Master Thesis - Applied Computer Science
Albert-Ludwigs-Universität Freiburg im Breisgau

Contextual Sentence Decomposition with Applications to Semantic Full-Text Search

Elmar Haussmann

July 14th, 2011

Albert-Ludwigs-Universität Freiburg im Breisgau
Faculty of Engineering
Department of Computer Science
Supervisor Prof. Dr. Hannah Bast

Supervisor
Prof. Dr. Hannah Bast

Primary Reviewer
Prof. Dr. Hannah Bast

Secondary Reviewer
Prof. Dr. Martin Riedmiller

Date
July 14th, 2011

Declaration

I hereby declare, that I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work. I hereby also declare, that my thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Freiburg, July 14th, 2011

Signature

Abstract

In this thesis, we introduce and study *contextual sentence decomposition*, which, intuitively, decomposes a given sentence into parts that semantically "belong together". For example, a valid decomposition of the sentence "*Usable parts of rhubarb include the edible stalks and the medicinally used roots, however its leaves are toxic*" are the *sub-sentences* "*Usable parts of rhubarb include the edible stalks",* "*Usable parts of rhubarb include the edible stalks*" and *"however its leaves are toxic*".

Our motivation for this problem comes from *semantic full-text search.* For a query *plant edible leaves,* semantic full-text search returns passages where instances of a plant, such as "*rhubarb*" (and not the word *"plant*"), are mentioned along with the words "*edible*" and *"leaves*". One of the results this query might erroneously return is the original sentence above. With contextual sentence decomposition we avoid this false-positive, while at the same time maintaining the true factual contents of the original sentence.

We propose two approaches for our problem, one based on a set of rules and one using machine learning. On a manually assembled ground truth, we achieve an F-measure of about 65 percent for the former and of 40 percent for the latter. For the semantic full-text search based on these approaches, evaluated on the English Wikipedia (27 GB of raw text), we achieve improvements nearly doubling the F-measure for some queries.

Zusammenfassung

Wir präsentieren und untersuchen *Contextual Sentence Decomposition.* Intuitiv geht es dabei um die Zerlegung eines Satzes in Teile, die „inhaltlich zusammengehören“. Zum Beispiel besteht eine gültige Zerlegung des Satzes „*Usable parts of rhubarb include the edible stalks and the medicinally used roots, however its leaves are toxic*“ aus den *sub-sentences „Usable parts of rhubarb include the edible stalks“, „Usable parts of rhubarb include the edible stalks“ und „however its leaves are toxic“.*

Die Motivation für dieses Problem hat ihren Ursprung in der semantischen Volltext-Suche. Für eine Anfrage *plant edible leaves,* zum Beispiel, findet diese Textstellen in denen eine Pflanze, wie „*rhubarb*“ (und nicht das Wort „*plant*“), zusammen mit den Wörtern „*edible*“ und „*leaves*“ vorkommen. Eines der Ergebnisse, das diese Anfrage fälschlicherweise zurückgeben könnte, ist der ursprüngliche Satz oben. Mit Hilfe von Contextual Sentence Decomposition vermeiden wir dieses false-positive und behalten gleichzeitig den richtigen, faktischen Inhalt des ursprünglichen Satzes bei.

Wir präsentieren zwei Ansätze für dieses Problem, einen basierend auf einer Menge an Regeln und einen, der maschinelles Lernen verwendet. Auf einer händisch erstellten Ground Truth erzielen wir ein F-measure von 65 Prozent für den ersten, und 40 Prozent für den zweiten Ansatz. Für die auf diesen Ansätzen basierende semantische Volltext-Suche, die auf der englischen Wikipedia (27GB reiner Text) untersucht wurde, erzielen wir eine Verbesserung, die bis zur annähernden Verdopplung des F-measures für einige Anfragen reicht.

Acknowledgments

First and foremost, I want to thank my supervisor Prof. Dr. Hannah Bast for providing me with guidance and support as well as a countless number of ideas and suggestions throughout this thesis. Working with her is a privilege and I very much appreciate and enjoy it. Clearly, this thesis would not have been possible without her.

I also want to thank my colleague Björn Buchhold for the many fruitful discussions over countless amounts of coffee. I very much enjoy our inspiring collaboration.

Furthermore, I owe my gratitude to all who helped a lot by proofreading this document, especially Alexander Gutjahr and Björn Buchhold.

Last but not least, I want to thank my girlfriend, Simonette, who had to endure me and my lack of time during the last months. I cannot imagine having accomplished this without her.

Contents

List of Tables

1 Introduction

In this thesis, we introduce and study the problem of contextual sentence decomposition that occurs as an important sub-task of *semantic full-text search.* To understand the problem and its motivation we first require an understanding of semantic full-text search.

Consider the task of composing a list of all *plants that have edible leaves.* A usual approach is to perform a search on a large document collection, such as the English Wikipedia, by formulating a keyword query like *plant edible leaves* and returning passages in which all the words occur. Though feasible, the results would not be satisfying, because we are not interested in the occurrences of the mere word "*plant",* but rather we are looking for instances of it such as "*rhubarb*", "*lemon*" or "*shallot*".

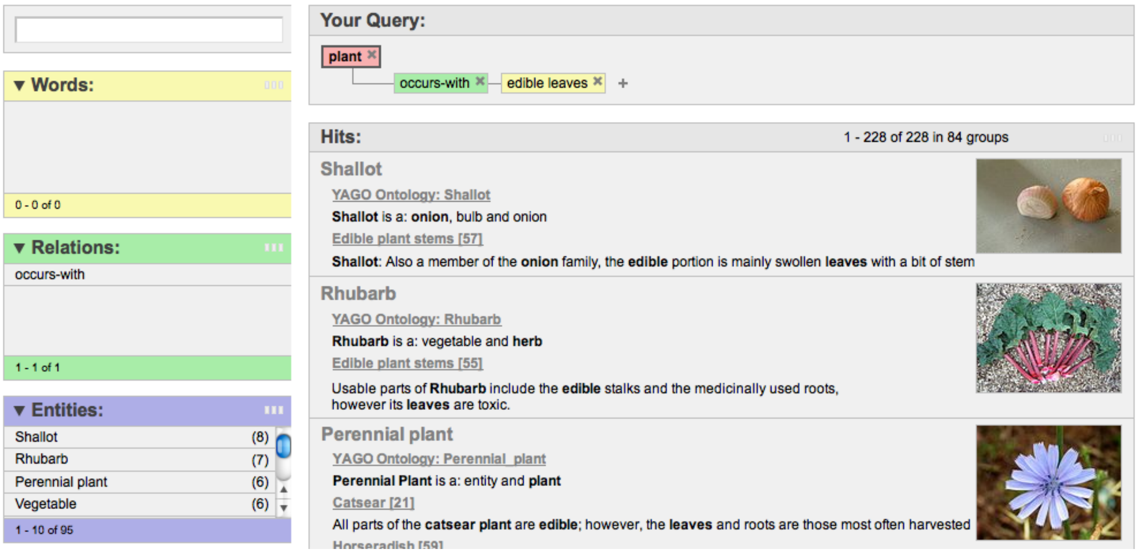
For the same query, semantic full-text search is able to return sentences that contain an instance of a plant, for example "*shallot*", along with the words "*edible*" and "*leaves*". Figure 1.1 shows a screenshot of the query running against the semantic full-text search engine prototype Broccoli[1].

One of the results returned is "*Shallot: Also a member of the onion family, the edible portion is mainly swollen leaves with a bit of stem*", because "*Shallot*" represents an instance of a plant and it occurs with the words "*edible*" and "*leaves*".

Instead of relying only on full-text, semantic full-text search combines it with structured knowledge based on an ontology that provides information about entities and their relations. For example, this includes the facts that "*rhubarb*" and "*shallot*" are plants.

With the above being a satisfactory result, another result that is returned by the query is "*Usable parts of rhubarb include the edible stalks and the medicinally used roots, however its leaves are toxic*". Clearly the words "*rhubarb*", "*edible*" and "*leaves*" occur in the same sentence, but nonetheless the result is obviously wrong because the leaves of rhubarb are in fact toxic. One can conclude that the words "*rhubarb*", "*edible*" and "*leaves*" do not "*belong together*" - they are part of different factual information and appear in different contexts. To remedy this, some understanding of natural language is required, namely understanding that the words of each of the passages *"Usable parts of rhubarb include the edible stalks"*, *"Usable parts*

[1]Broccoli is ongoing work at the Chair for Algorithms and Data Structures of the University of Freiburg. It is based on SUSI(Wikipedia Search Using Semantic Index Annotations) [Buchhold (2010)] and CompleteSearch [Bast and Weber (2007)] and uses a new user interface [Baeurle (2011)].


▼ Words:

0 - 0 of 0

▼ Relations:

occurs-with

1 - 1 of 1

▼ Entities:

Shallot	(8)
Rhubarb	(7)
Perennial plant	(6)
Vegetable	(6)

1 - 10 of 95

Your Query:

plant ✖ occurs-with ✖ edible leaves ✖ +

Hits: 1 - 228 of 228 in 84 groups

Shallot

YAGO Ontology: Shallot

Shallot is a: **onion**, bulb and onion

Edible plant stems [57]

Shallot: Also a member of the **onion** family, the **edible** portion is mainly swollen **leaves** with a bit of stem

Rhubarb

YAGO Ontology: Rhubarb

Rhubarb is a: vegetable and **herb**

Edible plant stems [55]

Usable parts of **Rhubarb** include the **edible** stalks and the medicinally used roots, however its **leaves** are toxic.

Perennial plant

YAGO Ontology: Perennial_plant

Perennial Plant is a: entity and **plant**

Catsear [21]

All parts of the **catsear plant** are **edible**; however, the **leaves** and roots are those most often harvested

Horseradish [59]

Figure 1.1: Screenshot of the semantic full-text search prototype Broccoli. The screenshot shows the query for *plants with edible leaves.*

of rhubarb include the medicinally used roots" and "*however its leaves are toxic*" belong to the same context, but that besides that, no other words are part of a same context. If the original search is restricted to each of these passages we can eliminate the false result and provide a semantically just search. This thesis is concerned with the natural language processing required for decomposing a sentence in the just described fashion.

Consider another result that is returned by the query: "*Broccoli: the edible portion is stem tissue, flower buds, and some small leaves*". This is an acceptable result because broccoli is indeed a plant and the words "*Broccoli*", "*edible*" and "*leaves*" appear in the same context. However, *"Bananas are eaten deep fried, or steamed in glutinous rice, which is wrapped in a banana leaf"* is not an acceptable result, because the words "*Bananas*", "*eaten*"[2] and "*leaf*" do not belong together. To provide the intuition on the required natural language processing we can identify several parts in the first sentence: *"Broccoli: the edible portion is*", "*stem tissue*", "*flower buds*" and *"some small leaves"*. We can recombine these to the passages "*Broccoli: the edible portion is stem tissue*", *"Broccoli: the edible portion is flower buds*" and "*Broccoli: the edible portion is some small leaves*". For the second sentence one can then identify the parts "*Bananas are eaten*", "*deep fried*", "*steamed in glutinous rice*" and "*which is wrapped in a banana leaf*" and recombine these parts to the passages "*Bananas are eaten deep fried*", "*Bananas are steamed in glutinous rice*" and "*glutinous rice, which is wrapped in a banana leaf*". If we then consider the results of our intuitive process separately it becomes clear that pre-processing sentences this way and applying the

[2]In this case we assume that the search query was formulated in such a way that results contain the word "*edible*" or the word "*eaten*".

same search strategy will eliminate the wrong search results but keep the correct ones.

This thesis provides a definition, implementation and analysis of the intuitive process just described. The overall process[3], contextual sentence decomposition, consists of the two tasks *sentence constituent identification (SCI)* to identify certain parts of a sentence and *sentence constituent recombination (SCR)* to recombine these parts. The result is then a set of *sub-sentence*s, where each sub-sentence consists of words originating from the same factual context and therefore represents some factual information. By applying contextual sentence decomposition as a pre-processing step we aim at providing the semantic full-text search capabilities described above. To the best of our knowledge this has not been studied before.

Our goal is two-fold. On the one hand we want to find a viable and efficient approach for an implementation. On the other hand we want to evaluate how well contextual sentence decomposition works in terms of improving search quality. Although the decomposition can be performed as a pre-processing step, the amount of full-text data is huge, e.g. 27GB of raw text for the current English Wikipedia, and thus the approach must also allow an efficient implementation. This rules out approaches performing deep natural language processing, for example the construction complete syntactic parse trees. The approaches we provide work on a more shallow and efficient level of natural language processing.

The next section outlines the main contributions of this thesis and section 1.2 then describes the structure of this thesis.

1.1 Contributions

The following are the main contributions which are part of this thesis:

- The introduction and definition of a new problem: contextual sentence decomposition consisting of the sub-tasks sentence constituent identification and sentence constituent recombination
- Two approaches to and implementations of SCI; one based on a set of manually crafted rules and one based on Machine Learning techniques, utilizing Support Vector Machines and an inference algorithm
- An implementation of SCR to perform contextual sentence decomposition by combining it with an SCI approach from above
- The incorporation of contextual sentence decomposition into a semantic search engine to provide semantic full-text search

[3]We deliberately decided to formulate CSD as a (computational) process (instead of as a problem), and in the following also refer to it as such. The definition as a process proved to be more intuitive and flexible.

- An extensive evaluation and comparison of the different approaches to SCI, their influence on the resulting contextual sentence decomposition and overall search quality

1.2 Thesis Structure

The rest of this thesis is structured as follows.

- Chapter 1: **Introduction** provided the motivation for this thesis, gave some intuition on the tasks involved for contextual sentence decomposition and outlined our main contributions.
- Chapter 2: **Related Work** gives an overview of other work and research areas that are in one way or another related to this thesis. We argue why other approaches in natural language processing (by themselves) are not sufficient for what we are trying to achieve.
- Chapter 3: **Problem Definition** provides a detailed problem statement including a formal definition of contextual sentence decomposition and its subtasks sentence constituent identification and sentence constituent recombination, as well as various examples.
- Chapter 4: **Sentence Constituent Identification Using Rules** describes the first approach to SCI. We present some manually crafted rules and provide pseudo-code for the relevant parts that describe their implementation on a high level.
- Chapter 5: **Sentence Constituent Identification Using Machine Learning** describes the second, alternative approach to SCI. We show how Machine Learning techniques together with an inference algorithm can be applied to solve this problem.
- Chapter 6: **Semantic Wikipedia Full Text Search** provides a description of how we integrate contextual sentence decomposition with an existing search engine in order to obtain a fully functional semantic full-text search engine benefiting from CSD.
- Chapter 7: **Evaluation** compares the two approaches to sentence constituent identification both with respect to their direct results and their influence on the overall results for contextual sentence decomposition. Finally we also provide an analysis of the qualitative improvements of semantic full-text incorporating CSD.
- Chapter 8: **Discussion** sums up our contributions, points out shortcomings and outlines further research directions.

2 Related Work

In this chapter we introduce some work and research areas that are related to the topics of this thesis. To the best of our knowledge the exact goal we are pursuing is unique and not standard in natural language processing. Nonetheless, there are some research areas that in one way or the other relate to the tasks at hand and are worth mentioning.

2.1 Text Simplification

One of the areas closely related to contextual sentence decomposition is *Text Simplification*, also referred to as S*yntactic Simplification.* As the name implies it describes the process of simplifying text using basically two different approaches [Siddharthan (2003)]:

- Reducing lexical complexity, for example by exchanging infrequently used words against well known synonyms.
  An example is exchanging the word "*legible*" with "*readable*" [Siddharthan (2003)].
- Reducing syntactical complexity, for example by splitting long sentences into several smaller sentences that are easier to understand.
  An example is simplifying "*...extracted the £75 on-the-spot cash fine which has outraged him and other clamped motorists...*" to "*... extracted the £75 on-the-spot cash fine. It has shocked him and other clamped motorists...*" [Siddharthan (2003)].

According to [Chandrasekar et al. (1996)], one of the first works dedicated to the topic, Text Simplification is based on mainly two motivations. On the one hand the simplified text is easier to read and understand for humans with reading disabilities, such as aphasics. On the other hand it can provide a pre-processing step for parsers, because parsing shorter sentences is computationally cheaper. The parses of the smaller sentences can then be combined to gain the parse of the original full sentence. Usually the result of Text Simplification is some form of a "simple sentence" that is defined using a set of constraints. For example, an *Easy Access Sentence* [Klebanov et al. (2004)] has to be grammatical, can contain only one finite verb and must adhere to the semantics of the original sentence. This shall guarantee that the result remains readable for humans, at the same time preserving meaning and informational content.

Note that for a human the ordering of the resulting sentences plays a crucial role because anaphoric relations between elements, or in a broader sense text cohesion, must be preserved.

Consider the example from [Siddharthan (2003)]:

"Mr. Anthony, who runs an employment agency, decries program trading, but he isn't sure it should be strictly regulated."

which is simplified to:

"Mr Anthony decries program trading. Mr Anthony runs an employment agency. But he isn't sure it should be strictly regulated."

Here the simplified version does not preserve the original meaning because "*it*" in the simplified sentence seems to refer to "*employment agency*" instead of "*program trading*". However, a re-ordering of the sentences changes this.

The idea of Text Simplification differs from contextual sentence decomposition in the following ways. First of all we note that of course the motivation is different. Whereas the main motivation of Text Simplification lies in really simplifying the text for human readers our decomposition is of a rather "mechanical" nature, that is designed to be input to a full-text search engine. We neither require the resulting sub-sentences to be grammatically correct nor do we impose any other constraint on the content. However, as we will see, the process is designed in a way that most sub-sentences are in fact grammatically correct, something we gladly accept. We are also not concerned with the order of sub-sentences and instead we assume that all required anaphoric resolution has already taken place. Furthermore, as we will see, we tolerate slight losses of factual information, for example in causal or temporal dependencies.

To this respect contextual sentence decomposition is the easier task, and after all this is the intentional result of not imposing unnecessary constraints, thus enabling a faster and easier implementation. Nonetheless contextual sentence decomposition considers structures the Text Simplification systems we are aware of simply ignore. For example, the decomposition of enumerations, as in the sentence below

"Chaplin added Eric Campbell, Henry Bergman, and Albert Austin to his stock company."

which shall be decomposed into:

- "*Chaplin added Eric Campbell to his stock company*"
- "*Chaplin added Henry Bergman to his stock company*"
- "*Chaplin added Albert Austin to his stock company*"

Altogether Text Simplification is of different motivation and nature, but some of the basics tasks performed are similar. Thus a look at the approaches of Text Simplification might provide a direction. The main systems are based on a set of hand-crafted rules [Siddharthan (2003)], a complete dependency parse [Klebanov et al. (2004)] or a Lexical Tree Adjoining Grammar combined with rules [Chandrasekar et al. (1996)]. All of the approaches besides the set of hand-crafted rules are computationally expensive - a burden we are not willing to take. The fact that some rule-based systems, e.g. [Siddharthan (2003)], perform well according to their authors, motivates devising an own set of rules.

2.2 Semantic Role Labeling and Clause Identification

Semantic Role Labeling and *Clause Identification* are related to this thesis because they represent natural language processing tasks that are to some extent similar to what we are trying to achieve with sentence constituent identification. Although their goal is of a different nature we may benefit by including them or learn from the way they are implemented. We provide a short description and also explain why and to what extent they differ.

Semantic Role Labeling. The task of Semantic Role Labeling is *"to recognize arguments of verbs in a sentence, and label them with their semantic role"* [Carreras and Màrques (2004), p.1]. Each verb of a sentence has one or more arguments related to it, also providing the semantic role of the respective arguments. For example, in the sentence

"$[_{A0}$*Chaplin*$]$ ***added*** $[_{A1}$*Eric Campbell, Henry Bergman, and Albert Austin*$]$ $[_{A2}$*to his stock company*$]$*."*

the verb "*added*" has as arguments the subject (A0) "*Chaplin*", the object (A1) "*Eric Campbell, Henry Bergman, and Albert Austin"* and the indirect object (A2) *"to his stock company".* The possible argument roles of each verb are predefined.

Clearly the semantic identification of roles might provide a basis for a contextual decomposition. However, we note that our current definition of CSD does not require the level of detail provided by Semantic Role Labeling. Consider the in fact simple sentence

"$[_{A0}$*Chaplin*$]$ ***added*** $[_{A1}$*Eric Campbell*$]$ $[_{A2}$*to his stock company*$]$ $[_{AM-TMP}$*in 1915*$]$*."*

where additionally a temporal argument was found. Yet we do not require this information, in fact we are fine with leaving the sentence as is. This would not be a problem by itself, but the price to pay for such detail is processing speed.

Most accurate Semantic Role Labeling approaches are based on the output of a full syntactic parse [Carreras and Màrques (2004)], which is unacceptable in terms of processing speed for the amount of data we need to process. On the other hand known approaches based on many shallow pre-processing steps only deliver moderate performance with best F-measures below 70% [Carreras and Màrques (2004)], and no major recent improvements have taken place. Furthermore, for the first sentence above we realize that the elements of the enumeration "*Eric Campbell, Henry Bergman...*" are not further distinguished - as we will see something we want to provide. Altogether, this makes Semantic Role Labeling an interesting related work and a possible part of future research, but it currently can not provide a basis for contextual sentence decomposition, nor can we benefit from actual implementations because of the required expensive pre-processing steps.

Clause Identification. We refer to Clause Identification as described in the shared task of the Conference on Natural Language Learning of 2001 as "*dividing text into clauses*" [Sang and Déjean (2001), p.1]. Roughly, clauses are word sequences which contain a subject and a predicate that form a hierarchical structure. For example, in the following sentence the clauses are indicated by matching brackets:

"(Coach them in (handling complaints) (so that (they can resolve problems immediately)).)"

As we can see the clauses are type-less. The definition of clauses is detail is based on the typed clauses of a full parse tree of the english grammar. For our concerns it is enough to realize that the clauses do not reflect most of the sentence parts we need to identify for CSD. For example, enumerations as in

"(Ship companies carrying bulk commodities, such as oil, grain, coal, and iron ore, have been able (to increase their rates in the last couple of years).)"

are completely ignored, as are appositions and certain forms of relative clauses starting with participles, all of which we also require. What is more interesting to us is the type of problem solved. It is a hierarchical identification of constituents - something we require as well, but for typed constituents as we will see. Furthermore, the machine learning based approaches presented in [Sang and Déjean (2001)] show good performance with F-measures of around 90% for the best systems. We can conclude that Clause Identification can also not build the basis for contextual sentence decomposition, but that the problem solved is similar to the task at hand, which also motivates a machine learning based approach.

3 Problem Definition

Having already outlined the basic motivation of this thesis, this chapter gives a more detailed definition and description of the problem at hand and how we plan to tackle it. We formally introduce contextual sentence decomposition and its sub-tasks but start with an example.

We want to decompose sentences based on context of contained words. Consider again our query for *plants with edible leaves*. One of the results might be the following sentence:

"Usable parts of rhubarb include the edible stalks and the medicinally used roots, however its leaves are toxic."

This obviously wrong result is caused by the fact that *"rhubarb"*[4], "*edible*" and "*leaves*" appear in the same sentence. What we therefore want to achieve is a decomposition of this sentence into groups of words that "belong together":

- *"Usable parts of rhubarb include the edible stalks"*
- *"Usable parts of rhubarb include the medicinally used roots"*
- *"however its leaves are toxic"*

The above shall be the result of *contextual sentence decomposition*: a set of *sub-sentences* that result from a decomposition based on the context of words within the sentence. Words of the same context in the original sentence result in the same sub-sentence. The words "*rhubarb*", "*edible*" and "*leaves*" now no longer appear in the same sub-sentence and consequently searching each sub-sentence independently in the same fashion as before will eliminate the wrong result. This allows semantic full-text search adhering to the factual content of the original sentence.

We can achieve the decomposition by using operations only on the syntactic level. For this we first identify certain parts of a sentence and then recombine the parts into sub-sentences using simple operations. For example, in our sentence from above we can identify that "*the edible stalks*" and "*the medicinally used roots*" are part of an enumeration and that "*however its leaves are toxic*" is another self-sufficient sentence. By combining each of the enumeration parts with its beginning to create a

[4]We can assume the knowledge about rhubarb being a plant is incorporated into the search engine - an assumption that holds for the actual search engine we use with contextual sentence decomposition.

sub-sentence and creating a sub-sentence from the self-sufficient, contained sentence we achieve the desired decomposition.

We call the first phase of this process *sentence constituent identification (SCI)* and use it to identify typed constituents on the syntactic level. Using a set of operations we then recombine the identified constituents into sub-sentences in the *sentence constituent recombination (SCR)* phase. These are the two building blocks that make up contextual sentence decomposition.

The rest of this chapter is organized as follows. The next section gives a detailed definition of contextual sentence decomposition. This includes the definition of sentence constituent identification as well as sentence constituent recombination. We then present the set of sentence constituent types in detail in section 3.2. The final section provides some preliminaries of natural language processing required for understanding the remaining parts of this thesis.

3.1 Contextual Sentence Decomposition

We provide a definition of the overall process of contextual sentence decomposition by formally defining its sub-processes sentence constituent identification and sentence constituent recombination.

Contextual sentence decomposition is the result of first applying sentence constituent identification, followed by sentence constituent recombination. SCI differentiates between three basic constituent types, and we describe their notions using examples.

A *relative clause constituent* R_i has an attachment a_i it closer describes. For example, in the following sentence

"*Chaplin, who was born in London on the 16th of April 1889*, grew up in Lambeth."

the attachment a_1 "*Chaplin*" is described by the relative clause constituent R_1 "*who was born in London on the 16th of April 1889*". The i-th enumeration in a sentence is described by *list item constituents* L_{ij} and the words following and preceding it: the *list contexts* c_{i-1} and c_i. For example, in the following sentence

"Chaplin added Eric Campbel, Henry Bergman and Albert Austin to his stock company."

the list context c_0, "*Chaplin added*" is followed by the enumeration consisting of list items L_{11}, *"Eric Campbel"*, L_{12}, *"Henry Bergman"* and L_{13}, *"Albert Austin"* and a final list context c_2, "*to his stock company*". Of course, a sentence can have more than one enumeration.

The last constituent type is a *separator constituent,* which splits a sentence into self-sufficient parts. For example, the sentence

"*While in Prague he met Albert Einstein for the first time and afterwards they remained close friends.*"

can be split into self-sufficient parts "*While in Prague he met Albert Einstein for the first time*" and *"afterwards they remained close friends"*. We can describe this using an enumeration consisting of the list items L_{11}, "*While in Prague...*", and L_{12},*"afterwards they..."* and empty list contexts.

Definition 1. Given a sentence S, *one level of sentence constituent identification (1-level SCI)* yields r relative clause constituents $R_1, ..., R_r$ and their attachments $a_1, ..., a_r$, where $r \geq 0$, as well as l enumerations consisting of the list items $L_{11}...L_{1m_1}, ..., L_{l1}...L_{lm_l}$ and their list contexts $c_0, ..., c_l$, where $l \geq 0$. A *complete sentence constituent identification (SCI)* of S recursively applies 1-level SCI to each R_i and each L_{jk}.

Remark. Note that all R_i are substrings of S whereas $c_0 L_{11}...L_{1m_1}...c_{m-1} L_{lm_l} c_m$ is a subsequence. The two are disjoint, i.e. the exact words at their position that make up R_i are not part of the subsequence, however, the corresponding attachments, a_i, are.

Example 1. Consider the sentence *"Chaplin, who added Eric Campbell , Henry Bergman, and Albert Austin to his stock company, was born in London on the 16th of April 1889."*. SCI identifies the relative clause constituent R_1, "*who added Eric Campbell, Henry Bergman, and Albert Austin to his stock company*", and its attachment a_1, "*Chaplin*". On the contained level of R_1 it identifies one enumeration consisting of the list item constituents L_{11}, "*Eric Campbell*", L_{12}, "*Henry Bergman*", L_{13}, "*Albert Austin*" with the list contexts c_0, "*who added*" and c_1, "*to his stock company*".

Example 2. Consider the (fictitious) sentence "*The contemporary scientists Stephen Hawking, Richard Dawkins and Anthony James Legget were invited to the Nobel Prize award ceremonies and the Pulitzer Prize award ceremonies*". SCI identifies two enumerations. The first one consists of the list item constituents L_{11}, "*Stephen Hawking*", L_{12}, "*Richarrd Dawkins*", L_{13}, "*Anthony James Legget*" and the list context c_0, "*The contemporary scientists*". The second one consists of the list items L_{21}, "*Nobel Prize award ceremonies*", L_{21}, "*Pulitzer Prize award ceremonies*" and the list context c_1, "*were invited to*".

We can now define sentence constituent recombination.

Definition 2. Given a sentence S, let $R_1, ..., R_r$, $a_1, ..., a_r$, $L_{11}...L_{1m_1}, ..., L_{l1}...L_{lm_l}$, $c_0, ..., c_l$ be r relative clause constituents, their attachments, l enumerations, their list contexts, respectively, of one level of SCI. Then *sentence constituent recombination (SCR)* computes the sub-sentences as $SCR(S) = SCR_R(S) \cup SCR_L(S)$, where $SCR_R(S)$ and $SCR_L(S)$ are defined as follows. For $r = 0$, $SCR_R(S) = \emptyset$ and for $l = 0$, $SCR_L(S) = \{c_0\}$. For $r > 0$ or $l > 0$:

$$\begin{aligned} SCR_R(S) = \; & \{ \quad a_i s_i : s_i \in SCR_L(R_i)\} \cup \bigcup_{1 \leq i \leq r} SCR_R(R_i) \cup \bigcup_{j=1}^{l} \bigcup_{k=1}^{m_j} SCR_R(L_{jk}) \\ SCR_L(S) = \; & \{ \quad c_o s_1 c_1 ... c_{m-1} s_m c_m : \forall i, 1 \leq i \leq l \, \exists j, 1 \leq k_j \leq m_i, \, s_i \in SCR_L(L_{ik_j})\} \end{aligned}$$

We illustrate the definition continuing with the examples from above.

Example 3. For the sentence of example 1 above, after performing SCI as described, the sub-sentences are computed as follows. $SCR_R(R_1)$ attaches the results of $SCR_L(R_1)$ to a_1, "*Chaplin*". On the contained level, R_1 consists of a list enumeration with list items L_{11},L_{12},L_{13} and list contexts c_0 and c_1. Because these list items are not further nested, the recursive applications of SCR_L on the single list items result in exactly the sequences of words they are made up of. These are then combined to the sequences $c_0 L_{11} c_1$, $c_0 L_{12} c_1$, $c_0 L_{13} c_1$ and each of it is attached to a_1 resulting in the following sub-sentences:

- "*Chaplin who added Eric Campbell to his stock company*"
- "*Chaplin who added Henry Bergman to his stock company*"
- "*Chaplin who added Albert Austin to his stock company*"

Furthermore the extraction of list items on the highest level $SCR_L(S)$ returns c_0, which as previously noted, does contain the relative clause constituents and therefore corresponds to the last sub-sentence:

- *"Chaplin was born in London on the 16th of April 1889"*

Example 4. For the sentence of example 2 above, after performing SCI as described, the sub-sentences are computed as follows. The sentence does not contain any relative clause constituents, therefore $SCR_R(S)$ results in the empty set. The recursive applications of SCR_L on the list items $L_{11}, L_{12,} L_{13}, L_{21}$ andL_{22} result in exactly the sequence of words they are made up of. The enumeration $L_{11}, L_{12,} L_{13}$ with its list context c_0 and the enumeration $L_{21,} L_{22}$ with list context c_1 are therefore combined to $c_0 L_{11} c_1 L_{21}$, $c_0 L_{12} c_1 L_{21}$, $c_0 L_{13} c_1 L_{21}$, $c_0 L_{11} c_1 L_{22}$, $c_0 L_{12} c_1 L_{22}$, $c_0 L_{13} c_1 L_{22}$, which are the desired the sub-sentences:

- "*The contemporary scientists Stephen Hawking were invited to the Nobel Prize award ceremonies*"

- "*The contemporary scientists Richard Dawkins were invited to the Nobel Prize award ceremonies*"
- "*The contemporary scientists Anthony James Legget were invited to the Nobel Prize award ceremonies*"
- "*The contemporary scientists Stephen Hawking were invited to the Pulitzer Prize award ceremonies*"
- "*The contemporary scientists Richard Dawkinswere invited to the Pulitzer Prize award ceremonies*"
- "*The contemporary scientists Anthony James Legget were invited to the Pulitzer Prize award ceremonies*"

This concludes the definition of SCI and SCR with the introduction of a more intuitive notation. To denote a sentence with identified constituents to the reader a notation using brackets is more convenient. For example, the sentence with identified constituents from example 1 above, can be described as follows:

"Chaplin, $(_{REL}$*who added* $(_{LIT}$*Eric Campbell*$)_{LIT}$ *,* $(_{LIT}$*Henry Bergman*$)_{LIT}$*, and* $(_{LIT}$*Albert Austin*$)_{LIT}$ *to his stock company*$)_{REL}$*, was born in London on the 16th of April 1889."*

Matching brackets identify the constituent and the subscript gives the constituent type. This is the notation we follow throughout this thesis.

We conclude the problem definition with some remarks about contextual sentence decomposition.

Due to the detail of our definition no further description of an SCR implementation is required. The central problem of CSD is therefore the task of SCI: identifying constituents and a corresponding structure that properly reflects the constituents and their context in the sentence. Because it was irrelevant to the definition, we have only considered one type of relative clause constituent so far. There is another type of relative clause constituent: the *apposition constituent*. All constituent types are described in detail in the next section. Furthermore, the definition of SCI includes the identification of attachments for relative clauses. We note that our approaches do not explicitly identify those, but use a simple heuristic, attaching relative clauses to the closest noun that precedes them.

The resulting sub-sentences, as formally defined above, are sequences, however, for the intended use in semantic full-text search we only require the sub-sentences to be multi-sets. We do not require them to be grammatically correct (although this is actually the case for most of them) and the order of words within a sub-sentence is irrelevant as well. However, frequencies of words may be relevant for search internal ranking.

The process of CSD is designed to avoid adding factual information that was not present in the original sentence. More importantly however, we also have to avoid losing factual information due to the decomposition. Consider the following sentence:

"While in Prague Paul Ehrenfest met Albert Einstein for the first time and afterwards they remained close friends."

There are two obvious sub-sentences that can be extracted:

- *"While in Prague Paul Ehrenfest met Albert Einstein for the first time"*
- *"afterwards they remained close friends."*

To avoid losing information we assume that all references, also called "anaphora", within the sentence were resolved beforehand. In the example above this means that it is known that "*they*" in the second sub-sentence refers to "*Paul Ehrenfest*" and "*Albert Einstein*". Therefore, each sub-sentence can be considered independently and is self-sufficient, even after decomposition. However, in the above sentence another piece of information got lost during decomposition. The temporal relation that Paul Ehrenfest and Albert Einstein remained friends *after* they met in Prague is no longer present when considering each sub-sentence independently. However, we argue that this is impractical to avoid with reasonable effort and also unnecessary for our intended use. Because the sub-sentences are intended for use in semantic full-text search one must take one step back and realize that formulating a query for such a fact is a hard task in the first place. This would be asking for queries like "What happened after Paul Ehrenfest met Albert Einstein in Prague for the first time?" or "After what did they remain close friends?". Given these are questions that, from our experience, hardly occur and are hard to formulate, we can safely accept the loss of these temporal or causal relations. Besides those temporal or causal relations all the relevant information should, of course, be preserved during decomposition.

A final subtle detail to note is that the process of contextual sentence decomposition deliberately does not define the exact extent of decomposition. It could for example be argued whether the sub-sentence *"Chaplin was born in London on the 16th of April 1889"* should be split further into "*Chaplin was born in London*" and "*Chaplin was born on the 16th of April 1889*". It is indeed an open question to which extent this is reasonable for semantic full-text search, but the definition plays to our advantage. The degree of freedom allows us to experiment with different strategies and by adapting the SCI and SCR phrases we are free to do so in a later implementation.

What remains open is a detailed description of the different constituent types. This is provided in the next section.

3.2 Sentence Constituent Types

In the following we describe four different constituent types used in contextual sentence decomposition and give their grammatical or semantic relations. For illustration purposes an example is provide with each type.

Relative Clause Constituent. As the name implies a *relative clause constituent* resembles a relative clause and we refer to it as a constituent of type *REL*.

"*Chaplin,* $(_{REL}$*who was born in London on the 16th of April 1889*$)_{REL}$, grew up in Lambeth."

It is recombined by attaching it to the noun it describes, resulting in:

- "*Chaplin who was born in London on the 16th of April 1889"*
- "*Chaplin grew up in Lambeth*"

The relative clause constituent can be a relative clause in any form: restrictive or non-restrictive, reduced or not reduced. All relative clauses provide information about the noun or noun-phrase they refer to. In the case of restrictive relative clauses they more closely identify what they refer to as in "*The house that collapsed was sold*". The restrictive relative clause "*that collapsed*" identifies the house that was sold. Non-restrictive relative clauses only provide additional information and are enclosed by commas whereas restrictive relative clauses are not. Both relative clause types can be in reduced form by removing the relative pronoun as in *"The house providing shelter collapsed*", with "*that provides shelter*" being the original restrictive relative clause, or "*The houses, shining with new paint, were demolished*", with "*which were shining with new paint*" being the original non-restrictive relative clause. The current SCI implementations only consider non-restrictive relative clauses.

Apposition Constituent. An *apposition constituent* resembles some apposition in the sentence and we refer to it as a constituent of type *RELA*.

"His father was Hermann Einstein , $(_{RELA}$ *a salesman and engineer*$)_{RELA}$*."*

It is recombined by attaching it to the noun it describes, resulting:

- "*His father was Hermann Einstein*"
- "*Hermann Einstein a salesman and engineer*"

Here a "*a salesman and engineer*" is the apposition and also said to be *in apposition* with *"Hermann Einstein*". An apposition is in its nature similar to a relative clause, hence the similar name RELA. It offers additional information or identifies the usually preceding noun or noun-phrase. In contrast to a relative clause there is usually a "is-a" relation between an apposition and whatever it modifies. Therefore, one can replace the apposition and noun or noun-phrase it modifies and the sentence remains grammatically correct. Like relative clauses, appositions occur restrictive, as in "*My friend Tom bought me an ice*", with "*Tom*" being the apposition, or non-restrictive as in "*The burden fell on Tony Blair, the prime minister*", with "*the prime minister*" being the apposition.

List Item Constituent. In contrast to the constituents seen so far the *list item constituent* has no directly assignable grammatical element. We use it to identify the parts of a list or enumeration and refer to it as a constituent of type *LIT* (for list item). Consider our example from the beginning:

"Chaplin added $(_{LIT}$ *Eric Campbell* $)_{LIT}$*,* $(_{LIT}$ *Henry Bergman* $)_{LIT}$*, and* $(_{LIT}$ *Albert Austin* $)_{LIT}$ *to his stock company."*

It is recombined by a "multiplication" as previously defined, resulting in:

- "*Chaplin added Eric Campbell to his stock company.*"
- "*Chaplin added Henry Bergman to his stock company.*"
- "*Chaplin added Albert Austin to his stock company.*"

Obviously this is an enumeration of people Charlie Chaplin added to his stock company, and each person has been identified in form of a list item constituent. Note that an enumeration is made up of list items and that a sentence can contain several enumerations.

Separator Constituent. The *separator constituent* identifies a word that splits a sentence into two self-sufficient sentences. We refer to it as a constituent of type *SEP*. It represents a special case because it spans only one word and can not contain other constituents.

"*While in Prague he met Albert Einstein for the first time* $(_{SEP}$ *and* $)_{SEP}$ *afterwards they remained close friends.*"

It is recombined by independently considering the self-sufficient sentences:

- "*While in Prague he met Albert Einstein for the first time*"
- "*afterwards they remained close friends.*"

This example creates two self-sufficient sentences that can be analyzed independently. No other constituent is allowed to span over a SEP-constituent and thus it is a good strategy to first identify the SEP-constituents and afterwards continue with identification of the other constituents in the resulting sentences.

3.3 Natural Language Processing Preliminaries

This section provides some preliminaries of natural language processing, namely *part-of-speech* tagging as well as *text chunking*, which are required and applied throughout the rest of this thesis. Both provide only a partial analysis of a sentence, and especially text chunking is often described as an instance of *shallow* or *partial parsing.* In contrast to *full parsing,* shallow parsing does not provide the complete syntactic structure of a sentence, but works on a lower, coarser level. Because it does not have to disambiguate between all syntactic elements of a sentence it is faster, but also less expressive.

Part-of-speech Tagging. Each word in the english language can be assigned a category such as noun, verb, preposition or adjective. This category is also referred to as a word's *part-of-speech* (POS). Each word category can be abbreviated using a so called *tag* and *POS-tagging* is the task of assigning each word one of those tags. For example the sentence "*While in Prague he met Albert Einstein for the first time.*" can be tagged in the following way:

"While	*in*	*Prague*	*he*	*met*	*Albert*	*Einstein*	*for*	*the*	*first*	*time"*
IN	IN	NP	PP	VBD	NP	NP	IN	DT	JJ	NN

where each word's part-of-speech tag is given below the word. The tag NP stands for a proper noun, IN for a preposition, VBD for a verb in past tense, NN for a singular noun, JJ for an adjective and DT for a determiner. This is of course only a small subset of the possible tags. A description of common tags can be found in [Marcus et al. (1993)].

Note that a word's part-of-speech depends on the context it is used in. For example, the word "run" which is typically used as a verb can also take the role of a noun, for example in "*The last evaluation run was not successful.*". The set of all possible tags is often called the *tagset.* Different tagsets exist but for the english language the tagset of the Penn Treebank Project[5], consisting of 48 different tags, is often used.

The task of POS-tagging is performed by a POS-tagger and a large number of different approaches and POS-taggers exist. A well-known rule based POS-tagger is the Brill tagger [Brill (1992)]. For our implementation tasks we used TreeTagger [Schmid (1994)], a probabilistic tagger based on decision trees, which was trained on the Penn tagset. The accuracy of the respective POS-taggers reported by [Brill (1992)] and [Schmid (1994)] is well above 95% so one could consider POS-tagging a solved problem. In any case we can safely assume the results to be correct in the context of our task.

[5]http://www.cis.upenn.edu/~treebank/

Text Chunking. *Text chunking* is the task of "*dividing a text into phrases in such a way that syntactically related words become member of the same phrase*" [Tjong Kim Sang and Buchholz (2000)]. For example, our sentence from above can be divided into:

$$({}_{SBAR}\textit{While})_{SBAR}\ ({}_{PP}\textit{in})_{PP}\ ({}_{NP}\textit{Prague})_{NP}\ ({}_{NP}\textit{he})_{NP}\ ({}_{VP}\textit{met})_{VP}\ ({}_{NP}\textit{Albert Einstein})_{NP}\ ({}_{PP}\textit{for})_{PP}\ ({}_{NP}\textit{the first time }})_{NP}.$$

where NP denotes a noun phrase, VP a verb phrase, PP a prepositional phrase and SBAR the begin of a subordinated clause. The resulting chunks may not overlap and each word is assigned a chunk type, i.e. the assignment is not *sparse*. For a list of chunk types we refer to [Tjong Kim Sang and Buchholz (2000)] but we note that according to them usually more than 90% of all chunk types observed are noun, verb or prepositional phrases.

The terms text chunking and syntactic chunking are often used equivalently. Text chunking is an intermediary step to a full-parse, therefore an instance of shallow parsing, that provides us with a coarser grained sequential sentence structure. A text chunk can be seen as a unit and as such the constituents defined by sentence constituent identification never split chunks but only properly embed them. This is a great aid for any implementation of sentence constituent identification.

Text chunking was the shared task of the Conference on Computational Natural Language Learning[6] in 2000. [Tjong Kim Sang and Buchholz (2000)] provides a detailed description of the task as well as results. A large training set was provided and 11 systems competed against each other. Results showing F-scores far above 90% allow us to rely on the accuracy for our approaches. The input of a text chunker has to be POS-tagged, suggesting a pipelined approach where as a first step POS-tagging is followed by text chunking. In our implementation tasks we used YamCha [Kudoh and Matsumoto (2000)], an approach utilizing Support Vector Machines, due to its fast performance and high accuracy.

[6]http://www.cnts.ua.ac.be/conll2000/

4 Sentence Constituent Identification Using Rules

In the previous chapter we outlined the basic problem setting of this thesis and we also introduced the process of sentence constituent identification (SCI). With SCI being the challenging part of contextual sentence decomposition (CSD) this chapter now describes the first of two approaches, which is based on a set of rules.

The chapter is structured as follows: The next introductory section provides some preliminaries and conventions. In section 4.2 we introduce our rules to identify the start of constituents followed by a section describing rules to determine their end. Results and a comparison to the Machine Learning based approach are provided in chapter 7.

4.1 Preliminaries

Consider the following sentence:

"*Kofi Annan, who is the current U.N. Secretary General , has spent much of his tenure working to promote peace in the Third World.*"

What we want to achieve is an identification of the sentence constituents, as defined in the previous chapter. In this case the result must simply be the following:

"*Kofi Annan,* $(_{REL}$ *who is the current U.N. Secretary General* $)_{REL}$ *, has spent much of his tenure working to promote peace in the Third World.*"

A first trivial observation is that the relative clause starts with the word "*who*" preceded by a comma and extends to the next comma. This simple rule can then be used as the basis for a set of more elaborate rules to recognize relative clause constituents. Of course, a complete set of rules also has to account for possibly embedded constituents. By carefully observing syntactic and semantic relations within a sentence we developed such a set of rules and present them in this chapter in form of algorithms written in pseudocode. The main advantages of a rule based approach are the possibility of fast rule evaluation and the comprehensibility of the rules and, therefore, also the final results.

To allow the definition and implementation of efficient rules and to keep the ruleset small, we require that the input has *part-of-speech* as well as *text chunking* tags assigned. These were already described in the preliminaries section 3.3. The benefit of this is best explained with our example above, where we already saw an instance of a simple rule: a relative clause constituent starts with a comma, followed by the word "*who*", and extends to the next comma. Of course, one can create the same rule for a relative clause starting with "*which*", "*what*" and so on. This results in an unmanageable amount of rules and it is thus preferable to move to a level that abstracts from the lexical items. Therefore, we make use of part-of-speech (POS) tags, which assign each word a category. We can now formulate more generic rules of the following form: a relative clause constituent starts with a comma, followed by a word with POS-tag WDT[7], and extends to the next comma. Text chunking already provides us with a good notion of "belonging together". Consider the text chunking result of our example sentence:

"($_{NP}$ *Kofi Annan* $_{NP}$) , ($_{NP}$ *who* $_{NP}$)($_{VP}$ *is* $_{VP}$)($_{NP}$ *the current U.N. Secretary General* $_{NP}$) , ($_{VP}$ *has spent* $_{VP}$) ($_{ADVP}$ *much* $_{ADVP}$) ($_{PP}$ *of* $_{PP}$) ($_{NP}$ *his tenure* $_{NP}$)($_{VP}$ *working to promote* $_{VP}$)($_{NP}$ *peace* $_{NP}$)($_{PP}$ *in* $_{PP}$)($_{VP}$ *the Third World* $_{NP}$)."

where *"NP"* denotes a noun phrase, "*VP*" a verb phrase, "*PP"* a prepositional phrase and "*ADVP*" an adverbial phrase. An obvious thing to note is that SCI never identifies constituents that split a phrase, or the other way round, phrases are always properly embedded in constituents. This reduces the amount of positions we can start or end a constituent, something an implementation can utilize.

To keep the pseudocode representing the rules readable, we use a set of conventions and helper functions. First of all we apply some marking to the list of phrases. The markers RELSTART, LITSTART or RELASTART mean the apparent start of a relative, list item or appositive constituent. We implicitly keep two lists of phrases: one of all phrases and one containing only the marked phrases. Figure 4.1 shows the functions and conventions used within the pseudocode.

[7]The POS-tag WDT stands for "which determiners": words like, which, who or whose etc.

phrases	Is the sequential list of phrases of the sentence we are currently processing.
phrase matches postag Y	Is true, if the regular expression pattern in Y matches the POS-tags of the sentence starting at *phrase*.
nextMarkedPhrase(phrase)	Returns the phrase following *phrase* in the list of marked phrases.
previousMarkedPhrase(phrase)	Returns the phrase preceding *phrase* in the list of marked phrases.
nextPhrase(phrase)	Returns the phrase following *phrase* in the list of phrases.
enumerationEnd(*phrase*)	Returns true, if the phrase consists of a word with POS-tag CC or if the phrase is of the type CONJP. These typically introduce ends of enumerations ("*The meal includes pepper, salt* ***as well as*** *peanuts.*" etc.).

Figure 4.1: Functions and conventions for pseudocode.

4.2 Finding Constituent Starts

It is effective to begin with the identification of constituent starts because they appear in an environment that is easier distinguishable and more characteristic than that of their ends. The following rules and algorithms build on each other and are therefore designed to be executed in sequence.

Algorithm 4.1 Identifying Non-restrictive Relative Clause Starts

```
1: for phrase in phrases do
2:   if previousPhrase(phrase) = "," then
3:     if phrase matches postag "(IN WDT|WDT|VBN|VBG).*" then
4:       mark phrase as RELSTART
5:       mark previousPhrase(phrase) as IGNORE
6:     end if
7:   end if
8: end for
```

Identifying non-restrictive relative clause starts. As we have seen in the previous section, identifying the beginning of non-restrictive relative clauses is not that complicated. Therefore, we start by applying algorithm 4.1 to the sentence.

Non-restrictive relative clauses are enclosed in commas (line 2) and start with words like "*who*", "*which*" or "*what*", but can also start with verbs, e.g., in a participle form (VBG) as in *"Globe Managing Manager Thomas Culvoy, bending to the will of his troops, scrapped the new drawings.*" (line 3). Furthermore they may start with a preposition followed by a which-determiner as in "*The CFTC plans to curb dual trading on commodities markets, in which traders buy and sell both for their own account and for clients.*". We mark a matching phrase with RELSTART. Because the comma in front of a relative clause is no longer relevant in further considerations we mark it as IGNORE.

Algorithm 4.2 Coloring Decision Points

```
1:  for phrase in list of unmarked phrases do
2:    if phrase = "," then
3:      if nextPhrase(phrase) matches postag "CC" or nextPhrase(phrase) is type
        CONJP then
4:        mark nextPhrase(phrase) as RED
5:      else
6:        mark phrase as RED
7:      end if
8:    else
9:      if phrase matches postag "CC" or phrase is type CONJP then
10:       mark phrase as RED
11:     end if
12:   else
13:     if phrase is type VP then
14:       mark phrase as VIO
15:     end if
16:   end if
17: end for
```

Coloring decision points. Having identified commas belonging to relative clauses has reduced the amount of remaining points we need to make decisions for. Algorithm 4.2 colors the remaining decision points in red (RED), and, to aid further decisions, verb-phrases in violet (VIO).

The remaining decision points include all remaining commas as well as phrases consisting of coordinating conjunctions (CC), which resemble words like "*and*", "*or*" and "*but*". These are typical points that separate a sentence or appear at the end of an enumeration. Conjunctive phrases (CONJP), e.g. "*as well as*", have an identical function and are also marked RED (line 10). Lines 3 and 5 are responsible for the cases where the current phrase is a comma. If the following phrase is a coordinating conjunction or similar we mark it and ignore the comma (line 4), otherwise we simply mark the comma (line 6). This way we avoid marking two consecutive phrases as RED . Furthermore, as a helper, we mark verb-phrases in violet (VIO) (line 14).

Algorithm 4.3 Evaluating Decision Points

```
 1: for phrase in list of marked phrases do
 2:     if phrase is marked RED then
 3:         # If next color is VIO, but there is some phrase in-between mark as SEP
 4:         if nextMarkedPhrase(phrase) is marked VIO
            and nextPhrase(phrase) != nextMarkedPhrase(phrase) then
 5:             mark phrase as SEP
 6:         end if
 7:     else
 8:         mark phrase as LITSTART
 9:     end if
10: end for
```

Evaluating decision points. We now iterate through the list of all phrases marked RED and make a decision for them as described in algorithm 4.3.

All phrases marked RED are changed to a separator constituent, if the following marker is VIO (verb-phrase), but only if the verb-phrase is not the directly following phrase (line 5). To illustrate this consider the example sentence "*While in Prague he met Albert Einstein for the first time and afterwards they remained close friends.*". The "*and*" would be marked as a separator constituent, because the next marked phrase "*remained*" is VIO, but it does not directly follow. This is based on the observation that a separator needs to split into self-sufficient sentences, which always contain a verb phrase. On the other hand they also need a subject, which usually precedes the verb-phrase, and, as a consequence, the verb-phrase can not directly follow the separator. If we cannot ensure we are dealing with a relative clause or separator constituent we mark it as the beginning of a list item (line 8). Note that we have already marked the relative clause constituents in a previous step. In the end this will result in the sentence "*Chaplin added Eric Campbell**,** Henry Bergman, **and** Albert Austin to his stock company.*" where the first comma and the "*and*" were marked RED, to be changed to *"Chaplin added Eric Campbell* $(_{LIT}$, *Henry Bergman ,* $(_{LIT}$*and Albert Austin to his stock company.*".

Correction Run. We have now marked relative clause, separator and list item constituents. However, in the previous step we deliberately marked everything as list item start that we could safely identify as relative clause or separator constituent start. Thus we have one more set of rules presented in algorithm 4.4 that corrects list item start marks to either an appositive constituent start or a helper marker CLOSE. Note that switching between list item and appositive constituents is reasonable even on a grammatical level, because their structure is indeed very similar. Both usually only consist of noun-phrases and are distinguishable only in the immediate context they appear in the sentence. Compare the sentence *"One of the contemporary scientists, Stephen Hawking, made a remarkable contribution."* with

Algorithm 4.4 Correcting List Items

```
1: $listOpen \leftarrow false$
2: for phrase in reversed list of marked phrases do
3:   # Not a correct ending of an enumeration
4:   if not listOpen and phrase is-marked LITSTART
       and not enumerationEnd(phrase) then
5:     # This is a typical continuation of an intermitted clause
6:     if nextPhrase(phrase) is type VP
         and not nextPhrase(phrase) matches postag "(VBG|VBN)" then
7:       mark phrase as CLOSE
8:       # If it is no continuation it is an appositive
9:     else
10:      mark phrase as RELASTART
11:    end if
12:  else
13:    # Correct ending of an enumeration
14:    if not listOpen and phrase is marked LITSTART
         and enumerationEnd(phrase) then
15:      $listOpen \leftarrow true$
16:    end if
17:  else
18:    # Now the enumeration is finished
19:    if listOpen and not phrase is marked LITSTART then
20:      $listOpen \leftarrow false$
21:    end if
22:  end if
23:  # Default: consume the list item
24: end for
```

"*Some contemporary scientists include Stephen Hawking , Richard Dawkins and Nobel prize-winner Anthony James Leggett.*". In the first *"Stephen Hawking"* appears as the appositive whereas in the second it is a list item.

In detail algorithm 4.4 works as follows. We move backwards through all marked phrases. If a phrase is marked as a list item start and no enumeration has been started so far we check if this last list item is introduced by an acceptable construction. This is the case if it starts with a coordinating conjunction (CC), like "*and*" or "*or*", as in "*Tom likes chicken, beef* ***and*** *pork.*" or it starts with a conjunctive phrase (CONJP), as in *"Tom likes chicken, beef* ***as well as*** *pork.*". In this case we remember that we opened a list (line 15). If the list is not correctly ended we check if a verb-phrase in a certain form directly follows, which might indicate the continuation after an intermitted constituent. If this is the case we use our helper marker CLOSE and move on (line 7). If it is no continuation we revert the phrase to an apposition constituent start (line 10). For example, in "*He was born in Birmingham* *in 1921 and is the father of Sir Tim Berners-Lee , the inventor of the World Wide Web.*" the comma will be marked as as list item start first, because no verb-phrase followed for a separator, but will then be reverted to an apposition constituent start. Note that exchanging the comma against the word "*and*" would result in a correct enumeration and we would not revert to an apposition. If we are inside a correct list enumeration and another constituent marker appears we end the list (line 20). If we are inside a list and see a list item we consume it (line 23).

Algorithm 4.5 Finding First List Item of Enumerations

```
1: for phrase in list of marked phrases do
2:   if phrase is marked LITSTART
       and not previousMarkedPhrase(phrase) is marked LITSTART then
3:     if phrase is type VP then
4:       mark first verb-phrase after previousMarkedPhrase(phrase) as LIT-
         START
5:     end if
6:   else
7:     mark last noun-phrase after previousMarkedPhrase(phrase) as LITSTART
8:   end if
9: end for
```

Finding missing List Items. The previous identification of list item starts missed an important thing about enumerations: we have yet to discover the start of the first list item. For the sentence "*Chaplin added Eric Campbell* ($_{LIT}$, *Henry Bergman ,* ($_{LIT}$*and Albert Austin to his stock company.*" we have only marked two of the three list item constituent beginnings. "*Eric Campbell*" still has to be marked. Although, fixing this is based on a simple rule, we provide it in algorithm 4.5 for completeness.

We search for the first item of an enumeration in front of the currently first list item (line 2). We then have two possibilities. Either we are in an enumeration of verbs (line 3) and thus mark the first verb-phrase after the previous marked phrase, or we are in a common enumeration (line 5) and thus mark the last noun-phrase after the previous marked phrase. For example, in "*He was born in Birmingham in 1921 and is the father of Sir Tim Berners-Lee*" this will result in "*He* ($_{LIT}$ *was born in Birmingham in 1921 and* ($_{LIT}$ *is the father of Sir Tim Berners-Lee*", which is what we call an enumeration of verbs. For our example "*Chaplin added Eric Campbell* ($_{LIT}$, *Henry Bergman,* ($_{LIT}$*and Albert Austin to his stock company.*" the result will be "*Chaplin added* ($_{LIT}$ *Eric Campbell* ($_{LIT}$, *Henry Bergman,* ($_{LIT}$ *and Albert Austin to his stock company.*"

4.3 Finding Constituent Ends

Algorithm 4.6 uses some simple heuristics to find constituent ends we describe in the following.

The basic principle is to iterate through all phrases, observe and remember the constituent start markers and assign constituent end markers for them at the appropriate words. Whenever we observe a constituent start marker we push it onto a stack to keep track of the open constituents. Depending on the sentence structure and markers, we then close all or only particular constituent types at certain positions, which also allows embedding of certain constituents.

Algorithm 4.6 Finding Constituent Ends

```
1: listOpen ← false
2: # Nothing
3: for phrase in phrases do
4:    if phrase is marked SEP or CLOSE then
5:       pop and close all open constituents at phrase
6:    end if
7:    if phrase is marked RELSTART then
8:       if a relative clause is already opened then
9:          if phrase matches postag "CC WD(T|P).* then
10:            pop and close open REL constituent at phrase
11:         end if
12:         pop and close open LIT constituent at phrase
13:         pop and close open RELA constituent at phrase
14:         push new REL constituent start at phrase
15:      end if
16:   end if
17:   if phrase is marked LITSTART then
18:      pop and close open LIT constituent at phrase
19:      pop and close open RELA constituent at phrase
20:      push new LIT constituent start at phrase
21:   end if
22:   if phrase is marked RELASTART then
23:      pop and close open LIT constituent at phrase
24:      pop and close open RELA constituent at phrase
25:      push new RELA constituent start at phrase
26:   end if
27: end for
28: close all open constituents
```

In detail algorithm 4.6 works as follows. The sentence end, a separator constituent or a CLOSE marker will end all open constituents, so we insert corresponding constituent end markers and remove their beginning markers from the stack (line 5 and 24). If we observe a new relative clause constituent start (line 7) and another relative clause constituent has already been opened (line 8), we check if the new one starts with a coordinating conjunction followed by a which determiner (line 9). To illustrate this situation consider the example sentence "*The inquiry also will cover the actions of Charles Keating Jr., who is chairman of American Continental Corp. , Lincoln 's parent, and who contributed heavily to several U.S. senators.*" where "*and who contributed* [...]" is the relative clause we are currently looking at that matches these POS-tags. This is an indication that it attaches to the same noun as the already opened relative clause. Therefore, we close the last open relative clause constituent (line 10) and treat it as any other new beginning of a relative clause (line 14). This will create two independent relative clauses that the SCR-phase then needs to attach to the same noun. An alternative approach would have been to merge the two relative clauses and treat them as one, as Siddharthan does in [Siddharthan (2003)]. This would however result in less powerful contextual sentence decomposition because we would identify only one relative clause constituent, which would then also result in only one sub-sentence. The remaining cases are simple. Whenever a new list item or appositive constituent starts we close all opened ones and start a new constituent (line 20, line 25). All constituents end, when a separator constituent or a CLOSE marker is observed. Overall, this results in the possibility of embedding list item and appositive constituents in relative clauses. However, list items and appositive constituents never embed other constituents. In practice, indeed, appositions almost never embed one of the other constituents. However, more complicated sentences can embed constituents within a list item. This is a current limitation of the rule based approach.

Having assigned a constituent end to each constituent start concludes the rule based approach of sentence constituent identification.

4.4 Examples

To illustrate the rules we apply them on two examples.

Throughout the examples, a word marked as the start or end of a constituent is identified with the corresponding bracket immediately before respectively after the word. This follows the notation already introduced in the problem definition chapter. All other marks are given as subscript of the marked word. Note that a comma is treated as any other word.

The first sentence we consider is:

"*Camphauser Straße expressway , which leads to the B 268 and the B 51, was destroyed.*"

Algorithm 4.1 is applied first and identifies the start of a relative clause:

“*Camphauser Straße expressway* $,_{IGNORE}$ $(_{REL}$*which leads to the B 268 and the B 51 , was destroyed.*”

Next, according to algorithm 4.2, decision points are marked RED and verb phrases VIO:

“*Camphauser Straße expressway* $,_{IGNORE}$ $(_{REL}$*which leads*$_{VIO}$ *to the B 268 and*$_{RED}$ *the B 51* $,_{RED}$ *was destroyed*$_{VIO}$*.*”

Now, the decision points are evaluated by applying algorithm 4.3:

“*Camphauser Straße expressway* $,_{IGNORE}$ $(_{REL}$*which leads*$_{VIO}$ *to the B 268* $(_{LIT}$*and the B 51* $(_{LIT}$, *was destroyed*$_{VIO}$*.*”

This caused some wrong decisions and algorithm 4.4 is supposed to repair them:

“*Camphauser Straße expressway* $,_{IGNORE}$ $(_{REL}$*which leads*$_{VIO}$ *to the B 268* $(_{LIT}$*and the B 51* $,_{CLOSE}$ *was destroyed*$_{VIO}$*.*”

This corrected the errors and algorithm 4.5 now adds the missing list item constituents at the start of the enumeration:

“*Camphauser Straße expressway* $,_{IGNORE}$ $(_{REL}$ *which leads*$_{VIO}$ *to the* $(_{LIT}$*B 268* $(_{LIT}$*and the B 51* $,_{CLOSE}$ *was destroyed*$_{VIO}$*.*”

With the help of the CLOSE marker the ends of each opened constituent are determined by algorithm 4.6:

“*Camphauser Straße expressway* $,_{IGNORE}$ $(_{REL}$*which leads*$_{VIO}$ *to the* $(_{LIT}$*B 268*$)_{LIT}$ $(_{LIT}$*and the B 51*$)_{LIT})_{REL}$ $,_{CLOSE}$ *was destroyed*$_{VIO}$*.*”

Ignoring the superflous marks gives the following final and correct result:

“*Camphauser Straße expressway* $,_{IGNORE}$ $(_{REL}$*which leads to the* $(_{LIT}$*B 268*$)_{LIT}$ $(_{LIT}$*and the B 51*$)_{LIT})_{REL}$, *was destroyed.*”

The next sentence we consider is:

"The original fortress, known as Gyel-khar-tse , was attributed to Pelkhor-tsen, son of the anti-Buddhist king Langdharma , who probably reigned from 838 to 841 CE."

Now, algorithm 4.1 identifies the start of two relative clauses:

"The original fortress $,_{IGNORE}$ $(_{REL}$*known as Gyel-khar-tse, was attributed to Pelkhor-tsen , son of the anti-Buddhist king Langdharma* $,_{IGNORE}$ $(_{REL}$*who probably reigned from 838 to 841 CE."*

Algorithm 4.2 marks decision points and verb phrases:

"The original fortress $,_{IGNORE}$ $(_{REL}$*known*$_{VIO}$ *as Gyel-khar-tse* $,_{RED}$ *was attributed*$_{VIO}$ *to Pelkhor-tsen* $,_{RED}$ *son of the anti-Buddhist king Langdharma* $,_{IGNORE}$ $(_{REL}$*who probably reigned*$_{VIO}$ *from 838 to 841 CE."*

As before, the decision points are evaluated by applying 4.3:

"The original fortress $,_{IGNORE}$ $(_{REL}$*known*$_{VIO}$ *as Gyel-khar-tse* $(_{LIT},$ *was attributed*$_{VIO}$ *to Pelkhor-tsen* $(_{LIT},$ *son of the anti-Buddhist king Langdharma* $,_{IGNORE}$ $(_{REL}$*who probably reigned*$_{VIO}$ *from 838 to 841 CE."*

Again, no seperators were found, but the points were reverted to list item starts. Algorithm 4.4 is supposed to repair this:

"The original fortress $,_{IGNORE}$ $(_{REL}$*known*$_{VIO}$ *as Gyel-khar-tse* $,_{CLOSE}$ *was attributed*$_{VIO}$ *to Pelkhor-tsen* $(_{RELA},$ *son of the anti-Buddhist king Langdharma* $,_{IGNORE}$ $(_{REL}$*who probably reigned*$_{VIO}$ *from 838 to 841 CE."*

The list item was reverted to an apposition constituent. No further list item starts exist and algorithm 4.5 therefore doesn't change anything:

"The original fortress $,_{IGNORE}$ $(_{REL}$*known*$_{VIO}$ *as Gyel-khar-tse* $,_{CLOSE}$ *was attributed*$_{VIO}$ *to Pelkhor-tsen* $(_{RELA},$ *son of the anti-Buddhist king Langdharma* $,_{IGNORE}$ $(_{REL}$*who probably reigned*$_{VIO}$ *from 838 to 841 CE."*

All that remains is to identify constituents ends with algorithm 4.6:

"The original fortress $,_{IGNORE}$ $(_{REL}$*known*$_{VIO}$ *as Gyel-khar-tse*$)_{REL}$ $,_{CLOSE}$ *was attributed*$_{VIO}$ *to Pelkhor-tsen* $(_{RELA},$ *son of the anti-Buddhist king Langdharma*$)_{RELA}$ $,_{IGNORE}$ $(_{REL}$*who probably reigned*$_{VIO}$ *from 838 to 841 CE.*$)_{RELA}$*"*

The final result is then given by:

"The original fortress ,($_{REL}$known as Gyel-khar-tse)$_{REL}$, was attributed to Pelkhor-tsen ($_{RELA}$, son of the anti-Buddhist king Langdharma)$_{RELA}$, ($_{REL}$who probably reigned from 838 to 841 CE.)$_{REL}$"

Note that this final result is slightly wrong. The last relative clause constituent should be embedded into the previous apposition constituent, because it refers to "*anti-Buddhist king Langdharma*". This nicely illustrates that a weakness of the rule based approach is the identification of embedded structures.

5 Sentence Constituent Identification Using Machine Learning

The previous chapter introduced some hand-crafted rules for sentence constituent identification (SCI). Hand-crafted rules are easy to comprehend and usually so are their final results. However, the more rules there are, the more difficult it becomes to manage and develop them. One must be careful not to create contradictions and the interaction of different rules is sometimes not easy to understand. In this chapter we present another approach to SCI based on supervised machine learning. Instead of using hand-crafted rules we train machine learning classifiers to identify possible starts and ends of constituents and use an inference algorithm to obtain a solution.

This chapter is organized as follows. In section 5.1 we provide an intuitive overview and introduce machine learning to an extent that is required to understand the rest of this chapter. We also give a detailed description of the approach. In section 5.1.1 the exact input for our classifiers and the way we train them is described. To select the best features for our classifiers and to obtain an understanding of how they work we used a set of different methods that are described in section 5.3. In section 5.4 the inference algorithm we use to find an optimal solution is described. Throughout the chapter we restrain from details that are implementation specific and provide those in the final section. Results and a comparison to the rule based approach outlined in chapter 4 can be found in chapter 7.

5.1 Overview

Machine learning has become an integral part of natural language processing. We do not further elaborate on the historic reasons and refer the interested reader to [Marquez and Salgado (2000)] which gives a good overview of the topic. For us it is important to realize that current state-of-the-art systems achieve good results by applying machine learning to tasks similar to ours, for example the task of clause identification, as presented in chapter 2.2. We are therefore keen to evaluate and compare an SCI approach based on machine learning with the rule-based approach of the previous chapter.

Instead of using hand-crafted rules we apply proven machine learning techniques to train classifiers that help us to identify possible sentence constituents. A classifier tries to predict a certain class for an observation, for example, whether a certain word in a sentence starts a list item constituent or not. Consider the following sentence:

"*Having received the appointment on the recommendation of Truman aide Donald Dawson, Whitehair was seen as a political appointment and was unpopular with the admirals of the United States Navy.*"

Given we have some trained machine learning classifiers that tell us at which word certain types of constituents might start or end, applying those to the sentence might give the following, partially incorrect result:

"*Having received* $(_{LIT}$ *the appointment on the recommendation of Truman aide Donald Dawson* $(_{SEP}$ *,* $)_{SEP}$ *Whitehair* $(_{LIT}$*was seen as a political appointment*$)_{LIT}$ *and* $(_{LIT}$ *was unpopular with the admirals of the United States Navy* $)_{LIT}$*.*"

In general, we assume it is impossible to train classifiers that are always correct. Therefore, we treat the results of the classifiers as suggestions. This is a good starting point but there are still a number of different ways we can incorporate these suggestions into a final solution. We could arbitrarily ignore one or more of them - they are suggestions after all. Furthermore, the final set of constituents still needs to adhere to some structural constraints. The identified constituents are allowed to embed but must never overlap. We can therefore design an algorithm that identifies all admissible constituent sets from the suggestions and selects an optimal solution. The optimal solution should then coincide with the correct solution:

"*Having received the appointment on the recommendation of Truman aide Donald Dawson* $(_{SEP}$*,* $)_{SEP}$ *Whitehair* $(_{LIT}$ *was seen as a political appointment* $)_{LIT}$ *and* $(_{LIT}$ *was unpopular with the admirals of the United States Navy* $)_{LIT}$*.*"

We can structure this approach into two phases. The first one is a filtering phase that identifies possible constituent starts and ends. The second phase applies an inference algorithm to select an optimal solution adhering to the structural constraints. We point out that the idea of filtering and inference is not new, but has been applied previously for similar problems, e.g. the task of clause identification [Carreras (2005)].

We next provide a short introduction to the machine learning techniques and foundations that matter here. For a detailed description and holistic background we refer the reader to one of the many and easily found detailed descriptions of Support Vector Machines (SVM), for example [Burges (1999)]. Having introduced the machine learning background we then complete this introductory section by giving a more detailed description of our approach.

5.1.1 Machine Learning and Support Vector Machines

In general machine learning can broadly be classified into three categories: *supervised*, *unsupervised* and *reinforcement* learning. Unsupervised learning is concerned with finding anomalies or hidden regularities in data and group together similar classes, often called Clustering. In contrast to supervised learning it requires no training data, that consists of data samples with a correct solution, often called label, assigned. Supervised learning is concerned with finding the answer or label to a question for which example instances are provided in the training data. A prominent example is the task of classification, i.e. given a description of an example without correct label decide to which of two or more classes it belongs, even (and especially) if the exact example was never seen before. Reinforcement learning is concerned with how an agent has to react in certain situations in order to maximize a reward it receives. This process can be performed on-line (learning by doing) and a prominent example is "learning to balance a pole".

From the above it should be clear that only supervised learning is of interest to our task. A large number supervised learning methods exist in this area and we refer to [Carreras (2005)] for a good overview, especially for natural language processing. One of the most prominent methods are Support Vector Machines (SVM). In natural language processing, features[8] are constructed from words or other syntactic structures. These are numerous, but when looking at a certain instance most of them are empty. SVMs have the advantage of providing good performance in these high-dimensional, *sparse* feature spaces common in NLP, even when there are few training examples [Joachims (1998)]. Furthermore it is a proven method used in a large number of different applications. Libraries for implementation are freely available. The following is a short and shallow description, but holds in general and is not specific to our actual implementation.

We start by formulating the problem to be solved. We want to induce a function h mapping from an input space X to an output space Y.

$$h : X \to Y$$

For example, X could be the set of all possible words and the surroundings[9] they appear in and $Y = \{+1, -1\}$ indicates whether the word starts a list item constituent or not. This is a typical binary classification problem (as apposed to multi-class classification where $|Y| > 2$) and consequently an SVM providing this function is also called *binary classifier*. For Support Vector Machines the input is a vector in $\mathbb{R}^d$ also called feature vector. The training data of size n, used for learning a SVM,

[8]In machine learning a feature represents an attribute or characteristic of some instance. For example, features of a tree might be its size, width, height, number of leaves, color of leaves etc.

[9]This refers to the environment of a word within a sentence, i.e. surrounding words and their grammatical forms etc.

consists of tuples (r_i, y_i) for $i = 1...n$ and $r_i \in \mathbb{R}^d, y_i \in Y = \{+1, -1\}$. Mapping from the actual words and their surroundings in X to a feature vector in $\mathbb{R}^d$ is described in the next section. For now we just assume there is a feature extraction function $\varphi : X \rightarrow \mathbb{R}^d$ that, given a word and surrounding it occurs in, extracts information and maps it to $\mathbb{R}^d$. A Support Vector Machine then tries to find a separating hyperplane in $\mathbb{R}^d$ where ideally, on each side only vectors belonging to the class $+1$, -1, respectively, lie. For predicting the class of an unseen example it is then just a matter of finding its representation in $\mathbb{R}^d$ by applying φ and then determining on which side of the hyperplane the resulting vector lies. Figure 5.1 shows an example of a hyperplane separating instances of two classes.

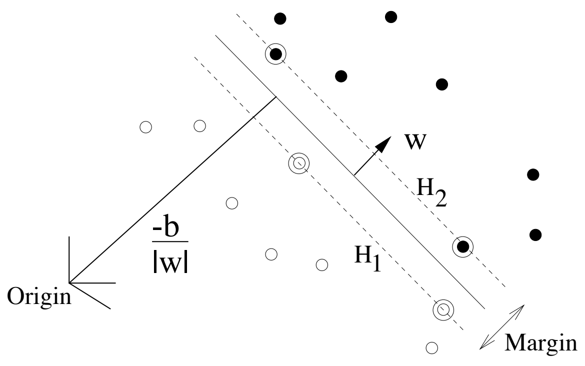


Figure 5.1: SVM hyperplane for linearly separable case, from [Burges (1999), p.9]. The hyperplane splits the space, separating between the classes of black and white instances with maximum margin. Support vectors are circled.

The hyperplane is computed based on a set of constraints to create a maximum margin between the two classes, that is: maximizing the minimal distance to all close vectors. It is the fact that the hyperplane can be described using a small set of the training vectors, the so called *support vectors*, that is responsible for the name *Support Vector* Machine. Of course, there are cases where data is not linearly separable. Intuitively, for example, if we cannot split the classes using a line in the two dimensional case, as shown in figure 5.1, but would need a circle. SVMs can then apply so called *kernel functions,* which implicitly compute the dot product of feature vectors in a higher-dimensional space, where they are possibly linearly separable [Burges (1999)]. This happens without ever explicitly computing the mapping and is therefore efficient. We say the dot product of a support vector x_i and a candidate x in a space $\mathcal{H}$, that is mapped to by $\Phi : X \rightarrow \mathcal{H}$, is given by a function $K(x_i, x) =< \Phi(x_1), \Phi(x) >$. In the linear case the function K is the dot product of the two vectors $K(x_i, x) =< x_i, x >$ whereas for the common *Gaussian radial basis function* (RBF) it is a kernel function $K(x_i, x) = e^{-\gamma||x_i - x||^2}$ for a selected parameter γ. We do not further consider the RBF kernel here but come back to it in section 5.3.

The previously mentioned hyperplane is computed by solving a set of optimization problems we do not consider here. The result of training a Support Vector Machine then consists of a set of support vectors that have parameters assigned and without further ado we present a final equation (stemming from the dual-notation used by SVMs) used to classify a feature vector x:

$$h(x) = sgn(\sum_{i=1}^{m} \alpha_i \gamma_i K(x_i, x) + b)$$

Each of the support vectors $i = 1...m$ has real-valued parameters α_i and γ_i assigned. The bias b is another computed parameter of the hyperplane. Taking the sign of the sum then intuitively decides on which side of the hyperplane the vector lies and provides the final result. This concludes our very short introduction.

5.1.2 Detailed Approach

Using the knowledge about machine learning we can now outline the approach in more detail. For this we recall that in chapter 3, we defined four different types of constituents: relative clause, apposition, list item and separator constituents. Besides the separator constituent, which consists of only one word, all can span several words. We therefore train seven *constituent start* and *end classifiers* to recognize possible beginnings and endings of our constituent types:

- classifiers $c_{REL(}$ and $c_{REL)}$ for identifying the possible start respectively end of a relative clause constituent
- classifiers $c_{RELA(}$ and $c_{RELA)}$ for identifying the possible start respectively end of an apposition constituent
- classifiers $c_{LIT(}$ and $c_{LIT)}$ for identifying the possible start respectively end of a list item constituent
- classifier c_{SEP} for identifying a possible separator constituent

Each of the SVMs is a binary classifier of the form $c : \mathbb{R}^d \rightarrow \{+1, -1\}$ predicting to which respective class a provided feature vector belongs. For example, the classifier $c_{LIT(}$ predicts whether it represents a list item constituent start or not. As we did for for the rule based approach, we also assume our text has *part-of-speech* as well a *text chunking* tags assigned. We then iteratively apply one of the classifiers on all feature vectors extracted from the first and last word of each text chunk of a sentence using some feature extraction function $\varphi_w : X_w \rightarrow \mathbb{R}^d$. Here X_w is the set of all possible words and their surrounding within a sentence they appear in. Restricting ourselves to the first and last word of each text chunk is reasonable. As we outlined in chapter 4, constituents only properly embed text chunks and never split them. So the only feasible start and end points of a constituent are the first or last word of each text chunk. We apply the classifiers in increasing order of difficulty: $c_{SEP}, c_{REL(}, c_{REL)}, c_{RELA(} c_{RELA)}, c_{LIT(}, c_{LIT)}$. The results of a classifier are always provided as part of the features input to its following classifiers. This is reasonable because, for example, identifying separators works better, as we will see in the evaluation. Having identified a word as a separator at the first iteration gives the following classifiers a good clue that the same word does not start another constituent.

Because we assume achieving perfect, error-less classifiers is unrealistic we treat the results as indications or suggestions. A hypothetical result of applying our classifiers might be the following set where we denote that the word at position i has been classified as a possible start of constituent type k with $s_{k,i}$, and analogously for the end $e_{k,i}$

$$\{s_{LIT,1}, s_{LIT,4}, s_{REL,6}, e_{LIT,8}, e_{REL,10}\}$$

These now provide a set of possible constituents we can form from these suggestions, where a constituent of type x is a pair $(s_i, e_j)_x$ with $i \leq j$:

$$\{(s_1, e_{10})_{REL}, (s_6, e_{10})_{REL}, (s_4, e_8)_{LIT}\}$$

We say two constituents $c = (\mathrm{s}_1, e_1)_x$ and $d = (s_2, e_2)_x$ overlap iff $s_1 < s_2 \leq e_1 < e_2$ or $s_2 < s_1 \leq e_2 < e_1$, and we denote it by $c \sim d$. For our task, the constituents are not allowed to overlap, however they may embed. The final result is therefore part of the subset of structurally possible constituents, adhering to this constraint. In our example this is one of the following:

$$\{\}, \{(s_1, e_{10})_{REL}\}, \{(s_4, e_8)_{LIT}\}, \{(s_6, e_{10})_{REL}\},$$
$$\{(s_{1,}, e_{10,})_{REL}, (s_{,4}, e_{,8})_{LIT}\}, \{(s_{1,}, e_{10,})_{REL}, (s_{,6}, e_{10})_{REL}\}$$

We can see that the possible start or end of a constituent does not exactly say how many constituents start or end there, it may be several. Using the constraints an inference algorithm can, based on the suggestions, decide which solutions are admissible. It then selects an optimal one, ideally coinciding with the correct one. To allow the inference of an optimal solution we train three *constituent classifiers:*

- c_{REL}: identifying whether we have a relative clause constituent at hand
- c_{RELA}: identifying whether we have a apposition constituent at hand
- c_{LIT}: identifying whether we have a list item constituent at hand

In contrast to the previous classifiers we are no longer classifying single words but a whole span of words that make up the constituent. Therefore, in the mapping $h : X_c \to Y$, X_c denotes the set of pairs of words (start and end of a constituent) and their environments. The feature extraction function $\varphi_c : X_c \to \mathbb{R}^d$ works accordingly.

The inference algorithm used applies a mapping to the maximum weight independent set problem. We construct a graph where each node corresponds to a constituent and has a weight assigned based on the outcome of the constituent classifiers. An edge between two nodes is inserted for all pairs of constituents that overlap. As a result the independent set with maximum weight consists of constituents that do not overlap, and whose sum of weights is maximum, and should therefore closely resemble the correct solution. The inference algorithm is described in detail in section 5.4.

5.2 Feature Selection and Training Classifiers

This section describes the features extracted for each of the classifiers. We start by describing how, in general, features are represented and then provide the exact features used for identifying constituent start and ends followed by the features extracted for recognizing complete constituents. The last part of this section describes how the training set is generated as well as how the classifiers are trained.

5.2.1 Feature Representation

Features describe characteristics and attributes of some instance. In general features are represented in $\mathbb{R}^d$ for Support Vector Machines and each of the d dimensions of a feature vector corresponds to one feature. Only real-valued features are used which requires transforming string based or categorial properties to a numerical domain.

We use binary valued features. Each feature indicates whether a certain property is present in the instance or not. For example, a feature might be the property of a tree indicating whether it is evergreen or not. A fir would have a feature value of 0, opposed to an oak tree with a feature value of 1. Moving to the domain of natural language processing a feature might be whether the word we are looking at has the part-of-speech tag "DT" (a determiner), and another feature might be whether the next word has the POS-tag "NN" (a noun) and so on. Depending on the amount of information or properties that are represented as features this results in a very high-dimensional feature space. In addition the resulting feature vectors are sparse. Only a small amount of features has a non-trivial value because a lot of the features exclude each other or are simply not present. As we will see SVMs are capable of handling such situations well.

5.2.2 Feature Selection for Constituent Start and End Classifiers

Constituent start and end classifiers are used to classify a word in a sentence. To provide the classifiers with a feature vector we extract information from the immediate surrounding of the word[10]. Figure 5.2 shows a graphical example with windows

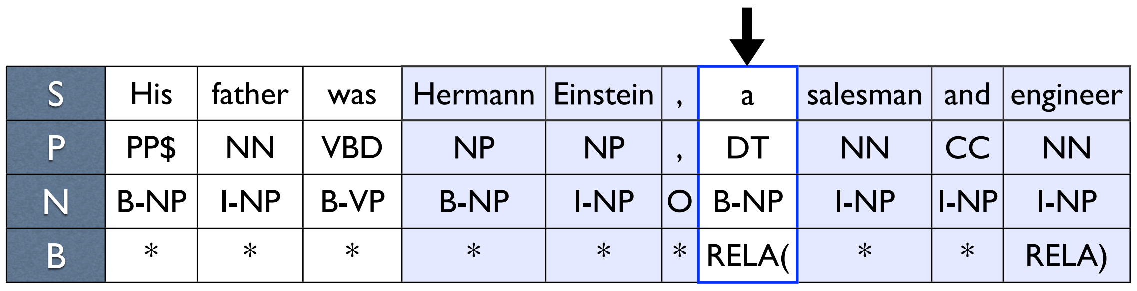
[10]This is in essence the function $\varphi_w : X_w \to \mathbb{R}^d$ of section 5.1.2.

centered around the currently classified word "*a*". Note that each window can have a different size on each level of abstraction. Using the feature window we try to capture the function of the word in the sentence. Instead of using all possible features we can think of we restrict ourselves to reasonable choices to keep the number of features as small as possible. For a sentence consisting of a sequence of n words $[w_1, ..., w_n]$, we use the following windows for the j-th word w_j of a sentence:

- **word window** with half-size 4: for $[w_{j+i}]_{i=-4}^{i=4}$ the word if it is contained in $\{$*including, such as, as well as, and, or, but, thus, however, that, who, which, whose, after, before*$\}$.
- **POS-tag window** with half-size 4: for $[w_{j+i}]_{i=-4}^{i=4}$ the POS-tag
- **text chunk window** with half-size 4: for $[w_{j+i}]_{i=-4}^{i=4}$ the type of text chunk
- **constituent tag window** with half-size 6: for $[w_{j+i}]_{i=-6}^{i=6}$: the constituent tag, if it is non-trivial (i.e. not "*")

These feature windows worked best in our evaluations. However, they are not assumed to be optimal and only represent a first result. Instead of using any observed word in the word window we restrict ourselves to a set of words to reduce the number of resulting features. The depicted set of words included represents an initial choice of words that seemed reasonable, for example those that start an enumeration "*including*", "*such as*" or those that end one "*and*", "*as well as*". However, further evaluation is necessary to conclude how important they actually are.

Consider the sentence shown in figure 5.2 centered at the word "*a*" . One of the features extracted would be "the POS-tag of the word two to the right is CC", another one "the text chunk type two to the left is NP" and yet another one "the word two to the right is and". Each of the features would have an index in the feature vector and if the respective feature is present for an instance the value in the vector would be 1 and 0 else.



S	His	father	was	Hermann	Einstein	,	a	salesman	and	engineer
P	PP$	NN	VBD	NP	NP	,	DT	NN	CC	NN
N	B-NP	I-NP	B-VP	B-NP	I-NP	O	B-NP	I-NP	I-NP	I-NP
B	*	*	*	*	*	*	RELA(	*	*	RELA)

Figure 5.2: The feature extraction windows is marked blue and centered at the word "*a*". The first row shows the words, the second one the part-of-speech tags, the third one the text chunking tags in B-I-O notation and the last one the tags for sentence constituents.

5.2.3 Feature Selection for Constituent Classifiers

Constituent classifiers are applied to complete constituents. Constituents represent a span of a sentence given by the start and end of the constituent. To determine whether a given span denotes a valid constituent we use the start and end word as well as the span itself to extract features[11]. The same feature windows as in the previous subsection are used to extract features from the first and last word. Of course, the features for start and end word must be distinguishable, i.e. the features "the start word has POS-tag DT" and "the end word has POS-tag DT" denote two different features.

Furthermore we extract a pattern of text chunk tags between the two words. We only consider the tags if they are in the set *{NP, VP, SBAR, ADJP, O, PP, ADVP}*. This is again based on initial evaluations. An inclusion of additional tags or, for example, POS-tags in the patterns resulted in decreased classifier performance. The tags in the pattern are delimited by the '+' character, for example the pattern "*PP+NP*" denotes that the constituent consisted of a prepositional phrase followed by a verb phrase. Since all features are binary each observed pattern will be represented by a different feature.

Consider the sentence shown in figure 5.2 and assume we classify the span between the words "*was*" and "*engineer*" to find out if it represents a list item. The features would consist of the pattern of that span, "*VP+NP+O+NP*", as well as a feature extraction window centered at "*was*" and "*engineer*".

The intuition behind this is that a certain constituent type often consists of a certain pattern. For example, list items often only consist of noun phrases, as do appositions. Relative clause consistent however often contain a verb which helps to distinguish them. The features extracted from the start and end word can provide further characteristics.

5.2.4 Generating Training Sets and Training Classifiers

For training the classifiers we have manually annotated a ground truth of 200 sentences (4296 words) with golden constituent starts and ends. The sentences have been carefully selected from the English Wikipedia in terms of the constituents they contain, to provide a good training set. They are annotated with part-of-speech tags, text chunking tags and a set of tags for identifying constituent start and ends corresponding to their name. The part-of-speech and text chunking tags have been automatically provided by respective tools, i.e. TreeTagger and YamCha as introduced in chapter 3.3.

Figure 5.2 already showed an example of a sentence in the ground truth. Creating the actual training set used to train the classifiers is then straightforward.

[11]This is in essence the function $\varphi_c : X_c \rightarrow \mathbb{R}^d$ presented in the section 5.1.2.

For each constituent start and end classifier we extract the features of each first and last word of a text chunk in the ground truth sentences from left to right. We assign the label +1 if the respective word starts or ends the constituent and -1 else wise. Together with the extracted feature vector this is a learning instance and overall we obtain about 4296 learning instances. During feature extraction we must account for the order the classifiers will be applied later. Because the features cover constituent tags it is possible that the output of a previous classifier shows up as the input to another classifier[12] and the classifiers must therefore also be trained accordingly.

For the constituent classifiers we proceed similarly. For each constituent classifier we select the matching constituents from the ground truth, extract the features as previously described and assign the label +1 to obtain a positive learning example. To create negative examples we construct “false” constituents from the possible combinations of all golden constituent starts and ends within a sentence (that resemble no constituent of the type we are training for).

The classifiers are all using the RBF-Kernel and we perform a grid search for the best kernel parameter γ and misclassification cost C using 5-fold cross validation on the training set. The learning module gets as input a file containing the training set and outputs a model file that contains resulting support vectors and parameters which are needed for classification. For further details we refer to the last section of this chapter.

5.3 Classifier Analysis

In this section we analyze our classifiers in order to obtain an understanding of the influence different features have on the decisions. By applying a trained SVM to a feature vector we can get a prediction to which class it belongs, e.g. whether the word we are looking at starts a list item constituent or not. However, the rationale behind the classification is hidden in the mathematical computations and we cannot directly comprehend what caused or influenced the decision. We therefore present a set of methods that allow some insight on the influence of features in this section. This also helped in selecting the final selection of features we used. For each method we provide a few example results, however the methods are of course applicable to all classifiers. The focus lies on the presentation of methods and not the interpretation of results. With the large amount of features and classifiers this is out of the scope of this thesis.

An important factor is what type of kernel is applied because, as we will see, for the linear kernel the question can easily be answered but for the RBF-Kernel it is disproportionately more difficult. However, we start with a method that can be performed before even training the classifiers and is thus independent of the used

[12]As a matter of fact, because the constituent window spans previous constituent tags, the decision of a classifier can be part of the same classifiers input in the classification of a following word.

kernel. The following sub-sections then deal with the classifiers using the linear and RBF-Kernel respectively.

5.3.1 Analysis on Training Set

This method simply analyzes the learning instances in the training set, even before any training occurs. We use the following computation suggested in [wei Chen (2005)] to produce the so-called *F-Score* for a feature i:

$$F(i) \equiv \frac{\left(\bar{\mathbf{x}}_{\mathbf{i}}^{(+)} - \bar{\mathbf{x}}_{\mathbf{i}}\right)^2 + \left(\bar{\mathbf{x}}_{\mathbf{i}}^{(-)} - \bar{\mathbf{x}}_{\mathbf{i}}\right)^2}{\frac{1}{n_+-1}\sum_{k=1}^{n_+}\left(x_{k,i}^{(+)} - \bar{\mathbf{x}}_{\mathbf{i}}^{(+)}\right)^2 + \frac{1}{n_- - 1}\sum_{k=1}^{n_-}\left(x_{k,i}^{(-)} - \bar{\mathbf{x}}_{\mathbf{i}}^{(-)}\right)^2}$$

Remember that each feature vector of the training set is in $\mathbb{R}^d$ so the i-th feature occurs in the i-th row of the vector and that furthermore each vector has a label in $Y = \{+1, -1\}$ assigned to specify its class. Now for the i-th feature $\bar{\mathbf{x}}_{\mathbf{i}}, \bar{\mathbf{x}}_{\mathbf{i}}^{(+)}, \bar{\mathbf{x}}_{\mathbf{i}}^{(-)}$ define the overall average value, average value of the positive instances and average value of the negative instances. In addition $x_{k,i}^{(+)}$ is the value of the i-th feature in the k-th positive instance and likewise $x_{k,i}^{(-)}$ is the value of the i-th feature in the k-the negative instance.

In the numerator we measure the discrimination of the average value of positive and negative instances against the overall average. Intuitively if the feature only appears in either the positive or the negative instances this will result in a large value. In the denominator we measure the discrimination within each of the sets in form of their variance. A feature value that is constant within the positive or negative instances results in a small value. In total a feature that only appears in either the positive or negative instances and has a constant value therein will achieve a large F-score and it is intuitively comprehensible that this feature is a good indicator for the respective class.

Table 5.1 and 5.2 show the top five features for the separator classifier and the relative clause constituent start classifier.

We shortly explain the illustration. Index denotes the position in the feature vector and description provides a textual translation of the feature. The description is split into three parts by a colon. The first part identifies the type of window, i.e. "*ww*" for word window, "*cw*" for chunk window and "*pw*" for POS-tag window. The second part gives a relative position. For example, for the POS-tag window 0 for the current word, 1 for the next word and -1 for the previous word in the sentence. The last part denotes the actual observation at that position, e.g. "*but*" if the word occurs or "*VP*" if there is a verb phrase at that position. For the translation of all tags we refer to the work cited in the introduction of text chunking and POS-tagging in chapter 3.3.

Index	Description
475	ww:0:but
105	cw:0:O
123	pw:0:CC
20	cw:2:VP
342	cw:0:SBAR

Table 5.1: The top five features for the separator classifier according to F-Score.

Index	Description
256	ww:0:which
185	pw:-1:,
252	pw:0:WDT
420	ww:0:including
134	cw:-1:O

Table 5.2: The top five features for the relative clause start classifier according to F-Score.

What we can for example see is that a good indication for the separator constituent classifier is if the current word is "*but*" and it is followed by a verb phrase but not immediately, e.g. in "*... but he went home*". Good indications for the relative clause constituent start classifiers are whether the current word is "*which*", and the previous POS-tag was a comma. for example in "*The car, which is green*".

5.3.2 Analysis with Linear Kernel

Although we do not use the linear kernel in our implementation[13] we shortly show how we can determine relevant features in this case. Remember that the classification works as follows:

$$c(x) = sgn(\sum_{i=1}^{m} \alpha_i \gamma_i K(x_i, x) + b)$$

and that $i = 1...m$ ranges over the support vectors and all corresponding α_i, γ_i are a result of the training. Furthermore, the linear kernel is defined as the dot product: $K(x_i, x) = x_i \cdot x$. Therefore, the following holds:

$$c(x) = sgn(\sum_{i=1}^{m} \alpha_i \gamma_i x_i^T x + b)$$

$$c(x) = sgn(\sum_{i=1}^{m} (\alpha_i \gamma_i x_i^T) x + b)$$

We can now calculate $\sum_{i=1}^{m} (\alpha_i \gamma_i x_i)$ to obtain a weight vector w. In fact, we now have a trivial decision function for classification:

$$c(x) = sgn(w^T x + b)$$

where a hyperplane can be described by a vector w normal to it and its distance to the origin is influenced by b. So by calculating w and comparing its values we can conclude on the relative importance of features.

Weight	Index	Description
0.91	20	cw:2:VP
0.7	105	cw:0:O
0.64	102	pw:1:VBZ
0.63	19	cw:1:NP
0.59	342	cw:0:SBAR

Table 5.3: The top five features in favor of a classification as a separator classifier according to their weight.

Weight	Index	Description
0.81	152	pw:-2:IN
0.66	85	cw:1:O
0.57	257	pw:3:NN
0.55	53	cw:2:O
0.52	18	cw:0:VP

Table 5.4: The top five features against classification as a separator classifier according to their weight.

Table 5.3 and 5.4 show the top five features in favor and against a classification as separator constituent according to their weight.

The description of the illustration can be taken from the previous section and we leave the detailed interpretation to the reader. However, we note that some of the features of the previous tables 5.1 and 5.2 for the F-Score based analysis reappear.

5.3.3 Analysis with Radial Basis Function Kernel

Unfortunately the analysis for the RBF-Kernel can not be performed in the same way as for the linear kernel. The kernel computes a dot product in an implicitly mapped features space and even if we were able to compute a weight vector, as for the linear kernel, it would be in the mapped feature space, where we are unable to interpret it.

We therefore follow two different approaches. On the one hand we try to find the most influential support vectors for a decision as described in [Barbella et al. (2009)]. This allows the intuition to say that a learning instance, or rather its feature vector, was classified a certain way because it is very similar to these support vectors. Especially the RBF-Kernel allows this intuition. On the other hand we try to find a feature vector in the training set with maximum distance from all support vectors in order to argue that it is a very good example of that class. Both methods do not allow a direct conclusion on relative importance of features, because they only compare complete vectors. Nonetheless this allows a valuable insight on the decisions made.

We remember that the final classification was based on the following function:

$$c(x) = sgn(\sum_{i=1}^{m} \alpha_i \gamma_i K(x_i, x) + b)$$

[13]Of course, we experimented with the linear kernel during development.

It is now easy to see that the i-th support vector contributes exactly:

$$p_i(x) = \alpha_i \gamma_i K(x_i, x)$$

to the final sum. This is what [Barbella et al. (2009)] define as the *pull* of a support vector following the intuition that the support vector pulls the vector in one direction or the other. We furthermore realize that the RBF-kernel $K(x_i, x) = e^{-\gamma||x_i - x||^2}$ gives a very good similarity measure [Barbella et al. (2009)], especially its value is maximal (1) if $x_i = x$ and decreases with increasing distance of x_i and x. A large pull is therefore caused either by large $\alpha_i \gamma_i$, which result from training the SVM, or because the distance to the support vector is small. Computation is trivial and we compute the pull for each support vector against all instances in the test set used in the evaluation chapter. We then select the support vectors that contributed most to a classification in favor or against a certain class. We show two vectors with greatest influence for and against a classification for the separator constituent classifier in tables 5.5 and 5.6.

Again the interpretation of the description has been covered in section 5.3.1 and we do not further interpret the results but give an intuition. The support vector in table 5.5 for example, stems from the sentence "*Checchi wanted to extend the death penalty [...]* ***and*** *he wanted a drug rehabilitation programs [...*]" with the word in bold showing the word being classified. A clear example where a separator constituent should be placed. The support vector in table 5.6 stems from the word "*and*" in bold in the sentence *"All Dogs Go to Heaven 2 [...] is a 1996 American animated family film,* ***and*** *a sequel to United Artists ' 1989 animated film All Dogs Go to Heaven*". A clear example of a list item start, but not of a separator constituent.

To provide another intuition we also compute the feature vectors of the test set with greatest distance to all support vectors. This allows finding best candidates for a certain class, because we assume they are farthest away from all support vectors of their respective class. To compute the distance we realize that the following holds:

$$\begin{aligned}
|x - y|^2 &= (x - y) * (x^T - y^T) \\
&= x * x^T + y * y^T - y * x^T - x * y^T \\
&= < x, x > + < y, y > -2* < x, y >
\end{aligned}$$

and therefore:

$$|\Phi(x) - \Phi(y)| = K(x, x) + K(y, y) - 2 * K(x, y)$$

Index	Description
115	cw:-1:Oă
30	cw:-2:NPă
84	cw:-3:VPă
149	cw:-4:Oă
105	cw:0:Oă
19	cw:1:NPă
20	cw:2:VPă
21	cw:3:NPă
86	cw:4:PPă
167	pw:-1:,ă
57	pw:-2:NNSă
119	pw:-3:NNă
108	pw:-4:VBă
101	pw:0:CCă
273	pw:1:PPă
155	pw:2:VBDă
49	pw:3:DTă
50	pw:4:NNă
96	ww:0:andă

Table 5.5: The support vector contributing most in favor of a separator classification according to pull. Only the features with non-trivial value are shown.

Index	Description
115	cw:-1:O
30	cw:-2:NP
84	cw:-3:VP
149	cw:-4:O
105	cw:0:O
19	cw:1:NP
116	cw:2:PP
21	cw:3:NP
10	cw:4:NP
167	pw:-1:,
110	pw:-2:NN
119	pw:-3:NN
151	pw:-4:JJ
101	pw:0:CC
70	pw:1:DT
71	pw:2:NN
28	pw:3:TO
124	pw:4:NP
96	ww:0:and

Table 5.6: The support vector contributing most most against a separator classification according to pull. Only the features with non-trivial value are shown.

Now because $K(x, x)$ is trivially 1 for the RBF-Kernel:

$$|\Phi(x) - \Phi(y)| = 2 - 2 * K(x, y)$$

We now compute the minimum distance of each feature vector to all support vectors of their respective class, i.e. $min_{i \in sv_+} \alpha_i \gamma_i (2 - 2 * K(sv_i, y))$ for feature vectors y that belong to the class, and analogously $min_{i \in sv_-} \alpha_i \gamma_i (2 - 2 * K(sv_i, y))$ for feature vectors y that do not belong to the class. sv_+ and sv_- denote the support vectors of the positive and negative class respectively. Sorting the two lists by descending minimum distance leaves the feature vectors farthest away at the top. Tables 5.7 and 5.8 show the two feature vectors with maximal minimal distance to all support vectors of their respective class for the separator classifier. Again only non trivial values in the vector are shown.

Index	Description
107	cw:-1:O
30	cw:-2:NP
166	cw:-3:PP
167	cw:-4:PP
324	cw:0:SBAR
19	cw:1:NP
20	cw:2:VP
97	cw:3:PP
10	cw:4:NP
157	pw:-1:,
153	pw:-2:NP
156	pw:-3:NP
179	pw:-4:VBN
128	pw:0:IN
269	pw:1:PP
84	pw:2:VBZ
202	pw:3:VBN
106	pw:4:IN

Table 5.7: The feature vector of positive test set instances furthest away from all support vectors in favor of a separator constituent.

Index	Description
237	bw:-1:SEP
107	cw:-1:O
117	cw:-2:O
51	cw:-3:NP
200	cw:-4:ADVP
6	cw:0:NP
7	cw:1:VP
8	cw:2:NP
97	cw:3:PP
10	cw:4:NP
102	pw:-1:CC
161	pw:-2:,
111	pw:-3:NN
110	pw:-4:DT
1	pw:0:PP
2	pw:1:VBD
3	pw:2:CD
115	pw:3:IN
98	ww:-1:and

Table 5.8: The feature vector of the negative test set instances furthest away from all support vectors against a separator constituent.

We note two things. The first feature shown in table 5.8 "*bw:-1:SEP*" shows that a particular good instance of a word that is not a separator constituent is one that directly follows a separator constituent. Furthermore in table 5.7 the start of a sub-ordinate clause, "*cw:0:SBAR*", followed by a noun phrase, "*cw:1:NP*", and a verb phrase, "*cw:2:VP*", is a particular good example of a separator constituent. An example is "*The penalty of life imprisonment is not provided for in the Revised Penal Code , **although** it is imposed by other penal statutes [...]*" with the separator constituent in bold.

5.4 Inference Algorithm

In this section we present an algorithm to infer an optimal solution given the suggested constituent start and end points of our classifiers. We achieve this by a mapping to the *maximum weight independent set probl*em (MWIS)[14]. We start by giving some preliminaries and then describe the mapping. The maximum independent set problem is NP-complete, so no efficient algorithm for solving it is known. We therefore provide an approach based on enumeration and a greedy approach for solving it.

5.4.1 Preliminaries

Let $G(V, E)$ be an undirected graph with no multiple edges, vertices V and edges E. Moreover let $w(i)$ be a positive weight assigned to each vertex i. With $|V|$ we denote the cardinality of V and with $\omega(V)$ the sum of weights $w(i)$ of all vertices in V, i.e. $\omega(V) = \sum_{i \in V} w(i)$.

Definition 3. A set of vertices $V' \subseteq V$ is *independent* if for all distinct vertices $u, v \in V'$ it holds that$\{u, v\} \notin E$, i.e. no two vertices of V' are adjacent.

Definition 4. The *maximum weight independent set (MWIS)* problem is the problem of finding an independent set $V' \subseteq V$ such that the sum of all weights, $\omega(V')$, is maximum among all independent sets of V. If we assign the same weight w to all vertices of V the problem is called the *maximum independent set problem* and as a result we are interested in finding an independent set $V' \subseteq V$, such that $|V'|$ is maximum among all independent sets of V.

5.4.2 Mapping to the Maximum Weight Independent Set Problem

We map the problem of finding an optimal structure of constituents to the maximum weight independent set problem as follows.

We are given a set of constituent start and end suggestions $S = \{s_{x,i}, e_{x,,j} ...\}$ where $s_{x,i}$ denotes the start of constituent type x at the i-th word in the sentence and likewise $e_{x,j}$ the constituent end at the j-th word. A possible constituent is a pair $c = (s_i, e_j)_x$ such that $i \leq j$. We denote the set of all possible constituents based on these suggestions as $C = \{(s_i, e_j)_x | s_{x,i}, e_{x,j} \in S \land i \leq j\}$. Two constituents $c = (\mathrm{s}_1, e_1)_x$ and $d = (s_2, e_2)_x$ overlap iff $s_1 < s_2 \leq e_1 < e_2$ or $s_2 < s_1 \leq e_2 < e_1$, and we denote it by $c \sim d$. We construct a graph $G(V, E)$ as follows:

1. For each possible constituent $c \in C$ add a vertex to G
2. Apply the classifier c_x to the constituent c of type x, and if, according to the classifier, c belongs to x assign the weight 100, and 1 else.
3. Add an edge between all distinct vertices $u, v \in V$ if their corresponding constituents c and d overlap, i.e. $c \sim d$.

[14]This is not to be confused with the **maximal** independent set problem.

4. Add an edge between all distinct vertices $u, v \in V$ if their corresponding constituents c and d have the same start and type, i.e. using the notation for c and d from above if $s_{x,i} = s_{y,j}$

Step 1 is self-explanatory. Step 2 assigns a weight based on the outcome of our constituent classifiers. Because currently the constituent classifiers are not robust enough we cannot ignore possible other suggestions, but assign a small weight. As a result a correct suggestion of the constituent classifier is practically guaranteed to be included in the solution, but other possible suggestions are not ignored completely. With rule 3 we avoid that one suggestion causes the start of two constituents. For example, when $S = \{s_{REL,1}, s_{REL,4}, e_{REL,3}, e_{REL,10}\}$ it is always possible to construct a surrounding constituent $(s_1, s_{10})_{REL}$. At first this seems like a restriction, but note that this only applies to constituents of the same type and that in practice no two constituents of the same type start at the same word.

To further improve results we apply a simple heuristic that allows closing each constituent at the end of a sentence or before a separator constituent. In essence this means inserting end suggestions for all constituent types into S before construction of the graph.

After construction the graph $G(V, E)$ consists of weighted vertices representing all possible constituents. In an independent set $V' \subseteq V$ no two vertices are adjacent and thus the corresponding constituents do not overlap or have the same start suggestion. The maximum weight independent set is the independent set $V' \subseteq V$ where $\omega(V')$ is maximum among all independent set and corresponds to the final constituents identified.

5.4.3 Solving the Maximum Weight Independent Set Problem

The maximum weight independent set problem is NP-complete, so currently no efficient algorithm is known for solving it. We therefore use a naive algorithm enumerating all the possible solutions and for large problem sizes fall back to a greedy algorithm. We start by explaining the naive algorithm for enumeration followed by the greedy algorithm. To provide simple pseudo-code we first define a set of operations. Given a graph $G(V, E)$, $N(v)$ defines the neighborhood of vertex v, i.e. all adjacent nodes. Furthermore, the standard set operations work on graphs, i.e. $G \backslash N(v)$ removes all neighbors of v from G and $G \cup v$ inserts v into G without adding any edges.

Enumeration. Algorithm 5.1 shows how to enumerate the problem recursively and is a slightly modified version of an approach in [R et al. (1986)] for finding the size of the maximum independent set.

At each call we split the graph into two sub-graphs at vertex v. Either the final MWIS contains v or it does not. The set that contains v cannot contain any neighbors of v so we remove them and recursively compute the MWIS. We also remove v

and later add it again, to avoid it from being randomly chosen in the sub-calls. We also recursively compute the MWIS set for the graph that does not contain v. The resulting MWIS is then the set returned by the sub-calls with larger ω. In [R et al. (1986)] it has already been shown that runtime is exponential $O(2^n)$ in the number of vertices of G.

Algorithm 5.1 Recursively Enumerate Maximum Weight Independent Set Problem

```
 1: maxInSet(G):
 2: if G has no edges then
 3:     return G
 4: else
 5:     $v \leftarrow$ vertex from G with maximum weight
 6:     $withoutV \leftarrow maxInSet(G \setminus v)$
 7:     $withV \leftarrow maxInSet(G \setminus N(v) \setminus v)$
 8:     $withV \leftarrow withV \cup v$
 9:     if $\omega(withV) > \omega(withoutV)$ then
10:         return $withV$
11:     else
12:         return $withoutV$
13:     end if
14: end if
```

Greedy. We fall back to the greedy approach if the sentence is longer than 30 words. The greedy approach does not always obtain an optimal solution but is guaranteed to run in $O(n)$. Algorithm 5.2 shows the pseudo-code. It greedily choses the vertex in G with largest weight, removes adjacent vertices and continues like that until G is empty.

Algorithm 5.2 Greedy Approach for Maximum Weight Independent Set Problem

```
 1: greedyMaxInSet(G):
 2: begin
 3: $I \leftarrow \emptyset$
 4: while $G \neq \emptyset$ do
 5:     $v \leftarrow$ vertex with maximum weight from G
 6:     $G \leftarrow G \setminus N(v) \setminus v$
 7:     $I \leftarrow I \cup v$
 8: end while
 9: return $I$
10: end
```

5.5 Implementation Details

We use *libSVM* [15] as a Support Vector Machine implementation. It is written in C++, so it integrates well with our existing code we used for implementing for feature extraction and remaining parts of contextual sentence decomposition. For training we used provided utilities of *libSVM* that perform cross-validation and parameter selection for kernel values.

We provide some information on how we tuned learning parameters. The constituent start and end classifiers are supposed to deliver suggestions. Because we can recover from wrong suggestions in the inference phase it is desirable to train the start and end classifiers in such a way that they rather make wrong suggestions but don't miss a correct one, instead of the other way round. We can achieve this by adjusting the cost of misclassifying a certain class during the learning phase. We simply set the cost of misclassifying a positive class, e.g. "is a relative clause start", to a much larger value than mis-classifying the negative class, i.e. "is not a relative clause start". As a result classifying as a negative class, when it fact it should be positive, is much worse than the other way round - exactly what we want to achieve. Table 5.9 shows the settings and kernel parameters for all classifiers used for the best results achieved.

Classifier	γ	C	cost +1	cost -1
SEP	0.03125	32	3	1
REL(	0.0078125	128	10	0.1
REL)	0.0078125	32	10	0.1
RELA(	0.03125	2	10	0.1
RELA)	0.0078125	128	10	0.1
LIT(	0.03125	8	10	0.1
LIT)	0.03125	32	10	0.1
REL	0.0078125	8	3	1
LIT	0.03125	8	3	1
RELA	0.0001221	512	3	1

Table 5.9: Training parameters γ of the RBF-Kernel and misclassification cost C for all classifiers used. "REL(" denotes the relative clause constituent start classifier, "REL)" the according end classifier and "REL" the complete constituent classifier. Analogously for all other constituent types. "*Cost +1*" denotes the cost multiplier for the positive class and "*Cost -1*" for the negative class.

[15] http://www.csie.ntu.edu.tw/~cjlin/libsvm/

6 Semantic Wikipedia Full-Text Search

In the previous chapters we provided the necessary pieces for contextual sentence decomposition (CSD) and the introduction already presented our understanding of semantic full-text search. This chapter now describes how we can integrate contextual sentence decomposition with an existing search engine to provide semantic full-text search on the English Wikipedia. Having a fully functional search engine at hand allows us to draw more reliable conclusions on the benefits and quality of contextual sentence decomposition and consequently we also use it in the following chapter for evaluation purposes.

The first section provides an overview of the components involved. Afterwards we describe how we integrate contextual sentence decomposition with a search engine so that the executed queries can benefit from it. To demonstrate the effect of contextual sentence decomposition in this set-up the last section provides some selected example queries and results.

6.1 Overview

A semantic full-text search engine should allow us to search for example for plants with edible leaves and as a result return a list of documents[16] that mention plants that have edible leaves. More precisely, if we formulate the query as ***plant*** *edible leaves* [17] we expect it to return a list of sentences where an instance of the class plant, such as "*Rhubarb*" (and not the word "*plant*") is mentioned along the words "*edible*" and "*leaves*".

We use SUSI (Wikipedia Search Using Semantic Index Annotations) [Buchhold (2010)] with our approaches of contextual sentence decomposition to allow semantic full-text search on the English Wikipedia. SUSI is based on ESTER (Efficient Search on Text, Entities and Relations) [Bast et al. (2007)]. Both utilize structured

[16]In the context of search engines a *document* represents some unit of information. Here, a document could e.g. refer to a complete article in Wikipedia or just to one sentence of an article.

[17]For simplicity for now we assume that **`plant`** in bold letters in this search query represents an entity class whereas `edible` and `leaves` are regular words.

knowledge from the ontology YAGO, but SUSI extends ESTER by providing an improved entity recognition and experimental anaphora resolution, which is important for CSD. Furthermore SUSI applies a novel index annotation allowing faster search queries with only minimal index blow-up.

SUSI already allows simple semantic full-text search, but it performs no natural language processing. The returned results are merely the result of full-text search in an index annotated with semantic information. Matches of a query must simply occur within the same document, in this case a sentence. By applying contextual sentence decomposition on the sentences we now require the matches to occur within the same sub-sentence. This provides a better semantic full-text search, that not only is semantic because of the information incorporated via an ontology, entity recognition and anaphora resolution, but also because it adheres to the semantics of the sentence part of the natural language.

Queries for SUSI are composed of words, entities and classes. Throughout this and the evaluation chapter, a query is composed of regular words, such as "*edible*", or special words starting with "*:e*". The special words are used to represent the structured information of YAGO. For example, the special word *:e:entity:physicalentity :object* resembles that an object is a physical entity which is an entity etc. Due to the length of the special words, we only provide the start and the characterstic end. SUSI performs prefix search and therefore the query *edi** matches the words "*editor*" and "*edible*" and the query *:e:entity:physicalentity:object...:person:** matches all instances of persons, for example "*Albert Einstein*".

We have provided an implementaion in C++ of the respective approaches of sentence constituent identification and sentence constituent recombination to allow contextual sentence decomposition and the next section describes how we can integrate it with SUSI.

6.2 Integrating with the Existing Index

To integrate contextual sentence decomposition with SUSI we need to find a suitable integration point. Of course, we want to perform the decomposition as a preprocessing step, after all the result is static, as is the index of the search engine. Therefore, ideally we want to integrate with the existing index generation, without having to change any of the structures. The underlying index of SUSI is constructed in a pipeline of index transformations, so an obvious approach is to extend the pipeline at the correct place with yet another transformation. This is actually a viable approach and the process described in this section.

A number of index files play a role for the final search engine but we concentrate on the one relevant to our task and begin by describing it. Table 6.1 shows how the sentence *"Albert Einstein took an entry examination, which he failed."* is representend in the index file. At this stage the index data is still in ASCII format and we can use it to apply contextual sentence decomposition.

word	document	score	position
:e:entity[...]physicist:alberteinstein:Albert_Einstein	19373	1	0
Albert	19373	1	0
Einstein	19373	1	0
___eofEntity	19373	1	0
took	19373	1	0
an	19373	1	0
entry	19373	1	0
examination	19373	1	0
,	19373	1	0
which	19373	1	0
:e:entity[...]physicist:alberteinstein:Albert_Einstein	19373	1	0
he	19373	1	0
___eofEntity	19373	1	0
failed	19373	1	0

Table 6.1: Example Sentence in Index Format with Recognized Entities.

The first column shows the word, the second one a document identifier, the third one a score value and the fourth one a position identifier. The words represent the actual content of the English Wikipedia and contain some *special words* that are the result of entity recognition and anaphora resolution provided by SUSI. For example the special word "*:e:entity[...]scientist:physicist:alberteinstein:Albert_Einstein*"[18] marks the entity Albert Einstein and provides the information that he is a physicist which is a scientist which is a person and so on. The special word "*___eofEntity*" is used to express across which words the last mentioned entity spans. The document identifier in our case uniquely identifies a sentence and the score value is irrelevant for us and we ignore it. The position identifier is used to assign each word some position within a document and is currently unused.

The interesting detail that allows using the same structure for sub-sentences is that it is possible to assign the same position to several words and to restrict a query using a special operator to match not only in the same document but also at the same position. This is exactly our use case. If the words of each sub-sentence of a sentence share the same position and same document id, a set of words must occur within the same sub-sentence to generate a match. Thus it is possible to restrict the search to sub-sentences.

To perform contextual sentence decomposition we then proceed as follows.

1. Perform part-of-speech tagging on the index.

[18]The complete special word has been shortened for reasons of brevity.

2. Perform text chunking on the index.
3. Perform contextual sentence decomposition outputting a new index file.
4. Continue the index generation with the new index file.

Both, part-of-speech as well as text chunking tags are required for the sentence constituent identification phases of contextual sentence decomposition. We therefore apply part-of-speech TreeTagger [Schmid (1994)], and afterwards the text chunker YamCha [Kudoh and Matsumoto (2000)] on the index file by transform the format to the required input formats of the respective tools. The final input for contextual sentence decomposition is then the original index enriched with columns containing the word's part-of-speech and text chunking tag. The implementation of sub-sentenceSentence Decomposition remembers document id, score and entity definitions for each word and assigns each resulting sub-sentence a new position denoted in the word's position column.

Table 6.2 depicts the resulting sub sentences "*Albert Einstein took an entry examination*" and "*an entry examination, which he failed.*" of the sentence above in the desired index format. This again nicely shows why the anaphora resolution is required beforehand. The second sub-sentences still contains the information that "*he*" refers to Albert Einstein, something that would be lost instead. The index generation pipeline can then be continued as usual.

word	document	score	position
:e:entity[...]physicist:alberteinstein:Albert_Einstein	19373	1	**0**
Albert	19373	1	**0**
Einstein	19373	1	**0**
___eofEntity	19373	1	**0**
took	19373	1	**0**
an	19373	1	**0**
entry	19373	1	**0**
examination	19373	1	**0**
an	19373	1	**1**
entry	19373	1	**1**
examination	19373	1	**1**
,	19373	1	**1**
which	19373	1	**1**
:e:entity[...]physicist:alberteinstein:Albert_Einstein	19373	1	**1**
he	19373	1	**1**
___eofEntity	19373	1	**1**
failed	19373	1	**1**

Table 6.2: Decomposed Example Sentence in Index Format.

6.3 Example Queries and Results

We next show some sample results of queries executed against SUSI using contextual sentence decomposition. For each query we show an actual result returned, and a result not returned because of CSD, which would otherwise be returned. This provides some intuition of the final result and overall use of semantic full-text search and especially contextual sentence decomposition.

In the queries shown the "=" operator is used instead of a space to require the matches to occur in the same sub-sentence. The prefixes are followed by the wildcard operator "*". Matches in the results are shown in bold, and additionally the respective entity we are looking for with our query is underlined.

Query for all *friends of Albert Einstein:*

friend=:ee:entity:alberteinstein:*=*
*:e:entity:[...]:livingthing:organism:person:**

Returned *Leó Szilárd*:

"*During August 1939* ***he*** *[Leó Szilárd] approached his old* ***friend*** *and collaborator* ***Albert Einstein*** *[...]*"

Did not return *Yoshio Nishina*:

"Yoshio Nishina*, a* ***friend*** *of Niels Bohr and a close associate of* ***Albert Einstein."***

Query for *athletes disqualified due to doping:*

disqualif=doping=:e:entity:[...]:livingthing*
*:organism:person:contestant:athlete:**

Returned *Lyidmila Blonska*:

"Lyudmila Blonska *was later* ***disqualified*** *for failing a* ***doping*** *test [...]*"

Did not return *Thomas Alsgaard*:

"*However Mühlegg was found guilty of* ***doping*** *and* ***disqualified*** *by the IOC in February 2004, therefore upgrading Estil and* ***Alsgaard*** *to joint gold medalists.*"

Query for *plants with edible leaves:*

*edible=leaf|leaves=:e:entity:[...]:livingthing:organism:plant**

Returned *Parsley*:

"*Carrot, celery and **parsley** are true biennials that are usually grown as annual crops for their **edible** roots, petioles and **leaves**, respectively.*"

Did not return *Pandanus*:

***"Pandanus**: from pandan, a tropical tree or shrub with a twisted stem, long spiny **leaves**, and fibrous **edible** fruit.*"

Query for *politicians that died because of diabetes:*

die|death=:ee:entity:diabetes:*=*
*:e:entity:[...]:person:leader:politician:**

Returned *Bernard Dowiyogo*:

"***He [Bernard Dowiyogo] died** in office in March 2003 (having been president on this occasion since January 2003) at George Washington University Hospital in Washington, D.C. from heart complications brought on by his struggle with **diabetes**, a common ailment on Nauru.*"

Did not return *Joan Littlewood:*

"*In 1975, her collaborator and partner, Gerry Raffles **died** of **diabetes**, and in 1979, a devastated **Joan Littlewood** moved to France, and ceased to direct.*"

7 Evaluation

Evaluation shall provide the answer to two questions: on the one hand how well does the approach of contextual sentence decomposition (CSD) work for semantic full-text search, and on the other hand how do the two approaches of sentence constituent identification (SCI) compare to each other. Evaluation is therefore performed on three different levels. First we directly compare the two approaches of sentence constituent identification in terms of how well they recognize constituents. We then compare their influence by evaluating the resulting contextual sentence decomposition. In order to asses the overall effect of CSD on semantic full-text search we examine the search quality of a semantic search engine incorporating it. For each evaluation we closely inspect results and give an interpretation.

7.1 Measures

We shortly describe the measures relevant throughout the evaluations. For each experiment performed we determine expected results and compare them against actual results. We can categorize each element in the set of expected or actual results into one of three categories:

- **True:** for an element present in the actual as well as the expected results
- **false-positive** (False-Pos)**:** for an element present in the actual but not in the expected results
- **false-negative** (False-Neg)**:** for an element present in the expected but not in the actual results

We then compute the *precision*, *recall* and *F-measure ($F_{\beta=1}$)* which are typical measures in an Information Retrieval context. Let TRUE, FALSE-POS and FALSE-NEG be sets containing all elements of the respective categories. The measures can be computed as follows:

$$precision = \frac{|TRUE|}{|TRUE| + |FALSE\text{-}POS|} \quad recall = \frac{|TRUE|}{|TRUE| + |FALSE\text{-}NEG|}$$

$$F_{\beta=1} = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

7.2 Sentence Constituent Identification

To evaluate the quality of our sentence constituent identification approaches we use a test set of 50 sentences selected from the English Wikipedia. The sentences mainly consist of false-positives observed during semantic search queries, but also contain sentences selected due to an interesting structure of constituents. The test set is therefore a particularly hard set of sentences compared to the average sentences of the English Wikipedia. For each sentence a *gold* annotation in form of the type, start and end of each contained constituent has manually been provided. We measure how well starts as well as ends of each constituent type are determined compared to the golden assignments. We also measure how well complete constituents (consisting of correct start and end pair) are determined.

7.2.1 Measurements

In the tables below the start of a relative clause constituent is denoted by "*REL*(" and an end by "*REL*)" and analogously for the remaining constituents. A separator constituent consists of only one word, therefore no distinction between start and end is necessary.

Type	SCI	True	False-Neg	False-Pos	Precision	Recall	F-measure
REL	RULE-SCI	16	7	2	88.9%	69.6%	78%
	ML-SCI	13	10	4	76.5%	56.5%	65%
RELA	RULE-SCI	2	3	7	22.2%	40%	28.6%
	ML-SCI	3	2	13	18.8%	60%	28.6%
LIT	RULE-SCI	41	36	24	63.1%	53.2%	57.7%
	ML-SCI	24	53	24	50%	31.2%	38.4%
SEP	RULE-SCI	23	2	14	62.2%	92.5%	74.2%
	ML-SCI	15	10	6	71.4%	60%	65.2%
TOTAL	RULE-SCI	82	48	47	63,6%	63,1%	63,3%
	ML-SCI	55	75	47	53,9%	42,3%	47,4%

Table 7.1: Evaluation of sentence constituent identification. Results for the rule based SCI (RULE-SCI) and machine learning based SCI (ML-SCI) are shown. Matched constituents must have same start and end.

Type	SCI	True	False-Neg	False-Pos	Precision	Recall	F-measure
REL(	RULE-SCI	18	5	0	100%	78.3%	87.8%
	ML-SCI	16	7	1	94.1%	69.6%	80%
REL)	RULE-SCI	16	7	2	88.9%	69.6%	78.0%
	ML-SCI	13	10	4	76.5%	56.2%	64.8%
RELA(	RULE-SCI	3	2	6	33.3%	60%	42.9%
	ML-SCI	5	0	11	31.3%	100%	47.7%
RELA)	RULE-SCI	2	3	7	22.2%	40%	28.6%
	ML-SCI	3	2	13	18.8%	60%	28.6%
LIT(	RULE-SCI	48	29	17	73.8%	62.3%	67.6%
	ML-SCI	32	45	16	66.7%	41.6%	51.2%
LIT)	RULE-SCI	50	27	15	76.9%	64.9%	70.4%
	ML-SCI	31	46	17	64.6%	40.3%	49.6%
SEP	RULE-SCI	23	2	14	62.2%	92.5%	74.2%
	ML-SCI	15	10	6	71.4%	60%	65.2%
TOTAL	RULE-SCI	160	75	61	72,4%	68,1%	70,2%
	ML-SCI	115	120	45	71,9%	48,9%	58,2%

Table 7.2: Results for the evaluations of identified starts and ends of constituents. The results for the rule based SCI (RULE-SCI) and machine learning based SCI (ML-SCI) are shown. For ML-SCI this represents the final result after inference and not an intermediate classification.

7.2.2 Interpretation

First we consider each of the approaches by itself. For the rule based approach of sentence constituent identification we can see in table 7.1 that the best results are achieved for identifying relative clause constituents followed by separator constituents. Separator and relative clause constituents don't share many characteristics with other constituents and are easily distinguishable and recognized fairly well. Compared to them the identification of list item constituents and appositive constituents is worse. This can be attributed to the fact that list item constituents and appositive constituents are similar in their grammatical environment. Both usually only consist of noun phrases and the rule based approach sometimes assigns the wrong type. For example consider the following passage showing correct constituents:

“[...] this position,($_{REL}$ which had passed ($_{LIT}$ from Jacques-Champion Chambonnières)$_{LIT}$ ($_{LIT}$ to Jean-Henri D'Anglebert)$_{LIT}$($_{LIT}$ to François Couperin)$_{LIT}$($_{LIT}$ to his daughter , ($_{RELA}$Marguerite-Antoinette Couperin)$_{RELA}$)$_{LIT}$, and then ($_{LIT}$to Bernard de Bury)$_{LIT}$)$_{REL}$”

The appositive *"Marguerite-Antoinette Couperin"* is identified as another list item constituent by the rule based approach. Unfortunately assigning the wrong type in this case poses a problem because the recombination phase treats the types in a different way[19]. Another issue influencing list item recognition is that the rule based approach uses the assumption that list items are separated by commas - something that for example does not hold in the sentence above.

Looking closer at the starts and ends of each constituent type in table 7.2 we can see that the starts of each constituent type are usually better recognized than their ends. However, the absolute numbers of correct matches are relatively close and remembering that the rule based approach first discovers the starts and then assigns ends to the constituents we can conclude that the corresponding ends are often correctly assigned and thus the heuristic is effective. What stands out is the precision of 100 percent for discovering relative clause starts. However, we miss some starts which results in a recall of only 78.3 percent showing that the rule applied is effective but might be further extended to matching the missed starts. For the comparably bad results of appositive constituent starts one has to realize that the test set only provided a small number of examples. A test set containing more appositive constituents might give some more insight here.

For the list item constituents we observe that a lot of false starts are identified. This is caused by missing parts of a first list item constituent. For example, in the sentence with correctly identified constituents:

"Taro is a tropical plant grown $(_{LIT}$ *primarily as a root vegetable for its edible corm* $)_{LIT}$*, and* $(_{LIT}$ *secondarily as a leaf vegetable* $)_{LIT}$*"*

the first list item constituent actually identified is "*its edible corm".* Improving the heuristic for identifying the first list item constituents may provide a solution here.

The identification of separator constituents shows a good recall of 88.5 percent but also a comparably large number of false-positives indicating that the applied rule might be too optimistic. Remaining errors are often caused by not perfectly recognizing embedded constituents as in:

"Yoshio Nishina , $(_{REL}(_{LIT}$ *a friend of Niels Bohr* $)_{LIT}$ *and* $(_{LIT}$ *a close associate of Albert Einstein*$)_{LIT})_{REL}$*"*

for which the constituents actually identified are:

"Yoshio Nishina , $(_{LIT}$ *a friend of Niels Bohr* $)_{LIT}$ *and* $(_{LIT}$ *a close associate of Albert Einstein*$)_{LIT}$*"*

[19]As opposed to for example assigning the type REL instead of RELA.

It is interesting to note that in this case the resulting sub-sentences after the recombination phase would be identical. This is not necessarily always the case. The next evaluations, measuring results after recombination, will show how large the actual influence of wrong identifications are.

Let us next consider the results for the machine learning based approach of SCI. As table 7.1 shows basically the same observations hold as for the rule based approach: relative clause and separator constituents seem to be easier to discover than appositive and list item constituents. However, the machine learning approach causes a larger number of false-positives and negatives reducing the overall F-measures. Furthermore, the measures for the recognized list item constituents are worse than for the rule based approach. To understand the results we need to remember that the machine learning based approach is based on a filtering phase and an inference phase. The presented results are the final results after the inference phase. In the filtering phase starts and ends of constituents are suggested and the inference phase builds a constituent structure out of these suggestions. Because the inference phase cannot recover from missing suggestions, false-negatives might be caused by missing suggestions. False-positives however should be filtered out. We therefore need to consider each of the phases in more detail.

Looking at table 7.2 we can see that starts of the final constituents are better identified than their ends. This can be caused by either the filtering phase not suggesting the correct ends or by the inference phase selecting the wrong ends. Therefore, table 7.3 shows how well the classifiers of the filtering phase identified constituent starts and ends in the test set, which resembles the input to the inference algorithm.

Type	True	False-Neg	False-Pos	Precision	Recall	F-measure	Accuracy
REL(	18	5	1	94.7%	78.3%	85.7%	99.5%
REL)	16	7	44	26.7%	69.6%	38.6%	95.5%
RELA(	5	0	16	23.8%	100%	38.4%	98.7%
RELA)	3	2	5	37.5%	60%	46.2%	99.4%
LIT(	39	36	22	63.9%	52%	57.3%	95.1%
LIT)	53	22	5	91.4%	70.7%	79.7%	97.8%

Table 7.3: Filtering phase classifier performance on the test set. For each constituent type we show the number of constituent starts and ends correctly identified, missed and erroneously identified. Accuracy in the last column is based on all 1189 instances of words classified.

The classifiers were trained to produce a good recall for the sake of a good precision. Looking at table 7.3 we can indeed observe that , with respect to recall, the classifiers of constituent ends are worse than those of the corresponding start classifiers explaining our observation. An exception here is the list item constituent start classifier which performs even worse than its end classifier also explaining why the overall result for this constituent type is influenced. This may be caused by the fact that some list item constituents in the training set start with a preposition or a verb and confuse the classifier. A larger training set should help improving the end classifiers as well as the list item start classifier. Another approach might also be the utilization of a heuristic, similar to the one of the rule based approach, to deduct constituent ends for suggested starts[20].

To draw conclusions on how well the inference phase works we remember that three additional classifiers are used to decide whether a possible constituent start and end pair actually resemble a constituent of the given type. Depending on the outcome the possible suggestion is assigned a weight in the mapped graph. Table 7.4 shows the performance of these classifiers on the test set. Especially the large number of false-positives poses a problem here: a larger weight will be assigned to those candidates causing the final resulting false-positives we observed. The comparably bad performance of these classifiers directly influences the robustness of the inference phase. Again, especially a larger training set and more parameter tuning and feature selection should be the solution here.

Type	True	False-Pos	False-Neg	Precision	Recall	F-measure	Accuracy
REL	14	19	9	42.4%	60.9%	50%	87.4%
RELA	2	5	3	28.6%	40%	33.3%	96.4%
LIT	57	18	20	76%	74%	75%	82.9%

Table 7.4: Inference phase classifier performance on the test set. Accuracy in the last column is based on all 222 instances of constituents classified.

Comparing the two approaches first of all we can see that the rule based approach of sentence constituent identification outperforms the machine learning based approach in all measures taken above. Especially the machine learning based approach produces both, more false-positives and negatives compared to the rule based approach. As we saw this is mainly due to our used classifiers still not being robust and accurate enough. The filtering phase fails to predict some ends of constituents causing the false-negatives and the inference phase is not robust enough to filter out some wrong suggestions causing the false-positives. A larger training set will definitely increase the performance of the classifiers and further parameter and feature tuning should help as well. As we saw the rule based approach could be improved by developing new and improving existing heuristics. However, care must be taken not to influence existing heuristics. Furthermore the rule based approach is limited by its ability to discover more complex embedded constituents - something that is not easily solved.

[20]A simple heuristic, suggesting ends for each open constituent at the end of a sentence, is already present, but obviously not effective enough.

As a final example consider the passage with correct annotations:

"Yoshio Nishina , $(_{REL}(_{LIT}$a friend of Niels Bohr $)_{LIT}$ and $(_{LIT}$ a close associate of Albert Einstein $)_{LIT})_{REL}$"

and how the machine learning based approach identified the constituents

"Yoshio Nishina , $(_{RELA}$a friend $(_{LIT}$of Niels Bohr $)_{LIT}$ and $(_{LIT}$ a close associate of Albert Einstein $)_{LIT})_{RELA}$"

resulting in one correctly identified constituent and two false-positive constituents. However, this result is not far from the desired one, something that is not reflected in this evaluation. Identifying such an embedded structure is a hard task and it demonstrates the potential of the machine learning based approach. Therefore, one must bear in mind that the above evaluation is strict and does not consider the final outcome of contextual sentence decomposition - and after all, this is what we are interested in. Still, the results for the rule based approach are superior. The next section provides an evaluation of the final outcome of contextual sentence decomposition.

7.3 Contextual Sentence Decomposition

For each of the sentence constituent identification approaches we evaluate the resulting contextual sentence decomposition. The same set of sentences as in the previous evaluation is used and for each sentence the set of expected sub-sentences was manually provided. For the evaluation we first define sub-sentence equality as set equality filtered on a set of relevant word categories. Let c be a sub-sentence consisting of a multi-set of words $\{w_1, w_2...w_n\}$. To denote the part-of-speech tag of a word w we use $pos(w)$ and we define the set P to contain relevant word categories. The set of relevant words D corresponding to sub-sentence C is simply the set of words whose part-of-speech is in P:

$$D = \{w_i | w_i \in C \wedge pos(w_i) \in P\}$$

and sub-sentence equality is easily defined as equality of the corresponding sets

$$C_1 = C_2 \Longleftrightarrow D_1 = D_2$$

Using the set P of relevant word categories allows us to ignore certain type of words like "*and*","*or*" and "*but*" which are not relevant for the use-case we consider. We also use the Jaccard distance to measure the similarity of two sub-sentences. The Jaccard distance d of two sub-sentences C_1,C_2 is based on their corresponding sets of relevant words D_1 and D_2 , and defined as follows:

$$d = 1 - \frac{|D_1 \cap D_2|}{|D_1 \cup D_2|}$$

A Jaccard distance of 0 results from two identical sub-sentences and a distance of 1 from two completely different sets.

Evaluation is then performed for each sentence by comparing resulting actual with expected sub-sentences and categorizing them into true, false-positive and false-negative using above defined sub-sentence equality. Additionally, for each false-positive sub-sentence we compute the minimum Jaccard distance to the sub-sentences marked false-negative and vice versa. This allows a measure of how "wrong" the resulting sub-sentences actually are.

7.3.1 Measurements

In the following evaluation the set of part-of-speech tags regarded relevant are:

$$\begin{aligned} P = \{&PP\$, NNP, NNS, NP, NN, VBD, VBZ, \\ &VBP, JJ, VB, RB, PP, PRP, VBN, CD\} \end{aligned}$$

For a description of these tags we refer to [Marcus et al. (1993)], but mainly they include all nouns, personal pronouns, verbs, adjectives, adverbs and numbers.

	True	False-Pos	False-Neg	Precision	Recall	F-measure
ML-CD	56	63	93	47%	37.6%	41.8%
RULE-CD	98	57	51	63.4%	65.8%	64.5%

Table 7.5: Results for the evaluation of contextual sentence decomposition using the Machine Learning (ML-CD) and Rule Based (RULE-CD) sentence constituent identification.

	Avg.Distance False-Pos	Avg.Distance False-Neg
ML-CD	0.369	0.456
RULE-CD	0.463	0.443

Table 7.6: Average Jaccard distances for the results of contextual sentence decomposition using the Machine Learning (ML-CD) and Rule Based (RULE-CD) Sentence Constituent Identification.

7.3.2 Interpretation

As we can see in table 7.5 the rule based approach of contextual sentence decomposition (RULE-CD) clearly outperforms the machine learning based approach (ML-CD) regarding quality of resulting sub-sentences. This is not unexpected considering the quality of identified constituents evaluated previously.

The machine learning based approach produces a large number of false-negatives and it also produces a smaller total amount of sub-sentences than the rule based approach, accounting for much of the difference. A close look reveals that this is often due to single list item constituents being identified, that are missing the remaining list items of the enumeration. For example, in

"Bernard de Bury ($_{LIT}$ resided in Versailles his entire life)$_{LIT}$, and ($_{LIT}$ held various positions at the court)$_{LIT}$"

only the last list item was identified which, by itself, does not result in a new sub-sentence. Furthermore some slight errors in constituent identification lead to slightly wrong sub-sentences, resulting in false-positive as well as false-negative sub-sentences. Consistent with the previous evaluation, some constituents are missed completely further attributing to the smaller amount of resulting sub-sentences. The large number of false-positive sub-sentences can be attributed to wrongly identified constituents.

The rule based approach performs as expected: the average F-measure of identified constituents in the previous evaluation is 59.6 percent and for the resulting sub-sentences it is 62 percent. A large number of errors are caused by not recognizing long lists or complex embedded constituents. For example, the embedded structure in

"Harrison ($_{LIT}$ asserts the existence of female trinities)$_{LIT}$, ($_{LIT}$ discusses the Horae as chronological symbols representing the phases of the Moon)$_{LIT}$ and ($_{LIT}$ goes on to equate the Horae with ($_{LIT}$ the Seasons)$_{LIT}$, ($_{LIT}$ the Graces)$_{LIT}$ and ($_{LIT}$ the Fates)$_{LIT}$)$_{LIT}$"

is not recognized but the enumeration is continued on the same level, resulting in false-positive as well as false-negative sub-sentences. For smaller and simpler sentences the rule based approach performs very well however.

Looking at table 7.6 we can see that the average distance of false-positives and false-negatives is smaller for the machine learning based approach. One can conclude that on average each of the false-positive sub-sentences generated by the machine learning approach was contained to about 63 percent (100 - 36.9 percent) in one of the false-negative sub-sentences (of the same sentence of course). In other words, although it made more errors, the average "significance" of an error is smaller. This also matches the observation that the machine learning based approach makes a lot of smaller errors in constituent identification.

Overall we can say that RULE-CD is the more aggressive type of decomposition. More sub-sentences are generated, and it produces less errors, but these are worse than the errors produced by ML-CD. ML-CD on the other hand identifies less constituents resulting in less sub-sentences and the erroneous sub-sentences are on average closer to the desired ones.

For both approaches it holds that on average more than 50 percent of a missed sub-sentence are contained in one of the false-positives generated. Therefore, the final evaluation of search quality will have to show how large the influence of slightly wrong sub-sentences actually is.

Something we have not considered so far is the fact that when we measure the quality of contextual sentence decomposition we also measure the quality of sentence constituent recombination. SCR uses a heuristic for attaching relative clauses to the noun they describe, which is not always correct. For example, in

"[...] various addresses in and around Kennington Road in Lambeth , ($_{REL}$ including ($_{LIT}$ 3 Pownall Terrace)$_{LIT}$, ($_{LIT}$ Chester Street)$_{LIT}$ and ($_{LIT}$ 39 Methley Street)$_{LIT}$)$_{REL}$."

the relative clause refers to "*various addresses in and around Kennington Road in Lambeth*" but we actually attach it to *"Lambeth"* resulting in wrong sub-sentences. A close inspection shows that this only caused a relatively low number of false-positive sub-sentences for the approaches, therefore we do not further consider it.

7.4 Search Quality

The previous evaluations examined how the approaches of sentence constituent identification compare to each other in terms of resulting decomposition. We now evaluate how contextual sentence decomposition influences search quality for semantic full-text searches.

A search engine incorporating contextual sentence decomposition as described in chapter 6 is used to execute a selected set of queries, allowing us to evaluate search quality by examining the returned results against a ground truth. The baseline is provided by executing the queries against a regular index[21] of the English Wikipedia. We then compare it against the queries on the index for which CSD has been performed using each of the SCI approaches.

In detail evaluation is performed as follows. Each selected query returns a list of entities together with a witness, an excerpt of the full-text where the match occurred[22]. For example, the query

edible leaf/leaves :e:entity:[...]livingthing:organism:plant"*

matches all sentences where the words "*edible*", "*leaf*" or "*leaves*", and an instance of a plant occurs. Each returned result then consists of an entity representing the plant, together with the sentence the match occurred in. Consequently it is also possible that an entity is returned several times - once for each hit in the corpus. We then compare the set of returned entities with our ground truth and evaluate the results using our standard measures. Note that the witnesses are not used in this evaluation, but will allow a more detailed reasoning for some cases as we will see.

We use two different sets of queries shown in table 7.7. The first one (A1-A5) is evaluated against an automatically generated ground truth using Wikipedia lists, identical to the evaluation already performed for SUSI [Buchhold (2010)]. A Wikipedia list contains facts of a certain nature and has been manually populated or automatically extracted, for example a list of edible plants, which can then be parsed for entity names. Although a good source of information it has been shown to be incomplete or to contain wrong facts [Buchhold (2010)], making the evaluation less accurate. We therefore use a second set of queries (M1-M5) for which the ground truth has been manually generated by executing each query against the search engine running our baseline index and examining returned results. Consequently these queries can only generate correct or false-positive results on the baseline.

While being a very important evaluation we note that several influences play a role in its results, possibly falsifying them to some extent. First of all contextual sentence decomposition assumes that entity recognition and anaphora resolution is performed beforehand. Therefore, the evaluation depends on the quality of entity recognition and anaphora resolution incorporated into SUSI. Especially an unresolved anaphora may lead to false elimination of some results. If for example in the following sentence

"Usable parts of Rhubarb include the edible stalks and the medicinally used roots, however its leaves are toxic."

it remains unresolved that "*its*" refers to "*Rhubarb*", the resulting sub-sentence "*its leaves are toxic*" looses very vital information.

[21]SUSI generates the index by parsing an XML dump of the English Wikipedia.

[22]The actual implementation is slightly different but for our concerns we can assume the aforementioned behavior.

Furthermore natural language processing can be influenced by the lexical quality of the index. It has been parsed from Wikipedia markup and, although error-less to large extents, may contain dubious characters and incorrect sentence boundaries. This hardly influences search results on the standard index, but it can inhibit natural language processing and thus contextual sentence decomposition to some extent. We need to remember that the contextual sentence decomposition approaches are based on a part-of-speech tagger and text chunkers. These assume correct english sentences as input and their performance can influence all further processing.

As a last point we note that the evaluation does not properly reflect the full effect of contextual sentence decomposition. This stems from the already mentioned fact that the result of a query can contain an entity-excerpt pair several times - once for each sentence or sub-sentence that matched. Assume a false-positive entity that is contained five times because it occurs in five different matching sentences, but each time with a different, wrong meaning. If contextual sentence decomposition can eliminate four of the five false-positives results, the entity will still be in the result set and the effect of Context Decomposition is not visible in the standard measures.

7.4.1 Measurements

The following table shows the queries used for search quality evaluation.

ID	Query
A1	*drug=died/death=:e:entity:[...]:person:**
A2	*united=states=elected=:e:entity:[...]:president:**
A3	*english=:e:entity:[...]:sovereign:**
A4	*political=:e:entity:p[...]:writer:**
A5	*computer=:e:entity:[...]:scientist:**
M1	*edible=leaf/leaves=:e:entity:[...]:plant**
M2	*friend*=:ee:entity:alberteinstein:*=:e:entity:[...]:person:**
M3	*blood=sugar/glucose/:e:entity:[...]:monosaccharide:* =:e:entity:[...]:hormone:**
M4	*die*/death=:ee:entity:diabetes:*=:e:entity:[...]:politician:**
M5	*disqualif*=doping=:e:entity:[...]:athlete:**

Table 7.7: Queries for search quality evaluation. The prefix M indicates evaluation using a manually generated ground truth, and analogously the prefix A evaluation against an automatically generated ground truth. For brevity the queries have been shortened. The full queries can be found in the appendix.

Query	Index	True	False-Neg	False-Pos	Precision	Recall	F-measure
A1	BASE	106	108	1124	8.62%	49.53%	14.68%
	ML-CD	92	122	848	9.79%	42.99%	15.94%
	RULE-CD	83	131	560	12.90%	38.79%	19.37%
A2	BASE	42	1	139	23.2%	97.67%	37.5%
	ML-CD	40	3	111	26.49%	93.02%	41.24%
	RULE-CD	39	4	69	36.11%	90.7%	51.66%
A3	BASE	48	11	1409	3.29%	81.36%	4.37%
	ML-CD	48	11	1262	3.66%	81.36%	7.01%
	RULE-CD	47	12	1103	4.09%	79.66%	7.78%
A4	BASE	29	18	13209	2.19%	61.7%	4.37%
	ML-CD	28	19	11755	2.38%	59.57%	4.73%
	RULE-CD	26	21	10519	2.47%	55.32%	4.91%
A5	BASE	294	63	2850	9.35%	82.35%	16.8%
	ML-CD	290	67	2570	10.14%	81.23%	18.03%
	RULE-CD	289	68	2403	10.74%	80.95%	18.96%
TOTAL A1-A5	BASE	519	201	18731	2,7%	72,1%	5,2%
	ML-CD	498	222	16546	2,9%	69,2%	5,6%
	RULE-CD	484	236	14654	3,2%	64,7%	6,1%
M1	BASE	41	0	58	41.41%	100%	58.57%
	ML-CD	33	8	42	44%	80.49%	56.9%
	RULE-CD	25	16	16	60.98%	60.98%	60.98%
M2	BASE	35	0	65	35%	100%	51.85%
	ML-CD	28	7	52	35%	80%	48.7%
	RULE-CD	21	14	15	58.33%	60%	59.15%
M3	BASE	21	0	19	52.5%	100%	68.85%
	ML-CD	18	3	18	50%	85.7%	63.16%
	RULE-CD	17	4	9	65.38%	80.95%	72.34%
M4	BASE	21	0	7	75%	100%	85.71%
	ML-CD	20	1	4	83.33%	95.23%	88.89%
	RULE-CD	19	2	1	95%	90.48%	92.68%
M5	BASE	42	0	24	63.63%	100%	77.78%
	ML-CD	37	5	15	71.15%	88.1%	78.72%
	RULE-CD	34	8	8	80.95%	80.95%	80.95%
TOTAL M1-M5	BASE	160	0	173	48%	100%	64,9%
	ML-CD	136	24	131	50,9%	85%	63,7%
	RULE-CD	116	44	49	70,3%	72,5%	71,4%

Table 7.8: Search query results for executing against the baseline index (BASE) and the indices pre-processed with contextual sentence decomposition using the rule based approach (RULE-CD) and the machine learning based approach (ML-CD) are shown.

7.4.2 Interpretation

First of all we note that applying contextual sentence decomposition can not increase the recall of a search query. Sub-sentences represent subsets of the words of an original sentence, therefore if a set of words and entities is not present in a sentence, it can not be present in its sub-sentences. Ideally we expect an increase in precision, by weeding out many false-positives, at the same time maintaining recall, resulting in an increase of the overall F-measure.

As we can see in table 7.8 for all queries performed precision has increased. Most notably the increase is always higher with the rule based approach of contextual sentence decomposition. This is consistent with our previous evaluations, showing that RULE-CD outperforms ML-CD. Because precision is low for the queries based on an automatically generated ground truths (A1-A5), comparing absolute values is not meaningful and we compute the averages of relative increases. On average the machine learning based approach caused a relative increase in precision of 12.5 percent, and 31.5 percent for the rule based approach. For the queries evaluated against a manually created ground truth (M1-M5) the relative increase was 4.7 percent and 38.8 percent respectively. The difference for the machine learning based approach is caused by the two queries M2 and M3. A problem here seems to be that the ground truth and result size is relatively small, for example, for M3 the incorrect elimination of three and correct elimination of one results, already causes a decrease of precision. Overall the increase in precision is not as large as we might expect based on the previous evaluations. We therefore closely inspect the errors, but first interpret the remaining results on recall and F-measure.

As already noted the recall of each query can not be increased using contextual sentence decomposition. Maintaining recall is therefore the primary goal. As shown in table 7.8 however, recall is affected in a negative way by contextual sentence decomposition. The decrease is slightly larger for the rule based contextual sentence decomposition because it seems to be more aggressive. This is consistent with our previous evaluations: ML-SCI causes a smaller number of constituents to be identified resulting in less decomposition and furthermore the average distance of false-positives, as shown in table 7.6 is smaller. Thus, if an error occurs it has less impact than with RULE-CD. Nonetheless the decrease is small for most queries, especially when considering the absolute number of true results for some queries - causing only a few entities not to be returned already has a large impact on recall.

Considering the F-measure that incorporates both precision and recall we still observe an increase for practically all queries. The decrease in recall is always counterbalanced by an even larger increase in precision. An exception are the queries M2 and M3 where ML-SCI incorrectly eliminates few true results already causing a too large decrease in recall. We can see some fluctuation throughout all queries. This can be caused by different levels and areas of natural language used within certain topics. For example, the query M3 for *hormones that play a role in blood sugar* matches a lot of sentences using medicinal terminology, that are typically long and intertwined. In contrast an inspection of the results of query M4 for *politicians whose death was caused by diabetes* reveals a relatively simple use of language, with short and easy to understand sentences. This is in accordance with the results of the respective queries.

Although the results are comprehensible so far we want to perform a deeper analysis of the errors. For CSD only two different error scenarios must be considered: it can cause a correct result to no longer be returned, causing a decrease in recall, and it can still return a false-positive result it should have eliminated, causing a decrease in precision. If none of these errors occur recall will remain unchanged and precision equals 100 percent.

We finish this evaluation with a close inspection of results for the queries M1-M5 for RULE-CD. For each of the returned faulty entity-witness pairs we categorize them into "incorrectly not deleted", for false-positives that have not been eliminated but should be, or "incorrectly deleted", for correct results that were eliminated. For each query we inspect 10 errors and analyze why the respective error was made by assigning one of the following categories:

- **R** for an erroneous relative clause attachment. For example, in the sentence *"A total of twelve weightlifters were disqualified for doping, amongst them Greek star Leonidas Sampanis, who had won two silver medals in [...]"* the relative clause "*amongst them Greek star Leonidas Sampanis ...*" refers to "*weightlifters*", but the current implementation attaches it to the closest noun which is "*doping*".
- **L** for a lexical error in the index caused by the Wikipedia parser. An example is a wrong sentence boundary as in *"the toxin found in many wood-sorrels and other edible plants A characteristic of members*" or remaining characters of Wikipedia markup.
- **E** for a problem with entity recognition and anaphora resolution. For example, in "*Hans Poelzig designed a summer house for Albert Einstein, one of his lifelong friends.*" the anaphora "*his*" was resolved to Albert Einstein, although it refers to Hans Poelzig.
- **D** for an error on behalf of contextual sentence decomposition. This is mostly due to erroneous constituent identification for example as in *"Austria's Johannes Eder* $(_{LIT}$*originally finished fourth in this event* $)_{LIT}$ *, but* $(_{LIT}$ *was disqualified on November 22* $)_{LIT}$ $(_{SEP},)_{SEP}$ *2007 after the FIS issued a two-year doping suspension [...]"* where the separator is wrongly identified, resulting in wrong decomposition and loss of factual information.
- **I** for an error that occurs, although contextual sentence decomposition is performed successfully. For example, in *"[...] all the resulting hybrids of this crossing have radish leaves and cabbage roots, the two non-edible parts of its ancestors*" we cannot distinguish the fact that we are talking about non-edible instead of edible parts. As an other example consider "*In the late 1960s,*

he and his Cambridge friend and colleague, Roger Penrose, applied a new, complex mathematical model they had created from Albert Einstein's general theory of relativity" where current decomposition puts "*he and his friend ...*" and *"applied a new complex mathematical model [...] created from Albert Einstein's [...]"* into the same sub-sentence, resulting in an erroneous relation of the words "*friend*" and "*Albert Einstein*" and the entity referenced by the anaphora "*he*".

The results are presented in the following table.

	R	L	E	D	I
Number of errors	3	5	7	23	12

Table 7.9: Category based error analysis for the queries M1-M5. An error is the incorrect elimination of a correct entity-witness pair from the result set, respectively a false-positive entity-witness pair that was not eliminated. Ten errors have been analyzed for each of the queries.

As we can see in table 7.9 half of the errors really stem from problems in contextual sentence decomposition (category D) and the other half can be attributed to the remaining categories. The errors for contextual sentence decomposition are caused by complex sentences and grammatical constructs RULE-CD cannot handle. For example, the comma used in dates as in *"November 22, 1999 ..."* confuses the algorithms, as do participle clauses, e.g. *"When blood sugar levels become too high, insulin is released from the pancreas, lowering the blood sugar levels".* Extending the rules might alleviate these problems. The errors are however also caused by some sentence constructions that are currently not properly reflected by our constituents. For example the participle construction in the sentence above is recognized as a relative clause, and therefore attached to a noun. However, it expresses and describes a circumstance or causal relation. The same is true for adverbial clauses as in *"Dembo was also wounded in the same attack that killed Savimbi and, because he was weakened by diabetes, died ten days later"* where "*because he was weakened by diabetes*" describes how Dembo died. A future direction therefore might be the extension of constituent types to express such constructs.

Second most errors are caused by category I, where contextual sentence decomposition seems not powerful enough. On the one hand this is attributed to negations as in "*non edible*" as opposed to "*edible*". This is hard to solve, but might be a future research direction. On the other hand this are problems where although contextual sentence decomposition is applied, words in the same sub-sentence do not "belong together". For example, in the sub-sentence "*[He and his friend] applied a new, complex mathematical model they had created from Albert Einstein's general theory of relativity"* the words "*he*", "*friend*" and "*Albert Einstein*" do not directly belong together. In this case the passage *"they had created from Albert Einstein's general theory of relativity"* is as a reduced restrictive relative clause but ignored by the current approach. Therefore, a future direction might be to identify and recombine more detailed constituents, resulting in finer grained sub-sentences.

The remaining categories R, L and E make up about a third of the errors. Especially the quality of input to contextual sentence decomposition is however out of its control. Some slight improvements in the parser and entity recognition will certainly provide a benefit. Errors caused by relative clause attachment are infrequent, and therefore should be of lower priority in future research. Still, a simple extension of the heuristic might prove usable, for example the recognition of number and gender of nouns is a first direction.

8 Conclusion

The final chapter provides a conclusion of the work presented in this thesis. We summarize the major contributions and results and also discuss areas where future work looks promising and improvements can be made.

8.1 Summary and Results

In the introduction of this thesis we provided the motivation for contextual sentence decomposition (CSD) that comes from for semantic full-text search. We then gave a detailed problem definition that formally introduced the building blocks of CSD: sentence constituent identification and sentence constituent recombination. We realized that the difficult part of CSD is the identification of constituents, whereas the formal definition of sentence constituent recombination already allows an immediate implementation. Following this, we therefore described two different approaches for sentence constituent identification, one based on a set of carefully devised rules, and the other one based on machine learning techniques and an inference algorithm. This allows for two different implementations of CSD and we described what the integration with an actual semantic search engine looks like. Using this search engine we then evaluated the impact on semantic full-text search quality, but we also provided a detailed comparison of the two approaches for sentence constituent identification.

All queries evaluated showed an increase in search quality due to contextual sentence decomposition. Depending on the query the absolute increases in F-measure ranged from a few percent up to 14 percent causing a relative increase of up to 90 percent. This confirmed the idea, definition and implementation of contextual sentence decomposition for semantic full-text search. Nonetheless the results did not meet our expectations and we outlined several reasons why the evaluation did not reflect the true potential of contextual sentence decomposition.

The evaluation and comparison of the rule based and machine learning based approaches of sentence constituent identification confirmed that both are eligible and we revealed that initially our set of simple, carefully devised rules performs better than the rather complicated machine learning based approach. However, we pointed out that there is much potential for the machine learning based approach by increasing its training set and improving its inference algorithm. This should also allow for correct decomposition of complex, long and intertwined sentences, where the rule based approach has most problems. Of course, further engineering of rules is possible as well, but it requires careful work, linguistic knowledge and experience and is also language specific. Nonetheless the rule based approach already contains a few simple yet effective rules, which lends itself to suggest a hybrid approach that combines these effective rules with robust machine learning classifiers as a promising candidate for future work.

No explicit comparison of performance with respect to processing speed has been performed, but initial experiments show that the evaluation of rules is almost two orders of magnitude faster than the machine learning based approach. However, processing speed was not a top priority in the current implementations and we are therefore confident that, when stressing this, both approaches allow an acceptable implementation to that respect.

8.2 Future Work

In the following we present areas, coarsely ordered by decreasing priority, where future research work seems promising.

More detailed evaluation. As already noted, the search quality evaluation does not truly reflect the power of contextual sentence decomposition. This is because the evaluation is performed against a ground truth containing only entities. If for example, a query returns the same false-positive entity five times, each time with a different witness, and contextual sentence decomposition manages to remove four of these false-positive results, the remaining entity-witness pair will still be in the result set and so will the entity. The removal of the four false-positive results is not reflected in the evaluation at all. If instead the ground truth contains all the entity-witness pairs, the evaluation can better reflect the power of contextual sentence decomposition and will show better results. This is in line with our subjective observation when using the search engine, which gave reason to higher expectations. However, the annotation of a large amount of entity-witness pairs is a tedious and costly task with too much effort for the context of this thesis and should therefore be part of future work.

Improvement of the machine learning based approach to sentence constituent identification. First of all the size of the training set must be increased. During development increasing the training set size from 130 to 200 sentences improved the F-measure of each classifier at least 5 percent and it can be expected to increase even further. This will make the resulting classifiers more robust. Some more work on feature and parameter selection can then further improve the inference phase, making it less sensitive to wrong suggestions. If the used classifiers work better different weighting approaches in the inference algorithm might show far better results and, as a consequence, a major performance increase can be expected. Other improvements might be the training of only one constituent end classifier instead of three, the incorporation of errors of the filtering phase into the training of inference phase classifiers or the integration of rules to form a hybrid approach (see below) as well as the exchange of Support Vector Machines against other machine learning techniques.

Improvement of the rule based approach to sentence constituent identification. The rule based approach works well already, but some minor and easy changes should provide a further increase in performance. For example, the recognition of commas in dates like "*4 July, 2011*" or brackets as relative clauses *"the photoelectric effect (which gave rise to quantum theory)*" could easily be accomplished. One of the major drawbacks of this approach is currently that the formulation of new rules is done using algorithms, in essence implemented in C++. The development of a domain specific language for rules would make implementation and thus evaluation of new rules easier and faster. A next step might then be the improvement of rules to better recognize embedded constituents.

Designing a hybrid approach. Depending on the outcomes above, a promising approach seems to be the integration of easy yet effective rules with robust machine learning techniques. This increases processing speed and might decrease problem sizes for the inference algorithm, for example, if sentences can reliably be split beforehand. This would combine the best of both worlds.

Extending contextual sentence decomposition. An extension of contextual sentence decomposition could be evaluated. This refers to additional constituent types as well as a possible adaption of SCR. We note that it is in fact debatable to what extent a decomposition of sentences is reasonable. For example, should the subsentence "*He and his friend applied a new, complex mathematical model they had created from Albert Einstein's general theory of relativity"* be further decomposed into *"He and his friend applied a new, complex mathematical model"* and *"a new, complex mathematical model they had created from Albert Einstein's general theory of relativity"*? An extension of contextual sentence decomposition, in the above case for restrictive relative clauses, but also in general, is certainly feasible and an evaluation would allow insight to its advantages and disadvantages and shed some light. Another thing to consider in future is whether an extension of constituent types allows better representation of certain sentence structures. For example, the participle construction in *"When blood sugar levels become too high, insulin is released from the pancreas, lowering the blood sugar levels"* or the adverbial phrase in *"Dembo [...], because he was weakened by diabetes, died ten days later*" are currently not ideally represented, resulting in erroneous decomposition.

Evaluation of further natural language processing techniques.

The current selection of natural language techniques focused on fast, but shallow approaches. Evaluations of other natural language processing techniques that provide a semantically richer output might allow better recognition of constituents and facilitate contextual sentence decomposition. For example, information similar to a dependency graph, or more close to a complete parse tree would greatly benefit contextual sentence decomposition. Annotations similar to Semantic Role Labeling might prove useful as well. If these approaches allow an implementation that is fast and accurate enough, they might prove usable and replace or extend the currently used approaches.

Sentence constituent recombination. We can improve sentence constituent recombination in mainly two ways. On the one hand the attachment of relative clauses can be extended by discovering gender and number of nouns they possibly attach to. Because, according to our evaluation, the current heuristic works well we cannot expect a large increase of performance, however the effort might be small enough to justify it. On the other hand the assignment of list items to their enumeration is currently very simple, assuming continuous list items belong to the same enumeration. This does not allow intermediate structures. An improvement is certainly possible in this area, for example by assigning list items syntactic or semantic categories and grouping them accordingly.

Bibliography

Baeurle, F. 2011. A user interface for semantic full text search. Master thesis.

Barbella, D., Benzaid, S., Christensen, J. M., Jackson, B., Qin, X. V., and Musicant, D. R. 2009. Understanding support vector machine classifications via a recommender system-like approach. In *Int. Conf. on Data Mining*. 305–311.

Bast, H., Chitea, A., Suchanek, F. M., and Weber, I. 2007. Ester: efficient search on text, entities, and relations. In *SIGIR*, W. Kraaij, A. P. de Vries, C. L. A. Clarke, N. Fuhr, and N. Kando, Eds. ACM, 671–678.

Bast, H. and Weber, I. 2007. The completesearch engine: Interactive, efficient, and towards ir& db integration. In *CIDR*. www.crdrdb.org, 88–95.

Brill, E. 1992. A simple rule-based part of speech tagger. In *ANLP*. 152–155.

Buchhold, B. 2010. Susi: Wikipedia search using semantic index annotations. Master thesis.

Burges, C. 1999. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*.

Carreras, X. 2005. Learning and inference in phrase recognition: A filtering-ranking architecture using perceptron. Ph.D. thesis, Universitat Politècnica de Catalunya.

Carreras, X. and Màrques, L. 2004. Introduction to the conll-2004 shared task: Semantic role labeling. In *Proceedings of CoNLL-2004*. Boston, MA, USA, 89–97.

Chandrasekar, R., Doran, C., and Srinivas, B. 1996. Motivations and methods for text simplification. In *Proceedings of the 16th conference on Computational linguistics - Volume 2*. COLING '96. Association for Computational Linguistics, Stroudsburg, PA, USA, 1041–1044.

Joachims, T. 1998. Text categorization with support vector machines: Learning with many relevant features.

Klebanov, B. B., Knight, K., and Marcu, D. 2004. Text simplification for information-seeking applications. In *Conference on Cooperative Information Systems*. 735–747.

KUDOH, T. AND MATSUMOTO, Y. 2000. Use of support vector learning for chunk identification. In *Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on Computational natural language learning - Volume 7*. ConLL '00. Association for Computational Linguistics, Stroudsburg, PA, USA, 142–144.

MARCUS, M. P., MARCINKIEWICZ, M. A., AND SANTORINI, B. 1993. Building a large annotated corpus of english: the penn treebank. *Comput. Linguist. 19*, 313–330.

MARQUEZ, L. AND SALGADO, J. G. 2000. Machine learning and natural language processing.

R, C. C., S, S., I, B. M. T., A, F. R., BROWMAN, K. E., AND CRABBE, J. C. 1986. Wilf: Algorithms and complexity. In *Proceedings of ISSAC 94*. Prentice-Hall, 264–268.

SANG, E. F. T. K. AND DÉJEAN, H. 2001. Introduction to the conll-2001 shared task: Clause identification. *Computing Research Repository cs.CL/0107*.

SCHMID, H. 1994. Probabilistic part-of-speech tagging using decision trees.

SIDDHARTHAN, A. 2003. Syntactic simplification and text cohesion. Ph.D. thesis.

TJONG KIM SANG, E. F. AND BUCHHOLZ, S. 2000. Introduction to the conll-2000 shared task: chunking. In *Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on Computational natural language learning - Volume 7*. ConLL '00. Association for Computational Linguistics, Stroudsburg, PA, USA, 127–132.

WEI CHEN, Y. 2005. Combining svms with various feature selection strategies. In *Taiwan University*. Springer-Verlag.