [31].

Other Domains

Others have worked on argumentation mining within the scientific community, with one goal being the automatic extraction of arguments in scientific articles. [71] has made some theoretical progress in modelling scientific argumentation, and using these to determine the implicit intentions of scientific authors. [27] has worked on a corpus of argumentation in biomedical genetics research articles. [12] works on using arguments to enable computers to understand motives, intentions and reasoning of characters in stories.

Finally the field of artificial intelligence has shown an interest in argument modelling, as a communications tool for multi-agent systems. In these systems the agents have different and limited knowledge about the world, and communicate to make collective decisions. The artificial intelligence perspective does not handle human arguments, as in argumentation mining, but has made a lot of progress in the modelling of arguments and in the combination of different agents' knowledge. [7] works with collective rationality, in which a group has to make the best possible collective decision. As they each come with different knowledge they can use argumentation-schemes to combine knowledge and make a rational decision. [73] works with strategies for the individual agent, who can persuade other agents by tactically weakening the others arguments for decisions. [24] uses argumentation for multi-agent negotiation, in which the individuals have their own utilities that can be collectively maximized.

Databases

In the survey of this project, some databases were found available online. An overview of these databases are found in appendix A.5 and include:

1. Araucaria
2. ComArg
3. NoDE
4. Habernal 2015
5. RTE-7

Representation and Usage

One notable effort made towards argumentation mining is the development of the Argument Interchange Format by [23]. This format has set a standard for how to represent arguments in computers, and is important for the distribution of relevant datasets. This format was intended to be flexible enough for all situations, while maintaining some common definitions. [29] discuss the various abstract models and annotation schemes for arguments and concludes that no single representation will suffice for all online content. They do although recommend two annotation models; Toulmin's model and a Claim-Premises scheme, and argue the benefits of the two. [1] takes the abstract argumentation models and AIF format into a more practical setting and designs a Relational Argument DataBase (RADB).

Others have worked with specific situations in argumentation. [2] investigates modelling and detection of counter arguments through a case study. [17] takes argumentation mining into a dialogue domain, instead of written documents which is the norm of the field. This creates additional challenges due to the amount of implicit information in dialogue. [9] works on including the uncertainty specifications in natural language into argumentation representations in computers (for example the words "definitely" and "possible" indicates different levels of certainty in an argument). [33] analyses algorithms for analysing argumentation graphs, and so prepares the "goal" of argumentation mining while motivating the field by identifying possibilities.

2.1.2 Argumentation Mining

Argumentation Mining

Argumentation mining largely consists of two tasks: extracting the atomic elements of arguments (claims and premises) and connecting the components to create argumentation structures. The first part is in some cases divided further into two parts: ① delimiting sentences/propositions and ② classifying the resulting elements into being argumentative or not and possibly into premises/claims. Likewise the structure-creation is sometimes done by first ③ linking the atomic elements into individual arguments and then ④ linking these arguments depending on whether they support or attack each other. Figure 2.1 illustrates this process.

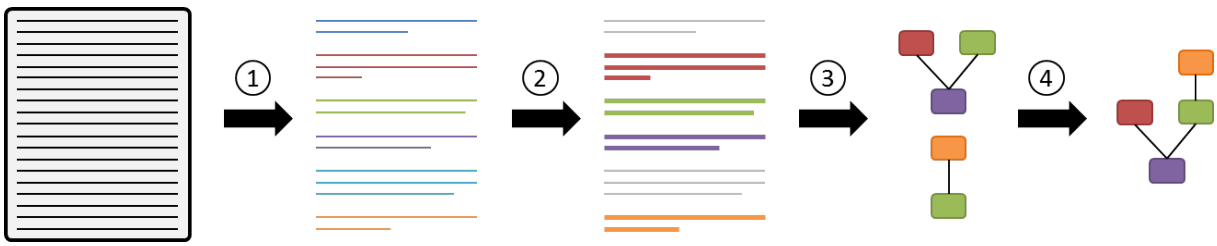


Figure 2.1: Main tasks/process of argumentation mining. Some steps are merged in some systems.

In [40] various machine-learning methods and features used for argumentation mining are investigated. One important conclusion drawn here, is that most work focus on selecting clever features rather than customizing machine-learning methods (off-the-shelf implementation are usually used). They thus promote a larger focus on implementing specific machine-learning methods for the purpose. Some of the methods that have been tried are Naive Bayes, Support

Vector Machine, Maximum Entropy, Logistic Regression, Decision Trees and Random Forests.

A system started by [46] and developed since through [53], [44] and [45], made large progress in argumentation mining. They first detected argumentation elements using an entropy maximization classifier, with accuracy of about 73%. The features they chose were:

- Unigrams, bigrams and trigrams
- Part-Of-Speech tagged: adverbs, verbs and modal auxiliaries
- Word couples (any pair of two words in same segment)
- Text statistics: sentence length, average word length and number of punctuation marks
- Sequence of punctuation marks
- Key words and phrases characteristic to argumentation developed by [35]
- Parse features: depth of parse-tree and number of subclauses in tree

A second system detects the argumentative role (premise or claim) of each proposition using an entropy-maximisation and SVM combination. They maintain an F1-score of 74%. They also compare the last method with a context-free grammar, which produces an F1-score of 67%. Finally they create a system to detect argumentative structures, which performs with an F1-score of 60%.

This system is modified in [48] by creating a topic model and removing topic-related words. This leaves argumentation-related words back, which reduces the dimensionality of the problem. On the structural aspect they use a parser to detect the main subject and main verb in a sentence and use these pairs as features (ex. "I.think" and "view.be" from sentences "I think that ..." and "My view is that ...").

The argumentation mining problem is sometimes cast into a Textual Entailment problem, as done in [19]. Textual entailment is defined as a relation between a hypothesis and a text. Entailment holds if the hypothesis can be inferred from the text. A textual entailment system detects such a relation between the text and the hypothesis or a contradiction between the text and the hypothesis. Given a claim, textual entailment therefore represents a "guided" argumentation mining, aimed and proving/disproving a specific claim instead of detecting all arguments. This version of argumentation mining is similar to opinion mining. [16] uses textual entailment, textual text similarity and stance alignment for a five-class classifier. The classes are (1) explicit attacking arguments, (2) implicit attacking arguments, (3) indifferent propositions, (4) implicit supporting arguments and (5) explicit supporting arguments in text. They have an accuracy of 70.5% to 81.8% depending on problem formulation.

Instead of relying on parsers for segmentation, [36] investigates a machine learning approach to argument mining. Their system segments the text, then classify propositions as argumentative or not, after which it connects the propositions into argument structures. They use two Bayes classifiers to detect the starting word of a propositions and the ending word of a proposition, with fairly simple features. They compute a topic model and uses the distance between propositions in the topic space as a basis for detecting argumentative structure. Those propositions that are not connected into any structure are afterwards sorted out and considered non-argumentative.

2.2 Argumentation

Argumentation is an important part of human communication and is constantly used in web debates, social forums, for legal matters, in educational systems, academia, social life, politics, democracy etc. As previously outlined, the automatic analysis of human argumentation is especially being research with respect to legal documents and debates (like [4], [53], [46]), as well as online, user-generated contents (such as [16], [18], [22], [29])

Computational-argumentation has also created interest in the field of artificial intelligence and multi-agent systems ([51]). Here argumentation can be used for agents to agree on solutions and understanding other agents rationale. Using argumentation as a communicative tool in multi-agent systems is a big field of research and has an collaborative, university arranged annual workshop ([62]).

This section describes various approaches to analysing arguments. First a brief description of argumentation theory is made, which is the philosophical approach to analysing arguments. These create the foundation and motivation for analysing arguments with computers. The next section is on argumentation mining, which is the focus of this project.

An brief overview of logic related to argumentation is found in chapter 3.

2.2.1 Argumentation Theory

Argumentation theory is an interdisciplinary field, analysing the way in which conclusions are made from logical reasoning. It is studied within politics, philosophy, science, mathematics, computer science as well as other fields. The philosophical perspective on argumentation is the cornerstone of the modern tools and abstract models used to represent and analyse arguments, whether manually or by computers. These studies typically start by defining argumentation, creating various definitions depending on context and formality. Some definitions are fairly broad in an attempt to contain all possible versions of argumentation, which is useful when building systems aimed at analysing any argument. [76] defines argumentation as follows:

> ***Argumentation*** *is a verbal, social, and rational activity aimed at convincing a reasonable critic of the acceptability of a standpoint by putting forward a constellation of propositions justifying or refuting the proposition expressed in the standpoint.*

This definition above is used here, with the exception that *verbal* is broadened to include any sort of communication: verbal, written, symbolic etc. There are a few important aspects of this definition. First of all argumentation is a way of *communicating*. Also it is a *social* construct requiring multiple agents (due to the usage of argumentation systems in artificially intelligent multi-agent systems, we don't restrict the social interactions to be between people). Argumentation is also a *rational* activity, where the agents use rationale and logic in their communication, which excludes other communicative ways of affecting opinions, such as seduction, various rhetorical methods and emotional persuasion. It also requires the listening agents to use rationale to understand the arguments. Finally it consists of some kind of standpoint, along with various supporting or refuting propositions.

Components of Argumentation

To analyse arguments, we first need to define the components of argumentation. The following definitions are inspired by [11]:

Argument: An argument is composed by a set of premises and a claim, where the claim can be logically drawn through deductive reasoning from the premises.

Deduction: Deductive reasoning is the process where a claim is logically true, given a set of premises (if the premises are true, then the claim must be true).

Claim: The conclusion of an argument. Also called the consequent.

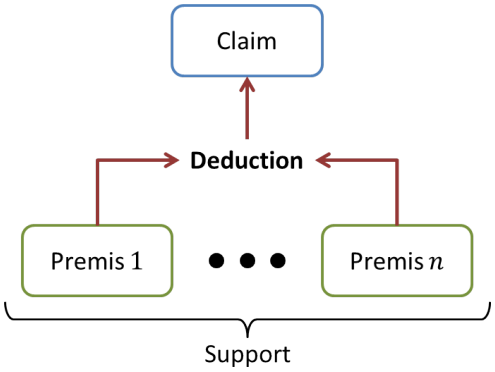


Figure 2.2: Graphical representation of an **argument**.

Premis A proposition used to support a claim.

Support: The support of an argument is the set of premises used to deduce the claim. Also called the justification.

Proponent: The agent making the argument, advocating and defending the claim.

Audience: The recipients of the argument, who will decide whether to believe the argument or not (and possibly argue against the claim).

In human argumentation probability is typically an important aspect. The logical analysis is therefore often augmented by probabilities and uncertainties.

Interaction of Arguments

Argumentation is also defined by [11], but with more focus on argumentation being the process by which arguments and counterarguments are constructed and handled. Here handled means evaluating the arguments and judging their justification and truthfulness. In this context, multiple arguments are used with and against each other. The following constructs describe the relationships between multiple arguments (classical logic used to assist descriptions):

Contradiction: One statement contradicts another statement if and only if, they are mutually inconsistent. This happens if one claims is equal to another claim negated (claims a and $\neg a$ are contradicting), but also if a set of claims can deduce the negation of another claim ($\neg a$ contradicts $b, b \rightarrow a$).

Counterargument: A counterargument appears when the claim of one argument contradicts a proposition (claim or premise) of another argument. There are two types of counterarguments: rebutting arguments and undercutting arguments.

Rebutting Argument: A rebutting argument is a counterargument, where the argument's claim contradicts another arguments claim. Example: $a, a \rightarrow b$ rebuts $c, c \rightarrow \neg b$.

Undercutting Argument: An undercutting argument is a counterargument, where the argument's claim contradicts a premise of another argument. Example: $a, a \rightarrow \neg c$ is an undercutting argument for $c, c \rightarrow b$.

Toulmins Argumentation Layout Model

To represent argumentation within a computer, an abstract model of arguments and their interactions is needed. [75] was early in creating an abstract model of argumentation, and many uses his theories as a foundation for models used in computer science. These abstract models for arguments can be represented as directed graphs. Some of the philosophical models of argumentation is motivated by the manual, practical analysis of argumentation and the models used in computers are commonly more generalized version of the philosophical ones.

Toulmin also based his model on a claim (or qualified claim), which is the conclusion of an argument. The support/premises is split into two:

Support: Pieces of information. Also known as evidence, proof, data, grounds, etc. In logic, the Toulmin support would be of the form $a, \neg b, c$.

Warrant: The part of the argument that links the support to the claim. They can be represented as rules. In logic these look like $a \rightarrow b, \neg c \rightarrow d, e \rightarrow \neg f$. Toulmin emphasises that warrants may be implicit. For example *"He committed the crime, therefore he must be punished."* This argument contains an implicit warrant that people who commit crimes should be punished. The support is that he committed the crime and the claim that he should be punished.

Furthermore Toulmin introduces two other concepts to argumentation-structures:

Backing: Any statement that supports a warrant.

Qualifier: A word that emphasizes the strength of a statement. Argument structures can be represented as logic, but in actual argumentation, arguments are typically not absolute. Rather argumentation includes a lot of probability. Qualifiers are words like "definitely", "always" and "sometimes", which indicates the degree of confidence in a proposition. In a more scientific setting, qualifiers could for example be actual probabilities derived from empirical evidence. This inclusion of probability could be a motivation for using fuzzy logic when representing argumentation.

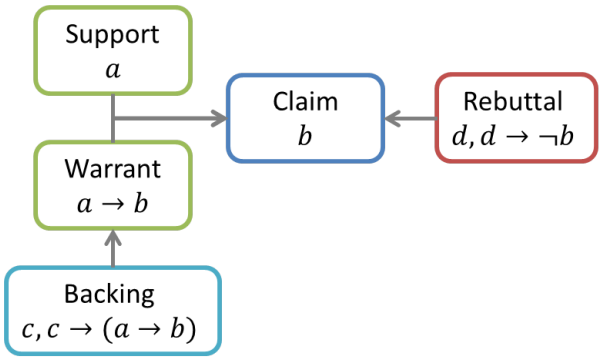


Figure 2.3: Toulmin's argumentation model. The structure is recursive, as the rebuttals and claims are arguments themselves.

In Toulmin's view, there are four different basic argument structures: A simple argument is a standalone argument for a claim (the base case). **Subordinatively compound argumentation** occurs when a chain of arguments supports each other and finally the claim. When several arguments has the same claim, they can either be arranged in **Coordinatively compound argumentation**, which is when each of the arguments depend on each other. In this case the

arguments could be collected to one argument, as none of the arguments can stand alone. The other arrangement is **Multiple argumentation**, where multiple standalone arguments have the same claim. This is a stronger way of arguing, as each of the arguments are valid without the others, and multiple counterarguments will therefore be needed to rebut.

Generic Argumentation Layout Model

The theories of Toulmin were philosophically motivated, as opposed to models used in computers. If claims are broadened, so that claims and premises includes both logical literals (like a and $\neg b$) and logical statements (such as $a \rightarrow b$), then a warrant together with its backing becomes an argument itself. The same goes for rebuttals together with any support they may have. This creates a generic model for argumentation structures, which creates a recursive, directed graph. In this representation, any collection of premises which logically deduces a claim will, together with the claim, compose an argument. If we have two rebutting arguments, these two will have the same claim, but with a negation. This could create two distinct graphs, as there may be no shared propositions in the graphs. In argumentation one would often want to analyse all arguments for and against some claim. Thus a useful representation would be one where every argument contain a set of premises, a claim and an identifier for whether the argument rebuts or supports the claim (two different types of edges). This would connect all related arguments in one directed graph, which can be recursively illustrated as:

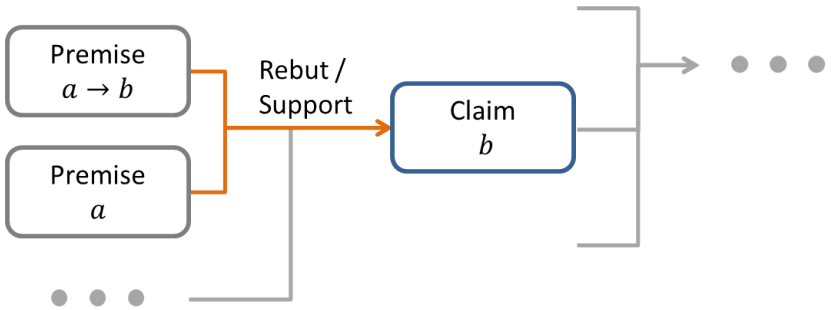
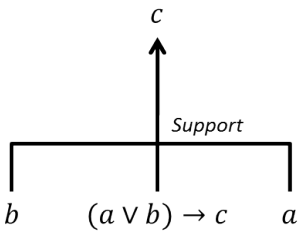


Figure 2.4: Generic argumentation layout model, with examples expressed in logic. Arguments can have more than two premises and any claim can be a premise of another argument.

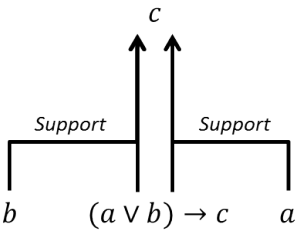
It is natural to represent an argument with a directed graph, as the premises can deduce the claim, while the claim can not deduce the premises. From the above illustration, one might infer that the representation is a tree. An argumentation structure could though be created, where repeated arguments creates a cycle, so that a chain of arguments from a set of premises ends up creating one or more of the same premises. This creates a so-called circular argument, which is often considered a logical fallacy ([10]). Unless some self-reinforcing / positive feedback loop is needed in the argumentative representation, it may therefore be wise to keep the representation as a tree. If special cases of needed circular reasoning arise then one could expand the model at that time.

Another important aspect of the model is what set of premises to allow in an argument. In human argumentation one might find arguments with multiple premises, for which several subsets of the premises can entail the claim. Although this may be the way the arguments are presented in language, it may create a cluttered representation. For example undercutting arguments

may not be sufficient counterarguments, as they may only reject a subset of the premises. An alternative is to restrict arguments to only contain a minimal set of premises, that entails the claim. This is illustrated below:



(a) Example of an argument, which is a conjunction of two minimal arguments.



(b) Example of two minimal arguments, which together composes the same information as the conjunctive version.

Figure 2.5

The disjunct representation does not cause need for storing more premises, as the two arguments can refer to some of the same premises.

A final possible restriction to the modelling of arguments, would be to only allow one instance of each argument: that is, not store two or more arguments using the same premises and claim, even if they are made multiple times in the source of the arguments.

Combining two logical expression produce a conjunction of the two expressions (combining a with b creates $a \wedge b$). Conjunction therefore compose the natural limits of given pieces information in argumentation. For an argument to be useful, it generally requires at least one conjunction (at least two logical expressions separated by $\wedge$) in order to produce new information. For much of the work in argumentation mining this restriction is ignored, as detecting multiple premises for one claim allows the programmer to later combine pieces of information into complete arguments.

2.2.2 Argument Interchange Format (AIF)

The initial specifications of an argument interchange format was proposed in [23]. The format was developed to be a unified abstract model for arguments, which could efficiently be processed by machines. AIF introduces a standard for three main concepts: argumentation representation, communication of arguments and description of argument context. The argumentation representation is the most relevant aspect to this report, and is thus briefly described here.

AIF represents arguments and their interactions with a directed graph, which they name an Argument Network (AN). An AN is not restricted to be a-cyclic or the like, as the framework is meant to be very flexible and restrictions can be introduce for specific situations. An AN contains two types of nodes. An **information node** or **I-node** relates to content and represent claims and premises. A **Scheme node** or **S-node** represents argumentation schemes and reasoning. They connect I-nodes (and other S-nodes) to assert the kind of reasoning used. Scheme nodes are further divided into three kinds:

RA-node RA-nodes represents an inference scheme. Inference is when a conclusion is reached

on the basis of evidence and reasoning.

PA-node PR-nodes represents preference schemes. A preference scheme is when the proponent concludes that one situation is preferable over another. The preference may be implicit.

CA-node CA-nodes are conflict application nodes. They are used to connect arguments whose claims conflict.

AIF also proposes some standard attributes to nodes, such as title, creation date, type etc. Although the specific implementation can use any attributes wanted.

In [13]'s analysis of AIF they define AN mathematically as:

Definition 2.1. An AIF argument graph G is a simple digraph (V, E) where:

- $V = I \cup RA \cup CA \cup PA$ is the set of nodes in G, where I and the I-nodes, RA are the RA-nodes, CA are the CA-nodes and PA are the PA-nodes; and
- $E \subseteq (V \times V) \backslash (I \times I)$ is the set of the edges in G; and
- if $v \in V \backslash I$ then v has at least one direct predecessor and one direct successor.

■

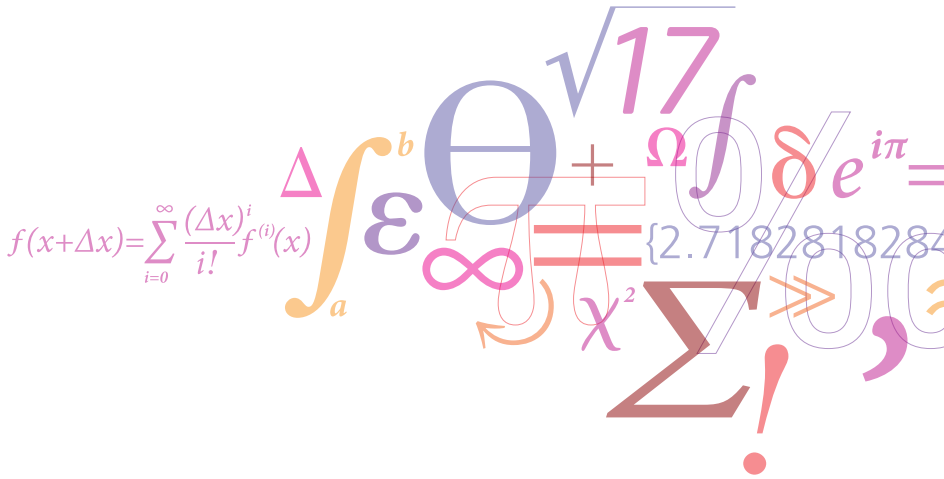
[23] also defines the semantics of edges between any types of nodes. For example RA-node with I-nodes as parents uses premises to apply reason, while a CA-node having RA-nodes as parents notes a conflict of reasoning.

2.2.3 Argument Representation in This Project

This project focus on two tasks:

→	*Detection of Argumentative Elements*	Corresponding to task ② in figure 2.1.
→	*Detection of Links Between Premises and Claims*	Corresponding to task ③ in figure 2.1, with the modification that premises and claims have been classified/separated.

The first task is a classification of text-segments and thus uses no particular representation of arguments. The seconds task creates argumentation graphs represented very simplistically in the project. A bipartite graph is created with a premise/claim bisection (a text segment can appear as two nodes, one in each section). A link between two nodes in the bipartite graph notes a textual entailment relationship between the premise and the claim; the claim can be backed up up by the premise. Complex argument-structures are represented when a text segment appears in two nodes, one in each section, and there are edges adjacent to both nodes.



Chapter 3

Theory

The following describes in detail the different methods, schemes, theories and terminology relevant to the project.

3.1 Logic

When describing arguments it is useful to use logical terminology and notation. The following describes some basic components of propositional logic, which can be further elaborated in [49]. A **proposition** or **logical statement** is a sentence or statement which is either true or false. These exist in three forms: atoms, literals and compound propositions:

Atom An atom is a single piece of information. For example: *"Socrates is a man"* and *"All men are mortal"*. In logical notation these are represented by a single letter or word, for example: a, α and $word$.

Literal A literal is an atom or a negated atom. Notation-wise negation is represented by $\neg$. Thus examples of literals are: a, $\neg\alpha$ and $\neg word$.

Compound Propositions Any collection of literals separated by logical operators are compound propositions. *"Socrates is a man and all men are mortal"* is therefore a compound proposition, which in notation could be represented as $a \wedge b$.

A **logical operator** is a function on logical values, that produces another logical value dependent on the input. Table 3.1 shows a collection of common propositional operators.

Table 3.1: Logical Operators.

Operator	**Short name**	**Symbol**
Conjunction	and	$\wedge$
Disjunction	or	$\vee$
Negation / denial	non / not	$\neg$
Implication	implies / only-if	$\rightarrow$
Reverse implication	if	$\leftarrow$
Double implication	if-and-only-if / iff	$\leftrightarrow$
Exclusive disjunction	exclusive or / xor	$\oplus$

Entailment occurs when one proposition s follows logically from a logical sentence S. s is here called a **logical consequence** of S. This is noted as

$$S \models s, \tag{1}$$

which means s is true whenever S is true.

A logical **clause** is any proposition of the form

$$A_1 \vee A_2 \vee ... \vee A_n \leftarrow B_1 \wedge B_2 \wedge ... \wedge B_m. \tag{2}$$

This can be considered a rule where, if all the propositions on the right (All B's) are true, then at least one proposition on the left (one A) must be true. A special case of a clause is the **definite clause**, where there is only one A

$$A \leftarrow B_1 \wedge B_2 \wedge ... \wedge B_m. \tag{3}$$

The definite clause is relevant to argumentation mining, because it represents the structure of a typical argument: if a certain set of premises are true then the arguments claim must be true.

A clause can be rewritten to a form consisting exclusively of a disjunction of literals:

$$A_1 \vee ... \vee A_n \leftarrow B_1 \wedge ... \wedge B_m \quad \Longrightarrow \quad A_1 \vee ... \vee A_n \vee \neg B_1 \vee \neg ... \vee \neg B_m \tag{4}$$

$$A \leftarrow B_1 \wedge B_2 \wedge ... \wedge B_m \quad \Longrightarrow \quad A \vee \neg B_1 \vee \neg B_2 \vee \neg ... \vee \neg B_m \tag{5}$$

This makes it useful to consider clauses as being sets of literals, that are implicitly joined by disjunction. This leads to two equations, which are identical across two notations

$$C = L_1 \vee L_2 \vee ... \vee L_n = \bigvee_{i=1}^{n} L_i \quad \overset{notation}{\Longleftrightarrow} \quad C = \{L_1, L_2, ..., L_n\} = \bigcup_{i=1}^{n} L_i. \tag{6}$$

A **tautology** is a proposition which is true in every interpretation (always true). An example is

$$a \rightarrow a, \tag{7}$$

which is true irrelevantly of the value of a.

An **inconsistency** or contradiction is a compound proposition which can not be true. An example is

$$a \wedge \neg a. \tag{8}$$

3.2 Additional Notation

3.2.1 Sets

As set-theory compliments logic, the following set symbols and operators are commonly used in argument analysis (and other mathematical/computational projects):

$\varnothing$	The empty set
$\mathbb{Z}$	The set of integers
$\mathbb{Z}^+ = \mathbb{N}$	The set of positive inters (natural numbers)
$\mathbb{Z}^*$	The set of positive inters and zero
$\mathbb{Z}^-$	The set of negative inters
$\mathbb{R}$	The set of rational numbers
$\subset$	Proper subset
$\subseteq$	Subset
$\in$	Element in set
$\cap$	Set intersection
$\cup$	Set union

3.2.2 Strings

String-notation is used to mathematically work with strings. ϵ represents the empty string and $\mathbin{+\!\!+}$ denotes concatenation ($A \mathbin{+\!\!+} B$ is the resulting string found from concatenating A and B).

The Kleene star and Kleene plus are useful both in logic and grammars, but also due to their importance in regular expressions.

Definition 3.1. The Kleene star *, also called the free monoid, is a unary operator on sets of strings, which is recursively defined as

$$\begin{aligned} V_0 &= \{\epsilon\} \\ V_i &= \{w \mathbin{+\!\!+} v \mid w \in V_{i-1}, v \in V\} \qquad i \in \mathbb{N} \\ V^* &= \bigcup_{i \in \mathbb{Z}^*} V_i. \end{aligned}$$

The Kleene star can be expressed as a "zero or more"-set. ■

Definition 3.2. The Kleene plus $^+$, is a unary operator on sets of strings, which is recursively defined as

$$\begin{aligned} V_1 &= \{v | v \in V\} \\ V_i &= \{w \mathbin{+\!\!+} v \mid w \in V_{i-1}, v \in V\} \qquad i \in \mathbb{N}\backslash\{1\} \\ V^+ &= \bigcup_{i \in \mathbb{Z}^+} V_i. \end{aligned}$$

The Kleene plus can be expressed as a "one or more"-set and can alternatively be defined as

$$V^+ = V^*\backslash\{\epsilon\}. \tag{9}$$

■

3.3 Grammars

For a computer scientist, a formal **language** is a set of strings and symbols constrained by specified rules. Meaning can often be derived from sentences from the language, but it is not required by the formal definition. The **alphabet** of a language is the set of symbols, letters, or tokens from which the strings of the language may be formed. The alphabet is usually finite. A **grammar** is an exact finite recipe for constructing any sentence in a language, and so describes all constraints of the language. As sentences can be constructed from the grammar, they are said to be *generative*. Languages and grammars are used for many purposes, like for example defining the syntax and structure of programming languages. [28] was used for inspiration to the following descriptions and is recommended for learning more about grammars and parsing.

The building blocks for grammars are **production rules**, which can be logically written in the following as

```
A  ->  B
A  ->  C.
```

The rules state that token `A` can be replaced by either `B` or `C`.

In a frequently used and more compact notation the two rules are replaced by

```
A  ->  B | C,
```

where | here indicates alternatives in the rule. Furthermore a grammar contains **terminal** symbols and **non-terminal** symbols. Terminal symbols are the final symbols that compose the sentences of the language, while non-terminal symbols are intermediate symbols used for the construction.

The formal definition for a generative grammar is now:

Definition 3.3. A generative grammar is a 4-tuple (V_N, V_T, R, S) such that:

1. V_N is a finite set of symbols called the non-terminal symbols.
2. V_T is a finite set of symbols called the terminal symbols.
3. $V_N \cap V_T = \emptyset$
4. R is the rules and is a set of pairs (P, Q) such that $P \in (V_N \cup V_T)^+$ and $Q \in (V_N \cup V_T)^*$
5. $S \in V_N$

■

S is a non-terminal symbol, which is the starting-point of a sentence-generation. Thus to generate a sentence, one starts with S and uses rules to go from there to the wanted terminals. A correctly created sentence contains only non-terminals.

An example grammar is (inspired by [28]):

```
V_N  :  Item, List, End, Sentence
V_T  :  chair, ball, cup
 S   :  Sentence
 R   :  Sentence     ->  Item | List End
        List         ->  Item | Item, List
        Item         ->  chair | ball | cup
        , Item End   ->  and Item
```

This grammar can create any list of the three items, "chair", "ball" and "cup", while correctly placing commas and finishing off the list with an "and". To use the grammar one starts with "Sentence", which can be turned into a single item or into a list of items.

3.3.1 Hierarchy of grammars

The Chomsky Hierarchy of grammars is an ordering of the expressiveness of grammars. Any lower type grammar contains all following grammars.

Type-0 grammar: ***Unrestricted grammars***
This type include all grammars than can be expressed within the above definition.

Type-1 grammar: ***Context-sensitive grammars***
Rules are of the form

$$\alpha A \beta \texttt{ -> } \alpha \gamma \beta, \tag{10}$$

where

$$\alpha, \beta \in (V_T \cup \epsilon), \qquad A \in V_N \qquad \text{and} \qquad \gamma \in (V_N \cup V_T)^+. \tag{11}$$

Thus the rules of context-sensitive grammars changes A while noticing and maintaining the context α and β. Note that α and β can be empty, but γ can not. A rule can thus not decrease the number of symbols. Furthermore these grammars can only change non-terminals, where unrestricted grammars can change anything.

Type-2 grammar: ***Context-free grammars***
Rules are of the form:

$$A \texttt{ -> } \gamma, \tag{12}$$

where

$$A \in V_N \qquad \text{and} \qquad \gamma \in (V_N \cup V_T)^+. \tag{13}$$

Context-free grammars are a special-case of context-sensitive grammars, where $\alpha = \beta = \epsilon$. Again γ can not be empty. Since no terminal symbol will never be used in the head of a rule and only one symbol appears on the left-hand side of the rules, these grammars create tree-structures. Context-free grammars are the theoretical basis for most parsers and programming languages.

Type-3 grammar: ***Regular grammars (left/right)***
Exists in two equivalent forms and have rules like:

$$\begin{array}{ll} \text{(right regular)} & \text{(left regular)} \\ A \texttt{ -> } \gamma B & A \texttt{ -> } B\gamma, \end{array} \tag{14}$$

where

$$A \in V_N, \qquad B \in (V_N \cup \epsilon) \qquad \text{and} \qquad \gamma \in V_T. \tag{15}$$

Here γ can only be one symbol and each rule can maintain at most one non-terminal. These grammars are used in regular expressions, which is used to define search patterns.

3.4 Natural Language Processing (NLP)

Natural Language Processing is the overall field of automated processing natural language (for example English). NLP is generally concerned with text, but, when combined with signal processing, it also creates the foundation for speech recognition, which operates on audio, and optical character recognition, which operates on images. Some of the most known tasks within NLP are (although many other tasks exist as well):

- Automatic Summarization
  Creates an automatically generated summary of a text.
- Machine Translation
  Translates a specified text from one language to another.
- Natural Language Generation
  Translating data in a machine to informative sentences readable by a human.
- Sentence Breaking
  Determining the breaking-points for sentences in a text.
- Sentiment/Opinion Analysis
  Determining the opinion of the writer towards a specific subject.

3.4.1 Parsing and Part-Of-Speech Tagging (POS)

Parsers are systems that are used to analyse the syntactical structure of a sequence of symbols. In context with NLP, parsers are used to detect the structure of sentences and the types of words (part-of-speech). Listing 3.1 shows The Stanford Parser's ([34]) output on a test sentence and illustrates the use. The parser has broken the sentence "This is a classical test sentence." into the individual words and annotated what use the words have. The output shows the structure of the sentence, together with the part-of-speech tags. For example "test" and "sentence" has been classified as NN (noun), while "This" has been classified as DT (determiner).

Listing 3.1: Output from the Stanford Parser with input sentence: "This is a classical test sentence."

```
(ROOT
  (S
    (NP (DT This))
    (VP (VBZ is)
      (NP (DT a) (JJ classical) (NN test) (NN sentence)))
    (. .)))
```

Parsers have often been implemented using context-free grammars, where the possible structures of the related language are noted as rules. A simple example is:

```
V_N : S, NP, VP, V, N, Det, Adj
V_T : dog, cat, sees, meets, big, small, scared, a, the
 S  : S
 R  : S    ->  NP VP
      VP   ->  V NP
      NP   ->  Det N | Det Adj N
      N    ->  dog | cat
      V    ->  sees | meets
      Adj  ->  big | small | scared
      Det  ->  a | the
```

If a parser with this grammar was given the sentence "the big dog meets the scared cat", it would attempt to reconstruct the sentence using the above rules. From this reconstruction it would reach the conclusion, that the part-of-speech tags were "Det Adj N V Det Adj N". The grammars of a language can become very big and traversing the possible search graphs can be very cumbersome. Furthermore there may be different ways to parse the same sentence, in which case it can be useful to denote some probabilities within the grammar (experience from training data), which denotes the probabilities of the rules and therefore each parsing. The Stanford Parser is a probabilistic natural language parser, which can estimate the probabilities of different parsings for a sentence. The specifics of this parser can be found in [34].

3.5 Machine Learning

The following pages describe the used machine learning techniques and is heavily inspired by [14]. N will refer to the number of samples and M the number of dimensions in a feature-space. $||\boldsymbol{x}||$ is the 2-norm / Euclidean length of vector $\boldsymbol{x}$. $\mathbf{1}$ is a vector of 1's, where the length will can be derived from the context.

There are many different machine learning algorithms (or learners), and they differ in their ability to handle different problems and data. The algorithms are made to learn underlying structures and find patterns in data. The data used to train the algorithms will typically exhibit the underlying structure to a certain extend, but may also contain some random variation which is typically not the target interest. Furthermore it can happen that some of the training data has been incorrectly classified or measured, giving rise to **outliers** which are not useful for learning. Two important terms in machine learning are the concepts of **overfitting** and **bias**. Overfitting occurs when an algorithm is sufficiently flexible to learn, not only the underlying structure of the training data, but also the noise that may occur in the samples. By learning this random variation the model becomes unable to generalize when given new data, as the new data have been affected differently by the noise. Bias occurs when the algorithm is not flexible enough to capture the underlying structure. The result is a system that is unable to learn the "true" model, irrelevantly of how small the noise may be or how many samples are given. With such a system one could incorrectly blame the deficiencies of the system on the noise of the data, which may be smaller than observed with the learner.

When some algorithms are given very high-dimensional data, a phenomenon called the **Curse of Dimensionality** can occur. Having many dimensions creates many internal parameters in the algorithms, whose value needs to be learned from the training data. For many machine learning schemes there will be one or more parameters per feature. This creates situations where the machine learning scheme has many degrees of freedom relative to (and possible more than) the number of datapoints. The high flexibility of the algorithm allows it to learn the variation of the data causing overfitting. It can thus be attractive to reduce the number of features to a smaller feature space (which hopefully still captures the target structure of the data).

In machine learning it is common practise to split data into **test**-data and **training**-data. The training data is used to train and optimize learners, while the independent test set is used to evaluate the final performance. In order to correctly measure performance, the test-set can not be used to train or optimize any parameters of the learning algorithm, as this could bias the final performance estimate (due to overfitting and inability to generalize). When splitting data it is important to keep enough data both for the training part, but also for evaluating performance, as small test-sets can create variance in performance estimates. ***k*-Cross validation** is a method used to get good performance estimates, where the data is split into k parts. Each of the parts i taken out as a test-set, after which an algorithm is trained on the remaining $k-1$ parts (creating a total of k trained models). Each model is then evaluated on the independent test set, and their performances averaged. This reduces the bias in the performance estimate from averaging, as well as allowing the user to train and test with all datapoints. Cross validation is used internally in the training set, when optimizing learning algorithms, although the final performance is measured on a single test set, as cross-validation on the entire project's experiments was computationally too heavy. In some cases there may be hyper-parameters which the learning algorithm can not train itself. These parameters are often tried within some interval, after which a **validation** set is used to select the best choice of parameters. Finally the test set is again used for estimating performance. The training set, validation set and test set are all independent. Another important aspect of splitting data, is how the classes or target values are distributed in the test and training set. In order to make both sets representative of the original distribution one makes a **stratified split**, which splits data into subsets with similar distributions of the target values or classes. This method is used for all splits in this project.

3.5.1 Probability Theory and Bayes

As commonly practices in probability theory, in the following $p(x)$, $p(x, y)$ and $p(x|y)$ will denote the probability of x, the joint probability of x and y and the conditional probability of x given y respectively.

An important theorem in machine learning is Bayes theorem which states

$$p(y \mid x) = \frac{p(x \mid y)p(y)}{p(x)}. \tag{16}$$

The theorem is commonly used in statistical modelling, where x represents some sampled data and y represents some hypothesis/model on the original distribution. Here $p(y \mid x)$ is called the *posterior probability* of y given x, $p(x)$ and $p(y)$ are called the *prior probabilities* of x and y, and $p(x \mid y)$ is called the *likelihood* of y.

Machine learning is typically about finding the most probable model or prediction for some data. If x is the data then y would be a model or a prediction, of which we want to select the most probable given x. As $p(x)$ is independent of the choice of model it can be disregarded when searching for the most probable model,

$$p(y \mid x) \propto p(x \mid y)p(y). \tag{17}$$

The *maximum a posteriori* estimate is the model that maximizes this quantity,

$$y_{MAP} = \arg\max_y p(x \mid y)p(y). \tag{18}$$

A common, simplifying assumption is that the prior of the model is uniform, which further simplifies the relation,

$$p(y \mid x) \propto p(x \mid y). \tag{19}$$

This gives rise to the *maximum likelihood* solution y_{ML} to machine learning problems, which selects the model that maximizes the likelihood,

$$y_{ML} = \arg\max_y p(x \mid y). \tag{20}$$

Simplifying with Logarithms

The likelihood of model y given dataset S is computed by (assuming independence between samples)

$$p(S \mid y) = \prod_{x \in S} p(x \mid y). \tag{21}$$

Instead of maximizing the likelihood directly it is often easier to maximize the *log-likelihood*, as it maps the product of the probabilities to a sum

$$\ln\big(p(S \mid y)\big) = \sum_{x \in S} \ln\big(p(x \mid y)\big), \tag{22}$$

which allows for easier differentiation. As probabilities are in the interval $[0, 1]$ the log-likelihood will be in the interval $[-\infty, 0]$.

Similarly, the maximum a posteriori model can be found by maximizing

$$p(S \mid y) = \prod_{x \in S} p(x \mid y)p(y) \tag{23}$$

and

$$\ln\big(p(S \mid y)\big) = \sum_{x \in S} \ln\big(p(x \mid y)p(y)\big) = \sum_{x \in S} \ln\big(p(x \mid y)\big) + |S| \ln\big(p(y)\big). \tag{24}$$

Like previously the logarithmic sum will be in the interval $[-\infty, 0]$.

As many optimization algorithms work by minimizing expressions, the above are typically converted to a loss-function by negating the terms, so that

$$y_{MAP} = \arg\min_y -\left(\sum_{x \in S} \ln\big(p(x \mid y)\big) + |S| \ln\big(p(y)\big)\right) \tag{25}$$

and

$$y_{ML} = \arg\min_{y} -\sum_{x \in S} \ln\big(p(x \mid y)\big), \tag{26}$$

where the minimization function in the latter is called the *negative log-likelihood*. Both these function are in the interval $[0, \infty]$. The second term in the minimization function of the maximum a posteriori loss-function is called a regularization term and constitutes restrictions on the parameters of the model.

Regularization

The maximum likelihood solution has the advantage that no prior knowledge is needed. This simplifies modelling and also avoids biased prior-function chosen by the modeller, which may incorrectly alter the results. Not incorporating a prior can also be a disadvantage. Maximum likelihood solutions tend to find very extreme solutions, which the inclusion of prior probabilities can avoid. An example is to consider a weather prediction system that starts learning on a Monday. If it happens to rain Monday, Tuesday and Wednesday, then the maximum likelihood model will predict rain for the remainder of the week (and all of time). A prior distribution on the model could avoid this, by including some knowledge about the number of rainy days in a year.

To avoid this sort of overfitting, the maximum a posteriori solution can be determined by using *regularization*. Regularization are the added terms in a loss-function, which puts restrictions on the model's ability to form itself after the data. Regularization parameters can be difficult to optimize: if too much regularisation is used the model will be restricted and unable to fit the data very, causing an error from bias. If not enough regularization is used the problems of overfitting enter as mentioned above. It is common to test different levels of regularization and choose the best one on a validation set.

3.5.2 Regression Problems

In a regression problem, $y(\boldsymbol{x}, \boldsymbol{w})$ is made to predict the value of an unknown function $f(\boldsymbol{x})$, based on the shared sample-vector $\boldsymbol{x}$. In many cases $f(\boldsymbol{x})$ is stochastic, which makes perfect regression impossible. In these situations a probabilistic view is applied, and the unknown function $f(\boldsymbol{x})$ is replaced by a joint probability $p(t, \boldsymbol{x})$ describing the probability of seeing t and $\boldsymbol{x}$. $p(\boldsymbol{t}, \boldsymbol{X})$ is the multi-sample case, where $\boldsymbol{X}$ contains a column for each datapoint, and $\boldsymbol{t}$ contains the function's evaluation for each datapoint.

L2 Loss Function

A common assumption is that the noise of data is normally distributed and independent between datapoints. Although this assumption can sometimes be incorrect, it tends to hold in many cases and give fairly good results. Under this assumption, a probabilistic model would be

$$p(\boldsymbol{t} \mid \boldsymbol{X}, \boldsymbol{w}, s) = \prod_{n=1}^{N} \mathcal{N}(t_n \mid y(\boldsymbol{x}_n, \boldsymbol{w}), s). \tag{27}$$

With the normal distribution defined as

$$\mathcal{N}(t \mid y(\boldsymbol{x}, \boldsymbol{w}), s) = \frac{1}{s\sqrt{2\pi}} \exp\left(-\frac{1}{2s^2}\big(t - y(\boldsymbol{x}, \boldsymbol{w})\big)^2\right). \tag{28}$$

Here a regression model $y(\boldsymbol{x}, \boldsymbol{w})$ is used to estimate the mean of a normal-distribution, after which the standard deviation s is determined through simple statistics. Equation 27 is also the likelihood of the model.

The log-likelihood is:

$$\ln\left(p(t \mid \boldsymbol{x}, \boldsymbol{w}, s)\right) = -\frac{1}{2s^2}\sum_{n=1}^{N}(t_n - y(\boldsymbol{x}_n, \boldsymbol{w}))^2 + N\ln\left(\frac{1}{s\sqrt{2\pi}}\right) \tag{29}$$

$$= -\frac{1}{2s^2}\sum_{n=1}^{N}(t_n - y(\boldsymbol{x}_n, \boldsymbol{w}))^2 + N\ln\left(\frac{1}{s}\right) + N\ln\left(\frac{1}{\sqrt{2\pi}}\right) \tag{30}$$

$$= -\frac{1}{2s^2}\sum_{n=1}^{N}(t_n - y(\boldsymbol{x}_n, \boldsymbol{w}))^2 - N\ln(s) - \frac{N}{2}\ln(2\pi). \tag{31}$$

Maximizing the log-likelihood amounts to minimizing the sum of squared errors between the model and the given samples,

$$\boldsymbol{w}_{ML} = \arg\min_{\boldsymbol{w}}\left(L2(\boldsymbol{w})\right) = \arg\min_{\boldsymbol{w}}\sum_{n=1}^{N}(t_n - y(\boldsymbol{x}_n, \boldsymbol{w}))^2. \tag{32}$$

This is the motivation behind the *sum-of-squares* loss function. It maximizes the likelihood of the model, given that the noise of the data is normally distributed,

$$L_2(\boldsymbol{w}) = \sum_{n=1}^{N}(t_n - y(\boldsymbol{x}_n, \boldsymbol{w}))^2. \tag{33}$$

The loss function is called the L2 loss function, because it computes the squared *L2 norm* of the difference between the target vector $\boldsymbol{t}$ and the predicted vector $\boldsymbol{y}(\boldsymbol{x}_n, \boldsymbol{w})$, as in

$$L_2(\boldsymbol{w}) = ||\boldsymbol{t} - \boldsymbol{y}(\boldsymbol{x}_n, \boldsymbol{w})||_2^2. \tag{34}$$

R2-Regularization

A common prior on the parameters of the linear model, is to assume a multivariate Gaussian distribution on the parameters of the model, with zero means and with covariance matrix being a diagonal-matrix with identical diagonal-elements β, which is defined by

$$p(\boldsymbol{w}) = \frac{1}{\sqrt{(2\pi)^k\beta^k}}\exp\left(-\frac{\beta}{2}\boldsymbol{w}^T\boldsymbol{w}\right). \tag{35}$$

The negative logarithm of this expression becomes:

$$-\ln\left(p(\boldsymbol{w})\right) = -\ln\left(\frac{1}{\sqrt{(2\pi)^k\beta^k}}\exp\left(-\frac{\beta}{2}\boldsymbol{w}^T\boldsymbol{w}\right)\right)$$

$$= -\ln\left(\frac{1}{\sqrt{(2\pi)^k\beta^k}}\right) + \frac{\beta}{2}\boldsymbol{w}^T\boldsymbol{w}, \tag{36}$$

which is proportional to

$$\ln\big(p(\boldsymbol{w})\big) \propto \frac{\beta}{2}\boldsymbol{w}^T\boldsymbol{w}. \tag{37}$$

This lead to the regularization of the L2-loss function by adding the term

$$L_2(\boldsymbol{w}) + R_2(\boldsymbol{w}) = L_2(\boldsymbol{w}) + \lambda\frac{1}{2}||\boldsymbol{w}||_2^2 = L_2(\boldsymbol{w}) + \lambda\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}. \tag{38}$$

λ is the regularisation parameter, and determines how much the model should be penalised for large weights. Through regularising, weights related to features, whose relevance are not supported by data, will be forced to have small values. Furthermore the regularization keeps the loss-function quadratic in $\boldsymbol{w}$, which is advantageous for the solver used to optimize the weights. This type of regularisation (adding a term that penalises size of weights) is called *weight decay*, because it keeps weights small.

L1 Loss Function

The L2 norm is, together with the L1 norm, part of the L^p-space, which defines a set of length measures of vectors, defined as

$$||\boldsymbol{x}||_p = \left(\sum_{m=1}^{M} x_m^p\right)^{\frac{1}{p}}. \tag{39}$$

More specifically

$$||\boldsymbol{x}||_1 = \sum_{m=1}^{M} |x_m|, \tag{40}$$

$$||\boldsymbol{x}||_2 = ||\boldsymbol{x}|| = \sqrt{\sum_{m=1}^{M} x_m^2}, \tag{41}$$

$$||\boldsymbol{x}||_\infty = \max_m(x_m). \tag{42}$$

The L1-norm (40) is another typical choice for a loss-function, so that y is selected by minimizing

$$L_1(\boldsymbol{w}) = ||\boldsymbol{t} - \boldsymbol{y}(\boldsymbol{x}_n, \boldsymbol{w})||_1. \tag{43}$$

Instead of assumed Gaussian noise, the L1 loss-function comes from assuming a Laplace distributed noise, so that

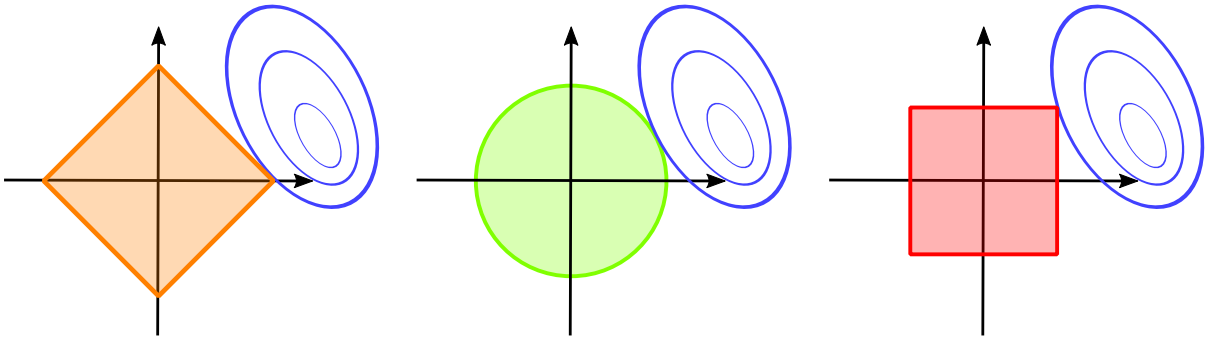
$$p(t_n \mid \boldsymbol{X}, \boldsymbol{w}, b) = \frac{1}{2b}\exp\left(-\frac{|t_n - y(\boldsymbol{x}_n, \boldsymbol{w})|}{b}\right). \tag{44}$$

From the negative logarithm of product of $p(t_n \mid \boldsymbol{X}, \boldsymbol{w}, b)$ over all t_n's, an expression proportional to 43 is found.

Regularisation of the L1 loss-function is done with another type of weight decay

$$L_1(\boldsymbol{w}) + R_1(\boldsymbol{w}) = L_1(\boldsymbol{w}) + \lambda||\boldsymbol{w}||_1. \tag{45}$$

This regularisation keeps the loss function in the same form (linear), with respect to $\boldsymbol{w}$, and is motivated by assuming a Laplace-distributed probability on the parameters.



(a) R1-regularisation. ***(b)*** R2-regularisation. ***(c)*** L∞-regularisation.

Figure 3.1: Three types of regularisations. The figure is extrapolated from figure 3.3 in [14]. Weight decay regularisations restricts the parameter-search to be within a restricted area. All solutions within the area are "cheaper" that those outside of the area. A solution outside of the area thus needs to be considerably better in order for the algorithm to use it. The R2-regularisation uses a circular (spherical in multi-dimensional cases) search-area, restricting the total length of the weight vector. The R1-regularisation will often find values of either 1 or 0. This creates a tendency to make the weight vectors sparse, which can be useful for detecting the most relevant features and making the weight vector easier to understand. The R∞-regularisation typically sets all weights to the same value, and thus simply varies the length of the weight-vector when learning.

Table 3.2: Explanation of the nature of the regularisations. Assume $a > b >> \delta > 0$ and $a - b > \delta$.

$\boldsymbol{w} = \begin{pmatrix} a \\ b \end{pmatrix}$	$\boldsymbol{\delta}_0 = \begin{pmatrix} \delta \\ 0 \end{pmatrix}$	$\boldsymbol{\delta}_1 = \begin{pmatrix} 0 \\ \delta \end{pmatrix}$
$R_1(\boldsymbol{w}) = a + b$	$R_1(\boldsymbol{w} + \boldsymbol{\delta}_0) = a + b + \delta$	$R_1(\boldsymbol{w} + \boldsymbol{\delta}_1) = a + b + \delta$
$R_2(\boldsymbol{w}) = a^2 + b^2$	$R_2(\boldsymbol{w} + \boldsymbol{\delta}_0) = a^2 + 2a\delta + \delta^2 + b^2$	$R_2(\boldsymbol{w} + \boldsymbol{\delta}_1) = a^2 + b^2 + 2b\delta + \delta^2$
$R_\infty(\boldsymbol{w}) = a$	$R_\infty(\boldsymbol{w} + \boldsymbol{\delta}_0) = a + \delta$	$R_\infty(\boldsymbol{w} + \boldsymbol{\delta}_1) = a$

Table 3.2 shows how the regularisations act. Assume a weight vector contains two values where the first element (a) is larger than the other (b). When using the R1-regularisation adding a small value to either weight increases the regularisation term equally. If feature a is slightly more useful than feature b the learner might as well set a to 1 and b to 0, which causes the sparsity. In the R2-regularisation the term δ^2 is practically negligible. Furthermore adding a term to b is less costly than adding a term to a, since $2a\delta > 2b\delta$. Therefore having a small values for b is very cheap, which keeps b larger than 0, even if the feature with weight a is more useful. In the R∞-regularisation it only becomes more costly to increase the value of a. Thus b can be chose to be anything in the interval $[0; a]$, which causes the algorithm to typically choose the same value for all weights.

3.5.3 Classification Problems

In machine learning, a classification problem is the problem of creating a function $y(\boldsymbol{x}, \boldsymbol{w})$ which takes a sample's vector $\boldsymbol{x}$ together with a parameter-vector $\boldsymbol{w}$ as input, and selects one of $\mathcal{C}$ classes related to the sample described by $\boldsymbol{x}$. $y(\boldsymbol{x}, \boldsymbol{w})$ can be made to output the actual class, but it is also common to let the sign of $y(\boldsymbol{x}, \boldsymbol{w})$ decide the class, so that

$$\text{Prediction}(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \ y(\boldsymbol{x}, \boldsymbol{w}) > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{46}$$

In the classification perspective, one can state the probability of a sample $\boldsymbol{x}$ belonging to a class C_k, using Bayes theorem (equation 16), as

$$p(C_k \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid C_k)p(C_k)}{p(\boldsymbol{x})}. \tag{47}$$

The prior probability of $\boldsymbol{x}$, in the multi-class problem, is given by

$$p(\boldsymbol{x}) = \sum_{i=1}^{\mathcal{C}} p(\boldsymbol{x} \mid C_i)p(C_i). \tag{48}$$

Modelling of the density distributions of the classes therefore allows evaluation of the posterior probabilities $p(C_i \mid \boldsymbol{x})$, with

$$p(C_k \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid C_k)p(C_k)}{\sum_{i=1}^{\mathcal{C}} p(\boldsymbol{x} \mid C_i)p(C_i)}. \tag{49}$$

3.5.4 Linear Regression

An often used regression method is linear regression. A linear model has the form

$$y(\boldsymbol{x}, \boldsymbol{w}) = w_0 + \sum_{m=1}^{M} x_m w_m, \tag{50}$$

where M is the dimension of the feature-space $\boldsymbol{x}$. w_0 is commonly referred to as the *bias*, while the remainder of $\boldsymbol{w}$ is called the *weights*. The sum can be expressed as a dot-product between the features and the weights

$$y(\boldsymbol{x}, \boldsymbol{w}) = w_0 + \boldsymbol{x}^T \boldsymbol{w}. \tag{51}$$

If one includes the bias in the weight-vector and prepends a 1 to all feature-vectors, the linear model can be expressed as a single dot-product

$$y(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{x}^T \boldsymbol{w}. \tag{52}$$

If the sum-of-squares loss function is used, linear regression chooses the model $y(\boldsymbol{x}, \boldsymbol{w})$ with parameters given by

$$\boldsymbol{w} = \arg\min_{\boldsymbol{w}} \sum_{n=1}^{N} (t_n - \boldsymbol{x}_n^T \boldsymbol{w})^2. \tag{53}$$

3.5.5 Logistic Regression

As shown earlier, the posterior distribution of a class can be determined by

$$p(C_k \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid C_k)p(C_k)}{\sum_{i=1}^{\mathcal{C}} p(\boldsymbol{x} \mid C_i)p(C_i)}. \tag{54}$$

This can be rewritten as

$$p(C_k \mid \boldsymbol{x}) = \frac{\exp(a_k)}{\sum_{i=1}^{\mathcal{C}} \exp(a_i)} \tag{55}$$

$$a_i = \ln(p(\boldsymbol{x} \mid C_i)p(C_i)). \tag{56}$$

Equation (55) is called the *softmax* function or the *normalized exponential*. The softmax formulation is useful because it will always output a number in the interval $[0; 1]$, while the classes' softmax-evaluations will sum to one. Therefore, if one approaches the classification task by approximating the a_i's, the softmax can ensure that the output can be used as a probability distribution.

In the two-class case, the normalized exponential is

$$p(C_1 \mid \boldsymbol{x}) = \frac{\exp(a_1)}{\exp(a_1) + \exp(a_2)} \tag{57}$$

$$= \frac{1}{1 + \frac{\exp(a_2)}{\exp(a_1)}} \tag{58}$$

$$= \frac{1}{1 + \exp(a_2 - a_1)} \tag{59}$$

$$= \frac{1}{1 + \exp(-a)} = \sigma(a). \tag{60}$$

Where the difference in the exponential has been replaced by

$$a = a_1 - a_2 \tag{61}$$

$$= \ln(p(\boldsymbol{x} \mid C_1)p(C_1)) - \ln(p(\boldsymbol{x} \mid C_2)p(C_2)) \tag{62}$$

$$= \ln\left(\frac{p(\boldsymbol{x} \mid C_1)p(C_1)}{p(\boldsymbol{x} \mid C_2)p(C_2)}\right). \tag{63}$$

The probability for the other class can be computed by a similar expression or simply with

$$p(C_2 \mid \boldsymbol{x}) = 1 - p(C_1 \mid \boldsymbol{x}). \tag{64}$$

Equation (60) describes the *logistic sigmoid* function σ. Estimating a using linear regression give way to *logistic regression*, where the posterior distribution is estimated as

$$p(C_1 \mid \boldsymbol{x}) = y(\boldsymbol{x}) = \sigma(\boldsymbol{x}^T \boldsymbol{w}). \tag{65}$$

This allows linear regression to be used for estimating probabilities in a classification task, even though the evaluation of the linear expression could be any real number.

For logistic regression, L1 and L2 loss functions are imported from linear regression, together with their related regularisations R1 and R2. If the noise of the data is Gaussian, one would

expect L2+R2 to give most accurate results. L1+R1 can be used to create more sparse solutions and detect i relevant features.

3.5.6 Kernel Methods

The linear models use the $\boldsymbol{x}$-features directly. A different approach is to first process the features with a function, and then use the results as new features. This approach can handle non-linearity in data, as the function can be non-linear. The approach is illustrated as

$$y(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{x} \qquad \Rightarrow \qquad y(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n). \tag{66}$$

Where $\boldsymbol{\phi}$ is a non-linear function that produces a new vector from the input vector. With the L2-loss function and R2-regularisation, a model is chosen to minimize

$$L_2(\boldsymbol{w}) = \frac{1}{2} \sum_{n=1}^{N} (\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n) - t_n)^2 + \frac{\lambda}{2} \boldsymbol{w}^T \boldsymbol{w}. \tag{67}$$

For finding a minima, the derivative with respect to $\boldsymbol{w}$ is determined as

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{w}} L_2(\boldsymbol{w}) = \sum_{n=1}^{N} (\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n) - t_n) \boldsymbol{\phi}(\boldsymbol{x}_n) + \lambda \boldsymbol{w}. \tag{68}$$

Setting this to zero a doing bit of rearranging produces

$$\boldsymbol{w} = -\frac{1}{\lambda} \sum_{n=1}^{N} (\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n) - t_n) \boldsymbol{\phi}(\boldsymbol{x}_n) = \sum_{n=1}^{N} a_n \boldsymbol{\phi}(\boldsymbol{x}_n) \tag{69}$$

$$a_n = -\frac{1}{\lambda} (\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n) - t_n). \tag{70}$$

It appears the weight vector $\boldsymbol{w}$ can be expressed as a linear combination of the $\boldsymbol{\phi}(\boldsymbol{x}_n)$'s. By estimating the a_n's instead of the weight vector, the fitting problem is changed from having as many parameters as the dimensionality of the input vector-space, to having as many parameters as there are datapoints. If one has a very large feature-space, this becomes a very attractive property as it reduces the curse of dimensionality.

The regression function $y(\boldsymbol{x}, \boldsymbol{w})$ determines a single evaluation for one sample. A multi-sample version can be expressed as

$$\boldsymbol{y}(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{\Phi}^T, \tag{71}$$

where $\boldsymbol{\Phi}$ are defined as

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\phi}(\boldsymbol{x}_1)^T \\ \boldsymbol{\phi}(\boldsymbol{x}_2)^T \\ \vdots \\ \boldsymbol{\phi}(\boldsymbol{x}_n)^T \end{bmatrix}. \tag{72}$$

The sum from (69) can similarly be converted to matrix-vector notation by

$$\boldsymbol{w} = \sum_{n=1}^{N} a_n \boldsymbol{\phi}(\boldsymbol{x}_n) = \boldsymbol{\Phi}^T \boldsymbol{a} \qquad \boldsymbol{a} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}. \tag{73}$$

By inserting this into 71 we have

$$\boldsymbol{y}(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{w}^T \boldsymbol{\Phi}^T = (\boldsymbol{\Phi}^T \boldsymbol{a})^T \boldsymbol{\Phi}^T = \boldsymbol{a}^T \boldsymbol{\Phi} \boldsymbol{\Phi}^T = \boldsymbol{a}^T \boldsymbol{K}^T. \tag{74}$$

The matrix product on the right computes a different matrix called the *kernel matrix* $\boldsymbol{K}$, which is the collection of dot-products between all kernel-vectors

$$\boldsymbol{\Phi}\boldsymbol{\Phi}^T = \boldsymbol{K}^T \qquad K_{n,m} = \boldsymbol{\phi}(\boldsymbol{x}_n)^T \boldsymbol{\phi}(\boldsymbol{x}_m). \tag{75}$$

Given a kernel $\boldsymbol{\phi}$ it is thus possible to fit a linear regression using $O(M)$ parameters, by first computing the dot products between the kernel-vectors.

Another important advantage of using kernels is the ability to use **kernel substitutions**, which replaces $\boldsymbol{\phi}$ with various interesting functions. This enables user to create many interesting kernel matrices with different properties. A few common kernels are described in 3.5.7 under *SVM Kernels*.

3.5.7 Support Vector Machine

Support Vector Machine (SVM) is a machine learning concept, which approaches classification problems from a geometric point of view. An SVM attempts to create a decision boundary between two classes, which maximizes the distance from any point to the boundary. The distance from the closest point to the boundary is called the *margin*.

First note the following geometric properties. A hyperplane is defined by a perpendicular vector $\boldsymbol{v}$ and an offset v_0, so that any point $\boldsymbol{x}$ which is on the hyperplane satisfies

$$\boldsymbol{v}^T \boldsymbol{x} - v_0 = 0. \tag{76}$$

v_0 thus represents the distance from the origin to the hyperplane. The coordinate distance from $\boldsymbol{x}$ to the hyperplane is given by

$$d = \frac{\boldsymbol{v}^T \boldsymbol{x} - v_0}{||\boldsymbol{v}||}. \tag{77}$$

In case the hyperplane is defined using a unit-vector the distance is simply $d = \boldsymbol{v}^T \boldsymbol{x} - v_0$.

Now assume a classification problem, in which the features $\boldsymbol{x}$ have been cast by a kernel $\boldsymbol{\phi}$ into a space $\boldsymbol{\phi}(\boldsymbol{x})$ where two classes (C_1, C_2) are linearly separable. In this space, a support vector machine will find the hyperplane, separating the two classes, with the largest margin. The hyperplane is defined by $\boldsymbol{w}$ and b, with

$$\text{hyperplane:} \qquad y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) + b = 0. \tag{78}$$

The side of the hyperplane, which a sample resides on, can be determined by the sign of the hyperplane expression

$$\text{side:} \qquad \text{sign}(\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) + b). \tag{79}$$

Assume a set of training data $\{x_n, t_n\}, n \in [1, N]$. Where t_n is the class of sample described by x_n, with class values being -1 and 1. If the hyperplane correctly classifies all samples in the training set, one can assess that

$$t_n y(\boldsymbol{x}_n) > 0 \qquad n \in [0, N]. \tag{80}$$

The distance to the hyperplane is

$$\text{distance:} \qquad d = \frac{|\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) + b|}{||\boldsymbol{w}||} = \frac{|y(\boldsymbol{x}_n)|}{||\boldsymbol{w}||} = \frac{t_n y(\boldsymbol{x}_n)}{||\boldsymbol{w}||}. \tag{81}$$

The SVM attempts to maximize the minimum distance (81) from any point to the hyperplane. That is

$$\arg\max_{\boldsymbol{w},b} \left(\frac{1}{||\boldsymbol{w}||} \min_n \left[t_n y(\boldsymbol{x}_n) \right] \right). \tag{82}$$

The same hyperplane can be described with a vector $\boldsymbol{w}$ times any factor. Using this freedom, one can condition the problem, so that the datapoint closest to the hyperplane has a distance of 1 and

$$\forall n \in [1, N] : \; t_n y(\boldsymbol{x}_n) \geqslant 1, \tag{83}$$

$$\exists n \in [1, N] : \; t_n y(\boldsymbol{x}_n) = 1, \tag{84}$$

$$\Rightarrow \qquad \min_n \left[t_n y(\boldsymbol{x}_n) \right] = 1. \tag{85}$$

The maximization problem now becomes

$$\arg\max_{\boldsymbol{w},b} \frac{1}{||\boldsymbol{w}||}. \tag{86}$$

This is equivalent to the minimization problem

$$\begin{gathered} \arg\min_{\boldsymbol{w},b} \frac{1}{2} ||\boldsymbol{w}||^2 \\ \text{Subject to:} \quad t_n y(\boldsymbol{x}_n) \geqslant 1. \end{gathered} \tag{87}$$

Where b is implicitly determined as it appears in the constraints.

Before solving this problem, the initial assumption that the classes are linearly separable in the $\boldsymbol{\phi}(\boldsymbol{x})$ space is reconsidered. In many situations this will not be so, and allowing a few samples to be misclassified is necessary for making the SVM work on the dataset. This is done by softening the (83)-constraint to

$$t_n y(\boldsymbol{x}_n) \geqslant 1 - \xi_n \tag{88}$$

$$\xi_n \geqslant 0. \tag{89}$$

Datapoints for which the so-called slack variables $\xi_n = 0$, are correctly classified (like previously), where the equality holds for the points closest to the hyperplane. When $0 < \xi_n \leqslant 1$

the datapoint is allowed within the margin (and potentially on the hyperplane) of the SVM. Finally when $\xi_n > 1$ the datapoints are incorrectly classified and reside on the wrong side of the hyperplane.

The change in constraints allow the SVM to softly penalize incorrect classifications, by solving

$$\arg\min_{\boldsymbol{w},b} \left(C \sum_{n=1}^{N} \xi_n + \frac{1}{2}||\boldsymbol{w}||^2 \right). \tag{90}$$

C controls the penalty for misclassification. As any misclassification has $\xi_n > 1$, $\sum_{n=1}^{N} \xi_n$ is an upper limit of the number of misclassifications. Therefore C can be considered the penalty per misclassification. A small C will allow many misclassifications and $C \to \infty$ creates the SVM that attempts to perfectly classify the training samples.

The final problem becomes

$$\arg\min_{\boldsymbol{w},b} \left(C \sum_{n=1}^{N} \xi_n + \frac{1}{2}||\boldsymbol{w}||^2 \right) \tag{91}$$

$$\text{Subject to:} \quad t_n y(\boldsymbol{x}_n) \geqslant 1 - \xi_n \tag{92}$$

$$\xi_n \geqslant 0. \tag{93}$$

In order to handle this problem the following Lagrangian with Karush-Kuhn-Tucker (KKT) terms is created

$$L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{a}, \boldsymbol{\mu}) = \underbrace{- \sum_{n=1}^{N} a_n \Big(t_n \big(\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) + b \big) - 1 + \xi_n \Big)}_{\substack{\text{KKT term for} \\ \text{Condition (92)}}} \underbrace{- \sum_{n=1}^{N} \mu_n \xi_n}_{\substack{\text{KKT term for} \\ \text{Condition (93)}}} + \underbrace{\frac{1}{2}||\boldsymbol{w}||^2}_{\text{Objective}} + \underbrace{C \sum_{n=1}^{N} \xi_n}_{\substack{\text{Misclassification} \\ \text{penalty}}} \tag{94}$$

The KKT conditions are the necessary conditions for a minima

$$t_n y(\boldsymbol{x}_n) - 1 + \xi_n \geqslant 0 \tag{95}$$

$$\xi_n \geqslant 0 \tag{96}$$

$$a_n (t_n y(\boldsymbol{x}_n) - 1 + \xi_n) = 0 \tag{97}$$

$$a_n \geqslant 0 \tag{98}$$

$$\mu_n \geqslant 0 \tag{99}$$

$$\mu_n \xi_n = 0. \tag{100}$$

When 94 is minimized, the system will attempt to minimize all ξ_n's. The samples that can be correctly classified will reach $\xi_n = 0$ (condition (96)). Any sample, that cannot be correctly classified or lies in the margin, will have $\xi_n > 0$ (condition (95)). The a_n's will be zero for all samples that can not be correctly classified and lies outside of the margin (condition (97)). For

correctly classified samples, a_n will be positive (condition (98)), as that will add negative terms to 94 which is to be minimized, and thus create the support vectors.

The derivatives of the Lagrangian (shortened L) is

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{w}}L = \boldsymbol{w} - \sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\boldsymbol{x}_n) \quad \xrightarrow{\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{w}}L=0} \quad \boldsymbol{w} = \sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\boldsymbol{x}_n) \tag{101}$$

$$\frac{\mathrm{d}}{\mathrm{d}b}L = -\sum_{n=1}^{N} a_n t_n \quad \xrightarrow{\frac{\mathrm{d}}{\mathrm{d}b}L=0} \quad \sum_{n=1}^{N} a_n t_n = 0 \tag{102}$$

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\xi}}L = C\mathbf{1} - \boldsymbol{a} - \boldsymbol{\mu} \quad \xrightarrow{\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\xi}}L=0} \quad a_n = C - \mu_n. \tag{103}$$

By inserting 101, 102 and 103 into the Lagrangian, a different version is obtained as derived on the next page.

$$L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{a}, \boldsymbol{\mu}) = -\sum_{n=1}^{N} a_n\Big(t_n\big(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) + b\big) - 1 + \xi_n\Big) - \sum_{n=1}^{N}\mu_n\xi_n + \frac{1}{2}||\boldsymbol{w}||^2 + \sum_{n=1}^{N} C\xi_n \tag{104}$$

(101)

$$= -\sum_{n=1}^{N} a_n\left(t_n\left(\left(\sum_{m=1}^{N} a_m t_m \boldsymbol{\phi}(\boldsymbol{x}_m)\right)^T \boldsymbol{\phi}(\boldsymbol{x}_n) + b\right) - 1 + \xi_n\right) \tag{105}$$

(101), (103)

$$- \sum_{n=1}^{N}\mu_n\xi_n + \frac{1}{2}\left\|\sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\boldsymbol{x}_n)\right\|^2 + \sum_{n=1}^{N}(a_n + \mu_n)\xi_n$$

$$= -\sum_{n=1}^{N} a_n\left(t_n\sum_{m=1}^{N} a_m t_m \boldsymbol{\phi}(\boldsymbol{x}_m)^T\boldsymbol{\phi}(\boldsymbol{x}_n) + t_n b - 1 + \xi_n\right) \tag{106}$$
$$- \sum_{n=1}^{N}\mu_n\xi_n + \frac{1}{2}\left(\sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\boldsymbol{x}_n)\right)^T\left(\sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\boldsymbol{x}_n)\right)$$
$$+ \sum_{n=1}^{N} a_n\xi_n + \sum_{n=1}^{N}\mu_n\xi_n$$

$$= -\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m \boldsymbol{\phi}(\boldsymbol{x}_m)^T\boldsymbol{\phi}(\boldsymbol{x}_n) - b\sum_{n=1}^{N} a_n t_n + \sum_{n=1}^{N} a_n \tag{107}$$
$$+ \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m \boldsymbol{\phi}(\boldsymbol{x}_n)^T\boldsymbol{\phi}(\boldsymbol{x}_m)$$
$$+ \sum_{n=1}^{N} a_n\xi_n - \sum_{n=1}^{N} a_n\xi_n$$

(102)

$$= -\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m \boldsymbol{\phi}(\boldsymbol{x}_m)^T\boldsymbol{\phi}(\boldsymbol{x}_n) + \sum_{n=1}^{N} a_n \tag{108}$$

$$L(\boldsymbol{a}) = -\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\boldsymbol{x}_n, \boldsymbol{x}_m) + \sum_{n=1}^{N} a_n \tag{109}$$

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \boldsymbol{\phi}(\boldsymbol{x}_m)^T\boldsymbol{\phi}(\boldsymbol{x}_n). \tag{110}$$

(109) is a dual representation of the SVM's Lagrangian which uses an N-long vector of parameters $\boldsymbol{a}$ and uses a kernel k.

SVM Kernels

A few commonly used kernels for SVM are:

Linear

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \boldsymbol{x}_n^T \boldsymbol{x}_m \tag{111}$$

The dot-product between two vectors indicates the similarity between two vectors, which the linear kernel uses.

Polynomial

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = (\gamma \boldsymbol{x}_n^T \boldsymbol{x}_m + r)^d \tag{112}$$

The polynomial kernel creates all terms of the form

$$\prod_{s \in S} x_{ns} x_{ms} \qquad S = \{s \mid s \in [1, N], s \in \mathbb{Z}\} \qquad |S| \leqslant d. \tag{113}$$

Radial Basis Function

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = e^{-\gamma ||\boldsymbol{x}_m - \boldsymbol{x}_n||_2^2} \tag{114}$$

RBF is thus an exponentially decreasing distance measure between datapoints, which is indifferent about the direction of the displacement between the vectors.

Sigmoid

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \tanh(\gamma \boldsymbol{x}_n^T \boldsymbol{x}_m + r) \tag{115}$$

This kernel compresses the linear kernel to produce values between zero and one. This can be interpreted as selecting a radius for which we consider datapoints within the radius neighbours and those outside as non-neighbours, and using the neighbours for predicting.

The kernels allow SVM's and other kernel methods to handle data in a linear description-space, while the decision surfaces in the actual feature space is very non-linear.

One Class SVM

A specialized version of the SVM is the One-Class SVM, which attempt to train a classifier to recognize a specific class, without given any information about other possible classes. This is done by creating a hyperplane with a small margin, which separates the class from the remaining feature space. It uses a classification function to classify points similar to the previously used one

$$\text{class:} \qquad \text{sign}(\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n) - b). \tag{116}$$

This classification expression uses the kernels high value for close datapoints, to make the sum large for points close to the training data, while datapoints further away creates a small sum and a negative sign when subtracting b.

Since all samples in the training data is from the same class, there is no need for a target vector as used when describing the previous SVM. The classification-margin constraint (88) is therefore

replaced by

$$\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) \geqslant b - \xi_n \tag{117}$$

$$\xi_n \geqslant 0. \tag{118}$$

If all classes have large evaluations of the dot-product, then all ξ_n's will be zero and all datapoints will be within the positive region. For a datapoint outside of the region ξ_n will be positive.

To make the margin small, the same minimization of $\frac{1}{2}||\boldsymbol{w}||^2$ is used (as in (87)). The penalization with the slack-variables is introduced by solving

$$\arg\min_{\boldsymbol{w},b}\left(\frac{1}{2}||\boldsymbol{w}||^2 + \frac{1}{\nu N}\sum_{n=1}^{N}(\xi_n - b)\right). \tag{119}$$

The penalization is on the average difference between ξ_n and b, where ν controls the size of the penalty.

The resulting Lagrangian is

$$L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{a}, \boldsymbol{\mu}) = \underbrace{\frac{1}{2}||\boldsymbol{w}||^2}_{\text{Objective}} + \underbrace{\frac{1}{\nu N}\sum_{n=1}^{N}(\xi_n - b)}_{\substack{\text{Misclassification}\\ \text{penalty}}} - \underbrace{\sum_{n=1}^{N} a_n\Big(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}) - b + \xi_n\Big)}_{\substack{\text{KKT term for}\\ \text{Condition (117)}}} - \underbrace{\sum_{n=1}^{N}\mu_n\xi_n}_{\substack{\text{KKT term for}\\ \text{Condition (118)}}} . \tag{120}$$

The derivatives of the Lagrangian is

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{w}}L = \boldsymbol{w} - \sum_{n=1}^{N} a_n\boldsymbol{\phi}(\boldsymbol{x}_n) \quad \xrightarrow{\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{w}}L=0} \quad \boldsymbol{w} = \sum_{n=1}^{N} a_n\boldsymbol{\phi}(\boldsymbol{x}_n) \tag{121}$$

$$\frac{\mathrm{d}}{\mathrm{d}b}L = -\frac{1}{\nu} + \sum_{n=1}^{N} a_n \quad \xrightarrow{\frac{\mathrm{d}}{\mathrm{d}b}L=0} \quad \sum_{n=1}^{N} a_n = \frac{1}{\nu} \tag{122}$$

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\xi}}L = \frac{1}{\nu N}\mathbf{1} - \boldsymbol{a} - \boldsymbol{\mu} \quad \xrightarrow{\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\xi}}L=0} \quad a_n = \frac{1}{\nu N} - \mu_n. \tag{123}$$

By inserting (121), (122) and (123) into the Lagrangian, the following derivation is made

$$L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{a}, \boldsymbol{\mu}) = -\sum_{n=1}^{N} a_n \Big(\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) - b + \xi_n \Big) - \sum_{n=1}^{N} \mu_n \xi_n + \frac{1}{2}||\boldsymbol{w}||^2 + \frac{1}{\nu N} \sum_{n=1}^{N} (\xi_n - b) \tag{124}$$

(121)

$$= -\sum_{n=1}^{N} a_n \left(\sum_{m=1}^{N} a_m \boldsymbol{\phi}(\boldsymbol{x}_m) \right)^T \boldsymbol{\phi}(\boldsymbol{x}) + b \sum_{n=1}^{N} a_n - \sum_{n=1}^{N} a_n \xi_n - \sum_{n=1}^{N} \mu_n \xi_n + \frac{1}{2} \left\| \sum_{n=1}^{N} a_n \boldsymbol{\phi}(\boldsymbol{x}_n) \right\|^2 + \frac{1}{\nu N} \sum_{n=1}^{N} \xi_n - \frac{b}{\nu} \tag{125}$$

(122) (123)

$$= -\sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m \boldsymbol{\phi}(\boldsymbol{x}_m)^T \boldsymbol{\phi}(\boldsymbol{x}) + \frac{b}{\nu} - \sum_{n=1}^{N} \left(\frac{1}{\nu N} - \mu_n \right) \xi_n - \sum_{n=1}^{N} \mu_n \xi_n + \frac{1}{2} \left(\sum_{n=1}^{N} a_n \boldsymbol{\phi}(\boldsymbol{x}_n) \right)^T \left(\sum_{m=1}^{N} a_m \boldsymbol{\phi}(\boldsymbol{x}_m) \right) + \frac{1}{\nu N} \sum_{n=1}^{N} \xi_n - \frac{b}{\nu} \tag{126}$$

$$= -\sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m \boldsymbol{\phi}(\boldsymbol{x}_m)^T \boldsymbol{\phi}(\boldsymbol{x}) - \frac{1}{\nu N} \sum_{n=1}^{N} \xi_n + \sum_{n=1}^{N} \mu_n \xi_n - \sum_{n=1}^{N} \mu_n \xi_n + \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m \boldsymbol{\phi}(\boldsymbol{x}_n)^T \boldsymbol{\phi}(\boldsymbol{x}_m) + \frac{1}{\nu N} \sum_{n=1}^{N} \xi_n \tag{127}$$

$$L(\boldsymbol{a}) = -\frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m \boldsymbol{\phi}(\boldsymbol{x}_m)^T \boldsymbol{\phi}(\boldsymbol{x}) \tag{128}$$

$$k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \boldsymbol{\phi}(\boldsymbol{x}_m)^T \boldsymbol{\phi}(\boldsymbol{x}_n) \tag{129}$$

The dual representation problem is therefore

$$\arg\min_{\boldsymbol{w},b} \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m k(\boldsymbol{x}_n, \boldsymbol{x}_m) \tag{130}$$

$$\text{Subject to:} \quad 0 \leqslant a_n \leqslant \frac{1}{\nu N} \tag{131}$$

$$\sum_{n=1}^{N} a_n = 1 \tag{132}$$

By solving for $\boldsymbol{a}$ the SVM can separate a class from datapoints that does not belong. An example of a usage is to use the method to detect outliers. The datapoints that get moved outside of the hyperplane can be considered so different from the remaining datapoints, that they may be outliers with errors in their features or wrong classifications. Again many kernels can be used to project the hyperplane into different shapes in the original feature-space.

3.5.8 Principal Component Analysis

Principal Component Analysis (PCA) is a common analysis and dimension-reduction tool. It is motivated by considering the variance of samples in many dimensions. Assume a set of data points $\boldsymbol{x}_1..\boldsymbol{x}_N$. The mean and and covariance matrix are given by

$$\bar{\boldsymbol{x}} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n \qquad \boldsymbol{S} = \frac{1}{N}\sum_{n=1}^{N} (\boldsymbol{x}_n - \bar{\boldsymbol{x}})(\boldsymbol{x}_n - \bar{\boldsymbol{x}})^T. \tag{133}$$

Assume a unit-vector $\boldsymbol{u}$, which has some interesting direction in the hyper-dimensional space. The projection of a datapoint $\boldsymbol{x}_n$ onto this vector is

$$\boldsymbol{u}^T \boldsymbol{x}_n. \tag{134}$$

Similarly the projection of the mean-vector is

$$\boldsymbol{u}^T \bar{\boldsymbol{x}}. \tag{135}$$

Thus the variance in the direction of $\boldsymbol{u}$ is

$$\sigma_{\boldsymbol{u}} = \frac{1}{N}\sum_{n=1}^{N} (\boldsymbol{u}^T \boldsymbol{x}_n - \boldsymbol{u}^T \bar{\boldsymbol{x}})^2 = \boldsymbol{u}^T \boldsymbol{S} \boldsymbol{u}. \tag{136}$$

Say the direction in the feature space with the biggest variance is of interest. The following problem must therefore be solved (where the Lagrange multiplier comes from the constraint that $\boldsymbol{u}$ is a unit vector)

$$\max_{\boldsymbol{u}} \boldsymbol{u}^T \boldsymbol{S} \boldsymbol{u} + \lambda(1 - \boldsymbol{u}^T \boldsymbol{u}). \tag{137}$$

The derivative of the expression with respect to $\boldsymbol{u}$ is set to zero, to find the maxima

$$\boldsymbol{S}\boldsymbol{u} - \lambda \boldsymbol{u} = 0 \qquad \Rightarrow \qquad \boldsymbol{S}\boldsymbol{u} = \lambda \boldsymbol{u}. \tag{138}$$

This is the eigenvalue-problem, and $\boldsymbol{u}$ must thus be an eigenvector of $\boldsymbol{S}$ with eigenvalue λ.

Furthermore we have

$$\boldsymbol{S}\boldsymbol{u} = \lambda\boldsymbol{u} \qquad \Rightarrow \qquad \boldsymbol{u}^T\boldsymbol{S}\boldsymbol{u} = \boldsymbol{u}^T\boldsymbol{u}\lambda \qquad \Rightarrow \qquad \boldsymbol{u}^T\boldsymbol{S}\boldsymbol{u} = \lambda = \sigma_{\boldsymbol{u}} \tag{139}$$

The direction with maximum variance can therefore be found by solving the eigenvalue/eigenvector problem, and selecting the eigenvector with the highest eigenvalue. By selecting the M largest eigenvalues and their respective eigenvectors, where $M < N$, one can use M dimensions to describe a part of the variance given by (assuming that the eigenvectors are sorted with the largest values first)

$$r_M = \frac{\sum_{i=0}^{M} \lambda_i}{\sum_{i=0}^{N} \lambda_i}. \tag{140}$$

The chosen M vectors are called the M first principal components. If they describe a large part of the variance (r_M is close to one) it is possible to make fair approximations of the samples, by describing each sample as a linear combination of the principal components, plus some error vector that is hopefully relatively small, which reduces dimensionality.

3.5.9 Non-Negative Matrix Factorization

Non-negative Matrix Factorization (NMF) is another possible way to reduce dimensionality [38]. This method assumes that all features are positive (such as they are in bag-of-words representations). If $\boldsymbol{X}$ is a samples-features matrix with size $(N \times M)$, one can approximate two matrices $\boldsymbol{W}$ and $\boldsymbol{H}$, so that

$$\boldsymbol{W}\boldsymbol{H} \approx \boldsymbol{X}, \qquad X_{ij} \geqslant 0, \tag{141}$$

where the two auxiliary matrices have dimensions $(N \times k)$ and $(k \times M)$. $\boldsymbol{W}$ and $\boldsymbol{H}$ can be interpreted in the following manner: $\boldsymbol{W}$ maps the N samples into a k-dimensional space which describe some underlying structure and $\boldsymbol{H}$ maps the k underlying dimensions to the features of the original space. If $\boldsymbol{W}$ and $\boldsymbol{H}$ create a good approximation of $\boldsymbol{X}$, the samples can be mapped into the k-dimensional space, which is used as a new feature-space for training and prediction using other machine learning techniques.

The problem of finding $\boldsymbol{W}$ and $\boldsymbol{H}$ are typically solved numerically through an iterative process that can use various loss-functions and regularizations.

3.5.10 Word2Vec Features

A popular representation of documents, is to create word-embeddings as described by [42]. They use a so called *skip-gram* model where a word is used to predict the next c and previous c words in a sentence. The scheme is illustrated in figure 3.2.

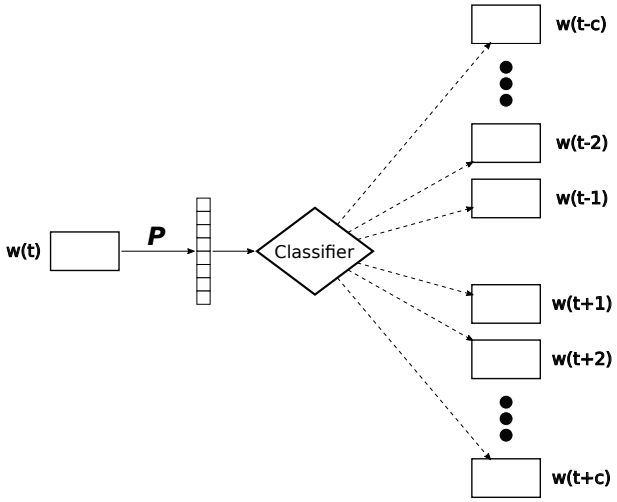


Figure 3.2: The Word2Vec scheme. Each word is projected to a vector of continuous values, which is used by a classifier to predict the c words before and after.

For a vocabulary V, the model uses a projection matrix $\boldsymbol{P}$ which maps each word to a d-length, continuous vector

$$\boldsymbol{P}\boldsymbol{e}_i = \boldsymbol{v}_i \qquad e_{ij} = \begin{cases} 1 & j = i \\ 0 & \text{otherwise} \end{cases} \qquad \boldsymbol{v}_i \in \mathbb{R}^d. \tag{142}$$

The $\boldsymbol{v}_i$-vectors are used as input to a logistic regression classifier, that predict the c previous and c future words. The classifiers and projection matrix are trained to maximize

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-c\leqslant j\leqslant c \\ j\neq 0}} \log p(w_{t+j} \mid w_t), \tag{143}$$

where T is the number of training examples, and $p(w_{t+j} \mid w_t)$ is the predicted probability of observing word w_{t+j} given word w_t.

When training, the projected representation of the words are fitted to the problem. This internal representation of the words can then be used as features for other uses, as it converts words to a continuous-vector format, which most algorithms can easier handle (than for example bag-of-words). Thus the projection matrix $\boldsymbol{P}$ is often the goal of the training process.

Although a relatively simple model, the training of the Word2Vec-system can be very costly, as the vocabulary and training data is typically very big. Thus much of the work done related to this word embedding has been on tuning the algorithms and finding estimation-solutions to speed up the process. One example is the replacing of the softmax in the logistic regression classifier with cheaper approximate alternatives (such as hierarchical softmax [43]).

3.5.11 Doc2Vec Features

In [37], the Word2Vec method is modified to create embeddings of paragraphs instead (called Doc2Vec). Figure 3.3 explains how this concept is able to create vector representations of variable-length paragraphs.

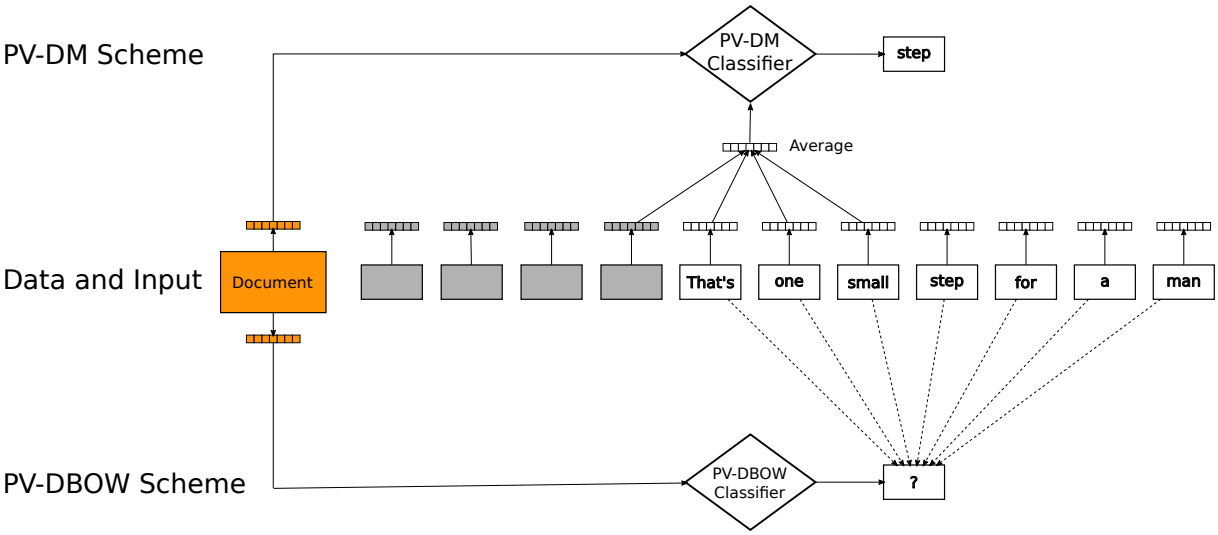


Figure 3.3: Doc2Vec scheme. The vertical middle of the illustration shows the data used for the scheme. A paragraph (which in this example is "That's one small step for a man") is split into words. Every word in the vocabulary has its own feature vector and so does every paragraph.

The first of the Doc2Vec schemes is the *Distributed Memory Model for Paragraph Vectors* (PV-DM). A window size n is chosen, which in this case is $n = 4$. n null-words are prepended the paragraph, so the window can end before any word in the paragraph. For each window the averaged vector of the words is computed and given to a classifier, which attempts to predict the word immediately following each window. The classifier and the weights of all words and paragraphs are iteratively updated with gradient descent to train the whole scheme. When new paragraphs are given, the classifier's and the word-vectors' weights are locked and gradient descent is used to compute the new paragraph's vector representation.

The second Doc2Vec scheme is the *Distributed Bag-Of-Words Paragraph Vector* (PV-DBOW). This scheme only uses the document vector as input, and trains a classifier for predicting the the words of the paragraph, which are randomly sampled. Again gradient descent is used to update both the classifier and the paragraph vectors. When new samples are given, the classifier's weights are locked and the paragraph's weights are determined.

In the schemes proposed by [37], both methods are used and the paragraph vectors are concatenated to create a final vector representation of paragraphs.

3.5.12 Ensemble Learning

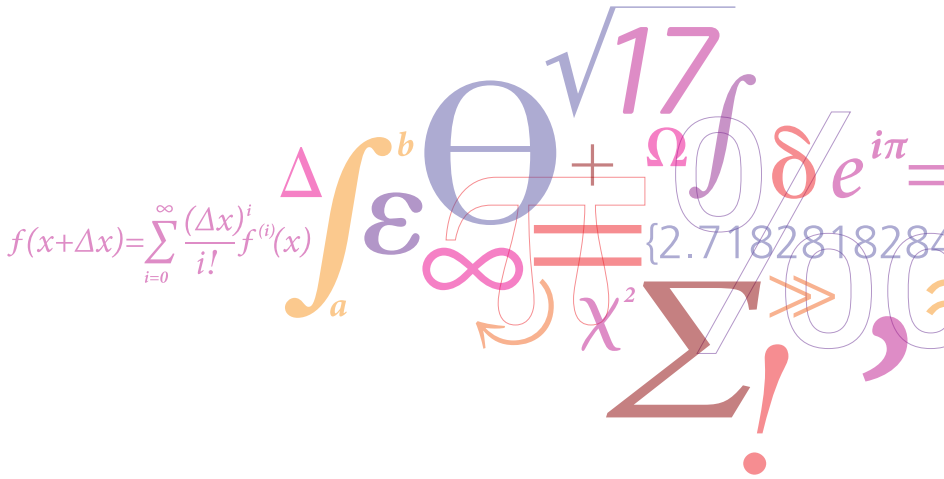
Ensemble learning ([32]) is the usage of multiple learning algorithms for machine learning tasks. The learners can typically output a prediction (for example the predicted class) or a predictive value, which corresponds to the confidence in the prediction (for example a value between zero and one, corresponding to the probability of being in a class). An ensemble can combine the learning algorithms in various way, some of which are:

Bootstrap Aggregation / Bagging Trains multiple models with randomly sampled subsets of the training data, and votes equally between the models for prediction.

Bucket of Models Multiple different models are training for different segments of the training data, and a model selection algorithm is used to select the appropriate algorithm for prediction. On a single segment of the data, the method can only perform as well as the best model, but on the average problem the method can perform better than the best model.

Stacking In stacking a st of models are trained independently and usually on all the training data. An additional learning algorithm is then trained to use the predictive values of the models, to come up with a final prediction.

In this project a simple stacking-ensemble is made, where the predictive values of several learners are combined in a logistic regression algorithm to make the final prediction. In this set-up, the predictive values of the model are used in a weighted average and then turned into a probability of the respective classes.



Chapter 4

Methods

Contents

4.1 Set-up

4.1.1 Experiment Set-Up

For training and testing algorithms, all data was initially split into a training set and a test set, using stratified sampling to ensure even distribution of the classes. The test set was kept aside and not used in any training or optimization process, and only used to evaluate the learning algorithms after optimization and training. Only one test set was used and no cross-validation made for determining the final performance, as this was task proved very time consuming. Cross validation was though used internally, when using the training data to optimize learners.

4.1.2 Python

The software of the project was implemented in Python. Python is an outstanding language for prototyping and has vast open-source libraries available, with state-of-the-art algorithms and data structures. This project combines many resources and handles large datasets. It was therefore essential to make use of any relevant, available tools. This made Python a great choice for this project.

The following lists some packages for Python, which were essential for the implementation of the project. Some of them are standard libraries that most Python-programmers know, while some of them are not as frequently used.

Table 4.1: Essential Python Libraries.

pickle[58]	Allows storage of Python-objects.
shelve[60]	Builds on pickle and creates a database for Python objects on the disk. This allows data to be stored in many Python-objects, each one accessible in a single query, without cluttering the computers directories with thousands of files. Shelves are not thread-safe, but a simple combination with the queue-library can solve this problem.
queue[59] + **threading**[61]	A classical combination used to make multi-threaded computations. As large amounts of data needed to be processed, multiple threads were needed to avoid programs that needed days to finish. The queues are thread-safe and can therefore be used to pass data to threads and safely retrieve results.
re[55] + **regex**[56]	Two different versions of Python's interface with regular expression, essential for fast processing of strings. One particularly useful feature from the newer regex-library, is its ability to perform fuzzy-searches with regular expression.
numpy[52]	The fundamental library for efficiently working with matrices. Numpy is partly implemented in lower-level programming languages, which makes the matrix operations extremely fast.
scipy[52]	*"... a Python-based ecosystem of open-source software for mathematics, science, and engineering."*[68]. SciPy contains implementations of sparse-matrices in several formats. These matrices combined with NumPy makes it possible to work with extremely large, sparse matrices.
sklearn[66]	Contains many well-known machine-learning and data processing algorithms. This allows for fast testing of various methods on ones data.
networkx[47]	Implements data structures and common algorithms for graphs. The digraph implementation is used for representing arguments in this project.
nltk[50]	*Natural Language ToolKit* implements natural language methods as the name indicates.
bisect[57]	Was used for performance reasons. The package implements fast searches, deletions and insertion in sorted lists.
gensim[63]	Implementations of Word2Vec and Doc2Vec.

4.1.3 Other Resources

Stanford Parser

A useful resource for a natural language processing project, is a parser. The Stanford Parser[34], by the Stanford Natural Language Processing Group[70], is a well-known parser. The parser has many useful features, one of them being the ability to make probabilistic parsings, with multiple possible parsings for a single sentence, together with an approximate probability for each parsing.

Google News Word2Vec Dataset

With the gensim-library one can train word embeddings and document embeddings. Google has published a pre-trained word-embedding based on a 100 billion words dataset (Google News dataset). This word-embedding contains a 300-dimensional vector representation of 3 million words and phrases. Googles description of word2vec is found in [26], while the pre-trained word-embedding can be downloaded in [74]. The file takes up 2GB when compressed and 3.4GB when uncompressed. Using the data can initially be difficult, as it requires a large RAM to load the model. In this project the model was opened on a server and the vectors for the vocabulary of the data were extracted and stored in a smaller file, which was much easier to handle.

Rhetorical Relations

A fair hypothesis is that certain words are more likely to be used in argumentation, and can be used to detect arguments. Using various representations of texts, machine learning systems should be able to learn and find such words. Some multi-word phrases may also be relevant for argumentation (for example "*as a consequence*" and "*despite that*"), but some text-representations (such as bag-of-words) will throw out information about the sequence of words and neighbourhood of words. To counter this problem, one can find the relevant rhetorical relations that are common in argumentation, and create extra features which counts these relations. Knott[35] has developed such a list of relations, which have previously been used in argument detection and will also be used in this project.

4.2 Preprocessing

4.2.1 Data

The main database used in this project is the Araucaria database, as it has previously been used for similar investigations and used by articles leading up to this project. The database consists of a number of text-files and a number of JSON-files, named in the following manner:

nodeset7.json	nodeset7.txt
nodeset8.json	nodeset8.txt
nodeset9.json	nodeset9.txt
nodeset10.json	nodeset10.txt
nodeset11.json	nodeset11.txt
...	...

The text files contains text associated with the database. The JSON-files contains argument maps and follows the AIF format. Listing 4.1 shows a part of the Araucaria database.

```
{"nodes":[
    {"nodeID":"255","text":"Alexander Downer has derided more",
        "type":"I","timestamp":"2012-04-12 18:18:22"},
    {"nodeID":"256","text":"If the conjecture that members more",
        "type":"I","timestamp":"2012-04-12 18:18:22"},
    more
    ],
"edges":[
    {"edgeID":"251","fromID":"256","toID":"258","formEdgeID":null},
    {"edgeID":"252","fromID":"257","toID":"258","formEdgeID":null},
    more
    ]}
```

Listing 4.1: Snippet from the Araucaria database. "more" indicates a continuation of text/data.

In order to use the database, some essential preprocessing was needed. The following describe the preprocessing performed before each of the investigated tasks of the project.

4.2.2 Preprocessing for Argument Element Detection

Palau and Moens[53] uses a system to detect elements of a string as being argumentative or not. For reproducing these results, a dataset with strings labelled as argumentative and strings labelled as non-argumentative was needed. The JSON-files only contained argumentative elements, and non-argumentative strings were therefore sought. The JSON-files and text-files came in pairs, and so initially it was believed that the text-files contained the source for the strings used in the arguments of the similarly named JSON-file. This was partly true. As an example `nodeset8.json` contained eight strings, where two of them was found in `nodeset8.txt`. This inspired a preliminary investigation, where all sentences in all elements of the argument maps found in the JSON-files were mapped to the sentences from the text-files, as explained in the following section. This would reveal whether some non-argumentative strings were included in the data, and whether the origins of all arguments were given.

From the Araucaria database, a set of sentences that are argumentative were extracted. A bit of cleaning was performed on the data. As some of the text-files in the Araucaria database had

Spanish text a removal of non-English sentences was made, similar to the one of 4.2.3 was used. The result was 3,773 sentences from Araucaria, all labelled as argumentative.

4.2.3 Database Cleaning and Initial Analysis

A concatenated string of 422,181 characters was created from the database's text-files, which was segmented using NLTK's sentence tokenizer into 3,057 sentences. These sentences were potential candidates for being argumentative, as some of them will appear in the argument maps of the database. Some sentences contained only a URL-link or were a headline with just a few words, and no possible argumentative structure. Using NLTK's word tokenizer, the sentences were inspected and all sentences with 2 or less non-symbolic tokens (actual words), were removed. 2,994 "long" sentences remained. Furthermore some sentences were in Spanish, which would further increase the difficulty of the task. To handle this problem, a collection of all English words was downloaded from SIL International [69] (there are Python libraries created for detecting English words, but, at the time of this writing, these had problems on running with Python 3.5 on a 64-bit system). Any sentence containing 15% or fewer English tokens (of the non-symbolic tokens) were removed leaving 2,863 candidate English sentences of reasonable length.

All argument-segments were extracted from the argument maps, after which all sentences containing argument elements as substrings were found. The number of comparisons between argumentative elements and sentences was

$$\text{comparisons} = \frac{n \times m}{2}. \tag{1}$$

With n being the number of argumentative elements (3,773) and m being the number of sentences (2,868). This creates about 10,000,000 comparisons, which easily becomes time-consuming. The exact distance were not needed. Only the detection of very similar sentences was needed and the methods described in *String Searching and Distance Measure* were therefore used.

Of the 3,773 argument elements found in the database, 3,771 of them were mapped to 2,798 of the 2,863 sentences. This leaves very few of the database's argument elements belonging to none of the text-files and few sentences being non-argumentative. This concluded that there are basically no non-argumentative texts to find in the Araucaria-database.

String Searching and Distance Measure

Using regular expression allows for very fast string searches. Furthermore, one can compile a regular expression object based on a pattern and apply that object for searching for the same thing many times. This avoids the recreation of a regular expression systems at each search and improves speed. With the `regex`-package for Python one can make string searches while allowing errors. The three errors that the library can handle are:

- Insertions
- Deletions
- Substitutions

The user can specify a roof on each of the three types, or one roof on the sum of the three

error-types. The searches become significantly slower as more errors are allowed, and if too many errors are accepted a string may be selected in the search with very little similarity to the search-string. The fuzzy-search can be used as a way of detecting possible candidates for duplicates or superstrings, after which a (computationally heavier) distance measure can be used to find the actual similarity. In all searches, the strings were converted to all lower-case, as capitalization is irrelevant for analysing the semantics of the sentences.

As a distance measure between strings, the Levenshtein distance[39], also called edit-distance, was used. This distance measures the number of insertions, deletions and substitutions are needed, for two strings to be identical (and therefore relates to the fuzzy-search). The fuzzy-search with the `regex`-library does not compute the Levenshtein distance, but rather approves or disproves matches based on the specified Levenshtein distance limit. The distance was therefore compute with the `distance`-library instead, when needed. Another important difference between the fuzzy-search and the Levenstein distance is, that the fuzzy search goes through substrings in the search and disregards the rest of the superstring. The Levenshtein distance and implementation uses the entirety of two strings, making it an ill choice for searching for a substring in a larger string. Figure 4.1 illustrates the difference. Using dynamic programming, the Levenshtein distance can be computed in time $O(m \times n)$ and space $O(n + m)$, where m and n are the lengths of the two strings to be compared.

String 1:	*a*
String 2:	*abbb*
Levenshtein Distance:	*3*
Fuzzy Search:	*Exact match (0 errors)*

Figure 4.1: Difference between Levenshtein distance and fuzzy searching.

By finding candidates and using the Levenshtein distance as a ranking, the most similar sentences were mapped together.

4.2.4 Obtaining Unlabelled Data

As no non-argumentative elements were given, an attempt was made to find some of the origins of the Araucaria database. By having Google search for long pieces of text from the database, a few candidates of the original sources were found. The following sites were used:

BBC News
`http://news.bbc.co.uk/2/hi/talking_point/2624539.stm`
`http://news.bbc.co.uk/2/hi/talking_point/2491327.stm`
`http://news.bbc.co.uk/2/hi/talking_point/3701582.stm`

UK Parliament
`http://www.publications.parliament.uk/pa/cm200304/cmhansrd/vo040916/debtext/40916-02.htm`
`http://www.publications.parliament.uk/pa/cm200304/cmhansrd/vo040916/debtext/40916-17.htm`

CNN
`http://edition.cnn.com/2003/WORLD/meast/03/17/sprj.irq.bush.transcript/`
`http://edition.cnn.com/2004/ALLPOLITICS/09/30/debate.transcript.5/index.html`

New York Times
`http://www.nytimes.com/2003/03/18/politics/18BTEX.html?pagewanted=all`

Cornell University Law School
`https://www.law.cornell.edu/supct/html/1-10868.ZA2.html`

The Age
`http://fddp.theage.com.au/articles/2003/05/26/1053801334519.html?from=storyrhs`

From these sites 1,234 unlabelled sentences were extracted.

Unrelated Data

A few more websites was used to extract similar data, which is unrelated to the subjects of the texts above. The data was extracted from sites that were already used in the above list, but on other subjects. The three sites are:

BBC News "*Shuttle disaster: You asked the experts*"
`http://news.bbc.co.uk/2/hi/talking_point/forum/2718811.stm`

UK Parliament "*Indebted Prepayment Customers*"
`http://www.publications.parliament.uk/pa/cm201516/cmhansrd/cm160324/debtext/160324-0001.htm#16032433000016`

CNN "*Why are the French on strike ... again?*"
`http://edition.cnn.com/2016/06/02/europe/france-strikes-labor-reform-bill/index.html`

4.2.5 Palau and Moens Features

Palau and Moens[53] uses a set of features for detecting argumentative elements in text. Throughout this report these will be referred to as the *argument-features*, as they are the only features that have been specifically created to detect arguments. Their features are used as a starting point for the tasks in this project. As will be noted later, the feature-space is very high-dimensional when using these features. Table 4.2 describes some upper bounds on the dimensionalities of the features-spaces. Some upper bounds have two different variants, based on the variables used to estimate complexity.

Variables:

L**:** Concatenated word-length of texts.

V**:** Size of vocabulary.

T**:** Highest number of words in a text.

P**:** Number of different symbols.

N**:** Number of texts.

T_P**:** Highest number of symbols in a text.

Feature	**Representation**	**Complexity**
Word related		
Unigrams	Bag-Of-Words	$O(V)$ **or** $O(L)$
Bigrams	Bag-Of-Words	$O(V^2)$ **or** $O(L)$
Trigrams	Bag-Of-Words	$O(V^3)$ **or** $O(L)$
Co-occurrence of two words in paragraph	Bag-Of-Words	$O(V^V)$ **or** $O(N \times T^T)$
Text statistics		
Sentence length	Integer	$O(1)$
Average word length	Float	$O(1)$
Number of punctuation marks	Integer	$O(1)$
POS and parsing features		
Number of adverbs	Integer	$O(1)$
Number of verbs	Integer	$O(1)$
Number of modal auxiliary words	Integer	$O(1)$
Depth of parse tree	Integer	$O(1)$
Number of subclauses in sentence	Integer	$O(1)$
Punctuation sequence	One-Hot	$O(T_P^P)$
Keys words and phrases developed by [35]	Bag-Of-Words	$O(1)$
Largest contributors		$O(N \times T^T + V^3 + L + T_P^P)$

Table 4.2: Paragraph-features inspired by [53], shown with upper bounds on feature-space dimensionality.

Some of the features used by [53] are very high-dimensional, and scale disproportionally with the size of the vocabulary and data-set. The table below shows the actual dimensionalities of the dataset used in this project.

Feature	**Representation**	**Size**
Word related		
Unigrams	Bag-Of-Words	≈ 7,900
Bigrams	Bag-Of-Words	≈ 28,000
Trigrams	Bag-Of-Words	≈ 37,000
Co-occurrence of two words in paragraph	Bag-Of-Words	≈ 260,000
Text statistics		
Sentence length	Integer	1
Average word length	Float	1
Number of punctuation marks	Integer	1
POS and parsing features		
Number of adverbs	Integer	1
Number of verbs	Integer	1
Number of modal auxiliary words	Integer	1
Depth of parse tree	Integer	1
Number of subclauses in sentence	Integer	1
Punctuation sequence	One-Hot	100
Keys words and phrases developed by [35]	Bag-Of-Words	≈ 110
Total		≈ 340,000

Table 4.3: Paragraph-features inspired by [53] used for detecting argument elements. **Size** is the number of dimensions in the respective feature-space.

The features presented above creates a very large features space with approximately 340,000 dimensions for detecting argument elements. When detecting links between argumentative elements (textual entailment), both the *text* and the *hypothesis* must be represented, together with possible similarity measures. This creates twice as many features (670,000) for this task. The features are also very sparse. 6.3% of the sample-vs-feature matrix was non-zero for the training data, while 4.3% of the sample-vs-feature matrix for the test data was non-zero (attributed to the difference in vocabulary between the two datasets).

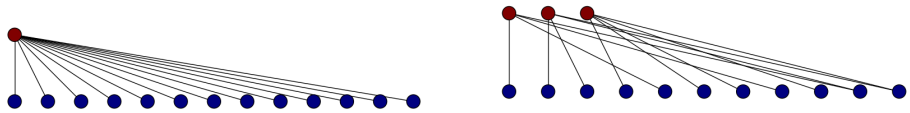
4.2.6 Preprocessing for Argument Structure Detection

Detecting argument structures is done by identifying links between texts and hypotheses/claims.
The JSON-files of the Araucaria database contain argument maps. From this data, a large bipartite graph was created, with the bisection consisting of claims vs. supporting texts/premises. As some argument maps may have subordinatively compound arguments, a node from the Araucaria database may appear both on the claim-side and the premise-side, although no duplicates were allowed on the same side. Figure 4.2 shows two of the largest connected components from the bipartite graph. Most of the components were much smaller and many consists of a claim with only one or two premises. A few statistics are shown in table 4.4. A notable feature is that more than half of the claims also exist as premises, which comes from the sequential argument-structures in the Araucaria dataset.

Table 4.4: Information regarding the bipartite premises-claims graph from the Araucaria database.

	Graph Info
Nodes	4543
Edges	3289
Connected Components	1261
Premises	3185
Claims	1358
Shared Nodes*	770

* Samples that exists both as premises and claims.



(a) A large connected component. Claims are red and premises are blue.

(b) A large connected component with multiple claims. Claims are red and premises are blue.

Figure 4.2: Two of the largest connected components created from the Araucaria dataset.

The bipartite graph describes the true linking between claims and premises. To learn this structure, we consider the detection of links between nodes to be a classification problem, where each link is independent of the neighbourhood of the nodes. Note that for using this system, we assume that a different system has made a perfect segmentation of a text into claims and premises. Description of the systems used for detecting these links, are described in 4.4.

4.2.7 Outlier Creation

In some of the following sections it was useful to have some synthetically generated datapoints for testing the methods ability to separate the real argument data from noise/random data. A common way to create outliers, is to add noise to sampled data. Three methods of generating outliers were considered, as illustrated in the following. Ideally the outliers should be similar to the features that could be found from real texts, although they should not be argumentative.

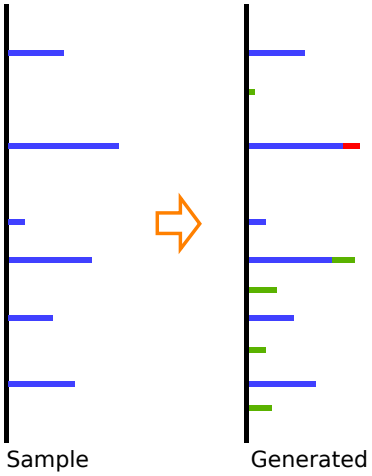


Figure 4.3: **Randomly distributed noise**. A sample from the training data is used and noise is added. The noise can be positive (green) or negative (red), but since the original data is all positive this restriction is maintained on the new data. The sparsity of the data is changed as more features will be in use after adding noise. As many of the features are counted values, the noisy data may not truly make sense, as the noisy data could represent a floating point number of occurrences of a word of word-combination.

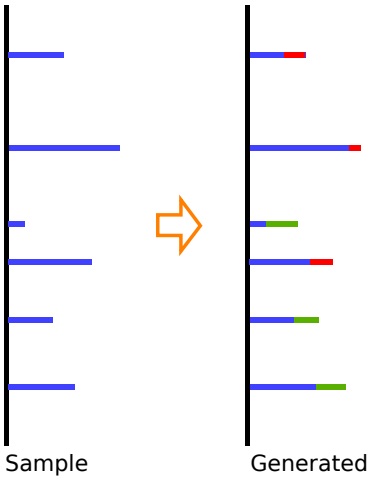


Figure 4.4: **Specifically added noise.** A sample is again used from the training data, but noise is only added to the features that are already in use. This maintains sparsity fo the data as no features will be solely composed by noise. The original structure of the sample is maintained though and the new datapoint may realistically not be an outlier, as the same features in a sparse situation may represent an almost-identical text-sample. Again the noisy data may not truly make sense, as the noisy data could represent a floating point number of occurrences of a word of word-combination.

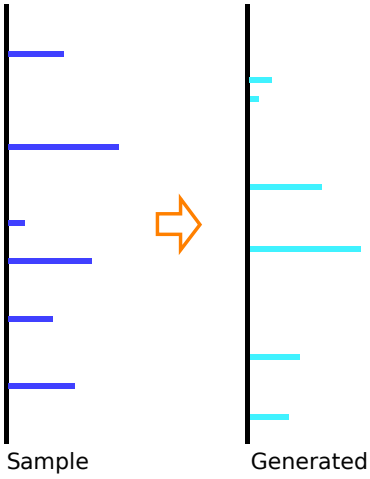


Figure 4.5: **Generated sparse points.** Random, sparse points are generated by using statistics from the training data. the probability for each feature being non-zero are estimated from their frequency. From these probabilities new datapoints can be sampled with sparse features. The mean-values for each feature is used with a Poisson or a Gaussian distribution to sample the actual values of the features. The argument features are all positive and discrete and so use the Poisson distribution, while Doc2Vec and Word2Vec create continuous data and would use a Gaussian.

All of the three methods for creating outliers were used, and the features stacked together to outlier datasets including all three types. As the outlier datasets were only used to compare methods for how well they could separate the outliers, it does not matter whether some of the outliers are potentially undetectable.

4.3 Argumentative Element Detection

[53] describes the task at hand as:

> "*The detection of all the arguments presented in a free text is similar to the binary classification of all the propositions of the text as argumentative or non-argumentative.*"

This project attempts to reproduce the system [53] made. The idea is that a text contain a number of propositions, which can be segmented. Furthermore it assumes that these propositions are either argumentative or not. The segmentation is not described thoroughly, but could be performed with a sentence-tokenizer, such as the ones available from NLTK, ([50]) or from a text's parsing. This method assumes that propositions are contained within sentences. This is not always the case, as some sentences uses information from previous sentences. For example "It is a commonly used algorithm" could be an argument for testing a specific algorithm on a problem, although information is needed from the previous sentences about what algorithm is in question. Dependency parsers, such as the one available from the Stanford group ([70]), can be used to detect such relations. It is thus possible that propositions could be contained within sentences if some additional information is appended. In this project, the assumption of propositions being contained in sentences is used as an initial approach to the problem. The segmented sentences can now be classified as argumentative or not, depending on whether they contain an argumentative propositions or not. A classifier is thus supposed to classify argumentation by detecting the features of argumentative propositions within sentences, while disregarding any irrelevant information that may also be in the sentence.

From section 4.2.4 a set of unlabelled sentences were gathered. Using the methods of 4.2.7, a set of synthetically generated samples were created as outliers. Using these datasets, some experiments were run to test how well the unlabelled data could be used to supplement the original data.

4.3.1 Preliminary Analysis of System

Propositions as Being Argumentative and Non-Argumentative

The definition of a propositions, as given in 3.1, is a statement which is either true or false. From a completely logical perspective, it is always possible to create a logical rune that includes a proposition. For example, given some propositions a (which may be a compound of many propositions), one could claim a new proposition (which may be true or false)

$$a \wedge b \rightarrow c, \tag{2}$$

using arbitrary propositions b and c.

In human argumentation the rules are a little stricter, as picking a completely random propositions b and c would typically lead to nonsense and inconsistency. Although, in a similar fashion one could typically come up with some propositions b and c that could be used together with an initial proposition a. Since a proposition contains some kind of information, it is difficult to imagine propositions that can not be used in *any* argument in *any* context. This questions the usefulness of a system detecting argumentative and non-argumentative propositions.

Exceptions would be tautologies and inconsistencies, which are propositions whose truth-values are the same irrelevantly of other information. These statements do not contain any information and can thus be irrelevant in argumentation.

Another exception in natural language are sentences that do not contain any propositions. An example of such a sentence could be a question. The question is a query for information from another participant in a context, and may thus be without any information.

Overall it seems that the use of a detection-system for detecting argumentative parts would sort out non-informational sentences, tautologies and inconsistencies, to keep informational propositions for argumentation.

Relation to Topic Modelling

Topic modelling is the use of statistical and machine learning methods for discovering abstract "topics" occurring in documents. A topic will contain documents that are more similar to each other in phrasing and vocabulary, than they are to documents from other topics. Ideally argument mining should be able to work on any topic, as the semantic structures of arguments do not need to reside within specific vocabularies. There are though a few similarities to topic modelling, some of which may be unwanted. When representing documents as bag-of-words, bigrams, trigrams and co-occurrences, the features become very dependent on the vocabulary of the respective documents. When detecting arguments in data, the system may actually detect whether other documents are from the same topic, as only arguments within the training-data's topic have ever been encountered and arguments typically contain statements in the same topic. When considering the textual entailment task (Link Detection in this report 4.4), the topic modelling aspect may be wanted, as knowing whether the premise and claim are from the same topic would be a very relevant information. Although it could also make the system less generalizable to new texts, as large corpora would repeatedly need to be tagged when a new topic is encountered.

4.3.2 Separation of Outliers

The synthetically generated outlier-dataset is assumed to be non-argumentative. The argument-detection system should therefore ideally be able to separate these samples from the argumentative samples found in the Araucaria database. For testing this hypothesis, a radial-basis function SVM were tested to separate the two classes, creating the confusion matrix in figure 4.6.

Test Outlier Separation

Total 427	Real Positives 287	Real Negatives 140	Accuracy: 88.52%
Predicted Positives 286	262 61.36%	24 5.62%	Precision: 91.61%
Predicted Negatives 141	25 5.85%	116 27.17%	NPV: 82.27%
Error rate: 11.48%	Recall: 91.29%	Specificity: 82.86%	F1-score: 91.45%

Figure 4.6: Confusion matrix created from a trained SVM separating original samples from synthetically generated outliers. The definitions of the matrix can be seen in appendix A.1. Positives denote the original distributions, while negatives are outliers.

The fairly high F1-score (and general performance) of the classifier indicates that the outliers have been generated in such a fashion, that they can be detected as foreign samples by classifiers. This could suggest that the classifier is capable of detecting arguments, but the synthetically generated outliers could also have been generated too different from the original data, to represent features of actual texts.

4.3.3 Separation of Labelled and Unlabelled Data

For testing whether the labelled data and the unlabelled data comes from similar distributions, a radial-basis function SVM were optimized for separating the samples of the two datasets.

Test Separation

Total 411	Real Positives 287	Real Negatives 124	Accuracy: 85.40%
Predicted Positives 297	262 63.75%	35 8.52%	Precision: 88.22%
Predicted Negatives 114	25 6.08%	89 21.65%	NPV: 78.07%
Error rate: 14.60%	Recall: 91.29%	Specificity: 71.77%	F1-score: 89.73%

Figure 4.7: Confusion matrix created from a trained SVM separating labelled samples from unlabelled samples. The definitions of the matrix can be seen in appendix A.1. Positives denote the labelled data and the negatives are the unlabelled.

Figure 4.7 shows how the classifier was quite capable of separating many of the labelled samples from the unlabelled. This can be caused by the unlabelled data having many non-argumentative elements, which is the ideal case as it proves the system useful for argument detection. Although it could also be caused by the new data being too different from the original data, with respect to topic and vocabulary.

Nonetheless, the elements that were inseparable from the labelled data could very well be arguments, and will later be used as extra labelled samples (4.3.5).

4.3.4 Separation of Labelled and Unrelated Data

In 4.2.4, a set of unrelated sites were selected. These are from similar sites and should have a similar distribution of arguments, although they do not use the same

Test Separation Unrelated

Total 411	Real Positives 287	Real Negatives 124	Accuracy: 82.73%
Predicted Positives 316	266 65%	50 12%	Precision: 84.18%
Predicted Negatives 95	21 5%	74 18%	NPV: 77.89%
Error rate: 17.27%	Recall: 92.68%	Specificity: 59.68%	F1-score: 88.23%

Figure 4.8: Confusion matrix created from a trained SVM separating original labelled samples from texts extracted from a few sites with unrelated topics. The definitions of the matrix can be seen in appendix A.1. Positives denote the labelled data and the negatives are the unrelated.

The separation of the unrelated data is not considerably better than the separation of unlabelled data. This indicates that the topic modelling aspect is not the main factor in the argument detection system.

4.3.5 Adding Unlabelled Data as Training Data

An additional attempt to separate the data from outliers were made, by adding the inseparable, unlabelled samples to the training data of an SVM classifier (as in 4.7). The number of labelled sentences was therefore increased from 2,868 to 2,982.

Test Outlier Separation w Unlabelled

Total 439	Real Positives 299	Real Negatives 140	Accuracy: 87.93%
Predicted Positives 286	266 61%	20 5%	Precision: 93.01%
Predicted Negatives 153	33 8%	120 27%	NPV: 78.43%
Error rate: 12.07%	Recall: 88.96%	Specificity: 85.71%	F1-score: 90.94%

Figure 4.9: Confusion matrix created from a trained SVM separating original samples (labelled and unlabelled) from synthetically generated outliers. The definitions of the matrix can be seen in appendix A.1. Positives denote the labelled data and the negatives are the unlabelled.

There is no significant increase in performance from using the additional training data, although the size of the added training set was relatively small. Using more a larger unlabelled dataset could potentially increase performance.

4.3.6 Argument Detection

The system used for argument detection seems capable of detecting synthetically generated outliers with similar feature-statistics. The system should therefore be capable of separating completely random text-segments from argumentative text-segments, although using actual text-segments from real texts (random text-segments would be meaningless), could prove more difficult to separate. The system has similar performance when separating unlabelled data (similar topic) from the argumentative elements, as when separating unrelated data (different topic) from the argumentative elements. This indicates that the system does not rely on topic modelling when detecting arguments. That is, it does not rely on detecting the topics of its training data, when classifying argumentative elements. Finally the inclusion of unlabelled data to the classifiers training, did not increase performance for detecting synthetic outliers. The number of unlabelled samples added was though very small relative to the number original training samples.

Overall the system is difficult to test with the Araucaria database and, due to the reasons described in 4.3.1, the system may not be the ideal approach on argumentation mining.

4.4 Link Detection

System used by Palau and Moens

The system created by [53] uses a context free grammar for detecting the structure of arguments. The grammar analyses the context and sequence of the text-segments in a generative manner to find probable argument-structures. The grammar knows various argumentative markers, such as (quoting):

- Conclusive rhetorical marker: *therefore, thus, ...*
- Support rhetorical marker: *moreover, furthermore, also, ...*
- Verb related to a premise: *note, recall, state, ...*
- Verb related to a conclusion: *reject, dismiss, declare, ...*

amongst others, to identify the relations between segments.

Since the original texts are unavailable, the grammar system was not reproducible, as the original text-sequence is necessary for the grammar to detect the structures. For the same reasons the problem is defined slightly differently in the project. The argument-structure detection is here defined as detecting links between premises and claims, where the clams and premises have been pre-separated to the bipartite graph described in 4.2.6. This task is commonly known as textual entailment.

The data used consists of all edges in the Araucaria database, as well a twice as many samples from non-adjecent node-pairs (non-arguments). The number of possible edges in the graph is dependent on the number of premises N_P and number of claims C_P by

$$C_P \times N_P. \tag{3}$$

As the number of actual arguments may not increase by this quadratic term, the number of non-adjacent nodes can increase much faster than the number of adjacent nodes. For example adding unrelated text to the database would quickly skew the ratio of argument- and non-argument-relationships between nodes. This creates a highly unbalanced dataset, where the ratio between argument-edges and non-argument edges can basically become infinitesimal. It is thus possible that the following methods may only work for relatively small text-passages with many arguments. This problem is not handled in this project, which focus on the detection task in the created dataset with approximately two non-argument relationship for every argument edge.

System of Project

In the following we attempt to detect the argumentation-structures of the bipartite graph from 4.2.6, by creating features from the nodes' texts and using various machine learning classifiers for detecting links between premises and claims. In order to best utilize the algorithms and data a few approaches were investigated:

4.4.1 Outlier Detection:
Investigates whether some datapoints may be misclassifications and should be disregarded.

4.4.2 Normalization:
Attempts to improve performance by normalizing either the data-matrix's rows or columns.

4.4.3 Feature Reduction:
Tests how well classification is done when the feature-space dimensionality is reduced.

4.4.4 Data Choice:
Compares the different data-sources' use for link detection.

4.4.5 Error Comparison + 4.4.6 Ensemble:
Compares previous results and combines these in a simple ensemble.

These approaches were all tested on three classifiers:

1. Logistic Regression
2. Linear SVM
3. Radial Basis Function SVM

Where a one-class radial basis function SVM was used for outlier detection.

4.4.1 Outlier Detection

Meaning of ν in One-Class SVM

For outlier detection, a radial-basis function, one-class SVM was used and trained on the argument-feature dataset. As explained by [64], this learning algorithm can be used to create a hyper-dimensional surface, which separates a region with most of the datapoints, from the rest of the feature-space. The `OneClassSVM` from `Sklearn` was trained using the data created from the links of the argument maps in the database. According to `Sklearn`'s website, the SVM was implemented so that the hyperparameter ν is: "*An upper bound on the fraction of training errors and a lower bound of the fraction of support vectors*"[67]. Thus different outlier-detectors could be created with different ν's, which would be more or less prone to classify samples as outliers. ν could then be selected for a reasonable ratio of outliers.

To test the effect of ν, an initial experiment was made where the SVM was training on the training data of the project, and ν varied while the number of training-set outliers was observed. Figure 4.10 shows the outcome of this experiment. Unfortunately the wanted effect of ν was not found, as far more outliers were assigned in the training data that ν indicated. Thus ν did not work as an upper bound on training outliers. This makes the performance of the outlier detector more difficult to define, as one has to weight the importance of outliers and training data samples. The SVM tended to classify way too many samples as outliers, which would reduce the training data to a impractically small fraction.

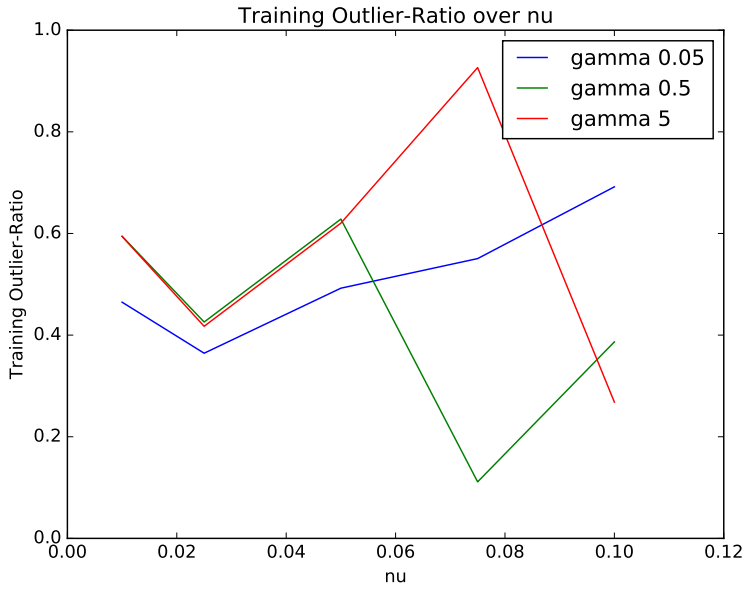


Figure 4.10: The plot shows the relative number of outliers detection in the training data, when using various values for ν and γ, with a radial-basis-function SVM.

To test whether this was a general property of the implementation, a simulated experiment was created on a simple distribution (Gaussian) with some generated noisy outliers (uniformly generated). The outcome of this example is illustrated in appendix A.4. This experiment showed that ν was capable of acting as the wanted ratio of outliers for simpler data. It was therefore concluded that the reason for the inconsistency between the outlier-ratio and ν is caused by properties of the data used in this project (the argument features). It is likely that the extreme sparsity, high-dimensionality relative to samples and discrete values of all features makes the data difficult for the radial basis function SVM to handle. From the experiments in appendix A.4 it is made clear, that distributions that are close to residing within a subspace of the feature-space (such as the stretched distribution of figure A.3c) were difficult to handle. The data of this project reside in a subspace that is much lower dimensional that the feature-space, because there are so many more features than samples. This could thus be the cause of the problem.

Alternative Outlier Detection

Instead of using the direct classifications from the SVM as outlier-detection and using ν for controlling the ratio of outliers, a different approach was used. Three different ratios of outliers were chosen: 1%, 5% and 10%, so that

$$r = \begin{pmatrix} 0.01 \\ 0.05 \\ 0.10 \end{pmatrix} \tag{4}$$

For each outlier-ratio r_n, a one-class, radial-basis function SVM was trained on the argument-feature dataset, with $\nu = r_n$. γ was determined by a simple search, with the goal of separating

as many of the synthetically generated outliers as possible. The SVM computes the product

$$y(\boldsymbol{x}_n) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n) \tag{5}$$

for all $\boldsymbol{x}_n$'s. The value of $y(\boldsymbol{x}_n)$ is negative if the classifier marks it as an outlier (which occurs with too many samples). The value of $y(\boldsymbol{x}_n)$ can be seen as a confidence in sample n being from the true distribution. A high value means the classifier is very confident that the sample is not an outlier. This ranking was normalized into a measure, which in the following will be denoted as the *distrust* in a sample

$$\text{distrust}(\boldsymbol{x}_m) = \frac{-y(\boldsymbol{x}_m) + \ell}{\ell - k} \tag{6}$$

where

$$k = \min_{\substack{\boldsymbol{x} \in \text{Training} \\ \text{Data}}} y(\boldsymbol{x}) \qquad \ell = \max_{\substack{\boldsymbol{x} \in \text{Training} \\ \text{Data}}} y(\boldsymbol{x}). \tag{7}$$

The distrust in a sample is a positive value, which for the training set will be $\text{distrust}(\boldsymbol{x}_m) \in [0, 1]$ and for the test data will be $\text{distrust}(\boldsymbol{x}_m) \in [0, \infty]$, although values much higher than one are unlikely.

From r_n a suitable threshold on the distrust-measure was found for the training data. This threshold was used to remove outliers from the test data. Figure 4.11 shows how the ratios of outliers between the training data and the test data compare, based on threshold. Ideally the graphs should follow the diagonal of the graph, so that a similar proportion of outliers is chosen in the two data sets, which is almost the case. This indicates that the SVM's predictions are distributed similarly across the training data and test data, and removing outliers does not have a significantly higher impact on one data set than the other.

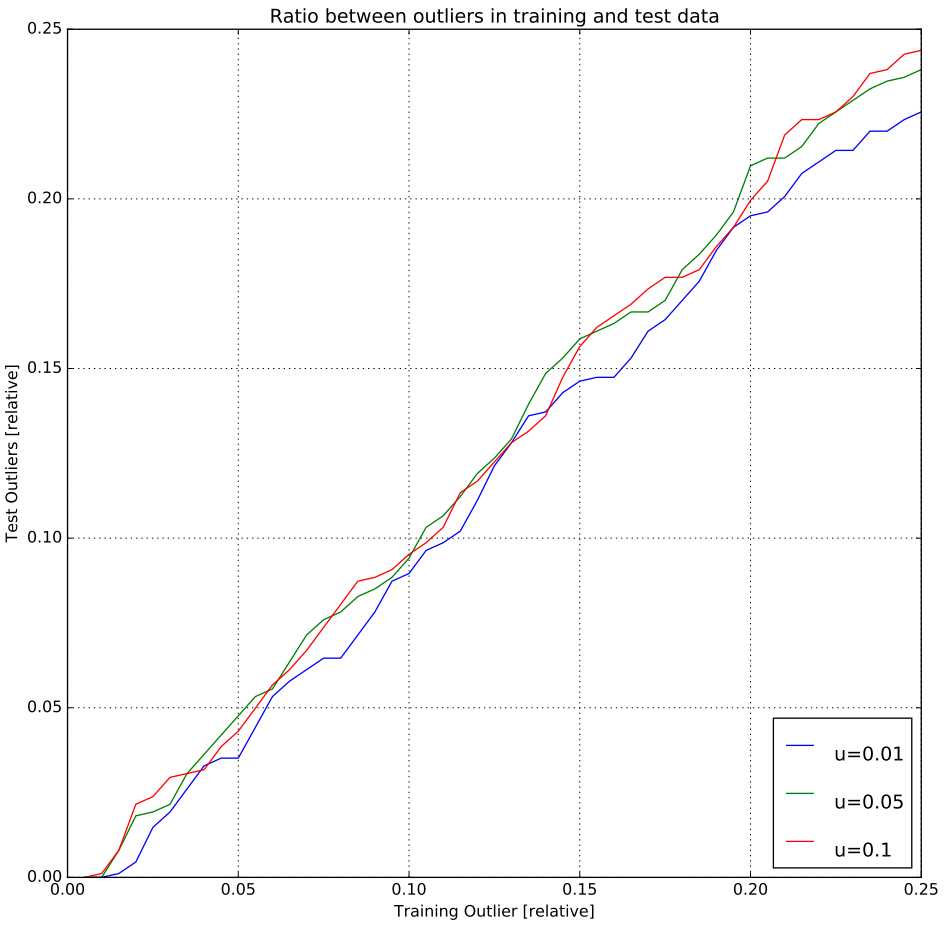


Figure 4.11: Comparison between outlier-removal from the training data and the test data. For each outlier-ratio, a threshold was chosen based on the training data. This threshold was used to remove outliers from the test data, after which the outlier-ratio in the test data was noted and plotted above.

Performance Without Outliers

By thresholding the distrust measures, three dataset were created (subsets of the original data), named by:

No Outliers 1 Dataset with 1% outliers removed.

No Outliers 2 Dataset with 5% outliers removed.

No Outliers 3 Dataset with 10% outliers removed.

The training performance using these datasets can be seen in appendix A.2, where it can be compared to the performance of other methods. When using logistic regression and radial basis function SVM the removal of the outliers significantly increased training performance and for the linear SVM the performance were more or less stable. This suggests that the outliers are samples too difficult for the algorithms to handle. When looking at the test-performance there is no improvement from removing outliers. It is therefore unlikely that the outliers disrupts the ability for the algorithms to learn the underlying structure, but are simply difficult for the model to understand.

4.4.2 Normalization

In some situations, the features of samples can be scaled and normalized to ease the work of a machine learning algorithm. As data is typically represented as a matrix with samples along the rows and features along the columns, the normalization/scaling is here referred to as either row or column normalization/scaling. All results from training of the different scalings/normalizations of the data can be seen in A.2.

Row Normalization

The argument features are mostly bag-of-word representations with a few other features. Thus most of the features are zero and a few will have a small, positive, integral value, generated from bag-of-word representations of the original texts. The lengths of the texts may vary considerably, which causes the vector-lengths to vary considerably as well (as long sentences have more words and word combinations). This can be difficult for machine learning algorithms to handle. An example of how this can happen, is to consider two sentences where one is much longer than the other and contains the other as a substring. The distance between the vectors of two sentences can be very big because the long sentence has a long feature-vector, but the machine learning algorithms should rather notice their similarity. A common preprocessing of data is thus to normalize the feature-space, so that all sample-vectors have length 1. This arranges the samples on a hyperdimensional sphere, so that difference between the samples relate to the angle on the feature-sphere.

Row-normalization greatly increased training performance. Unfortunately this improvement does not generalize to the test data, where the performance is more or less the same. The incapability of generalizing to the test data can be caused by the low amount of data relative to features, and since the increase in training performance is so significant, the sample-normalized data still seems better for describing the data. Sample normalization should therefore be tested in future work if using similar features.

Column Scaling

A different common problem is when the range of the dimensions differ dramatically. In the argument features used, most of the features are bag-of-word representations and will generally take on small, integral values. The remaining features can take higher values, and can therefore have a larger influence on the learning algorithm used. To handle this problem it is common to standardize the features, so that they all have a standard deviation of 1. This was done by multiplying all features by a positive factor. This keeps all zeros and thus maintains the sparsity of the data. The transformation also moves the mean by multiplication of the same factor.

The column scaling provides similar results as the row-normalization when it comes to the training performance, but the test performance drop dramatically. Column scaling therefore reduced the model's ability to generalize to new data and is not recommended.

4.4.3 Feature Reduction

The argument-features are highly multi-dimensional. This can create over-fitting when learning from data, as well as being computationally difficult to handle. An interesting investigation is thus how the classification performance changes when the number of features is reduced. There are many ways of doing this, and two of them are tested in the following.

Principal Component Analysis

Dimensionality reduction using PCA choose linear-combinations of features based on their ability to describe the variance of the data. In this project, the top 300 principal components were selected and trained on to test whether such a feature reduction can assist the classification task. The number of features is similar to that of the Word2Vec scheme allowing fair comparison. The PCA dataset had much lower overfitting tendencies as the previously used features, with a very small performance decrease between training set and test set. The test performance increased slightly with the logistic regression and the RBF SVM, although the linear SVM had a small performance decrease.

Non-Negative Matrix Factorization

Non-negative matrix factorization attempts to determine some inner structure of data, which describes the data to a fair extend with fewer features. For this project the data was mapped into a 100-dimensional feature space using NMF decomposition. The choice of dimensions was caused by the complexity of the task, as estimating the two matrices of NMF was very time-consuming for larger feature spaces. The training performance dropped significantly when using this dataset, indicating either that the number of internal features was too small or that the method is not useful. The training performance generalized to the test set without a significant drop.

4.4.4 Data Choice

Word2Vec

Using the Word2Vec scheme, an additional dataset was created by using the average, 300-dimensional word-embedding of the text-segments as features. The performance of using these features were generally very low compared to the argument features. This may be expected as all sequential information, which should be important for argument detection, is removed. Precisely this limitation was inspiration for the development of the Doc2Vec scheme[37], which is therefore also tested.

Doc2Vec

The Doc2Vec features are claimed to be able to capture sequences of words, and could therefore potentially be useful for detecting arguments, although this capability did not prove useful here. The logistic regression used the Doc2Vec features to fit onto the training data with quite a good training performance, but the test performance was worse than with any other data-set. Both the linear and radial basis function SVMs had bad training performance (very bad for the linear one) and similarly bad test performance. One limitation of the Doc2Vec scheme, relative to the Word2Vec scheme, is that it used a much smaller corpora for training. The Doc2Vec features are trained from the training data derived from the Araucaria database, which is limited to a few thousand sentences, whereas the Word2Vec scheme is pretrained by Google on a 100 billion words dataset (see 4.1.3). The bad performance could therefore be credited to the small amount of training data.

4.4.5 Error Comparison

After tuning the classifiers with various methods, a comparison was made concerning the similarity in errors of the classifiers. The error similarities are visualized from a matrix illustrated by figure 4.12. The diagonal element shows the error rate for each classifier (0% is perfect classification). Off-diagonal elements shows the percentage of samples that *both* classifiers classified incorrectly.

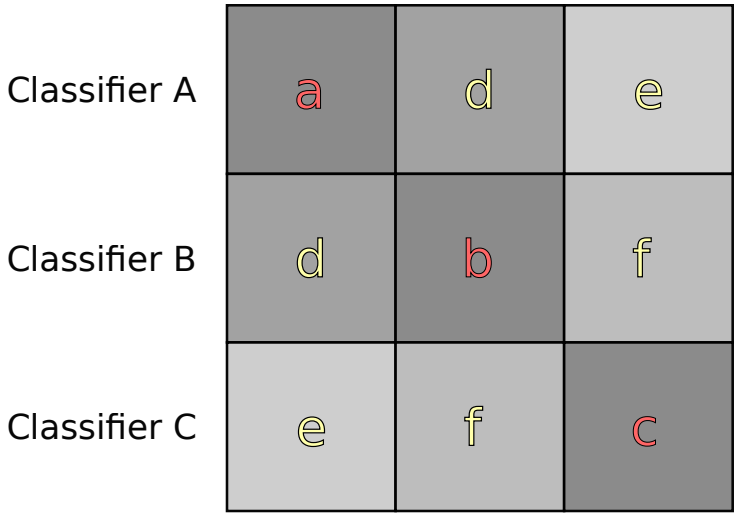


Figure 4.12: An example of a matrix comparing three classifiers (the letters inside the matrix represent rates expressed in percentage). The gray-scaled colouring indicates the error-rate, where darker indicates a higher rate. Elements a, b and c will show the error-rates of classifiers A, B and C respectively. d will show the rate of samples what both classifiers A and B classify incorrectly. Likewise e and f show the rate of samples incorrectly classified by A and C, and B and C respectively. Therefore we have $d \leqslant \min(a, b)$, $e \leqslant \min(a, c)$ and $f \leqslant \min(b, c)$. The off-diagonal elements indicates a lower bound on the possible error rate made from combining two classifiers (assuming knowledge about what classifier to believe at each sample).

The following abbreviations are used in the visualizations:

LogReg Logistic Regression

LinSVM Linear Support Vector Machine

RBFSVM Radial Basis Function Support Vector Machine

Palau The argument-features inspired by Palau and Moens[53]

RowNorm Row normalized (unit length samples)

ColScal Column scaled (unit variance features)

PCA Principal Component Analysis

NMF Non-negative Matrix Factorisation

Figure A.1 in appendix A.3 shows the error-similarity across all learners for all datasets but is quite cluttered. Figure 4.13 has therefore been made for selected learners and datasets.

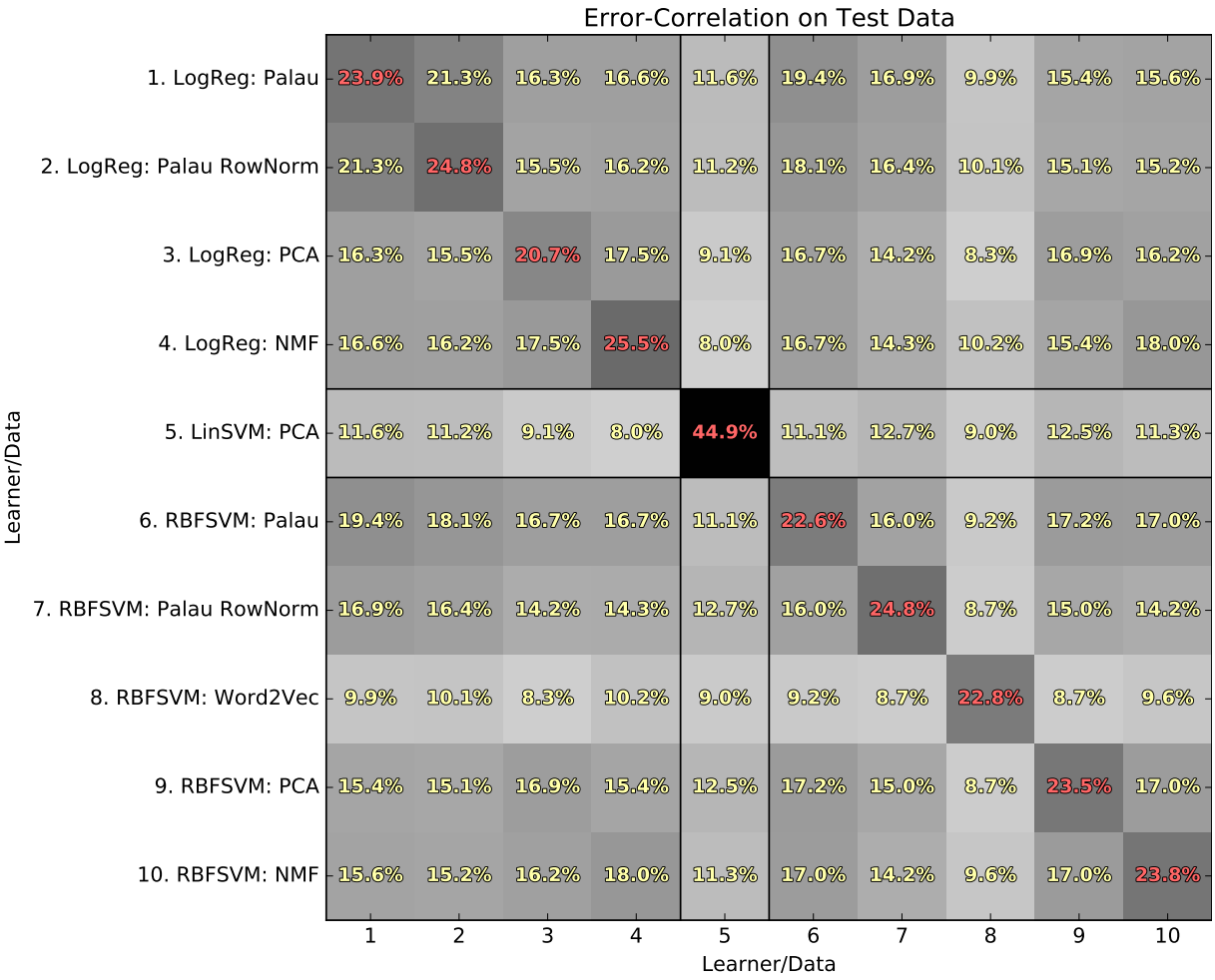


Figure 4.13

There are several combinations that have small error-similarity, especially concerning the linear SVM's usage of the PCA-reduced dataset and the radial basis function SVM's usage of the Word2Vec dataset. The lower bounds for some of the combined classifiers are below 10%. This motivates the next section, which combines some fo the classifiers in a simple ensemble.

4.4.6 Ensemble

A few simple ensembles were constructed by using the predictive values of a set of learners as input to a logistic regression, which produced a final prediction. The ensembles used the following classifiers and data:

Table 4.5: Data and classifiers used in ensembles.

	Ens. 0	Ens. 1	Ens. 2
Logistic: Palau	X	X	X
Logistic: Palau Row Normalized	X	X	X
Logistic: Word2Vec	X		
Logistic: Palau PCA	X	X	X
Logistic: Palau NMF	X		
Linear SVM: Palau	X	X	
Linear SVM: Palau Row Normalized	X	X	
Linear SVM: Word2Vec	X		
Linear SVM: Palau PCA	X	X	X
Linear SVM: Palau NMF	X		
RBF SVM: Palau	X	X	X
RBF SVM: Palau Row Normalized	X	X	X
RBF SVM: Word2Vec	X	X	X
RBF SVM: Palau PCA	X	X	X
RBF SVM: Palau NMF	X	X	

All of the ensemble learners are capable of perfectly fitting to the training data, and so overfitting is expected. The first ensemble use all of the above datasets. Its performance drops quite a bit on the test set attributed to its overfitting. Its performance is lower than some of the learning algorithms it uses. Thus some reduction of the number of learning algorithms should increase performance. The second ensemble has an increased performance which is higher than that of any of the previous single classifiers. The overfitting is thus reduced from the first ensemble to improve performance, motivating the third ensemble, which has further reduced the number of classifiers, but also decreases performance. Its evident that a large number of possible ensembles could be created, but this project limited the number of tests to the above three.

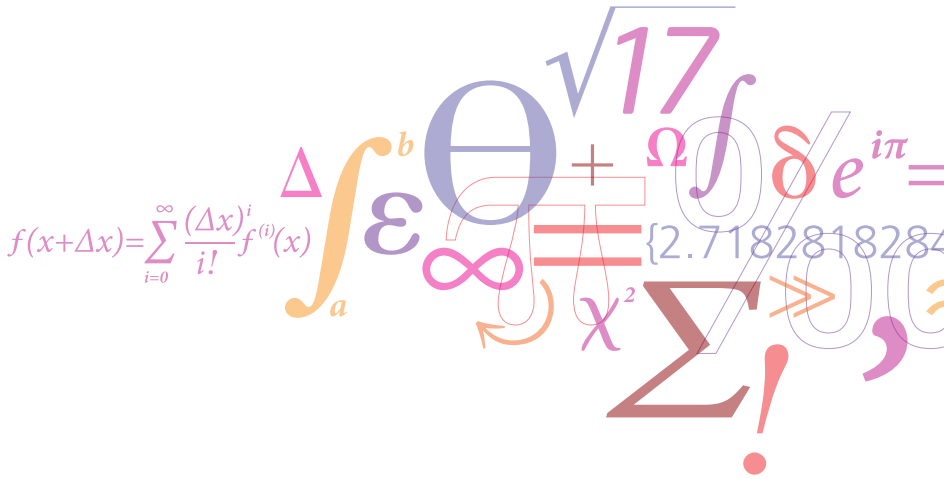
4.4.7 Overall

All three classifier algorithms shows a tendency to overfitting, especially when using the argument-features. This could be caused by the large number of features, which allows for many degrees of freedom in the classifiers and increased flexibility. The overfitting tendency is not seen with the feature reduced PCA and NMF datasets, and is reduced when using Word2Vec and Doc2Vec. The test performance is highest when using logistic regression with the PCA-dataset or RBF SVM with the PCA-dataset or the row-normalized dataset, although the performance boost from using these datasets are not much higher than using the original argument-features. Removing outliers generally did not increase performance.

By combining multiple learners, an ensemble was able to further increase performance by a few percent. The final F1-performance from this system is at 68% and an accuracy of 73%. Comparably the performance reported in [53] was:

> *Using the context-free grammar for parsing the texts we obtain around 60% accuracy in detecting the argumentation structures, while maintaining around 70% F1-measure for recognizing premises and conclusions.*

As mentioned, the task is slightly differently defined in this project, as the premises and claims were already separated in the database and the origins of the data were not known. Furthermore the definition of a correctly detected argumentation structure may be different, as the grammar defined by [53] can detect complex structures, and not just premise-claims pairs. These structures could potentially be created similarly by connected the premise-nodes and claims-nodes that come from the same text-segments in the dataset of this project.



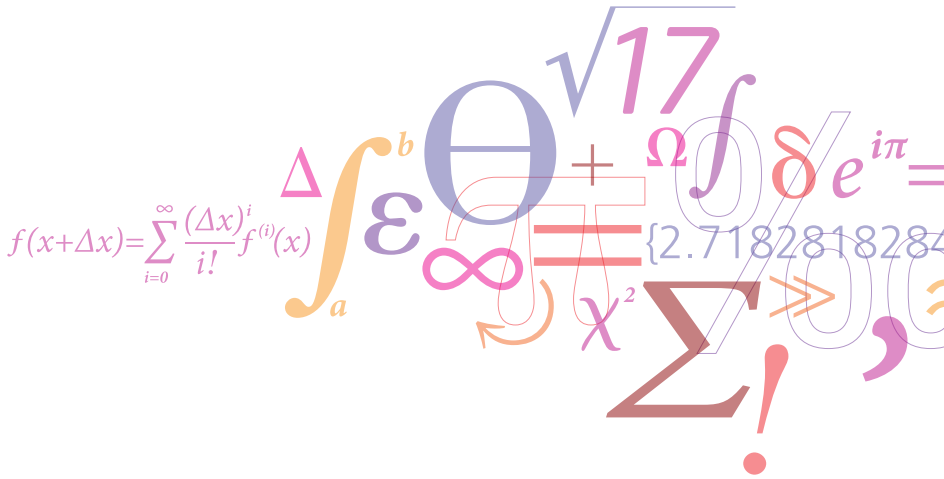
Chapter 5

Conclusion

A survey of argumentation mining has been made, which summarizes both the philosophical theories used, as well as the machine learning techniques and datasets available.

For detecting argumentative elements, a system was created and tested by separating synthetically generated datapoints from the data. Three different methods were used to generate the outlying datapoints, which were all used for challenging the system. The generated samples corresponds to randomly created text, which does not realistically resemble real text, although it allows some initial testing. The system was quite capable of separating the generated outliers, with an F1-score of 91%. A system was also created for separating the labelled argumentative data from a set of unlabelled data, extracted from similar sites as the Araucaria data. The system was able to separate most of the unlabelled data, which either indicates low amount of argumentative elements in the unlabelled data or a large difference in the texts (vocabulary, structures, sentence lengths etc.). The method was used to label some of the unlabelled data, and add this to the training data. The amount of added data was very small (as fairly few of the unlabelled data were inseparable from the argumentative data), which may be the reason for there being no detectable performance increase from adding the training data. The method could though be used in future project for increasing the size of the database. Finally the method of argumentative element detection was questioned, as it may not be the most useful way of handling the task of argumentative mining. The reasoning behind this conclusion largely relies on the hypothesis that most sentences can be used argumentatively in some context (with a few exceptions such as questions, tautologies and inconsistent statements).

The task of detecting argumentative structures were in this project defined as detecting links in a bipartite graph, with premises in one section and claims in the other. The same features were used as in the previous task, together with Word2Vec and Doc2Vec schemes which created alternative datasets. Different ways were tested for making the most use of the argument features, such as sample-normalization, feature-scaling, outlier removal and features reduction (using PCA or NMF). The tested machine learning methods were logistic regression, linear SVM and radial basis function SVM, as well as a simple ensemble combining these methods with a stacking approach. The best performance was create with an ensemble using multiple datasets and all three models, whose outputs were fed to an additional logistic regression. This system was able to detect argumentation links with an F1-performance of 68% and and accuracy of 73%. This performance is similar to the performance quoted in [53], although the comparison is difficult as the problems are defined slightly differently.



Chapter 6

Future Work

6.1 Data

The Araucaria database is mostly useful for detecting argument structured between already-segmented elements, as all data needed for this task exists. Unfortunately the origins of the text-segments do not exist, which creates problems when attempting to use the methods on real data (that is not pre-segmented).

Alternatives to the Araucaria data exists, four of which were briefly looked at in this project (see appendix A.5). The ComArg dataset (A.5.2) was briefly analysed as well and, although no thorough testing was made, this data seems quite useful. The database only has a total of 13 claims, but has many arguments for these claims and is more focussed on the textual entailment task, where the conclusion is known and premises are searched for. The database contains comments from online forums and is therefore inherently different from the law-focussed Araucaria database language-wise. One interesting aspect of the ComArg database is that it is directly extracted from an online forum and methods applied to the database should therefore be applicable on the online site straight away, while more data may be available if needed.

6.2 Argument Detection

The method of detecting argumentative propositions is not considered ideal for argumentation mining. A few alternatives is therefore mentioned here, which could be relevant for future work.

One alternative to detecting all arguments in a text is knowing a given proposition and finding all arguments using that proposition. If the proposition is a claim then this task resembles textual entailment. This problem is the motivation for much of the argumentation mining field; juries often want to search in legal material for proving or disproving a specific claim and online debates often concentrate on specific problems within a domain. Restricting argumentation detection to only search for arguments with specific claims is therefore a very natural approach and could considerably simplify the task.

If all arguments within a text is needed one could modify use a textual entailment system to detect entailment between all sentences. This would scale by N^2 (where N is the number of sentences) and could become intractable. If intractable, topic modelling could be used to encapsulate sentences in related groups an perform entailment detection within the groups.

Similarly distance measures could be used to only compare sentences that already seems related.

Textual entailment detects an entailment relationship between two texts. These texts can be composed by multiple sentences. If one splits the textual data into sentences, then these entailment relationships may be difficult to detect. Dependency parsers and named entity parsers can be used to detect such situations. A logically motivated way te segment texts is *relation extraction*, which attempts to detect logical relationships within texts. These relationships can then be combined to compose th arguments of the text. A survey on relation extraction can be found in [8].

6.3 Link Detection (Argument Structure)

Outliers

4.4.1 concerned the detection of links in a bipartite graph between premises and conclusions, in order to find argument structures. The section detected outliers in the dataset and tested the possible performance improvement from removing these. The performance did not improve on a test-set which suggest that the outliers did not distort the classification task by making the classifier focus on incorrect/noisy data. The training performance did increase a bit though which could indicate that the outliers are samples that are very difficult for the classifier to understand. It could therefore be interesting to see what properties the outliers have in common, and whether these properties can be handled somehow (examples of shared properties could be long sentences, words that only occur in that sentence etc.).

Features

The Word2Vec scheme has gotten a lot of attention and success within the natural language community. The usage in this project is very naive, as it only uses the mean word-embedding of text-segments and thus throws away any sequence knowledge. An interesting experiment would be to combine Word2Vec with a parser in order to make more useful features. Examples of features that seem relevant to argumentation are the word embeddings of the subjects, objects and verbs of each sentence. Another interesting feature would be the mean word-embedding of the delimiters between each sentence or sub-sentence, as these can describe the relationship betweens statements, possibly in the same was that the phrases of [35] can.

A further experiment would be to allow a classifier access to all the generated features: argument-features, PCA-reduced or NMF-reduced dataset, Word2Vec and Doc2Vec. The dimensionality of the new problem will only be slightly increased, as the argument features already has such a high-dimensional feature-space.

Improved Performance Estimate

In order to better determine the performance of the final ensemble system (or one of the individual systems), it would be interesting to make a cross-validation test. The cross-validation would require the training and optimizing of the learning algorithms and dataset listed under "Ens. 1" in table 4.5 multiple times. As one optimization typically required multiple hours, the total processing time for a cross-validation can be very long and has thus not been run yet. From the multiple test results gathered, the variance of the performance can be determined indicating how certain the performance is.

Appendix A

Appendix

A.1 Confusion Table Declaration

Table Definitions

Total n	Real Positives P	Real Negatives N	Accuracy (TN + TP) / n
Predicted Positives P_pred	TP TP / n	FP FP / n	Precision TP / P_pred
Predicted Negatives N_pred	FN FN / n	TN TN / n	NPV TN / N_pred
Error rate (FN + FP) / n	Recall TP / P	Specificity TN / N	F1-score 2*TP/(2*TP+FP+FN)

Abbreviations:

n	Total Samples
P	Positives in Dataset
N	Negatives in Dataset
P_pred	Positives Predicted by Classifier
N_pred	Negatives Predicted by Classifier
TP	True Positives
FP	False Positives
FN	False Negatives
TN	True Negatives
NPV	Negative Prediction Value

A.2 Comparison Between Link Detection Learners

The following two pages lists the performances of various classifiers and data-types on a training set and a test set.

	TP	FP	TN	FN	F1	Recall	Precision	Accuracy	Error rate
Logistic: Palau	2,513	227	5,763	482	0.88	0.84	0.92	0.92	0.08
Logistic: Palau Row Normalized	2,787	113	5,877	208	0.95	0.93	0.96	0.96	0.04
Logistic: Palau Column Scaled	2,906	83	5,907	89	0.97	0.97	0.97	0.98	0.02
Logistic: Word2Vec	473	388	5,602	2,522	0.25	0.16	0.55	0.68	0.32
Logistic: Doc2Vec	868	588	5,402	2,127	0.39	0.29	0.60	0.70	0.30
Logistic: Palau PCA	1,422	377	5,613	1,573	0.59	0.47	0.79	0.78	0.22
Logistic: Palau NMF	957	295	5,695	2,038	0.45	0.32	0.76	0.74	0.26
Logistic: Palau No Outliers 1	2,671	159	5,779	285	0.92	0.90	0.94	0.95	0.05
Logistic: Palau No Outliers 2	2,574	151	5,585	224	0.93	0.92	0.94	0.96	0.04
Logistic: Palau No Outliers 3	1,839	154	5,304	788	0.80	0.70	0.92	0.88	0.12
Linear SVM: Palau	2,741	372	5,618	254	0.90	0.92	0.88	0.93	0.07
Linear SVM: Palau Row Normalized	2,798	333	5,657	197	0.91	0.93	0.89	0.94	0.06
Linear SVM: Palau Column Scaled	2,745	259	5,731	250	0.92	0.92	0.91	0.94	0.06
Linear SVM: Word2Vec	1,524	2,530	3,460	1,471	0.43	0.51	0.38	0.55	0.45
Linear SVM: Doc2Vec	134	81	5,909	2,861	0.08	0.04	0.62	0.67	0.33
Linear SVM: Palau PCA	2,332	3,288	2,702	663	0.54	0.78	0.41	0.56	0.44
Linear SVM: Palau NMF	367	31	5,959	2,628	0.22	0.12	0.92	0.70	0.30
Linear SVM: Palau No Outliers 1	2,681	396	5,542	275	0.89	0.91	0.87	0.92	0.07
Linear SVM: Palau No Outliers 2	2,600	309	5,427	198	0.91	0.93	0.89	0.94	0.06
Linear SVM: Palau No Outliers 3	2,325	189	5,269	302	0.90	0.89	0.92	0.94	0.06
RBF SVM: Palau	2,040	151	5,839	955	0.79	0.68	0.93	0.88	0.12
RBF SVM: Palau Row Normalized	2,995	1	5,989	0	1.00	1.00	1.00	1.00	0.00
RBF SVM: Palau Column Scaled	2,992	5	5,985	3	1.00	1.00	1.00	1.00	0.00
RBF SVM: Word2Vec	2,994	1	5,989	1	1.00	1.00	1.00	1.00	0.00
RBF SVM: Doc2Vec	2,995	5,990	0	0	0.50	1.00	0.33	0.33	0.67
RBF SVM: Palau PCA	1,594	516	5,474	1,401	0.62	0.53	0.76	0.79	0.21
RBF SVM: Palau NMF	1,434	491	5,499	1,561	0.58	0.48	0.74	0.77	0.23
RBF SVM: Palau No Outliers 1	2,407	550	5,388	549	0.81	0.81	0.81	0.88	0.12
RBF SVM: Palau No Outliers 2	2,191	649	5,087	607	0.78	0.78	0.77	0.85	0.15
RBF SVM: Palau No Outliers 3	2,627	1	5,457	0	1.00	1.00	1.00	1.00	0.00
Logistic: Ensemble 1	2,994	1	5,989	1	1.00	1.00	1.00	1.00	0.00
Logistic: Ensemble 2	2,994	1	5,989	1	1.00	1.00	1.00	1.00	0.00
Logistic: Ensemble 3	2,994	1	5,989	1	1.00	1.00	1.00	1.00	0.00
Logistic: Ensemble 4	2,995	1	5,989	0	1.00	1.00	1.00	1.00	0.00

Table A.1: **Training Performance** Comparison Table of Link Detection Systems

	TP	FP	TN	FN	F1	Recall	Precision	Accuracy	Error rate
Logistic: Palau	151	68	520	143	0.59	0.51	0.69	0.76	0.24
Logistic: Palau Row Normalized	150	75	513	144	0.58	0.51	0.67	0.75	0.25
Logistic: Palau Column Scaled	48	68	520	246	0.23	0.16	0.41	0.64	0.36
Logistic: Word2Vec	40	66	522	254	0.20	0.14	0.38	0.64	0.36
Logistic: Doc2Vec	53	108	480	241	0.23	0.18	0.33	0.60	0.40
Logistic: Palau PCA	145	34	554	149	0.61	0.49	0.81	0.79	0.21
Logistic: Palau NMF	91	22	566	203	0.45	0.31	0.81	0.74	0.26
Logistic: Palau No Outliers 1	153	76	512	141	0.59	0.52	0.67	0.75	0.25
Logistic: Palau No Outliers 2	147	69	489	135	0.59	0.52	0.68	0.76	0.24
Logistic: Palau No Outliers 3	127	47	486	138	0.58	0.48	0.73	0.77	0.23
Linear SVM: Palau	173	148	440	121	0.56	0.59	0.54	0.70	0.30
Linear SVM: Palau Row Normalized	172	122	466	122	0.59	0.59	0.59	0.72	0.28
Linear SVM: Palau Column Scaled	50	82	506	244	0.23	0.17	0.38	0.63	0.37
Linear SVM: Word2Vec	130	261	327	164	0.38	0.44	0.33	0.52	0.48
Linear SVM: Doc2Vec	14	27	561	280	0.08	0.05	0.34	0.65	0.35
Linear SVM: Palau PCA	243	345	243	51	0.55	0.83	0.41	0.55	0.45
Linear SVM: Palau NMF	38	0	588	256	0.23	0.13	1.00	0.71	0.29
Linear SVM: Palau No Outliers 1	164	144	444	130	0.54	0.56	0.53	0.69	0.31
Linear SVM: Palau No Outliers 2	159	120	438	123	0.57	0.56	0.57	0.71	0.29
Linear SVM: Palau No Outliers 3	142	98	435	123	0.56	0.54	0.59	0.72	0.28
RBF SVM: Palau	152	57	531	142	0.60	0.52	0.73	0.77	0.23
RBF SVM: Palau Row Normalized	171	96	492	123	0.61	0.58	0.64	0.75	0.25
RBF SVM: Palau Column Scaled	52	104	484	242	0.23	0.18	0.33	0.61	0.39
RBF SVM: Word2Vec	184	91	497	110	0.65	0.63	0.67	0.77	0.23
RBF SVM: Doc2Vec	294	588	0	0	0.50	1.00	0.33	0.33	0.67
RBF SVM: Palau PCA	165	78	510	129	0.61	0.56	0.68	0.77	0.23
RBF SVM: Palau NMF	146	62	526	148	0.58	0.50	0.70	0.76	0.24
RBF SVM: Palau No Outliers 1	170	113	475	124	0.59	0.58	0.60	0.73	0.27
RBF SVM: Palau No Outliers 2	167	121	437	115	0.59	0.59	0.58	0.72	0.28
RBF SVM: Palau No Outliers 3	132	85	448	133	0.55	0.50	0.61	0.73	0.27
Logistic: Ensemble 1	276	351	237	18	0.60	0.94	0.44	0.58	0.42
Logistic: Ensemble 2	248	192	396	46	0.68	0.84	0.56	0.73	0.27
Logistic: Ensemble 3	188	87	501	106	0.66	0.64	0.68	0.78	0.22
Logistic: Ensemble 4	274	449	139	20	0.54	0.93	0.38	0.47	0.53

Table A.2: **Test Performance** Comparison Table of Link Detection Systems

A.3 Complete Error Similarity

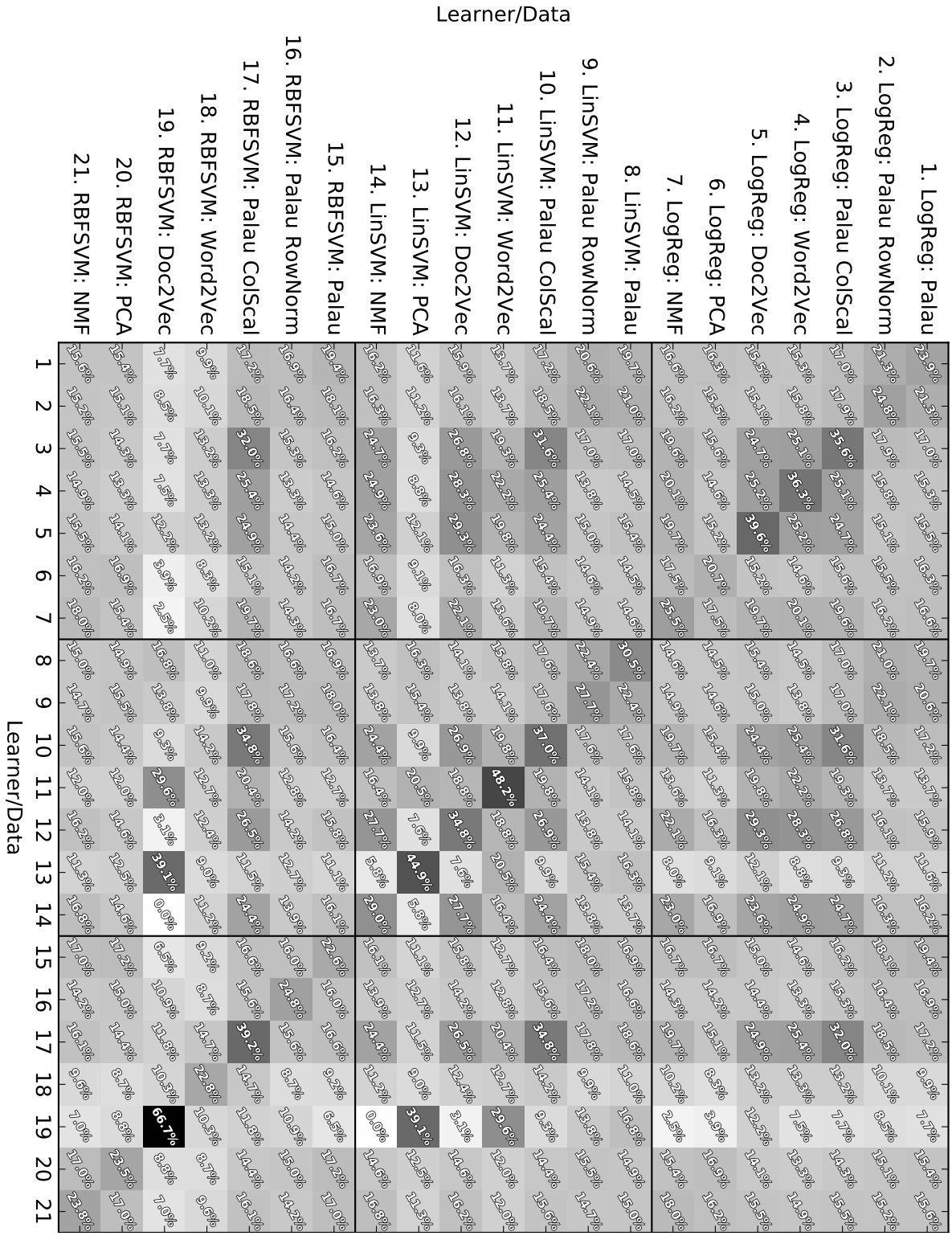
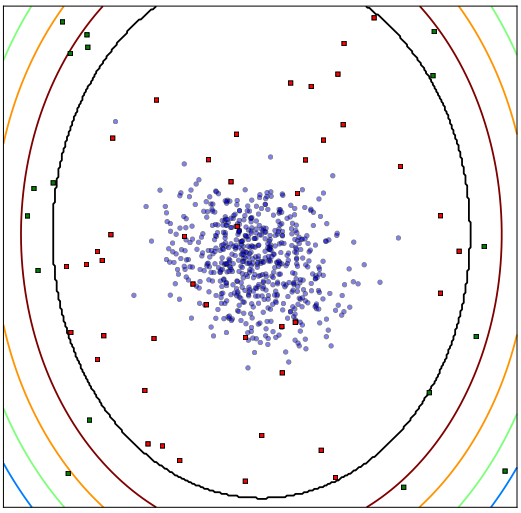
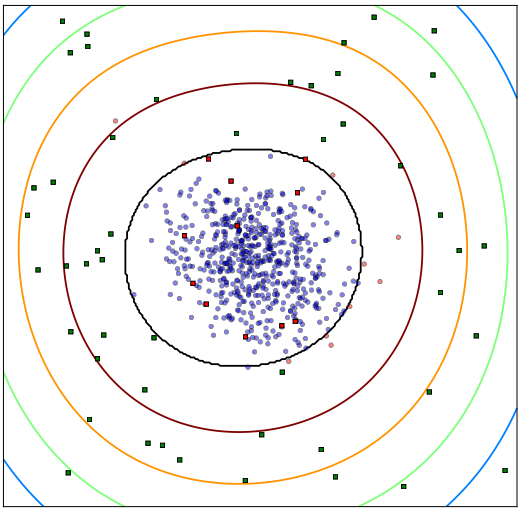


Figure A.1: This figure show the error similarity between all learners and datasets (except the ensembles). The diagonal elements show the error rate of the related classifier, while the off-diagonal elements show the rate of errors made by *both* classifiers.

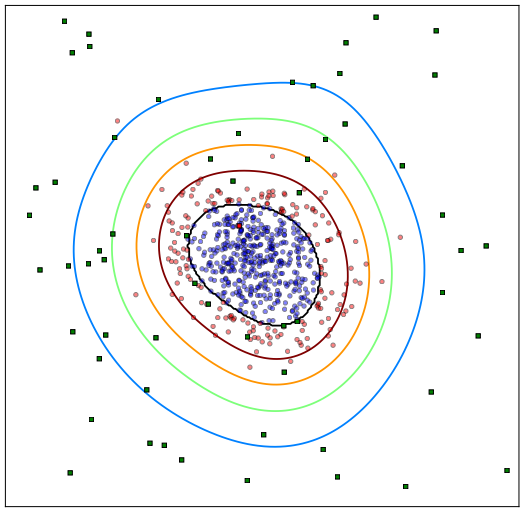
A.4 Outlier Detection on Simulated Data



(a) $\nu = 1.80 \times 10^{-2}, \gamma = 3.70 \times 10^{-7}$
17 / 60 of the uniform samples classified as outliers. 600 / 600 of the Gaussian samples classified as normal samples.



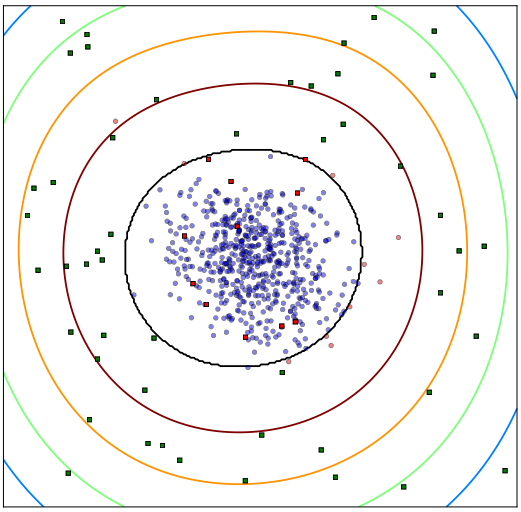
(b) $\nu = 9.10 \times 10^{-2}, \gamma = 1.20 \times 10^{-3}$
49 / 60 of the uniform samples classified as outliers. 590 / 600 of the Gaussian samples classified as normal samples.



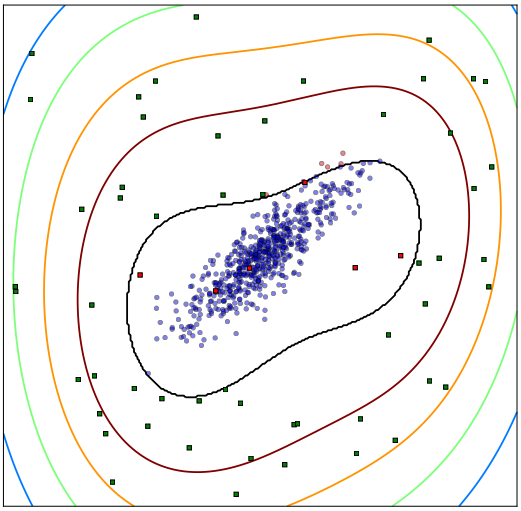
(c) $\nu = 3.60 \times 10^{-1}, \gamma = 9.50 \times 10^{-3}$
59 / 60 of the uniform samples classified as outliers. 420 / 600 of the Gaussian samples classified as normal samples.

Figure A.2: Decision functions for outlier detection, using one-class SVM with radial basis functions, drawing samples from the same distributions. 600 samples were drawn from a Gaussian distribution and 60 samples were drawn from a uniform distribution as outliers (true ν is 9.10×10^{-2}). The Gaussian points are round, while the uniformly distributed points are squares. The Gaussian datapoints are blue in classified as inliers and red if classified as outliers. The uniformly distribution samples are green if classified as outliers, and red otherwise. The "confidence" in classifying as outliers increases for samples distance to the black line, as marked by the coloured contour lines. Parameter ν have been varied in the plots and γ optimized with a simple search.

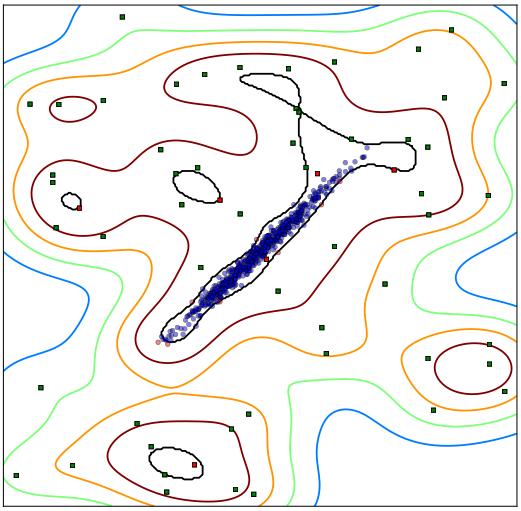
Using the true value of ν (the ratio of outliers) provides a classifier which encapsulates the data in a very appropriate region with few errors.



(a) $\nu = 9.10 \times 10^{-2}, \gamma = 1.20 \times 10^{-3}$
49 / 60 of the uniform samples classified as outliers. 590 / 600 of the Gaussian samples classified as normal samples.



(b) $\nu = 9.10 \times 10^{-2}, \gamma = 3.90 \times 10^{-5}$
54 / 60 of the uniform samples classified as outliers. 595 / 600 of the Gaussian samples classified as normal samples.



(c) $\nu = 9.10 \times 10^{-2}, \gamma = 1.20 \times 10^{-3}$
54 / 60 of the uniform samples classified as outliers. 592 / 600 of the Gaussian samples classified as normal samples.

Figure A.3: These illustrations are similar to those of figure A.2. The true value for ν have been selected, γ optimized with a simple search and the underlying distributions varied.

Some distributions are clearly easier for the SVM to model than others. The radial basis function SVM uses the Euclidean distance from datapoints as part of its kernel, which in indifferent of direction. The round distribution of figure A.3a is easily captured by the distance measure, as datapoints in the main distributions will have many more neighbours than those outside. The more stretched distributions have a different scenario, as datapoints at the edges of the original distribution is only close to part of the original samples. This makes the threshold for neighbours smaller, causing the SVM to detect non-outliers at locations where multiple outliers happen to be close.

A.5 Data Survey

A.5.1 Araucaria

Source	[65]
Purpose	Detect argument structures in set of pre-segmented propositions.
Contains	Argument maps.
Data Size	661 argument maps.
File type	JSON
Format Description	AIF
Format Example	{"nodes":[{"nodeID":"255","text":"Alexander Downer has derided [...]","type":"I","timestamp":"2012-04-12 18:18:22"}, {"nodeID":"256","text":"If the conjecture that members [...]","type":"I","timestamp":"2012-04-12 18:18:22"}, ...], "edges":[{"edgeID":"251","fromID":"256","toID":"258","formEdgeID":null}, {"edgeID":"252","fromID":"257","toID":"258","formEdgeID":null}, ...]}

A.5.2 ComArg

Source	[16] and [15]
Purpose	Textual entailment and opinion mining.
Contains	Data from two debates: "Gay Marriage" and "Under God in Pledge". Each unit is an comment-argument pair, indicating how the comment relate to the argument. Each argument appears in many comments and each comment can have multiple arguments. No segmentation is labelled.
File type	XML
Data Size	Two datasets: **UGIP** 175 comments and 6 arguments **GM** 198 comments and 7 arguments
Format Description	A unit contains: **Comment** The original statement together with the stance of the statement (Pro/Con) **Argument** The argument made (different statements use same arguments) specified by the annotators together with the stance of the argument (Pro/Con) **Label** from 1-5, indicating whether the comment attack, supports or have a neutral stance towards the argument.
Labels:	**1** Explicitly attack argument (A) **2** Implicitly attack argument (a) **3** Neutral position (N) **s** Implicitly support argument (s) **S** Explicitly support argument (S)

Format Example	`<unit id="414721685arg1">`
	`<comment>`
	`<text>Simple, maybe I believe in Allah, or the flying spaghetti monster, or no god at all. Why won't they put and to the republic, for which it stands, one nation under the flying spaghetti monster...?</text>`
	`<stance>Con</stance>`
	`</comment>`
	`<argument>`
	`<text>Separation of state and religion</text>`
	`<stance>Con</stance>`
	`</argument>`
	`<label>3</label>`
	`</unit>`

A.5.3 NoDE

Source	[20] and [21]
Purpose	Textual entailment.
Contains	Pairs of texts and hypotheses.
Data Size	Three datasets: **Debatepedia / Procon** 260 pairs: 140 supports and 120 attacks in 24 argumentation graphs **Twelve Angry Men** 80 pairs: 25 supports and 55 attacks in 3 argumentation graphs **Wikipedia** 452 pairs: 215 supports and 237 attacks in 416 argumentation graphs
File type	XML
Format Description	Each datapoint is a pair between two texts, indicating whether one text entails the other. They contain: **task** Identical throughout database. Indicates that it is from the argumentation database. **id** ID of pair. Unique within topic only. **topic** The topic for the pair. The individual text-ids are unique within each topic. Thus to identify one specific text one needs both the topic and the text-id. **entailment** Whether the text entails the hypothesis or not. **t** Text together with id. **h** Hypothesis together with id.
Labels:	Entailment: YES/NO

Format Example	`<pair task="ARG" id="1" topic="Violentgames" entailment="NO">`
	`<t id="2">Violent video games do not increase aggression.</t>`
	`<h id="1">Violent games make youth more aggressive/violent.</h>`
	`</pair>`

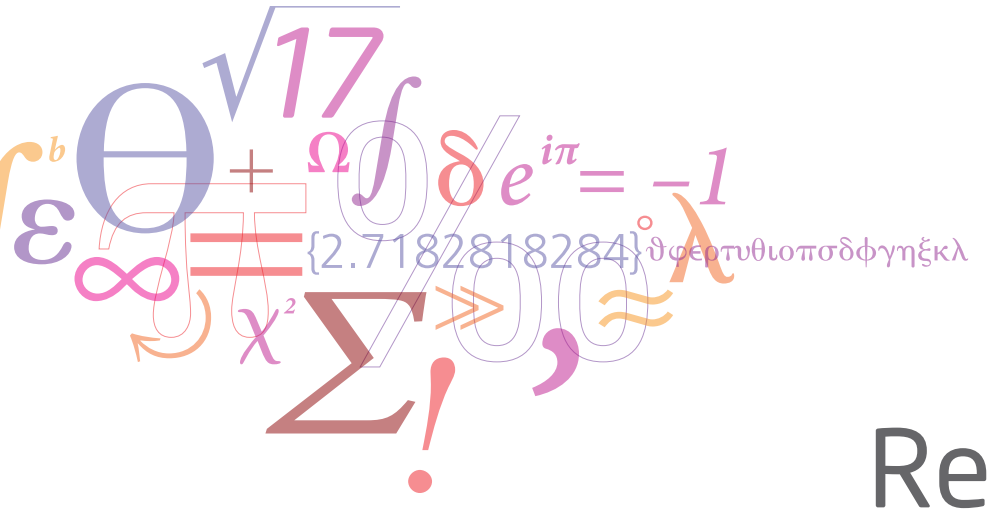
A.5.4 Habernal 2015

Source	[31] and [30]
Purpose	Argumentation mining.
Contains	Three datasets: **gold.data.persuasive** 990 comments and forum posts labeled as persuasive or non-persuasive **gold.data.toulmin** 340 documents annotated with extended Toulming model **unlabeled.raw.corpus** Remaining 4334 documents from the raw corpus that have not been annotated in either of the studies
File type	XML
Format Description	Complicated for showing in single example.

A.5.5 RTE-7

From TAC 2011 RTE Track (RTE-7). Must be requested from `tac-admin@nist.gov`.

Source	[72]
Purpose	Recognizing Textual Entailment.
Contains	A set of hypotheses and a set of texts. One file contains all texts with could potentially entail each of the hypotheses (only consider these texts).
Data Size	Between 25 and 45 hypotheses, each with up to 100 candidate texts.
File type	XML
Format Description	Several files with several formats.



References

[1] S. Abbas and H. Sawamura. Argument Mining Using Highly Structured Argument Repertoire. *Proceedings of the 1st International Conference on Educational Data Mining*, pages 202–209, 2008.

[2] S. Afantenos and N. Asher. Counter-Argumentation and Discourse: A Case Study. *CEUR Workshop Proceedings*, 1341(1999), 2014.

[3] M. Araszkiewicz and A. Lopatkiewicz. Legal Argumentation Concerning Almost Identical Expressions (AIE) in Statutory Texts. *Ceur Workshop Proceedings*, 1341, 2014.

[4] K. D. Ashley. Applying Argument Extraction to Improve Legal Information Retrieval. *CEUR Workshop Proceedings*, 1341, 2014.

[5] I. D. E. Association. Debatepedia. `http://www.debatepedia.org/en/index.php/Welcome_to_Debatepedia%21`, 2016.

[6] I. D. E. Association. Idebate.org. `http://idebate.org/`, 2016.

[7] E. Awad, R. Booth, F. Tohme, and I. Rahwan. Judgment Aggregation in Multi-Agent Argumentation. *CoRR*, abs/1405.6:1–35, 2014.

[8] N. Bach and S. Badaskar. A Review of Relation Extraction. *Literature Review for Language and Statistics II*, 2007.

[9] P. Baroni, M. Giacomin, B. Liao, and L. Van Der Torre. Encompassing uncertainty in argumentation schemes. *CEUR Workshop Proceedings*, 1341, 2014.

[10] B. Bennet. *Logically Fallacious*. eBookIt.com, updated ac edition, 2013.

[11] P. Besnard. *Elements of argumentation*. MIT Press, 2008.

[12] F. Bex and T. Bench-capon. Extracting and Understanding Arguments about Motives from Stories. *Ceur Workshop Proceedings*, 1341, 2014.

[13] F. Bex, H. Prakken, and C. Reed. A formal analysis of the AIF in terms of the ASPIC framework. *Frontiers in Artificial Intelligence and Applications*, 216(17):99–110, 2010.

[14] C. M. Bishop. *Pattern Recognition and Machine Learning*, volume 16. Springer, 2006.

[15] F. Boltužić and J. Šnajder. ComArg - Corpus of Online User Comments with Arguments. `http://takelab.fer.hr/data/comarg/`, 2014.

[16] F. Boltuzic and J. Snajder. Back up your Stance: Recognizing Arguments in Online Discussions. *Proceedings of the First Workshop on Argumentation Mining*, pages 49–58, 2014.

[17] K. Budzynska, M. Janier, J. Kang, C. Reed, P. Saint-Dizier, M. Stede, and O. Yaskorska. Towards Argument Mining from Dialogue. *Frontiers in Artificial Intelligence and Applications*, 266:185–196, 2014.

[18] E. Cabrio and S. Villata. Combining Textual Entailment and Argumentation Theory for Supporting Online Debates Interactions. *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL 2012)*, pages 208–212, 2012.

[19] E. Cabrio and S. Villata. Natural language arguments: A combined approach. *20th European Conference on Artificial Intelligence (ECAI 2012)*, 2012.

[20] E. Cabrio and S. Villata. NoDe - Natural language arguments in online DEbates. `http://www-sop.inria.fr/NoDE/`, 2016.

[21] E. Cabrio, S. Villata, and A. Wyner. Frontiers and Connections between Argumentation Theory and Natural Language Processing. *CEUR-WS.org*, 1341, 2014.

[22] L. Carstens, F. Toni, and V. Evripidou. Argument Mining and Social Debates. *Frontiers in Artificial Intelligence and Applications*, 266:451–452, 2014.

[23] C. Chesñevar, J. McGinnis, S. Modgil, I. Rahwan, C. Reed, G. Simari, M. South, G. Vreeswijk, and S. Willmott. Towards an argument interchange format. *The Knowledge Engineering Review*, 21(04):293–316, 2006.

[24] A. B. El-Sisi and H. M. Mousa. Argumentation Based Negotiation in Multiagent System. *Proceedings - ICCES 2012: 2012 International Conference on Computer Engineering and Systems*, 3(3):261–266, 2012.

[25] European Court of Human Rights. European Court of Human Rights. `http://www.echr.coe.int/Pages/home.aspx?p=home`, 2016.

[26] Google. word2vec. `https://code.google.com/archive/p/word2vec/`, 2016.

[27] N. L. Green. Argumentation for Scientific Claims in a Biomedical Research Article. *CEUR Workshop Proceedings*, 1341, 2014.

[28] D. Grune and C. Jacobs. *Parsing Techniques: A Practical Guide*. Ellis Horwood Limited, 1990.

[29] I. Habernal, J. Eckle-Kohler, and I. Gurevych. Argumentation Mining on the Web from Information Seeking Perspective. *Proceedings of the Workshop on Frontiers and Connections between Argumentation Theory and Natural Language Processing*, pages 26–39, 2014.

[30] I. Habernal and I. Gurevych. Argumentation Mining in User-Generated Web Discourse. *Computational Linguistics*, 2016.

[31] I. Habernal and I. Gurevych. Habernal2016 Corpus. `http://goo.gl/PmbkOl`, 2016.

[32] L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.

[33] H. Kido and Y. Ohsawa. Defensibility-based Classification for Argument Mining. *2014 IEEE International Conference on Data Mining Workshops*, pages 575–580, 2014.

[34] D. Klein and C. D. Manning. Accurate unlexicalized parsing. *Association for Computational Linguistics*, pages 423–430, 2003.

[35] A. Knott and R. Dale. Using Linguistic Phenomena to Motivate a Set of Rhetorical Relations. *Discourse Processes*, 18:35–62, 1994.

[36] J. Lawrence, C. Reed, C. Allen, S. McAlister, A. Ravenscroft, and D. Bourget. Mining Arguments From 19th Century Philosophical Texts Using Topic Based Modelling. *Proceedings of the First Workshop on Argumentation Mining*, pages 79–87, 2014.

[37] Q. Le and T. Mikolov. Distributed Representations of Sentences and Documents. *International Conference on Machine Learning - ICML 2014*, 32:1188–1196, 2014.

[38] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–91, 1999.

[39] V. I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, 1966.

[40] M. Lippi and P. Torroni. Argument Mining: A Machine Learning Perspective. *Theory and Applications of Formal Argumentation - Third International*, pages 163–176, 2010.

[41] I. C. London. Quaestio-it. `http://www.quaestio-it.com/`, 2016.

[42] T. Mikolov, G. Corrado, K. Chen, and J. Dean. Efficient Estimation of Word Representations in Vector Space. *Proceedings of the International Conference on Learning Representations (ICLR 2013)*, pages 1–12, 2013.

[43] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed Representations of Words and Phrases and their Compositionality. *Nips*, pages 1–9, 2013.

[44] R. Mochales and M. F. Moens. Argumentation mining. *Artificial Intelligence and Law*, 19(1):1–22, 2011.

[45] M.-F. Moens. Argumentation Mining: Where are we now, where do we want to be and how do we get there? *Post-proceedings of the forum for information retrieval evaluation (FIRE 2013)*, 2014.

[46] M.-F. F. Moens, E. Boiy, R. M. Palau, and C. Reed. Automatic Detection of Arguments in Legal Texts. *Proceedings of the International Conference on Artificial Intelligence and Law*, pages 225–230, 2007.

[47] NetworkX. NetworkX. `https://networkx.github.io/`, 2016.

[48] H. V. Nguyen and D. J. Litman. Extracting argument and domain words for identifying argument components in texts. *Association for Computational Linguistics: 2nd Workshop on Argumentation Mining*, pages 22–28, 2015.

[49] J. F. Nilsson. *Data Logic*. Technical University of Denmark, 1998.

[50] NLTK Project. Natural Language Processing Toolkit. `http://www.nltk.org/`, 2016.

[51] T. J. Norman. Position paper: Argument in multi-agent systems. *Department of Computing Science, University of Aberdeen*, 2000.

[52] NumPy. NumPy. http://www.numpy.org/, 2016.

[53] R. M. Palau and M.-F. Moens. Argumentation Mining: The Detection, Classification and Structure of Arguments in Text. *Proceedings of the International Conference on Artificial Intelligence and Law, Proc Int Conf Artif Intell Law*, pages 98–107, 2009.

[54] ProCon.org. Procon.org. http://www.procon.org/, 2016.

[55] Python Software Foundation. Re. https://docs.python.org/3.5/library/re.html, 2013.

[56] Python Software Foundation. Regex. https://pypi.python.org/pypi/regex, 2013.

[57] Python Software Foundation. Bisect. https://docs.python.org/3.5/library/bisect.html, 2016.

[58] Python Software Foundation. Pickle. https://docs.python.org/3.5/library/pickle.html, 2016.

[59] Python Software Foundation. Queue. https://docs.python.org/3/library/asyncio-queue.html, 2016.

[60] Python Software Foundation. Shelve. https://docs.python.org/3/library/shelve.html, 2016.

[61] Python Software Foundation. Threading. https://docs.python.org/3/library/threading.html, 2016.

[62] I. Rahwan. Argumentation in Multi-Agent Systems (ArgMAS). http://www.mit.edu/~irahwan/argmas/, 2004-present.

[63] R. Řehůřek and P. Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. http://is.muni.cz/publication/884893/en.

[64] B. Scholkopf, R. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt. Support Vector Method for Novelty Detection. *Nips*, 12:582–588, 1999.

[65] School of Computing at the University of Dundee. AraucariaDB. http://www.arg-tech.org/index.php/projects/araucariadb/, 2016.

[66] Scikit Learn. Scikit Learn. http://scikit-learn.org/stable/, 2016.

[67] Scikit Learn. Scikit Learn OneClassSVM. http://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html, 2016.

[68] SciPy. Scipy. https://www.scipy.org/, 2016.

[69] SIL International. English Word List. http://www.sil.org/ / http://www-01.sil.org/linguistics/wordlists/english/wordlist/wordsEn.txt, 2016.

[70] Stanford University. The Stanford Natural Lanugage Processing Group. http://nlp.stanford.edu/, 2016.

[71] S. Teufel. Scientific argumentation detection as limited-domain intention recognition. *CEUR Workshop Proceedings*, 1341, 2014.

[72] Text Analysis Conference. 2011 RTE Track (RTE-7). `http://www.nist.gov/tac/data/past/2011/RTE-7_Main_Task.html`, 2011.

[73] M. Thimm. Strategic Argumentation in Multi-Agent Systems. *KI - Künstliche Intelligenz*, 28(3):159–168, jun 2014.

[74] Tomas Mikolov. GoogleNews-vectors-negative300.bin.gz. `https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit`, 2016.

[75] S. E. Toulmin. *The Uses of Argument (Updated edition 2003)*. Cambridge, 2003.

[76] F. H. van Eemeren. *A systematic theory of argumentation: the pragma-dialectical approach*. Cambridge University Press, 2004.