Semantics Driven Anaphora Resolution

Håvar Skaugen

Masteroppgave i datalingvistikk og språkteknologi
Institutt for lingvistiske, litterære og estetiske studium
Universitetet i Bergen
2015

UNIVERSITETET I BERGEN

UNIVERSITETET I BERGEN

Institutt for lingvistiske, litterære og estetiske studium

DASP350

Masteroppgave i datalingvistikk og språkteknologi

Høstsemesteret 2015

Semantics Driven Anaphora Resolution

Håvar Skaugen

Abstract

This thesis describes a method for generating semantically motivated antecedent candidates for use in pronominal anaphora resolution. Predicate-argument structures are extracted from a large corpus of text parsed by the NorGram grammar and used as the basis for a fuzzy classification model. Given a pronominal anaphor, the model generates antecedent candidates ranked by the frequency by which they co-occur in the same lexical context as the anaphor. This set of candidates is intersected with the set of nouns gathered from the anaphor's recent context. A selection basic heuristics are then introduced to the model in a permutational fashion to gauge their individual and combined effect on the model's accuracy. The model reached an accuracy of 56.22% correct predictions. Additionally, in a slightly modified model the correct antecedent was found within the antecedent candidate list for 87.12% of the anaphora.

Sammendrag

I denne oppgaven beskriver jeg en metode for å generere semantisk motiverte antesedentkandidater til bruk i anaforoppløsning. Predikat-argument strukturer blir ekstrahert fra et stort korpus med tekst tagget med NorGram-grammatikken og brukt som basis i en "fuzzy" klassifikasjonsmodell. Modellen genererer antesedentkandidater for pronominelle anaforer rangert etter hvilken frekvens de forekommer i samme leksikale kontekst som anaforen. Et snitt blir foretatt mellom dette settet av kandidater og settet av substantiver i anaforens foregående kontekst. Et utvalg enkle heuristikker blir tilført modellen i forskjellige permutasjoner for å måle deres samlede og individuelle effekt på modellens treffsikkerhet. Modellen nådde en treffsikkerhet på 56.22% korrekte klassifiserte antesedenter. For en delvis modifisert versjon av modellen finnes den korrekte antesedenten blant antesedentkandidatene i 87.12% av tilfellene.

Acknowledgements

First, I whish to thank my supervisor, Professor Koenraad De Smedt, for his invaluable input and enlightening discussions along the way. Every meeting was an opportunity to hash out ideas, something which always delights me.

My parents deserve huge credit for instilling a love of knowledge in me and my brothers, and for always being supportive of my choices. The same goes for my two brothers, whose academic excellence in physics has inspired me to pursue academics myself, in addition to being

A special thanks to Professor Christer Johannson for inspiring me to pursue a degree in computational linguistics while I was still a bachelor student by inviting me along to a field trip to the Radboud university in Nijmegen.

I'm also grateful for my fellow masters students, Victoria and Anja, who managed to finish their degrees ahead of me. Without you, this degree would have been a solitary venture.

A thank you to Paul Meurer at Uni Research for helping me with downloading the resources from INESS, even going so far as to implement a new feature on my request.

Unni Eiken deserves thanks for laying out the groundwork for the method which I had the pleasure of exploring further.

Finally, I wish to thank all friends and fellow students who have made my soon to be 10 year stay here in Bergen near irreversible.

Contents

Abstract iii

Sammendrag v

Acknowledgements vii

1 Introduction 1
- 1.1 Introduction . . . 1
- 1.2 Selectional constraints . . . 1
- 1.3 Algorithms for anaphora resolution . . . 2
- 1.4 Real-world knowledge . . . 2
- 1.5 Parsing Norwegian . . . 3
- 1.6 Semantically motivated antecedent candidates . . . 3
- 1.7 Thesis outline . . . 3

2 Theory and method 5
- 2.1 Anaphora resolution . . . 5
  - 2.1.1 Morphosyntactic anaphora resolution algorithms . . . 5
  - 2.1.2 Morphological limitations . . . 6
- 2.2 Real-world knowledge . . . 6
  - 2.2.1 Elementary Predicate Argument Structures . . . 7
- 2.3 INESS . . . 8
- 2.4 Classification . . . 8
  - 2.4.1 Machine learning . . . 9

3 Data extraction and modelling 11
- 3.1 Material . . . 11
- 3.2 The Prolog file format . . . 11
- 3.3 Building a data structure . . . 12
  - 3.3.1 Serializing the data . . . 13
- 3.4 Parsing the files . . . 13
  - 3.4.1 Extracting EPAS . . . 14
  - 3.4.2 Extracting sentences . . . 16
- 3.5 Annotating antecedents . . . 17
- 3.6 Machine learning with TiMBL . . . 18
  - 3.6.1 TiMBL . . . 19
  - 3.6.2 Formatting data . . . 19
  - 3.6.3 Classification . . . 20
- 3.7 A fuzzy classification model . . . 22
  - 3.7.1 Considering the context . . . 22
  - 3.7.2 Building the model . . . 23
- 3.8 An ontology model . . . 24
  - 3.8.1 Grouping words . . . 24

3.9 Aggregating the two models . . . 26
3.10 Applying some heuristics . . . 27
3.10.1 Recency weighting . . . 27
3.10.2 Gender agreement . . . 27
3.10.3 Animacy . . . 28
3.10.4 Combining the constraints . . . 29
3.11 Adjusting the model . . . 31
3.12 Summary of the results . . . 31

4 Final remarks **35**
4.1 Conclusion . . . 35
4.2 Future work . . . 36

Bibliography **37**

A Listings **39**

B Source code **47**

C Additional data **65**

Chapter 1

Introduction

1.1 Introduction

A central problem in many Natural Language Processing tasks is that of reference resolution: "determining what entities are referred to by which linguistic expressions" [Jurafsky and Martin, 2009, p. 729]. Some kind of reference resolution is vital to any Natural Language Processing task that interprets discourse, examples of which include machine translation, automatic abstracting and information extraction.

For a concrete example in machine translation, consider translating the sentence in example (2) below to French. The third person plural pronoun *They* translates to either *Ils* or *Elles* depending on the gender of what *They* refers to. The referent in the sentence in example (1) translates to *anaphores*, a feminine word, yielding the the correct translation *Elles*.

(1) In natural language discourse **the most common type of anaphors** are pronomial.

(2) **They** take a noun phrase as antecedent.

Reference resolution is equally important for abstraction and abbreviation tasks. A hypothetical system that extracts all information pertaining to the word *anaphors* would need reference resolution to consider the information in the sentence in example (2) as relevant information.

1.2 Selectional constraints

The linguistic expressions that are performing reference are called *referring expressions*, and the entities they are referring to are called *referents*. There are several types of referring expressions available in natural language and five of them are presented in Jurafsky and Martin [2009]: *Indefinite Noun Phrases, Definite Noun Phrases, Pronouns, Demonstratives* and *Names*, and among these types the pronouns requires the strongest constraints on the possible referents.

Referring expressions that refer to an entity that has already been introduced in the text are denoted as *anaphors*, while their referents are called *antecedents*. The most common type of anaphors are pronominal, where the anaphor consists of a pronoun and the antecedent is a noun phrase. These anaphors place several constraints on the selection of their antecedent and some examples of these include: *Number agreement, Person agreement*, *Gender agreement* and *Recency* [Jurafsky and Martin, 2009]. Again, consider the sentences in examples (1) and (2).

The anaphora is highlighted in the sentence in example (2) and its antecedent is highlighted in the sentence in example (1). Note, however, that the sentence in example (1) contains two noun phrases, both possible candidates for being the antecedent. This is an example where number agreement can help you pick out the right noun phrase: *They* and *anaphors* are both plural, while *natural language discourse* is singular. In addition to the syntactical and morphological constraints mentioned above, collectively called morphosyntactic

constraints, Jurafsky and Martin [2009] propose constraints that make use of semantic information through *Verb semantics*.

1.3 Algorithms for anaphora resolution

There are three common algorithms for pronominal anaphora resolution presented in [Jurafsky and Martin, 2009, pp. 738-734]: the *Hobbs* algorithm, a *Centering* algorithm and a *log-linear* algorithm. All of these algorithms take as input a pronoun and the current and preceding sentences.

The Hobbs algorithm searches through a syntactic parse of the sentences to find noun phrases to propose as candidates for the antecedent. Starting at the pronominal anaphora, the algorithm uses a left-to-right breadth first traversal of the all the nodes marked as NP and checks to see if they are in agreement with regards to *gender*, *person* and *number*. The order in which the trees are traversed implicitly approximate the *binding theory*, *recency* and *grammatical role* constraints.

The Centering algorithm also requires a syntactic parse of the sentences containing the references to be resolved. Given two adjacent sentences this algorithm keeps track of all the entities mentioned in the first sentence in an ordered list based on a grammatical role hierarchy. For any pronoun encountered in the second sentence, all possible pairings of the pronoun and the entities from the first sentence are created. These pairs are then filtered by the constraints discussed in section 1.2 and ranked by the relation between the members of the pairs. The highest ranked pair is chosen as the resolution for the pronominal anaphora. By keeping track of all the entities and the relationships between them, the Centering algorithm achieves an explicit representation of a discourse model, something the Hobbs algorithm is incapable of.

The log-linear algorithm is a supervised machine learning approach that uses a training set consisting of a corpus where each pronoun has been linked to the proper antecedent. During the training phase the classifier classifies each preceding noun phrase according to different features. These features can include the constraints discussed in the previous section in addition to semantic constraints. The main advantage of this approach to anaphora resolution is that it does not require full syntactic parses of the sentences, unlike the Hobbs and Centering algorithms.

1.4 Real-world knowledge

In her master's thesis, Eiken [2005] presents the sentences in examples (3) and (4) where real-world semantic knowledge is needed to resolve the antecedents.

(3) The policeman shot at the **murderer** and **he** fell.

(4) **The policeman** shot at the murderer and **he** missed.

The sentences are morphologically and syntactically identical, and the only difference is the last verb. However, **he** refers to **murderer** in the first sentence, and **The policeman** in the second sentence. She then goes on to demonstrate that knowledge-free algorithms that does not incorporate real-world semantic knowledge are unable to correctly resolve these examples. The Hobbs and Centering algorithms discussed in the previous section both rely heavily on syntactic and morphological constraints, but have no way of representing semantic constraints. A supervised machine learning approach, however, can incorporate semantic knowledge as a classifying feature.

To be able to use semantic knowledge as a classifying feature a way to extract and represent the knowledge is needed. Lech and De Smedt [2005] argues that semantic classes can be extracted using noun phrase/verb co-occurrences. This is based on the *distributional hypothesis* that nouns that occur in similar contexts share a semantic similarity. Predicate-argument structures are derived from the verb-object-subject relation and the similarity of two noun phrases can be measured based on how many such structures they co-occur in.

Both Eiken [2005] and Lech and De Smedt [2005] have shown that ontologies extracted this way can aid in correctly classifying the antecedents for the type of sentences show in examples (3) and (4).

1.5 Parsing Norwegian

Mitkov identifies the low accuracy of the pre-processing tools that processes the input before feeding it to the anaphora resolution algorithms as one of the main problems facing anaphora resolution [Mitkov, 2001]. While this is a general problem for the whole field of anaphora resolution, the problem is compounded for resolving anaphora in the Norwegian language. Tools for parsing text are generally very language specific, and the Norwegian parsing tools have not been very reliable. Eiken's use of the NorGram grammar to extract predicate-argument structures proved to require a lot of manual work due to the poor accuracy of it's XLE implementation [Eiken, 2005]. Likewise, the poor accuracy of this deep parser spurred Lech and De Smedt to use the shallow Oslo-Bergen PoS Tagger for extracting their ontology [Lech and De Smedt, 2005]. Luckily, recent work done at the INESS project [Rosén et al., 2012] has greatly increased the accuracy of the XLE implementation of the NorGram lexical functional grammar.

1.6 Semantically motivated antecedent candidates

An integral part of anaphora resolution is to locate antecedent candidates for the anaphor. These candidates are then ranked according to morphosyntactic features, either intrinsicly as in Hobbs' algorithm [Hobbs, 1978], or by salience factors as described by Lappin and Leass [1994]. Recent work done in anaphora resolution algorithms for Norwegian by Nøklestad [2009] also focus on morphosyntactic features in a machine learning approach to the problem.

Recognizing the need for a semantic approach to anaphora resolution, Eiken [2005] explored a method for classifying antecedents based on 223 predicate-argument structures extracted from a small dataset. However, due to the fairly small number of predicate-argument structures and the amount of manual intervention required to construct it, Eiken concluded that a larger scale study was needed to conclude on the feasibility of the method.

Given the improvements to the NorGram grammar and the large collection of parsed texts in INESS, the time is opportune to do perform a larger scale study based on the exploratory work done by Eiken. In this thesis I will study how predicate-argument structures extracted from a large corpus parsed by a deep parser can aid in generating semantically motivated antecedent candidates for use in pronominal anaphora resolution. Additionally, I will design a system for automating all the necessary steps in the process.

1.7 Thesis outline

The thesis is structured as follows: In chapter 2 different approaches to anaphora resolution are discussed and the theoretical background for Eiken's work is presented. In chapter 3, the process of extracting the predicate-argument structures is described, as well as the process of creating an ontology model and a classification model. The results of the modelling is shown at the end of the chapter, in section 3.12. In chapter 4, there is a concluding discussion as well as some suggestions for future work.

Chapter 2

Theory and method

2.1 Anaphora resolution

Anaphora resolution is part of the larger problem of *reference resolution*. Within all natural language discourse there will be expressions that refer to other expressions or concepts. We call these types of expressions *referring expressions* and the entity they refer to *referents*. Anaphora are the subset of referring expressions where the referent has already been used in the discourse. In these cases the referent is called the *antecedent*.

There exists several types referring expression: *indefinite noun phrases, definite noun phrases, pronouns, demonstratives*, and *one-anaphora* [Jurafsky and Martin, 2009, p. 673]. Among these, the pronoun is the most common referring expression, and it is usually anaphoric in that it refers to an entity that has been previously introduced in the discourse. In some cases, however, pronouns can refer to entities introduced after them in the discourse, making them *cataphoric*. Pronominal anaphora are easy to locate in texts by simply marking any pronoun as an anaphor, though there are examples where pronouns are used non-anaphorically in fiction material [Nøklestad, 2009, p. 238].

The combination of these characteristics means that pronominal anaphora are more easily studied than other types of referring expressions, and are often used to prototype systems that can tackle a wider range of co-reference phenomena. As this thesis is meant as an exploratory study on the feasibility of using co-occurrence frequencies as a representation of real-world-knowledge, this project is no exception and will focus on pronominal anaphora resolution.

2.1.1 Morphosyntactic anaphora resolution algorithms

Traditionally, anaphora resolution systems have focused on examining the syntactic and morphological features of the anaphor and its antecedent. Once the anaphora have been located, the process of anaphora resolution typically consists of three steps:

1. Parsing the text. This is typically done with a syntactic parser.
2. Finding antecedent candidates. This is done by extracting noun phrases in the sentences leading up to and including the sentence where the anaphora occurs.
3. Ranking antecedent candidates based on different factors, either explicitly stated or implicitly represented in a model.

The factors in the third step can be reached through morphological and syntactic features, collectively called *morphosyntactic* features, and through semantic considerations. These features can include morphological features such as *number, gender, person* and *case* and syntactical features such as *recency* and *syntactic category*.

Like to the *Hobbs' algorithm* [Hobbs, 1978] outlined in section 1.3, the *Lappin and Leass'* algorithm Lappin and Leass [1994] is an algorithm that uses syntactic features. The algorithm builds a discourse model where

each noun phrase in the sentence with the anaphor and the preceding sentences are added. For each of these noun phrases, a salience score is calculated based on the salience factors shown in table 2.1 [Jurafsky and Martin, 2009, p. 685].

Table 2.1: Salience factors in Lappin and Leass's system

Sentence recency	100
Subject emphasis	80
Existential emphasis	70
Accusative (direct object) emphasis	50
Indirect object and oblique complement emphasis	40
Non-adverbial emphasis	50
Head noun emphasis	80

All the salience factors are based on the syntactic features of the noun phrase, and each factor that applies to the noun phrase is counted towards the total salience score. Additionally, selectional restrictions based on the morphological features *gender* and *number* are applied to disqualify unfit noun phrases from the discourse model.

2.1.2 Morphological limitations

Despite abundant use of morphosyntactic methods for anaphora resolution, there are several cases where they fall short. Consider the sentences in example (5).

(5) a. *En filosof er klar over at hun i grunnen vet svært lite.*
A philosopher is clear over that she basically knows very little.
'A philosopher is aware that she basically knows very little.'

b. *Nettopp derfor prøver hun igjen og igjen å oppnå virkelig innsikt.*
Exactly therefore tries she again and again to attain real insight.
'That's exactly why she again and again tries to attain real insight.'

Norwegian nouns have, as opposed to English nouns, a grammatical gender, and the grammatical gender of *filosof* is masculine. An algorithm that relies on the gender agreement selectional constraint would fail to select *filosof* as the antecedent of *hun* because they disagree on gender. Likewise, the number agreement selectional constraint can fail. Consider the sentence in example (6) where the antecedent for the plural pronoun *they* is the singular noun *team*. This violates the number agreement constraint and excludes the correct antecedent from the list of antecedent candidates.

(6) The team won the match because they scored many goals.

2.2 Real-world knowledge

Eiken [2005] demonstrated that both the *Lappin and Leass* and *Hobbs* algorithms applied to the sentences in example (7) will decide upon **Lensmannen** as the antecedent for both examples. This is in disagreement with an intuitive reading of the sentences which interprets **gjerningsmannen** as the antecedent in example (7b).

(7) a. **Lensmannen** som leder etterforskningen, sier at gjerningsmannen trolig kommer til å drepe igjen. **Han** etterlyser vitner som var i sentrum søndag kveld.
The sergeant *leading the investigation says that the perpetrator probably will kill again.* ***He*** *puts out a call for witnesses who were in the city centre Sunday evening.*

b. Lensmannen som leder etterforskningen, sier at **gjerningsmannen** trolig kommer til å drepe igjen. **Han** er observert i sentrum.
*The sergeant leading the investigation says that **the perpetrator** probably will kill again. **He** is observed in the city centre.*

The reason for the shortcomings of the algorithms is that there are no syntactical differences between the anaphor in example (7a) and the anaphor in example (7b). The only way to distinguish between the two is to consider real-world knowledge about their differences; Who is more likely to put out a call for witnesses and who is more likely to be observed in the city centre? We can reach the conclusion that the police sergeant is more likely to call for witnesses based on previous experience with police sergeants. Likewise, we can conclude that a perpetrator is more likely to be observed based on prior knowledge. As language users, this sort of real-world knowledge comes naturally, but for a computational system such knowledge poses a dual problem:

1. How can real-world knowledge be collected?
2. How can real-world knowledge be represented?

Traditional approaches to these questions have often included manual hand-coded knowledge bases that strives to represent general semantic concepts. Such efforts include the Precondition/Postcondition constraints proposed by Carbonell and Brown [1988] which for example can determine that after an act of *giving* has been carried out, the object that was given can no longer be in the possession of the one who carried out the act. The limitations of such a technique, however, is noted by the ones who proposed the technique themselves: “The strategy is simple, but requires a fairly large amount of knowledge to be useful for a broad range of cases”.

An alternative approach that has seen traction in recent years is the use of big data sets in combination with machine learning algorithms. For example, Modjeska et al. [2003] use search queries in Google to gather semantic knowledge about other-anaphora, type of referential noun phrases with the modifiers *other* or *another*. The representation is handled by a Naive Bayes classifier. Ponzetto and Strube [2006] extracted data from Wikipedia to be used as semantic relatedness measures between words. These measures were used as features in a Maximum Entropy learning model. These efforts suggest that a collection of large amounts of data can serve as a representation of real-world knowledge.

A similar method can be used to gather intuitions about whether the police sergeant or perpetrator from example (7) took part in a certain action. The Distributional Hypothesis as proposed by Harris [1968], suggests that the semantic meaning of words can be inferred from the context in which they occur. Building on this hypothesis, Hindle [1990] showed that predicate-argument structures extracted from a corpus can be used to classify the semantic similarity of nouns. This is based on the idea that there is a restricted set of verbs that a noun can appear as a subject or object to. Going back to our sentences in example (7), there is a restricted set of verbs that the perpetrator from example (7b) can appear as a subject to, and the Norwegian verb *etterlyse* is likely not among them. Similarly, it is more likely that a perpetrator appears as an object to the verb *observed* than that a police sergeant does.

This is the basis for the method that Eiken [2005] used to represent real-world knowledge about the similarity between anaphora and antecedents. By extracting predicate-argument structures from the corpus, the semantic similarity between the noun and a pronoun can be quantified using a supervised machine learning algorithm. The same approach I will be used in this project.

2.2.1 Elementary Predicate Argument Structures

The predicate-argument structures used by Eiken [2005] differ from the structures used by Hindle [1990] in one respect. While Hindle extracted his structures from the subjects and objects of verbs, Eiken extracted *verbal predicates*. The difference between the two is that a verb-subject-object structures are syntactically defined, while verbal predicates are semantically defined. This has the advantage that different linguistic constructs that give rise to the same meaning are represented alike. Active and passive constructs, while

syntactically different, can have the same semantic content, and the verbal predicate preserves this equality in the semantic representation. To emphasize the difference between the two structures, Eiken coined the term *Elementary Predicate-Argument Structure* or *EPAS*.

The extraction of EPAS is made possible by the f-structures produced by the NorGram-grammar as provided by INESS, as discussed in the following section.

2.3 INESS

INESS, the Norwegian Infrastructure for the Exploration of Syntax and Semantics, is a collection of tree-banks of syntactically and semantically parsed corpora. The project is partially devoted to developing a large treebank for Norwegian using the computational NorGram-grammar [Rosén et al., 2012]. The NorGram-grammar is part of the international Parallel Grammar Project (ParGram), based on the Lexical Functional Grammar formalism. The grammar is implemented using the XLE parser which allows fragmented parses for anomalous input [Rosén et al., 2005]. This last feature is of significance to this project. While traditional anaphora resolution algorithms typically require full syntactic tree parses to work, fragmented trees should be sufficient for extracting EPAS as long as the predicates are represented. Furthermore, the XLE parser also supports a stochastic disambiguator that returns the top ranked parse based on previously parsed sentences. The combination of these two features makes the Norwegian treebanks in INESS suitable for extracting a large number of EPAS.

Each sentence parsed using the NorGram-grammar in INESS has representations in a c-structure (a phrase structure tree) and a f-structure (an attribute-value graph). The features in the f-structure are based on the naming conventions in the ParGram-grammar [Butt et al., 2002], thus making the extraction process described in chapter 3 suitable for texts parsed with other ParGram-grammars as well.

The INESS web interface allows you to visualize both the c-structure and f-structure of the parsed sentences. Consider the sentence in example (8).

(8) *Det første stykket hadde hun gått sammen med Jorunn.*
The first part had she walked together with Jorunn.
'The first part she had walked together with Jorunn'

The corresponding f-structure as seen on the web interface is presented in figure 2.1. We can clearly see two binary predicates in the f-structure: *ha-perf* and, *gå*. Both of them have corresponding arguments: *gå* and *hun*, and *hun* and *stykke* respectively. This gives us the EPAS in examples (9) and (10).

(9) *ha-perf, gå, hun*
have-perfectum, go, she

(10) *gå, hun, stykke*
go, she, part

2.4 Classification

In order to use the extracted EPAS to model the similarities between anaphora and antecedents, a classification using machine learning has to be performed. Given the large corpus used in this project, I will only classify anaphora that appear as argument 1 in the EPAS, but the same method is equally valid for classifying anaphora that appear as argument 2 as well.

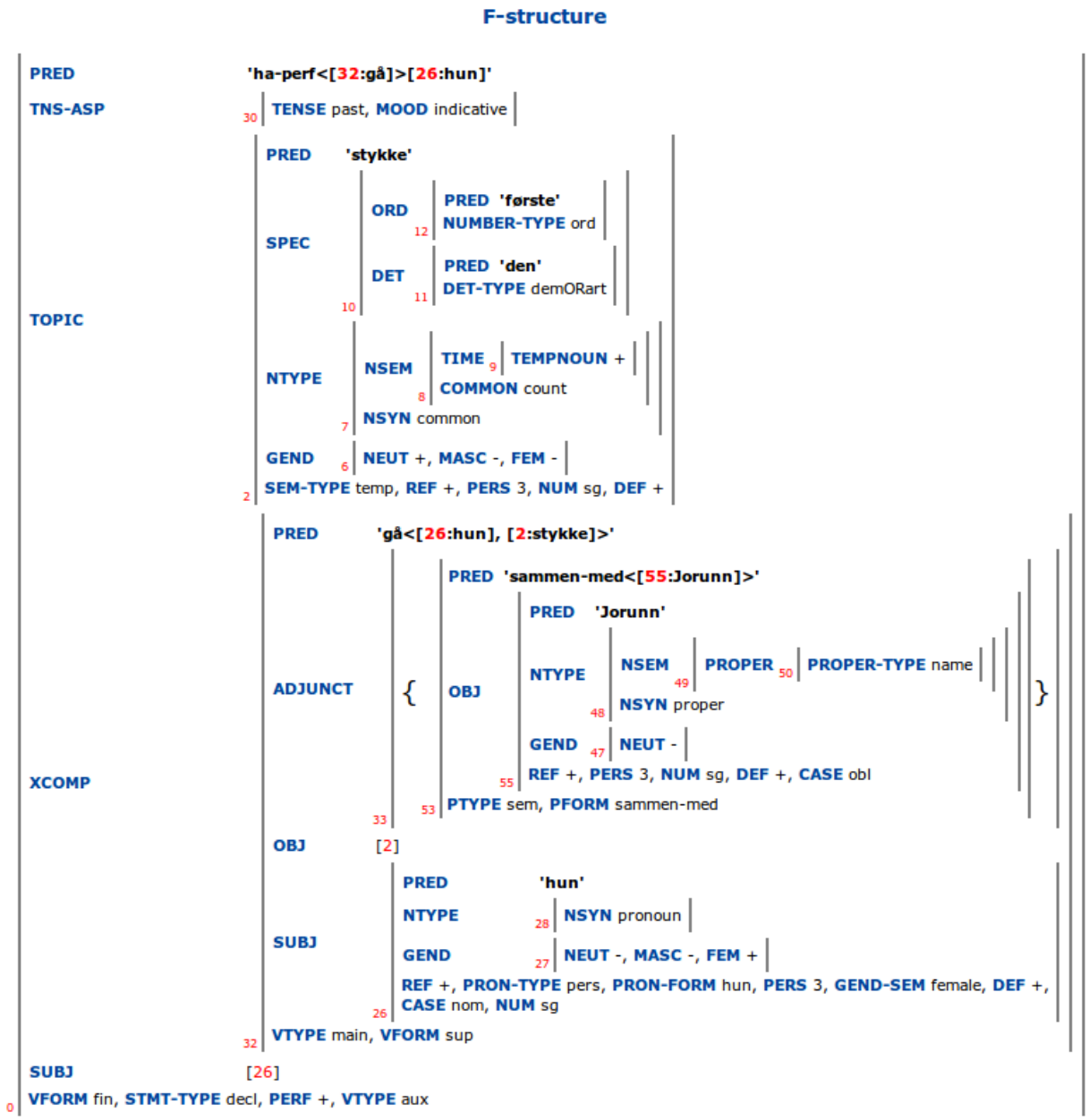


Figure 2.1: F-structure of sentence in example (8)

2.4.1 Machine learning

Tom M. Mitchell [2006] describes the machine learning as computer systems that automatically improves with experience:

> To be more precise, we say that a machine learns with respect to a particular task T, performance metric P, and type of experience E, if the system reliably improves its performance P at task T, following experience E. Depending on how we specify T, P, and E, the learning task might also be called by names such as data mining, autonomous discovery, database updating, programming by example, etc.

A common way to specify T, P and E is that of a supervised classification algorithm where T is the task of classification, P is how accurate the classification is and E is a set of labelled examples. More specifically, the task is to approximate an unknown function $f : X \to Y$ where the labelled examples is the set $\{\langle x_i, y_i \rangle\}$ of inputs x_i and outputs $y_i = f(x_i)$. Anaphora resolution can be seen as a such a classification task, where the function f has to classify an antecedent Y based on the anaphor and its context X.

The labelled examples in this case will be the list of extracted EPAS. As we are interested in classifying the correct noun for a given anaphor, the label for each example will be the noun that appears as argument 1 in the EPAS, while the rest of the elementary predicate-argument structure, the verb and argument 2, will form the example.

Chapter 3

Data extraction and modelling

3.1 Material

One key requirement identified by Eiken [2005] when selecting texts from which the EPAS are to be extracted, is that they belong to the same thematic domain. In addition to this, Eiken proposed a set of criteria for text selection:

- Relatively long chains of discourse
- Fairly high occurrence of anaphora, pronouns in particular
- Several paragraphs where the same phenomenon is discussed
- Low occurrence of tables and illustrations, ideally all the information in the texts should be expressed in complete and grammatical sentences

Eiken found that while prose texts were easy to confine to one thematic domain, they failed to fulfil the other criteria. For the purposes of this project, I propose two additional criteria: The text collection should be fairly large as to produce a big data set of EPAS, and it should be easily parsed by the INESS parser. Looking at the collection of parsed texts in INESS, one corpus stands out:

Sofies Verden [Gaarder, 1991] is a famous Norwegian novel by Jostein Gaarder that follows the protagonist, *Sofie*, on a journey to explore the history of philosophy. This provides for a fairly confined thematic domain concerning philosophy, and given that it is a full novel, the other criteria are also fulfilled with long chains of discourse with discussions of coherent phenomena. The number of personal pronouns occurring in the novel is shown in table 3.1.

Table 3.1: Occurrences of personal pronouns

hun	henne	han	ham	Total
1780	278	1940	189	4187

The novel was generously provided by Gaarder to the INESS project for use in language technology research and development [Gaarder and Rosén, 2011]. The first chapter is manually disambiguated and serves as a gold standard corpus, while the rest of the novel is automatically disambiguated by the stochastic disambiguator. The full novel consists of 14 526 sentences out of which 13 529 of them were subject to a full parsing.

3.2 The Prolog file format

Each of the selected sentences in INESS can be exported as a Prolog file, a plain text file with utf-8 encoding the format of which is described in the XLE documentation [Crouch et al., 2008]. An extract of the f-structure from figure 2.1 can be seen represented in Prolog in listing 3.1 (the whole file can be seen in listing A.1).

Listing 3.1: Extract of a Prolog file format from INESS

```
cf(1,eq(attr(var(0),'PRED'),var(15))),
cf(1,eq(attr(var(0),'SUBJ'),var(26))),
cf(1,eq(attr(var(0),'XCOMP'),var(32))),
cf(1,eq(attr(var(0),'TOPIC'),var(2))),
cf(1,eq(attr(var(0),'CHECK'),var(13))),
cf(1,eq(attr(var(0),'TNS-ASP'),var(30))),
cf(1,eq(attr(var(0),'VTYPE'),var(31))),
cf(1,eq(attr(var(0),'PERF'),'+')),
cf(1,eq(attr(var(0),'STMT-TYPE'),'decl')),
cf(1,eq(attr(var(0),'VFORM'),'fin')),
cf(1,eq(var(15),semform('ha-perf',55,[var(32)],[var(26)]))),
cf(1,eq(attr(var(26),'PRED'),semform('hun',60,[],[]))),
cf(1,eq(attr(var(26),'GEND'),var(27))),
cf(1,eq(attr(var(26),'NTYPE'),var(28))),
cf(1,eq(attr(var(26),'NUM'),var(29))),
cf(1,eq(attr(var(26),'CASE'),'nom')),
cf(1,eq(attr(var(26),'DEF'),'+')),
cf(1,eq(attr(var(26),'GEND-SEM'),'female')),
cf(1,eq(attr(var(26),'PERS'),'3')),
cf(1,eq(attr(var(26),'PRON-FORM'),'hun')),
cf(1,eq(attr(var(26),'PRON-TYPE'),'pers')),
cf(1,eq(attr(var(26),'REF'),'+')),
cf(1,eq(attr(var(27),'FEM'),'+')),
cf(1,eq(attr(var(27),'MASC'),'-')),
cf(1,eq(attr(var(27),'NEUT'),'-')),
cf(1,eq(attr(var(28),'NSYN'),'pronoun')),
cf(1,eq(var(29),'sg')),
cf(1,eq(attr(var(32),'PRED'),var(61))),
cf(1,eq(attr(var(32),'SUBJ'),var(26))),
cf(1,eq(attr(var(32),'OBJ'),var(2))),
cf(1,eq(attr(var(32),'ADJUNCT'),var(33))),
cf(1,eq(attr(var(32),'CHECK'),var(59))),
cf(1,eq(attr(var(32),'VFORM'),'sup')),
cf(1,eq(attr(var(32),'VTYPE'),'main')),
cf(1,eq(var(61),semform('gå',62,[var(64),var(77)],[]))),
```

This representation consists of a set of equivalences between variables and features distributed throughout the whole file. Extracting the predicate in example (9) is not as straight forward as it seemed from the f-structure in figure 2.1. For example, you have to traverse the equivalence between `var(0)` and `var(15)` to get to the word representation of the predicate, while other relevant features, such as *VFORM*, are connected to `var(0)`. I have developed a method for extracting EPAS from the Prolog files using a program coded in Perl, outlined in section 3.4. Note that the filename extension for both the Prolog files and the Perl scripts is the same (*.pl).

3.3 Building a data structure

While the simple text-only structure of the EPAS is sufficient for training the machine learning algorithm in section 3.6, there are several reasons to design a more complex data structure for this project:

- The f-structure contains part of speech information which is useful for sorting and organizing the EPAS for the different experiments.
- It is important to be able to keep track of which sentences the EPAS were extracted from when evaluating the classifications and when annotating antecedents.
- A way to represent sentences and the lemmas they contain.
- A way to represent antecedents in relation to their anaphora.

A robust data structure also has the benefit that the suite of programs programmed for this project are easy to maintain and expand. Furthermore, it makes it easier to put the data gathered for this project to other uses.

There is no practical way to represent these complex requirements using just the list and hash data types available in Perl. The programming language, however, also has Object Oriented capabilities, and using

these I created three classes to represent the data. The source code for the classes is shown in appendix B. For an overview of the classes and their attributes, see table 3.2.

Lemma.pm This class represents the lemmas that make up the predicates, sentences and antecedents. The lemma itself is stored in the *semform* attribute and information about part of speech is stored in the *attribute* attribute. A reference to the sentence that the lemma appeared in is stored in the *sentenceID* attribute and the variable of the lemma from the f-structure is stored in the *var* attribute. The *frequency*, *distance* and *score* attributes all store information pertaining to the selectional constraints used in section 3.10. The *similars* attribute stores the similar words that are extracted in section 3.8.

Predicate.pm This class represents the EPAS. The lemma of the predicate is stored in the *pred* attribute as plain text, while the arguments of the EPAS are stored as Lemma-objects in the *arg1* and *arg2* attributes. As with the Lemma-class, a reference to the sentence that the predicate appeared in is stored in the *sentenceID* attribute. A reference to a possible antecedent can be stored as a Lemma in the *antecedent* attribute. The index attribute is used in the annotation program described in section 3.5.

Sentence.pm This class represents a sentence. The surface form of the sentence is stored in the *sentence* attribute and the sentence's identity number is stored in the *id* attribute. A list of all the lemmas that make up the sentence is stored in an array of Lemma objects in the *lemmas* attribute.

It is worth noting that both the arguments in the *Predicate* class, the similars in the *Lemma* class and the lemmas in the *Sentence* class are stored as instances of the *Lemma* class, making this a recursive data structure.

Table 3.2: Overview of the data structure

Lemma	Predicate	Sentence
semform: string	pred: string	sentence: string
attribute: string	arg1: Lemma	id: int
var: int	arg2: Lemma	lemmas: array(Lemma)
sentenceID: int	sentenceID: int	
similars: array(Lemma)	antecedent: Lemma	
frequency: int	index: int	
distance: int		
score: float		

3.3.1 Serializing the data

Using a complex data structure necessitates the need to serialize the data before storing it. Serialization enables persistence of the data structure used to represent the data by storing a reference to the classes together with the data. One such example is the *!!perl/hash:My::Predicate* statement in listing 3.4. Perl offers several options for saving the data in a native Perl data format such as *Storable* and *Data::Dumper*. Another option is the YAML data format, a "human-friendly, cross language, Unicode based data serialization language designed around the common native data types of agile programming languages" Ben-Kiki et al. [2009]. This format has the major advantage that it is language independent and has implementations in many popular programming languages, thus ensuring that the data from this project can be used for other projects. Additionally, the human-friendly mark-up of this format serves to clearly illustrate instances of the objects presented in this section. For examples of this, see listings 3.4 , 3.6 and 3.8.

3.4 Parsing the files

Extracting the data from the Prolog files is done in two steps using the two Perl scripts outlined in sections 3.4.1 and 3.4.2. The Prolog files, one file per sentence, are downloaded in bulk as a compressed archive from the INESS web interface once a document is selected. Several download modes are available:

- Sentence
- Bracketed sentence
- Labeled bracketed s.
- Prolog
- Prolog (disamb.)
- Prolog (highest ranked.)
- Tiger-XML

Among these modes, it is the Prolog modes that offer the full f-structure of the sentences. The *Prolog* and *Prolog (disamb.)* modes gives you all the possible parses of the c- and f-structures, while the *Prolog (highest ranked.)* mode only gives you the highest ranked parse according to the stochastic disambiguator. This last mode was kindly added to the INESS interface upon my request, as this will provide the smallest file size and consequently enable faster parsing. Using this last mode, an archive containing 13 529 Prolog files is downloaded. Each of the files are labelled with a document identification code and a sentence number in the file name. Both of the extraction scripts described in the following sections takes an entire folder as input and parses each of the files in that folder, but I will use a single Prolog file as an example when describing the process. This file can be seen in listing A.1.

A small note on terminology: When talking about variables in this section, I refer to the integers contained in a `var()` structure. When referring to lemmas, I mean any string of text enclosed within apostrophes, as any word in the f-structure is lemmatized.

The Perl language

The brunt of the work for this project involves interacting with a large number of files and an extensive use of pattern matching within them. These are both areas at which Perl excels. Perl was first released in 1987 and built specifically to be a replacement for UNIX shell utilities such as *awk*. It offers powerful built-in facilities for pattern matching in files on a line per line basis, and "it is unsurpassed at this" [Raymond, 2003, Chapter 14]. These pattern matching facilities come in the form of excellent support for Regular Expressions. Reading and writing files and directories is also fairly easy.

3.4.1 Extracting EPAS

Extraction of the EPAS is done using the script *predicateExtractor.pl* taking two input arguments and outputting one YAML file. The source code is shown in listing B.4. The first input argument is the path to the folder containing the Prolog files discussed in section 3.2, and the second argument is the path to where you want to save the output file. The output is in the form of a YAML file in addition to the diagnostic output printed to the terminal. An example execution of the script is shown in listing 3.2.

Listing 3.2: Example execution of *predicateExtractor.pl*

```
$ perl predicateExtractor.pl nob-sofie-hele/ preds.yaml
```

First off, the script saves all the file names in the directory specified in the first argument in an array. For each of the file names, the corresponding file is loaded and the sentence number is extracted. In our example case, the file name is *oai:bibsys.no:biblio:932407552-5-hr.pl* which can be divided into three parts using the dash as the delimiter. The first part is the document identification, the second part is the sentence number, and the last part is the download mode, in this case *hr* for *Prolog (Highest ranked.)*.

The basis for each EPAS is the verb, so the first order of business is to locate all the lines the file that corresponds to a verb. To do this, the script searches for patterns that match the *VFORM* feature and returns the corresponding variable. All the verbs correspond to a predicate in the f-structure and these can be found by matching lines where the variables found in the previous step appear alongside the *PRED* feature. These

lines will either contain a variable pointing to where the predicate and its arguments can be found, in which case this variable is stored, or the predicate and corresponding arguments itself, in which case the variable from the previous step is stored. Considering our example file in listing A.1, see table 3.3 for an overview of the variables and the line numbers they were extracted from.

Table 3.3: Variables and their corresponding lines

	Verbs			Predicates		
Variable:	0	32	73	15	61	61
Line:	34	57	164	25	52	159

For each of the variables found in the previous step the script finds the lemma and the variables for the arguments. The lemma is found by matching the string contained in the *semform* feature, while the variables representing the arguments are found by matching the following variables. Matching the variables can be a bit tricky as their order can vary between lines and between files as you can see in the two predicates in the example file in listing A.1. In the first predicate, the variables are contained within one set of brackets, while in the second they are contained within separate brackets, as shown in listing 3.3.

Listing 3.3: Lines 35 and 59 from the file in A.1

```
cf(1,eq(var(15),semform('ha-perf',55,[var(32)],[var(26)]))),
cf(1,eq(var(61),semform('gå',62,[var(64),var(77)],[]))),
```

At this point an object of type *Predicate* is initiated, the sentence number passed as the *sentenceID* attribute of the object, and the lemma is passed as the *pred* attribute of the object. Still following our example file, see table 3.4 for an overview of the variables and line numbers.

Table 3.4: Variables and their corresponding lines

Line number	Lemma	arg1	arg2
35	ha-perf	32	26
59	gå	64	77

The next step is to find the lemmas and part-of-speech information for the arguments which requires a somewhat convoluted recursive subroutine that takes a variable as input. At first, a *Lemma* object is created and each line in the file is checked to see if it matches the pattern of the given variable occurring alongside any of the following features: *PRON-FORM, VFORM* and *NTYPE*, corresponding to a *pronoun, verb* and *noun* respectively. Additionally, if the *NTYPE* is matched, the variable contained within this features is followed to see if it is a normal noun or a personal noun. Upon matching any of these features, the *attribute* attribute of the *Lemma* object is set accordingly. The lemma of the arguments can be reached through to three separate structures:

1. A predicate containing the *semform* feature
2. A predicate only containing a variable
3. An equality statement between two variables

If the first structure is matched, the base case of the recursion is reached and a *Lemma* object is returned where the *semform* attribute is set to the string matched in the *semform* feature. Additionally, if the lemma equals any of the singular personal pronouns, the *attribute* attribute is updated to reflect this. If the second or third structure is matched, the new variable is passed recursively as the argument to the same subroutine. The part-of-speech information is also passed along as an argument in the next recursive run.

Still following our example file in listing A.1 we can see the results from running this step on the variables from the *ha-perf* predicate. The result and the corresponding line numbers can be seen in table 3.5. Note that

Table 3.5: Extracted lemmas for arguments

	Argument 1		Argument 2
	recursion 1	recursion 2	recursion 1
Evaluated variable	32	61	26
Extracted feature	VERB		PRON+
Extracted variable	61		
Extracted lemma		gå	hun
Line number	52	59	36

the variable for the first argument matches the second type of structure which means that the variable found here is passed on recursively.

Once the arguments are extracted, the *Lemma* instances are set as the *arg1* and *arg2* attributes in the *Predicate* object. The predicates are then stored in the YAML file specified in the second input argument of the script. The *Predicate* instance of the *ha-perf* predicate can be seen represented in the YAML format in listing 3.4. The *predicateExtractor.pl* script ends up extracting 37 608 EPAS in total.

Listing 3.4: An instance of a Predicate in YAML

```
--- !!perl/hash:My::Predicate
arg1: !!perl/hash:My::Lemma
  attribute: VERB
  semform: gå
arg2: !!perl/hash:My::Lemma
  attribute: PRON+
  semform: hun
pred: ha-perf
sentenceID: 5
```

3.4.2 Extracting sentences

A representation of the sentences and the lemmas they contain is necessary when annotating the anaphora in section 3.5 and when applying the selectional constraints in section 3.10. The method for extracting the lemmas is a simplified version of the method used for predicate extraction in section 3.4.1. As with the predicate extraction method, the *sentenceExtractor.pl* (listing B.5 script takes two input arguments: The folder containing the files to be parsed and the name of the output file. The output is saved in a YAML formatted file containing instances of *Sentence* objects. An example execution of the script is shown in listing 3.5.

Listing 3.5: Example execution of *sentencesExtractor.pl*

```
$ perl sentenceExtractor.pl nob-sofie-hele/ sentences.yaml
```

For each file in the folder specified in the first input argument, the sentence number is extracted from the file name by the same process as described in section 3.4.1. A *Sentence* object is created and its *id* attribute is set to the extracted sentence number. The surface form of the sentence is extracted by matching the contents of the *markup_free_sentence* feature and added as the objects *sentence* attribute. We are only interested in the lemmas of the words in the sentences, so we don't have to use a recursive method as in the previous section. The script simply matches any line with a *semform* feature and extracts the string contained within it in addition to its variable. For each of these matches, a *Lemma* object is created and the extracted string and variable are set as the *semform* and *var* attributes respectively. An array of the extracted lemmas are set as the *lemmas* attribute in the sentence object. Finally, the sentence is stored in the YAML file defined in the second input argument of the script. The sentence from our example file in listing A.1 can be seen in listing 3.6.

Listing 3.6: An instance of a Sentence in YAML

```
--- !!perl/hash:My::Sentence
id: 5
lemmas:
  - !!perl/hash:My::Lemma
    semform: ha-perf
    var: 15
  - !!perl/hash:My::Lemma
    semform: hun
    var: 26
  - !!perl/hash:My::Lemma
    semform: gå
    var: 61
  - !!perl/hash:My::Lemma
    semform: stykke
    var: 2
  - !!perl/hash:My::Lemma
    semform: den
    var: 11
  - !!perl/hash:My::Lemma
    semform: første
    var: 12
  - !!perl/hash:My::Lemma
    semform: sammen-med
    var: 51
  - !!perl/hash:My::Lemma
    semform: Jorunn
    var: 45
  - !!perl/hash:My::Lemma
    semform: Jorunn
    var: 55
  - !!perl/hash:My::Lemma
    semform: hun
    var: 132
  - !!perl/hash:My::Lemma
    semform: Jorunn
    var: 127
sentence: Det første stykket hadde hun gått sammen med Jorunn.
```

3.5 Annotating antecedents

In order to measure the accuracy of a model for anaphora resolution, a corpus where the the antecedents are reliably annotated is needed. One such corpus exists for Norwegian, the BREDT corpus [Borthen et al., 2007], and this was used by Nøklestad [2009]. This corpus, however, fail to meet the criteria provided in section 3.1 given that it consists of several fiction stories not confined to one thematic domain. The best course of action then remains to manually annotate the anaphora in Sofies Verden with their antecedents.

Annotating anaphora is quite a tedious manual task, so I ventured to speed up the process by developing a tool for antecedent annotation of the EPAS extracted in section 3.4.1. The goal is that a subset of the EPAS containing a singular personal pronoun as its first argument is matched with its corresponding antecedent. The program developed for this, *guiAnnotator.pl*, takes three input arguments: A YAML file containing the EPAS extracted in section 3.4.1, a yaml file containing the sentences extracted in section 3.4.2 and file name of where you want to store the annotated EPAS. The source code for the program is shown in listing B.7. An example execution of the program is shown in listing 3.7.

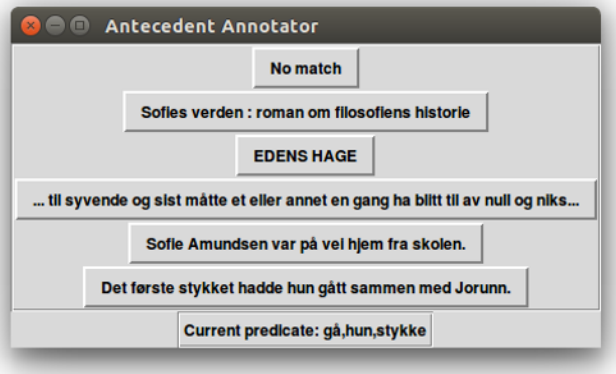
Listing 3.7: Example execution of *guiAnnotator.pl*

```
$ perl guiAnnotator.pl preds.yaml sentences-hele.yaml annotated.yaml
```

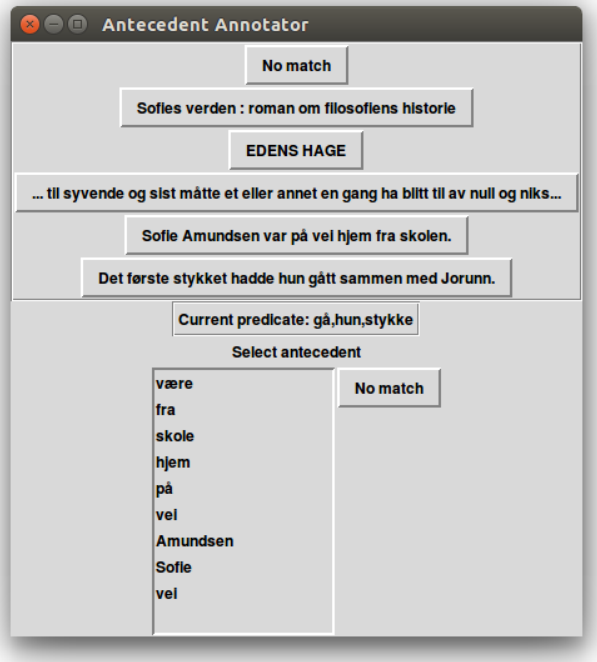
The program works on the principle that the user is presented with an EPAS, the sentence it occurs in and a context of the preceding sentences. The user then selects the sentence where the antecedent occurs, whereby a list of the lemmas in the selected sentence is presented and the user selects one of them. The selected lemma is then added as the antecedent for the EPAS and the process is repeated. In order to make the task of selecting

antecedents as easy as possible for the user, I decided to make a graphical interface. Perl supports the popular *TK* toolkit for graphical user interface widgets which allows you to easily display windows, buttons and lists.

The program starts by loading the EPAS and sentences into memory. If the output file specified in the third input argument already exists, the *index* attribute of the last EPAS is extracted, enabling the program to keep track of which EPAS have already been annotated between executions of the program. The main loop of the program iterates through all the EPAS, and the iteration starts at the index extracted from the output file. If the first *attribute* attribute of the *arg1* attribute of the EPAS equals a personal pronoun (PRON+), the dialogue window in figure 3.1a is shown. A textual representation of the EPAS is shown at the bottom of the window, and the sentence the EPAS was extracted from is shown immediately above it. This sentence is extracted by comparing the *sentenceID* attribute of the EPAS to the *id* attribute of the *Sentence*. Likewise, the context of up to nine preceding sentences is found by subtracting one through nine from the EPAS *sentenceID* and extracting the sentences with matching *id* attributes.



(a) A choice of sentences



(b) A choice of lemmas

Figure 3.1: The *guiAnnotator.pl* dialog boxes

Each of the sentences is a clickable button, and once the user clicks the sentence where the antecedent is located, the dialogue window in figure 3.1b is shown. As this model only operates with single lemmas as antecedents, the correct antecedent in this case is decided to be the lemma *Sofie*. Once this lemma is clicked, the *Lemma* object is extracted from the sentence and set as the *antecedent* attribute of the EPAS. The resulting *Predicate* object is shown in listing 3.8. There is also the option for the user to click the *No match* button if the antecedent can't be found in any of the context sentences, in which case the *antecedent* attribute of the EPAS will be marked as not found with a dummy lemma. Using the *guiAnnotator.pl* script, 466 anaphora were annotated with their antecedent.

3.6 Machine learning with TiMBL

In this section I will describe how to format the data for machine learning, and how to build a model for anaphora resolution using TiMBL.

Listing 3.8: An instance of an annotated Predicate in YAML

```
--- !!perl/hash:My::Predicate
antecedent: !!perl/hash:My::Lemma
  semform: Sofie
  sentenceID: 4
  var: 126
arg1: !!perl/hash:My::Lemma
  attribute: PRON+
  semform: hun
arg2: !!perl/hash:My::Lemma
  attribute: ''
  semform: stykke
index: 5
pred: gå
sentenceID: 5
```

3.6.1 TiMBL

The Tilburg Memory Based Learner is a software package that implements a *k*-Nearest Neighbour (*k*-NN) algorithm, but stores the representation of the training set as a decision-tree structure [Daelemans et al., 2004]. This enables both basic *k*-NN modelling, decision-tree modelling and combinations of the two. In the TiMBL software package, these algorithms are called *IB1, IGTREE* and *TRIBL* respectively. It has been especially designed to be used in natural language processing tasks, as traditional machine learning algorithms are generally optimized for numerical feature values instead of the string feature values that often appear in NLP tasks. TiMBL achieves this by implementing similarity metrics like *Levenshtein distance* and the *Dice coefficient*. This project, however, will not use the string-based similarity metrics, as the Predicate-Argument pairs are symbolic features which are not to be interpreted as strings.

3.6.2 Formatting data

TiMBL requires that each instance to be learned in the training file is represented as a feature-vector. Several input formats are supported, the simplest of which is a comma separated file (.csv) where each line represents an instance with the features delimited by a *comma*. The last feature on each line is interpreted as the class value.

The EPAS extracted in section 3.4.1 are used to make the training set. It will consist of two features and a class assignment. Since we are interested in classifying the antecedent for pronouns occurring as the first argument in the EPAS, all the EPAS that have a noun as the first argument and any lemma as the second argument are extracted using the Perls script *predicateExtractor.pl*. This results in a training set with 4093 instances. The 10 most frequent lemmas occurring as predicates, argument 2 and class are shown in table 3.6.

In addition to these two features and the class, I am also including a couple of bookkeeping features that will serve a purpose in the aggregation in section 3.9, but will be ignored by the classifier. These are the sentence identifier and gender place holder features. This last feature is needed because TiMBL requires the exact same amount of features in the training set and test set, and the gender of the pronoun will be included in the test set. As such, the gender place holder will carry no information in the training set and an arbitrary value is set for it, in this case the string *gendPlaceholder*. Note that this feature is in the same position as the pronoun in listing 3.10. The comma separated file will now consist of 4 features, out of which two will be ignored by the classifier, and one classification. A typical line in the training set is shown in listing 3.9. The training set is stored in the file *IdSubstArg1.csv*.

Listing 3.9: An instance in the training set

```
7044,gendPlaceholder,fortsette,vandring,Alberto
```

Table 3.6: The 10 most frequent lemmas for extracted EPAS

(a) Predicates

predicate	freq.	%
være	1152	28.14
root-kunne	240	5.86
ha	226	5.52
si	124	3.02
mene	95	2.32
bli	91	2.22
root-måtte	90	2.19
begynne	83	2.02
få	78	1.90
med	40	0.97

(b) Argument 2

argument 2	freq.	%
være	195	4.76
ha	47	1.14
ha-perf	45	1.09
i	44	1.07
root-kunne	35	0.85
gå	34	0.83
som	33	0.80
menneske	29	0.70
verden	27	0.65
på	26	0.63

(c) Class

class	freq.	%
Sofie	450	10.99
menneske	235	5.74
Alberto	94	2.29
Hilde	78	1.90
filosof	75	1.83
Gud	71	1.73
mor	59	1.44
Aristoteles	37	0.90
tanke	36	0.87
far	36	0.87

The test set consist of the annotated EPAS from section 3.5 where the pronoun occurs as the first argument of the EPAS. The annotated antecedent is set as the class feature and the predicate and argument 2 from the EPAS is set as the remaining features. Like the training set, the sentence identifier from the EPAS is added as a feature. The pronoun is added as a feature in the same place as the gender place holder feature in the training set. A typical line in the test set is shown in listing 3.10. The test set is stored in the file *testSet.csv*.

Listing 3.10: An instance in the test set

```
827,han,se,spire,Thales
```

As for the training set, the 10 most frequent lemmas occurring as predicates, argument 2 and class are shown in table 3.7. The predicate frequencies are fairly consistent with between the training set and the test set, but the most frequent class, *Sofie* is much more frequent in the test set than in the training set (45.49% vs. 10.99%). This discrepancy is largely due to the fact that the classes in the training set are drawn from all the EPAS that contain a noun as argument 1 while the classes in the test set are dependent on the pronouns from a limited part of the whole text, mainly chapter 1 and 2. Having one class with such a dominant frequency as *Sofie* makes the model prone to overemphasize the importance of this class and going as far as predicting the class for all instances in the test set. This is related to the concept of *overfitting* in supervised learning algorithms. A definition of overfitting is given as: “An induction algorithm overfits the dataset if it models the given data too well and its predictions are poor.” by Kohavi and Sommerfield [1995]. If such overfitting occurs, we will expect to see the classification accuracy on the test set increase until it reaches a peak of 45.49%, the same percentage as the frequency of the most frequent class in the test set.

3.6.3 Classification

Once you have prepared the training and test sets, building the model in TiMBL is quite a straight forward task. TiMBL is operated through a command line interface by specifying the training file and the test file. The command given in listing 3.11 will give us the default IB1 model.

Listing 3.11: Building the default IB1 model in TiMBL

```
$ timbl -f IdSubstArg1.csv -t testSet.csv
```

Recall, however, that the first two features are bookkeeping features that the model needs to ignore. The command line argument *-m* determines the overlap metric to be used on the different features, and among these is the ignore metric (*I*). We set the default metric to be the overlap metric (*O*) since the features are to

Table 3.7: The 10 most frequent lemmas for annotated EPAS

(a) Predicates

predicate	freq.	%
være	45	9.66
si	19	4.08
mene	17	3.65
root-kunne	14	3.00
vite	14	3.00
se	13	2.79
lese	11	2.36
få	10	2.15
ha	10	2.15
gå	9	1.93

(b) Argument 2

argument 2	freq.	%
pro	160	34.33
være	21	4.51
den	13	2.79
konvolutt	10	2.15
brev	7	1.50
ha-perf	6	1.29
epist-måtte	5	1.07
ark	4	0.86
filosof	4	0.86
øye	4	0.86

(c) Class

class	freq.	%
Sofie	212	45.49
Sokrates	64	13.73
mor	18	3.86
Anaxagoras	16	3.43
Thales	13	2.79
mann	12	2.58
Demokrit	11	2.36
Parmenides	11	2.36
Tor	10	2.15
far	8	1.72

be seen as atomic, not string based, and the first through second metric to be ignored using the command line argument *-mO:I1-2*. In addition, we redirect the standard output to a file, as information about the operation of the algorithm is sent here. The whole command is given in listing 3.12.

Listing 3.12: Building the default IB1 model with ignored features in TiMBL

```
$ timbl -f IdSubstArg1.csv -t testSet.csv -mO:I1-2 > IB1-exp1
```

In addition to the IB1 algorithm, TiMBL also implements a decision tree algorithm, the IGTREE algorithm. This algorithm features faster computation times with an accuracy comparable to the IB1 algorithm, but sometimes performs even better [Daelemans et al., 2004]. To select this algorithm the command line argument *-a* is needed, and save the addition of this argument, the rest of the command given in listing 3.13 is equal.

Listing 3.13: Building IGTREE model TiMBL

```
$ timbl -f IdSubstArg1.csv -t testSet.csv -a 1 -mO:I1-2 > IGTREE-exp1
```

These models give us an accuracy of **34.97%** and **39.69%** correctly classified instances respectively. The whole output of the algorithms are shown in listings A.2 and A.3. For each of the algorithms an output file offered by TiMBL, in which each of the instances of the test set is displayed along with the predicted class, enabling us to study the results more closely. One instance from one such file is given in listing 3.14. The frequencies of the 10 most frequently predicted classes are displayed in table 3.8.

Listing 3.14: Classification of one instance in TiMBL

```
1785,han,mene,føre*til,Sokrates,Aristoteles
```

The very high frequencies of the *Sofie* class is clear evidence of overfitting by the model, and the IGTREE algorithm fares worse in this respect. Indeed, if we raise the k-number in the k-NN algorithm the accuracy approaches the 45.59% mark until it stops there and won't go higher. Overfitting of the *Sofie* class aside, the rest of the classes follow the frequencies of the training set in table 3.6c closely, showing that the model still has merit. Consequently, we can use the IB1 $k = 4$ model as the baseline to which we compare the other models, as it has the same accuracy as a ZeroR classification which always predicts the majority class. As the IB1 algorithm has less problems with overfitting than the IGTREE algorithm, this algorithm will be used going forward. The fairly low accuracy of the model on the other classes than *Sofie* is to be expected given that the model does not take into account the context of where the anaphora were introduced. For this I will make use of the ignored *sentenceID* feature.

Table 3.8: The 10 most frequent predicted classes for the IB1 and IGTREE algorithms

(a) IB1

class	freq.	%
Sofie	290	62.23
menneske	57	12.23
Alberto	8	1.71
Hilde	7	1.50
Empedokles	5	1.07
Sokrates	4	0.85
filosof	4	0.85
far	4	0.85
Aristoteles	4	0.85
mål	3	0.64

(b) IGTREE

class	freq.	%
Sofie	337	72.31
menneske	16	3.43
Hilde	6	1.28
Gud	6	1.28
Empedokles	5	1.07
Alberto	5	1.07
Sokrates	4	0.85
mål	4	0.85
Jorunn	4	0.85
filosof	4	0.85

(c) IB1 k=4

class	freq.	%
Sofie	466	100

3.7 A fuzzy classification model

A typical classification algorithm will only provide you with the class with the highest probability, which is sufficient for classification tasks where the classes are mutually exclusive or when there there are few classes. Our training set, however, contains 1030 different classes. Given the high number of classes, the highest probable class classified by the classifier has lower chances of being the correct one, but the second or third most probable class might still be correct. For each instance in the test set, the IB1 algorithm in TiMBL builds a set of its nearest neighbours and chooses the most frequent among these as the most probable class [Daelemans and Van den Bosch, 2005, p. 28]. This whole set can be output by TiMBL in the output file, one instance of which is shown in listing 3.15.

Listing 3.15: Fuzzy classification with TiMBL

```
1785,han,mene,føre*til,Sokrates,Aristoteles { Jorunn 2.00000, Sofie 1.00000, far 2.00000, 1
     1.00000, filosof 2.00000, ord 1.00000, neger 1.00000, tanke 1.00000, Aristoteles
    11.0000, Thales 1.00000, Anaximenes 1.00000, Heraklit 3.00000, Empedokles 5.00000,
    Anaxagoras 1.00000, Demokrit 5.00000, sofist 1.00000, Sokrates 5.00000, Platon 6.00000,
     kyniker 1.00000, stoiker 1.00000, Plotin 1.00000, Paulus 1.00000, Augustin 1.00000,
    Thomas 2.00000, Aquinas 1.00000, astronom 1.00000, Darwin 2.00000, Spinoza 2.00000,
    Descartes 3.00000, rasjonalist 2.00000, Berkeley 1.00000, empirist 1.00000, Locke
    2.00000, gudsforestilling 1.00000, Hegel 2.00000, Rousseau 1.00000, Kant 5.00000, kant
    2.00000, romantiker 1.00000, Kierkegaard 2.00000, Marx 3.00000, Freud 4.00000, rektor
    1.00000, Lamarck 1.00000, Sartre 2.00000, Beauvoir 1.00000 }
```

We can see that the most frequent class, *Aristoteles* with a frequency of 11, is the wrongfully selected antecedent. The correct antecedent, *Sokrates*, however, is among the third most frequent classes with a frequency of 5. Furthermore, we can see, as the distributional hypothesis predicts, that the nouns are neatly grouped by their semantics. Sokrates and Aristoteles are joined by their fellow philosophers in addition to more general philosophic characterisations like *stoic*, *cynic* and *empiricist*.

3.7.1 Considering the context

A fuzzy machine learning algorithm is not very helpful without connecting it to the context of where the anaphora appears. In his doctoral thesis, [Nøklestad, 2009, p. 242] observes that the proximity of the antecedent to the anaphor is an important factor in earlier anaphora resolution work. Lappin and Leass's algorithm ranks sentence recency highly as a salience factor, and the Hobbs algorithm selects the closest available antecedent. This preference was observed when annotating the antecedents in section 3.5. Antecedents were

usually found within the same sentence as the anaphor or in the preceding sentence. Some antecedents, however, were found as far as 9 sentences before the anaphor, and some antecedents were even beyond this scope and were not annotated. This observation is consistent with the observation from Nøklestad [2009] about Norwegian fiction data, where anaphor-antecedent distances have a wider span.

Considering that the maximum anaphor-antecedent distance that was annotated in section 3.5 was 9, a method for selecting antecedent candidates from the fuzzy classification obtained in section 3.7 emerges. For an anaphor, given the set A of fuzzy classifications, and the set B of all lemmas from the 9 preceding sentences, antecedent candidates (C) can be found by intersecting the two sets: $C = A \cap B$. A Perl script that performs this intersection, *fuzzyModel.pl*, is described in the following section 3.7.2.

3.7.2 Building the model

This Perl script *fuzzyModel.pl* takes two input arguments: The output file from the fuzzy classification from section 3.7 and the extracted sentences from section 3.4.2. The output is a model that classifies antecedents based on candidates that are selected from the intersection between the fuzzy classification and the sentence context.

The output from the fuzzy classification consists of all of the instances in the test set where each instance is formatted as in listing 3.15. Each line in the input file corresponds to one instance, so for each line the following is extracted: Sentence number, pronoun (anaphor), predicate, argument 2, antecedent, predicted antecedent and the set of fuzzy classification set. For each of the candidates in the fuzzy classification set, a *Lemma* object is created where the *semform* attribute is set to the lemma and the *frequency* attribute is set to the number listed next to the lemma. These lemmas will be the set A. The context of the anaphor is extracted using the sentences from the sentence input file. Using the extracted sentence number, the sentence with the corresponding sentence number and its nine preceding sentences are extracted, and from each of these sentences the *lemmas* attribute is extracted. In order to evaluate this model, all the antecedents from the fuzzy classification will be set aside and only consulted during the evaluation phase. In addition to the frequency of the antecedent candidates, their distance to the anaphor is also calculated by subtracting the sentence number of the antecedent candidate from the sentence number of the entity being analysed. This gives us a distance metric that is equal to the number of sentences between the anaphor and the antecedent candidate.

The model starts with the assumption that any prediction from the fuzzy classification that is not found in within the context of the nine preceding sentences is wrong and needs to be reassessed, while any prediction that is found within the context is assumed to be valid. When a prediction needs to be reassessed, the candidates from the fuzzy classification are consulted. Any of the candidates that appear in the context are proposed as antecedent candidates. The resulting list is sorted by frequency, and the candidate with the highest frequency is proposed as the valid prediction. When the prediction is updated, this new prediction is compared to the annotated antecedent. If the prediction is equal to the antecedent, it is counted as correctly classified. If the prediction is not equal to the antecedent, but any of the following antecedent candidates are equal to the antecedent, the prediction is counted as partially correct.

The output of the model is a text file where each instance is printed. If the original prediction was found within the context, the instance is prefixed with “In context: ”. If original prediction was not found within the context, the instance is prefixed with “Not in context: ”, and the sorted list of antecedent candidates is presented. The instance presented in listing 3.15 in the previous model can be seen presented in this model in listing 3.16, where the frequency of each antecedent candidate is followed by its distance to the anaphor. Statistics on the model are printed at the end of the file: The total number of instances, the number of partially correct predictions, the number of correct predictions, and the accuracy calculated as the number of correct predictions divided by the total number of instances. At the end of the file, the 10 most frequent predictions are printed along with their frequencies.

Listing 3.16: The updated instance from listing 3.15

```
Not in Context: 1785,han,mene,føre*til,Sokrates,Aristoteles: Correct!Sokrates(5.00000,2),
    rasjonalist(2.00000,4),
```

This model reaches an accuracy of **45.06%** while drastically reducing the problem of overfitting. The number of correct classifications is 210, while the number of partially correct classifications is 62. As shown in table 3.9, the frequency of *Sofie* is considerably lower than the ones achieved by the IB1 algorithm in table 3.8a, and on par with the actual frequency of *Sofie* in the test set in table 3.7c. If the partially correct predictions are counted towards the number of correct predictions, an accuracy of **58.36%** is reached.

Table 3.9: The 10 most frequent predicted classes for the IB1-fuzzy model

class	freq.	%
Sofie	204	43.78
menneske	55	11.80
filosof	20	4.29
Sokrates	14	3.00
pappa	8	1.72
fornuft	5	1.07
vann	4	0.86
mor	4	0.86
kvinne	3	0.64
natur	3	0.64

3.8 An ontology model

The *k*-NN algorithm discussed in the previous section categorizes antecedents based on a metric that emphasizes co-occurrences in an identical context as the anaphor, as the nearest neighbour is the one where all the features are equal. Eiken [2005] argues that antecedents that co-occur in similar contexts as the anaphor will also aid in the task of anaphora resolution, and describes a method for calculating this similarity in three steps:

level 0: words which co-occur with the target predicate are returned

level 1: words which occur in the same context as the target argument are returned

level 2: words which occur in the same context as the words found in level 1 are returned

By examining which words occurred in each others context, Eiken compiled six *associated concept classes* [Eiken, 2005, p. 73]. Given the small data set of 195 EPAS used in her experiment, manually compiling these classes was a surmountable task, but doing the same with the 37 608 EPAS in my data set is not possible. This necessitates the need for another method for representing the similarities between words.

3.8.1 Grouping words

As acquiring the necessary overview to group words in general association classes is not feasible on a big data set, an alternative is to group the words at a word-level. Instead of constructing new overreaching data structures to group words, each word can contain a set of words that it is similar to, as specified in the *similars* attribute in the *Lemma.pm* class. To find the similar words, the *associate.pl* script is used, taking the extracted EPAS from section 3.4.1 as input and outputting a list of *Lemmas* expanded with their similar lemmas. The source code for the script is shown in listing B.6. The script is an adaptation of the algorithm specified by [Eiken, 2005, p. 72]:

For each predicate:

1. Level 0:
   What is ARG1 and ARG2 in the corpus/EPAS list?

2. Level 1:
   For each ARG1 = x that was found in 1:
   In connection with which other predicates is ARG1 also=x?
   For each of these predicates:
   Which other words occur as ARG1

   Produces a list of words which occur in the same contexts as x
3. Level 2:
   For each word = y in the list from level 1:
   Which other predicates does this word also co-occur with?
   For each of these predicates:
   Which other words occur as ARG1?

   Produces a list of words which occur in the same contexts as y

My scripts starts by extracting all unique lemmas L that appear as *arg1* in the EPAS list A and are marked as a noun in the *attribute* attribute. This corresponds to *Level 0* in Eiken's algorithm. For each lemma $x \in L$, all EPAS B where the lemma from its argument 1 is equal to x are extracted. For each EPAS $z \in B$, all *arg1* that appear in EPAS $y \in A$ where $pred(y) = pred(z) \wedge arg2(y) = arg2(z)$ are extracted. These lemmas x' make up the list of words that occur in the same context as x. One lemma x' can appear in multiple equal contexts, and if that is the case the total number of equal contexts where x' appears are recorded in the *frequency* attribute of x'. In that way, the frequency of lemma x' is the number of times that lemma appears in the same context as the lemma x. All the lemmas x' are passed to the *similars* attribute of the lemma x.

For example: If the lemma x is the word *father*, all EPAS that have *father* as the first argument are extracted. This could result in the set {(be,father,home), (make*it,father,cozy)}. For each of these EPAS the words that appear in the same context as *father* are extracted. These contexts can for example be {(be,mother,home), (be,Sofie,home), (make*it,Sofie,cozy)}. The lemmas x' that would be extracted in this case would be *mother* with a frequency of 1 and *Sofie* with a frequency of 2.

At this point in the algorithm a total of 2808 lemmas are processed, and 1 393 804 similar words are associated with their respective lemmas. This took 5,8 hours of processing time on my personal computer, so processing all the 1 393 804 words for *Level 2* in Eiken's algorithm would take approximately 165 days, making this step highly impractical. In any case, an average of 496 similar words were associated with each lemma, with a median value of 380. This should be a sufficient number of associated words, eliminating the need for the taxing second level in Eiken's algorithm. As an example of the similar words produced by this algorithm, see listing 3.17 for an excerpt of the words similar to *Platon* sorted by frequency. Note that several philosophers are represented, including Aristoteles, Marx, Sokrates, Kant, Alberto, Kierkegaard, Descartes and Hume. The list is stored in a YAML file, *args.yaml*, containing 2808 *Lemma.pm* objects.

Listing 3.17: Similar words to *Platon*

```
menneske(95),Sofie(77),verden(76),filosof(72),hvor(67),være(61),Gud(58),fornuft(55),Libanon
    (54),Aristoteles(53),ha-perf(50),vesen(48),eksempel(47),far(46),skyld(42),Hilde(42),liv
    -life(41),dag(41),Marx(41),Sokrates(40),Kant(40),natur(38),filosofi(36),Alberto(33),dyr
    (32),mor(32),Kierkegaard(32),spørsmål(30),år(26),øye(26),Descartes(26),mann(25),Hume
    (25),rettighet(25),ord(24),tall(23),tilværelse(23),erkjennelse(22),tid(22),bord-table
    (22),gang-time(21),pappa(21),skole(21),idé(21),frelse(21),renessanse(21),barn(20),Hegel
    (20),kanin(19),root-kunne(19),slik(19),Jesus(18),filosofilærer(18),tanke(18),person(18)
    ,Plotin(18),klokke(18),hest(17),kristendom(17),vann(17),forandring(17),del(17),
    vitenskap(17),Demokrit(17),tro(17),drøm(17),Sartre(17),brev(16),ting-thing(16),hjelp
    (16),lov-law(16),død(15),bilde(15),stund(15),forhold(15),øyeblikk(15),bevissthet(15),
    Kristus(14),kvinne(14),skje(14),virkelighet(14),prosess(14),major(14),i(14),for(14),
    side(13),mat(13),middelalder(13),stol(13),Buddha(13),romantikk(13),gang-corridor(12),
    flosshatt(12),si(12),vinter(12),sol(12),metode(12),teater(12),bål(12),betydning(12),
    kirke(12),ånd(12),Berkeley(12),hvordan(12),Freud(12),hytte(11),time(11),_date_(11)
```

3.9 Aggregating the two models

The ontology model made in section 3.8 bears some similarities to the way the IB1 classifier finds the nearest neighbour to a set of features, mainly in the way the similarity metric works. However, while the IB1 classifier requires a set of features as input to return similar words, the ontology model can return similar words based on a single word input. This ability can be put to use in the fuzzy classification model described in section 3.7. For several of the instances classified by the fuzzy classifier, the set of nearest neighbours contains very few candidates, some examples of which can be seen in listing 3.18.

Listing 3.18: Instances with few nearest neighbours

```
871,han,velge,epist-ville,Parmenides,far { Sofie 1.00000, far 2.00000, mor 1.00000,
    Aristoteles 1.00000, Alberto 1.00000, utvalg 1.00000 }
897,han,bruke,ord,Heraklit,Spinoza { Jesus 1.00000, Spinoza 2.00000 }
903,han,si,root-kunne,Heraklit,Alberto { Alberto 1.00000 }
1723,han,være,oppta,Sokrates,Platon { Platon 2.00000, estetiker 1.00000, Marx 1.00000 }
```

This poses a problem for the model, as fewer candidates means that the probability that a candidate appears in the context is lower. In these cases, the similar words from the ontology model can be used. Consider the last instance in listing 3.18 where *Platon* is the predicted antecedent, while *Sokrates* is the correct antecedent. The set of nearest neighbours only contain two other candidates than *Platon*, and none of them is *Sokrates*. Recall, however, that *Sokrates* is marked as a similar word to *Platon* in the ontology model, as shown in listing 3.17. By expanding the set of classification candidates with the words that are similar to *Platon, estetiker* and *Marx*, *Sokrates* is added to the set. This is done for any instance where the number of nearest neighbours is below a cutoff value. The *fuzzyModel.pl* script is modified to include this method and saved as the *aggregateModel.pl* script.

With a cutoff value of 5, an accuracy of **45.49%** is reached, where there are 212 correct classifications and 115 partially correct classifications. If the partially correct classifications are counted towards the number of correct classifications, an accuracy of **70.17%** is reached.

With a cutoff value of 10, an accuracy of **44.42%** is reached, where there are 207 correct classifications and 144 partially correct classifications. If the partially correct classifications are counted towards the number of correct classifications, an accuracy of **75.32%** is reached. The distribution of the frequencies of the classifications can be seen in table 3.10.

Table 3.10: The 10 most frequent predicted classes for the aggregated models

(a) With a cutoff value of 5

class	freq.	%
Sofie	245	52.58
menneske	58	12.45
filosof	27	5.79
Sokrates	14	3.00
verden	6	1.29
fornuft	6	1.29
liv-life	5	1.07
flosshatt	4	0.86
vann	4	0.86
natur	4	0.86

(b) With a cutoff value of 10

class	freq.	%
Sofie	237	50.86
menneske	67	14.38
filosof	24	5.15
Sokrates	14	3.00
verden	9	1.93
fornuft	8	1.72
natur	7	1.50
liv-life	6	1.29
vann	5	1.07
gang-time	4	0.86

As we can see, the frequency of *Sofie* is higher than with the fuzzy model in table 3.9, but it is still lower than the IB1 algorithm in table 3.8a. This rise in frequency can be attributed to the ontology model having the same bias towards *Sofie* as the IB1 model. The accuracy of the model is not affected much compared to the fuzzy model, but the number of partially correct classifications is considerably higher. This is promising, but

the mismatch between the number of correct and partially correct predictions indicates that the frequencies of the candidates alone are not sufficient in selecting the correct antecedent.

3.10 Applying some heuristics

Having produced a list of semantically probable antecedent candidates for many of the instances in the test set, the challenge now lies in selecting the correct antecedent among them. Most anaphora resolution systems use morphosyntactic features to restrict the selection of antecedent candidates, as discussed in section 2.1.1. These featured are extracted by parsing the text with one or more parsers that tag the text with part-of-speech, morphological and syntactic information. The f-structure in INESS contains many such features, and using the data structure described in section 3.3 these features can be maintained throughout the anaphora resolution process. Using these features, a number of selectional constraints can be applied to help selecting the correct antecedent candidate. This is done by adjusting the frequency score for each antecedent candidate based on it's morphosyntacitc features. Some of the methods provided in this section are a bit crude, but serves to demonstrate a proof of concept for further development of this anaphora resolution model.

With the exception of the recency weighting, the selectional constraints used in this section are minimal requirements for any pronominal antecedent candidate. They will in principle only disqualify clearly unsuitable antecedent candidates without changing the internal ordering of the candidates' semantic probability.

3.10.1 Recency weighting

According to [Jurafsky and Martin, 2009, p. 682], most theories of reference ranks recently introduced entities higher than those introduced further back. Our model already incorporates a notion of this by restricting the choice of antecedent candidates to a context of 9 sentences, but the ranking of the resulting candidates can still be adjusted with the distance metric. The *aggregateModel.pl* script is modified to apply a modified score s to the frequency score f of the antecedent candidates based on its distance d to the anaphor so that the candidates with a closer distance to the anaphor receive a higher score. Two experiments were run, where s was calculated by a quadratic function 3.1 and a linear function 3.2.

$$s = \frac{f}{(d+1)^2} \tag{3.1}$$

$$s = \frac{f}{(d+1) * 20} \tag{3.2}$$

These experiments does not affect the accuracy of the model much, with an accuracy of **45.06%** for the quadratic function and **45.49%** for the linear function, but they alter the distribution of which antecedents were correctly predicted compared to the basic aggregated model, as seen in table 3.11. In a model where there is considerable bias towards the most frequent class, *Sofie*, this can improve the accuracy for classifying antecedents other than *Sofie*, resulting in an improved accuracy when combined with other constraints, as seen in section 3.10.4.

3.10.2 Gender agreement

While the grammatical gender of the anaphor does not need to be congruent with the gender of the antecedent, the semantic gender of the antecedent can have an effect. The noun *far (father)* has a masculine grammatical gender in Norwegian, but it also has a clear semantic masculine gender and will typically co-refer with *han (he)*. This is unlike the masculine noun *hund (dog)* which can co-refer with both *han (he)* and *hun (her)*. Nand [2008] used this notion of a semantic gender to pre-process the nouns to group them by semantic gender based on structures such as *Mr.* and *Mrs.* and whether they are members of a set of masculine or feminine terms.

Table 3.11: The 10 most frequent correctly predicted classes for the distance weighted model

(a) For equation 3.1

class	freq.
Sofie	181
Sokrates	14
mor	2
filosof	2
Tor	2
pappa	1
Hermes	1
Heraklit	1
Thales	1
Anaxagoras	1

(b) For equation 3.2

class	freq.
Sofie	185
Sokrates	10
filosof	3
Demokrit	2
mor	2
Heraklit	1
Thales	1
mamma	1
Hermes	1
mann	1

(c) Fuzzy model with cutoff = 5

class	freq.
Sofie	187
Sokrates	13
filosof	3
mor	2
Thales	1
Demokrit	1
mann	1
Hermes	1
Anaxagoras	1
Heraklit	1

Unfortunately, no comprehensive resource of nouns marked with semantic gender exists for Norwegian save for a particular class of nouns, *proper nouns*.

INESS provides a list[1] of 16 345 first names marked with gender. Granted, some names can be used by both genders, such as *Kim* and *Chris*, but this applies to only 711 of the names in the list. The names of the philosophers figuring as antecedent candidates were added to the list together with their gender.

The first letter of any proper noun in the f-structure in INESS is capitalized, unlike all regular nouns where all the letters are lowercase. This lets us identify any proper nouns that appear as antecedent candidates and we can consult the list of first names to see if they are congruent with the anaphor. The anaphor is included as the second feature in the test set, and its gender is masculine for *han* and feminine for *hun*.

The *aggregateModel.pl* script is updated with a new definition of when the predicted class is correct: If the predicted class is a proper noun where the gender is not congruent with the anaphor, the instance is marked with "Gender doesn't match: ", and the prediction is reassessed in the same manner as in section 3.7.2. If the gender of any of the antecedent candidates are incongruent with the anaphor, the candidate is discarded. A proper noun is only counted as incongruent if the name can be located in the list of first names.

Implementing the selectional restraint of gender agreement on proper nouns increased the accuracy considerably with an accuracy of **47.42%**, 221 correctly classified candidates and 106 partially correct candidates. The frequency distribution of predicted and correctly classified classes is shown in table 3.12. Note that the number of partially correct has decreased by 9 and the number of correctly classified candidates have increased by 9 in comparison to the aggregate model with a cutoff of 5 in section 3.9. This shows that the increase in accuracy is due to a better ranking of the antecedent candidates.

3.10.3 Animacy

As shown by Nøklestad [2009], one of the most valuable features for pronominal anaphora resolution is animacy information. Information on the animacy of English nouns is usually extracted from WordNet, but no such readily available resource exists for the Norwegian language. Nøklestad described a method for automatically extracting animacy information from the World Wide Web which achieved a good performance, but it is beyond the scope of this project to implement that. I can, however, use a proxy for animate nouns which will serve as a proof of concept. Recall from section 3.10.2 that proper nouns among the antecedent candidates can be located by seeing if the first letter of the lemma is capitalized. Proper nouns are used to

[1] http://clarino.uib.no/iness/resources/morphology/names/no-first-names.txt

Table 3.12: The 10 most frequent classes for the gender agreement model

(a) Predicted classes

class	freq.	%
Sofie	221	47.42
menneske	67	14.38
filosof	28	6.01
Sokrates	15	3.22
pappa	8	1.72
fornuft	6	1.29
verden	6	1.29
liv-life	5	1.07
mor	5	1.07
vann	4	0.86

(b) Correct classes

class	freq.
Sofie	187
Sokrates	14
pappa	4
mann	3
filosof	3
hund	2
mor	2
Anaxagoras	1
Heraklit	1
Hermes	1

denote several entities like countries, places and companies, but the most common type of proper noun in this material is the personal name, and any personal name is by definition animate.

The *aggreagateModel.pl* script is modified to increase the score of any antecedent candidate identified as a proper noun. Three experiments are run where the score is increased by a factor of 5, 10 and 20. The experiments reached an accuracy of **49.57%**, **53.00%** and **53.21%** respectively. The effect of increasing the factor seems to have diminishing returns after a factor of 10. The frequencies of the correct predictions are shown in table 3.13. Note that the increase in overall accuracy comes at the expense of lowering the accuracy for nouns like *mor*, *far* and *hund* that, while being animate nouns, are not considered as such by this approximate method.

Table 3.13: The 10 most frequent correctly predicted classes for the animacity model

(a) With a factor of 5

class	freq.
Sofie	195
Sokrates	21
Demokrit	3
Tor	2
Empedokles	2
mor	1
Thales	1
Aristoteles	1
Heraklit	1
Hermes	1

(b) With a factor of 10

class	freq.
Sofie	197
Sokrates	31
Tor	3
Parmenides	3
Demokrit	3
Empedokles	2
Heraklit	1
mor	1
Anaxagoras	1
Hermes	1

(c) With a factor of 20

class	freq.
Sofie	198
Sokrates	31
Demokrit	3
Tor	3
Parmenides	3
Empedokles	2
Hermes	1
Anaxagoras	1
Heraklit	1
Anaximenes	1

3.10.4 Combining the constraints

The different selectional constraints have up till now been tested in isolation with, different factors for adjusting the score of the antecedent candidates for each constraint. In this section I will try a few different permutations of these constraints and factors to see how they affect the accuracy and the distribution of the predictions. I will use a cutoff value of 5 as defined in section 3.9 for all of the experiments save the last one.

Gender and animacy agreement

Because of the diminishing returns when increasing the factor for the animacy concurrence beyond 10, only a factor of 5 and 10 is considered when adding the gender agreement constraint. For the factor of 5, an accuracy of **51.07%** is observed, an increase from the accuracy of 49.57% without the gender agreement model. For the factor of 10, an accuracy of **54.93%** is observed, also an increase from the accuracy of 53.00% without the gender agreement model. The frequencies of the correct predictions is shown i table 3.14.

Table 3.14: The 10 most frequent classes for the gender and animacy agreement model

(a) With a factor of 5

class	freq.
Sofie	195
Sokrates	22
Demokrit	3
Tor	2
hund	2
Empedokles	2
mann	2
mor	1
Hermes	1
Thales	1

(b) With a factor of 10

class	freq.
Sofie	197
Sokrates	34
Parmenides	3
Tor	3
Demokrit	3
hund	2
mann	2
Empedokles	2
Thomas	1
filosof	1

Gender and animacy agreement with recency weighting

The addition of recency weighting of the antecedent candidate scores can potentially serve to mitigate some of the bias from the animacy agreement constraint where proper nouns are preferred over other animate nouns. This is apparent for an animacy factor of 5 where an accuracy of **52.57%** when the linear distance weighting is used, an increase from the accuracy of 51.07% without the weighting. The effect disappears, however, when using an animacy factor of 10 where an accuracy of **53.86%** is reached, a lower accuracy than the accuracy of 54.93% without the weighting. The distribution of correct predictions is shown in table 3.15.

Table 3.15: The 10 most frequent classes for the gender and animacy agreement model with recency weighting

(a) With a factor of 5

class	freq.
Sofie	195
Sokrates	28
Tor	3
Demokrit	3
Empedokles	2
mann	2
hund	2
Jesus	1
Hermes	1
mor	1

(b) With a factor of 10

class	freq.
Sofie	197
Sokrates	31
Tor	3
Demokrit	3
mann	2
Empedokles	2
hund	2
Thales	1
Hermes	1
Jesus	1

3.11 Adjusting the model

Given the increase in accuracy provided by the selectional constraints in section 3.10 increasing the number antecedent candidates should produce an even higher accuracy. One way of doing that is by increasing the cutoff value described in section 3.9, as this will increase the number of instances in the test set that will be infused with similar words from the ontology model.

With all the selectional constraints from section 3.10 active, an animacy factor of 10 and a cutoff value of 10, an accuracy of **55.57%** is reached. Increasing the cutoff value to 30 does not affect the accuracy, but the number of partially correct classifications is increased from 92 to 104. This suggests that given a more accurate selectional constraints model, the accuracy could be increased by increasing the cutoff value.

The recency weighting proved to be inconsistent in affecting the accuracy of the model. When excluding that from the model, an accuracy of **56.22%** is reached for the cutoff value of 30. The distribution of the predicted and correct classifications is shown in table 3.16.

The number of partially correct candidates can be raised even further if we ignore the assumption that predictions from the IB1 model that are found within the context are the most probable predictions. Doing this comes at the cost of lowering the accuracy of correct predictions to **41.63%**, but the number of partially correct candidates sees a heavy increase to 212 candidates. A combination of the correct and partially correct candidates yields a potential accuracy of **87.12%**.

The source code for the final model is shown in listing B.8.

Table 3.16: The 10 most frequent classes for the most accurate model

(a) Predicted classes

class	freq.	%
Sofie	241	51.72
menneske	45	9.66
Sokrates	45	9.66
Athen	15	3.22
Thomas	11	2.36
Aristoteles	8	1.72
filosof	7	1.50
Gud	6	1.29
vann	5	1.07
Tor	5	1.07

(b) Correct classes

class	freq.
Sofie	199
Sokrates	41
Demokrit	4
Parmenides	4
Tor	3
Empedokles	2
Thomas	2
mann	1
Heraklit	1
Aristoteles	1

3.12 Summary of the results

Several different models and combinations of models were presented in the previous sections. A summary of their accuracy will be presented here along with an abbreviated identification for each model. All the statistics for the models were extracted from their output file. An extract from the output file for the AG30+GEND+ANI10 model is shown in listing A.4.

IGTREE The decision tree algorithm from TiMBL yields an accuracy of **39.69%**, but suffers from overfitting of *Sofie*. It is described in section 3.6.3.

IB1 The TiMBL implementation for the *k*-NN algorithm yields an accuracy of **34.97%**. The model sufferes a bit from overfitting of *Sofie*, but not as much as IGTREE. All following models build on this model. It is described in section 3.6.3.

FUZZY The fuzzy model is based on the IB1 model where the whole set of nearest neighbours is used to form antecedent candidates. The model reaches an accuracy of **45.06%** with 62 partially correct classifications while reducing the overfitting problem. It is described in section 3.7

AG5 *and* AG10 In the aggreagate model, the ontology model from section 3.8 is used to increase the number of antecedent candidates. This results in an accuracy of **45.49%** with 115 partially correct candidates when the cutoff value is 5, and an accuracy of **44.42%** with 144 partially correct candidates when the cutoff value is 10. It is described in section 3.9.

AG5+REC-S *and* AG5+REC-L The recency model is presented in two forms: A square function and a linear function for weighting the antecedent candidates. The square function yields an accuracy of **45.06%** and the linear function yields an accuracy of **45.49%**. It is described in section 3.10.1.

AG5+GEND The gender agreement model increases the accuracy to **47.42%**. It is described in section 3.10.2.

AG5+ANI5 *and* AG5+ANI10 *and* AG5+ANI20 The animacy model comes with three different factors: A factor of 5 yields an accuracy of **49.57%**, a factor of 10 gets **53.00%**, and a factor of 20 gets **53.21%**. The model is described in section 3.10.3.

AG5+GEND+ANI5 *and* AG5+GEND+ANI10 The gender agreement model was added to the animacy model for the factors of 5 and 10. This yields an accuracy of **51.07%** and **54.93%** respectively. This is described in section 3.10.4.

AG5+REC-L+GEND+ANI5 *and* AG5+REC-L+GEND+ANI10 When the recency weighting is added to the combination the gender and animacy models, an improved accuracy of **52.57%** is reached for the animacy factor of 5. However, when the animacy factor is 10, the accuracy is decreased to **53.86%** compared to the model without the recency weighting. This is described in section 3.10.4.

AG10+REC-L+GEND+ANI10 *and* AG30+REC-L+GEND+ANI10 *and* AG30+GEND+ANI10 Increasing the cutoff value for the aggregate motel increases the accuracy when all the other models are active. For a cutoff value of 10, an accuracy of **55.57%** is reached, and the accuracy is the same for the cutoff value of 30. However, the number of partially correct classifications is increased for this cutoff value. When removing the recency weighting from the model, an accuracy of **56.22%** is reached. This is described in section 3.11.

-CA+AG30+GEND+ANI10 Ignoring the context assumption lowers the accuracy of the model drastically to an accuracy of **41.63%**, but an increase in the number of partially correct candidates increases the potential accuracy of the model to **87.12%**.

A simplified overview of all the results is presented in table 3.17. Note that the number of partially correct classifications decreases as the accuracy increases until a larger cutoff value is introduced in the last three models. The introduction of more antecedent candidates has no effect until simple heuristics like gender agreement for proper nouns and and a bias towards proper nouns is introduced. With more advanced heuristics for ranking the antecedent candidates, like the improvements for the gender and animacy models discussed in sections 3.10.2 and 3.10.3, it is anticipated that more of the antecedent candidates will be ranked correctly, thus improving the accuracy further. This is made plausible by the fact that a clearly inanimate noun, *Athen (Athens)*, is among the top ranked antecedent candidates in table 3.16a because of the crude animacy model applied in section 3.10.3.

Table 3.17: Table over the accuracy of the different models

Model	Partially correct	Correct	Potential accuracy %	Accuracy %
IB1	NA	163	NA	34.97
IGTREE	NA	185	NA	39.69
-CA+AG30+GEND+ANI10	212	194	**87.12**	41.63
AG10	144	207	75.32	44.42
FUZZY	62	210	58.37	45.06
AG5+REC-S	117	210	70.17	45.06
AG5	115	212	70.17	45.49
AG5+REC-L	115	212	70.17	45.49
AG5+GEND	106	221	70.17	47.42
AG5+ANI5	96	231	70.17	49.57
AG5+GEND+ANI5	89	238	70.17	51.07
AG5+REC-L+GEND+ANI5	82	245	70.17	52.58
AG5+ANI10	80	247	70.17	53.00
AG5+ANI120	79	248	70.17	53.22
AG5+REC-L+GEND+ANI10	76	251	70.17	53.86
AG5+GEND+ANI10	71	256	70.17	54.94
AG10+REC-L+GEND+ANI10	92	259	75.32	55.58
AG30+REC-L+GEND+ANI10	104	259	77.90	55.58
AG30+GEND+ANI10	101	262	77.90	**56.22**

Chapter 4

Final remarks

4.1 Conclusion

In this thesis I have described a method for generating semantically motivated antecedent candidates for use in pronominal anaphora resolution. While traditional anaphora resolution algorithms both locate and rank antecedent candidates based on morphosyntactic features, my method takes a novel approach. The fuzzy classification model in conjunction with the ontology model generates a set of semantically motivated antecedent candidates that are ranked based on the frequency by which they co-occur with the anaphor. After filtering out the candidates which does not occur within a defined distance from the anaphor, this ordered set serves as a starting point for different heuristics that can either disqualify candidates from the set, like the gender agreement heuristic, or give certain features a higher ranking by applying a salience function to the frequency, as with the animacy heuristic. The fact that these fairly unsophisticated heuristics managed to reach an accuracy of **56.22%** in correctly predicting the antecedent indicates that the co-occurence frequency scores of the antecedent candidates shows promise as a representation of real-world-knowledge.

As discussed in section 3.1, the material for this project was chosen because it displayed a confined thematic domain. Being a novel with a protagonist, *Sofie*, the protagonist was naturally overrepresented in both the training set and the test set. This created some problems with overfitting in the model, as discussed in section 3.6, which may have affected the results. However, the final accuracy of the model is far higher than the maximum accuracy that overfitting alone can account for.

The semantically ranked antecedent candidates could also see use in other anaphora resolution systems. One of my models in section 3.11 sacrificed prediction accuracy in favour of providing more antecedent candidates for the anaphora, ensuring that **87.12%** of the anaphora had its antecedent represented in the antecedent candidate list. In the machine learning system developed by Nøklestad [2009], the co-occurence frequencies of these antecedent candidates could be used as a numerical feature, thereby adding a semantic representation to the system.

My method also has the advantage of being fully automatic, provided that an interface to the NorGram parser in INESS is made. Given a directory of parsed sentences (*prolog-files/*) and a set of anaphora (*testSet.csv*), a full anaphora resolution model can be built using the simple shell script shown in listing 4.1.

Listing 4.1: Shell script for executing the model

```
1 $ perl predicateExtractor.pl prolog-files/ preds.yaml
2 $ perl sentenceExtractor.pl prolog-files/ sentences.yaml
3 $ perl associate.pl preds.yaml similars.yaml
4 $ perl printPreds.pl preds.yaml IdSubstArg1.csv
5 $ timbl -f IdSubstArg1.csv -t testSet.csv -mO:I1-2 > IB1-exp1
6 $ perl fullModel.pl testSet.csv.IB1.O:I1-2.gr.k1.out similars.yaml sentences.yaml 30 | tee
      AG30+GEND+ANI10
```

In addition to constituting an automatic anaphora resolution model, the scripts also form a useful tool set that can be applied to any text parsed with a ParGram-grammar, as semantic modelling can serve a purpose

in other natural language processing tasks. The script for annotating anaphora may also find use in further annotation efforts.

4.2 Future work

Given that the material studied in this project only exhibits one thematic domain, material which exhibits different thematic domains needs to be studied to validate the model's validity across domains. For the same reason, material from other genres should also be studied.

Furthermore, as the heuristics applied in section 3.10 only served as a proof of concept to gauge the quality of the antecedent candidates, improving the heuristics could potentially lead to an accuracy near the 87% mark.

Bibliography

Oren Ben-Kiki, Clark Evans, and Brian Ingerson. YAML ain't markup language (YAML)(tm) version 1.2. *YAML. org, Tech. Rep., September*, 2009. URL `http://yaml.org/spec/1.2/spec.pdf`. 13

K Borthen, L Johnsen, and C Johansson. Coding anaphor-antecedent relations–the annotation manual for bredt. In *Proceedings from the first Bergen Workshop on Anaphora Resolution (WAR I)*, pages 86–111, 2007. 17

Miriam Butt, Helge Dyvik, Tracy Holloway King, Hiroshi Masuichi, and Christian Rohrer. The parallel grammar project. In *Proceedings of the 2002 workshop on Grammar engineering and evaluation-Volume 15*, pages 1–7. Association for Computational Linguistics, 2002. 8

Jaime G. Carbonell and Ralf D. Brown. Anaphora resolution: A multi-strategy approach. In *Proceedings of the 12th Conference on Computational Linguistics - Volume 1*, COLING '88, pages 96–101, Stroudsburg, PA, USA, 1988. Association for Computational Linguistics. ISBN 963 8431 56 3. doi: 10.3115/991635.991656. URL `http://dx.doi.org/10.3115/991635.991656`. 7

Dick Crouch, Mary Dalrymple, Ron Kaplan, Tracy King, John Maxwell, and Paula Newman. XLE documentation. *Palo Alto Research Center*, 2008. URL `http://www2.parc.com/isl/groups/nltt/xle/doc/xle.html#Prolog_Output`. 11

Walter Daelemans and Antal Van den Bosch. *Memory-based language processing*. Cambridge University Press, 2005. 22

Walter Daelemans, Jakub Zavrel, Kurt van der Sloot, and Antal Van den Bosch. TiMBL: Tilburg memory-based learner. *Tilburg University*, 2004. 19, 21

Unni Eiken. Corpus-based semantic categorisation for anaphora resolution. Master's thesis, University of Bergen, 2005. 2, 3, 6, 7, 11, 24

Jostein Gaarder. *Sofies verden: roman om filosofiens historie*. Aschehoug, Oslo, Norway, 1991. 11

Jostein Gaarder and Victoria Rosén. The INESS sofie norwegian treebank, 2011. URL `http://hdl.handle.net/11509/86`. 11

Zellig Sabbettai Harris. Mathematical structures of language. 1968. 7

Donald Hindle. Noun classification from predicate-argument structures. In *Proceedings of the 28th Annual Meeting on Association for Computational Linguistics*, ACL '90, pages 268–275, Stroudsburg, PA, USA, 1990. Association for Computational Linguistics. doi: 10.3115/981823.981857. URL `http://dx.doi.org/10.3115/981823.981857`. 7

Jerry R Hobbs. Resolving pronoun references. *Lingua*, 44(4):311–338, 1978. 3, 5

Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Pearson Education, 2nd edition, 2009. 1, 2, 5, 6, 27

Ron Kohavi and Dan Sommerfield. Feature subset selection using the wrapper method: Overfitting and dynamic search space topology. In *KDD*, pages 192–197, 1995. 20

Shalom Lappin and Herbert J Leass. An algorithm for pronominal anaphora resolution. *Computational linguistics*, 20(4):535–561, 1994. 3, 5

Till Christopher Lech and Koenraad De Smedt. Ontology extraction for coreference chaining. 2005. 2, 3

Tom Michael Mitchell. *The discipline of machine learning*, volume 17. Carnegie Mellon University, School of Computer Science, Machine Learning Department, 2006. 9

Ruslan Mitkov. Outstanding issues in anaphora resolution. In *Computational Linguistics and Intelligent Text Processing*, pages 110–125. Springer, 2001. 3

Natalia N. Modjeska, Katja Markert, and Malvina Nissim. Using the web in machine learning for other-anaphora resolution. In *Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, EMNLP '03, pages 176–183, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics. doi: 10.3115/1119355.1119378. URL `http://dx.doi.org/10.3115/1119355.1119378`. 7

Parma Nand. On the use of salience weights in anaphora resolution. In *Proceedings of New Zealand Computer Science Research Student Conference (NZCSRSC)*, 2008. 27

Anders Nøklestad. *A machine learning approach to anaphora resolution including named entity recognition, PP attachment disambiguation, and animacy detection*. PhD thesis, University of Oslo, 2009. 3, 5, 17, 22, 23, 28, 35

Simone Paolo Ponzetto and Michael Strube. Exploiting semantic role labeling, wordnet and wikipedia for coreference resolution. In *Proceedings of the Main Conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, HLT-NAACL '06, pages 192–199, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics. doi: 10.3115/1220835.1220860. URL `http://dx.doi.org/10.3115/1220835.1220860`. 7

Eric S Raymond. The art of UNIX programming, 2003. URL `http://www.catb.org/esr/writings/taoup/html/ch14s04.html#perl`. 14

Victoria Rosén, Paul Meurer, Koenraad De Smedt, Miriam Butt, and Tracy Holloway King. Constructing a parsed corpus with a large LFG grammar. *Proceedings of LFG'05*, pages 371–387, 2005. 8

Victoria Rosén, Koenraad De Smedt, Paul Meurer, and Helge Dyvik. An open infrastructure for advanced treebanking. In *META-RESEARCH Workshop on Advanced Treebanking at LREC2012*, pages 22–29, 2012. 3, 8

Appendix A

Listings

Listing A.1: Prolog file format from INESS

```
% -*- coding: utf-8 -*-

fstructure('Det første stykket hadde hun gått sammen med Jorunn.',
  % Properties:
  [
  'markup_free_sentence'('Det første stykket hadde hun gått sammen med Jorunn.'),
  'xle_version'('XLE release of Apr 25, 2012 09:37.'),
  'grammar'('/home/iness/local/xledir/pargram/norwegian/bokmal/bokmal-mrs.lfg'),
  'grammar_date'('Oct 28, 2013 15:28'),
  'word_count'('9'),
  'statistics'('31 solutions, 0.570 CPU seconds, 64.773MB max mem, 1317 subtrees unified'),
  'rootcategory'('ROOT'),
  'max_medial_constituent_weight'('25'),
  'max_medial2_constituent_weight'('20'),
  'hostname'('node-9')
  ],
  % Choices:
  [
  ],
  % Equivalences:
  [
  ],
  % Constraints:
  [
  cf(1,eq(attr(var(0),'PRED'),var(15))),
  cf(1,eq(attr(var(0),'SUBJ'),var(26))),
  cf(1,eq(attr(var(0),'XCOMP'),var(32))),
  cf(1,eq(attr(var(0),'TOPIC'),var(2))),
  cf(1,eq(attr(var(0),'CHECK'),var(13))),
  cf(1,eq(attr(var(0),'TNS-ASP'),var(30))),
  cf(1,eq(attr(var(0),'VTYPE'),var(31))),
  cf(1,eq(attr(var(0),'PERF'),'+')),
  cf(1,eq(attr(var(0),'STMT-TYPE'),'decl')),
  cf(1,eq(attr(var(0),'VFORM'),'fin')),
  cf(1,eq(var(15),semform('ha-perf',55,[var(32)],[var(26)]))),
  cf(1,eq(attr(var(26),'PRED'),semform('hun',60,[],[]))),
  cf(1,eq(attr(var(26),'GEND'),var(27))),
  cf(1,eq(attr(var(26),'NTYPE'),var(28))),
  cf(1,eq(attr(var(26),'NUM'),var(29))),
  cf(1,eq(attr(var(26),'CASE'),'nom')),
  cf(1,eq(attr(var(26),'DEF'),'+')),
  cf(1,eq(attr(var(26),'GEND-SEM'),'female')),
  cf(1,eq(attr(var(26),'PERS'),'3')),
  cf(1,eq(attr(var(26),'PRON-FORM'),'hun')),
  cf(1,eq(attr(var(26),'PRON-TYPE'),'pers')),
  cf(1,eq(attr(var(26),'REF'),'+')),
  cf(1,eq(attr(var(27),'FEM'),'+')),
  cf(1,eq(attr(var(27),'MASC'),'-')),
  cf(1,eq(attr(var(27),'NEUT'),'-')),
  cf(1,eq(attr(var(28),'NSYN'),'pronoun')),
  cf(1,eq(var(29),'sg')),
  cf(1,eq(attr(var(32),'PRED'),var(61))),
  cf(1,eq(attr(var(32),'SUBJ'),var(26))),
```

```
54  cf(1,eq(attr(var(32),'OBJ'),var(2))),
55  cf(1,eq(attr(var(32),'ADJUNCT'),var(33))),
56  cf(1,eq(attr(var(32),'CHECK'),var(59))),
57  cf(1,eq(attr(var(32),'VFORM'),'sup')),
58  cf(1,eq(attr(var(32),'VTYPE'),'main')),
59  cf(1,eq(var(61),semform('gå',62,[var(64),var(77)],[]))),
60  cf(1,eq(attr(var(2),'PRED'),semform('stykke',39,[],[]))),
61  cf(1,eq(attr(var(2),'CHECK'),var(4))),
62  cf(1,eq(attr(var(2),'GEND'),var(6))),
63  cf(1,eq(attr(var(2),'NTYPE'),var(7))),
64  cf(1,eq(attr(var(2),'SPEC'),var(10))),
65  cf(1,eq(attr(var(2),'DEF'),'+')),
66  cf(1,eq(attr(var(2),'NUM'),'sg')),
67  cf(1,eq(attr(var(2),'PERS'),'3')),
68  cf(1,eq(attr(var(2),'REF'),'+')),
69  cf(1,eq(attr(var(2),'SEM-TYPE'),'temp')),
70  cf(1,eq(attr(var(4),'_ANTECED'),var(5))),
71  cf(1,eq(attr(var(4),'_DEF-MORPH'),'+')),
72  cf(1,eq(attr(var(4),'_NOUN'),'+')),
73  cf(1,eq(attr(var(4),'_PREDEF'),'+')),
74  cf(1,eq(attr(var(4),'_PREDET'),'+')),
75  cf(1,eq(attr(var(6),'FEM'),'-')),
76  cf(1,eq(attr(var(6),'MASC'),'-')),
77  cf(1,eq(attr(var(6),'NEUT'),'+')),
78  cf(1,eq(attr(var(7),'NSEM'),var(8))),
79  cf(1,eq(attr(var(7),'NSYN'),'common')),
80  cf(1,eq(attr(var(8),'TIME'),var(9))),
81  cf(1,eq(attr(var(8),'COMMON'),'count')),
82  cf(1,eq(attr(var(9),'TEMPNOUN'),'+')),
83  cf(1,eq(attr(var(10),'DET'),var(11))),
84  cf(1,eq(attr(var(10),'ORD'),var(12))),
85  cf(1,eq(attr(var(11),'PRED'),semform('den',2,[],[]))),
86  cf(1,eq(attr(var(11),'DET-TYPE'),'demORart')),
87  cf(1,eq(attr(var(12),'PRED'),semform('første',21,[],[]))),
88  cf(1,eq(attr(var(12),'NUMBER-TYPE'),'ord')),
89  cf(1,in_set(var(53),var(33))),
90  cf(1,eq(var(34),var(53))),
91  cf(1,eq(attr(var(34),'PRED'),var(51))),
92  cf(1,eq(attr(var(34),'OBJ'),var(45))),
93  cf(1,eq(attr(var(34),'CHECK'),var(37))),
94  cf(1,eq(attr(var(34),'PFORM'),'sammen-med')),
95  cf(1,eq(attr(var(34),'PTYPE'),'sem')),
96  cf(1,eq(var(51),semform('sammen-med',86,[var(45)],[]))),
97  cf(1,eq(var(45),var(55))),
98  cf(1,eq(attr(var(45),'PRED'),semform('Jorunn',95,[],[]))),
99  cf(1,eq(attr(var(45),'CHECK'),var(46))),
100 cf(1,eq(attr(var(45),'GEND'),var(47))),
101 cf(1,eq(attr(var(45),'NTYPE'),var(48))),
102 cf(1,eq(attr(var(45),'CASE'),'obl')),
103 cf(1,eq(attr(var(45),'DEF'),'+')),
104 cf(1,eq(attr(var(45),'NUM'),'sg')),
105 cf(1,eq(attr(var(45),'PERS'),'3')),
106 cf(1,eq(attr(var(45),'REF'),'+')),
107 cf(1,eq(attr(var(46),'_ANTECED'),var(39))),
108 cf(1,eq(attr(var(46),'_NE'),'+')),
109 cf(1,eq(attr(var(39),'NUM'),var(40))),
110 cf(1,eq(attr(var(39),'PERS'),var(41))),
111 cf(1,eq(var(40),var(65))),
112 cf(1,eq(var(41),var(66))),
113 cf(1,eq(attr(var(47),'NEUT'),'-')),
114 cf(1,eq(attr(var(48),'NSEM'),var(49))),
115 cf(1,eq(attr(var(48),'NSYN'),'proper')),
116 cf(1,eq(attr(var(49),'PROPER'),var(50))),
117 cf(1,eq(attr(var(50),'PROPER-TYPE'),'name')),
118 cf(1,eq(attr(var(37),'_ANTECED'),var(39))),
119 cf(1,eq(attr(var(53),'PRED'),var(58))),
120 cf(1,eq(attr(var(53),'OBJ'),var(55))),
121 cf(1,eq(attr(var(53),'CHECK'),var(54))),
122 cf(1,eq(attr(var(53),'PFORM'),var(57))),
123 cf(1,eq(attr(var(53),'PTYPE'),'sem')),
124 cf(1,eq(var(58),var(51))),
125 cf(1,eq(attr(var(55),'PRED'),semform('Jorunn',95,[],[]))),
126 cf(1,eq(attr(var(55),'CHECK'),var(56))),
127 cf(1,eq(attr(var(55),'GEND'),var(47))),
128 cf(1,eq(attr(var(55),'NTYPE'),var(48))),
```

```
  cf(1,eq(attr(var(55),'CASE'),'obl')),
  cf(1,eq(attr(var(55),'DEF'),'+')),
  cf(1,eq(attr(var(55),'NUM'),'sg')),
  cf(1,eq(attr(var(55),'PERS'),'3')),
  cf(1,eq(attr(var(55),'REF'),'+')),
  cf(1,eq(var(56),var(46))),
  cf(1,eq(var(297),var(290))),
  cf(1,eq(var(54),var(37))),
  cf(1,eq(var(299),var(301))),
  cf(1,eq(var(57),'sammen-med')),
  cf(1,eq(var(302),var(304))),
  cf(1,eq(attr(var(59),'_AUX1COMP'),'+')),
  cf(1,eq(attr(var(59),'_SUPINE'),'+')),
  cf(1,eq(attr(var(59),'_TOPVP'),'-')),
  cf(1,eq(var(64),var(26))),
  cf(1,eq(attr(var(64),'NUM'),var(65))),
  cf(1,eq(attr(var(64),'PERS'),var(66))),
  cf(1,eq(var(65),var(29))),
  cf(1,eq(var(66),'3')),
  cf(1,eq(attr(var(13),'_ANTECED'),var(5))),
  cf(1,eq(attr(var(13),'_MAIN-CL'),'+')),
  cf(1,eq(attr(var(30),'MOOD'),'indicative')),
  cf(1,eq(attr(var(30),'TENSE'),'past')),
  cf(1,eq(var(31),'aux')),
  cf(1,eq(attr(var(69),'LEFT_SISTER'),var(70))),
  cf(1,eq(attr(var(70),'LEFT_SISTER'),var(71))),
  cf(1,eq(attr(var(70),'RIGHT_SISTER'),var(69))),
  cf(1,eq(attr(var(71),'RIGHT_DAUGHTER'),var(72))),
  cf(1,eq(attr(var(71),'RIGHT_SISTER'),var(70))),
  cf(1,eq(var(73),var(32))),
  cf(1,eq(attr(var(73),'PRED'),var(61))),
  cf(1,eq(attr(var(73),'SUBJ'),var(64))),
  cf(1,eq(attr(var(73),'OBJ'),var(77))),
  cf(1,eq(attr(var(73),'ADJUNCT'),var(74))),
  cf(1,eq(attr(var(73),'CHECK'),var(75))),
  cf(1,eq(attr(var(73),'VFORM'),var(79))),
  cf(1,eq(attr(var(73),'VTYPE'),'main')),
  cf(1,eq(var(77),var(2))),
  cf(1,eq(var(311),var(312))),
  cf(1,eq(var(313),var(314))),
  cf(1,eq(var(74),var(33))),
  cf(1,eq(var(75),var(59))),
  cf(1,eq(attr(var(75),'_SUPINE'),var(76))),
  cf(1,eq(attr(var(75),'_TOPVP'),'-')),
  cf(1,eq(var(76),'+')),
  cf(1,eq(var(79),'sup')),
  cf(1,eq(attr(var(86),'LEFT_SISTER'),var(87))),
  cf(1,eq(attr(var(86),'RIGHT_DAUGHTER'),var(88))),
  cf(1,eq(var(87),var(230))),
  cf(1,eq(attr(var(88),'RIGHT_DAUGHTER'),var(89))),
  cf(1,eq(attr(var(89),'LEFT_SISTER'),var(90))),
  cf(1,eq(attr(var(90),'LEFT_SISTER'),var(91))),
  cf(1,eq(attr(var(90),'RIGHT_SISTER'),var(89))),
  cf(1,eq(attr(var(91),'LEFT_SISTER'),var(92))),
  cf(1,eq(attr(var(91),'RIGHT_SISTER'),var(90))),
  cf(1,eq(attr(var(92),'RIGHT_SISTER'),var(91))),
  cf(1,eq(attr(var(98),'RIGHT_DAUGHTER'),var(69))),
  cf(1,eq(attr(var(98),'RIGHT_SISTER'),var(99))),
  cf(1,eq(proj(var(98),'f::'),var(73))),
  cf(1,eq(attr(var(99),'LEFT_SISTER'),var(100))),
  cf(1,eq(attr(var(99),'RIGHT_DAUGHTER'),var(86))),
  cf(1,eq(var(100),var(98))),
  cf(1,eq(var(106),var(98))),
  cf(1,eq(attr(var(106),'RIGHT_DAUGHTER'),var(69))),
  cf(1,eq(proj(var(106),'f::'),var(73))),
  cf(1,eq(attr(var(163),'RIGHT_DAUGHTER'),var(164))),
  cf(1,eq(attr(var(163),'RIGHT_SISTER'),var(165))),
  cf(1,eq(var(164),var(186))),
  cf(1,eq(attr(var(165),'LEFT_SISTER'),var(163))),
  cf(1,eq(attr(var(165),'RIGHT_DAUGHTER'),var(166))),
  cf(1,eq(attr(var(166),'LEFT_SISTER'),var(167))),
  cf(1,eq(attr(var(166),'RIGHT_DAUGHTER'),var(169))),
  cf(1,eq(attr(var(167),'RIGHT_DAUGHTER'),var(168))),
  cf(1,eq(attr(var(167),'RIGHT_SISTER'),var(166))),
  cf(1,eq(attr(var(171),'RIGHT_DAUGHTER'),var(172))),
```

```
204   cf(1,eq(attr(var(171),'RIGHT_SISTER'),var(174))),
205   cf(1,eq(attr(var(172),'LEFT_SISTER'),var(173))),
206   cf(1,eq(var(173),var(210))),
207   cf(1,eq(attr(var(174),'LEFT_SISTER'),var(171))),
208   cf(1,eq(attr(var(174),'RIGHT_DAUGHTER'),var(175))),
209   cf(1,eq(attr(var(175),'LEFT_SISTER'),var(176))),
210   cf(1,eq(attr(var(175),'RIGHT_DAUGHTER'),var(184))),
211   cf(1,eq(attr(var(176),'RIGHT_DAUGHTER'),var(177))),
212   cf(1,eq(attr(var(176),'RIGHT_SISTER'),var(175))),
213   cf(1,eq(attr(var(177),'RIGHT_DAUGHTER'),var(178))),
214   cf(1,eq(attr(var(178),'LEFT_SISTER'),var(179))),
215   cf(1,eq(attr(var(179),'LEFT_SISTER'),var(180))),
216   cf(1,eq(attr(var(179),'RIGHT_SISTER'),var(178))),
217   cf(1,eq(attr(var(180),'LEFT_SISTER'),var(181))),
218   cf(1,eq(attr(var(180),'RIGHT_SISTER'),var(179))),
219   cf(1,eq(attr(var(181),'LEFT_SISTER'),var(182))),
220   cf(1,eq(attr(var(181),'RIGHT_DAUGHTER'),var(183))),
221   cf(1,eq(attr(var(181),'RIGHT_SISTER'),var(180))),
222   cf(1,eq(attr(var(182),'RIGHT_SISTER'),var(181))),
223   cf(1,eq(attr(var(184),'RIGHT_DAUGHTER'),var(185))),
224   cf(1,eq(var(185),var(99))),
225   cf(1,eq(attr(var(185),'LEFT_SISTER'),var(98))),
226   cf(1,eq(attr(var(186),'LEFT_SISTER'),var(187))),
227   cf(1,eq(attr(var(186),'RIGHT_DAUGHTER'),var(174))),
228   cf(1,eq(attr(var(187),'RIGHT_DAUGHTER'),var(188))),
229   cf(1,eq(attr(var(187),'RIGHT_SISTER'),var(186))),
230   cf(1,eq(attr(var(188),'LEFT_SISTER'),var(189))),
231   cf(1,eq(attr(var(188),'RIGHT_DAUGHTER'),var(194))),
232   cf(1,eq(attr(var(189),'RIGHT_DAUGHTER'),var(190))),
233   cf(1,eq(attr(var(189),'RIGHT_SISTER'),var(188))),
234   cf(1,eq(attr(var(190),'LEFT_SISTER'),var(191))),
235   cf(1,eq(attr(var(191),'LEFT_SISTER'),var(192))),
236   cf(1,eq(attr(var(191),'RIGHT_SISTER'),var(190))),
237   cf(1,eq(attr(var(192),'LEFT_SISTER'),var(193))),
238   cf(1,eq(attr(var(192),'RIGHT_SISTER'),var(191))),
239   cf(1,eq(attr(var(193),'RIGHT_SISTER'),var(192))),
240   cf(1,eq(attr(var(194),'LEFT_SISTER'),var(195))),
241   cf(1,eq(attr(var(194),'RIGHT_DAUGHTER'),var(204))),
242   cf(1,eq(attr(var(195),'RIGHT_DAUGHTER'),var(196))),
243   cf(1,eq(attr(var(195),'RIGHT_SISTER'),var(194))),
244   cf(1,eq(attr(var(196),'LEFT_SISTER'),var(197))),
245   cf(1,eq(attr(var(196),'RIGHT_DAUGHTER'),var(203))),
246   cf(1,eq(attr(var(197),'LEFT_SISTER'),var(198))),
247   cf(1,eq(attr(var(197),'RIGHT_SISTER'),var(196))),
248   cf(1,eq(attr(var(198),'LEFT_SISTER'),var(199))),
249   cf(1,eq(attr(var(198),'RIGHT_DAUGHTER'),var(202))),
250   cf(1,eq(attr(var(198),'RIGHT_SISTER'),var(197))),
251   cf(1,eq(attr(var(199),'LEFT_SISTER'),var(200))),
252   cf(1,eq(attr(var(199),'RIGHT_SISTER'),var(198))),
253   cf(1,eq(attr(var(200),'LEFT_SISTER'),var(201))),
254   cf(1,eq(attr(var(200),'RIGHT_SISTER'),var(199))),
255   cf(1,eq(attr(var(201),'RIGHT_SISTER'),var(200))),
256   cf(1,eq(attr(var(204),'RIGHT_DAUGHTER'),var(205))),
257   cf(1,eq(attr(var(205),'LEFT_SISTER'),var(206))),
258   cf(1,eq(attr(var(206),'LEFT_SISTER'),var(207))),
259   cf(1,eq(attr(var(206),'RIGHT_SISTER'),var(205))),
260   cf(1,eq(attr(var(207),'LEFT_SISTER'),var(208))),
261   cf(1,eq(attr(var(207),'RIGHT_SISTER'),var(206))),
262   cf(1,eq(attr(var(208),'LEFT_SISTER'),var(209))),
263   cf(1,eq(attr(var(208),'RIGHT_SISTER'),var(207))),
264   cf(1,eq(attr(var(209),'RIGHT_SISTER'),var(208))),
265   cf(1,eq(attr(var(210),'LEFT_SISTER'),var(211))),
266   cf(1,eq(attr(var(210),'RIGHT_SISTER'),var(172))),
267   cf(1,eq(attr(var(211),'RIGHT_DAUGHTER'),var(212))),
268   cf(1,eq(attr(var(211),'RIGHT_SISTER'),var(210))),
269   cf(1,eq(var(219),var(187))),
270   cf(1,eq(attr(var(219),'RIGHT_DAUGHTER'),var(188))),
271   cf(1,eq(attr(var(230),'RIGHT_DAUGHTER'),var(231))),
272   cf(1,eq(attr(var(230),'RIGHT_SISTER'),var(86))),
273   cf(1,eq(attr(var(231),'LEFT_SISTER'),var(232))),
274   cf(1,eq(attr(var(231),'RIGHT_DAUGHTER'),var(236))),
275   cf(1,eq(attr(var(232),'RIGHT_DAUGHTER'),var(233))),
276   cf(1,eq(attr(var(232),'RIGHT_SISTER'),var(231))),
277   cf(1,eq(attr(var(233),'LEFT_SISTER'),var(234))),
278   cf(1,eq(attr(var(233),'RIGHT_DAUGHTER'),var(235))),
```

```
  cf(1,eq(attr(var(234),'RIGHT_SISTER'),var(233))),
  cf(1,eq(var(132),var(26))),
  cf(1,eq(attr(var(132),'PRED'),semform('hun',60,[],[]))),
  cf(1,eq(attr(var(132),'GEND'),var(133))),
  cf(1,eq(attr(var(132),'NTYPE'),var(28))),
  cf(1,eq(attr(var(132),'CASE'),'nom')),
  cf(1,eq(attr(var(132),'DEF'),'+')),
  cf(1,eq(attr(var(132),'GEND-SEM'),'female')),
  cf(1,eq(attr(var(132),'NUM'),'sg')),
  cf(1,eq(attr(var(132),'PERS'),'3')),
  cf(1,eq(attr(var(132),'PRON-FORM'),'hun')),
  cf(1,eq(attr(var(132),'PRON-TYPE'),'pers')),
  cf(1,eq(attr(var(132),'REF'),'+')),
  cf(1,eq(var(133),var(27))),
  cf(1,eq(attr(var(133),'FEM'),'+')),
  cf(1,eq(attr(var(133),'MASC'),'-')),
  cf(1,eq(attr(var(133),'NEUT'),'-')),
  cf(1,eq(var(319),var(292))),
  cf(1,eq(var(320),var(294))),
  cf(1,eq(var(127),var(45))),
  cf(1,eq(attr(var(127),'PRED'),semform('Jorunn',95,[],[]))),
  cf(1,eq(attr(var(127),'CHECK'),var(128))),
  cf(1,eq(attr(var(127),'GEND'),var(47))),
  cf(1,eq(attr(var(127),'NTYPE'),var(48))),
  cf(1,eq(attr(var(127),'DEF'),'+')),
  cf(1,eq(attr(var(127),'NUM'),'sg')),
  cf(1,eq(attr(var(127),'PERS'),'3')),
  cf(1,eq(attr(var(127),'REF'),'+')),
  cf(1,eq(var(322),'obl')),
  cf(1,eq(var(128),var(46))),
  cf(1,eq(attr(var(128),'_ANTECED'),var(129))),
  cf(1,eq(attr(var(128),'_NE'),'+')),
  cf(1,eq(var(129),var(39))),
  cf(1,eq(attr(var(129),'NUM'),var(130))),
  cf(1,eq(attr(var(129),'PERS'),var(131))),
  cf(1,eq(var(130),var(40))),
  cf(1,eq(var(131),var(41))),
  cf(1,eq(var(323),var(325))),
  cf(1,eq(var(326),var(290)))
  ],
```

Listing A.2: Results of IB1 model in TiMBL

```
Examine datafile 'IdSubstArg1.csv' gave the following results:
Number of Features: 4
InputFormat       : C4.5

Phase 1: Reading Datafile: IdSubstArg1.csv
Start:          0 @ Tue Nov 10 19:09:30 2015
Finished:    4093 @ Tue Nov 10 19:09:30 2015
Calculating Entropy         Tue Nov 10 19:09:30 2015
Lines of data     : 4093
DB Entropy        : 8.1733532
Number of Classes : 1030

Feats Vals  InfoGain  GainRatio
    1 (ignored)
    2 (ignored)
    3    595  3.3810489 0.55539440
    4   1796  6.4095428 0.64932533

Preparation took 0 seconds, 24 milliseconds and 202 microseconds
Feature Permutation based on GainRatio/Values :
< 3, 4, 1, 2 >
Phase 2: Building multi index on Datafile: IdSubstArg1.csv
Start:          0 @ Tue Nov 10 19:09:30 2015
Finished:    4093 @ Tue Nov 10 19:09:30 2015

Phase 3: Learning from Datafile: IdSubstArg1.csv
Start:          0 @ Tue Nov 10 19:09:30 2015
Finished:    4093 @ Tue Nov 10 19:09:30 2015

Size of InstanceBase = 6409 Nodes, (256360 bytes), 26.51 % compression
Learning took 0 seconds, 45 milliseconds and 134 microseconds
Examine datafile 'testSet.csv' gave the following results:
Number of Features: 4
InputFormat       : C4.5


Starting to test, Testfile: testSet.csv
Writing output in:          testSet.csv.IB1.O:I1-2.gr.k1.out
Algorithm     : IB1
Global metric : Overlap
Deviant Feature Metrics:(none)
Ignored features : { 1, 2 }
Weighting     : GainRatio
Feature 1  : 0.000000000000000
Feature 2  : 0.000000000000000
Feature 3  : 0.555394399348072
Feature 4  : 0.649325325756339

Tested:      1 @ Tue Nov 10 19:09:30 2015
Tested:      2 @ Tue Nov 10 19:09:30 2015
Tested:      3 @ Tue Nov 10 19:09:30 2015
Tested:      4 @ Tue Nov 10 19:09:30 2015
Tested:      5 @ Tue Nov 10 19:09:30 2015
Tested:      6 @ Tue Nov 10 19:09:30 2015
Tested:      7 @ Tue Nov 10 19:09:30 2015
Tested:      8 @ Tue Nov 10 19:09:30 2015
Tested:      9 @ Tue Nov 10 19:09:30 2015
Tested:     10 @ Tue Nov 10 19:09:30 2015
Tested:    100 @ Tue Nov 10 19:09:30 2015
Ready:     466 @ Tue Nov 10 19:09:30 2015
Seconds taken: 0.4703 (990.87 p/s)

overall accuracy:        0.349785  (163/466), of which 89 exact matches
There were 107 ties of which 39 (36.45%) were correctly resolved
```

Listing A.3: Results of IGTREE model in TiMBL

```
Examine datafile 'IdSubstArg1.csv' gave the following results:
Number of Features: 4
InputFormat       : C4.5

Phase 1: Reading Datafile: IdSubstArg1.csv
Start:          0 @ Tue Nov 10 23:18:10 2015
Finished:    4093 @ Tue Nov 10 23:18:10 2015
Calculating Entropy         Tue Nov 10 23:18:10 2015
Lines of data     : 4093
DB Entropy        : 8.1733532
Number of Classes : 1030

Feats Vals  InfoGain  GainRatio
    1 (ignored)
    2 (ignored)
    3    595  3.3810489 0.55539440
    4   1796  6.4095428 0.64932533

Preparation took 0 seconds, 23 milliseconds and 231 microseconds
Feature Permutation based on GainRatio :
< 4, 3, 1, 2 >
Phase 2: Building multi index on Datafile: IdSubstArg1.csv
Start:          0 @ Tue Nov 10 23:18:10 2015
Finished:    4093 @ Tue Nov 10 23:18:10 2015

Phase 3: Learning from Datafile: IdSubstArg1.csv
Start:          0 @ Tue Nov 10 23:18:10 2015
Finished:    4093 @ Tue Nov 10 23:18:10 2015

Size of InstanceBase = 2606 Nodes, (104240 bytes), 70.12 % compression
Learning took 0 seconds, 46 milliseconds and 78 microseconds
Examine datafile 'testSet.csv' gave the following results:
Number of Features: 4
InputFormat       : C4.5


Starting to test, Testfile: testSet.csv
Writing output in:          testSet.csv.IGTree.gr.out
Algorithm     : IGTree
Ignored features : { 1, 2 }
Weighting     : GainRatio
Feature 1  : 0.000000000000000
Feature 2  : 0.000000000000000
Feature 3  : 0.555394399348072
Feature 4  : 0.649325325756339

Tested:      1 @ Tue Nov 10 23:18:10 2015
Tested:      2 @ Tue Nov 10 23:18:10 2015
Tested:      3 @ Tue Nov 10 23:18:10 2015
Tested:      4 @ Tue Nov 10 23:18:10 2015
Tested:      5 @ Tue Nov 10 23:18:10 2015
Tested:      6 @ Tue Nov 10 23:18:10 2015
Tested:      7 @ Tue Nov 10 23:18:10 2015
Tested:      8 @ Tue Nov 10 23:18:10 2015
Tested:      9 @ Tue Nov 10 23:18:10 2015
Tested:     10 @ Tue Nov 10 23:18:10 2015
Tested:    100 @ Tue Nov 10 23:18:10 2015
Ready:     466 @ Tue Nov 10 23:18:10 2015
Seconds taken: 0.0048 (96440.40 p/s)

overall accuracy:        0.396996  (185/466)
```

Listing A.4: Extract from the output of the AG30+GEND+ANI10 model

```
Gender doesn't match: 1777,han,oppsøke,pro,Sokrates,Sofie: Correct!Sokrates(290,2),menneske
    (235.000,4),Athen(80,4),filosof(75.0000,5),by(11.0000,0),ingen(4.00000,6),orakel
    (2.00000,3),øye(1.00000,0),

Gender doesn't match: 1778,han,stille,pro,Sokrates,Sofie: Correct!Sokrates(290,0),menneske
    (235.000,0),Athen(80,5),filosof(75.0000,6),spørsmål(26.0000,0),by(11.0000,1),noen
    (5.00000,0),ingen(4.00000,7),orakel(2.00000,0),øye(1.00000,1),

Gender doesn't match: 1780,han,mene,finne,Sokrates,Sofie: Correct!Sokrates(1620,1),Athen
    (790,7),menneske(286,0),filosof(263,8),fornuft(225,0),erkjennelse(81,1),by(61,3),Delfi
    (40,7),rett-right(27,2),ingen(25,9),noen(23,2),athener(12,7),viten(11,8),orakel(4,2),

Gender doesn't match: 1780,han,finne,fundament,Sokrates,Sofie: Athen(2810,7),Sokrates
    (1620,1),fornuft(324,0),filosof(286,8),menneske(286,0),Delfi(280,7),by(131,3),
    erkjennelse(111,1),noen(51,2),rett-right(32,2),ingen(25,9),viten(23,8),athener(13,7),
    orakel(11,2),

In Context: 1781,han,være,rasjonalist,Sokrates,menneske

Not in Context: 1784,han,si,epist-ville,Sokrates,Alberto: menneske(169,3),Sokrates(70,1),by
    (43,7),samvittighet(15,1),erkjennelse(13,5),tro(13,3),innsikt(12,2),rett-right(11,6),
    rasjonalist(9,3),handling(7,2),fornuft(6,3),noen(5,6),orakel(1,6),

Not in Context: 1785,han,mene,føre*til,Sokrates,Aristoteles: Correct!Sokrates(50,2),
    rasjonalist(2.00000,4),

Number of preds: 466
Correct antecedent in candidates: 101
Correctly classified: 262
0.562231759656652

Frequency of predictions:
Sofie, 241
menneske, 45
Sokrates, 45
Athen, 15
Thomas, 11
Aristoteles, 8
filosof, 7
Gud, 6
vann, 5
Tor, 5

Frequency of correct predictions:
Sofie, 199
Sokrates, 41
Demokrit, 4
Parmenides, 4
Tor, 3
Empedokles, 2
Thomas, 2
mann, 1
Heraklit, 1
Aristoteles, 1
Hermes, 1
```

Appendix B

Source code

Listing B.1: Lemma.pm

```
use utf8;

package My::Lemma;
use base qw(Class::Accessor);
__PACKAGE__->mk_accessors(qw(semform attribute var sentenceID similars frequency distance
    score));

sub print_info {
  my $self = shift;
  print $self->semform ."(" . $self->attribute . ")\n";
}

1;
```

Listing B.2: Predicate.pm

```
use utf8;
use Lemma;

package My::Predicate;
use base qw(Class::Accessor);
__PACKAGE__->mk_accessors(qw(pred arg1 arg2 sentenceID antecedent index));
sub print_info {
  my $self = shift;
  my $arg1 = $self->arg1->semform;
  my $arg2 = $self->arg2->semform;
  if ($self->arg1->attribute) {
    $arg1 = $self->arg1->semform . "(" . $self->arg1->attribute . ")";
  }
  if ($self->arg2->attribute) {
    $arg2 = $self->arg2->semform . "(" . $self->arg2->attribute . ")";
  }
  print $self->sentenceID . ": " . $self->pred . "," . $arg1 . "," . $arg2 . "\n";
}

sub get_pred {
  my $self = shift;
  return $self->pred . "," . $self->arg1->semform . "," . $self->arg2->semform;
}

sub get_ant {
  my $self = shift;
  return $self->pred . "," . $self->arg1->semform . "," . $self->arg2->semform . "," .
      $self->antecedent;
}

sub TO_JSON { return { %{ shift() } }; }

1;
```

Listing B.3: Sentence.pm

```
use utf8;
use Lemma;

package My::Sentence;
use base qw(Class::Accessor);
__PACKAGE__->mk_accessors(qw(sentence id lemmas));
sub print_info {
  my $self = shift;
  print "setningsnr " . $self->id .": " . $self->sentence ."\n";
}

sub get_lemmas {
  my $self = shift;
  return $self->lemmas;
}
1;
```

Listing B.4: predicateExtractor.pl

```
use strict;
use warnings;
use Tie::File;
use utf8;
use Data::Dumper;
use YAML qw(Bless DumpFile);
use List::MoreUtils qw(uniq);
use Predicate;
use Sentence;
use Lemma;

$Data::Dumper::Purity = 1;

my @allPreds;
my $dummyArg = My::Lemma->new();
$dummyArg->semform("?");

my $dir = $ARGV[0];
my $dumpfile = $ARGV[1];

opendir DIR, $dir or die "cannot open dir $dir: $!";


while( defined (my $file = readdir DIR)) {
  if ($file =~ m/\d/) {
    my @preds = extract_predicates("$dir/$file");
    if ($preds[0]) {
      foreach my $pred (@preds) {
        print $pred->get_pred . "\n";
        push (@allPreds, $pred);
      }
    }
  }

}

closedir DIR;


sub extract_predicates {

  my $sentenceID;
  my @predVars;
  our @vforms;
  my $find_semform;
  my @preds;
  our @evaluatedVars;

  my $file = shift;

  if ($file =~ m/.*-(\d+)-hr\.pl/) {
    $sentenceID = $1;
    #print "ID: " . $sentenceID . "\n";
  }

```

```
56  print $file . "\n";
57
58  open my $filehandle, $file
59    or die "Could not open input file!";
60
61  our @lines = <$filehandle>;
62
63
64
65  # Find verbs.
66  foreach my $line (@lines) {
67
68    if ($line =~ m/.*var\(([0-9]+)\).*'VFORM'.*/) {
69      my $predLine = $line . "\n";
70      my $verbVar = $1;
71      push (@vforms, $verbVar);
72    }
73  }
74
75  @vforms = uniq(@vforms);
76
77
78  # Find predicates
79  foreach my $var (@vforms) {
80    foreach my $line (@lines) {
81      if ($line =~ m/var\($var\).*'PRED'\),var\(([0-9]+)\)/) {
82        my $predVar = $1;
83        push (@predVars, $predVar);
84      }
85      if ($line =~ m/var\($var\),'PRED'\),semform/) {
86        push (@predVars, $var);
87      }
88    }
89  }
90
91  @predVars = uniq(@predVars);
92
93
94  # Iterate through predicates and locate semforms
95  foreach my $var (@predVars) {
96    #print "predvar: " . $var . "\n";
97    if (find_pred($var, $sentenceID)) {
98      #print find_pred($var, $sentenceID)->get_pred . "\n";
99      push (@preds, find_pred($var, $sentenceID));
100   }
101 }
102
103 close $filehandle;
104
105
106 sub find_word {
107   my $var = shift;
108   my $word = My::Lemma->new();
109
110   if ($var =~ m/[a-zæøåA-ZÆØÅ]+/) {
111     $word->semform($var);
112   } else {
113     $word = find_semform($var, 0, "");
114   }
115   if (!$word->semform) {
116     return $dummyArg;
117   }
118   return $word;
119 }
120
121 sub find_pred {
122   my $var = shift;
123   my $sentenceID = shift;
124   #print "predvar: " . $var . "\n";
125   my $pred = My::Predicate->new;
126   $pred->sentenceID($sentenceID);
127   foreach my $line (@lines) {
128     if ( $line =~ m/.*var\($var\).*semform\('(.*?)',\d+,\[('NULL',|)var\(([0-9]+)\).*?var
          \(([0-9]+)\)/){
129       #print "predvar: " . $var . ": ";
```

```
130         #print "1: " . $line;
131         $pred->pred($1);
132         $pred->arg1(find_word($3));
133         $pred->arg2(find_word($4));
134         return $pred;
135       }
136       elsif ( $line =~ m/.*var\($var\).*semform\('(.*?)',\d+,\[('NULL',|)var\(([0-9]+)\)
              \]/){
137         #print "predvar: " . $var . ": ";
138         #print "2: " . $line;
139         $pred->pred($1);
140         $pred->arg1(find_word($3));
141         $pred->arg2($dummyArg);
142         return $pred;
143       }
144       elsif ( $line =~ m/.*var\($var\).*semform\('(.*?)',\d+,\[\],\[('NULL',|)var\(([0-9]+)
              \)\]/){
145         #print "predvar: " . $var . ": ";
146         #print "3: " . $line;
147         $pred->pred($1);
148         $pred->arg1($dummyArg);
149         $pred->arg2(find_word($3));
150         return $pred;
151       }
152     }
153     return 0;
154   }
155
156   # Recursive subroutine to find the semforms of argument variables.
157   sub find_semform {
158
159     my $var = shift;
160     my $recursions = shift;
161
162
163     my $attribute = shift;
164     my $word = My::Lemma->new();
165     my $semform =  "?";
166     #print "semformvar: " . $var . "\n";
167     foreach my $line (@lines) {
168       if ($line =~ m/.*var\($var\).*'PRON-FORM'.*/) {
169         $attribute = "PRON";
170       } elsif ($line =~ m/.*var\($var\).*'VFORM'.*/) {
171         $attribute = "VERB";
172       } elsif ($line =~ m/.*var\($var\).*'NTYPE'.*var\((\d+)\).*/) {
173         $attribute = "SUBST";
174         my $ntypeVar = $1;
175         if (is_proper($ntypeVar)) {
176           $attribute = "PROPER";
177         }
178       }
179     }
180     foreach my $line (@lines) {
181       if ($recursions > 7) {
182         return $dummyArg;
183       }
184
185       if ( $line =~ m/.*var\($var\).*semform.*\('(.*?)',\d.*/) {
186         #print $line;
187         #print $1 . "\n";
188         $word->semform($1);
189         $word->attribute($attribute);
190         if ($1 eq "pro") {
191           #return find_pro_sem($var);
192           $word->attribute("PRON-");
193         }
194         if ($1 eq "hun" || $1 eq "han" || $1 eq "henne" || $1 eq "ham" ) {
195           $word->attribute("PRON+")
196         }
197       } elsif ($line =~ m/.*eq\(var\($var\),var\(([0-9]+)\).*,/) {
198         #print $var . "nopred: " . $1 . "\n";
199         #print $line;
200         #print $1 . "\n";
201         $word = find_semform($1, ++$recursions, $attribute);
202       } elsif ($line =~ m/.*attr\(var\($var\),'PRED'\),var\(([0-9]+)\).*,/) {
```

```
203         #print $var . "pred: " . $1 . "\n";
204         #print $line;
205         #print $1 . "\n";
206         $word =  find_semform($1, ++$recursions, $attribute);
207       } elsif ($line =~ m/.*in_set\(var\(\(\d+)\),var\($var\).*/) {
208         $word =  find_semform($1, ++$recursions, $attribute);
209       }
210     }
211     return $word;
212   }
213
214   sub is_proper {
215     my $var = shift;
216     foreach my $line (@lines) {
217       if ($line =~ m/.*var\($var\).*NSYN.*proper.*/) {
218         return 1;
219       }
220     }
221     return 0;
222   }
223
224   sub find_pro_sem {
225     my $var = shift;
226     my $word = My::Lemma->new();
227     $word->semform("lol");
228     my $var2;
229     my $var3;
230     my $spec;
231     my $specpred;
232
233     print "$var\n";
234
235     foreach my $line (@lines) {
236       if ($line =~ m/.*var\(([0-9]+)\).*var\($var\).*/) {
237         print "var: $1\n";
238         $var2 = $1;
239         foreach my $line (@lines) {
240           if ($line =~ m/.*var\($var2\).*NTYPE.*var\(([0-9]+)\).*/) {
241             #print $1 . "\n";
242           } elsif ($line =~ m/.*var\($var2\).*SPEC.*var\(([0-9]+)\).*/) {
243             #print "spec: $1\n";
244             $spec = $1;
245             foreach my $line (@lines) {
246               if ($line =~ m/.*eq.*var\($spec\).*var\(([0-9]+)\).*/) {
247                 #print "spec2: $1\n";
248                 $specpred = $1;
249               } elsif ($specpred && $line =~ m/.*eq.*var\($specpred\).*semform.*\('(.*?)'
                    .*/) {
250                 #print "$1\n";
251                 $word->semform($1);
252               }
253             }
254           }
255         }
256       }
257     }
258     return $word;
259   }
260
261   # remove duplicate preds
262   my @uniquePreds;
263   my $equal = 0;
264   foreach my $candidate (@preds) {
265     foreach my $current (@uniquePreds) {
266       if (equal_preds($candidate, $current)) {
267         $equal = 1;
268       }
269     }
270     if (!$equal) {
271       push (@uniquePreds, $candidate);
272     }
273   }
274   return @uniquePreds;
275
276 } #end extract_predicates
```

```
277
278 sub equal_preds {
279   $a = shift;
280   $b = shift;
281   if ($a->pred eq $b->pred && $a->arg1->semform eq $b->arg1->semform && $a->arg2->semform
          eq $b->arg2->semform) {
282     return 1;
283   }
284   return 0;
285 }
286
287
288
289 @allPreds = sort { $a->sentenceID <=> $b->sentenceID } @allPreds;
290
291 foreach my $pred (@allPreds) {
292   $pred->print_info;
293   #print Dumper ($pred);
294 }
295
296 DumpFile($dumpfile, @allPreds);
```

Listing B.5: sentenceExtractor.pl

```
1  use strict;
2  use warnings;
3  use Sentence;
4  use Lemma;
5  use utf8;
6  use Data::Dumper;
7  use YAML qw(Bless DumpFile);
8
9
10 my @sentences;
11 my $dir = $ARGV[0];
12 my $dumpfile = $ARGV[1];
13
14 opendir DIR, $dir or die "cannot open dir $dir: $!";
15
16 while( defined (my $file = readdir DIR)) {
17   if ($file =~ m/\d/) {
18     extract_sentences("$dir/$file");
19   }
20 }
21
22 sub extract_sentences {
23   my $sentenceID;
24   my $sentence = My::Sentence->new();
25   my @lemmas;
26
27   my $file = shift;
28
29   if ($file =~ m/.*-(\d+)-hr\.pl/) {
30     $sentenceID = $1;
31   }
32
33   open my $filehandle, $file
34     or die "Could not open input file!";
35
36   our @lines = <$filehandle>;
37
38   foreach my $line (@lines) {
39     # Extract sentece
40     if ($line =~ m/.*markup_free_sentence'\('(.*).\)/i) {
41       $sentence->sentence($1);
42       $sentence->id($sentenceID);
43     }
44     # Extract lemmas
45     if ($line =~ m/var\((\d+)\).*semform\('(.*?)'/) {
46       my $lemma = My::Lemma->new();
47       $lemma->var($1);
48       $lemma->semform($2);
49       $lemma->sentenceID($sentenceID);
50       push (@lemmas, $lemma);
51     }
```

```
  }

  $sentence->lemmas(@lemmas);
  if (!$sentence->id) {
    print "fil: " . $file . "\n";
  }

  push (@sentences, $sentence);
  $sentence->print_info;
}

@sentences = sort { $a->id <=> $b->id } @sentences;

DumpFile($dumpfile, @sentences);
```

Listing B.6: associate.pl

```
use strict;
use warnings;
use Tie::File;
use utf8;
use Predicate;
use YAML qw(DumpFile Dump Bless LoadFile);
use List::MoreUtils qw(uniq);
use Lemma;
use Data::Dumper;

# Variables
my $file = $ARGV[0];
my $dumpfile = $ARGV[1];
warn "Loading preds";
my @epas = LoadFile($file);
warn "Preds loaded";
my %args1;
my @args1;
my @expandedArg1s;

my $counter = 1;
my $currentArgNumber = 0;
if ($ARGV[2]) {
  $currentArgNumber = $ARGV[2];
}
$counter = $currentArgNumber +1;

# Extract unique first arguments
foreach my $epa (@epas) {
  if (!exists($args1{$epa->arg1->semform})) {
    push @args1, $epa->arg1;
  }
  $args1{$epa->arg1->semform}++;
}

my $numberOfArgs = scalar(@args1);


for (my $i = $currentArgNumber; $i < scalar(@args1); $i++) {
  my $arg1 = $args1[$i];
  warn "\n";
  warn "evaluating arg $counter of $numberOfArgs";
  if (($arg1->attribute eq "SUBST" || $arg1->attribute eq "PROPER") && $arg1->attribute ne
      "PRON+" && $arg1->attribute ne "PRON") {

    #print "arg1: " . $arg1->semform . "\n";
    my $argstring = "arg1: " . $arg1->semform . "\n";
    my $result = "";
    my $expandedArg = My::Lemma->new();

    # Find all epas with arg1
    my @epas_l1;
    my %epas_l1;
    foreach my $epa (@epas) {
      if ($epa->arg1->semform eq $arg1->semform) {
        push @epas_l1, $epa;
        $epas_l1{$epa->pred}++;
      }
```

```
58      }
59
60      # Find other words that are in the same position as arg1
61      my @similar_arg1;
62      my %similar_arg1;
63      foreach my $epa (@epas) {
64        foreach my $epa_l1 (@epas_l1) {
65          if ($epa->pred eq $epa_l1->pred &&
66              $epa->arg2->semform eq $epa_l1->arg2->semform &&
67              $epa->arg1->semform ne $arg1->semform &&
68              $epa_l1->arg2->semform ne "?") {
69            my $semform = $epa->arg1->semform;
70            if ($semform ne "pro" && $semform ne "DUMMY" && $epa->arg1->attribute ne "PRON"
                  && $epa->arg1->attribute ne "PRON+") {
71              $similar_arg1{$semform}++;
72              push @similar_arg1, $epa->arg1;
73            }
74            #print "\t\t: " . $epa->print_info . "\n";
75          }
76        }
77      }
78
79      foreach (keys %similar_arg1) {
80        foreach my $similar (@similar_arg1) {
81          if ($similar->semform eq $_) {
82            $similar->frequency($similar_arg1{$_})
83          }
84        }
85      }
86
87      @similar_arg1 = remove_duplicates(\@similar_arg1);
88      #@similar_arg1 = remove_pronouns(\@similar_arg1);
89
90      @similar_arg1 = sort { $b->frequency <=> $a->frequency } @similar_arg1;
91
92      foreach my $similar (@similar_arg1) {
93        if ($similar->frequency > 3) {
94          $result.= $similar->semform . "(" . $similar->frequency . "),";
95        }
96      }
97      if ($similar_arg1[0]) {
98        my $semformToCopy = $arg1->semform;
99        my $attributeToCopy = $arg1->attribute;
100       $expandedArg->semform($semformToCopy);
101       $expandedArg->similars(@similar_arg1);
102       $expandedArg->attribute($attributeToCopy);
103       #push @expandedArg1s, $expandedArg;
104       open my $output, '>>', $dumpfile
105         or die "Could not open $dumpfile";
106         print $output Dump($expandedArg);
107
108       close $output;
109     }
110
111     if ($result) {
112       $result .= "\n\n";
113       print $argstring . $result;
114     }
115
116   }
117   $counter ++;
118 }
119
120
121
122 sub get_lemmas {
123   my $lemmas_string = Dumper(shift);
124   $lemmas_string =~ s/\[/\(/ig;
125   $lemmas_string =~ s/\]/\)/ig;
126   $lemmas_string =~ s/\$VAR1 = //ig;
127   return eval $lemmas_string;
128 }
129
130
131 sub remove_duplicates {
```

```
132   my @candidates = @{$_[0]};
133   my @uniques;
134
135   foreach my $candidate (@candidates) {
136     my $candidateIsUnique = 1;
137     foreach my $unique (@uniques) {
138       if ($candidate->semform eq $unique->semform && $candidate->frequency == $unique->
              frequency) {
139         $candidateIsUnique = 0;
140         last;
141       }
142     }
143     if ($candidateIsUnique) {
144       push @uniques, $candidate;
145     }
146   }
147   return @uniques;
148 }
149
150 sub remove_pronouns {
151   my @candidates = @{$_[0]};
152   my @uniques;
153
154   foreach my $candidate (@candidates) {
155     if ($candidate->attribute eq "PRON" && $candidate->attribute eq "PRON+") {
156       warn "Pronomen!";
157     }
158     else {
159       push @uniques, $candidate;
160     }
161   }
162   return @uniques;
163 }
```

Listing B.7: guiAnnotator.pl

```
1  use strict;
2  use warnings;
3  use Predicate;
4  use Sentence;
5  use Lemma;
6  use Data::Dumper;
7  use YAML qw(DumpFile Bless LoadFile);
8  use Tk;
9  use utf8;
10 use Switch;
11 use Encode qw(decode encode);
12 use Text::Unidecode;
13
14 my $predfile = $ARGV[0];
15 my $sentencefile = $ARGV[1];
16 my $dumpfile = $ARGV[2];
17 my $lastPredPos = 0;
18
19
20 my @annotatedPreds;
21 print "Loading predicates...\n";
22 my @preds = LoadFile($predfile);
23 print "Predicates loaded\nLoading Sentences...\n";
24 my @sentences  = LoadFile($sentencefile);
25 print "Sentences loaded\n";
26 my $dummyLemma = My::Lemma->new();
27 $dummyLemma->semform("NA");
28
29 print "Total number of predicates: " . scalar(@preds) . "\n";
30
31 if (-e $dumpfile) {
32   @annotatedPreds = LoadFile($dumpfile);
33   $lastPredPos = $annotatedPreds[-1]->index;
34   print "Current predicate number: $lastPredPos \n"
35 }
36 if ($ARGV[3]) {
37 # $lastPredPos = $ARGV[3];
38 }
39
```

```
40
41
42  my $index = $lastPredPos;
43  for (my $i = $lastPredPos+1; $i < scalar(@preds); $i++) {
44    my $pred = $preds[$i];
45    if ($lastPredPos > 0) {
46      @annotatedPreds = LoadFile($dumpfile);
47    }
48    annotate ($pred, $i);
49
50  }
51
52  sub annotate {
53    my $pred = shift;
54    my $i = shift;
55    if (is_pron($pred)) {
56      my $sentenceID = $pred->sentenceID;
57      our $antecedent;
58      print "\n\n\n\n\n\n\n\n\n\n" . $pred->get_pred . "\n\n";
59      my $predSem = $pred->get_pred;
60      my @contextSentences;
61
62      $predSem = format_characters($predSem);
63
64      our $window = MainWindow->new;
65      $window->title("Antecedent Annotator");
66
67      my $predFrame = $window->Frame(-borderwidth => 2, -relief => 'groove');
68      my $sentencesFrame = $window->Frame(-borderwidth => 2, -relief => 'groove');
69      my $predLab = $predFrame->Label(-text => "Current predicate: $predSem")->pack();
70      $sentencesFrame -> grid(-row=>1,-column=>1,-columnspan=>2);
71      $predFrame -> grid(-row=>2,-column=>1,-columnspan=>2);
72
73      foreach my $sentence (@sentences) {
74        if ($sentenceID-9 == $sentence->id) {
75          push (@contextSentences, $sentence);
76        }
77        if ($sentenceID-8 == $sentence->id) {
78          push (@contextSentences, $sentence);
79        }
80        if ($sentenceID-7 == $sentence->id) {
81          push (@contextSentences, $sentence);
82        }
83        if ($sentenceID-6 == $sentence->id) {
84          push (@contextSentences, $sentence);
85        }
86        if ($sentenceID-5 == $sentence->id) {
87          push (@contextSentences, $sentence);
88        }
89        if ($sentenceID-4 == $sentence->id) {
90          push (@contextSentences, $sentence);
91        }
92        if ($sentenceID-3 == $sentence->id) {
93          push (@contextSentences, $sentence);
94        }
95        if ($sentenceID-2 == $sentence->id) {
96          push (@contextSentences, $sentence);
97        }
98        if ($sentenceID-1 == $sentence->id) {
99          push (@contextSentences, $sentence);
100       }
101       if ($sentenceID == $sentence->id) {
102         push (@contextSentences, $sentence);
103       }
104     }
105
106     @contextSentences = reverse @contextSentences;
107     my $counter = 0;
108     foreach my $contextSentence (@contextSentences) {
109       switch ($counter) {
110         case 0 {
111           my $button9 = $sentencesFrame->Button(-text => format_characters($contextSentence
                ->sentence),
112             -command => sub { list_lemmas($contextSentence); })->pack(-side => "bottom"
                  );
```

```
113         }
114         case 1 {
115           my $button8 = $sentencesFrame->Button(-text => format_characters($contextSentence
                ->sentence),
116             -command => sub { list_lemmas($contextSentence); })->pack(-side => "bottom"
                    );
117         }
118         case 2 {
119           my $button7 = $sentencesFrame->Button(-text => format_characters($contextSentence
                ->sentence),
120             -command => sub { list_lemmas($contextSentence); })->pack(-side => "bottom"
                    );
121         }
122         case 3 {
123           my $button6 = $sentencesFrame->Button(-text => format_characters($contextSentence
                ->sentence),
124             -command => sub { list_lemmas($contextSentence); })->pack(-side => "bottom"
                    );
125         }
126         case 4 {
127           my $button5 = $sentencesFrame->Button(-text => format_characters($contextSentence
                ->sentence),
128             -command => sub { list_lemmas($contextSentence); })->pack(-side => "bottom"
                    );
129         }
130         case 5 {
131           my $button4 = $sentencesFrame->Button(-text => format_characters($contextSentence
                ->sentence),
132             -command => sub { list_lemmas($contextSentence); })->pack(-side => "bottom"
                    );
133         }
134         case 6 {
135           my $button3 = $sentencesFrame->Button(-text => format_characters($contextSentence
                ->sentence),
136             -command => sub { list_lemmas($contextSentence); })->pack(-side => "bottom"
                    );
137         }
138         case 7 {
139           my $button2 = $sentencesFrame->Button(-text => format_characters($contextSentence
                ->sentence),
140             -command => sub { list_lemmas($contextSentence); })->pack(-side => "bottom"
                    );
141         }
142         case 8 {
143           my $button1 = $sentencesFrame->Button(-text => format_characters($contextSentence
                ->sentence),
144             -command => sub { list_lemmas($contextSentence); })->pack(-side => "bottom"
                    );
145         }
146         case 9 {
147           my $button0 = $sentencesFrame->Button(-text => format_characters($contextSentence
                ->sentence),
148             -command => sub { list_lemmas($contextSentence); })->pack(-side => "bottom"
                    );
149         }
150       }
151       $counter++;
152     }
153     my $button = $sentencesFrame -> Button(-text => "No match", -command => sub {
154         $antecedent = $dummyLemma;
155         $window->destroy;
156 
157 
158       }) -> pack();
159 
160     MainLoop;
161 
162     sub list_lemmas {
163       my $sentence = shift;
164       warn $sentence->sentence;
165 
166       my @lemmas = get_lemmas($sentence->lemmas);
167       my @lemmaSems;
168       my $selectedLemma;
169       foreach my $lemma (@lemmas) {
```

```
        push (@lemmaSems, format_characters($lemma->semform));
        print $lemma->semform . ",";
      }

      print "\n";

      my $lemmasFrame = $window->Frame();
      $lemmasFrame -> grid(-row=>4,-column=>1,-columnspan=>2);
      my $lemmaLab = $lemmasFrame->Label(-text => "Select antecedent")->pack();

      warn "frame created";

      # Create Listbox and insert the list of choices into it
      my $lb = $lemmasFrame->Listbox(-selectmode => "browse")->pack(-side => "left");
      warn "listbox created";
      $lb->insert("end", @lemmaSems);
      warn "lemmas inserted";

      $lb -> bind('<<ListboxSelect>>'=> sub {
                my $selected_lemma =  $_[0]->get($_[0]->curselection);
                warn "lemma selected";
                foreach my $lemma (@lemmas) {
                  if (format_characters($lemma->semform) eq $selected_lemma) {
                    $selectedLemma = $lemma;
                  }
                }
            print $selectedLemma->semform . "\n\n";
            $antecedent = $selectedLemma;
            $lemmasFrame->destroy;
            $window->destroy;
                },
              );

      my $button = $lemmasFrame -> Button(-text => "No match", -command => sub {
        $antecedent = $dummyLemma;
        $lemmasFrame->destroy;
        $window->destroy;


      }) -> pack();

    }


    $pred->index($i);
    my @antecedents;
    push (@antecedents, $antecedent);
    $pred->antecedent(@antecedents);
    #print $pred->get_pred . Dumper $pred->antecedent . "\n\n\n\n\n";
    print $pred->get_ant;
    push (@annotatedPreds, $pred);
    DumpFile($dumpfile, @annotatedPreds);
  }
}




sub is_pron {
  my $pred = shift;
  if ($pred->arg1->attribute) {
    if ($pred->arg1->attribute eq "PRON+") {
      return 1;
    }
  }
  if ($pred->arg2->attribute) {
    if ($pred->arg2->attribute eq "PRON+") {
      return 1;
    }
  }
  return 0;
}

sub get_lemmas {
```

```
  my $lemmas_string = Dumper(shift);
  $lemmas_string =~ s/\[/\(/ig;
  $lemmas_string =~ s/\]/\)/ig;
  $lemmas_string =~ s/\$VAR1 = //ig;
  return eval $lemmas_string;
}

sub split_input {
  my $input = shift;
  chomp $input;
  my @numbers;
  if ($input =~ m/([0-9]+),([0-9]+)*/){
    push (@numbers, $1,$2);
  } elsif ($input =~ m/\d+/) {
    push (@numbers, $input);
  } else {
    return 0;
  }
  return @numbers;
}

sub format_characters {
  my $string = shift;
  $string = unidecode($string);
  $string =~ s/A\|/æ/g;
  $string =~ s/AY=/å/g;
  $string =~ s/A,/ø/g;
  $string =~ s/a\|/\.\.\./g;
  $string =~ s/A<</«/g;
  $string =~ s/A>>/»/g;
  return $string;
}
```

Listing B.8: fullModel.pl

```
use strict;
use warnings;
use utf8;
use YAML qw(DumpFile Bless LoadFile);
use List::MoreUtils qw(uniq);
use Predicate;
use Lemma;
use Data::Dumper;
use Sentence;

# Import result from IB1 algorithm
my $classifiedFile = $ARGV[0];
open my $classifiedfilehandle, $classifiedFile
    or die "Could not open input file!";
my @classifiedLines = <$classifiedfilehandle>;

# Import gender tagged names
open my $namesfilehandle, 'no-first-names.txt'
    or die "Could not open input file!";
my @names = <$namesfilehandle>;
my %names;
foreach my $nameLine (@names) {
  my @nameSplit = split " ", $nameLine;
  my $isMale = 1;
  if ($nameSplit[1] eq "+Fem" ) {
    $isMale = 0;
  }
  $names{$nameSplit[0]} = $isMale;
}

# Import ontology model
warn "loading args";
my $argsFile = $ARGV[1];
my @arg1s = LoadFile($argsFile);
warn "args loaded";

# Import sentences
warn "loading sentences";
my $sentencesFile = $ARGV[2];
my @sentences = LoadFile($sentencesFile);
```

```
41  warn "sentences loaded";
42
43  # Import cutoff value
44  my $cutoff = $ARGV[3];
45
46  # Field variables
47  my $correctlyClassified = 0;
48  my $classInCandidates = 0;
49  my @classifications;
50  my %classifications;
51  my %correctClassifications;
52
53
54  # Main loop. Evaluates all instances in the test set
55  foreach my $line (@classifiedLines) {
56
57    # Extract nearest neighbour distribution and probability for predicted class
58    my @argsAndDistribution = split " {", $line;
59    my @distribution = get_distribution($argsAndDistribution[1]);
60    my $probability = pop @distribution;
61    @distribution = sort { $b->frequency <=> $a->frequency } @distribution;
62
63    # Extract predicate, actual class and predicted class.
64    my @args = split ",", $argsAndDistribution[0];
65    my $id = $args[0];
66    my $pred = $args[2];
67    my $arg2 = $args[3];
68    my $class = $args[4];
69    my $predicted = $args[5];
70    my $genderString = $args[1];
71    my $isMale = 1;
72    if ($genderString eq "hun" || $genderString eq "henne" ) {
73      $isMale = 0;
74    }
75
76    # Load anaphor context
77    my @contextLemmas = get_context($id, 9);
78    my @minimalContextLemmas = get_context($id, 9);
79
80    my @candidates;
81    my @distCandidates;
82
83    # If predicted class is not in context, iterate through the distribution
84    if (!in_context($predicted, \@minimalContextLemmas) || !gender_matches($predicted,
          $isMale)) {
85
86      my $genderNotMatching = 0;
87      if (!gender_matches($predicted, $isMale)) {
88        $genderNotMatching = 1;
89        print "Gender doesn't match: $id,$genderString,$pred,$arg2,$class,$predicted: ";
90      } else {
91        print "Not in Context: $id,$genderString,$pred,$arg2,$class,$predicted: ";
92      }
93
94      # If number of nearest neighbours is below cutoff value, add similar words as
           candidates
95      @distCandidates = @distribution;
96      if (scalar(@distribution) < $cutoff ) {
97        my @similars = search_args($predicted);
98        if ($distribution[0]) {
99          foreach my $dist (@distribution) {
100           push @similars, search_args($dist->semform);
101         }
102       }
103       foreach my $sim (@similars) {
104         if ($sim) {
105           my $attribute = $sim->attribute;
106           if (($attribute eq "SUBST" || $attribute eq "PROPER") && $sim->semform ne
                $predicted) {
107             push @distCandidates, $sim;
108           }
109         }
110       }
111     }
112
```

```
113
114     # Located candidates that are in the context
115     if ($distCandidates[0]) {
116       my $numberOfCandidates = 5;
117       my $counter = 0;
118       foreach my $distCandidate (@distCandidates) {
119         foreach my $contextLemma (@contextLemmas) {
120           if ($contextLemma && $distCandidate) {
121             if ($contextLemma->semform eq $distCandidate->semform) {
122               $distCandidate->sentenceID($contextLemma->sentenceID);
123               $distCandidate->attribute($contextLemma->attribute);
124               push @candidates, $distCandidate;
125             }
126           }
127         }
128       }
129     }
130
131
132     if ($candidates[0]) {
133
134       # Update the score of each candidate according the distance and animacy heuristic
135       foreach my $candidate (@candidates) {
136         my $distance = $id - $candidate->sentenceID;
137         $candidate->distance($distance);
138         my $score = $candidate->frequency;
139         #my $score = $candidate->frequency / (($distance + 1)*20);
140         if ($candidate->semform =~ /^[A-ZÆØÅ].+/) {
141           $score = $score*10;
142         }
143         $candidate->score($score);
144       }
145
146       # Sort the candidates by score, descending
147       @candidates = sort {  $b->score <=> $a->score } @candidates;
148
149       # Best candidate is the one with the highest score
150       my $bestCandidate = $candidates[0];
151
152       # If gender of anaphor is incongruent with candidate, select the next candidate.
153       if ($genderNotMatching && !gender_matches($bestCandidate->semform, $isMale)) {
154         foreach my $candidate (@candidates) {
155           if (gender_matches($candidate->semform,$isMale)) {
156             $bestCandidate = $candidate;
157             last;
158           }
159         }
160       }
161
162       # Add candidate to hash of classifications
163       $classifications{$bestCandidate->semform}++;
164
165       # If the best candidate is identical to the annotated antecedent, it is counted as a
             correct classification
166       my $correct = 0;
167       if ($bestCandidate->semform eq $class) {
168         $correctlyClassified ++;
169         print "Correct!";
170         $correct = 1;
171         $correctClassifications{$bestCandidate->semform}++
172       }
173
174       # If the annotated candidate is found in the context, the classification is counted
             as a partially correct classification
175       my @uniqueCandidates = remove_duplicates(\@candidates);
176       foreach my $candidate (@uniqueCandidates) {
177         print $candidate->semform . "(" . $candidate->score . "," . $candidate->distance ."
               ),";
178         if ($candidate->semform eq $class && !$correct) {
179           $classInCandidates ++;
180         }
181       }
182     }
183
184
```

```
185   }
186
187   # If predicted class was found in context, compare it to the annotated antecedent to see
          if it is correct.
188   else {
189     print "In Context: $id,$genderString,$pred,$arg2,$class,$predicted";
190     if ($predicted eq $class) {
191       $correctlyClassified ++ ;
192       $correctClassifications{$predicted}++
193     }
194     $classifications{$predicted}++;
195   }
196
197   print "\n\n";
198
199 }
200
201 # Calculate and print statistics
202 print "Number of preds: " . scalar(@classifiedLines) . "\n";
203 print "Correct antecedent in candidates: $classInCandidates\n";
204 print "Correctly classified: $correctlyClassified\n";
205 print $correctlyClassified / scalar(@classifiedLines) . "\n";
206 print "\n";
207
208 print "Frequency of predictions: \n";
209 my @classes = sort { $classifications{$b} <=> $classifications{$a} } keys(%classifications)
        ;
210 my $counter = 0;
211 foreach my $key (@classes) {
212   print $key . ", " . $classifications{$key} . "\n";
213   if ($counter >=9) {last;}
214   $counter ++;
215 }
216
217 print "\nFrequency of correct predictions: \n";
218 my @corrects = sort { $correctClassifications{$b} <=> $correctClassifications{$a} } keys(%
        correctClassifications);
219 $counter = 0;
220 foreach my $key (@corrects) {
221   print $key . ", " . $correctClassifications{$key} . "\n";
222   if ($counter >9) {last;}
223   $counter ++;
224 }
225
226 # Submethod for finding similar words in ontology model
227 sub search_args {
228   my $predicted = shift;
229   foreach my $arg1 (@arg1s) {
230     if ($predicted eq $arg1->semform) {
231       my @similars = get_lemmas($arg1->similars);
232       @similars = sort { $b->frequency <=> $a->frequency } @similars;
233       return @similars
234     }
235   }
236   return 0;
237 }
238
239 # Submethod for finding the context of a given instance
240 sub get_context {
241   my $sentenceID = shift;
242   my $size = shift;
243   my @contextSentences;
244   my @contextLemmas;
245
246   foreach my $sentence (@sentences) {
247     for (my $i = $size; $i >=0; $i--){
248       if ($sentenceID-$i == $sentence->id) {
249         push (@contextSentences, $sentence);
250       }
251     }
252   }
253
254   my $counter = 0;
255   foreach my $sentence (@contextSentences) {
256     my @lemmas = get_lemmas($sentence->lemmas);
```

```
257     my @uniqueLemmas;
258     my $duplicate = 0;
259     foreach my $lemma (@lemmas) {
260       $lemma->sentenceID($sentence->id);
261       foreach my $uniqueLemma (@uniqueLemmas) {
262         if ($uniqueLemma->semform eq $lemma->semform) {
263           $duplicate = 1;
264         }
265       }
266       do push @uniqueLemmas, $lemma if (!$duplicate);
267       $duplicate = 0;
268     }
269     push @contextLemmas, @uniqueLemmas;
270   }
271
272   return @contextLemmas
273 }
274
275 # Submethod returns true if a lemma is found within a given context
276 sub in_context {
277   my $semform = shift;
278   my @context = @{$_[0]};
279   foreach my $contextLemma (@context) {
280     if ($contextLemma) {
281       if ($contextLemma->semform eq $semform) {
282         return 1
283       }
284     }
285   }
286   return 0;
287 }
288
289 # Submethod for extracting lemmas from the Sentence.pm class
290 sub get_lemmas {
291   my $lemmas_string = Dumper(shift);
292   $lemmas_string =~ s/\[/\(/ig;
293   $lemmas_string =~ s/\]/\)/ig;
294   $lemmas_string =~ s/\$VAR1 = //ig;
295   return eval $lemmas_string;
296 }
297
298 # Submethod for parsing output from the IB1 model
299 sub get_distribution {
300   my $line = shift;
301   my @candidates;
302   my $probability;
303   my @distribution = split ", ", $line;
304   my @candidateAndProbability = split " }", pop @distribution;
305
306   # Extract the probability for the predicted class
307   if ($candidateAndProbability[1] =~ /.*(\d+\.\d+)/m) {
308     $probability = $1;
309   }
310
311   push @distribution, $candidateAndProbability[0];
312
313   foreach my $candidateString (@distribution) {
314     my @candidateAndScore = split " ", $candidateString;
315     my $candidate = My::Lemma->new();
316     $candidate->semform($candidateAndScore[0]);
317     $candidate->frequency($candidateAndScore[1]);
318     if ($candidate->semform =~ /[a-zæøåA-ZÆØÅ]+/m) {
319       push @candidates, $candidate;
320     }
321   }
322
323   # put probability for predicted class at end of @candidates array.
324   push @candidates, $probability;
325
326   return @candidates;
327 }
328
329 # Submethod for removing duplicate candidates
330 sub remove_duplicates {
331   my @candidates = @{$_[0]};
```

```
332   my @uniques;
333
334   foreach my $candidate (@candidates) {
335     my $candidateIsUnique = 1;
336     foreach my $unique (@uniques) {
337       if ($candidate->semform eq $unique->semform && $candidate->sentenceID == $unique->
              sentenceID) {
338         $candidateIsUnique = 0;
339         last;
340       }
341     }
342     if ($candidateIsUnique) {
343       push @uniques, $candidate;
344     }
345   }
346   return @uniques;
347 }
348
349 # Submethod returns false if the given lemma matches a name in the names list, but not the
      gender. True otherwise
350 sub gender_matches {
351   my $semform = shift;
352   my $isMale = shift;
353
354   # Return true if semform is not a proper name
355   if ($semform =~ /^[a-zæøå].+/) {
356     return 1;
357   }
358
359   # Return false if gender doesn't match
360   if (exists $names{$semform} ) {
361     if ($names{$semform} != $isMale) {
362       return 0;
363     }
364   }
365   return 1;
366 }
```

Appendix C

Additional data

The training set, test set as well as outputs from all the models described in section 3.12 can be downloaded via the following url:

`https://www.dropbox.com/sh/0hnus89vnf3fjr9/AADd8dutvwze3dlg8JT02RfCa?dl=0`