UiO : Department of Informatics
University of Oslo

A portable toolkit for detecting negation

Martine Enger

Master's Thesis Autumn 2016

Abstract

In this work, we have developed an open source, portable toolkit for detecting negation cues and their scope in natural language. Our tool is designed with a two-phase architecture, where cue detection and scope resolution are solved using two independent machine learning classifiers. In our implementation, we have built upon the best practices from previous work, in terms of feature design, machine learning algorithms, datasets and evaluation methods, and built the entire system from scratch through large-scale experiments to assess the utility of features and different classifiers.

Acknowledgements

First and foremost, I would like to express my gratitude to my two supervisors, Erik Velldal and Lilja Øvrelid, for your continuous guidance over the past year. Thank you for supporting me when I have doubted myself, and thank you for everything you have taught me throughout this project. Working with this thesis has been much more enjoyable than I expected, and that is mostly thanks to you.

I would also like to give a special thank you to Kjetil Bugge Kristoffersen and Johanne Håøy Horn for taking time out of your own work and studies to read my thesis. Your feedback really made a difference and for that I am very thankful.

I would also like to thank the developers of PyStruct.

Finally, on a more personal note, I would like to thank my friends and family for always supporting me.

Contents

List of Tables

List of Figures

1 Introduction

Negation is a phenomenon in natural language which has been the object of an increasing research interest in the natural language processing (NLP) community in recent years. In logic, negation is simple: It is an operator that reverses the truth value of a proposition p to *not* p, or $\neg p$, which means that it evaluates to true when p is false and vice versa. In natural language, however, negation is much more complex and ambiguous. In its most simple form, it works the same way as in logic, meaning that it reverses the truth value of a statement, such as in "It is **not** sunny outside".

While in logic, the negation operator $\neg$ is well-defined, many different words in natural language can signalise negation, such as **no, never, nothing, not**, and the list goes on. These words are called *negation cues*, and cues are not even necessarily entire words, they can manifest as parts of a word, such as the prefix 'un' in Example 1.0.1. The scope of the negation cue is the parts of the sentence that are negated, i.e. the parts where the meaning is opposed by the negation. A cue does not necessarily affect the whole sentence, it can *scope* over different parts, and sometimes, nothing at all. In Example 1.0.1, the negation scopes over the base of the negation token *interesting*, and we can formulate the opposite meaning with "The lecture was interesting". In Example 1.0.2, we see a situation where the negation cue **No** has an empty scope, meaning that it does not actually reverse the truth value of any part of the sentence. In natural language, the scope of a negation cue can even be ambiguous, such as in Example 1.0.3. Here, it is unclear whether Lisa is not blonde, but blue-eyed, or not blonde and not blue-eyed.

1.0.1 The lecture was **un**interesting.

1.0.2 **No**, we did **not** go to school.

1.0.3 Lisa is **not** blonde and blue-eyed.

Besides from being an interesting phenomenon in itself, negation has proved to be useful for several applications, in particular in problems concerning compositional semantics, i.e. extracting meaning from text. An example of this is sentiment analysis, which is the analysis of opinions in texts, and there are also numerous practical usecases in the medical domain. Moreover, negation detection is very similar to speculation detection, which is the study of uncertainty in text, and systems developed for negation detection can often be used for speculation detection as well.

In recent years, identifying such negation cues and resolving the in-sentence scope has been the theme of several shared tasks in natural language processing. Many systems have been developed for identifying negation computationally, both rule-based and by using machine learning. Many of these systems have achieved an impressive performance, but the problem is that they are simply not publicly available, and they are often rather complex in terms of design and dependencies, which would have been hard to

work with in practice.

The aim of this project is to develop a simple, yet competitive tool for detecting negation cues and their scope in natural language text that will be publicly available. Our goal is that the tool is open-source, portable, with minimal dependencies, and relatively easy to use even for people with marginal experience in programming. We want to draw on the best practices established so far, in terms of feature design, data sets, models and evaluation measures, and conduct large-scale experiments in order to hopefully achieve a performance which is on par with the existing research systems while maintaining low complexity and minimal dependencies.

1.1 Chapter overview

Chapter 2 provides a description of theoretical aspects of negation as well as an overview of previous work and existing data sets. In this chapter, we also compare different machine learning systems for negation detection. Moreover, we provide an introduction to the problem topic of this project in light of previous work.

Chapter 3 contains an overview of the machine learning methods that we will use for our system, namely Support Vector Machines and Conditional Random Fields. Moreover, we present basic principles in structured prediction.

Chapter 4 presents general design choices for our system as well as descriptions of external libraries, details on our implementation and explanations of the evaluation methods that we will use for our machine learners.

Chapter 5 provides details on the implementation and experiments of the cue classifier, where the first part includes descriptions of features, how we handle different cue types, and other details on the design choices we have made for the cue system. The second part is a report of the experiments that we have done to find the best combinations of features, the best classifier and the best hyperparameters for the cue detection system.

Chapter 6 is an in-depth description of the scope resolution system, where the features, labels, hyperparameters and classifiers that we use are described.

Chapter 7 contains details on the experiments that we did with the system for scope resolution, where the main part is dedicated to the evaluation of different feature sets, but we also present experiments of different label sets and fine-tune the hyperparameters of our classifier.

Chapter 8 sums up the implementation of the end-to-end negation tool, where the cue detection system and the scope resolution system are combined to provide end-to-end results. Moreover, this chapter contains held-out evaluation results on the cue classifier and scope resolution classifier in isolation.

Chapter 9 concludes the work we have done in this project and suggests some improvements of our system as well as proposals for future work.

2 Background

In this chapter, we first present a theoretical overview of negation in natural language. Furthermore, previously developed methods for detecting negation cues and resolving negation scope computationally are described, where we focus on the best practices from existing systems. Finally, we present the problem topic of the master thesis.

2.1 Negation in natural language

2.1.1 Theoretical background

Negation is a phenomenon in natural language where, in its simplest form, the truth value of a statement is reversed, i.e. the meaning of an expression is opposed or denied. It is used to express that a state, event or situation does not hold (Morante and Sporleder, 2012). Negation is identified by a *negation cue*, which is one or more tokens that signalise negation. These can either be single-word, multi-word or affixal, such as the prefix 'im' in ***im****possible* in Example 2.1.5, or the suffix 'less' in *end****less***, and in some cases they are discontinuous, such as *neither/nor*. Further examples of negation cues for the different cue types are listed below. Some of the example sentences in this section are taken from the dataset by Morante and Daelemans (2012) which we will work with, described in Section 2.2.1.

- Single-word: No, nor, never, not

  2.1.1 Anna is **not** short and fat.

  2.1.2 **No**, we did not go to the market.

  2.1.3 I asked **no** questions.

- Multi-word: no longer, by no means, for nothing

  2.1.4 There is **no longer** snow outside.

- Affixes: im-, un-, dis-, ir-, in-, -less

  2.1.5 Boys are **im**possible to understand.

Negation cues do not necessarily affect the whole sentence. These operators have a *scope*, and only the part within the scope is affected by the negation, as we see in Example 2.1.8. Identifying the scope is hence a central part of negation detection. There are situations where a negation cue has no scope, i.e. where the scope spans no tokens in the sentence, and identifying these cases is equally important as predicting the right scope sequence. Examples of empty scopes include cue phrases like *not only, not just, not to mention, no wonder*, negative rhetorical questions, and interjections, like

"No" in Example 2.1.2. Furthermore, the scope of a negation cue can be ambiguous, as in Example 2.1.1, where it is unclear whether *not* includes the whole phrase *short and fat* or just *short*.

In some approaches to negation analysis, negations are also annonated with an *event*, which is the property that has its truth value reversed by the negation cue, for example "asked" in Example 2.1.3. This includes the dataset that we will work with, described in Section 2.2.1. However, we will not focus on event detection in this project and will therefore not go into any further explanation of this concept.

Negation can be expressed by several grammatical categories such as verbs, nouns, adverbs and prepositions, which is illustrated in Example 2.1.6- 2.1.11 (Morante and Daelemans, 2012). Furthermore, it can be expressed by affixes, e.g. ***un****happy*. Throughout this thesis, the cue is marked in bold, the scope (if relevant) is marked with square brackets, and the negated event (if relevant), is underlined.

2.1.6 *Verbs:* [I] **fail** to [see how you could have done more].

2.1.7 *Adverbs:* [It was] suggested, but **never** [proved, that the deceased gentlemen may have had valuables in the house, and that their abstraction was the motive of the crime].

2.1.8 *Prepositions and prepositional phrases:* [We did all this hard work] **for nothing**.

2.1.9 *Determiners:* [To us there is] **no** [fiend in hell like Juan Murillo].

2.1.10 *Pronouns:* [I have] **nothing** [to say to you].

2.1.11 *Conjunctions:* It wasn't black, **nor** [was it white].

While negation in its most simple form reverses the truth value of a statement like in Example 2.1.12, it can also be used to express rhetorical phrases. Multiple negation cues within a single phrase or sentence can either resolve to a positive (Example 2.1.13) or intensify the negated event (Example 2.1.14). Negation is also present in *litotes*, where the meaning is expressed by denying the opposite of the statement, such as in Example 2.1.15.

2.1.12 I'm **not** hungry.

2.1.13 I do**n't dis**agree.

2.1.14 I did**n't** go **no**where today.

2.1.15 **Not** bad.

2.1.2 Applications

As discussed by Morante and Sporleder (2012), adding information about negation has shown to be useful for several applications, including recognising textual entailment, machine translation, trustworthiness detection, clinical and biomedical text processing, and identification of text structure. Furthermore, negation is a fairly frequent phenomenon in texts, and twice as frequent in spoken text as in written text. It also occurs quite frequently in health records and biomedical texts (Morante and Sporleder, 2012). Negation detection has shown to be especially useful in the medical domain. In order

to automatically process information contained in clinical reports it is very important to determine whether symptoms, treatments, outcomes, or any other clinically relevant factors are present or not (Morante and Sporleder, 2012).

Negation detection also plays a role in sentiment analysis because it can affect the *polarity* of an expression, i.e. whether the expression is positive, negative or neutral. The relationship between polarity and negation is not entirely trivial – whereas negation can change the polarity of an expression from positive to negative (Example 2.1.16 and 2.1.17), it can also convert negative polarity to neutral or even positive polarity (Example 2.1.18) (Morante and Sporleder, 2012).

2.1.16 This is a fast car.

2.1.17 This is **not** a fast car.

2.1.18 This is **by no means** a slow car.

In the next section, we aim to present a systematic approach to the problem of negation detection. Furthermore, datasets and existing systems that have been used for negation are described, in particular in the context of the 2012 Joint Conference on Lexical and Computational Semantics (*SEM) shared task.

2.2 Previous work and existing data sets

It turns out that the problem of speculation detection is very similar to negation detection. Thus, the methods used for speculation detection can also be used for negation detection. Similarly, the information about uncertainty cues in the BioScope corpus Vincze et al. (2008) can also be used as data for systems that resolve negation cues and their scope because they are annotated similarly with a cue plus scope structure, as shown in Figure 2.2.1.

Until 2012, most research on negation revolved around the medical domain, where negation was used to detect whether a medical term, such as a symptom, is negated or not. A reason for this is that the only corpus available with annotation about negation and speculation was the BioScope corpus, which consists of texts from the biomedical domain only. This corpus was used in the 2010 Conference on Natural Language Learning (CoNLL) shared task (Farkas et al., 2010), where the goal was to detect uncertainty cues and their scope, also known as speculation detection. Until 2009, most research was dedicated to identifying negation or speculation cues and events, not the in-sentence scope of negation, which is the main focus of our project. In 2009, however, Morante and Daelemans (2009) developed a machine-learning system for identifying the full scope of speculation using the BioScope corpus. In the same year, the BioNLP shared task (Kim et al., 2009) was held, where the task was event detection in the biomedical domain.

In the 2010 CoNLL shared task, most systems apply a two-phase architecture, where they split detection of speculation cues and scope resolution. For the cue detection task, many approached the problem as a sequence labelling task (Tang et al. (2010); Zhou et al. (2010)), and some solved it as a word-by-word token classification task (Velldal et al. (2010); Vlachos and Craven (2010)). The problem of finding the in-sentence scope sequence was by many solved in the same way as the cue detection task,

using sequence labelling. While a few systems incorporated only lexical features, the top performing systems all additionally apply syntactic features (Velldal et al. (2010); Morante et al. (2010)).

If we were to use the BioScope corpus as data to train our system, it would make it very restricted in terms of domain because it is not trained on texts from other domains than medicine, and is likely to perform poorly on these texts because the form of the sentences are likely to differ from the biomedical sentences. The ConanDoyle (CD) corpus consists of annotated fictional texts. While the sentences in this corpus can be quite peculiar, we hope that using this as our training data will make our system more robust for general-domain texts because the sentences in the CD corpus are more varied. The annotation form in this corpus, which is based on the CoNLL format, is also easier to work with than the XML-based annotations in the BioScope corpus.

2.2.1 The ConanDoyle dataset

The 2012 *SEM shared task was dedicated to solving the in-sentence scope and focus of negation. Focus detection is the task of finding the part of the scope that is most prominently or explicitly negated, but in this project, we will not work with this task. In the 2012 *SEM shared task, the CD corpus is used for cue detection, scope resolution and event detection.

	CDT	**CDD**	**CDE**
Tokens	65450	13566	19216
Sentences	3644	787	1089
Sentences w/negation	848	144	235
Cues	984	173	264
Unique cues	30	20	20
Scopes	887	168	249

Table 2.2.1: Statistics for the CD dataset (Morante and Blanco, 2012)

The CD corpus contains annotated stories from Sherlock Holmes and consists of the training corpus (CDT), annotated with sentences from *The Hound of the Baskervilles*, the development corpus (CDD), with sentences from *The Adventure of Wisteria Lodge*, and the test corpus (CDE), with text from *The Adventure of the Red Circle* and *The adventure of the Cardboard Box*. We will also work with CDT and CDD together as one dataset, in which case it will be referred to as CDTD. These are all annotated with negation cues, scope and the event or property that is negated. Some statistics, including the number of sentences, cues and scopes in the different data set splits, are shown in Table 2.2.1. The entire corpus is available online[1], and the format of the data is as follows (Morante and Blanco, 2012):

- Column 1: The name of the file
- Column 2: The sentence number
- Column 3: The token number

[1] http://www.clips.ua.ac.be/BiographTA/corpora.html

- Column 4: The word
- Column 5: The lemma of the word
- Column 6: The PoS-tag of the word
- Column 7: The parse tree information
- Column 8-: The negation information
- *Note:* If a sentence does not contain a negation, column 8 contains "***" and there are no more columns
- *Note 2:* If a sentence does contain a negation, the information for each one is encoded in three consecutive columns: negation cue, scope, and negated event respectively

cardboard	8	0	But	But	CC	(S*	_	_	_
cardboard	8	1	the	the	DT	(NP*	_	the	_
cardboard	8	2	morning	morning	NN	*	_	morning	_
cardboard	8	3	paper	paper	NN	*)	_	paper	_
cardboard	8	4	was	be	VBD	(VP*	_	was	_
cardboard	8	5	uninteresting	uninteresting	JJ	(ADJP*))	un	interesting	interesting
cardboard	8	6	.	.	.	*)	_	_	_

Figure 2.2.1: An example sentence from CD with one negation cue.

From Figure 2.2.1, we see that there is one negation cue present in the sentence, namely the prefix 'un'. The scope is "the morning paper was interesting", and the negated event is "interesting". We see that the base that the prefix attaches to is both part of the scope and defined as the negated event.

WL2	108	0	After	After	IN	(S(S(PP*	_	After	_	_	_	_
WL2	108	1	his	his	PRP$	(NP*	_	his	_	_	_	_
WL2	108	2	habit	habit	NN	*))	_	habit	_	_	_	_
WL2	108	3	he	he	PRP	(NP*)	_	he	_	_	_	_
WL2	108	4	said	say	VBD	(VP*	_	said	said	_	_	_
WL2	108	5	nothing	nothing	NN	(NP*)))	nothing	_	_	_	_	_
WL2	108	6	,	,	,	*	_	_	_	_	_	_
WL2	108	7	and	and	CC	*	_	_	_	_	_	_
WL2	108	8	after	after	IN	(S(PP*	_	_	_	_	after	_
WL2	108	9	mine	mine	NN	(NP*))	_	_	_	_	mine	_
WL2	108	10	I	I	PRP	(NP*)	_	_	_	_	I	_
WL2	108	11	asked	ask	VBD	(VP*	_	_	_	_	asked	asked
WL2	108	12	no	no	DT	(NP*	_	_	_	no	_	_
WL2	108	13	questions	question	NNS	*)))	_	_	_	_	questions	_
WL2	108	14	.	.	.	*)	_	_	_	_	_	_

Figure 2.2.2: An example sentence from CD with two negation cues.

From Figure 2.2.2, we see that this sentence contains two cues, the first being "nothing" with the scope "After his habit he said", and the second being "no", with the scope "after mine I asked questions". Observe that the punctuation tokens are not included in the scope. We will not make use of the phrase structure trees in column 7, but convert these trees into dependency structures.

We move on to describe key points of annotations of cues, scope and events in the CD corpus. Only factual events are negated. Some constructions that are characteristic of

other domains are left out, such as contructions that express absence of an entity (e.g. "There were **no** signs of infection"), which are very frequent in clinical notes (Morante and Daelemans, 2012). The negation cue is not included in the scope, and the scope can be discontinuous. The CD corpus also includes annotations from affixal cues, such as **im**possible, where only the affix itself is defined to be the cue, and the base that the affix attaches to is considered to be in-scope, as in Figure 2.2.2. Moreover, Morante and Daelemans (2012) have defined several rules for scope annotation, including: when the negated event is a verb, the full clause is included in the scope of the negation, like in Example 2.2.1. When the object of a verb is negated, the scope includes the clause that has this verb as its dependency head, as in Example 2.2.2. For events that are adjectives, the full noun phrase is annotated as in-scope if the adjective is embedded as a noun phrase, such as in Example 2.2.3.

2.2.1 [We did] **not** [drive up to the door].

2.2.2 [I see] **no** [reason for further concealment].

2.2.3 And if we take this as a working hypothesis we have a fresh basis from which to start our construction of [this] **un**[known visitor].

In this project, we will break the problem of negation detection down to two subtasks, namely finding negation cues and resolving the scopes of these cues, similar to the approach of existing systems. In the 2012 *SEM shared task, event detection was implemented as a third subtask. Resolving the scope of negation is difficult because unlike cue- and event detection, it involves finding non-local and sometimes discontinuous parts of the sentence. Moreover, if there are multiple negation cues in the same sentence, the scopes of these cues can overlap, i.e. one token can be part of the scope for more than one negation cue. Since scope resolution also depends on the cue detection, the error in the cue detection system will propagate to the scope resolution task. These difficulties are reflected in the results, where scope resolution showed by far the lowest performance levels in terms of F-scores in the 2012 *SEM shared task (Morante and Blanco, 2012).

2.3 Systems from the 2012 *SEM shared task

Most systems divide the problem into three modules reflecting the subtasks mentioned in the previous section. In these modules, scope resolution and event detection are independent of each other and both depend on cue detection. Syntax information is widely used especially for the scope task, either in the form of rules or as features for the machine learning model. Multi-word and affixal negation cues are mainly treated separately (Morante and Blanco, 2012). The 2012 *SEM shared task is divided into two tracks. The first track is closed, and only the data supplied by the organisers, such as word form, lemma, PoS-tag and syntactic constituent for each token, may be employed. The other track is open, where participants are allowed to use additional tools and resources, such as external lexicons, parsers, etc (Morante and Blanco, 2012). The systems that participated in the shared task have in general used different features or tuned their features differently for the open and closed track. The choice of features and/or models is often a result of experiments with different setups.

2.3.1 Motivation for classifier and feature design

From existing systems, we see that the ones that choose to employ a machine learner generally use Support Vector Machines (SVMs; see Section 3.2) and/or Conditional Random Fields (CRFs; see Section 3.1) (Morante and Blanco, 2012). Two of the best performing systems in the scope resolution task use CRFs (Lapponi et al. (2012); White (2012)). This is also employed by the best performing system for cue detection (Chowdhury, 2012), but systems that use SVMs for cue detection perform nearly as well, with a score less than 1 percentage point lower than the systems with CRFs. A recurring approach to feature selection is to choose features that target special cases of the problem. For instance, several cue systems apply features that specifically resolve multi-word and/or affixal cues, and for scope resolution, special features are sometimes chosen to resolve discontinuous or overlapping scopes. If the classifier does not apply these specific features, that is, if the developers chose not to let the machine learner be sensitive to these corner cases, these cases are often resolved using post-processing heuristics, such as in the systems of Read et al. (2012) and Lapponi et al. (2012).

For cue detection, the majority of the systems apply morphological and lexical features, like the lemma, PoS-tag and n-grams, while syntax-based features are rarely used. For scope resolution, syntax- and dependency-based features are used more commonly, for instance dependency paths, dependency distances etc. For the scope resolution and event detection, dependency based features can be useful for modelling syntactic relations between a negation cue and the other tokens in a sentence. For cue detection, applying morphological features are especially useful because this problem is fairly lexical in nature, meaning that characteristics of a word is in itself informative towards predicting whether it is a cue or a non-cue. Moreover, several systems employed some sort of scoring function to rank the features in terms of how informative they were, and have chosen features for their classifiers based on those results.

The rationale for using SVMs for cue detection is that you can look at it as a binary problem: a token is either a negation cue, or it is not. An SVM is able to make this classification based on the features extracted from the training data. One could have used other binary classifiers like Decision Trees or Neural Networks, but apparently, SVMs have achieved the best results when experimenting with different classifiers. Several systems have made use of SVMs with default configurations from existing software, while others have tuned the classifier to fit this specific task better.

While using machine learning has shown to produce good results for both negation detection and scope resolution, Packard et al. (2014) developed a rule-based system for scope resolution based on propositional semantics which, in combination with the system of Read et al. (2012), outperformed all participating systems in the 2012 *SEM shared task. The system makes use of the Lingo English Resource Grammar (ERG; Flickinger (2000)) to produce a Minimal Recursion Semantics (MRS; Copestake et al. (2005)) graph. This graph is then traversed according to a set of rules which add constituents to the scope. Interestingly, the results presented in Packard et al. (2014) show that the performance of the MRS-based system improves when combined with the syntax-based system of Read et al. (2012), but on its own, the system has several exceptions which it is not able to handle using the MRS graph, hence the fallback configuration. Even with the fallback configuration, the system performs only marginally better than the existing systems from the 2012 *SEM shared task. The developers used the system of Read et al. (2012) both for cue detection and as a fallback configuration.

2.3.2 Comparison of different CRF-based scope resolvers

All the systems that applied a CRF for scope resolution in the 2012 *SEM shared task get similar F-scores, around 65% to 75% (Morante and Blanco, 2012). The systems used several features that are the same, like lemma, PoS-tag, and syntax-based features, and all the systems had a rather high number of features. What might set the best systems apart from the ones in the lower range, performance-wise, is that the better systems treat different kinds of cues explicitly, either in the form of features targeting different cue-types or post-processing rules. This is reasonable because different types of cues often produce different scope patterns. Also, even though the systems all used a CRF for scope resolution, the methods that were applied for cue detection are diverse. Further, because the cues are used as features, and because false positive and false negative cues will induce corresponding false positive/false negative scopes, the error from the cue detection propagates to the scope resolution task. In turn, this leads to further differences in performance. There will also be a difference due to the software they used, where some systems including Lapponi et al. (2012) used Wapiti, while others used CRFsuite or other tools.

Looking at the systems from the CoNLL 2010 shared task on speculation detection, we see that there is a wider spread in performance among the systems that applied CRFs for scope resolution, with F-scores ranging from 33% to 56% (Farkas et al., 2010). The best performing system (Tang et al., 2010) has applied heuristics both before and after running the CRF for scope resolution, and some of the rules that were used before the scope resolution, i.e. after cue classification, are rather complex. This system also employs many features for scope resolution, some of which are dependency based and quite complex, both in terms of understanding and implementation.

There seems to be two main differences that set the best CRF-based scope resolvers apart from the others. The first is that their systems for cue detection are performing well, so there is less error from that source affecting the scope resolution. The second is that their systems are generally more complex, both in terms of features, where complex dependency based features are often used, and also because their CRF is often supported by numerous pre- and post-processing rules that cover corner cases.

2.4 Existing software for negation detection

There are two known complete toolkits for negation detection that are publicly available, and both of these are developed for use in the medical domain:

- cTakes (clinical Text Analysis Knowledge Extraction System; Savova et al. (2010))
- NegEx (Chapman et al., 2002)

NegEx is an algorithm that can identify negated events in medical records by using regular expressions. The cTakes system is roughly based on NegEx but also applies finite state machines. While these systems do work well for their purpose, they are not able to resolve the in-sentence scope, and they are only applicable for texts from the medical domain, not on texts from other genres.

2.5 This project

One issue with the existing systems is that they are often far more complicated than what we intend to develop, especially with respect to the features they apply, which often utilise information from additional parsers and are hard to implement. Also, they are simply not publicly available, so no one is able to use them for practical purposes.

The purpose of this project is not to make a new "state of the art" program that outperforms all the systems from the 2012 *SEM shared task, but rather to make a toolkit that is available and easy to use for several applications. In order to make the software as applicable as possible, we aim to enable the users to employ the software with minimal dependencies. This means that we can not require that they for instance parse their data. However, we expect the users to be able to run the program from the terminal, and making a graphic user interface is unfortunately out of the scope of this project.

3 Sequence Labelling

Sequence labelling is the task of labelling each element in a sequence. One might go about this by labelling each token independently, but since the label for each token often depends on the neighbouring tokens, finding the globally best sequence of labels generally produces better results. An example of this is PoS-tagging. Some words, such as "lie", can be either a verb or a noun depending on the context. If we were to label "lie" in isolation, there would be no way to know whether it is a verb or a noun. If we look at the structure of the sentence, however, it would be easier to determine, as verbs and nouns differ with respect to their neighbouring tokens.

Scope resolution can be modelled as a sequence labelling task. The sentence can be interpreted as a sequence of tokens, hence the task of scope resolution is to label each token in the sentence as in or out of scope. For sequence labelling, statistical models have shown to be useful because the label of a state *i* in a sequence typically depends on the *i-1* already observed states. When we model the scope resolution task like this, *discriminative* statistical models are often suitable as classifiers because they can model $P(\mathbf{y}|\mathbf{x})$ directly, as opposed to modelling $P(\mathbf{x}, \mathbf{y})$, like *generative* models do. This is often harder because you need to model the underlying probability distribution for each class as opposed to just modelling the boundary between the classes, like discriminative classifiers do. Here, $\mathbf{x}$ is the input sequence, and $\mathbf{y}$ is the target sequence of labels. In general, discriminative models often improve over generative models such as Hidden Markov models because they do not rely on any independence assumptions along $\mathbf{x}$. As a consequence, discriminative models are better at modelling overlapping, highly dependent features, like we often have in natural language problems.

3.1 Conditional Random Fields

Conditional Random Fields (CRFs) is a discriminative statistical model used to predict a sequence of discrete class variables $\mathbf{y}$ given a sequence of observations $\mathbf{x} = \{x_1, x_2, ..., x_K\}$. It does this by modelling $p(\mathbf{y}|\mathbf{x})$. Sutton and McCallum (2012) define a linear-chain CRF as follows:

Given a parameter vector $\theta = \{\theta_k\} \in \mathbb{R}^K$, two sequences of random variables x and y, and a set of real-valued feature functions $\{f_k(y, y', \mathbf{x}_t)\}_{k=1}^K$, then $p(\mathbf{y}|\mathbf{x})$ is given by

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{T} exp \left\{ \sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t) \right\}$$

where t is the index of the state in the sequence, in our case the position in the sentence, and $Z(\mathbf{x})$ is an instance-specific normalisation function given by

$$Z(\mathbf{x}) = \sum_{y} \prod_{t=1}^{T} exp \left\{ \sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, \mathbf{x}_t) \right\}$$

This linear-chain CRF is the discriminative equivalent to the well-known Hidden Markov model, which is generative, meaning that it models $p(\mathbf{y}, \mathbf{x})$. For sequence labelling in natural language processing, this is the most commonly used form of CRFs. Normally, each feature-function f_k takes binary values, i.e. 0 or 1, and is non-zero for a single feature k only when that feature is present. This value is then multiplied by a weight θ_k whose value is learned during training.

To generalise a linear-chain CRF to a general CRF, one can simply move from a linear-chain factor graph to a general factor graph G. Let $F = \{\Psi_a\}$ be the set of factors in the graph G. Partition F into a set of *clique templates* $\mathcal{C} = \{C_1, C_2, ..., C_p\}$ whose parameters are tied, meaning that the parameters are the same for all clique templates in the set. Each clique template C_p is a set of factors which has a corresponding set of sufficient statistics $\{f_{pk}(\mathbf{x}_p, \mathbf{y}_p)\}$ and parameters $\theta_p \in \mathbb{R}^{K(p)}$. Then, the general CRF can be written as

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{y}_c; \theta_p)$$

where each factor is parameterised as

$$\Psi_c(\mathbf{x}_c, \mathbf{y}_c; \theta_p) = exp \left\{ \sum_{k=1}^{K(p)} \theta_{pk} f_{pk}(\mathbf{x}_c, \mathbf{y}_c) \right\}$$

and the normalisation function is

$$Z(\mathbf{x}) = \sum_{y} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{y}_c; \theta_p)$$

In the CRF described above, there are different ways to estimate the model parameter θ. In a conventional CRF, θ is learned through maximum likelihood estimation, while in the toolkit that we will use in this project, PYSTRUCT, θ is estimated through maximum margin learning, which in our case is done by using an SVM. In the remaining chapters of this thesis, we will refer to the maximum margin CRF simply as a CRF, even though it is not fully equal to the conventional CRF.

3.2 Support Vector Machines

A Support Vector Machine (SVM) (Vapnik, 1995) is a non-probabilistic linear binary classifier in which each example is represented as a point in a space. The instances of each class are typically clustered together because they share many of the same features, and the SVM model aims to find the widest gap between the different classes. This is done by finding the *optimal hyperplane* given the training data. It then classifies

unseen instances by mapping them into the same space and using the hyperplane as a decision boundary. An SVM can also be used as a non-linear classifier by mapping the data, using a *kernel function*, to a higher-dimensional feature space where the data is linearly separable.

An optimal hyperplane H is the one with the largest margin of separation between two classes. In other words, it is the hyperplane that is orthogonal to the shortest line connecting the convex hulls of each of the two classes (which is the smallest convex set that contains all the points in the class) and intersects it at the middle point, as shown in Figure 3.2.1.

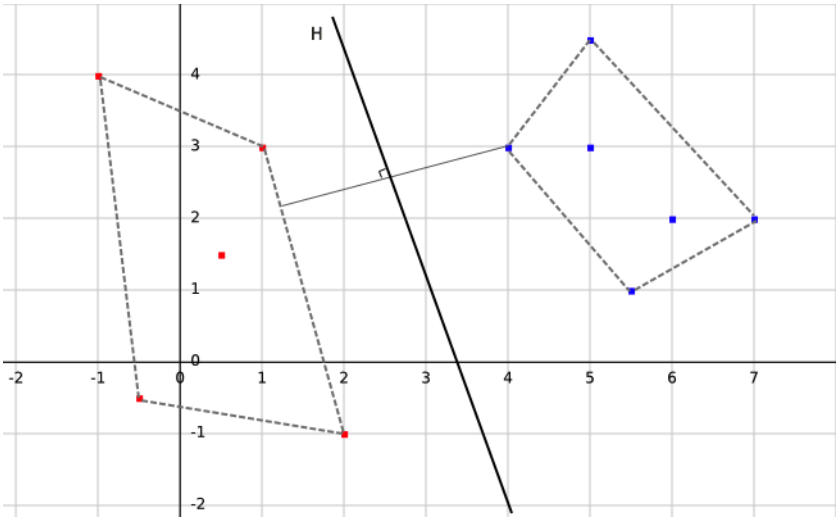

Figure 3.2.1: Support Vector Machine that separates red dots from blue dots.

Computing the optimal hyperplane turns out to be a quadratic *optimisation problem*. To maximise the separation between the two classes, a reasonable approach is to maximise the margin between the two classes. To do this, one can maximise the distance from the hyperplane to the nearest training examples in each class. The generalisation error of the classifier is in most cases lower for a larger margin.

If the data points are linearly separable, one can define two parallel hyperplanes, one for each class, such that the distance between them is maximised. These can be defined as

$$\vec{w} \cdot \vec{x} + b = 1$$
$$\vec{w} \cdot \vec{x} + b = -1$$

The total distance between these hyperplanes are $\frac{2}{||\vec{w}||}$, and to compute the largest margin hyperplane between the two classes, one can minimise $||\vec{w}||$ in that expression. Moreover, to prevent that any data point falls inside the margin, one can define a constraint where the distance between each data point and the margin must be at least 1. Thus, the optimisation problem reduces to:

$$\text{minimise} \quad ||\vec{w}|| \quad \text{subject to}$$
$$y_i(\vec{w} \cdot \vec{x_i} + b) \geq 1, \quad i = 1, ..., N$$

The optimal hyperplane, defined by $\vec{w_0} \cdot \vec{x} + b_0 = 0$, is the unique hyperplane that separates the training points with a maximum margin (Vapnik, 1995). The y-parameter is used to separate the classes and is defined as $y_i = 1$ if x_i belongs to the positive class, -1 otherwise.

3.2.1 Regularisation parameter

The SVM described in the section above is known as a *hard margin* SVM. This means that it will always find the maximum margin hyperplane and that no points will fall inside the margin. However in some cases, one might want to use a *softer* margin, i.e. a hyperplane with a smaller margin, in order to avoid overfitting to the training data. The size of the margin can be tuned with a *regularisation parameter*, C. A low value of C will produce a large margin and vice versa. The effect of this is illustrated in Figure 3.2.2, where the classifier with the large margin is the best if the unseen data points look like they do in (3), even if this margin does not classify the outlier training instance correctly. The classifier with the low margin is best if the unseen data points look like they do in (4). The value of C essentially controls how much weight you put on (a) finding the maximum margin hyperplane, and (b) getting the lowest possible error on the training data. In Figure 3.2.2, graph (1) gives more weight to (a), and graph (2) gives more weight to (b).

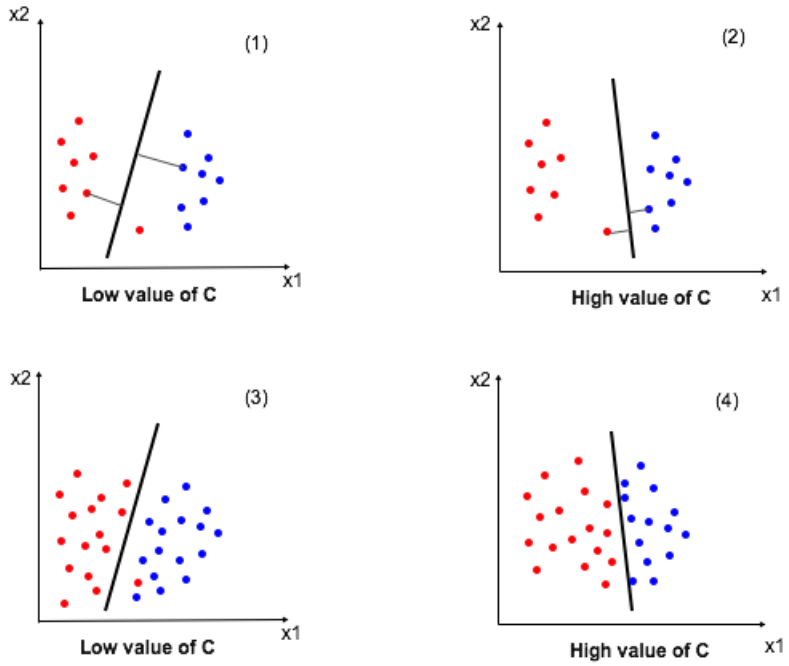

Figure 3.2.2: Effect of different values for C on unseen data.

3.3 Structured prediction

Sequence labelling of the negation scope is an example of a problem with fairly complicated output – we want our classifier to predict an entire sequence, not just a single discrete or real-valued output class. This is a typical case for so-called structured prediction. In structured prediction, a prediction $f(x)$ is made by maximising a compatibility function *g* between the input **x** and the possible labels **y** (Nowozin and Lampert, 2011), but unlike non-structured prediction, the labels **y** can be complex structures such as sequences or trees.

$$f(x) = \underset{y \in \mathcal{Y}}{\operatorname{argmax}}\, g(x, y)$$

The prediction is done by finding y in the equation above. If we use a linear parameterisation of *g*, the equation above becomes:

$$f(x) = \underset{y \in \mathcal{Y}}{\operatorname{argmax}}\, \theta^T \Phi(x, y)$$

In this expression, θ is the parameters of the model and $\Phi(x, y)$ is the *joint feature function* of x and y, i.e. it is the function that models the relationship between x and y by mapping them to a single vector or scalar. The learning of this classifier consists of learning the parameters θ from the training data.

To find θ^T in the compatibility function, one can for example use SVMs. After both θ and $\Phi(x, y)$ are found, one can use the compatibility function on unseen examples to find the predicted structured label for an input **x**. When using a CRF model, this compatibility function corresponds to the maximum margin CRF which we described in Section 3.1.

Finding the optimal hyperplane for an SVM, or finding the optimal structured label given the compatibility function in structured prediction, are both optimisation problems. Common to this class of problems is that you want to find the optimal value of some variable(s) given one or more constraints. For the SVM that our scope classifier applies, we have used the Block Coordinate Frank Wolfe algorithm (Lacoste-Julien et al., 2013). This algorithm proved to be very fast on our data set. Mathematical optimisation is a huge research field in itself, and to go into details of how optimisation algorithms work is unfortunately out of the scope of this thesis.

4 Methodology

This chapter contains descriptions of the implementation and design choices of the tool we developed. Details on feature set, label set, classifiers, frameworks and pre- and post-processing are presented in the sections below.

4.1 Design choices

We decided to implement our tool for negation detection in Python. Python is one of the most widely used programming languages for machine learning and natural language processing as well as in other fields such as biology and medicine, and because of this, there are numerous frameworks available for Python that we can make use of, including a large number of machine learning tools. Moreover, Python is portable and can easily be installed on several operating systems. This will make our tool easy to use, modify and extend by others. For these reasons, Python is a natural choice for programming the source code.

4.1.1 External frameworks and dependencies

We have used the Stanford Parser (Chen and Manning, 2014) before training in order to convert the phrase structure trees in the dataset to dependency relations.

For classification, we have used PYSTRUCT (Müller and Behnke, 2014), which has dependencies that include NUMPY, SCIPY and SCIKIT-LEARN (Pedregosa et al., 2011). When choosing frameworks, we emphasised maintenance and usability. First of all, the frameworks had to be actively maintained to make sure that newfound errors were continuously fixed. Furthermore, since we aim for our toolkit to be used by almost anyone, even people with marginal programming experience, any dependencies that the user has to deal with should be easy to obtain and install.

When we looked for suitable frameworks, we found that PYSTRUCT met all of the above requirements. It comes with an installation guide for both Linux, Mac OSX and Windows, and it can also be seamlessly installed with the Python Anaconda package. Moreover, it is open source, which made it easier for us as developers to use. Using a template-based method like WAPITI (Lavergne et al., 2010) would have produced less code, but with PYSTRUCT we were able to reuse most of the code from the scope resolution for cue detection, while with a template-based method, we would have to reformat the dataset and write new patterns for cue detection, which would have taken much more time than writing the extra code for PYSTRUCT.

4.1.2 PyStruct

PYSTRUCT is an open-source Python library for structured prediction (see Section 3.3). Recall that in structured prediction, a prediction is done by maximising the compatibility function

$$f(x) = \underset{y \in \mathcal{Y}}{\operatorname{argmax}} \, \theta^T \Phi(x, y)$$

between the input x and the labels y. In PYSTRUCT, the functions that model $\Phi(x, y)$ are stored in *model* classes, and the functions that estimate θ^T are *learner* classes. In the remaining parts of this thesis, model classes will be referred to as *models*, and learner classes will be referred to as *learners*. In PYSTRUCT, the learners are *structured* SVMs, SSVMs for short, and the optimisation algorithms that are implemented with this include cutting plane algorithms, block-coordinate Frank Wolfe and latent variable algorithms. In addition to these, PYSTRUCT has an implementation of the structured perceptron. The documentation for this is available online[1].

The training is done in three steps: Optimising the objective function $\theta^T \Phi(x, y)$ with respect to θ, encoding the joint feature function $\Phi(x, y)$, and solving the full optimisation problem above. In this context, the model class that encodes $\Phi(x, y)$ can, for instance, be an implementation of a CRF, and the learner class that estimates θ^T is an SSVM.

4.1.3 Program modes

Ideally, we want our program to be able to run in two different modes. One mode where the user inputs raw text and we parse and tag the data for them, and one mode where they have parsed and tagged their data themselves and we get the sentences on a CoNLL format. This makes the program applicable to more people, both those who are not familiar with taggers and parsers and do not want to do this part themselves and also those who may want to use a different parser than the one we have chosen internally. It is reasonable to think that in the future, we will definitely have access to better taggers and parsers than what we currently have, hence people should be able to use these state-of-the-art toolkits with our program in the future. With the option of providing pre-parsed data, we do not need to do much maintenance to cope with this development.

While being able to provide raw sentences to the program is a definite advantage for some users, it leaves us with a lot more preprocessing as we need to tag and parse the data ourselves. Because of this, we need to do an extra step where we tag and parse the sentences and convert them to CoNLL format, which is illustrated in "mode 2 start" on Figure 4.1.1.

[1] https://pystruct.github.io/references.html

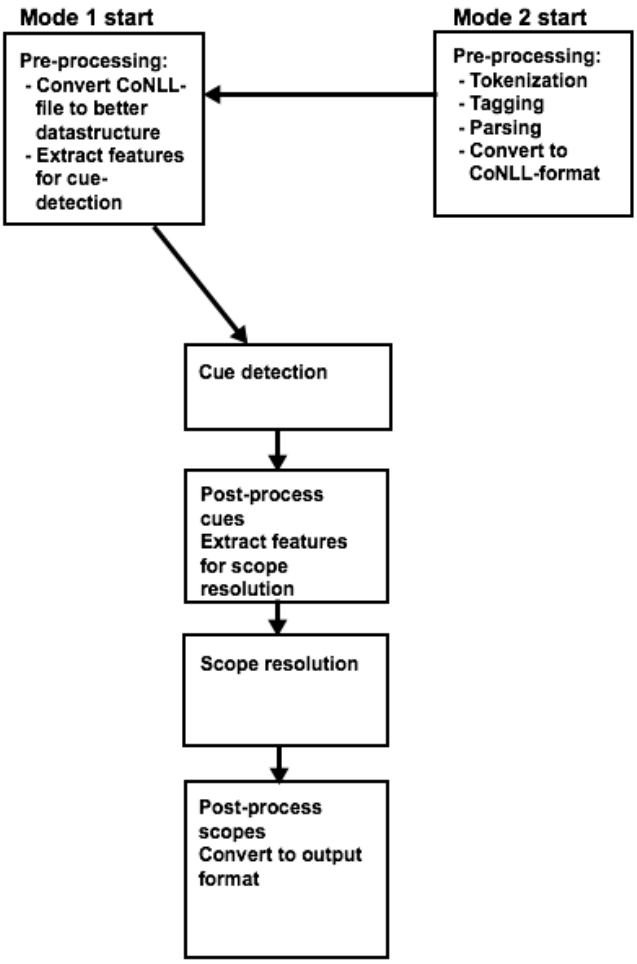

Figure 4.1.1: Pipeline sketch which shows the different processing steps with both modes of the program.

4.2 Preprocessing

In order to utilise dependency based features, we need to parse the input data. The CD dataset already contains phrase structure trees, and we have used the Stanford Parser to convert these to dependency relations between the tokens in each sentence. In order to do this, the format of the token-level phrase structure trees in the dataset was converted to sentence-based PTB-trees. Then, each of these trees was provided to the Stanford parser, which converted them to dependency graphs. The conversion from the token-based format in the dataset to PTB-trees in the standard bracketed format is illustrated with an example in Figure 4.2.1. The dependency graph, which is what we get from the Stanford Parser, for the same example is shown in Figure 6.2.1.

```
No              (S(NP*
woman           *)
would           (VP*
ever            (ADVP*)
send            (VP*
a               (NP*
reply-paid      *
telegram        *)))
.               )
(S1
        (S
                (NP     (DT No)         (NN woman))
                (VP     (MD would)      (ADVP           (RB ever))
                        (VP             (VB send)
                                        (NP             (DT a)
                                                        (JJ reply-paid)
                                                        (NN telegram))
                        ))
                (. .)
        )
)
```

Figure 4.2.1: Conversion of the phrase tree format in dataset to a PTB-phrasetree.

After parsing the input data with the Stanford Parser, we convert the training data to an internal format that makes it easier to extract features for the classifiers. For each token in a sentence, the token, lemma, PoS-tag, head, PoS-tag of the head and dependency relation are stored in a dictionary. In addition to this, each sentence has a "master dictionary" that stores the dictionary and its position in the sentence for each token as well as the cues, scopes, events and whether it is a negated sentence or not. When training the scope classifier, we have access to the gold standard cues and scopes from the training data. When predicting unseen examples, we want to add the cues to this dictionary after cue detection, so we can extract cue dependent features for the scope resolution. For each cue, we store the token, the position in the sentence, and the cue type, which is either 's' for single-word cues, 'a' for affixal cues, or 'm' for multiword cues. An example of how the dictionary will turn out for the sentence *"After his habit he said nothing, and after mine I asked no questions."* with two cues is shown in Figure 4.2.2. The annotations for this sentence in the CD dataset can be found in Figure 2.2.2.

```
{
0:      {0: 'wisteria02', 1: '108', 2: '0', 3: 'After', 4: 'After', 5: 'IN'
        'head': '3', 'head-pos': 'NN', 'deprel': 'case'}
1:      {0: 'wisteria02', 1: '108', 2: '1', 3: 'his', 4: 'his', 5: 'PRP$'
        'head': '3', 'head-pos': 'NN', 'deprel': 'nmod:poss'}
...
...
...
13:     {0: 'wisteria02', 1: '108', 2: '13', 3: 'questions', 4: 'question', 5: 'NNS'
        'head': '12', 'head-pos': 'VBD', 'deprel': 'dobj'}
14:     {0: 'wisteria02', 1: '108', 2: '14', 3: '.', 4: '.', 5: '.'
        'head': '5', 'head-pos': 'VBD', 'deprel': 'punct'}
'neg':      True
'cues':     [['nothing', 5, 's'], ['no', 12, 's']]
'scopes':   {0: [['After', 0], ['his', 1], ['habit', 2], ['he', 3], ['said', 4]]
            1: [['after', 8], ['mine', 9], ['I', 10], ['asked', 11], ['questions', 13]]}
'events':   {0: 'said', 1: 'asked'}
}
```

Figure 4.2.2: Dictionary structure that we extract features from. Example sentence with two negation cues.

The advantage of this data structure is that it is easy to extract features from for both cue classification and especially scope resolution, where the sentence-level entries are useful. The code for obtaining this data structure is somewhat complex, but the advantage is that we have much more control of the data on a sentence-level than what we would for the matrix-shaped data structure in the CD dataset because we have stored sentence-level information in the dictionary, which means that we can easily extract e.g. the cues, their position in the sentence, and the scopes for each cue. Note though, that the inner dictionary for each token corresponds to a matrix-shaped structure, we have just explicitly stored the token indices as keys.

After generating this dictionary structure and extracting the actual features for the sequence labeller, we need to convert our feature dictionaries to binary vectors so we can pass the instances to the PYSTRUCT-classifier in the right format. When we convert our features, we start with a dictionary for each token in the sentence, e.g. `{'token': 'woman', 'pos': 'NN','deprel:', 'nsubj', 'cue-dist': '1', ...}`, and for each feature, we have a set of values, i.e. a value space, that is encountered in the training data. We then let each value represent a dimension and generate a vector with binary values. If the value in the feature dictionary matches the value in dimension *i*, we set $fv[i] = 1$, and zero otherwise, where *fv* is the feature vector. If our value space for a feature is very high-dimensional, i.e. if the number of different values for a feature is high, we end up with a very sparse vector. To do the vectorisation, we use `DictVectorizer` from the SCIKIT-LEARN toolkit. This module creates a vector for each feature and then concatenates them so we end up with a single feature vector for each instance.

4.3 Post-processing

In the development phase, we evaluated our classifiers to reveal any issues and errors in our toolkit. We did this by using the 2012 *SEM evaluation script (see Sec-

tion 4.4.1), which requires the input to be in the same format as in the CD dataset (see Section 2.2.1). After running the scope classifier, we post-process the "flat" sequences of numbers, where the number for each token is either 0 for in-scope, 1 for out-of-scope, 2 for beginning-of-scope or 3 for cue, to match the format shown in Figure 2.2.1. This conversion includes using rules for special cases, shown in the list below. These rules are applied to make sure that we match the required annotation rules in the data file.

- The cue is always out-of-scope
- The base of an affixal cue is always in-scope
- Tokens with the label 2 (for beginning-of-scope) are always converted to in-scope

For the scope task, we get one such flat sequence for each cue in a sentence as output from the classifier. In the cue task, we have a continuous sequence of cue predictions from the cue classifier. When this gets outputed from the classifier prediction, it is one long sequence of all the words in the whole data set. To evaluate the cue classifier, we have split this long sequence up into subsequences that match the sentences in the data set and then converted the new sequences to the CD format. Here, we have added a few post-processing rules to resolve multi-word cues and affixal cues in order to match the required annotation rules.

4.4 Evaluation

Two common measures for evaluating machine learning systems are *precision* and *recall*. Precision is a measure of how good your system is at retrieving relevant instances, i.e. not retrieving items that occur as negative in the gold set. It is defined as

$$P = \frac{TP}{TP + FP}$$

Where TP is the number of true positives and FP is the number of false positives, i.e. instances that the system predicts as positive, but are negative in the gold set.

Recall is a measure of how good your system is at retrieving all positive instances in the gold set. It is defined as

$$R = \frac{TP}{TP + FN}$$

Where FN is the number of false negative instances, i.e. instances that are labelled positive in the gold set and negative by the system.

While precision and recall are both fairly good evaluation measures, we would often like to maximise both of them. In some cases, precision will be very high and recall very low at the same time, e.g. if your system classifies all instances as negative. In that case, P = 1 and R is close to 0. On the contrary, if your system classifies all instances as positive, recall would be 1 and precision close to zero. These two cases are both examples of poor classifiers, even though precision or recall alone is maximised. Because of this, we often use the *F-score* as an evaluation metric, which considers both

precision and recall to compute the score. The F-score is defined as

$$F_\beta = (1 + \beta^2)\frac{PR}{\beta^2 P + R}$$

Here, a higher value for β would put more emphasis on recall, and a lower value would put more emphasis on precision. Setting $\beta = 1$ yields the harmonic mean of precision and recall, and is what we will use in this project. When we mention the F-score in this report, the F_1 measure is what we refer to.

4.4.1 The 2012 *SEM evaluation script

To measure performance and to tune our classifier, we have used the evaluation script of the 2012 *SEM shared task (Morante and Blanco, 2012). For cues, precision, recall and F-score are evaluated using the definitions above. For scopes, there are two different ways of measuring performance, namely *scope-level* and *token-level*. On scope-level, the scope is only counted as a true positive if the entire scope sequence is correctly labelled. If the scope sequence is partially correct, the scope is counted as a false negative. Partially correct scopes occur when the negation cue is correct, but the scope is incorrect and vice versa, and also when both the negation cue and the scope are incorrect. A false positive occurs when the system predicts a non-existing scope, for instance if tokens are labelled as in-scope when the correct scope is empty. On scope-level, there are also two different measures, *cue match* and *no cue match*. In the cue match measurement, the entire cue needs to be correctly labelled in order for the scope to be counted as a true positive. This means that if a cue spans multiple tokens, all the tokens need to be correctly identified as the cue. In the no cue match measurement, however, only one of the cue tokens needs to be correctly labelled as the cue for the scope to be counted as a true positive. On token-level, the evaluation is based on whether each token in the scope sequence is correctly labelled. On this level, a true positive occurs when a token is correctly labelled as in or out of scope, and precision, recall and F-score is computed using the definitions in the previous section, in the same way as for cues.

The evaluation script also provides a second measurement called the B measure, which includes cues and scopes on scope-level. In this measure, precision is calculated as (TP/total of system predictions) instead of (TP/(TP + FP)). This way, precision can be interpreted as the portion of perfect matches among all system predictions. For scopes, this means that a partially correct scope will be counted as both a false positive and a false negative on scope-level, making the precision value much lower than in the main measure.

In this project, we will not use the B-measurements to evaluate our classifiers. For scopes, we will use both the token-level and the scope-level measurements, but we will put more emphasis on the scope-level measurement. When evaluating our scope system with predicted cues, we use the cue match measurement. With gold cues, the cue match and no cue match measure will always be the same.

4.4.2 Significance testing

When testing different configurations of our system, we would often like to know if one version performs better than the other. We are testing each configuration on the development set, but the fact that one system performs better than another on that specific dataset does not necessarily imply that the system is better than the other in general, because the difference might be due to chance. To get a better impression of how the systems perform relative to another, we will apply statistical hypothesis tests to check if the difference in performance is due to chance or not. In these tests, we formulate a *null hypothesis*, H_0, which we initially assume to be true, and an *alternative hypothesis*, H_a, which is the opposite of H_0. For our case, the null hypothesis is that there is no difference in performance between two systems, and the alternative hypothesis is hence that there is a difference. The result of the test will then be whether to reject the null hypothesis in favour of the alternative hypothesis or not. The null hypothesis will be rejected only if there is sufficient *sample evidence* that it is not true, i.e. if the sample data strongly differs from H_0. To determine how much the data is allowed to differ from H_0, we define a significance level α, typically 1% or 5%, which says how big the rejection region should be. The lower the α is, the stronger the evidence needs to be to reject H_0 and hence the smaller the rejection region is. We then want the probability of rejecting H_0 when it is in fact true, called the Type 1 error, to be less than α. Since we are using sample data to make this decision, we need to use a function of the data that we base our decision on. This is called the test statistic, and it is dependent on the distribution of the sample data and the type of hypothesis test we are conducting. To make a conclusion of the test, we calculate the area of the distribution curve of the test statistic that is to the right of the observed value, and also to the left of the negative value if the test is *two-tailed*. This area is called the *p-value*, and if the p-value is less than α, we reject the null hypothesis.

For scope resolution, our sample data is dichotomous and nominal, i.e it only takes two values because a prediction is either correct or incorrect. We choose to make a prediction-wise comparison of our two systems, meaning that we want to look at the prediction pairs, not the overall measures such as the F-score. On token-level, we have many samples, namely 3592 for the development set. Moreover, the values in each pair formed from the two configurations are independent of each other, i.e. x_{1i} is independent of x_{2i}. Having samples with binary values, there are only four possible values for each pair: $(0, 0)$, $(1, 0)$, $(0, 1)$ and $(1, 1)$. $(0, 0)$ means that both systems made a correct prediction, $(1, 0)$ means that system A was correct and system B was incorrect and so on. We will use the McNemar test, which has the following setup:

$H_0 : p_{n10} = p_{n01}$
$H_a : p_{n10} \neq p_{n01}$

Where n_{10} refers to the number of $(1, 0)$ pairs and so on. The test statistic is

$$\chi_2 = \frac{(n_{10} - n_{01})^2}{(n_{10} + n_{01})}$$

This is a standard non-parametric test for pairwise comparison of nominal dichotomous data.

It is important to note what we can and can not conclude from a significance test.

Even if the p-value is less than α and we reject the null hypothesis, we do not know how "untrue" it is, e.g. even if we reject the hypothesis that $p_{n10} = p_{n01}$, we do not know how different they are. Especially with large sample sizes, this difference may be marginal and of no practical significance. Moreover, one can often choose from several tests which have different sensitivity, meaning that especially with large sample sizes, one can find a test that detects a difference from H_0 even if it is very small. On the contrary, if the p-value is greater than α and we can not reject the null hypothesis, the test is simply inconclusive. This does not necessarily mean that H_0 is true, however, there is not enough evidence in the sample data to suggest that it is false.

5 Cue detection

In this chapter, we present details on the implementation of the cue classifier, including features, hyperparameters and learner algorithm. Moreover, we present a set of experiments to find the best configuration of the cue classifier, including tests of different feature sets and learner settings.

5.1 Implementation

We choose to solve the cue detection task as a binary classification problem where each word is classified as cue or non-cue. Our classification approach is heavily based on the systems of Velldal (2011) and Read et al. (2012). For this, we have used an SVM-learner independently as described in Section 3.2, without combining it with any CRF-models, which we have done for the scope classifier. For the classification, we have used PYSTRUCT, see Section 4.1.2. However, this is a simple binary problem and does not have complex structured labels as output. To do this, we chose to use `BinaryClf` as our model class. The reason for using PYSTRUCT even for a binary problem is that we are using PYSTRUCT for the scope resolution task and it is simpler to reuse the same library across the whole system. Moreover, we have tested different learner classes both from PYSTRUCT and SCIKIT-LEARN. These experiments are presented in Section 5.2.

Not all words in the input text are presented to the classifier – we extract a lexicon of known single-word cues from the training data, and if a word is not present in this lexicon, we immediately classify it as non-cue. In addition to this, we extracted a separate lexicon of affixal cues from the training data which consists of the prefixes {*dis, im, in, ir, un*}, the infix *less*, and the suffix *less*. The classifier is presented with any words that match either of these, e.g. words that have a prefix that matches any of the prefixes, so e.g. *impatient* and *image* will be classified, while *singular* and *simple* will not be presented to the classifier. Hence the cue problem is solved as a disambiguation problem for a closed class of words where each candidate cue is disambiguated as an actual cue or a non-cue. This approach will in practice leave more false negatives, leading to a lower maximum recall value. However, Velldal (2011) showed that solving the classification task like this outperformed the word-by-word classification, where one tries to classify all the words in the data set as a cue or non-cue. This is probably because the reduction in the number of false positives is greater than the small increase in false negatives in addition to the vast reduction of the feature space. It also means having much more balanced classes.

5.1.1 Multi-word cues

Multi-word cues are processed with a few simple rules when the classifier has disambiguated all candidate cues. Since there are so few of these words, only 16 occurrences in the entire CDTD, we choose not to let the classifier be sensitive to these instances. It would be counter-intuitive to add e.g. "by" and "means" from the multi-word cue "by no means" to the cue-lexicon, which would lead to the classifier trying to disambiguate every occurrence of "by" in the input data when in reality this is almost never a cue. The post-processing rules simply check if a single-word cue is a part of a multi-word cue in a given context. Multi-word cues are only detected if at least one of the cue tokens have been classified as a single-word cue. For instance, if "no" is present in the sentence and has been classified as a cue, we check if its left neighbour is "by" and its right neighbour is "means", and in that case treat these three words as one cue. Similarly, if "nor" is present, we check if "neither" is also present in the same sentence to the left of "nor" and treat these as a single cue. A list of multiword cues that are covered by these heuristics is shown below.

- neither/nor
- by no means
- no longer
- no more

We could have chosen to implement rules for more multiword cues, but several of the multiword cues in the CD dataset are somewhat annotation specific, like "on the contrary", meaning that even though they have been annotated as a cue in this dataset, we do not want to annotate them as cues.

5.1.2 Affixal cues

In reality, the task of classifying single-word cues and affixal cues are very different. The features used for instances that should be classified as affixal or non-cue are entirely different than the features used for regular instances. In relation to this, one might think that using two separate classifiers to predict cues and affixal cues is a good idea since our classifier is currently doing two somewhat different classification tasks at once. We did not implement this here because our classifier is already achieving a fairly high F-score, but it is definitely something worth looking into in the future.

5.1.3 Features for the cue classification

We intentionally designed our system with simple lexical features because the nature of the problem of detecting cues is fairly lexical, hence incorporating more complex syntactic features is not expected to improve performance. This was confirmed by Velldal (2011). In our system, we have implemented some features that are only active for candidate affixal cues. All of the features that we have used are based on the system of Read et al. (2012). Our initial feature design was aimed to replicate this system's features as precise as we could from the article, but we empirically evaluated different

features and different versions of the features to ensure that the system was optimised with our learner class. The experiments are reported in Section 5.2.

In the list of features below, the two-way character 5-grams for affixal cues are the character 5-grams of the base that the affix attaches to, both from the beginning and the end of the base form.

List of features for cue classification:

- Token
- Lemma
- PoS-tag
- Forward/backward lemma bigram
- Two-way character 5-grams (for affixal cues)

The realisation of these features for two example cues from the sentences 5.1.1 and 5.1.2 are illustrated in Table 5.1.1.

5.1.1 It is **impossible** for me to be absent from London

5.1.2 It may be that you are **not** yourself

Cue	**Features**
impossible	Token: impossible Lemma: impossible PoS-tag: JJ Forward lemma bigram: "*_for" Backward lemma bigram: "is_*" Forward character 5-gram: {possi, poss, pos, po, p} Backward character 5-gram: {sible, ible, ble, le, e} Character 5-gram with affix: sible-im
not	Token: not Lemma: not PoS-tag: RB Forward lemma bigram: "*_yourself" Backward lemma bigram: "are_*"

Table 5.1.1: Features for two example cues

Note that the character n-grams are implemented as *n* different features in our classifier, i.e. a separate feature for the 5-gram, 4-gram, 3-gram and so on. This is because it makes it easier for the classifier to find correlations between different affixal cue candidates even though the entire n-gram does not match, because the shorter the n-gram is, the more common it is across several instances. Moreover, we have tested different versions of these character n-grams. These experiments include adding the affix and the PoS-tag of the candidate cue and testing different values for n, namely 4-grams, 5-grams and 6-grams. We also experimented with different versions of the forward and backwards word bigrams, namely adding a bigram for PoS-tags and lexicalising the bigrams.

5.2 Experiments

In this section, we will carry out experiments with different learner classes, hyperparameters and feature configurations for the cue classifier. Since the cue detection task is an easier classification problem than scope resolution, we will not carry out as many tests as we will do with the scope classifier, but we aim to find the best feature set, learner and hyperparameter values for the cue detection system based on these experiments.

5.2.1 Baseline system

As a baseline for cue detection, we used a standard majority approach where each word was classified as its respective majority class in the training data. In other words, if a word was labelled more as a cue than as a non-cue in the training data, the baseline will classify the word as a cue. Table 5.2.1 shows the results of the baseline settings. The results show that the baseline system performs quite well, with an F-score of 87.42%.

System	Prec	Rec	F1
Baseline	90.68	84.39	87.42

Table 5.2.1: Baseline results for cue detection

5.2.2 Tuning the cue classifier

In this section, we present the results of testing different settings for our cue classifier. We want to investigate the impact that the regularisation parameter C makes on the classifier, and we also want to test different feature combinations and different PYSTRUCT learners as well as different bias values. All systems described in this section are evaluated with the 2012 *SEM shared task evaluation script. The significance tests that we conduct in this section are only based on the samples that the classifier is exposed to, i.e. only candidate cues, which yields 574 samples in total from CDD. The test procedure is described in Section 4.4.2.

All experiments in this section are conducted with a basic configuration, both in terms of hyperparameters and features, that is fixed for all experiments (except e.g. learner class in the experiments with different learner classes). This basic configuration is described in Table 5.2.2. For the `NSlackSSVM` learner, one can split the training data into batches, hence the `batch_size` parameter. When we set this to -1, the learner will process the whole data set as one batch. The only parameter that is not fixed is the regularisation parameter C. This parameter will be fine-tuned for each configuration in order to ensure that it is individually optimised for each configuration.

Parameter	Configuration
Model	`BinaryClf`
Learner	`NSlackSSVM`
Features	Token, lemma $\pm$ 1, PoS-tag, character 5-grams (for affixal cue candidates)
`batch_size`	-1

Table 5.2.2: Basic configuration for the experiments with the cue classifier

To find the best configuration for our system, we tested a range of SVM learners from PYSTRUCT. However, all of these achieved the same results on the development set. Benchmark tests[1] from PYSTRUCT showed that the binary SVM from SCIKIT-LEARN outperforms structured SVM's from PYSTRUCT on an example problem. In light of this, we compare the `SVC` from SCIKIT-LEARN with a linear kernel to the `NSlackSSVM` classifier in PYSTRUCT. The results of this are reported in Table 5.2.3.

Learner class	Prec	Rec	F1
`NSlackSSVM` (PyStruct), C = 0.40	**92.61**	**94.22**	**93.41**
`SVC`(scikit-learn), C = 0.23	92.49	92.49	92.49

Table 5.2.3: Experiments with different SVM classifiers for the cue problem

Table 5.2.3 shows that the SVM class from SCIKIT-LEARN does not outperform the learner class from PYSTRUCT on our data set, which yields an F-score of 93.41%. The F-score of the SCIKIT-LEARN class is however quite close to this, with a range of only 0.92 percentage points from the SCIKIT-LEARN class to the PYSTRUCT learner. A significance test showed that this difference is not significant on the 0.05-level. Moreover, the `NSlackSSVM` and the `OneSlackSSVM` learner classes build upon very similar algorithms, and when we do not work with structured output, it is plausible that these achieve an identical result as we see here. Furthermore, there was a noticeable difference in the training time of the different classifiers, where `SubgradientSSVM`, although identical to the other learners performance-wise, had a much longer training time.

The results of running the basic learner, `NSlackSSVM`, with different values for the regularisation parameter C is presented in Table 5.2.4.

[1] https://pystruct.github.io/auto_examples/plot_binary_svm.html#sphx-glr-auto-examples-plot-binary-svm-py

C-value	Prec	Rec	F1
10^{-3}	86.88	80.35	83.49
10^{-2}	84.38	90.17	87.39
10^{-1}	87.98	93.06	90.45
1	**90.96**	**93.06**	**92.00**
10	90.29	91.33	90.81
10^{2}	89.60	89.60	89.60
10^{3}	89.60	89.60	89.60

Table 5.2.4: Experiments with different C-values for the cue classifier

From Table 5.2.4, we see that the C-value does have a major impact on the performance of the classifier, although it does not seem to be as sensitive as it is for the scope-classifier (see Section 7.3.2). We see a peak at values around 1.0, so in all experiments reported we include an additional step of fine-tuning in the neighbourhood of the peak found in the first pass.

In the PYSTRUCT documentation[2] for the model class that we use, `BinaryClf`, it is recommended to add a constant bias feature to the data. Hence we want to test the effect of setting this bias feature to different values and compare it to the system which has no added bias at all. These experiments are presented in Table 5.2.5.

Bias value	Prec	Rec	F1
0.20	92.61	94.22	93.41
0.50	92.49	92.49	92.49
1.0	92.49	92.49	92.49
1.5	92.44	91.91	92.17
2.0	92.44	91.91	92.17
-1.0	92.49	92.49	92.49
No bias	**92.61**	**94.22**	**93.41**

Table 5.2.5: Tests of different bias values

From Table 5.2.5, we see that the bias value does influence the performance of the classifier. However, none of the systems with added bias outperform the system without bias, which achieves an F-score of 93.41%. There is, however, no difference between this system and the system with a bias of 0.2. We also see that all the F-scores of the systems with added bias are very close to each other, in fact, only two different F-scores are observed among the other bias-values: 92.17% and 92.49%. This is likely to be due to the fact that we add a bias directly to the feature vectors, so the bias will be affected by the regularisation of the SVM. Unfortunately, in PYSTRUCT it does not seem to be possible to add an independent bias term which is not affected by the regularisation.

We move on to test different feature combinations to find the best feature set for the classifier. These experiments are presented in Table 5.2.6. In the experiments described in this table, the character n-grams and affix features are only active for candidate affixal cues.

[2] https://pystruct.github.io/

Features	**Prec**	**Rec**	**F1**
Token, Lemma - 1, C = 0.25	85.23	86.71	85.96
Token, Lemma + 1, C = 0.29	82.45	89.60	85.88
Token, Lemma ± 1, C = 0.70	89.16	85.55	87.32
Token, Lemma, PoS-tag, C = 0.50	86.84	95.38	90.91
Token, Lemma ± 1, PoS C = 0.40	90.96	93.06	92.00
Token, Lemma ± 1, PoS ± 1 C = 0.25	91.91	91.91	91.91
Token, Lemma ± 1, PoS Character 5-grams, C = 0.40	**92.61**	94.22	93.41
Token, Lemma ± 1, PoS Character 4-grams C = 0.70	92.05	93.64	92.84
Token, Lemma ± 1, PoS Character 6-grams, C = 0.40	92.61	94.22	93.41
Token, Lemma ± 1, PoS Character 5-grams + affix C = 0.40	92.13	94.80	93.45
Token, Lemma ± 1, PoS Character 5-grams + affix + PoS C = 0.20	91.62	94.80	93.18
Token, Lemma ± 1, PoS Character 5-grams Affix, C = 0.20	91.67	**95.38**	**93.49**

Table 5.2.6: Experiments with different features for the cue classifier

From Table 5.2.6, we see that the systems that only use surface lexical features actually perform worse than the majority class baseline. It was expected that these initial systems would match the baseline, but with an F-score of 85.96% and 85.88%, this is not the case as the baseline achieves an F-score of 87.42% (see Table 5.2.1). When we add the target PoS-tag however, the F-score goes up to over 90%, which is a clear improvement over the baseline. Moreover, we see that the best performing systems all have affix-specific features incorporated. The system with character 5-grams for candidate affixal cues in addition to the token, lemma, forward/backwards lemma bigrams and PoS-tag achieves an F-score of 93.41% and adding a second feature for candidate affixal cues, namely the affix that the base attaches to, increases the F-score to 93.49%. For this feature, we could either choose to attach the affix to each n-gram or add it as an independent feature. The former configuration achieved an F-score of 93.45% and the latter an F-score of 93.49%, so there is almost no difference between the two settings. To check if the affix itself was really informative to the learner, we carried out a significance test where we compared the systems with character 5-grams with and without the affix feature. The test showed, however, that there was no statistically significant

difference between the two systems on prediction level.

The final configuration for our cue classifier is the system with the `NSlackSSVM` as learner and the features listed below, using C = 0.20 and `batch_size` = -1 as hyperparameters.

- token
- lemma
- forward/backward lemma bigrams
- PoS-tag
- character 5-grams (for candidate affix cues)
- affix (for candidate affix cues)

This system achieves a precision of 91.67%, a recall of 95.38% and an F-score of 93.49%. Note that this system has a slightly lower precision value than several other feature configurations – the highest value we observed for precision was 92.61%, see Table 5.2.6.

5.2.3 Error analysis

The analysis in this section is based on the best performing system with the features token, lemma, forward/backwards lemma bigram, PoS-tag, and character 5-grams and affix for candidate affixal cues.

Even though the SVM cue classifier does perform very well, there are a few words that it misclassifies. The common characteristic of most of these words is that they are all misclassified as affixal cues because they match one of the affixes in the affixal cue lexicon. Examples of these include *underlying, intelligent, insuperable*. One could cope with this problem by checking if the base that the affix attaches to is actually a word in itself. For example, in the case of *underlying*, "derlying" is not a word, hence it would be very unlikely that *underlying* is an affixal cue. Even though this could work, there are a lot of words where the base that the affix attaches to is indeed a word, such as *image*, but the word is still not a cue. We tried to implement this feature with a lexicon from `nltk` (Bird, 2006), however it did not show to increase performance, probably because there are as many new false positives as old false positives that we now are able to detect.

Other errors include fixed expressions like *none the less*, for which our classifier labels *none* as a cue, but the whole expression is labelled as a non-cue in the gold data set.

5.2.4 Comparison with previous work

We will compare our cue classifier with the UiO_1 system and the UiO_2 system from the 2012 *SEM shared task. These two systems used the same cue classifier system. Their results on CDD together with our results on the same data set are reported in Table 5.2.7.

System	Prec	Rec	F1
UiO_1	93.75	95.38	94.56
Our system	91.67	95.38	93.49

Table 5.2.7: Comparison of our cue classifier and previous systems' results on CDD

We see that the F-score of this system is roughly 1 percentage point higher than the score of our top performing system configuration. Moreover, our recall-value is identical, but their precision is over 2 percentage points higher than ours, i.e. their system yields fewer false positive cues. Implementation-wise, the main difference between this system and our system is that they implemented a second affix-specific feature where they look up the base that an affix attaches to and checks whether it is a valid word itself. This is done by generating a lexicon from the training data. By doing this, they will get fewer false positive instances because the model will be less likely to misclassify candidate affixal cues like *imagine* and *underlying*. As mentioned in the previous section, we did test this feature, except that we used a complete lexicon from `nltk` because we do not have the same constraints as the closed-track competition systems had (they could not use additional libraries or data to enhance their features). However, this showed no improvement in performance for us.

6 Scope Resolution

In this chapter, we present the details on the implementation of the scope resolution system. This includes the classifiers we have used, the features that we have chosen, and the output labels that we have used.

6.1 Classifiers

We choose to solve the scope resolution task as a structured prediction problem (see Section 3.3), where we label each token in the scope sequence as in or out of scope. To implement this in Python, we have used the library PYSTRUCT (see Section 4.1.2). In previous work, CRFs are widely used to solve sequence labelling tasks and also this specific problem of scope resolution, so we aimed to use CRFs as well. In PYSTRUCT, however, it is not possible to use CRFs alone, so we have used CRF as our model and used a structured SVM as our learner.

In our case it was natural to choose `ChainCRF` as our model because it is the only implementation of a linear chain CRF. All the other CRF-models are graph-based, which does not fit well with our problem. To choose a learner, we tested four different types, and we found that `FrankWolfeSSVM` was both fast and performed best. Thus, we ended up choosing this algorithm as our learner.

In addition to this, the learners can be tuned in several ways. The two most relevant tuning options for us is that one can vary the regularisation parameter *C*, and one can vary the maximum number of iterations that the learner will pass over the data set to find the optimal solution. The regularisation parameter will affect the hyperplane that is used as a decision boundary for the SVM. With a high value of C, the learner will choose a hyperplane with a smaller margin if that hyperplane predicts more points right from the training data. Conversely, a low value of C will make the learner choose a hyperplane with a larger margin, even if it misclassifies more training points.

All experiments mentioned in this section are described in Section 7.3.

6.2 Features

In the following section, we present a systematic description of the feature set that was used with our scope classifier. We have intentionally omitted complex features that are hard to code in our system. The reason for this is that as previously mentioned, we do not aim to outperform existing research systems in this project. The small performance gain we might obtain from encoding such features is not sufficient to justify the extra time and code for this. In addition to theoretical motives for choosing a feature set, we

also compared different sets of features empirically in order to find the best one. These tests can be found in Section 7.3. Moreover, we could have applied more combinations of lexicalised features, but testing showed that in most cases, adding the lexicalised version of a feature did not improve performance. Hence we only included the lexicalised version if it actually proved to be informative to the classifier. Too many combinations of that kind seemed to confuse the classifier more than aiding it to a better separation between the classes.

We also purposely avoided features that can take many different values. This is because if the feature takes too many values, there would be very few occurrences of each value in the training set and probably many unseen values in the new input data, and this would make it very difficult for the classifier to generalise across classes. Thus, such features would most likely provide no further information to the classifier. It would only contribute to making the feature space even more high dimensional because each value takes a dimension, and with many different values, the number of dimensions needed to encode this would be huge. For the classifier, this would substantially increase the complexity during training and hence also the training time.

Following the previous argument about reducing dimensionality, we discretised all distance features that take real values, i.e. the cue distance and the directed dependency distance, which are described in Section 6.2.3. For the directed dependency path, this was done by choosing two limits, 3 and 7, and stating that any token that had a distance less than 3 is an immediate neighbour to the cue head, and any token that had a distance less than 7 is nearby, and any token that had a distance bigger than this is far away. Hence we reduced this real-valued feature to only take three different values, which in most cases was helpful to the learner according to the tests.

Fancellu et al. (2016) found that 95% of all scope tokens fall in a window of 9 tokens to the left and 15 tokens to the right of the cue. We used this observation to set a limit of 9 to the left and 15 to the right when we made the cue distance discrete.

6.2.1 Lexical features

Previous work has shown that using lexical features produce good results. The lexical features we have decided to include are heavily based on the system of Lapponi (2012), but in addition to this, we tested different combinations to find the best set of lexical features.

- Token
- Lemma
- PoS-tag
- Lexicalised PoS-tag
- Backward lemma bigram
- Forward lemma bigram

6.2.2 Syntactic features

The dependency features are thought to model more general traits within the sentence to find correlations with the different labels that the lexical features are not able to grasp. In the list below, "second order head" refers to the head of the head.

- Dependency relation
- PoS-tag of head
- PoS-tag of second order head
- Directed dependency path from head of the cue to each token
- Dependency graph path from each token to cue

6.2.3 Cue dependent features

Results from previous work indicate that there is a lot to gain from treating different kinds of cues explicitly, i.e. single-word, multi-word and affixal. Hence the system should include features that specifically target these differences. Lapponi et al. (2012) found that the frequency distribution over PoS-tags varies with different kinds of cues and non-cues, hence this can be a useful feature in both cue classification and scope resolution. For example, the majority class for morphological cues is adjectives, which typically generate different scope patterns compared to adverbs, which is the majority class of standard cues. Hence the structure of the scope is likely to vary with the type of cue. We also included the distance between each token and the cue because a token is more likely to be inside the scope if it is close to the cue and vice versa. For multi-word cues, the distance is computed from the closest cue token.

- Cue type (single-word, multi-word, affixal)
- Distance between cue and each token, which is negative if the token is to the left of the cue and positive otherwise
- PoS-tag of cue

Figure 6.2.1 shows a negation sentence with one cue together with its dependency graph. For the token "woman", the features included are shown in Table 6.2.1.

Feature	**Value**
dependency relation:	nsubj
head-PoS:	VB
forward bigram lemma:	*_would
lexicalised forward bigram lemma:	woman_would
backward bigram lemma:	no_*
lexicalised backward bigram lemma:	no_woman

Table 6.2.1: Selected features for an example sentence

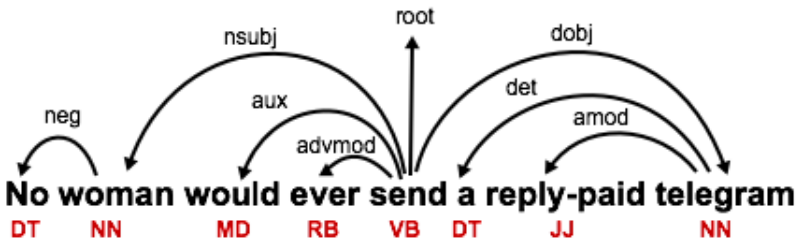

Figure 6.2.1: Dependency features and PoS-tags for an example sentence with one negation cue.

The directed dependency path is the shortest path from the head of the cue to the current token. This path is computed using Dijkstra's algorithm (Dijkstra, 1959). The reason why we compute the path from the head of the cue and not the cue itself is because the cue is very often a leaf node in the dependency graph, meaning that it has no outgoing edges and is not the head of any other token, which would yield no path to any token in the sentence. For example, the directed dependency path from "woman", which is the head of the cue, to "reply-paid" is ["1", "4", "7", "6"]. We experimented with different versions of this feature, including the actual distance of the path, i.e. the number of edges travelled, which is 3 for this example.

Since this feature proved to increase the performance, and in particular the recall value, we decided to develop another feature that combined information about the cue and the dependency graph. Inspired by Gildea and Jurafsky (2002), we implemented the dependency graph path from each token to the cue. This is computed by using the *bidirectional* dependency graph, i.e. the dependency graph where the edges are extended with the reversed edges, so you can move both ways through each arch, similar to an undirected graph. When computing this path, we record both the direction of the arch and the dependency relation between the connecting nodes on the edge. For the example sentence in Figure 6.2.1, the dependency graph path from "reply-paid" to "no" is "\amod\dobj/nsubj/neg", where "\" represents a reversed arch, and "/" represents an edge with the original direction.

The dependency graphs and shortest paths for these features are implemented using the NETWORKX package (Hagberg et al., 2008).

6.3 Label set

Since we solve the scope resolution as a sequence labelling task, we need to define a set of labels from which we predict a label for each token in the sequence. An intuitive approach is to use the labels *I, O*, for inside- and outside of scope, respectively. However, experiments showed that precision and recall were increased by using a more granular label set. We adopted the IOB chunking scheme (Jurafsky and Martin, 2009), which uses the tags inside, outside and beginning of scope. The B-tag is assigned to the first token in the in-scope part of the sequence. We extended this label set with a dedicated label for cues. The cue-label is a bit peculiar since it is information that

we do not really need to predict because the cue in the sequence is already known. However, it has shown to be useful to have a separate label for this, and this label set, i.e. {I, O, B, C}, achieved the best results in experiments with the development set. We could have chosen to use an even more granular label set. For example, we could have a dedicated label for morphological cues, but since we already have a feature to target this trait, the sequence labeller should be able to learn the difference without a separate label. This was also reflected in testing, where adding a label for morphological cues was not beneficial to performance. More information about the experiments can be found in Section 7.3.

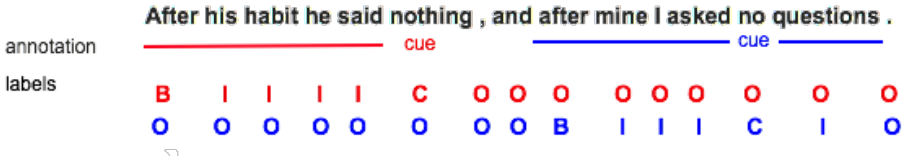

Figure 6.3.1: Labels for an example sentence from Conan Doyle with two negation cues.

Figure 6.3.1 shows scope labels for an example sentence with two negation cues. Since there are two cues in the sentence, we will get two scope sequences. We see that the cues are only labelled with 'C' in their respective scope sequences, otherwise they are labelled as any other token in the sentence. In this example, the scopes do not overlap, but any token in the sentence can be labelled as in-scope for more than one cue.

7 Experiments on the scope classifier

In this chapter, we present a set of experiments that are designed to determine the best learner and hyperparameters, features and labels for our scope classifier. After tuning the scope classifier, we also investigate what kind of errors it makes and look at how we can mitigate these issues.

7.1 Establishing a baseline

Before conducting these experiments and testing the performance of different label sets, learners and so on, we first want to establish a *baseline system*. A baseline system is the simplest way to classify our data, which often means to predict the majority class for all instances. For scope resolution, a suitable baseline is letting the whole sentence be in-scope if there is a negation cue present, and if not, let the whole sentence be out-of-scope. We will use this baseline as a reference point for comparing our results, so we can assess whether changes such as adding a feature or tuning a learner will add any value to our system's performance. Precision, recall and F-score for the baseline are reported in Table 7.1.1. We see that the baseline obtains an F-score of 32.03% on scope-level and 61.59% on token-level.

Scope Level			**Token Level**		
Prec	**Rec**	**F1**	**Prec**	**Rec**	**F1**
86.84	19.64	32.03	45.00	97.55	61.59

Table 7.1.1: Baseline results for scope resolution

7.2 A note on the Block Coordinate Frank Wolfe Structural SVM

The `FrankWolfeSSVM` learner class, which we chose as our learner class in the experiments in this chapter, uses the Block Coordinate Frank Wolfe optimisation algorithm as described in Lacoste-Julien et al. (2013). This algorithm includes initialisation based on random numbers, hence we need to take extra precautions when we test different feature sets, label sets, learners and so on. This is to make sure that any fluctuations in the results are solely due to the parameters we changed, e.g. because we added a

feature, and not due to the randomness in the algorithm, which may have produced a very good or bad result by chance. To cope with this, we *freeze* the random seed that the algorithm uses for all the tests in this chapter. This means that the algorithm will use the same numbers in each run, which would otherwise be random-generated, so in practice, there is no longer any randomness to it. This way, we can be sure that if e.g. a set of features improves performance over the baseline settings, it is correlated only to the features, not the randomness of the algorithm.

To freeze the seed, one can give an integer as input as the `random_state` parameter to the learner. The default value of this parameter is `None`, which will produce a new random seed for every run. After running ten experiments with the same basic settings described in Table 7.3.2 and random values between 0 and 100 for the `random_state` parameter, we found that the average F-score is 71.94 and the standard deviation is 0.7269. The maximum F-score was achieved with `random_state = 10`, and as a result of this, we have used this value for all the experiments in this section.

7.3 Tuning the scope classifier

In this section, we present the results of testing different settings for our scope resolver. We tested different feature sets, different label sets, different learners, and different parameters for our learner. When experimenting with these, we froze all other settings and changed only one at a time, so all tests with e.g. different feature sets are done with the same learner, model, label set and learner parameters. The only parameter that is not constant is the C-value, which was tuned because the feature set affects which C-value that yields the highest score. These basic settings are described in Table 7.3.2. All precision, recall and F1 values presented below are scope-level and token-level measurements without double penalty with gold standard cues, evaluated on the CDD dataset with the 2012 *SEM shared task evaluation script (see Section 4.4.1).

In order to get a reliable comparison of different feature settings, we start by identifying a basic set of lexical features from which we can sequentially add more complex features to check if they are informative to the learner. In Table 7.3.1, we show the results of these preliminary tests. Here, we start out by testing the performance of the system using only surface token and lemma features and then extend this feature set with simple cue features and lexical n-gram features. In the tables in this section, $\pm$ 1 refers to the token to the immediate left and right position of the current token. E.g. "lemma $\pm$ 1" refers to the forward/backward lemma bigrams as described in Section 6.2. Moreover, the cue features include the cue type, left and right cue distance and PoS-tag of the cue.

We will start by presenting the tests of different lexical features to find a set of features that will serve as our base for further experiments. The results for these tests are found in Table 7.3.1. We will then move on to present experiments of different PYSTRUCT learner classes in Section 7.3.1, followed by experiments of different hyperparameters in Section 7.3.2. After presenting this, we move on to the core part of this chapter, which is testing different feature sets and assessing the contribution of different features. This is presented in Section 7.3.3. Finally, we test different label sets in Section 7.3.4.

	Scope Level			Token Level		
Features	**Prec**	**Rec**	**F1**	**Prec**	**Rec**	**F1**
token, lemma, C = 0.50	95.83	13.69	23.96	72.41	48.66	58.21
token, lemma, cue-features, C = 0.50	100.0	51.79	68.24	86.41	79.75	82.95
token, lemma $\pm$ 1, cue-features, C = 0.30	100.0	56.55	72.25	88.26	79.23	83.50
token, lemma $\pm$ 2, cue-features, C = 0.07	100.0	52.38	68.75	87.91	76.63	81.88

Table 7.3.1: Preliminary experiments to find the best basic feature set for further experiments

From Table 7.3.1, we see that the system with the token, lemma, cue features and forward/backwards lemma bigrams obtain the best results both on scope-level and token-level with an F-score of 72.25% and 83.50%, respectively. An interesting thing to note about these preliminary tests is that the results show that omitting cue-dependent features will produce results that are worse than the baseline, which obtained a scope-level F-score of 32.03% (see Table 7.1.1). One would expect that using lexical features would at least match the baseline, but this is not the case. Another thing worth noting is that using forward and backwards trigram features actually produce worse results than using bigram features.

For all of the following experiments in this chapter, we have chosen a set of basic settings that are common for all tests, except e.g. learner for the experiment where different learners are tested. These basic settings are described in Table 7.3.2 and are based on preliminary tests of lexical features from Table 7.3.1 and test runs of different learner classes, label sets, and hyperparameters.

Parameter	**Configuration**
Model	`ChainCRF`
Learner	`FrankWolfeSSVM`
Label set	I, O, B, C
Random-state	10
Features	Token, lemma $\pm$ 1, cue-features
`max_iter`	10

Table 7.3.2: Basic configuration for the experiments with the scope classifier

7.3.1 Experiments with different learners

In these experiments, we recorded the elapsed time of the system in addition to P, R, and F-score. This is relevant here because as you can see in Table 7.3.3, there is a big difference in training time for the different learners, and having a short training time allows us to do many more tests of different feature sets and to fine-tune each system to get the best possible picture of how the systems perform relative to each other. If each

run took e.g. 45 minutes, we could never have tested the amount of different features that we have and would probably have to set a predefined C-value for each system instead of empirically finding the best one. The elapsed time was measured using Python's `time.clock` function. The learner has a big impact on training time because their complexity varies, and when we have 3644 training instances, this variation will make a noticeable difference in the actual elapsed time. The performance of each learner class on scope-level and token-level together with the elapsed training time is presented in Table 7.3.3.

	Scope Level			Token Level			
Learner	**Prec**	**Rec**	**F1**	**Prec**	**Rec**	**F1**	**Time(sec)**
`OneSlackSSVM`	100.0	15.48	26.81	71.97	48.96	58.28	38.70
`NSlackSSVM`	98.92	54.76	70.50	89.79	77.00	82.90	4225
`SubgradientSSVM`	100.0	50.60	67.20	87.44	76.93	81.85	106.2
`FrankWolfeSSVM`	**100.0**	**56.55**	**72.25**	88.26	79.23	83.50	65.22

Table 7.3.3: Tests of different learner algorithms

Table 7.3.3 shows that not only does the training and prediction time vary widely between the classifiers, but there is also a big span in how good they perform relative to the training time. All classifiers except for `OneSlackSSVM` significantly outperform the baseline, which has a scope-level F-score of 32%. It is clear that the best learner is the Frank Wolfe SSVM, which yields both the best performance in absolute value and the best results relative to training and prediction time, with an F-score of 72.25% on scope-level and 83.50% on token-level.

7.3.2 Experiments with different hyperparameters

We move on to test different hyperparameters that you can set in PYSTRUCT. Table 7.3.4 presents tests of different values for the regularisation parameter C. This parameter determines the margin of the optimal hyperplane in the SVM, see Section 3.2.1.

	Scope Level			Token Level		
C	**Prec**	**Rec**	**F1**	**Prec**	**Rec**	**F1**
10^{-3}	90.91	5.95	11.17	68.63	62.17	65.24
10^{-2}	98.46	38.10	54.94	77.61	77.89	77.75
10^{-1}	**100.0**	**54.17**	**70.27**	**88.37**	**78.34**	**83.05**
1	98.91	54.17	70.00	87.39	79.15	83.07
10	100.0	51.79	68.24	87.15	79.01	82.88
10^2	98.86	51.79	67.97	86.09	78.49	92.11
10^3	98.86	51.79	67.97	85.09	78.56	82.06

Table 7.3.4: Tests of different values for the regularisation parameter C

We can see from Table 7.3.4 that the regularisation parameter actually influences the performance of the classifier a lot. The classifier performs better with larger values for

C, which will produce a smaller margin. With the maximum at $C = 0.1$, it looks like values above this induce the same behaviour as with overfitting: The margin is tuned too much with the training data and is not able to generalise well enough, which in turn makes it misclassify more unseen instances. On the other hand, we see that very large margins, i.e. small values for C, also misclassifies more unseen examples.

The results of the tests of different values for the parameter `max_iter` are presented in Table 7.3.5. The `max_iter` parameter determines the number of iterations that the optimisation algorithm will run to find the optimal hyperplane during training.

	Scope Level			Token Level			
`max_iter`	**Prec**	**Rec**	**F1**	**Prec**	**Rec**	**F1**	**Time (sec)**
1	100.0	43.45	60.58	86.68	73.89	79.78	15.67
10	**100.0**	**56.55**	**72.25**	88.26	79.23	83.50	57.31
100	100.0	53.57	69.77	89.83	77.97	83.48	530.2
1000	98.94	55.36	71.00	89.95	77.00	82.97	4635

Table 7.3.5: Experiments with different values for max_iter

As we can see from Table 7.3.5, the performance does not increase from 10 iterations. This is probably because the algorithm converges towards the optimal solution after around ten iterations. In that case, running the algorithm longer does not increase performance, in fact, it may contribute to overfit the learner to the training data, which will decrease the prediction accuracy on unseen examples because it does not generalise as well. We see from Table 7.3.5 that iterating just one time obtains the lowest results with a scope-level F-score of just 60.58%. With 10 iterations, the F-score increases with 12 percentage points to 72.25%, from which it decreases again by 2.48 percentage points for 100 iterations. It actually increases from there to 1000 iterations, which obtain an F-score of 71.00%. However, with so many iterations, the training time is much longer than we can accept.

7.3.3 Experiments with different feature sets

Since the experiments with different values for the regularisation parameter C revealed a very significant peak, we decided that all experiments with different feature sets should be fine-tuned to find the optimal value of C for each feature set. The optimal value for C is expected to vary with the features that are applied because they affect how the feature space turns out and the size of the optimal margin. For each experiment, the precision, recall and F-score is reported with the C-value that gave the highest performance. To find the optimal C-value, we tested a range of values around the previously found peak at $C = 0.1$ (see Table 7.3.4), with a step-size of 0.05. If the optimal value was on the boundary, we tested a new set of values around the new optimum. After this, we tested a range of values around the new peak with a step-size of 0.01.

The preliminary tests, presented in Table 7.3.1, showed that among the simplest features, this set of features achieved the best result:

– token

- lemma
- forward and backward lemma bigram
- cue-features (left and right cue distance, PoS-tag of cue, cue-type)

The above set of features is what we will refer to as lexical features in the following experiments. In order to check the contribution of each additional feature individually, we check if adding one or more features in isolation to the lexical feature set will improve performance. We separate these experiments into two tables, one for part of speech features and one for dependency features. We only present the new features that we add to the baseline feature set in the following tables, but note that these lexical features themselves are also included in each system. Table 7.3.6 presents the results of adding features that depend on PoS-tags.

	Scope Level			Token Level		
Features	**Prec**	**Rec**	**F1**	**Prec**	**Rec**	**F1**
PoS-tag, C=0.07	100.0	55.36	71.27	89.08	78.04	83.20
PoS-tag, lexicalised C=0.07	100.0	57.74	73.21	88.22	77.74	82.65
PoS-tag $\pm$ 1 C=0.14	98.99	**58.33**	**73.41**	88.27	79.82	83.83
PoS-tag $\pm$ 1 Lexicalised PoS-tag C=0.07	100.0	57.74	73.21	88.40	77.74	82.65

Table 7.3.6: Experiments with PoS-features

Table 7.3.6 presents results of adding different PoS-tags to the lexical feature set and shows that adding the surface PoS-tag and the forward/backwards PoS-tag bigrams obtain the best result with a scope level F-score of 73.41%. Adding a lexicalised version of the surface PoS-tag obtain almost the same performance with only 0.20 percentage points difference. All configurations improve over the system with only lexical features. Looking at these results, we see that even though adding a lexicalised PoS-tag and a PoS-bigram alone did improve performance, combining them actually does not outperform the system where the bigram was used alone. This shows that indiscriminately adding more and more features might confuse the learner rather than helping it generalise.

In Table 7.3.7, the results of adding dependency features are presented.

Features	Scope Level Prec	Rec	F1	Token Level Prec	Rec	F1
PoS-tag of head, C = 0.15	100.0	55.36	71.27	87.24	77.60	82.14
Dependency relation, C = 0.20	100.0	54.76	70.77	88.02	79.60	83.60
PoS-tag of 2nd order head, C = 0.16	100.0	55.95	71.75	87.84	78.78	83.06
PoS-tag of head, PoS-tag of 2nd order head, Dependency relation, C = 0.21	98.95	55.95	71.48	86.32	79.60	82.82
Dependency graph path C = 0.20	99.01	59.52	74.35	90.22	80.04	84.83
Directed dependency distance, C = 0.12	100.0	58.33	73.68	90.36	79.97	84.85
Directed dependency distance, Dependency graph path, C = 0.13	**100.0**	**61.31**	**76.02**	90.05	81.23	85.41
PoS-tag of head, Dependency relation, Directed dependency distance, Dependency graph path, C = 0.08	100.0	60.71	75.55	91.20	80.71	85.63
PoS-tag of head, PoS-tag of 2nd order head, Dependency relation, Directed dependency distance, Dependency graph path, C = 0.05	100.0	60.12	75.09	90.63	80.34	85.18

Table 7.3.7: Experiments with dependency features

Table 7.3.7 shows that the system that achieves the best scope-level results is the system with directed dependency distance and dependency graph path added to the lexical feature set. However, the system with the head-PoS-tag, dependency relation, directed dependency distance and dependency graph path achieves the best token-level results. This is the only experiment where the maximum token-level and scope-level results differ. In this case, we will emphasise the scope-level results more than the token-level results. Furthermore, we see that only the systems with graph features improve over the basic system with only lexical features.

As we can see in Table 7.3.6 and Table 7.3.7, not all features are beneficial to the classifier on their own. However, some of them might be informative to the classifier in combination with others, and on the other hand, combining several of these features

might confuse the classifier even though the features alone have induced a performance gain. We want to investigate these effects by measuring the performance of different combinations of PoS-features and dependency features added to the baseline feature set. By doing this, we can see which combination of features that was most informative to the learner, and moreover if using e.g. dependency features in isolation, without PoS-features, is more informative than combining them and vice versa. The results of these experiments are presented in Table 7.3.8.

	Scope Level			**Token Level**		
Features	**Prec**	**Rec**	**F1**	**Prec**	**Rec**	**F1**
PoS-tag, lexicalised, PoS-tag of head, PoS-tag of 2nd order head, Dependency relation, C = 0.13	100.0	57.14	72.72	86.66	79.97	83.18
PoS-tag, lexicalised, Directed dependency distance, Dependency graph path, C = 0.15	100.0	62.50	76.92	90.96	82.12	86.31
PoS-tag $\pm$ 1, PoS-tag of head, PoS-tag of 2nd order head, Dependency relation, C = 0.20	98.95	55.95	71.48	86.23	79.90	82.94
PoS-tag $\pm$ 1, Directed dependency distance, Dependency graph path, C = 0.10	**100.0**	**63.10**	**77.38**	90.80	82.05	86.20
Pos-tag, Directed dependency distance, Dependency graph path, C = 0.07	100.0	63.10	77.38	91.64	81.31	86.17
PoS-tag $\pm$ 1, Lexicalised PoS-tag, PoS-tag of head, PoS-tag of 2nd order head, Dependency relation, Directed dependency distance, Dependency graph path, C = 0.05	100.0	58.93	74.16	89.80	81.01	85.18

Table 7.3.8: Experiments with different feature combinations

An interesting thing to note in Table 7.3.8 is that two different configurations obtain the same scope-level F-score, but different token-level F-scores. Since the token level scores are also very close, we want to test if there is a significant difference at token-

level. If this is the case, we will choose the configuration that has the best token-level score. If there is no significant difference, we will choose the simplest configuration to reduce the complexity of the feature space. To test this, we have used the McNemar test with $\alpha = 0.05$, see Section 4.4.2. This test gave a p-value of 0.519, which means that we can not reject the null hypothesis, hence there is no statistically significant difference between the two systems.

In addition to the experiments we have described in this section, we also tested the effect of discretising the features that take real values, e.g. the directed dependency distance. We did this by separating the distance into three classes, one for the tokens with a distance less than 3, one for a distance less than 7, and one for distances bigger than 7. These values were chosen empirically from tests of different limits. The benefit of using discrete values is that the dimension of the feature space is reduced, so the classifier can find connections between the instances of the classes more easily because more instances will take the same, discrete value. All distance features described in these experiments are discrete. Moreover, we also tested the effect of lexicalising the dependency relation, dependency graph path and several baseline features. However, most of these tests showed that lexicalising did not provide more information to the classifier, hence no performance gain, so we decided not to include lexicalisation. The exception was the surface PoS-tag, i.e. the PoS-tag of the current token, in which adding the lexicalised version did improve performance (see Table 7.3.6).

7.3.4 Experiments with different label sets

We move on to test different combinations of labels for each token in the scope sequence. The results of these experiments are shown in Table 7.3.9. In this table, 'I' is short for in-scope, 'O' means out-of-scope, 'B' means beginning-of-scope, 'C' means cue, and 'MC' means affixal (morphological) cue.

	Scope Level			Token Level		
Labels	**Prec**	**Rec**	**F1**	**Prec**	**Rec**	**F1**
I, O	98.21	32.74	49.11	80.14	76.63	78.35
I, O, B	98.48	38.69	55.55	80.06	76.56	78.27
I, O, C	98.77	47.62	64.26	84.63	78.86	81.64
I, O, B, C	**100.0**	**56.55**	**72.25**	88.26	79.23	83.50
I, O, B, C, MC	100.0	55.36	71.27	87.82	79.15	83.26

Table 7.3.9: Tests of different label combinations

The experiments with different label sets, which are presented in Table 7.3.9, show that the labels *I, O, B, C* obtain the best results. Adding a label for morphological cues was inspired by Lapponi (2012), who found that this label was informative to their classifier. Since the results of the two configurations in our experiments are fairly similar, we tested if there was any significant difference. With an $\alpha = 0.05$ and a p-value of 0.163, there is no significant difference between the two systems for our classifier.

7.4 Error analysis

As we can see from the tables in the previous section, the scope-level recall value for the scope resolver is fairly low. In the following section, we would like to investigate why this happens, and in particular, investigate if the system makes any special kinds of errors or if it makes the same mistakes on numerous instances. This error analysis is based on our best performing system, which obtained a scope-level F-score of 77.38% with basic lexical features, directed dependency distance and dependency graph path (see Table 7.3.8).

Table 7.4.1 shows examples of sentences from the development set where the predicted scope and the gold scope differ. When examining the sentences that our system predicted differently than the gold standard, it becomes quite apparent that our classifier struggles with discontinuous scopes, especially in somewhat complex sentences with subordinate clauses, and sentences with coordinations, like sentence 2, 3 and 4 in Table 7.4.1. From Table 7.4.1, we see that the classifier both labels parts of the sentence as in-scope when it is not, like in sentence 1, and vice versa, as in sentence 3. These types of sentences will be counted as a false negative because there is a partial match (see Section 4.4.1). Even though discontinuous scopes are a clear headache for the classifier, they do not account for the low recall value alone. Several previous systems had higher recall-values than we did and still misclassified all discontinuous scopes in the development data.

We also found other types of scopes that the classifier mispredicts, including some sentences with several cues where the gold-standard scopes overlap. Moreover, we observed that many of the sentences that are counted as false negatives on scope level often just have one token that is incorrectly labelled, which means that the scope overall is almost correct. This is reflected in the token-level F-score which is often 10% higher than the scope level F-score.

The evaluation also revealed that some of the systems predict false positive instances. This does not happen with the top-scoring settings, but it is worth noting since it is not immediately intuitive why a false positive can occur on scope level at all. Since partial matches are only counted as false negatives, these false positives must be the case of negation cues which have an empty scope, like the interjection *No* in "**No**, there was no attempt at robbery.".

	System	**Gold**
1	[The whole] **in**[explicable tangle seemed to straighten out before me].	[The whole] **in**[explicable tangle] seemed to straighten out before me.
2	...something which [he could] **not** [bear to part with], [had been left behind].	...something which [he could] **not** [bear to part with], had been left behind.
3	His friend and secretary, Mr. Lucas, [is] **un**[doubtedly a foreigner], chocolate brown, willy, suave, and catlike, with a poisonous gentleness of speech.	[His friend and secretary, Mr. Lucas, is] **un**[doubtedly a foreigner], chocolate brown, willy, suave, and catlike, with a poisonous gentleness of speech.
4	[It was] **not**, I must confess, a very alluring prospect.	[It was] **not**, I must confess, [a very alluring prospect].

Table 7.4.1: Examples of sentences where the system predicts a different scope sequence than the gold annotations.

To mitigate these errors, we could have continued to develop our features. For instance, we could have investigated the training data to find reasonable limits for the directed dependency distance instead of just making an educated guess. When doing this with the cue distance feature, our system was improved with 1.36 percentage points. Another option is to implement post-processing rules to deal with e.g. discontinuous scopes. However, we chose not to prioritise this as the performance of our system is already close to the best *SEM competition systems.

7.5 Comparison with *SEM 2012 competition systems

To get an impression of how well our system is doing compared to the systems that participated in the 2012 *SEM shared task, we would like to compare our results with other systems results on the development set. Precision, recall and F-score of previous systems are reported in 7.5.1. We see that the competition systems have a consistently higher recall value than our system, which in turn gives an F-score that is roughly 5% higher than ours.

System	Prec	Rec	F1
UiO_2	100.0	66.67	80.00
UiO_1	100.0	70.24	82.52
Our system	100.0	63.10	77.38

Table 7.5.1: CDD Scope-level results from previous systems for scope resolution

The most relevant comparison here is probably the UiO_2 system since this system also solves the scope resolution as a sequence labelling task and applies a CRF. It is hard to say why the UiO_2 system performs better than ours because we apply a lot of the same features and almost exactly the same label set. It might just be that applying a maximum likelihood CRF is better suited for this type of problem than a maximum margin CRF. It is also worth noting that even though the features and labels of our system are heavily inspired by the UiO_2 system, it is generally hard to replicate results, especially in NLP problems, as discussed by Fokkens et al. (2013). This is because researchers often do not document all aspects of their system, like all details of their preprocessing, how they represent their features, the hyperparameters they used and so on.

7.6 Conclusion

In this chapter, we have presented a series of experiments to examine which settings that are most beneficial to the scope classifier. Surprisingly, a setting with a relatively small set of features achieved the best results, using only the surface PoS-tag and dependency graph features in addition to the baseline feature set. This system outperformed systems with much larger feature sets and hence more information, but these were obviously not able to generalise as well as the simpler system. Moreover, adding a separate label for cues improved performance over the label set with *I, O, B* even though

the cue-label does not directly affect the prediction of in-scope and out-of-scope labels and is information that the classifier already has access to. But apparently, pairing this label with cue-dependent features helped the classifier predict the other labels more correctly. This is likely to be due to the fact that we are working with a discriminative model where the probability for each label is dependent on the previous labels.

The final configuration of our scope classifier is presented in Table 7.6.1.

Parameter	**Configuration**
Learner	`FrankWolfeSSVM`
Model	`ChainCRF`
Parameters	C = 0.07 max_iter = 10 random_state = 10
Features	Baseline features PoS-tag Directed dependency distance Dependency graph path
Labels	I, O, B, C

Table 7.6.1: Final configuration for the scope classifier

8 Final system configuration and results

In this chapter, we present held-out results on CDE for the cue detection classifier and the scope resolution classifier in isolation as well as end-to-end results for our final system on both the development set and the evaluation set. We also give details on the final implementation process, which consisted of enabling the system to run with both raw and parsed text, and sewing the system together so it can run end-to-end. Moreover, we give documentation on the full negation toolkit.

8.1 Final system configuration and implementation

Combining the cue classifier and scope classifier
In order to get end-to-end results and run the entire system as a tool, we need to integrate the cue-learner and the scope-learner. This is done by letting the cue-learner predict cues on a given data set, and then post-process the predicted cues to get a file on *SEM format, as described in Section 2.2.1. We can then give this file as input to the scope-learner and then predict scopes for the predicted cues.

Trained models
In our final toolkit, we want to provide a classifier that has already been trained – we do not wish to include the whole CD dataset in the final tool to train the classifier for every run. To make this work, we have stored all the objects we use, including the learner object from PYSTRUCT and the DictVectorizer object from SCIKIT-LEARN, in files using Python's `pickle` module, which can load and save Python objects.

Running the system
Recall from Section 4.1.3 that the program can be run in two modes: One where the user inputs a file with raw sentences separated by a blank line, and one where the user can input data which they have parsed themselves with a parser of their choice. If the user inputs raw text, we need to tokenise, tag and parse the text before we can classify the sentences. This is done with the CoreNLP tool[1] (Manning et al., 2014), which can perform both tokenisation, lemmatisation, tagging and dependency parsing.

Because our training data uses PTB PoS-tags and Stanford dependencies, we needed a tagger and a parser that use the same standard, and CoreNLP does that. It is also able to output the data in a CoNLL format which is almost identical to the CoNLL-X

[1] http://stanfordnlp.github.io/CoreNLP/

format, so the post-processing work needed before seeding the data file to the classifier is minimal. An example of how our negation tool is run in the two modes is listed below. If the user runs the toolkit in raw mode, they need to specify the directory of their CoreNLP program with the input option -d. This is an optional input parameter, but if it is not included in raw mode, the user will be asked to give the directory path.

- `python negtool.py -m parsed -f input_file.conll`
- `python negtool.py -m raw -f input_file.txt -d path/to/corenlp`

Python 2.7 or newer is required to run the toolkit. A full list of dependencies for our negation toolkit is listed below.

- SCIKIT-LEARN
- PYSTRUCT
- NUMPY
- NETWORKX

Note though that PYSTRUCT, SCIKIT-LEARN and NETWORKX have dependencies themselves which the user will need to install. If the packages are installed using e.g. Anaconda, these dependencies will be included in the installation. Moreover, if the user does not pre-parse the input data, they will need to install CORENLP.

Input/output format

When running the toolkit, the user can choose to provide pre-parsed data. The program expects the parsed data to be in the CoNLL-X format (Buchholz and Marsi, 2006), and it needs to have the following information encoded:

- Column 1: token index
- Column 2: token
- Column 3: lemma
- Column 5: PoS-tag
- Column 7: head index
- Column 8: dependency relation

The token indices need to start from 1 for each new sentence, and the head index needs to be either a token id, encoded in column 1, or 0, which only refers to the root. We do not make use of column 4 and 6, but in CoNLL-X, these columns encode coarse-grained PoS-tag and morphological/syntactic features, respectively. The columns must be separated by a tab.

The output of the classified sentences with predicted scopes and cues comes as a file, where the first 8 columns are identical to the (parsed) input file, and the columns containing cues and scopes are on the *SEM format described in Section 2.2.1.

8.2 Held out evaluation

In this section, we present final results for our cue classifier and scope resolver in isolation on the held-out evaluation set (CDE). For this experiment, we have trained our classifier on the entire CDTD, i.e. both CDT and CDD. However, the features and parameters for the classifiers are the same as described in Chapter 5 about cue classification and Chapter 6 about scope resolution.

The reason for doing experiments on the held-out evaluation set is that we want to see if our classifiers are truly able of generalising to unseen examples. Evaluating a machine learning model on the same data as it has been trained with has little value, because it is likely to be able to "remember" the training samples, meaning that with e.g. an SVM, the hyperplane that is used to classify data has been computed using the training data, so if the same data is used in evaluation, the instances will always fall on the right side of the hyperplane. With unseen data, however, you can see if the model is able to generalise, not just "remember" the training data.

We have chosen to compare the cue detection classifier mainly with the UiO_1 system, while the scope resolver is compared to the UiO_2 system. Note that these two systems in fact use the same cue classifier, so it does not matter whether we use UiO_1 or UiO_2 for comparison with our cue classifier. For the scope resolution task, our system is quite similar to the UiO_2 system, because we both solve the problem as a sequence labelling task, and we apply many of the same features. Therefore, it was reasonable to compare the performance of our system with UiO_2.

8.2.1 Cue detection

Our final cue classifier has the following configuration:

Parameter	**Configuration**
Model	`BinaryClf`
Learner	`NSlackSSVM`
Features	Token, Lemma $\pm$ 1, PoS
Affixal features	Character 5-grams, affix
C	0.20
`batch_size`	-1

Table 8.2.1: Final configuration for the cue classifier

It is this configuration that we will use in our toolkit and test the held-out evaluation set on.

Held out results for the cue detection system is presented in Table 8.2.2. The "Cues B" row refers to the B measure in the evaluation script (see Section 4.4.1), which calculates precision as TP/total of system predictions instead of as TP/(TP + FP). Our system obtained a total of 247 true positives, 27 false positives, and 17 false negatives.

Measure	Prec	Rec	F1
Baseline	87.10	92.05	89.51
Cues	90.15	93.56	91.82
Cues B	88.85	93.56	91.14

Table 8.2.2: Results for cue prediction on the held-out evaluation set

From Table 8.2.2, we see that our cue system performs nearly as good on the evaluation set as the development set, with a drop in F-score from 93.49% on CDD to 91.82% on CDE. Both recall and precision are lower for the evaluation set. Note, though, that the baseline actually performs better on the evaluation set than on the development set (89.51% on CDE versus 87.42% on CDD), so in principle, the performance-drop of our classifier is actually even bigger relative to the baseline. Although only slightly, our cue classifier outperforms the UiO_1 system, which we based our configuration on, on the evaluation set. This classifier achieved an F-score of 91.31%, 0.5 percentage points lower than ours. Our recall values are identical, but our precision value is almost 1 percentage point higher than theirs. Overall, our cue classifier would have placed third in the *SEM 2012 shared task.

8.2.2 Scope resolution

Our final scope classifier has the following configuration:

Parameter	Configuration
Model	`ChainCRF`
Learner	`FrankWolfeSSVM`
Features	Token, Lemma $\pm$ 1, PoS $\pm$ 1 Cue type, left/right cue distance, cue PoS Directed dependency distance, dependency graph path
Labels	I, O, B, C
C	0.10
`random_state`	10

Table 8.2.3: Final configuration for the scope classifier

Held-out results for the scope resolution system with gold cues are reported in Table 8.2.4. Our system achieved a total of 158 true positive scopes, 2 false positive scopes, and 91 false negative scopes. On token-level, our system achieved 1469 true positives, 137 false positives and 336 false negatives.

	Scope Level			Token Level		
Learner	Prec	Rec	F1	Prec	Rec	F1
Baseline	66.67	11.24	19.24	38.54	98.01	55.32
System	98.75	63.45	77.26	91.47	81.39	86.14

Table 8.2.4: Results for scope resolution on the held-out evaluation set with gold cues

We see from Table 8.2.4 that our scope resolution system performs almost identically on the evaluation set as the development set on scope-level, with only 0.12 percentage points decrease in F-score (from development to evaluation). On token-level, the system achieves a higher precision value than on development (0.67 percentage points in difference), and the token-level F-score decreases by only 0.06 percentage points from development to evaluation.

The baseline performs much worse on the evaluation set in terms of scope-level F-score, which decreased from 32.02% on CDD to 19.24% on CDE. This means that our classifier performs even better on the evaluation set relative to the baseline, because the baseline performs much worse, while our system is almost identical as with the development set. Unfortunately, we have no previous work to compare our held-out results to because all the *SEM competition systems report only end-to-end results on the held-out evaluation data, not isolated scope results with gold cues.

8.3 End to end results

In this section, we present the end-to-end results on both the development set and the held-out evaluation set. These results are obtained by first running the cue-detection system and then feed the data set with predicted cues as input to the scope learner so it can predict the scope. When we run the evaluation script (see Section 4.4.1) on the output file, we can obtain scores for both predicted cues and predicted scopes for these cues, which is what we mean by end-to-end results. The result of this is presented in Table 8.3.1.

Measure	**Prec**	**Rec**	**F1**
Cues	91.67	95.38	93.49
Scopes	88.14	61.90	72.73
Scope tokens	85.24	80.56	82.83

Table 8.3.1: End-to-end results for cue and scope prediction on the development set

From Table 8.3.1, we see that the scope-level F-score with predicted cues is 4.65 percentage points lower than the F-score that was achieved with gold standard cues. This drop is mostly due to precision, which was 100% for gold cues, now 88.14% with predicted cues. The reason for this is that the scope learner now predicts scopes for false positive cues, so the error from the cue classifier propagates to the scope classifier which now gets more false positive scopes.

End-to-end results on the held-out evaluation set are reported in Table 8.3.2. On scope level, our system achieved 153 true positives, 27 false positives and 96 false negatives. On token-level, we got 1449 true positives, 246 false positives, and 356 false negatives.

Measure	Prec	Rec	F1
Cues	90.15	93.56	91.82
Scopes	85.00	61.45	71.33
Scope tokens	85.49	80.28	82.80

Table 8.3.2: End-to-end results for cue and scope prediction on the held-out evaluation set

From the end-to-end results on the evaluation set, presented in Table 8.3.2, we see that the scope-level precision and token-level precision for scopes are much closer in value. The scope-level F-score decreased 1.5 percentage points from the end-to-end results on the development set to the evaluation set. This might be because the cue measurement is lower for the evaluation set and the increased error propagates to the scope measurements.

We see that the performance of our system is much closer to the UiO_2 system on the evaluation set than on the development set. The UiO_2 system achieves a scope-level end-to-end F-score of 72.39% on the held-out evaluation set, 1.06 percentage points higher than ours. Overall, our scope resolution system would have placed fourth in the *SEM 2012 shared task both in regards of scope-level and token-level measurements with predicted cues.

Even though our system places in the upper range in the 2012 *SEM shared task both for scope resolution and cue detection, new systems have been developed subsequently which should also be included in our comparison. The best performing scope resolution system that uses the CD dataset is as far as we know the system of Packard et al. (2014), which achieves a scope-level F-score of 78.7% on the evaluation set with gold cues, and a scope-level F-score of 73.1% with predicted cues. Even though this system does perform very well, it is worth noting that this is a complex system built upon deep linguistic analysis with many different dependencies. In order to run the system, one needs to parse the data with several different parsers, and the system uses the UiO_1 scope system with many heuristic extensions built on top. Because of this, it is practically impossible to run as a toolkit and only significant for research purposes. On the other hand, our system is fairly simple and intuitive, using only a few post-processing rules in addition to the machine learners, and achieves almost as high score as the Packard et al. (2014) system with only 1.44 percentage points in difference on scope-level F-score with gold cues, 2.06 percentage points in difference on token-level F-score with gold cues, and 1.77 percentage points lower on scope-level F-score with predicted cues. When our results are this close, it is safe to say that it is not worth the extra dependencies and complexity for such a low performance-gain.

Another system developed after the 2012 *SEM shared task is the system of Fancellu et al. (2016), which uses neural networks and word embeddings to solve the in-sentence scope of negation. While this is a very innovative approach to scope resolution, it is hard to say anything tangible about their results because in their summary of previous work, they have confused the scope results using gold cues with the ones using system cues. In particular, they have listed the scope-level F-scores of the systems from the 2012 *SEM shared task on system cues together with the Packard et al. (2014) system's scope-level F-score on gold cues, making it seem like this system performs much better than the others. Moreover, it is unclear from the article whether they have used gold

cues or system cues to measure the performance of their own scope resolution system, especially because they do not mention cue detection. If they had used system cues, it would be natural to describe how they obtained them. Because of this, we have decided not to compare our final results with this system.

9 Conclusion

In this work, we have developed an open source portable toolkit for automatic negation detection in natural language. The system is implemented in Python with PYSTRUCT, which is a library for structured prediction based on a maximum-margin approach. The system consists of two main parts, namely cue detection, which detects words that act as negation cues, such as *no*, *never*, *impossible*, and scope resolution, which finds the parts of the sentence that are affected by this negation cue.

An open source portable toolkit for negation is something that has been missing from the NLP community. While many research-based systems have been developed with the aim of performing as good as possible, they are often difficult to use in practice, with high complexity and many dependencies. Also, none of them are openly available. Our negation toolkit can be used for several purposes, including medical analysis and sentiment analysis, and because it is built with existing libraries that are actively maintained and easy to install, it should be straightforward to use. The source code is available online[1].

We have built the negation tool from scratch, and during development, we have based our system design on best practices from previous work, in particular systems from the 2012 *SEM shared task. When developing our cue classifier and scope classifier, we have emphasised a non-complex solution with only a few extra heuristics beyond the machine learning. Most of the work of developing this tool has been large-scale experiments with the cue classifier and scope resolver in isolation, both with different features, different labels, different classifiers and different hyperparameters.

The problem of detecting negation cues is solved as a disambiguation task, where each word that has previously been observed as a cue in the training data is disambiguated as a cue or non-cue. For this, we have used an SVM and made use of lexical and morphological features. We also implemented dedicated features for affixal cues, which are cues where the negation is an affix of the word, e.g ***im****possible* and ***un****solved*. These cues are quite special because the base that the affix attaches to is part of the meaning that is negated. Our final cue system outperforms the UiO_1 and UiO_2 systems which we have based our configuration on, with a precision of 90.15, a recall of 93.56 and an F-score of 91.82 on the held-out evaluation set. In the 2012 *SEM shared task, our cue detection system would have placed third overall.

We solved the scope resolution task as a sequence labelling task, heavily inspired by the UiO_2 system of Lapponi et al. (2012). In our experiments, we focused on finding the best feature set and label set. Moreover, we also fine-tuned the hyperparameters of the classifiers and tested different classifiers both from PYSTRUCT and SCIKIT-LEARN. Our scope classifier performs on par with the competition systems from the 2012 *SEM shared task, with a precision of 85.00, a recall of 61.45, and an F-score

[1] https://github.com/marenger/negtool

of 71.33 on scope-level with system cues on the held out evaluation set. Our scope resolution system would have placed fourth overall in the 2012 *SEM shared task.

While achieving the highest performance was never our main goal, the performance of our system is competitive with existing, often more complex systems. Moreover, the system can be executed with both raw and parsed input data, making it accessible to both users who have enough experience to parse the data themselves and possibly use state-of-the-art parsers, and users who does not want to deal with parsing and converting their data to the proper format. The system is easily portable to several tasks, such as speculation detection and sentiment analysis.

9.1 Improvements and future work

Our scope resolution system can easily be extended with post-processing heuristics for targeting e.g. discontinuous scopes. We did not implement this because of limited time, but it is definitely worth looking into, seeing that our scope learner struggled with discontinuous scopes. This was a source of error on scope-level for the UiO_2 system as well, which did not implement any post-processing rules either. The UiO_1 system implemented this, and while the rules themselves require some linguistic understanding, they seem to be fairly easy to implement. Another extension of the scope learning system is to utilise the phrase structure trees provided in the training data. Currently we convert these trees into dependency structures, but one might see some performance gain from combining dependency features with information from the phrase structure tree, though at the cost of increased complexity and additional system dependencies.

It would be interesting to see if making a separate classifier for affixal cues would be beneficial to performance. Since the problem of detecting single-word cues is very different in nature from detecting affixal cues, using two different classifiers is in theory a reasonable idea.

Recently, there has been some work on solving negation scope resolution by using neural networks, e.g. Fancellu et al. (2016). By using word embeddings and generating a synthetic data set from Simple English Wikipedia, they were able to generalise scope prediction across genres and making the system non-English specific.

Since our negation tool is open-source, anyone can fetch the code and extend it or change it in any way they see fit for their purposes. We have developed our tool in a quite abstract way, meaning that we have not hard-coded anything for negation, the models can be trained on any data on the CD format. For example, the cue lexicons are automatically generated from the training data rather than a manually compiled dictionary. Moreover, they are not sensitive to the specific task such as negation detection, as long as cue words are marked. This means that these cues can be uncertainty cues, negation cues or any other kind of cues.

This abstraction in our system enables anyone to train and predict e.g. speculation using the BioScope corpus, provided that they convert this corpus to the CD format. In turn, this raises many interesting research questions. In particular, it would be interesting to measure performance across multiple domains, such as negation detection versus sentiment analysis. Here, we can measure how well our system performs on BioScope when it is trained on CD and vice versa. Moreover, converting the BioScope corpus to

the CD format raises several questions about negation annotation itself. Not only do these datasets differ in terms of the format, but there are also differences in the annotation rules, such as which types of events that are annotated as negated. A common standard for negation annotation is something that could greatly benefit this research field, because it provides more freedom to use datasets from different domains.

Bibliography

Bird, S. (2006). NLTK: The Natural Language Toolkit. In *Proceedings of the COLING/ACL on Interactive Presentation sessions*, pages 69–72, Sydney, Australia.

Buchholz, S. and Marsi, E. (2006). CoNLL-X shared task on Multilingual Dependency Parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL 2006)*, pages 149–164, New York City, New York, USA.

Chapman, W. W., Bridewell, W., Hanbury, P., Cooper, G. F., and Buchanan, B. G. (2002). A Simple Algorithm for Identifying Negated Findings and Diseases in Discharge Summaries. *Journal of Biomedical Informatics*, 34:301–310.

Chen, D. and Manning, C. D. (2014). A Fast and Accurate Dependency Parser using Neural Networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Quatar.

Chowdhury, M. F. M. (2012). FBK: Exploiting Phrasal and Contextual Clues for Negation Scope Detection. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics (*SEM)*, Montreal, Canada.

Copestake, A., Flickinger, D., Sag, I. A., and Pollard, C. (2005). Minimal Recursion Semantics. An Introduction. *Research on Language and Computation*, 3:281–332.

Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik 1*, pages 269–271.

Fancellu, F., Lopez, A., and Weber, B. (2016). Neural Networks for Negation Scope Detection. In *Proceedings of The 54th Annual Meeting of the Association for Computational Linguistics (ACL 2016)*, pages 495–504, Berlin, Germany.

Farkas, R., Vincze, V., Mora, G., Csirik, J., and Szarvas, G. (2010). The CoNLL 2010 Shared Task: Learning to detect hedges and their scope in natural language text. In *Proceedings of the 14th Conference on Natural Language Learning*, pages 1–12, Uppsala, Sweden.

Flickinger, D. (2000). On building a more efficient grammar by exploiting types. *Natural Language Engineering (Special Issue on Efficient Processing with HPSG)*, 6:15–28.

Fokkens, A., van Erp, M., Postma, M., Pedersen, T., Vossen, P., and Freire, N. (2013). Offspring from Reproduction Problems: What Replication Failure Teaches Us. In *Proceedings of The 51st Annual Meeting of the Association for Computational Linguistics (ACL 2013)*, pages 1691–1701, Sofia, Bulgaria.

Gildea, D. and Jurafsky, D. (2002). Automatic labeling of semantic roles. *Computational Linguistics*, 28:245–288.

Hagberg, A. A., Schult, D. A., and Swart, P. J. (2008). Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conferences (SciPy 2008)*, pages 11–16, Pasadena, CA, USA.

Jurafsky, D. and Martin, J. H. (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics*. Prentice-Hall, 2nd edition.

Kim, J.-D., Ohta, T., Pyysalo, S., Kano, Y., and Tsujii, J. (2009). Overview of BioNLP'09 Shared Task on Event Extraction. In *Proceedings of the Workshop on BioNLP: Shared Task*, pages 1–9, Boulder, Colorado, USA.

Lacoste-Julien, S., Jaggi, M., Schmidt, M., and Pletscher, P. (2013). Block-Coordinate Frank-Wolfe Optimization for Structural SVMs. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pages 53–61, Atlanta, Georgia, USA.

Lai, C. and Bird, S. (2010). Querying linguistic trees. *Journal of Logic, Language and Information*, 19:53–73.

Lapponi, E. (2012). Why Not! : Sequence Labeling the Scope of Negation Using Dependency Features. *Master Thesis, University of Oslo*.

Lapponi, E., Velldal, E., Øvrelid, L., and Read, J. (2012). UiO_2: Sequence-Labeling Negation Using Dependency Features. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics (*SEM)*, Montreal, Canada.

Lavergne, T., Cappé, O., and Yvon, F. (2010). Practical very large scale CRFs. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL 2010)*, pages 504–513, Uppsala, Sweden.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., and McClosky, D. (2014). The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

Morante, R. and Blanco, E. (2012). *SEM 2012 Shared Task: Resolving the Scope and Focus of Negation. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics (*SEM)*, Montreal, Canada.

Morante, R. and Daelemans, W. (2009). A metalearning approach to processing the scope of negation. In *Proceesings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL)*, Boulder, Colorado, USA.

Morante, R. and Daelemans, W. (2012). ConanDoyle-neg: Annotation of negation in Conan Doyle stories. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics(*SEM)*, Montreal, Canada.

Morante, R. and Sporleder, C. (2012). Modality and Negation: An Introduction to the Special Issue. *Computational Linguistics*.

Morante, R., van Asch, V., and Daelemans, W. (2010). Memory-based resolution of in-sentence scope of hedge cues. In *Proceedings of the 14th Conference on Natural Language Learning*, pages 40–47, Uppsala, Sweden.

Müller, A. C. and Behnke, S. (2014). PyStruct - Learning Structured Prediction in Python. *Journal of Machine Learning Research*, 15:2055–2060.

Nowozin, S. and Lampert, C. H. (2011). Structured Learning and Prediction in Computer Vision. In *Foundations and Trends in Computer Graphics and Vision*, volume 6, pages 185–365, Hanover, MA, USA.

Packard, W., Bender, E. M., Read, J., Oepen, S., and Dridan, R. (2014). Simple Negation Scope Resolution through Deep Parsing: A Semantic Solution to a Semantic Problem. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014)*, pages 69–78, Baltimore, USA.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Édouard Duchesnay (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Read, J., Velldal, E., Øvrelid, L., and Oepen, S. (2012). UiO_1: Constituent-Based Discriminative Ranking for Negation Resolution. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics (*SEM)*, Montreal, Canada.

Savova, G. K., Masanz, J. J., Ogren, P. V., Zheng, J., Sohn, S., C-Kipper-Schüler, K., and Chute, C. G. (2010). Mayo clinical Text Analysis and Knowledge Extraction System (cTAKES): architecture, component evaluation and applications. *Journal of the American Medical Informatics Association*, 17:507–513.

Sutton, C. and McCallum, A. (2012). An Introduction to Conditional Random Fields. *Foundations and Trends in Machine Learning*, 4(4):267–373.

Tang, B., Wang, X., Wang, X., Yuan, B., and Fan, S. (2010). A cascade method for detecting hedges and their scope in natural language text. In *Proceedings of the 14th Conference on Natural Language Learning*, pages 13–17, Uppsala, Sweden.

Tottie, G. (1991). Negation in English Speech and Writing: A Study in Variation. *Academic Press*.

Vapnik, V. N. (1995). The Nature of Statistical Learning Theory. *Springer-Verlag*.

Velldal, E. (2011). Predicting speculation: a simple disambiguation approach to hedge detection in biomedical literature. *Journal of Biomedical Semantics*, 2.

Velldal, E., Øvrelid, L., and Oepen, S. (2010). Resolving speculation: MaxEnt cue classification and dependency-based scope rules. In *Proceedings of the 14th Conference on Natural Language Learning*, pages 48–55, Uppsala, Sweden.

Velldal, E., Øvrelid, L., Read, J., and Oepen, S. (2012). Speculation and negation: Rules, rankers and the role of syntax. *Computational Linguistics*, 38:369–410.

Vincze, V., Szarvas, G., Farkas, R., Móra, G., and Csirik, J. (2008). The BioScope corpus: biomedical texts annotated for uncertainty, negation and their scopes. *BMC Bioinformatics*, 9(11).

Vlachos, A. and Craven, M. (2010). Detecting speculative language using syntactic dependencies and logistic regression. In *Proceedings of the 14th Conference on Natural Language Learning*, pages 18–25, Uppsala, Sweden.

White, J. P. (2012). UWashington: Negation Resolution using Machine Learning Methods. In *Proceedings of the First Joint Conference on Lexical and Computational Semantics (*SEM)*, Montreal, Canada.

Zhou, H., Li, X., Huang, D., Li, Z., and Yang, Y. (2010). Exploiting MultiFeatures to Detect Hedges and Their Scope in Biomedical Texts. In *Proceedings of the 14th Conference on Natural Language Learning*, pages 56–63, Uppsala, Sweden.