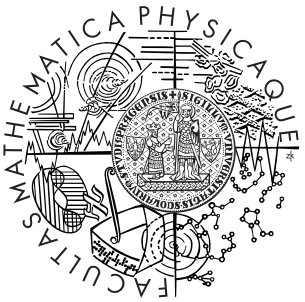
Charles University in Prague
Faculty of Mathematics and Physics

DIPLOMA THESIS



Ondřej Dušek

Deep Automatic Analysis of English

Institute of Formal and Applied Linguistics

Supervisor: Prof. RNDr. Jan Hajič, Dr.
Study program: Computer Science
Study field: Mathematical Linguistics

2010

I would like to express my thanks to Prof. RNDr. Jan Hajič, Dr. for inspiring and supervising this thesis.

I am also very grateful to my brother Pavel for his grammar corrections, to my whole family for their support and to my dearest Jana for her love and care.

I certify that this diploma thesis is my own work, and that I only used the cited literature. The thesis is freely available for all who can use it.

Prague, August 6th, 2010 Ondřej Dušek

Contents

Title: Deep Automatic Analysis of English
Author: Ondřej Dušek
Department: Institute of Formal and Applied Linguistics
Supervisor: Prof. RNDr. Jan Hajič, Dr.
Supervisor's e-mail address: hajic@ufal.mff.cuni.cz

Abstract: This thesis contains an account of our studies of deep or semantic analysis of English, particularly as described using predicate-argument structure description. Our main goal is to create a system for automatic inference of semantic relations between predicates and arguments — semantic role labeling. We developed a framework for parallel processing of our experiments, integrating third-party machine learning tools and implementing well-known as well as novel procedures. We investigated the current approaches to the problem and proposed several improvements, such as new classification features, separate handling of adverbial modifiers or special treatment for rare predicates. Based on our research, we designed and implemented our own semantic analysis system, consisting of predicate disambiguation and argument classification subtasks. We evaluated our solution using the CoNLL 2009 Shared Task English corpus.

Keywords: deep analysis, semantic role labeling, machine learning

Název práce: Hloubková automatická analýza angličtiny
Autor: Ondřej Dušek
Katedra (ústav): Ústav formální a aplikované lingvistiky
Vedoucí diplomové práce: Prof. RNDr. Jan Hajič, Dr.
e-mail vedoucího: hajic@ufal.mff.cuni.cz

Abstrakt: Tato diplomová práce popisuje studii hloubkové, tj. sémantické analýzy angličtiny, zejména na základě teoretického popisu pomocí propozic a jejich argumentové struktury. Hlavním cílem práce je vytvořit systém pro automatickou klasifikaci sémantických vztahů mezi propozicemi a jejich argumenty — značkování sémantických rolí. Vyvinuli jsme prostředí pro paralelní zpracování experimentů, přičemž jsme integrovali existující nástroje pro strojové učení a implementovali jak již popsané, tak nové postupy. Prostudovali jsme známé přístupy k tomuto problému a navrhli několik vylepšení, jako např. nové rysy pro klasifikaci, oddělené řešení pro příslovečná určení nebo zvláštní zacházení s řídkými predikáty. Na základě tohoto výzkumu jsme zkonstruovali vlastní systém pro sémantickou analýzu, který sestává z modulů pro disambiguaci predikátů a klasifikaci argumentů. Práce je zakončena testem našeho systému na anglickém korpusu určeném pro soutěž CoNLL 2009 Shared Task.

Klíčová slova: hloubková analýza, značkování sémantických rolí, strojové učení

1 Introduction

1.1 The Problem of Deep Language Analysis

Both in descriptive linguistic theories and in automatic natural language processing (NLP) applications, the formalization of the language concept is usually split into several linguistic layers, which differ in the level of abstraction — from the bare matter of spoken word or written text itself up to the cognitive content of the utterance. The number and extent of those layers may vary across different approaches and theories, but their hierarchical structure remains. Language descriptions most usually include phonetics or graphemics[1], morphology, syntax, semantics and pragmatics. Our work is mainly concerned with semantics, i.e. with obtaining the linguistic meaning (Sgall et al., 1986, p. 35ff.) of utterances[2]. We consider the language analysis to be a sequence of analyses on the individual levels, successively climbing from the most concrete to more abstract layers. Therefore, the deep or semantic analysis comes after the phonetic, morphological and syntactic (i.e. shallower) analyses have taken place.

In some theoretical approaches, such as the Prague *Functional Generative Description* (FGD) (Sgall et al., 1986), the description of linguistic meaning is included in the deep syntax (*tectogrammatical*) layer. On the other hand, most resources used in this thesis, such as the *Proposition Bank* (PropBank) (Palmer et al., 2005), denote the analogical layer of annotation as semantic. Nevertheless, both descriptions work with a labeled dependency structure, which is the main object of the following work. Other meaning representations, such as *FrameNet* (Baker et al., 1998), operate with comparable constructs.

[1] In NLP, the input text split to individual words or tokens is the first descriptive layer.

[2] Please note that linguistic meaning differs from the cognitive content, as it refers solely to the content expressed by the language, without taking mental processes or pragmatics into account.

While the FGD and the projects stemming from it, such as the *Prague Dependency Treebank* (PDT) (Hajič et al., 2006) or the *Prague English Dependency Treebank* (PEDT) (Cinková et al., 2009) use labeled dependency tree structure on the tectogrammatical level, the PropBank annotation (Kingsbury and Palmer, 2003) employs a predicate-argument description of *propositions — semantic predicates*, i.e. meaning representations of events that require or attract other objects, their *arguments*, to complete situation in various ways.

The underlying notions of *valency* (the ability to bind depending elements) and *semantic roles* (cf. e.g. Gildea and Jurafsky, 2002; Carreras and Marquez, 2005; Sgall et al., 1986, p. 123) (semantic types of the dependency relation) are visible in both structures, though. In this thesis, we focus on the predicate-argument approach to the description of these phenomena.

The task of deep analysis, also referred to as *Semantic Role Labeling* (SRL), of an English sentence using predicate-argument structure may be then divided into following sub-tasks:

1. *Predicate Identification* (PI) — one must first identify all words that function as semantic predicates in the given sentence.

2. *Predicate Disambiguation* (PD) — the word sense of each predicate must be determined.

3. *Argument Identification* (AI) — for each predicate in the sentence, we have to find all its arguments.

4. *Argument Classification* (AC) or *Argument Labeling* — the particular semantic role of each predicate needs to be identified.

The output of automated semantic analysis may then find further applications in natural language processing, namely in information retrieval, question answering or machine translation systems. Most of the current research in this field concentrates on using statistical approaches — applying a *machine learning* (ML) classifier algorithm, such as a *support vector machine* (SVM) (Boser et al., 1992) or a *maximum entropy model* (MaxEnt) (Jelinek, 1997), to estimate a set of unknown parameters using *features*[3] of the hand-annotated training data instances (i.e. any characteristics of the individual instances, such as word forms, parts of speech, syntactic features etc.) and then to use these parameters to automatically classify further texts.

1.2 The Aims of This Work

The annual *Conferences on Computational Natural Language Learning* (CoNLL) include usually a Shared Task — a competition of natural language processing systems. In 2009, the Shared Task (Hajič et al., 2009) featured syntactic parsing and semantic analysis tasks in seven languages, including English. It was possible to take part either in both syntactic and semantic subtasks, or in semantic

[3] In statistics, features are usually called *attributes*.

analysis alone. The participants of the SRL task obtained a semantically annotated corpus for each language to train their deep analysis systems, along with a set of evaluation sentences that was used to compare and rank the individual setups. The predicate identification was not included in the task assignment — the semantic predicates were marked in both training and evaluation data.

This competition provided us with an excellent source of inspiration. Since the training and evaluation corpora are now available to us, we can use them for further experiments. Our main goal is therefore to design and implement our own statistical classifier system for deep analysis of English and apply it to the CoNLL 2009 Shared Task corpora.

In doing so, we combine and test various known approaches and include our own ideas and settings in order to achieve better performance. We also evaluate our system using the same standard metric that was employed in the last year's competition, so that we may directly compare our system to other setups.

1.3 Structure of the Thesis

In Chapter 2, we discuss the related work in the field of semantic analysis, including, but not limited to the CoNLL 2009 Shared Task participants' papers, which describe the individual SRL system setups.

Chapter 3 is focused on the description of the CoNLL 2009 English corpus, which we adopt for our own experiments. We summarize its original sources, the annotation provided and the data format employed. A comparison to the Prague tectogrammatical annotation (Cinková et al., 2009) is also included.

We have designed and implemented our own Java framework for processing multiple machine learning tasks in parallel, which we then apply in our semantic analysis system, in combination with third-party ML libraries. The description of our framework and the integration of external libraries may be found in Chapter 4.

Chapter 5 then continues with a brief explanation of the known ML techniques we adopted for our SRL system, using the integrated third-party libraries and our own implementations: classifier algorithms, feature generation, ranking and selection. In addition, we describe several new features which we designed for our system, as well as our own implementation of the required input data conversion.

An account of our own research — a detailed description of the particular applications of the adopted approaches and newly introduced improvements in our own SRL solution then continues in Chapter 6, which concentrates on predicate disambiguation, in Chapter 7 focused on argument identification and classification and finally in Chapter 8, which contains the overall description of our setup and a comprehensive analysis of its performance.

Chapter 9 then concludes the thesis with a summary of our results, followed by improvement and further application proposals.

2 Related Work

Deep analysis in the form of automatic inference of abstract semantic roles, introduced by Gildea and Jurafsky (2002), has been a very active field of research in recent years. The application of supervised ML, i.e. estimating the unknown parameters of a classification model statistically from hand-annotated (*gold standard*) training data set, has been made possible with the emerging of semantically annotated treebanks, such as the PDT (Hajič et al., 2006) or PropBank (Palmer et al., 2005). Since it also has been a subject of the last year's CoNLL Shared Task, as we already described in Section 1.2, the proceedings of this contest contain an up-to-date reference of various approaches to SRL, which we discuss in Section 2.1. We also include a short account of other important research and practice in the field of semantic analysis in Section 2.2.

2.1 The CoNLL 2009 Shared Task

The CoNLL 2009 Shared Task represents a project, part of which is most relevant to the topic at hand[1]. Therefore, the participants' papers describing 18 of the competing system setups comprise a voluminous source of information and ideas regarding semantic analysis.

First, we will analyze the architecture of the CoNLL 2009 SRL systems in order to consider some selected concepts in our own system later. In general, all of the systems included a statistical learning algorithm and various feature selection and data pruning techniques. However, the particular ML methods applied vary dramatically, as do the overall organizations of the systems. Most of the participants divided the whole task into subtasks (similar to our listing in Section 1.1) and arranged the setup as a pipeline, each step of which solves one of

[1]Most of the systems are divided into the parsing and SRL subsystems. In the following, we will omit the syntactic subtask and concentrate solely on methods applied for the semantic analysis.

the subtasks. The individual steps could then make use of different classification and feature selection techniques. The organization of the pipeline varies among the systems: While most of them solve the PD task as the first step of the semantic analysis process, several (Bohnet, 2009; Zhao et al., 2009) prefer to solve the AI and AC tasks first. Some of the systems also combine the AI and AC tasks into one (Che et al., 2009; Björkelund et al., 2009); Meza-Ruiz and Riedel (2009) even created a system which joins all the subtasks into one.

Now let us take a look at the machine learning cores of the participating setups: The approaches of Zhao et al. (2009), Che et al. (2009) and Dai et al. (2009), as well as of several others, feature a MaxEnt classifier, which is a very common and successful practice in various NLP tasks. Björkelund et al. (2009) apply logistic regression on the data, which is in a way similar to the MaxEnt method[2]. Che et al. (2009) also use SVM classifers in the PD part of their system, as does Täckström (2009). Other systems (Bohnet, 2009; Watanabe et al., 2009) employ various maximum-margin algorithms, such as the margin-infused relaxed algorithm (MIRA) (Crammer and Singer, 2003). There is also a group of systems based on diverse logic devices: Meza-Ruiz and Riedel (2009) use Markov logic networks in combination with the MIRA algorithm, while Moreau and Tellier (2009) work with Conditional Random Fields and Gesmundo et al. (2009) utilize the incremental sigmoid belief networks. Many systems included not one, but multiple classifiers, one for each predicate lemma or sense (Che et al., 2009); (Björkelund et al., 2009). The results of the competition show that most of the various machine learning approaches used are capable of reaching a comparable quality level of results if trained properly[3].

Most systems use very similar kinds of features that are extracted from the training and evaluation data sets, exploiting the provided morphological and syntactical annotations and combining it to describe the various relations between the words[4] in a sentence. Watanabe et al. (2009) use also "global features" (a set of all arguments relating to one predicate) in their semantic labeler. Björkelund et al. (2009) incorporate feature bigrams in order to enlarge the set of relations indicated.

The applied feature selection techniques include the greedy search procedure (Björkelund et al., 2009; Zeman, 2009, among others), as well as beam search (Gesmundo et al., 2009; Björkelund et al., 2009, and others) and various ranking approaches. Some authors chose to prune the semantic argument candidates to reduce the amount of data that is needed be passed to the classifier — both Zhao et al. (2009) and Watanabe et al. (2009) describe very similar solutions to this problem.

In addition, most of the authors introduced further enhancements into their systems. The iterative approach, where the results of the semantic analysis are used as an input for a repeated classification, taken by Dai et al. (2009) is one

[2]We will discuss this similarity in Section 5.2.

[3]This becomes apparent also from our own classifier selection and tuning tests (see Sections 5.2 and 6.2).

[4]Please note that in this and the following chapters, the term "word" refers not only to words, but also to all other tokens in the input text, such as punctuation signs.

of them. Other researchers included global re-ranking of semantic argument candidates (Björkelund et al., 2009) or other forms of post-inference, such as integer linear programming (ILP) (Che et al., 2009).

The CoNLL 2009 Shared Task contest has provided us with a variety of options to consider for building our own SRL system setup, as well as further insights into the problem: Some of the competitors' papers, such as that of Zeman (2009), also included a study of the data we intended to use and raised some questions about its sparseness, i.e. how many training examples are available in the individual subproblems.

2.2 Other Approaches to Deep Language Analysis

There are also many other works in semantic analysis and particularly in automatic SRL systems using machine learning techniques. The CoNLL Shared Tasks of 2004, 2005 (Carreras and Marquez, 2004, 2005) and 2008 (Surdeanu et al., 2008) have all been dedicated to semantic analysis of English using the PropBank corpus[5]. The SRL solvers described in the proceedings of these contests mostly follow the pipeline classification scheme described in Section 2.1, with MaxEnt and SVM as the most widely used machine learning techniques. Some of the 2008 systems have been adapted for the 2009 competition, which provides us with more information about the versions of both years, as well as additional reports of gradual improvements (Che et al., 2008; Chen et al., 2008).

Further research concentrates on similar tasks: Jiang and Ng (2006) provide an analysis of NomBank (Meyers et al., 2004) semantic annotation and a description of a MaxEnt deep analysis system. Punyakanok et al. (2004) introduced the ILP technique into SRL and gave a detailed report. Giuglea and Moschitti (2006) presented a system that retrieves its semantic annotation training data from three different resources — PropBank, NomBank and FrameNet. Loper et al. (2007) are currently developing a project to link several annotated resources permanently, which has been a valuable tool for our preliminary analyses of predicate behavior.

[5]The 2008 contest also included syntactic parsing.

3 Data Used

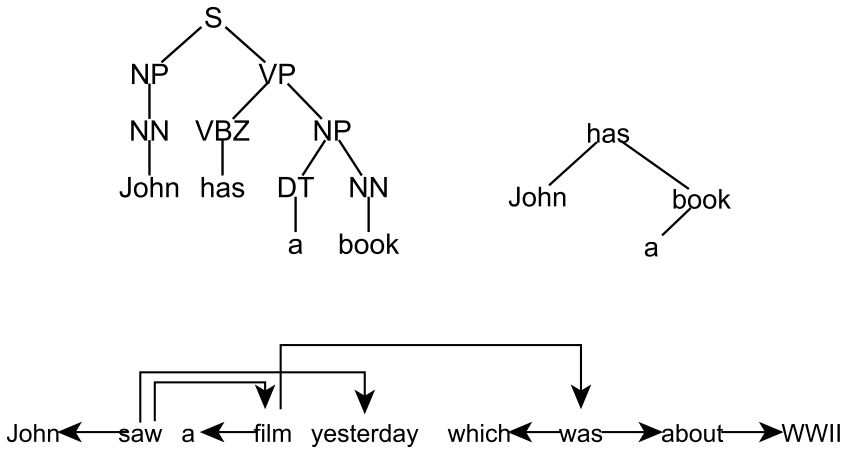
The CoNLL 2008/2009 English corpus (Surdeanu et al., 2008; Hajič et al., 2009), which we adopted for the experiments with our deep analysis system, unites data from different sources to provide the deep level of annotation. It consists of English articles from the Wall Street Journal with morphological and (thoroughly modified) syntactical annotation of the Penn Treebank (PTB) (Marcus et al., 1993) (described in Section 3.1), combined with the semantic annotation of verbal and nominal predicates from PropBank (Palmer et al., 2005) and NomBank (Meyers et al., 2004), respectively. We provide an account of the semantic schema applied (in Section 3.2) and compare it to the Prague FGD approaches in Section 3.3. Section 3.4 concludes this chapter with a detailed explanation of the original CoNLL corpus format, including basic data statistics.

3.1 Syntactic Annotation

The Penn Treebank was one of the first large syntactically parsed corpora and is still widely used as a standard English data set for various NLP tasks that require syntactic information. The morphological annotation of this corpus features a tagset (Santorini, 1990) with about forty mnemonic tag names that mark the individual English parts of speech and their inflectional subtypes. The data for this linguistic layer has been used for the CoNLL 2009 corpus without changes. However, the syntactic part of PTB has been adapted to reflect the dependency paradigm in syntactic and semantic description.

The original PTB employed a constituent parse tree schema (see Figure 3.1) based on the theory of Government and Binding (Chomsky, 1981). Each inner node of the tree corresponds to one or more immediately adjacent words in the sentence — a phrase. An edge in the tree corresponds to the relation of immediate constituency. The nodes in dependency trees (also displayed in Figure 3.1) by contrast do not represent constituents, but only the individual words. Its edges

Figure 3.1: Syntactic representations of sentences



The top left picture shows a parse tree, the top right picture displays a dependency tree. The bottom chart is a depiction of a non-projective sentence in a dependency paradigm (arrows represent dependencies). The non-projectivity is indicated by crossing of arrows.

then show the syntactic dependency relation. Since there is no limitation on the position of dependent nodes in the sentence as with constituent trees, one can easily represent non-projective sentences, where one or more dependent nodes are topologically separated from their head nodes by other nodes (an example is shown in Figure 3.1). Such sentences, which pose a problem for the constituent tree description (the phrases must not overlap), are not very common in English, but occur very often in certain other languages.

Since dependency parsing algorithms have become more effective and can handle non-projectivity (cf. McDonald et al., 2005), it is more convenient to use dependency structures to describe syntactic relationships. Therefore, the PTB corpus has been automatically converted to a dependency schema for the purposes of CoNLL 2008/2009 (Johansson and Nugues, 2007; Surdeanu et al., 2008), while its edge labels have been adapted to meet the new paradigm and further enriched, so that they can serve to a better automatic semantic analysis[1]. The modified dependency labels also include named entity indication, which originates from the BBN Pronoun Coreference and Entity Type Corpus (Weischedel and Brunstein, 2005).

3.2 PropBank and NomBank Semantic Annotation

The semantic information in the CoNLL English corpus incorporates annotations from PropBank and NomBank, both of which feature a predicate-argument

[1]For details on the dependency labels and syntax in this corpus, see the file format and the description of the DEPREL field at `http://yr-bcn.es/conll2008`.

structure (cf. Section 1.1 and Kingsbury and Palmer, 2002). Not only the structure itself is very similar, they also use the same approach to semantic argument labeling. They distinguish the following four kinds of semantic arguments[2] [3]:

- (Valency) *arguments*, semantically required complements of the given predicate, which are characteristic for a given predicate. The semantic labels of arguments are numbered (`A0...A5`) and although there are some common prototypes, the number assignment is specific for each predicate, thus creating predicate *frames* (see below).

- *Argument modifiers* or *adverbial modifiers*[4], voluntary semantic modifications or complements of predicates, which are not specific to one predicate, but may occur virtually in any sentence. They include e.g. the indication of time, manner or other properties of the event or entity described by the predicate.

- *References*, which carry out the same function as arguments or modifiers expressed elsewhere in the text. They are often represented by pronouns.

- *Coreferences*, which behave similarly to references, except for the fact that they must occur only after the referenced argument in the same sentence.

There are indeed minor differences between the observed labels in NomBank and PropBank, but their basic concepts remain the same.

As already mentioned, each predicate, i.e. each noun or verb in a given sense, has its own argument number assignment or *frame* — even nouns and verbs with the same base form (*lemma*) may use different numberings. The frames describe the individual arguments and their semantic properties, e.g. in most verbs, `A0` refers to the semantic actor of the event. A lexicon of frames is provided for all the predicates that occur in the corpus, thus completing its semantic description.

3.3 Comparison to Prague English Dependency Treebank Annotation

As we mentioned in Section 1.1, the predicate-argument description is not the only way of representing semantic information. The PEDT (Cinková et al., 2009), a semantic resource which is currently being developed at the Institute of Formal and Applied Linguistics, works with the same Wall Street Journal articles as the CoNLL corpus and features a syntactic layer (called *analytical* in the FGD-nomenclature) with dependency trees, which were also converted from the constituent PTB trees. However, it also employs a semantic (*tectogrammatical*)

[2]For more details on PropBank annotation, see (Moreda and Palomar, 2006) or the instructions at `http://verbs.colorado.edu/~mpalmer/projects/ace.html`.

[3]The NomBank annotation is thoroughly described in the guidelines at: `http://nlp.cs.nyu.edu/meyers/nombank/nombank-specs-2007.pdf`.

[4]For additional consequences of the distinction between valency arguments and adverbial modifiers for SRL tasks, please refer to Section 7.3.

dependency tree schema: rather than using predicates and arguments which pertain exclusively to nouns and verbs, it organizes all the words that carry semantic information into a hierarchy of dependencies. The words whose function is mainly grammatical, such as articles, prepositions or function verbs[5], do not occur in the semantic tree as separate nodes, but are represented as features of other nodes. In addition, there are also some nodes for words that are not explicitly expressed in the utterance, but their presence follows from its semantic content.

A comparison of both annotations is shown in Figure 3.2[6]. It is apparent that the PEDT annotation encompasses richer semantic information, because it also contains semantic dependencies of adjectives and adverbs, as well as more detailed structure when compared to the two-level structure of PropBank and NomBank. The semantic role labels, which adhere to the FGD, are also different from the numbered arguments of PropBank and NomBank, even though the PEDT valency lexicon EngVALLEX (Semecký and Cinková, 2006) is originally based on PropBank and NomBank frames. While a wide labels remain semantically constant across different predicates, the most frequent valency arguments are subject to *shifting* with verbs, i.e. the first argument of any verb is always called `ACT` and the second one `PAT`, even if they do not denote the semantic roles of an actor and an affected object, as it should be in a prototypical case[7]. The references are treated in a different way as well: Additional links are inserted into the tree structure to represent them.

It is however still possible to find some common grounds between the two sources — both of them include labeled dependencies between noun and verbal predicates and their required or voluntary arguments. The Czech part of the CoNLL 2009 contest included the PDT data whose annotation schema is also based on the FGD and uses the same dependency concept; therefore, it is probably feasible to adjust a SRL system designed for the CoNLL annotation for use with the PEDT data, as long as it remains within the limits of finding arguments of nominal and verbal predicates. Full semantic dependency parsing would probably require more radical modifications.

3.4 The CoNLL Corpus: Data Format and Statistics

Since we have described the annotation schema and sources of the CoNLL 2009[8] corpus, we may now turn our attention to its technical implementation. The data is pre-divided into three sets: *training*, *development* (tuning) and (final) *evaluation* sentences. Each of the sets is included in a single text file, which contains all the plain text, morphological, syntactic and semantic information.

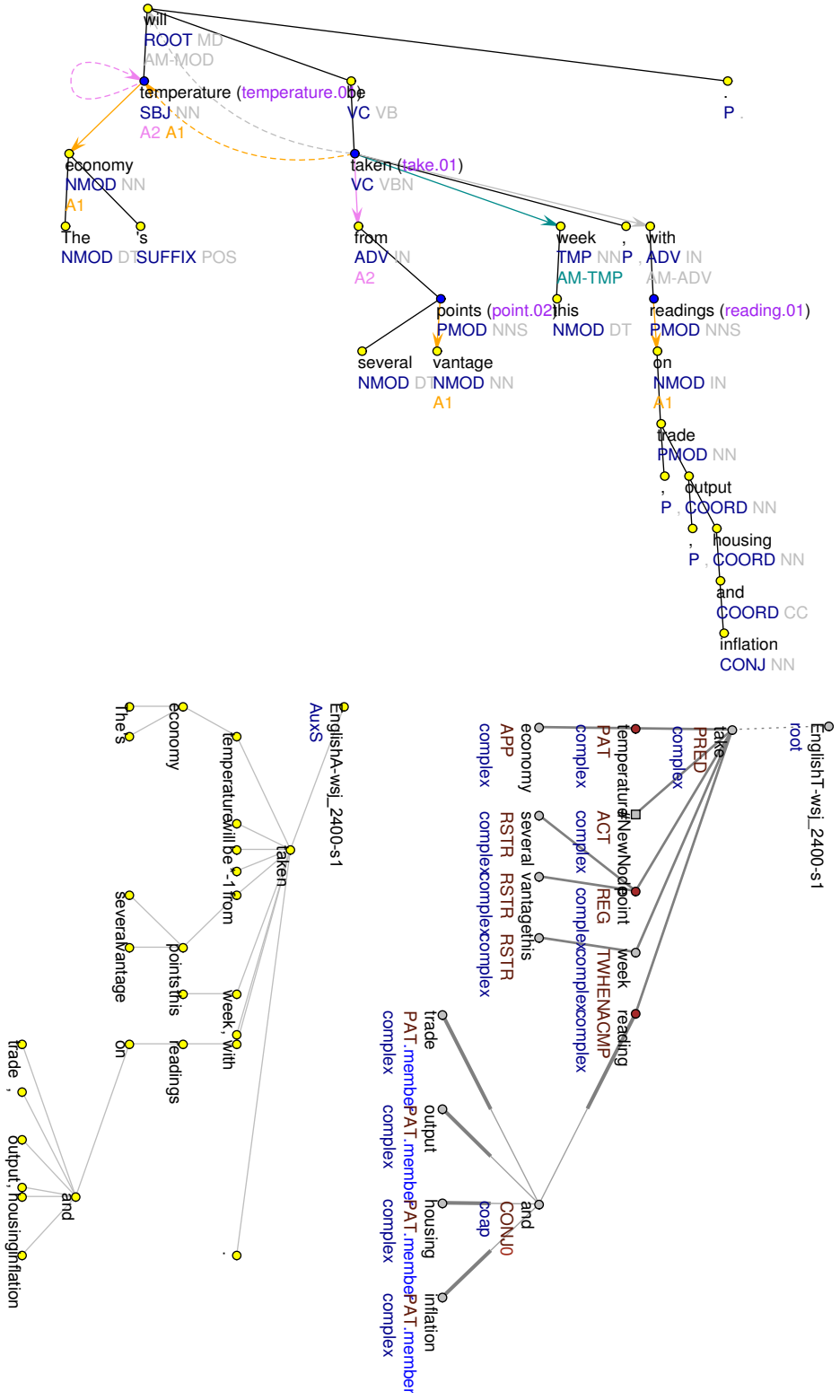
[5] This also pertains to punctuation tokens, which are usually included in syntax parse trees.

[6] The tree charts in this figure were created by the TrEd tree editor (`http://ufal.mff.cuni.cz/~pajas/tred/`).

[7] Cf. a detailed description in the annotation guidelines at: `http://ufal.mff.cuni.cz/~cinkova/TR_En.pdf`, pg. 36ff. and 107ff.

[8] As the concrete data format differs slightly from the 2008 version, we will now refer solely to the 2009 data sets used in this thesis.

Figure 3.2: A comparison of PEDT and CoNLL semantic annotation.
The top picture shows the CoNLL (both syntactic and semantic) annotation — semantic edges between predicates and arguments are marked in color. The bottom picture is the PEDT analytical (left) and tectogrammatical (right) tree for the same sentence.



Each line in every text file comprises all the knowledge about one word (or punctuation sign), the individual types of inputs separated by tabs, forming columns. The purpose of each column is described in Table 3.1[9]. Different sentences are divided by empty lines.

Table 3.1: The data columns of the CoNLL Shared Task corpus format

No.	Name	Contents
1	ID	Number of the token in the sentence
2	FORM	The word form as it appears in the original text
3	LEMMA	Gold standard dictionary lemma of the word form
4	PLEMMA	Automatically predicted lemma
5	POS	Gold standard part-of-speech
6	PPOS	Automatically predicted part-of-speech
7	FEAT	(Not used for English)
8	PFEAT	
9	HEAD	ID of the syntactic head of this word (0 for root node; gold standard)
10	PHEAD	Automatically predicted syntactic head ID
11	DEPREL	Dependency relation label (gold standard)
12	PDEPREL	Automatically predicted dependency relation label
13	FILLPRED	Contains `Y`, if the word is a predicate
14	PRED	The predicate name (lemma and sense number)
15+i	APREDi	Arguments of i-th predicate in the sentence

It is apparent from Figure 3.3 that the corpus data format is very economic — no data is repeated — and easily human-readable. Manual analysis of corpus sentences is further simplified by the CoNLL 2009 Shared Task Extension for the TrEd tool[10]. On the other hand, the `APRED` columns and their different number for different sentences, as well as the fact that their order is only determined by the order of predicates in the sentence, clearly implies that such a data format is not well suitable as an input to the commonly used machine learning classifiers, since they require a constant data structure for the whole set, thus requiring a data conversion. This topic will be discussed further in Section 5.3.

The CoNLL corpus covers virtually all of the original PTB texts, while the vast majority of the data is designated for training purposes, as in most statistical learning problems. Table 3.2 shows that most of the predicates and virtually all predicates with more than one occurrence in the development or evaluation set are covered by the training set, but a small group of unseen data will require to be treated separately.

The average number of predicates per sentence is 4.55 in the training data set. Therefore, if the system is set-up to classify the arguments of one predicate at a time (which is the only straightforward option with classic machine learning approaches), the amount of data will multiply by that number. This results in effectively more training data instances and therefore possibly a higher classifi-

[9] Please see `http://ufal.mff.cuni.cz/conll2009-st/task-description.html` for additional explanation.

[10] `http://ufal.mff.cuni.cz/~pajas/tred/extensions/`

Figure 3.3: An example sentence from the CoNLL Shared Task Corpus

1	The	the	the	DT	DT	_	_	2	2	NMOD	NMOD	_	_	_	_	_	_
2	economy	economy	economy	NN	NN	_	_	4	4	NMOD	NMOD	_	_	A1	_	_	_
3	's	's	's	POS	POS	_	_	2	2	SUFFIX	SUFFIX	_	_	_	_	_	_
4	temperature	temperature	temperature	NN	NN	_	_	5	5	SBJ	SBJ	Y	temperature.01	A2	A1	_	_
5	will	will	will	MD	MD	_	_	0	0	ROOT	ROOT	_	_	_	AM-MOD	_	_
6	be	be	be	VB	VB	_	_	5	5	VC	VC	_	_	_	_	_	_
7	taken	take	take	VBN	VBN	_	_	6	6	VC	VC	Y	take.01	_	_	_	_
8	from	from	from	IN	IN	_	_	7	7	ADV	ADV	_	_	_	A2	_	_
9	several	several	several	DT	DT	_	_	11	11	NMOD	NMOD	_	_	_	_	_	_
10	vantage	vantage	vantage	NN	NN	_	_	11	11	NMOD	NMOD	_	_	_	_	A1	_
11	points	point	point	NNS	NNS	_	_	8	8	PMOD	PMOD	Y	point.02	_	_	_	_
12	this	this	this	DT	DT	_	_	13	13	NMOD	NMOD	_	_	_	_	_	_
13	week	week	week	NN	NN	_	_	7	7	TMP	TMP	_	_	_	AM-TMP	_	_
14	,	,	,	,	,	_	_	7	7	P	P	_	_	_	_	_	_
15	with	with	with	IN	IN	_	_	7	7	ADV	ADV	_	_	_	AM-ADV	_	_
16	readings	reading	reading	NNS	NNS	_	_	15	15	PMOD	PMOD	Y	reading.01	_	_	_	_
17	on	on	on	IN	IN	_	_	16	16	NMOD	NMOD	_	_	_	_	_	A1
18	trade	trade	trade	NN	NN	_	_	17	17	PMOD	PMOD	_	_	_	_	_	_
19	,	,	,	,	,	_	_	18	18	P	P	_	_	_	_	_	_
20	output	output	output	NN	NN	_	_	18	18	COORD	COORD	_	_	_	_	_	_
21	,	,	,	,	,	_	_	20	20	P	P	_	_	_	_	_	_
22	housing	housing	housing	NN	NN	_	_	20	20	COORD	COORD	_	_	_	_	_	_
23	and	and	and	CC	CC	_	_	22	22	COORD	COORD	_	_	_	_	_	_
24	inflation	inflation	inflation	NN	NN	_	_	23	23	CONJ	CONJ	_	_	_	_	_	_
25	.	.	.	.	.	_	_	5	5	P	P	_	_	_	_	_	_

Table 3.2: Corpus size and coverage statistics for the CoNLL 2009 corpus used in this thesis

	Tokens	**Sentences**	**Predicates**		**Coverage**	
			Distinct	All	Distinct	All
Training	958167	39279	9228	179014	-	-
Development	33368	1334	2151	6390	94.9 %	98.2 %
Evaluation	57676	2399	2610	10498	95.6 %	98.8 %

The figures shown are: the total number of tokens and sentences in the individual data sets, the number distinct predicates with the total number of predicate instances and for the development and evaluation data sets, the percentage of distinct predicates and their occurrences covered by the training set.

cation precision. On the other hand, more computing resources will be needed to process such a large input.

4 The Used Machine Learning Environment

In order to achieve an easy configurability of experiments needed for the SRL system setup and their fast parallel processing on multiple interconnected computers in a computing grid or cluster, we have designed and implemented a special software framework[1]. We have chosen the Java language for our program because of code portability, high performance and simple debugging and maintenance with the help of integrated development tools. We incorporated existing ML classifiers and other NLP tools into our system, as many of the freely available utilities feature a Java API. We will now describe the basic concepts we applied in our implementation of the framework — the decomposition of an experiment into subtasks (Section 4.1) and their batch processing (Section 4.2) in parallel (Section 4.3). We also attach an account of our integration of third-party libraries (Section 4.4).

4.1 Splitting the Experiments into Subtasks

As we described already in Section 1.1, the whole SRL process that is the subject of this thesis consists of multiple subtasks. Having considered the pipeline approach to the system architecture and the implementation of multiple classifiers for different lemmas or senses (see Section 2 for details), we realized that it will be profitable to divide the whole analysis into simple, compact subtasks and process some of them in parallel. It is apparent that for multiple classifiers or other operations on independent parts of the data, parallelization is a possibility which speeds up the whole process, thus allowing more time-consuming computation.

[1] Available online at `http://code.google.com/p/en-deep/` or on the enclosed CD, see Appendix B.

Figure 4.1: Subtask definition example

```
# this is a commentary
task sttoarff; # id of the subtask
    algorithm: # used Java class
        en_deep.mlprocess.manipulation.StToArff;
    params: lang_conf="st-en.conf", # its settings
        divide_senses, one_file,  # parameters with no value (binary)
        generate="Children, DepPath, HeadPos", # parameters
        cluster_file="clusters-plain.txt";      # with values set
    in: "train.txt"; # inputs listing
    out: "train.arff"; # outputs listing
end;
```

We have considered employing the GNU Make utility[2], but decided to create our own, more specific application, which would allow us to define our custom description of the task with simpler subtask specifications[3], as well as launching the computation not only in multiple threads, but also in multiple processes running on different computers in a cluster.

We also decided to implement all the possible tasks as functions within a single executable application in order to reduce the OS overhead for process creation. The intended implementation in Java offered a simple concept which makes this possible while still maintaining modularity and extensibility: Every subtask is defined as a Java class derived from the base class called `Task`. In order to enable configurability, the individual settings or parameters of all the subtasks are then represented as name-value pairs handled by their concrete implementations[4]. We divided all the tasks implemented in our system (See Appendix A for a comprehensive list) into three groups for convenience: the classification, evaluation and data manipulation tasks.

Although many tasks may be processed in parallel without conflicts, there will necessarily be dependencies among some of them — one subtask requiring the output of another subtask as it input. The dependencies should further be handled in such a way that it is still possible to use multiple machines in a cluster for the computation. We have chosen the only straightforward way to implement this, even if it poses a load on the interconnecting network: The inputs and outputs of all tasks are always saved to files. This further simplifies the description of the subtask dependencies — it is possible to determine the necessary flow of the SRL process only by examining the names of input and output files of the individual tasks.

This all yields a task description which consists of a Java class name, its parameters and a list of inputs and outputs, which also define its dependencies

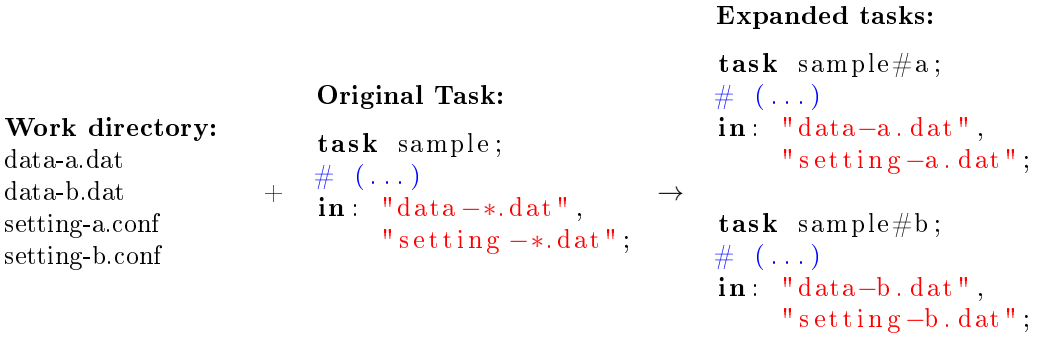
[2] `http://www.gnu.org/software/make/`

[3] This also includes the *task expansions* described in Section 4.2.

[4] It is of course possible that some tasks do not need any parameters, which is regarded as an empty list of parameters. Similarly, binary parameters are viewed as parameters with an empty value.

Figure 4.2: Task expansion example

Note that `data-a.dat` is not combined with `setting-b.dat`.



and prerequisites. The whole experiment is then a list of such (sub)tasks. We represent the descriptions in a simple configuration file called the *scenario* (see Figure 4.1), which is parsed and topologically sorted (Kahn, 1962) to create a to-do list or *plan*, used by the individual running processes (see Section 4.3) to retrieve the next subtasks that need to be computed: Each subtask description in the plan contains an indication whether all of its prerequisites have already been completed.

4.2 Wildcards and Task Expansion

Since we planned on using multiple machine learning classifiers or other operations which run multiple times on different data sets, we needed to have some means of describing that the same subtask should be launched in several variants. Therefore, we introduced simple wildcards into the description of the inputs and outputs of subtasks. An asterisk character ("*") in a file name within the input or output specifications may stand for any string, thus forming a pattern. All the files from the current directory that match it are then transfered to the subtask definitions in the following two modes determined by the number of asterisks in the file name:

- *Expanding mode ("*")*, which causes the subtask to *expand* to multiple tasks, each taking one of the matching files. If there are multiple expanding mode patterns in the input specifications, only the corresponding ones, i.e. those where the "*" stands for the same string, are put together to create an expanded task (see Figure 4.2 for an example).
- *Listing mode ("**")*, which lets the subtask take all the matching files as its input, or create multiple matching files on the output. The output expansions are left up to the individual impelementations of the subtasks.

These basic patterns were then further improved to suit all the batch processing needs during our experiments, so that they now allow for further specification

enclosed in vertical bar ("|") characters immediately following the asterisk. The detailed specifications are ignored when determining the subtask dependencies, which allows to select inputs from different sources whose file names do not match (in contrast to the situation shown in Figure 4.2), which results in a cartesian product. Further, it is possible to combine the two pattern modes, where at first, the listing mode is applied and then the expanding mode to the results of the first step.

The expansion of a task is also transferred to its dependencies, if they contain the expanding mode patterns in their input specifications. This is done already with the expansion of the prerequisite task, so that the individual computing processes (see Section 4.3) may start working on some of the expansions of the dependency even if other prerequisite expansions are not yet finished. However, the topological order of the whole plan is always preserved.

An important feature of our experiment framework is also the possibility that the individual tasks assign further subtasks themselves, which then occupy the immediately following position in the experiment plan. Inner dependencies of all such subtasks must then always result in one single termination subtask, so that the outer dependencies are left untouched. This allows for further parallelization of such time-consuming operations as greedy feature selection (see Section 5.5).

4.3 Running the Tasks in Parallel

As we already mentioned in the previous sections of this chapter, our whole experiment software is designed to run in several instances in parallel. The number of working threads, i.e. the level of maximum possible parallelization along with the maximum possible load on the CPUs and data flow, is determined by the user: The main executable may be launched in multiple copies on the same machine, as well as on different computers in a networked cluster. There is also a possibility to start multiple threads within one program instance, which simplifies the usage with stand-alone multi-core machines.

The running threads are then completely independent of one another — the only information exchange among them takes place in the plan file (see Section 4.1). Each thread will automatically parse the experiment scenario configuration file, if it does not find the plan file in the working directory, i.e. the first thread started will accept this duty. Consequently, all the working instances retrieve the pending subtasks from the plan file and remove the completed ones, while indicating that any depending subtasks are no longer blocked. The number of tasks retrieved at a time by one computing thread may be configured for the particular experiment based on the complexity of its subtask, so that program instances do not block each other and the input and output load on the plan file is not excessive.

If an error occurs within one of the subtasks, all the subtasks not depending on it will continue to be processed, so that the user can correct the error and relaunch only a part of the original plan. The logging feature with configurable verbosity and the possibility to launch a separate scenario parsing simplify the debugging of the experiment setup.

4.4 Integration of Third-Party Libraries

There is a wide variety of freely available NLP tools, such as classifiers or data filters, on the Internet. We decided to profit from some of them, while still keeping our experiment framework relatively independent. Therefore, the individual implementations of various subtasks in our system use the integrated tools, but the task descriptions, the scenario and plan file format and the mechanisms that launch them in the correct order are independent of any third-party tools[5]. It is very simple to use additional Java libraries and programs with our system, since for each tool, only one wrapper Java class which defines the subtask and calls the particular tool with the correct parameters needs to be implemented.

We have chosen to integrate the Waikato Environment for Knowledge Analysis (WEKA, Garner, 1995)[6] with most of the subtasks our system, since it offers a very wide collection of ML classifiers, feature selection algorithms and data filters with a unified API and a transparent default data format — the Attribute-Relation File Format (ARFF), which thus became the default format for our classification tasks.

As one of our classification subtasks makes use of the integer linear programming (ILP) technique (see 7.6), we also included the LP_Solve[7] tool into our system. Although the program is written in the C++ programming language and called via a wrapper library using Java Native Interface (JNI), which makes it less portable among different computer platforms, this problem is bound solely to the single subtask that uses this utility.

[5] Google Collections (`http://code.google.com/p/google-collections/`) and JavaGetOpt (`http://www.urbanophile.com/~arenn/hacking/getopt/`) are the only exceptions from this rule. They are invisible to the programmer who creates additional subtask algorithms, though.

[6] Version 3.7.1, `http://www.cs.waikato.ac.nz/~ml/weka/`

[7] Version 5.5, `http://lpsolve.sourceforge.net/`.

5 Fundaments of Our Deep Analysis System

We will now describe the basic strategy that we decided to pursue in the solution of our SRL problem (Section 5.1), along with a brief explanation of the classification algorithms (from publicly available ML libraries) that we tested in our experiments (Section 5.2). As we already mentioned in Section 3.4, the original data format of the used corpus is not well-suited for the operation of machine learning procedures. In addition, the set of properties for each word in the corpus does not describe the relations of the individual words. Therefore, we also include an account of our implementation of the data conversion (Section 5.3), as well as feature generation and selection (Sections 5.4 and 5.5), all of which is required for proper classifier function and high performance output. Our system uses various generated features described by other authors and several additional, which we originally designed for the purpose of this thesis. The feature selection algorithms comprise our own application of well-known approaches, using ranking algorithms provided by the WEKA framework and our own implementation of the mRMR algorithm. We conclude this chapter with a description of the standard comparison metrics we adopted and programmed for our experiments (Section 5.6).

5.1 Basic Approaches

After careful consideration of the various machine learning solutions available (most of which were already discussed in Chapter 2), we decided to use the traditional supervised classifiers, such as SVM, MaxEnt or logistic regression, in our SRL system due to their competitive performance and simple configuration. The inclusion of many different algorithms in one software package, WEKA, and

the extent of the available sources of information poses also an advantage of this choice.

It follows from the selected machine learning methods that it is now necessary to split the predicate disambiguation (PD) and argument classification (AC) task. The traditional classifiers are only able to infer the values of one target parameter at a time. Additionally, it is only needed to classify the predicates in the PD task, whereas all words could possibly appear as arguments of some of the predicates in the AC problem. The division among argument identification and argument labeling is then possible, but not necessary (please refer to Section 7.2 for this particular topic).

The order in which these two subtasks will be processed is also not given. Although the reports of Bohnet (2009) and Zhao et al. (2009) show that solving the AC problem before the PD problem may lead to very good results, after a preliminary analysis of the frame lexicons provided (see Section 3.2), we decided to take the more usual approach of disambiguating the predicates first, since the nature and number of arguments usually depends very closely on the predicate sense.

5.2 Classifiers

As we already pointed out in Chapters 1 and 2, the maximum entropy (MaxEnt) models (Jelinek, 1997, p. 219ff.) are one of the most used techniques in many different NLP tasks (Manning and Schütze, 2000, p. 607f.) including SRL (Jiang and Ng, 2006; Zhao et al., 2009; Che et al., 2009; Dai et al., 2009). This is mainly thanks to the fact that their base idea of finding a conditional probability distribution with a maximum entropy (average uncertainty Manning and Schütze, 2000, p. 61) subject to given constraints, i.e. "the most uniform model subject to our knowledge" (Berger et al., 1996, p. 41) is very straightforward and effective. The basic form of the conditional probability model (Malouf et al., 2002), which predicts a probability of target event (such as semantic class) $y \in Y$ given a context x, looks as follows:

$$q_\theta(y|x) = \frac{\exp(\theta^T f(x,y))}{\sum_{y' \in Y} \exp(\theta^T f(x,y'))} \tag{5.1}$$

The function f represents an N-dimensional vector of features (see Section 5.4) and θ is the corresponding weight vector. The weights of the model are then trained to satisfy the maximum entropy condition, which is equivalent to minimizing the relative entropy (or Kullback-Leibler Divergence; Manning and Schütze, 2000, p. 72) of the model q_θ to the empirical distribution p observed on the training data (expectation of the individual features' frequencies), or maximizing its a-posteriori probability (conditional probability given the training

data), which is usually expressed in terms of the log-likelihood function:

$$\begin{aligned}\min_\theta D(p||q_\theta) &= \min_\theta \sum_{y,x} p(y,x) \log \frac{p(y|x)}{q_\theta(y|x)} \\ &= \max_\theta L(\theta) = \max_\theta \sum_{y,x} p(y,x) \log q_\theta(y|x)\end{aligned} \tag{5.2}$$

The techniques of fitting the model to the training data usually involve an iterative approach to the zero gradient of the log-likelihood function.

The *logistic regression* (Hosmer and Lemeshow, 2000), another very common classifier type, is a variant of a generalized linear regression model (MacCullagh and Nelder, 1991). The following model form is used to predict the binary[1] target event $y \in \{-1, 1\}$ given a feature vector x and a weight vector θ (Lee et al., 2006):

$$p(y|x,\theta) = \frac{1}{1+\exp(-y\theta^T x)} \tag{5.3}$$

The model is fitted to the training data $x_i, y_i;\ i = 1 \ldots p$ by maximizing its a-posteriori probability — in a way analogous to MaxEnt (Equation 5.2):

$$\max_\theta L(\theta) = \max_\theta \sum_{i=1}^{p} \log \frac{1}{1+\exp(-y_i\theta^T x_i)} = \min_\theta \sum \log(1+\exp(-y_i\theta^T x_i)) \tag{5.4}$$

This optimization problem is also solved using iterative methods (Fan et al., 2008). Blower (2004) provides a proof that the logistic regression is equivalent to MaxEnt (for binary target events)[2]:

$$\begin{aligned}p(y=1|x) &= \frac{\exp(\theta^T f(x,1)}{\sum_{y' \in Y} \exp(\theta^T f(x,y'))} = \frac{\exp(\theta^T f(x,1))}{\exp(\theta^T f(x,1)) + \exp(\theta^T f(x,-1))} \\ &= \frac{1}{1+\exp(-\theta^T f(x,-1))} = \frac{1}{1+\exp(-\theta^T g(x))}\end{aligned} \tag{5.5}$$

This gives us two variants of the same basic idea. Given that logistic regression is integrated into WEKA via the LibLINEAR[3] package (Fan et al., 2008), we decided to test the performance of this very classifier for our experiments.

Another widely used approach to classification is the *support vector machine* (Boser et al., 1992). Its basic idea is representing the (training) data as points in $N+1$-dimensional space[4], whose coordinates are given by the function $\phi(x)$ of the features and by the value of the target event y, and trying to separate the different target events with a hyperplane that has the maximum margin (distance to the nearest point). If we assume that the target event y is binary[5] — $y \in \{1, -1\}$, the

[1] It is, however, technically possible to classify multi-class target event using a binary classifier, e.g. by using multiple classifiers, each predicting one of the classes.

[2] Cf. also the presentation by Pereira at:
`http://www.cis.upenn.edu/~pereira/classes/CIS620/lectures/maxent.pdf`.

[3] Version 1.51, `http://www.csie.ntu.edu.tw/~cjlin/liblinear/`.

[4] This means that each feature, and also the target class corresponds to one dimension.

[5] Most SVM libraries are also capable of solving a problem with multiple target classes.

hyperplanes that are parallel to the maximum margin separator and touch the nearest data points of both target classes (the *support vectors*) are then described by the equations:

$$w^T x + b = 1 \quad \text{and} \quad w^T x + b = -1 \tag{5.6}$$

Maximizing the distance $\frac{2}{||w||}$ leads to minimizing $||w||$ under the circumstances that the hyperplane fits the training data, which yields the following optimization problem (Hsu et al., 2003), given training examples $x_i, y_i;\ i = 1 \dots p$:

$$\min_{w,b} \frac{1}{2} w^T w \quad \text{subject to} \quad y_i(w^T \phi(x_i) + b) \geq 1 \tag{5.7}$$

This may then be transformed using Lagrangian multipliers (Cristianini and Shawe-Taylor, 2000):

$$\min_{w,b} \max_{\alpha} \frac{1}{2} w^T w - \sum_{i=1}^{p} \alpha_i (y_i (w^T \phi(x_i)_b) - 1) \tag{5.8}$$

Now we may solve the equation using quadratic programming and get the solution $w = \sum_{i=1}^{p} \alpha_i y_i \phi(x_i)$, $\sum_{i=1}^{p} \alpha_i y_i = 0$, which finally results in the optimization problem:

$$\max_{\alpha} = \sum_{i=1}^{p} \alpha_i - \frac{1}{2} \sum_{i=1}^{p} y_i y_j \alpha_i \alpha_j (\phi(x_i)^T \phi(x_j)) \quad \text{subject to} \quad \sum_{i=1}^{p} \alpha_i y_i = 0,\ \alpha_i = 0\, \forall i \tag{5.9}$$

The $\phi(x_i)^T \phi(x_j)$ term is called the *kernel* function, with the most usual kernels being linear[6] ($x_i^T x_j$) or radial ($\gamma x_i^T x_j + r)^d$). We chose to test some variants of SVMs in our experiments because of their reported high performance and usage simplicity: many different SVM types are integrated into WEKA through the LibLINEAR and LibSVM[7] (Chang and Lin, 2001) libraries.

Since in most real-life classification problems, it is impossible to find a hyperplane that would separate both target event values for all training examples, a regularization (Neumaier, 1998) term ξ is introduced into the Equation 5.7, which penalizes classification errors, transforming it to the following shape (Cortes and Vapnik, 1995):

$$\min_{w,b} \frac{1}{2} w^T w + C \sum_{i=1}^{p} \xi_i \quad \text{subject to} \quad y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i \tag{5.10}$$

The C parameter is then interpreted as the penalty cost. There are several possible shapes of the regularization function for the SVM, most common being L_1 ($\max(1 - y_i w^T x_i, 0)$) and L_2 ($\max(1 - y_i w^T x_i, 0)^2$) norms, both of them integrated into LibLINEAR or LibSVM (Fan et al., 2008; Chang and Lin, 2001).

[6] In the CoNLL 2009 contest, Che et al. (2009) describe using SVM with this type of kernel.

[7] Version 2.91, `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`.

As correct classification of all the training examples may not be feasible with logistic regression, too, regularization is applied to it as well (Ng, 2004; Fan et al., 2008), which transfers the Equation 5.4 to the following form:

$$\min_{\theta} \sum \log(1 + \exp(-y_i \theta^T x_i)) + R(\theta) \tag{5.11}$$

A function of the weight vector is used as the regularization term — Ng (2004) describe the application of L_1 and L_2 norms of θ, the LibLINEAR version of logistic regression is L_2-regularized.

Since WEKA in the combination of LibLINEAR and LibSVM libraries provides the described high-performance classifiers out-of-the-box, we decided to apply them for our experiments. We also performed some simple preliminary tests with other traditional classifiers implemented in WEKA libraries, such as Decision Trees, Naive Bayes or Perceptron. However, their performance did not indicate that they could possibly surpass the selected ones by a significant margin and the need for other tests and (albeit minimal) reconfiguration would delay our work excessively.

5.3 Data Conversion

In Section 3.4, we described the advantages and drawbacks of the input corpus data format. Since it is unsuitable for use with common ML classifiers, such as SVM or MaxEnt, in its original form, we needed to convert the sentences into the required layout with one instance, i.e. one token relating to one predicate, per line and a constant number of columns. We have designated the ARFF file format directly as the target of our conversion in order to easily use the generated files with WEKA classifiers (cf. Section 4.4).

The conversion of each sentence looks as follows:

1. The sentence is loaded and a unique ID is assigned to it, so that its data is easily recognized during the conversion process.

2. For each predicate in the sentence, the following is done:

   (a) Each pre-selected word of the sentence (see below for data selection options) is taken and all the input CoNLL corpus columns (see Section 3.4), excluding all the `APRED` values, are augmented with more generated features, mainly regarding the context of the word with respect to the predicate (see Section 5.4) to generate one line of output.

   (b) Only the value of `APRED` pertaining to the current predicate is added to the output line[8].

   (c) The output lines for each pre-selected word in the sentence are written into the output file.

[8] At this point, it is also possible to separate the `APRED` values into two groups. For details, see Section 7.3.

This means that if there are e.g. five predicates in the sentence, it appears five times on the output, but always in relation to a different predicate.

The precise character of the output file(s) and depends on the pre-set mode of operation. There are the following three settings regarding output file creation:

1. An output file is created for each predicate lemma, i.e. all sentences with relation to the same predicate lemma are output into one file. The nominal and verbal predicates are also split, e.g. all sentences that contain the verbal predicate "give" fall into the file whose name contains "give.v". This serves the purpose of predicate disambiguation (see Chapter 6).

2. There is an output file for each predicate sense, e.g. "give.01.v". This is used for argument labelling (see Chapter 7).

3. All the data from one input file (e.g. the whole training set) are written into one output file. The predicates to which the data lines relate are indicated in a special column. This mode is also used in the argument labelling process.

The selection of words that appear on the output may be set-up in the four following ways:

1. Each word is selected, nothing is left out. This is used for the argument classification subtask.

2. Only the predicates are selected. We use this setting for predicate disambiguation.

3. A syntactic pruning algorithm (Watanabe et al., 2009; Zhao et al., 2009) is applied, so that only the "syntactic neighborhood" of the predicate is selected. We also tested this mode for predicate disambiguation (see section 7.1).

4. Only tokens with the pre-selected part-of-speech values are selected. This is a somewhat simpler method of pruning, which has also been tested for predicate disambiguation (please refer to Section 7.1 for details).

This gives us a highly configurable tool that provides us with the input of all the various subtasks of our SRL solution.

5.4 Generated Features

We will now turn our attention to the generated features already mentioned in Section 5.3. Since all the classifiers we used regard the training and testing instances (that is words that may belong to the same sentence) as independent, the input set of data columns (see Section 3.4) is insufficient as a feature set for the classification of one word, since it does not provide any information about other words in the sentence, which often co-determine its semantic characteristics.

Therefore, additional features must be introduced that describe the particular word, its syntactical and topological neighborhood and the relations to other words in the sentence.

Our system is able to generate several types of features from the input data. To begin with, there are various qualities of the syntactical neighborhood of the given node and its relation to the predicate. The following types of features have been described by Björkelund et al. (2009), among others:

- *Siblings* — word form, lemma, part-of-speech (POS) and *coarse* POS (CPOS) (first character of the POS-tag; Che et al., 2009) of the nearest syntactical siblings of the given word, if there are some.
- *Dependency relation path* — a string consisting of "/" ("up") and "\" ("down") characters which describes the syntactic path from the given word to the predicate, interleaved with word forms, lemmas, POS, CPOS or the DEPREL labels (see Section 3.4) of the nodes encountered on the path, or left as such.
- *Children* — word forms, lemmas, POSs and CPOSs of the syntactic children of the given word.

There are further syntactic features, already reported by Zeman (2009), Dai et al. (2009) and Watanabe et al. (2009), and others:

- *Syntactic Dependence on Predicate* — the value of this feature is "1", if the given word depends syntactically (even indirectly) on the predicate, "0" otherwise.
- *Sibling or Child* — "1", if the given word is a sibling or a direct child of the predicate, "0" otherwise.
- *Head Position* — the topological position relative to the syntactic head (Before, After).
- *Head* — word form, lemma, POS and CPOS of the syntactic head of the given word.
- *Predicate* — the same properties of the predicate; also in a bigram with the properties of the given word.

In addition to the listed syntactic features, we introduced the following feature type, which is, to our knowledge, new:

- *Children Types* — this lists the word forms, lemmas, POSs and CPOSs and the total number of the children that belong to a word class. We distinguish the following word classes for the sake of this feature:
  - Nominal POSs (nouns, adjectives and pronouns)
  - Verbs (including modal verbs)
  - Open POSs (nouns, adjectives, pronouns and verbs)

- Prepositions
- Particles

We also include the following features that describe the topological neighborhood and morphological features of the given word:

- *Voice* — the voice of the verb (Che et al., 2009), inferred from its POS and the POS of its head node.
- *Left...Right* — morphological features (form, lemma, POS, CPOS) of the first three words to the left and right of the given word, including a bigram for the first two in each direction.
- *Position* — the topological position relative to the predicate (Before, After, On) (Björkelund et al., 2009).
- *Word Distance* — the topological distance from the predicate (absolute value, number of words separating the predicate and the given word) (Watanabe et al., 2009).

The last type of features generated by our system, *Cluster*, is our original concept, based on the technique of *word clustering*(Pereira et al., 1993), which tries to select and group together words that usually appear in similar contexts, thus automatically creating classes of words that share similar semantic properties. We used the SenseClusters[9] tool (Kulkarni and Pedersen, 2005) to generate clusters on word forms and lemmas as they appear in the original corpus sentences, also in combination with POS, thus creating a group of features that involve the different cluster types for topological and syntactical neighbors[10] of the given word.

5.5 Feature Filtering and Selection

Given such a large set of features as described in Section 5.4 (consisting of up to 150 features), it is very probable that some of them will be irrelevant or redundant for the inference of the target class (John et al., 1994), or even introduce noise that increases the classification error (Manning et al., 2008, p. 251). Therefore, we needed to implement some feature filtering and selection techniques.

As to the feature filtering, we implemented and apply these two simple techniques in our system:

1. *Filtering irrelevant attribute values* — for most features with multiple string values, such as Children, Dependency path or Left...Right, we filter all the values that do not occur very often in the training data and unify them under a new single value, "other". This removes all values that would otherwise be ignored by the classifier or introduce noise. This is analogous to frequency-based selection described by Manning et al. (2008, p. 257).

[9] Version 1.01, `http://senseclusters.sourceforge.net/`.

[10] In particular, they are analogous to the Left...Right, Dependency path, Head, Siblings and Children features.

2. *Filtering irrelevant attributes* — we filter all the attributes that are, in the given case, obviously irrelevant, i.e. they either do not contain any value which would repeat in the training data, or are constant across the whole training set.

In addition to filtering, one must select a subset of features that yields the optimal performance. For this purpose, the usual approach is to train the classifier iteratively for different feature subset and evaluate its performance on the *development* data set (see Section 3.4), thus obtaining the best performing subset. Since it is not computationally feasible to try every possible combination of features, we have to apply faster algorithms that may produce a suboptimal solution. We employ our own implementation of two different basic techniques in our SRL setup, applying parallelization:

1. *Feature ranking* — the features are ordered according to a given criterion (see below) and the N best are selected (Manning et al., 2008, p. 251ff.). In our system, we usually gradually increase the N in a certain range and then select the best value. All trials for different N may be launched in parallel.

2. *Greedy algorithm* — this iterative algorithm keeps a set of features and tries to add all the remaining features in each step, then selects the best one (Caruana and Freitag, 1994). In our setting, we either start with one feature, or try all possible combinations in a small subset of features (with the best ranking). Of equally performing features, we always select the one that has been rated as the best one in the ranking. Each round trying to add all remaining features may be processed in parallel.

Our system contains the following feature ranking criteria:

- χ^2-*Criterion* (Manning et al., 2008, p. 255) — this evaluates the features using the χ^2-statistics against the target feature, which is computed as $\sum_{f \in F} \sum_{t \in T} \frac{(N_{ft} - E_{ft})^2}{E_{ft}}$, where f, t are all possible values of the feature F and the target T, respectively, and E, N are the expected and observed frequencies of the values occurring together.

- *Mutual Information* (MI) (Manning and Schütze, 2000, p. 583), or *Information Gain* (Manning et al., 2008, p. 264) — this computes the difference of the entropy of the target feature alone and the conditional entropy of the target feature given the ranked feature, that is how much the knowledge of this feature lower the entropy of the target one.

- *Symmetric Uncertainty* (Witten and Frank, 2005, p. 291f.) — this statistics is defined as $\frac{2I(F,T)}{H(F)+H(T)}$, where H is the entropy, I is the MI and F, T stand for the evaluated and target feature, respectively.

- *ReliefF* (Kira and Rendell, 1992; Kononenko, 1994) — this is an algorithm that ranks the features depending on how well they distinguish between the values of the target event: In a pre-set number of iterations, it selects

random instances and then searches for the "nearest hit" (nearest instance, based on the value of the ranked feature, that has the same target event class) and "nearest misses" with different classes. The worth of the feature is then a sum of distance differences between the hits and misses weighted by the probability of possible target feature values.

- *Significance* (Ahmad and Dey, 2005) — this algorithm is based on a similar assumption to the previous one, namely that a "valuable" feature should differentiate between the values of the target event. Given the training data statistics, it computes the average probability of the most probable set of target classes (the "support set") for each value of the evaluated feature, then a symmetric value describing the most probable sets of the evaluated feature values for each target class. The average of the two values is then returned as the final rank.

- *Minimum Redundancy-Maximum Relevance* (mRMR) (Peng et al., 2005) — using the concept of MI, this technique maintains a set of features, starting with an empty set, and greedily selects a feature that is most relevant to the target event, but not too redundant with respect to the already selected features. This feature is a solution to the following optimization problem (where S_{m-1} is the currently selected set of features):

$$\max_{F_j \notin S_{m-1}} \left[I(F_j, T) - \frac{1}{m-1} \sum_{F_i \in S_{m-1}} I(F_i, F_j) \right] \tag{5.12}$$

All the criteria but mRMR were integrated into the WEKA framework; we implemented the mRMR ranking algorithm (including our own implementational variant of MI, which we also use in the rankings) for the purpose of this work. We also made performed tests with averaging the rankings — either all the rankings or only mRMR and ReliefF, following the ideas of Zhang et al. (2008) in a simplified way, so that the average is less prone to possible failures of one of the ranking algorithms.

Our decision to apply or combine the various rankings or to employ the greedy algorithm in the individual subtasks in our system (see Chapters 6 and 7) depends on the complexity and data volume in the particular case and is a trade-off between computation speed and final classifier performance.

Moreover, we always convert the multiple-valued string features to a set of binary features, each indicating the presence of one possible value, as the classifiers used in our experiments interpret all features as numeric[11]. The values of numeric attributes are then normalized (Witten and Frank, 2005, p. 56f.) to lie within the $[0, 1]$ interval, so that the classifiers may use them as a metric.

[11] We chose to perform the binarization only after feature selection, since this reduces the computation time needed. Selecting the binarized features would probably improve the performance, but also retard the process on the other hand.

5.6 Measuring the Classifier Performance

In order to compare the different subtask configurations of our system, we adopted the standard metrics used in NLP tasks (Manning and Schütze, 2000; Manning et al., 2008; Witten and Frank, 2005, p. 268ff., p. 142ff., p. 171ff., respectively) — *accuracy*, *precision and recall* and *F-measure* (F_1)[12].

Accuracy is the most straightforward one of them; it is computed as the percentage of correctly classified (*true*) data instances.

$$A = \frac{t}{t+f} \tag{5.13}$$

In the equation, t stands for true examples and f for *false* (wrongly classified) instances. It is obvious that this measure can be used for multi-class problems, such as the PD problem (see Chapter 6 for its application in our system).

The other measures of comparison are based on the concept of *negative* and *positive* data instances. It relates to a binary classification problem situation where a part of the data is relevant and needs to be retrieved. Such data examples are called positive, whereas the rest of the input is referred to as negative. Precision and recall lay a greater focus on the positive examples, since their number is usually much smaller than that of the negative ones, which could lead to all instances being classified as negative if the ultimate goal was to maximize accuracy.

The formulas for computing precision and recall are defined in the following way (tp, tn being true positive and negative instances and fp, fn false positive and negative instances, respectively):

$$P = \frac{tp}{tp+fp}, \quad R = \frac{tp}{tp+fn} \tag{5.14}$$

Maximizing precision thus means improving the system, so that it does not retrieve irrelevant data, while maximizing recall means striving to retrieve as much relevant data as possible, regardless of false positives. In many situations including SRL, it is important for the optimum performance of a solution that both precision and recall stay high. Therefore, an additional metrics has been introduced, which makes a compromise between them — the F-measure. The most usual variant, F_1, which weighs precision and recall equally, computes as the harmonic mean of the two values:

$$F_1 = \frac{1}{\frac{1}{P}+\frac{1}{R}} = \frac{2PR}{P+R} \tag{5.15}$$

In the CoNLL 2009 contest, a *labeled* variant of precision, recall and F_1 (Hajič et al., 2009) was used in addition to the standard versions to compare the results of the competing SRL systems. It uses the same notion of positive and negative examples, while effectively counting correctly retrieved, but wrongly labeled data instances as both false positive and false negative. We adopted this approach

[12] We implemented a simple evaluation subtask that is able to compute all four of them.

for our AC system (see Chapter 7), since it both measures the correctness of the semantic labeling and focuses on the positive instances, i.e. semantic arguments.

For the total evaluation of the whole system, we used the original CoNLL 2009 scorer[13], which computes labeled and unlabeled precision, recall and F_1 scores, while combining predicate senses with semantic arguments.

We also needed to measure the relevance of a performance difference between two variants of a subtask configuration in terms of one of the above metrics. For this purpose, we adopted and implemented the *bootstrap* technique (Efron, 1979; Witten and Frank, 2005, p. 152), which is very fast to compute and renders reliable results. We chose not to use other commonly applied approach, the *cross-validation* (Hastie et al., 2009, p. 241ff.), due to its higher computational requirements. The bootstrap is calculated in the following way: By iteratively selecting random subsets of the original output and determining the classifier performance on these subsets, we obtain an estimate of the probability distribution for the classifier output, whose percentiles give the desired confidence interval (90 % is used in this thesis). In conclusion, if the confidence intervals for two classifier variants do not overlap, one of them performs significantly better.

[13] `http://ufal.mff.cuni.cz/conll2009-st/scorer.html`

6 Predicate Disambiguation

This chapter contains the technical description of our own implementation of the predicate disambiguation, the first part of our SRL system. We have split the PD problem into several parts, according to the nature of the particular predicates (see Chapter 6.1), and applied various ML techniques described in Chapter 5. An account of our approach to classifier parameter tuning (using the third-party classifier libraries selected in Section 5.2) is given in Section 6.2, followed by the feature selection techniques (employing our own implementations of well-known algorithms, together with library-supplied feature rankings) in Section 6.3. We will then include an overview of the whole PD solution (Section 6.4).

6.1 Observations on the Data

The first and most significant observation we made on the data is that as all predicate labels consist of a lemma and a sense number, it is evident that predicate senses for different lemmas are completely independent of each other. Further, since the numberings for nouns and verbs overlap, nominal and verbal predicates must be treated as separate problems. This implies that splitting the data according to the lemma and training a separate classifier for each lemma is profitable, since this prevents the machine learning algorithm from selecting an incorrect lemma at all times, even with rare predicates. We therefore adopted this practice, similarly to Che et al. (2009) and Björkelund et al. (2009). Our data-conversion algorithm (introduced in Section 5.3) thus produced only the predicate words as the input for the entire PD task, split into individual files according to the predicate lemma[1].

[1] We will henceforth consider predicate lemmas to be distinct for nouns and verbs, which is technically insured by the conversion process, appending the POS indication after the predicate lemma (e.g. "progress.v").

As we already mentioned in Section 3.4 and depicted in Table 3.2, a very broad range of predicates occurs in the CoNLL 2009 corpus, with a small subset of them appearing only in one or two of the data sets. Additionally, Table 6.1 shows the numbers of predicate lemmas and their coverage. We must therefore distinguish among the following cases:

1. There are training and development instances of the given predicate lemma. This is by far the most common case, not only regarding the number of the unique lemmas, but mainly with respect to the number of instances. We are able to tune the classifier setting on the development set for such predicates.

2. Only training instances are available.
   In that case it is possible to train the classifier, but no tuning can be done.

3. There are no training data for the particular predicate lemma.
   With no training data, only a dummy classifier that assigns always the same value is possible.

This division does not take the evaluation data into consideration, apart from the fact that it is ready for previously unseen predicates. We can solve the individual cases and prepare the best possible classifier setting, which is then used only for those predicates that actually occur in the evaluation data.

Table 6.1: Predicate lemma statistics for the CoNLL 2009 corpus

	Predicates	**Lemmas**	**Coverage (Lemmas)**
Training	9228	7639	-
Development	2151	1946	95.5 %
Evaluation	2610	2337	96.2 %

The values shown are the total numbers of distinct predicates and distinct predicate lemmas, followed by the percentage of predicate lemmas in the development and evaluation sets covered by the training set.

There is another distinction that must be done prior to the actual classifier training, namely based on the number of senses for each predicate lemma. As is evident from Table 6.2, a vast majority of the lemmas has a unique sense (we will refer to them as *unary*). In total, a larger part of all predicate occurrences, although not as predominant, falls within this group. This renders the classification of those lemmas trivial; we will therefore always assign the only observed sense to them.

However, still a large number of predicate lemmas are not unary. We have divided them into three groups, which we treat differently. Most lemmas sport only two or three senses; we denote such lemmas *simple*, for their classification problems are not very complicated (as the number of instances is also not very high on average). This makes the classifier tuning less important and shifts the stress on the speed of the solution, so that all of the lemmas can be processed within a reasonable amount of time.

Table 6.2: Number of senses for the individual predicate lemmas as observed in the training data set

Senses	Predicate Lemmas			Number of occurrences		
	Total	%	Group	Total	%	Group
1	6541	85.6	6541	112994	63.1	112994
2	773	10.1	961	27395	15.3	41801
3	188	2.4		14406	8.0	
4	66	0.8	120	7726	4.3	16291
5	24	0.3		3571	2.0	
6	13	0.2		1745	1.0	
7	13	0.2		2522	1.4	
8	4	0.0		727	0.4	
9	3	0.0	17	883	0.5	7928
10	3	0.0		929	0.5	
11	3	0.0		882	0.5	
12	2	0.0		1638	0.9	
13	1	0.0		141	0.0	
14	2	0.0		1105	0.6	
15	-	-		-	-	
16	-	-		-	-	
17	1	0.0		585	0.3	
18	1	0.0		809	0.5	
19	-	-		-	-	
20	1	0.0		956	0.5	

The Group column contains the total numbers of lemmas assigned to the individual groups (unary, simple, harder, tough).

Another, smaller group of predicate lemmas subsumes a greater number of senses (four to eight, labeled *harder*). There are still considerably many such lemmas — therefore, the adjustment of the classifier must be more sophisticated, but still fast enough, so that our system can cope with many lemmas of this kind.

The last, but important group of lemmas is highly ambiguous (we will refer to them as *tough*) — in most cases it is functional verbs and verbs that serve as the base element of multiple phrasal verbs, such as "go", "take", "get" etc. Only very few nominal lemmas belong to this group, all of them representing very abstract and vague terms — "line", "way", "place", "hand". We believe that for such a small subset, a more time-consuming, but very accurate tuning algorithm should be applied.

6.2 Selecting the Classifier Setting

We have decided to use the same ML classifier with a constant set of parameters for the whole task, because selecting the best algorithm and configuration for each predicate lemma would complicate the training process and increase its duration excessively. Therefore, we needed to obtain a classifier and a set of parameters with the highest performance in an average case. We thus created an experiment that would test several classifier types from the LibLINEAR and

LibSVM packages (as indicated in Section 5.2) with many different settings and compare them to each other. We used *accuracy* as the measure of comparison for the individual configurations, since the PD task is a multi-class problem with no well-defined "positive" and "negative" data instances, which are needed for all other standard metrics described in Section 5.6.

Since the size of the input corpus is too great to test each pre-selected algorithm and configuration on the whole development set, we created a smaller subset of both training and development data for our experiments (referred to as *PD Experimental Subset*), selecting 100 predicate lemmas with the biggest development data sets. Since most of the plentifully represented predicates are also more ambiguous than the average, such a selection shows convenient for finding a configuration that would suit best to the harder and tough lemmas, where the parameter tuning is of highest importance.

We planned to apply feature selection to the training algorithms and hence took it into account for this experiment as well[2]: Our system used the filtering and all the feature rankings discussed in Section 5.5 and tried to find the optimal number of features in the range[3] $4-30$ by consecutively trying to add the next one in the order given by the particular ranking, then selected the best result of the individual rankings[4]. We then averaged the results of each classifier setting for all the lemmas in our subset and established an estimate of confidence intervals using the bootstrap method (see Section 5.6).

We discovered in our preliminary experiments with no feature selection that if the regularization penalty cost parameter is set higher than 10, the performance of all classifiers is lowered and training duration becomes much longer. Additionally, lower termination criterion for the iterative fitting step of LibLINEAR classifiers than 0.001 does not change the algorithm output. We have therefore pre-selected the parameter values accordingly and examined the following classifier setups:

- L_2-regularized logistic regression from the LibLINEAR package, with the penalty cost parameter $C \in \{0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10\}$ and the termination criterion $E = \{0.001, 0.01, 0.1, 1\}$.

- L_2-SVM primal and dual problem, L_1-SVM dual problem and multi-class SVM from the LibLINEAR package, with the same penalty cost and termination criterion parameters as previous.

- C-SVM with a linear kernel from the LibSVM package, with the penalty cost parameter $C \in \{0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10\}$.

- C-SVM with a radial basis kernel from the LibSVM package, with the same regularization cost and the parameter $\gamma \in \{0.001, 0.01, 0.1, 1\}$.

[2]Feature filtering and selection preliminaries are described in detail at the beginning of Section 6.3.

[3]If less features remain after the filtering, all of them are examined in the same fashion. This also applies to the following feature selection variants.

[4]We apply the same feature selection technique to harder lemmas in our PD system. Cf. Section 6.3 for details.

Table 6.3: Classifier performance on the PD Experimental Subset with feature ranking

Software	Classifier	Parameters	Acc.	5 %	95 %
LibLINEAR	L_2-Log. Regression	C = 2; E = 0.001	**92.9**	91.3	94.5
		C = 5, 10; E = 0.001	92.8	91.1	94.5
		C = 1; E = 0.001	92.7	91.0	94.4
		C = 5; E = 0.01	92.7	91.0	94.4
	Multiclass SVM	C = 2; E = 0.001, 0.01, 0.1	**92.9**	91.3	94.5
		C = 0.05; E = 1	92.8	91.0	94.4
		C = 0.1; E = 0.001, 1	92.8	91.2	94.2
		C = 2; E = 1	92.8	91.3	94.4
		C = 5; E = *	92.8	91.1	94.6
	L_1-SVM (Dual)	C = 2; E = 0.001, 0.01, 0.1	**92.9**	91.2	94.4
		C = 0.05; E = 1	92.8	91.2	94.4
		C = 0.1; E = 0.001, 1	92.8	91.1	94.4
		C = 2; E = 1	92.8	91.1	94.5
		C = 5; E = *	92.8	91.1	94.4
	L_2-SVM (Dual)	C = 2; E = 0.001, 0.01, 0.1	**92.9**	91.1	94.4
		C = 0.05; E = 1	92.8	91.2	94.4
		C = 0.1; E = 0.001, 1	92.8	91.2	94.5
		C = 2; E = 1	92.8	91.1	94.4
		C = 5; E = *	92.8	91.1	94.5
	L_2-SVM (Primal)	C = 2; E = 0.001, 0.01, 0.1	**92.9**	91.1	94.5
		C = 0.05; E = 1	92.8	91.2	94.4
		C = 0.1; E = 0.001, 1	92.8	91.2	94.4
		C = 2; E = 1	92.8	91.2	94.5
		C = 5; E = *	92.8	91.1	94.4
LibSVM	Linear Kernel	C = 2, 5	**93.1**	91.4	94.6
		C = 1, 10	92.8	91.2	94.5
	Radial Kernel	C = 2, 5; γ = *	**93.1**	91.5	94.6
		C = 1, 10; γ = *	92.8	91.1	94.4

The listed values are: accuracy percentage on the whole experiment set, bootstrap 5 %-percentile and bootstrap 95 %-percentile. Only up to five best performing configurations for each classifier are listed; if the performance on the whole set is identical, the figures are merged and only the outermost bootstrap values are shown. An asterisk ("*") in the parameters specifications signifies that any of the tested values is possible.

The list of the best-performing classifier algorithms and parameter settings, along with their results, is given in Table 6.3. It is now apparent that the various classifiers perform very similarly; the results of the individual SVM variants from the LibLINEAR package are almost identical. The confidence intervals estimated by bootstrap overlap for all the algorithms, and thus higher accuracy on the experiment data set cannot be considered a decisive criterion for selecting one of them.

However, the results of the test also show that all of the classifiers are relatively sensitive to parameter settings. The logistic regression stays under 87 % accuracy with the termination criterion set at 1, independent of the cost value; however, its results remained at the top level as long as this this variable was set at 0.01 or lower. Both LibSVM classifiers reach only approximately 90 % accuracy with inadequate configuration. The LibLINEAR SVM methods are less prone to misconfiguration, their worst outputs rate at 91.8 %. It is also apparent from Table 6.3 that the regularization penalty C set at 2 proves to be optimal for all the machine learning techniques, albeit by a very narrow margin.

Table 6.4: Classifier performance on the PD Experimental Subset with no feature selection

Software	**Classifier**	**Parameters**	**Acc.**	**5 %**	**95 %**
LibLINEAR	L_2-Log. Regression	C = 0.1; E = 0.1	**91.5**	89.5	93.1
		C = 0.1, 0.5; E = 0.001	91.4	89.7	93.1
		C = 0.1; E = 0.01	91.4	89.7	93.1
	Multiclass SVM	C = 0.05; E = 0.1, 0.01, 0.001	**91.1**	89.3	92.8
		C = 0.1; E = 0.01	90.6	88.9	92.4
	L_1-SVM (Dual)	C = 0.05; E = 1	**91.5**	89.8	93.3
		C = 0.05; E = 0.001	91.4	89.5	93.2
	L_2-SVM (Dual)	C = 0.05; E = 1	**91.2**	89.4	93.0
		C = 0.05; E = 0.1,0.01,0.001	91.1	89.3	92.8
	L_2-SVM (Primal)	C = 0.05; E = 0.001	**91.4**	89.5	93.1
		C = 0.05; E = 0.01	91.2	89.4	92.8
LibSVM	Linear Kernel	C = 0.1, 0.2	**86.7**	84.5	88.9
		C = 1, 2	86.3	84.2	88.4
	Radial Kernel	C = 2; γ = 0.001	**89.1**	87.2	91.1
		C = 5, 10; γ = 0.001	88.9	86.9	90.8

The format of this table is the same as in Table 6.3. Only a maximum of three best settings is listed.

In order to test classifier sensitivity to irrelevant features, we performed a variant of the above experiment with the same software and configuration, but no feature selection, i.e. all the filtered generated features were present in the classifier inputs. Table 6.4 lists the results of this trial, which showed that the performance of the LibSVM classifiers depends very strongly on the relevance of the input features. All the LibLINEAR methods proved superior, with the results of logistic regression performing slightly better than the SVM algorithms. In addition, the results of all LibLINEAR models and the LibSVM linear kernel SVM for different parameters do not vary nearly as much as with feature selection

(the highest interval between the best and the worst result was 1.1 % with L_1-SVM). The LibSVM radial kernel SVM is very sensitive to the parameter values — their proper setting yields as much as 5 accuracy percent points difference.

As a result of this experiment, we have chosen to perform the PD task classification with the L_2-regularized logistic regression from the LibLINEAR package, for it reaches very high performance with most configurations, provided the termination criterion parameter is low enough. Therefore, this variable was set at 0.001 for all subtasks, and the penalty cost remained at 2.

6.3 Feature Selection Approaches

Having selected the classifier settings, we focused on perfecting the feature selection techniques for the individual groups of predicate lemmas identified in Section 6.1. In our preliminary experiments on small data sets, we found it profitable to filter out all feature values that occur in less than 1 % of training instances and less than three times, we therefore use the filtering methods as described in Section 5.5 in this particular configuration. We also learned that using more than about 30 features is not likely to improve the classifier performance, as for every individual lemma, many features are irrelevant. It also must be noted that with the filtering techniques applied before feature selection, it is possible that a set of even less than 30 features remains and the rest is omitted, since it does not promise an asset to the system. Additionally, the binarization that is required before classification (cf. Section 5.5) raises the total number of features that occupy the machine learning algorithm up to thousands.

We decided to use feature ranking with the search for the appropriate number of features with the simple lemmas, since this method is very fast and yields relatively good results. However, we needed to select a ranking algorithm from the ones listed in Section 5.5 that would give the best results on average. Therefore, we designed another experiment on our PD subset from Section 6.2, which compares the results of the same classifier configuration when coupled with varying feature rankings. We used a logistic regression model with the settings described in Section 6.2 and measured its accuracy on our experiment set with all feature rankings, each time with the best number of features possible within the range $4 - 30$ (obtained by consecutively adding features to the subset of used ones in the order given by the ranking), and estimated the confidence intervals using the bootstrap method.

The results of this experiments are listed in Table 6.5. We can see that all the results are comparable, with mRMR, ReliefF and their average on the top, but only by a very narrow margin. The bootstrap estimates indicate that no ranking algorithm performs significantly better than the others. Nevertheless, we decided to use the mRMR-ReliefF average as a feature-ranking algorithm for the simple predicate lemmas since its results appear slightly better than that of the other rankings. Similarly to the previous experiments with ranking, our system searches for the best feature subset by consecutively adding features in the order given by the ranking, thus selecting a subset of $4 - 30$ features that is saved for later use on evaluation data.

Table 6.5: Results of the ranking test experiments on the PD Experimental Subset

Ranking	**Acc.**	**5 %**	**95 %**
χ^2	92.8	91.1	94.4
MI/Information Gain	92.8	91.1	94.2
MI (mRMR version)	92.8	91.1	94.2
Symmetric Uncertainty	93.3	91.7	94.9
ReliefF	**93.6**	92.0	95.0
Significance	92.8	91.3	94.5
mRMR	93.4	91.9	94.7
Average (all)	93.3	91.9	94.9
Average (mRMR + ReliefF)	**93.6**	92.0	95.2

Similarly to Tables 6.3 and 6.4, the listed values are: accuracy percentage on the whole experiment set, bootstrap 5 %-percentile and bootstrap 95 %-percentile.

With the harder predicate lemmas, more stress must be laid on the quality of the used feature subset. Therefore, we decided to employ all our ranking algorithms in combination with the consecutive adding approach and select the ranking and feature subset that gives the best results on the development data set, thus exploring a greater number of feature subsets, but still processing the individual predicate lemmas much faster than with the use of greedy selection. This is the same technique that was used for our classifier parameter selection test in Section 6.2.

We also apply ranking to predicate lemmas with multiple senses which do not occur in the development corpus. Since it is not possible to select the best performing number of features with no data examples to test the differences, we always take the 30 top-ranked ones.

We decided to use the greedy feature selection algorithm (see Section 5.5) with the tough lemmas, since this procedure, albeit lengthy, promises better results than the above ones. With such a small number of lemmas in this group as 17 and several machines computing in parallel, the processing time was not excessive. Our preliminary experiments showed an improvement if we first applied the following two different variants of the greedy selection and then selected the more successful one:

1. The system starts with no features selected and selects one at a time. From several equally performing features, it always selects the one with the higher mRMR-ReliefF ranking average. It finishes if all not yet used features degrade the performance, or if the size of the feature set reaches 30.

2. The system starts examining all the possible groups of four features out of the top ten mRMR-ReliefF ranked ones. Then it continues adding one feature at a time in the same way as above.

This will lead to very plausible feature subsets as the output of PD training, which are saved for later use in PD evaluation.

6.4 The Predicate Disambiguation System

We will now give an account of the whole organization of our PD system, which is split into the training phase and evaluation phase[5]. The former serves for obtaining the best classifier settings, i.e. feature sets, which are then used as the input to the latter. The training part then consist of the following steps (for a detailed schema, please refer to Figure 6.1):

1. Conversion from the CoNLL 2009 corpus format to the ARFF data format (as described in Section 5.3), with the generation of new features (see Section 5.4). Individual data files are created for all predicate lemmas.

2. Separation of files that do not have any development data instances (referred to as *orphans*), selection of lemmas with multiple senses, then their filtering and ranking, which results in the list of best features. Filtering settings and the selected feature set are stored for later use.

3. Filtering of the lemmas with development instances (for details on filtering, see Section 5.5). List of the attributes and attribute values that are to be filtered is saved for application in the evaluation phase.

4. The predicate lemmas are divided into groups according to number of senses — unary, simple, harder and tough. The first group is no longer needed for training.

5. The most favorable feature set is selected for each predicate lemma (refer to Section 6.3 for a thorough description):

   (a) Features for the simple lemmas are ranked using the mRMR and ReliefF average, the optimal number of top ranked features (between $4 - 30$) is then determined by subsequent addition trials.

   (b) The same technique applies to harder lemmas as well, except for the fact that multiple rankings and selections are performed, which yields several candidates for the best feature set. The system then picks the best-performing candidate.

   (c) The tough predicate lemmas have their features ranked by the mRMR-ReliefF averaging algorithm as well. The features list is then used as an input into two different setups of the greedy feature selection algorithm, which produce candidate feature sets. The better one is chosen in the end.

6. All the resulting feature sets are saved, so that they may be applied in the evaluation phase.

The ML models examined in the training stage are not stored in order to reduce disk usage — we train the classifiers anew in the evaluation phase. Ready

[5]The tasks description used in our batch processing framework is provided on the enclosed CD (see Appendix B).

Figure 6.1: A chart of our PD system, training phase

The computation steps are depicted as colored rectangles, the conversion and filtering subtasks are represented by ovals. Grey diamonds stand for selection tasks and finally, white rectangles with black borders contain the outputs of this phase.

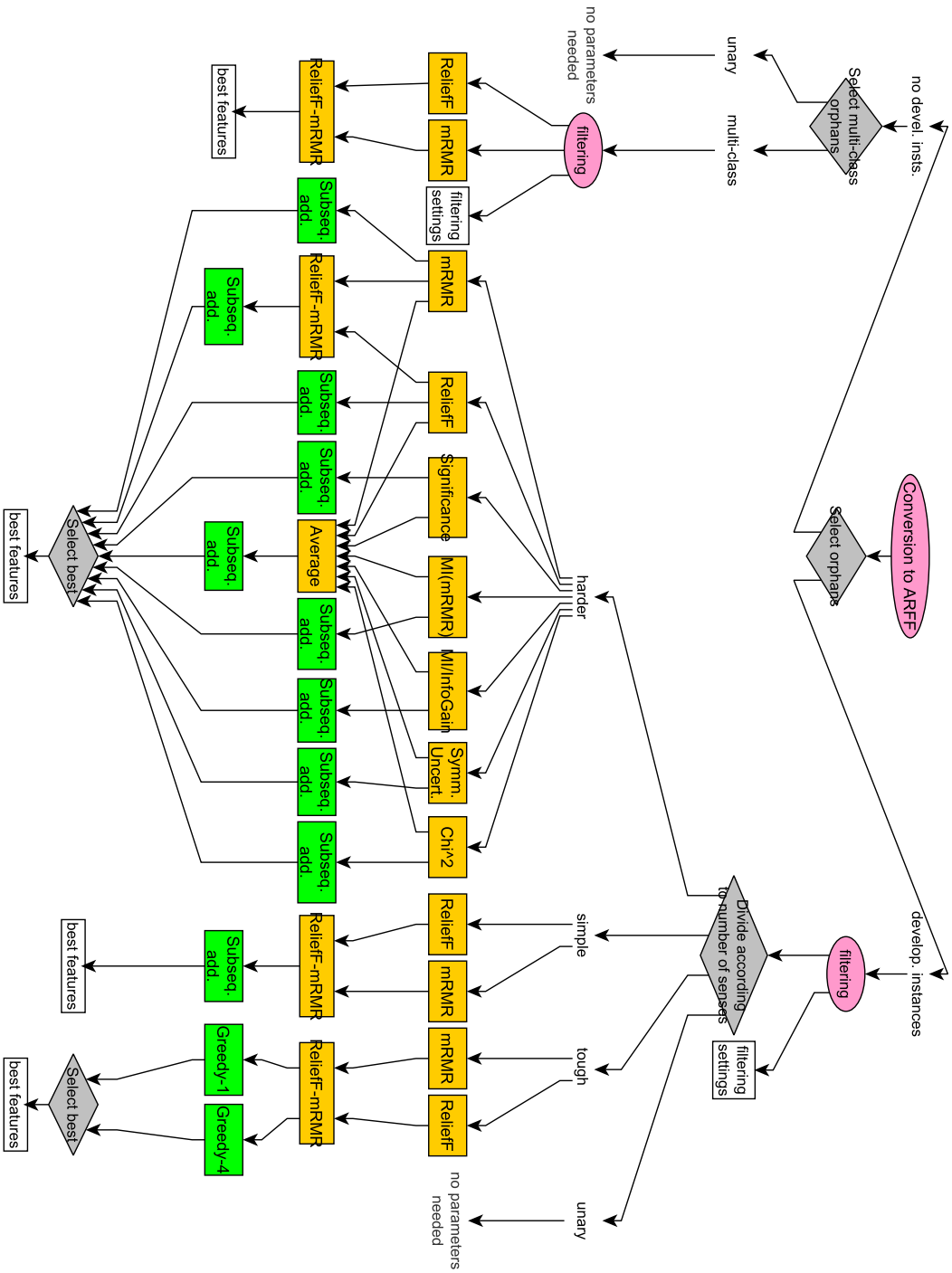
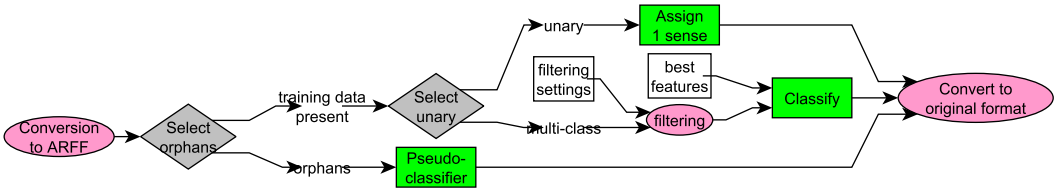


Figure 6.2: A chart of our PD system, evaluation phase
This schema has the same form as the one in Figure 6.1.



trained models could certainly be stored and used for evaluation as well, but since the training of a single model is not very time-demanding given the already preselected set of features, we decided not to save all the models examined during the training stage in order to save disk space. We thus have the best feature sets and the filtering settings as the input to the evaluation stage, which then proceeds as follows:

1. Data conversion, same as in the training phase.
2. Separation of lemmas that do not have any training data instances (also denoted *orphans*). We use a dummy classifier for these lemmas, which takes the lemma and appends ".01" to it, thus creating the usual representation of the first and most common sense.
3. We select the predicate lemmas with multiple senses; the others are assigned the only sense observed on the training data.
4. The system now filters the predicates according to the settings saved from the training stage and classifies them using appropriate the sets of viable features.
5. The results of the classification, as well as the two kinds of pseudo-classification, are collected back to the original file format.

A chart depicting the data flow in the evaluation stage is shown in Figure 6.2. The PD results are then used as one of the inputs for the second part of our system, the argument classification.

7 Argument Classification

We will explain the technical properties of our own AC solution in this chapter. Within our implementation of the data conversion from the original corpus format, we conducted several experiments with selecting only the most viable argument candidates, including syntactical approaches by Zhao et al. (2009) and Watanabe et al. (2009) and our own POS-based method, which are all described in Section 7.1. We have also examined the merits of splitting the AC solution into the identification (AI) and labeling parts or taking the joint approach, both of which featured in the CoNLL 2009 contest SRL systems (see Section 7.2). In addition, based on our linguistic knowledge of English, we originally propose a different approach to adverbial modifiers and co-references in opposition to valency arguments. An account of our tests regarding this topic is given in Sections 7.3 and 7.7.

Similarly to the PD approaches described in Sections 6.2 and 6.3, we searched for the optimal classifier configuration and feature selection techniques, which we describe in Section 7.4. Since a greater part of the predicates occurs extremely rarely, we designed a new method of dividing the independent data instances while still maintaining enough training examples (cf. Section 7.5). Our system also includes a post-inference on the classifier output (ILP as reported by Punyakanok et al. (2004) and our own, simpler approach), in order to adhere to global constraints for whole sentences (see Section 7.6). The final design of our AC system is then summarized in Section 7.8.

7.1 Argument Candidates

The first decision that we needed to make in our PD system concerned the format of the data conversion output. As we described in Section 5.3, our conversion algorithm either separates data relating to different predicates (or lemmas) into individual files, or outputs the whole data set into one large file. We considered

either feeding all the data to a single classifier, which would mix different predicate frames together, or using a different classification model for each predicate, which would separate data instances relating to distinct predicate frames, but thin the amount of training instances available considerably. Separating predicate lemmas includes both of the above drawbacks, therefore we eliminated this choice.

We created an experiment that would test the merits of both approaches. In order to reduce computation complexity, we created a smaller subset of the data (referred to as *AC Experimental Subset*), which we used for our experiments. We selected 100 predicates at random from training and development data sets, thus including data relating to frequent predicate senses as well as that of the rare ones and maintaining representativity of the whole data. We then trained the same classifier in the same configuration (L_2-regularized logistic regression from the LibLINEAR package with the cost parameter set to 1 and a termination criterion of 0.001, using all features filtered in the same way as in the PD experiments described in Section 6.3) to classify the following joint AC multi-class problems[1]:

1. On the data split for different predicates, creating a separate model for each one of them.

2. As a single model on the whole AC Experimental Subset.

We chose to use labeled F-measure as the means of comparing the different outputs (see Section 5.6 for explanation). The results of this experiment are shown in Table 7.1. It is apparent that splitting the data for different predicates yields slightly better results, although the difference is not significant in terms of the bootstrap 90 % estimates (please refer also to Section 5.6 for a description of bootstrap). Nevertheless, we decided to split the data for the individual predicates, since it offers much wider space for parallelization and the sparse data problem can be handled in a different way (an account of our solution will be given in Section 7.5).

Table 7.1: Splitting the predicates vs. using one classifier

Setting	F_1	**5 %**	**95 %**
Using one classifier on the whole subset	70.1	65.8	74.5
Separate classifiers for each predicate	71.4	67.2	75.3

The first value shown is the labeled F_1 percentage, followed by 5 % and 95 % percentiles of the bootstrap estimate.

Since the amount of data to be processed is very large if the conversion system produces all the instances (see Sections 3.4 and 5.3), we also implemented and tested two different pruning algorithms for argument candidates in a preliminary experiment.

[1]The decision to use a single AC task instead of splitting it into identification and labeling subproblems has been made for the sake of simplicity here, since the main goal was to compare the effects of splitting data in the same settings. We will review this option in Section 7.2.

The first one of them, described by Watanabe et al. (2009) and Zhao et al. (2009), is based on syntactic dependencies. It selects only the words that depend on one of the nodes syntactically superordinate to the predicate (i.e. on the dependency path from the predicate to the syntactic root of the sentence; including the predicate itself). Watanabe et al. state that this approach keeps 97.3 % of the arguments in the candidate set, while reducing the data amount by 63.1 % for English.

Observing our PD Experimental training subset, we discovered that some of the POSs, mainly punctuation[2], occur only seldom as arguments. Therefore, we developed an algorithm that selects instances based on their POS. This is a less aggressive approach in comparison to the previous one, as it leads to much smaller data reduction, but is still valid.

In order to compare these methods, we applied them to our set of predicates using the same classifier setting as above (with the data split for individual predicates) and evaluated their results using the CoNLL 2009 scorer[3] and labeled F_1 (see Section 5.6). We did not use our own implementation of the metrics, since it depends on the total number of instances, which differs across the individual settings. We therefore wrote the classification results back to the original format, using only the part of development data where our selected predicates appear, preserving the golden values for predicate senses and other predicates' arguments. Although the values itself do not tell anything about the total system performance, they still provide an insight into the merits of the particular methods.

Table 7.2: Application of pruning algorithms

Setting	F_1
No pruning	88.48
Syntactic pruning	87.97
POS filtering	88.39
Syntactic + POS pruning	87.75

The value shown is the labeled F_1 percentage as computed by the CoNLL 2009 Scorer, using a set of development data sentences where the selected predicates appear, including golden predicate senses and arguments of other predicates.

As is apparent from Table 7.1, none of the pruning techniques improves the overall results. In addition to that, we expected the differences to expand if classifier output for all predicates is used, which would affect the performance even more negatively. Therefore, we decided not to use any pruning technique in our final AC system, even if the computation speed is affected.

[2]The filtered POSs are: `” ( ) , . : “ EX HYPH LS NIL POS`.

[3]As the scorer does not output confidence intervals estimation, we do not include it for this experiment.

7.2 Separate and Joint Argument Identification

Many of the SRL systems described in literature, including the CoNLL 2009 competing setups of Dai et al. (2009), Bohnet (2009) and others, divide the AC task into two subtask: first, the arguments are identified by a binary classifier and then a label is assigned to each of the selected candidates. Another possible option is to combine both subtasks into one, creating an "empty" label and adding it to the set of possible SRL labels in a single classification task. This approach has also been reported to render high quality results (Che et al., 2009; Watanabe et al., 2009).

Table 7.3: A comparison of the joint AC and two-step AI-AC approaches

Setting	F_1	**5 %**	**95 %**
Separate AI and labelling	68.6	64.7	72.5
Joint AC task	71.4	67.2	75.3

The values listed are in the same format as in Table 7.1, the labeled F_1 percentage and the 5 % and 95 % bootstrap percentiles.

We therefore conducted an experiment on our PD Experimental Subset, comparing the performance of the two system variants, so that we could decide for one of them. We used the same classifier configuration as for our experiments described in Section 7.1. The results listed in Table 7.2 show that in our setting, the joint AC task performed better than the two-step identification-classification solution in terms of labeled F-measure. Although the results difference is not significant with 90 % confidence according to the bootstrap test, it is considerably larger than in our previous experiments. We thus chose to use the joint AC approach, as it shows to be more efficient, but also due to its simplicity.

7.3 Different Argument Types

In many linguistic theories that describe semantic (or deep syntactic) dependencies among the words of a sentence, a distinction is made between the valency *arguments* or actants and the *adverbial modifiers* or circumstantials (Sgall et al., 1986, p. 100ff.). While a specific set of the former is required by each predicate, each of its members being unique to the sentence, the latter are generally not limited to any certain group of predicates and can occur freely and repeatedly with most predicates. This distinction is, however, not absolutely clear, and thus some predicates may also require specific adverbial modifiers, whereas some of the arguments may be voluntary with certain predicates. It is also necessary to note that the situation is different for nouns and verbs and varies to some extent across different theories (Rambow et al., 2003), even if the fundamental idea is constant.

The PropBank (Kingsbury and Palmer, 2002; Palmer et al., 2005) and NomBank (Meyers et al., 2004) annotations used in the present thesis (cf. Section

3.2) also take this concept into account and specify a set of numbered arguments `A0...A5` for each nominal or verbal predicate (*frames*), while the sets of possible arguments, as well as their semantic functions change across different predicates. In addition to that, the semantic annotation contains labeled adverbial modifiers `AM-`, not limited to specific predicates and semantically constant. This applies also to reference labels `R-` and `C-`. Only a very small set of verbal frames contains adverbial modifiers as well[4].

Considering such a setting, it becomes apparent that while an individual approach to classification of different predicates is favorable (as discussed in Section 7.1), only the valency arguments benefit from this division in principle, whereas modifiers and references will profit more from a greater training data set. We thus designed an experiment on our PD Experimental Subset to support or disprove this hypothesis: using the same classifier settings as in Sections 7.1 and 7.2, we trained a two-stage system that first assigns arguments labels for each predicate individually and then uses a single classifier trained on the whole training set to label the adverbial modifiers. A comparison of this setting to a system which assigns all semantic labels in a single step is given in Table 7.3.

Table 7.4: A comparison of the joint AC and two-step AI-AC approaches

Setting	F_1	**5 %**	**95 %**
Single step for all labels	71.4	67.2	75.3
Separate `AM-,C-,R-` labels	75.5	71.7	79.1

The format of this table is identical to Tables 7.1 and 7.2.

Although the results difference is not significant with 90 % confidence, the experiment has shown that assigning all the labels which do not depend on the nature of the particular predicate using a separate classifier tuned on the whole training data is capable of improving the performance of the whole system. Therefore, we decided to use this method in our PD system. We will explain further technical issues of this approach in Section 7.7.

7.4 Classifier Setting

In the same way as with our PD system (see Section 6.2), we chose to use one classifier configuration for all the classification tasks of the AC system. Therefore, it was necessary to find the settings that would perform best in an average case. We thus conducted a series experiments with different classifiers and parameter settings, using one joint AC step and the PD Experiment Subset data split for individual predicates. The set of examined ML methods and individual parameter values is analogous to our PD experiment described in Section 6.2, i.e. logistic regression and various SVM implementations from the LibLINEAR package and radial and linear kernel SVM from the LibSVM

[4]For the sake of simplicity, we decided to ignore these modifiers in our experiments for the present time.

Table 7.5: Classifier configuration test on the AC Experimental Subset

Software	Classifier	Parameters	F_1	5 %	95 %
LibLINEAR	L_2-Log. Regression	C = 5	**84.5**	81.4	87.6
		C = 2	84.1	81.1	87.1
		E = 0.01,0.001	**84.1**	80.8	87.1
		E = 0.1	84.0	80.7	86.9
	Multiclass SVM	C = 0.5	**83.7**	80.8	86.8
		C = 0.1,0.2,5	83.6	80.2	86.7
		E = 1	**83.8**	80.8	86.8
		E = 0.001,0.01,0.1	83.3	79.7	86.6
	L_1-SVM (Dual)	C = 1	**84.2**	81.2	87.4
		C = 0.5,5	83.8	80.7	86.8
		E = 0.001	**83.8**	80.6	86.9
		E = 0.01	83.6	80.7	86.7
	L_2-SVM (Dual)	C = 0.2,2	**84.2**	81.0	87.3
		C = 1	84.0	80.9	86.9
		E = 0.001,0.01	**84.2**	81.0	87.2
		E = 0.1	84.1	81.2	87.0
	L_2-SVM (Primal)	C = 0.2,5	**84.3**	81.2	87.2
		C = 2	84.2	81.0	87.0
		E = 0.01	**84.5**	81.6	87.3
		E = 0.001	84.2	81.0	87.0
LibSVM	Linear Kernel	C = 5	**84.5**	81.5	87.3
		C = 2	84.1	80.7	87.2
	Radial Kernel	C = 5	**84.5**	81.3	87.5
		C = 2	84.1	81.0	87.1
		γ = *	**84.1**	80.7	87.2

The listed values are: labeled F_1 percentage on the whole experiment set, bootstrap 5 %-percentile and bootstrap 95 %-percentile. Only two best values for each parameter are listed. An asterisk ("*") in the parameters specifications signifies that any of the tested values is possible.

library have been tested, along with many different settings of the termination criterion $E \in \{1, 0.1, 0.01, 0.001\}$ (LibLINEAR), regularization penalty cost $C \in \{0.05, 0.1, 0.2, 0.5, 1, 2, 5\}$ (all methods) and the second parameter for the radial kernel SVM $\gamma \in \{1, 0.1, 0.01, 0.001, \frac{1}{\#\text{ feats.}}\}$.

However, since the AC task proved to be much more computationally expensive than the PD task, we decided not to use every possible combination of the two classifier parameters, but, assuming their independence, gradually change one of the parameters while fixating the other one[5], thus obtaining a possibly suboptimal (if the independence premise is not valid), but still near-optimal parameter setting by examining the values separately. Due to performance issues, we also used a very simple feature selection algorithm — for each feature ranking (please refer to Section 5.5 for a complete description), a set of the top 10, 20 or 30 features was examined, and the best result was selected[6]. This limits the

[5]We chose to fixate the parameters in the following way: $C = 2, \gamma = \frac{1}{\#\text{feats.}}, E = 0.001$.

[6]We apply the same feature selection technique in the final system, but examine more candidate feature sets.

computational requirements needed to a minimum.

Table 7.5 shows the best observed values on our AC Experimental Subset. It shows that the logistic regression and both LibSVM methods outperformed the LibLINEAR SVMs, although not nearly by a significant margin. Therefore, performance is not the most important criterion of selection. Since LibLINEAR has reportedly smaller memory and CPU requirements (our experiments were not designed to measure this, cf. Fan et al., 2008), we decided to use this library. The logistic regression was then the first choice, along with the best parameters found: $C = 5, E = 0.001$. An advantage of this choice is also the possibility to generate probability estimates (see Section 7.6 for application), which is not available for the other algorithms contained in this package.

Table 7.6: Results of the ranking test experiment on the AC Experimental Subset

Ranking	**Acc.**	**5 %**	**95 %**
χ^2	79.9	76.3	83.4
MI/Information Gain	**80.2**	76.6	83.4
MI (mRMR version)	79.9	76.5	83.1
Symmetric Uncertainty	78.7	75.0	82.0
ReliefF	79.5	75.8	82.8
Significance	79.3	75.7	82.4
mRMR	78.6	74.6	81.9
Average (all)	78.6	75.2	82.2
Average (mRMR + ReliefF)	78.2	74.5	81.9

Similarly to Table 7.5, the listed values are the labeled F_1 percentage on the whole experiment set, followed by the bootstrap percentiles.

In order to choose the default feature ranking, we performed an experiment analogous to the one described in Section 6.3. We applied the classifier configuration selected in the previous experiment to our AC Experimental Subset, selecting the best set of the top 10, 20 or 30 features for each ranking and then comparing the results, which are shown in Table 7.6. Although the differences are rather minor, it is apparent that the mRMR method and both the averages do not perform well in this setting, while MI/Information Gain turns out to be slightly better than the average. We thus selected this method as the default ranking for our AC experiments.

Following the above trials and the subsequent manual analysis of the outputs, we discovered that the previously used feature filtering techniques (keeping only the values that occur in more than 1 % of the cases) are rather coarse. We therefore resorted to filtering out the top 100 most frequent values for each feature, acknowledging the need for a broader analysis of this problem.

7.5 Merging Rare Predicates

As we described in Section 7.1, splitting the input corpora into individual classification problems for different predicates causes extreme data sparsity in some cases. The distribution of predicate frequencies follows the Zipf's law (Manning

and Schütze, 2000, p. 23ff.), which implies that there are only very few training sentences for most predicates (see Table 7.7). A classifier trained on such a small data set as the words of one or two sentences cannot solve the complex AC problem with high reliability. Therefore, a method must be found to prevent the extreme training data scattering.

Table 7.7: Predicate occurrence frequency on the whole training data set

Occurrence count	N	**Occurrence count**	N
101 and above	333	10	155
51 - 100	412	9	167
31 - 50	393	8	208
21 - 30	387	7	240
11 - 20	852	6	333
		5	429
		4	556
		3	781
		2	1315
		1	2667

N stands for number of predicates with the given occurrence count in the training data.

The ideal approach to this situation would be to merge the training data of semantically similar predicates, as the arguments they bind often correspond. Even the PropBank annotation reflects this fact — Kingsbury and Palmer (2002) state that the annotation strives to maintain the same valency argument numbering for synonymous predicates. However, there is no data resource reliably linking the PropBank/NomBank predicate senses to any kind of semantic clustering, such as the verb classes by Levin (1993), VerbNet classes (Kipper-Schuler, 2005), which are based on them, or the FrameNet frames (Baker et al., 1998; Baker and Ruppenhofer, 2002). SemLink, a project linking PropBank with VerbNet (Loper et al., 2007)[7], is currently in development and provides a great resource for semantic research, but as of today, many PropBank predicates, mainly the rare ones, are not yet assigned to any of the semantic groups, which hinders automated usage of this database in the present work.

Since there is no way of merging predicates automatically based on their semantic similarity, we resorted to using the sole presence or non-presence of the individual arguments in the predicate frame. This also might suggest some degree of similarity or at least distinguish a subset of roles that the classifier should assign. In order to test the merits of this rather coarse approach, we performed a test on our AC Experimental Subset. We implemented a script that extracted the frames from PropBank and NomBank framesets and applied the same classifier configuration to the following variants of the data set:

1. Data split for each individual predicate

2. Data merged for predicates which occur less than 10 times in the training data and have the same predicate frames

[7] Available online at `http://verbs.colorado.edu/semlink/`.

3. Same as previous, for predicate occurring less than 20 times in the training data

4. Same as previous, for predicate occurring less than 30 times in the training set

The classifier setup was the same as for the experiments described in Sections 7.1 and 7.3 — logistic regression with regularization cost set at 1 and a termination criterion of 0.001 with no feature selection.

Table 7.8: Results of the predicate merging experiment

Setting	F_1	**5 %**	**95 %**
Individual predicates	71.4	67.2	75.3
Identical frames merged for $N <= 10$	72.4	68.3	76.4
Identical frames merged for $N <= 20$	73.3	69.2	77.1
Identical frames merged for $N <= 30$	74.3	70.4	77.9

The figures listed are in the same format as in Tables 7.1 and 7.2, N stands for number of occurrences of the individual predicates in the training data.

The results of our experiments are displayed in Table 7.5. It shows that merging rare predicates with identical sets of frame arguments helps improve the classification performance, albeit not with 90 % confidence, and further that the improvement is visible even in predicates with 20–30 occurrences in the training data. We decided not to merge more frequent predicates in order to prevent unifying too many semantically unrelated predicates and also due to performance reasons. It is, however, possible that merging even more common predicates with the same frames will yield even better results.

7.6 Post-Inference on Valency Arguments

As we mentioned in Section 7.3, the valency arguments are not only distinct for each predicate, but also not repeatable in the same sentence. It is, however, obvious that an ML method which classifies the words individually is not capable of adhering to such global constraints as uniqueness of the arguments. Therefore, post-inference methods have been developed that modify the classifier output so that it is compliant to the linguistic rules.

Punyakanok et al. (2004) presented a post-inference solution using integer linear programming (ILP), which is also employed in the CoNLL 2009 competition system by Che et al. (2009). Given classifier probability estimates for each word and each semantic role, it tries to maximize the total probability under the constraints that each word is only assigned one semantic role and that the valency arguments may occur only once[8]. This transfers to an ILP task as follows:

[8]The original system of Punyakanok et al. (2004) featured additional constraints on argument references, which we do not include in our system for the sake of simplicity.

- Each semantic role of each word is a binary variable, whose coefficient is the probability estimate given by the classifier.
- The total sum of variables pertaining to one semantic role must be 1.
- The total sum of variables related to each valency argument must be lower or equal to 1 (i.e. 0 or 1).
- The target is to maximize the total sum of all variables with the given coefficients.

An additional threshold parameter may be introduced: If the probability of a semantic role for a given word is lower than the threshold, it is set to zero.

We also designed our own, much simpler post-inference algorithm, which proceeds in the following way:

- For each valency argument role, it finds the word for which this role has the greatest probability in the whole sentence.
- If the probability value found is above a given threshold, the particular role is assigned to the selected word and its probability is set to zero for all the other words.
- If all the valency roles (with sufficiently probable candidates) are selected, the most likely remaining role is assigned for each word that does not already have one.

Table 7.9: Post-inference experiment results

Setting		F_1	**5 %**	**95 %**
No post-inference		71.4	67.2	75.3
Simple post-inference	$T = 0.3$	**72.7**	68.8	76.6
	$T = 0.2$	72.3	68.4	76.3
	$T = 0.1$	72.0	68.1	75.9
	$T = 0.05$	70.2	66.2	74.3
	$T = 0.01$	66.8	62.5	70.7
ILP post-inference	$T = 0.3$	71.7	67.7	75.9
	$T = 0.2$	71.9	68.0	75.7
	$T = 0.1$	**72.1**	68.3	76.1
	$T = 0.05$	71.8	67.1	75.7
	$T = 0.01$	71.8	67.8	75.7
	$T = 0$	71.8	67.8	75.9

The figures listed are in the same format as in Tables 7.1 and 7.2, T stands for the threshold parameter.

We then tested the benefits of both post-inference approaches on our AC Experimental Subset. We programmed the LP_Solve library into our ML framework (see Section 4.4 for technical details) to solve the ILP problem and implemented the simple algorithm on our own. We trained the same classifier as in

Section 7.1 with the same settings, with data split for individual predicates and no feature selection. We then used either no post-inference, or one of the above algorithms with several different threshold values. The results of the experiment are shown in Table 7.6. We can see that keeping the valency arguments unique can bring improvements to our AC system, although only modest (and not significant in terms of the bootstrap test), whereas the simple inference algorithm reached slightly better results. The most viable threshold settings are 0.3 for the simple algorithm and 0.1 for ILP. We decided to employ both variants of post-inference in the final system and compare them to the baseline, since this approach does not require a separate training process and promises performance improvements, albeit small.

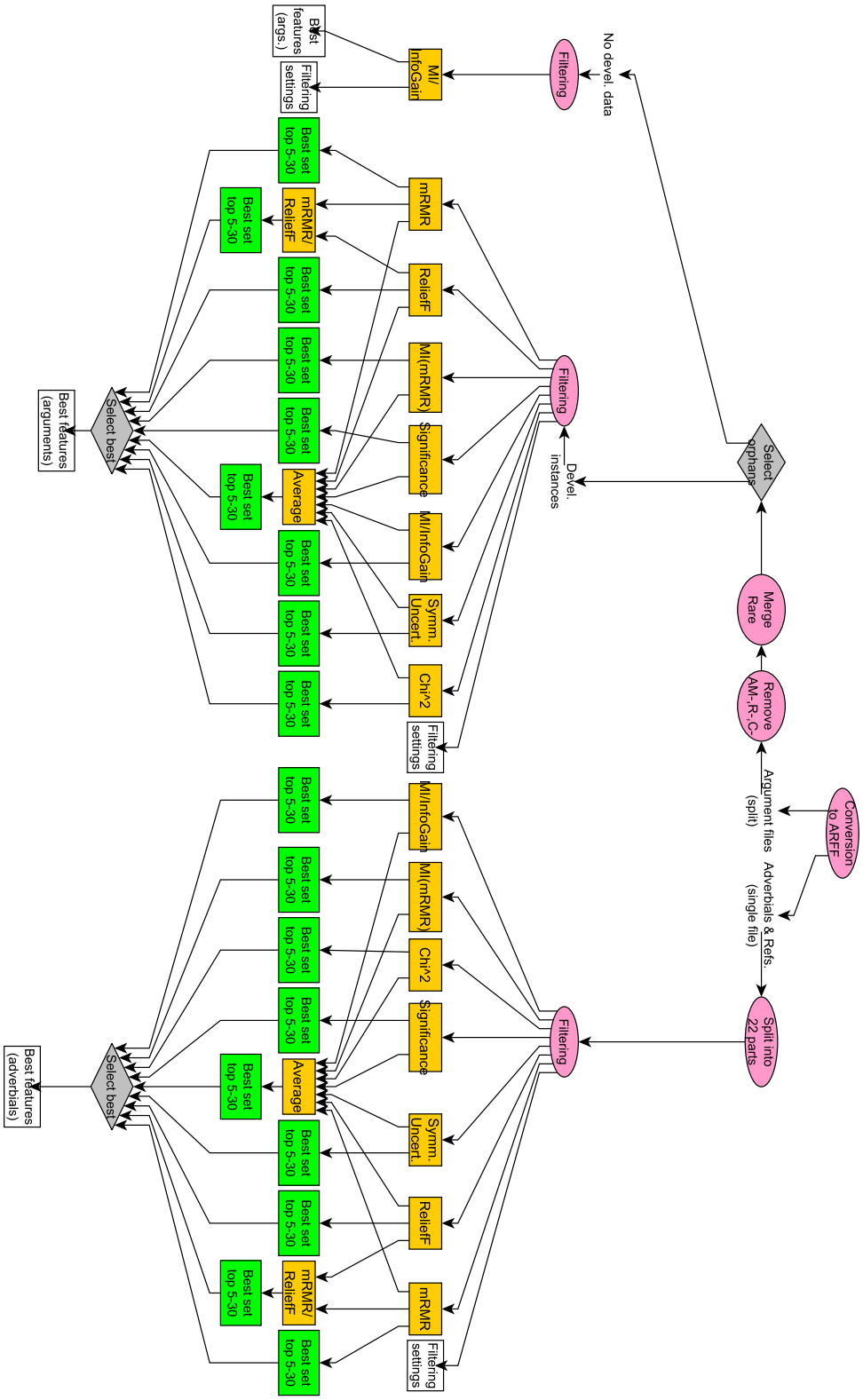
7.7 Adverbial Modifiers Labelling

We have explained the reasons for our intention to split the AC task into argument and adverbial modifier classification subtasks in Section 7.3. The latter subtask, retrieving free adverbial adjuncts and argument co-references, should then use the whole training set to tune the ML model. We discovered that even using up to 16 GB of RAM, which is the maximum for some of the machines in the computer cluster used, it is impossible to train the classifier on such a large data set. Therefore, instead of selecting machines with greater RAM capacity, we decided to make use of the concepts of *ensemble learning* (Opitz and Maclin, 1999; Hastie et al., 2009, p. 605ff.), or *boosting* (Witten and Frank, 2005, p. 321ff.) in particular, even if in a simplified version. Such a choice also allows for more parallelization.

The method itself is analogous to bootstrap and is usually applied to increase classification performance on smaller training data sets — several training sets are sampled from the original one at random, thus fitting a group of models, the outputs of which are then unified by *voting*: Each model votes for one of the classes for each evaluation instance and the class that gets the majority of all votes is selected as the final decision.

We implemented a somewhat different variant of the boosting method. In our setting, a very large input data set is available, which gave us the opportunity not to create random samples where some instances may repeat and others may not be chosen at all, but rather split the whole training set into subsets sequentially and use each one of them to train a different model, which is also technically simpler. Our own version of classifier output unification also added a minor improvement in case that there is a tie among several classes: In that case, our algorithm decides in favor of the classifier that performed best on the development data set.

Figure 7.1: An overall schema of our AC system, training phase
The meaning of the individual shapes and colors is the same as in Figures 6.1 and 6.2.



7.8 The Argument Classification System

Bearing all of the above results in mind, we will now present an overview of our AC setup[9]. Similarly to the PD system (see Section 6.4) and any machine learning method in general, it consists of two phases — training and evaluation. The first phase is depicted in Figure 7.1 and consists of the following steps:

1. The input corpora are converted to the ARFF with no pruning applied (see Section 7.1). Separate ARFF files for the classification of valency arguments and adverbial modifiers are created (as described in Section 7.3), while in the first case, the data are split into different files for the individual predicates, whereas the adverbial modifiers classification consists of a single file. The target class (the semantic label) is split accordingly.

2. Valency argument classification:

   (a) The adverbial and reference semantic labels are removed from the training data set, as they will not be present for evaluation.

   (b) The data examples of predicates that bear the same valency frame are merged if their number of training sentences is 30 or less (cf. Section 7.5).

   (c) The predicates with no development data[10] (denoted as "orphans", similarly to the PD system described in Section 6.4) are separated and their features first filtered and then evaluated using the MI/Information Gain method (cf. Section 7.4). First 30 of them are selected as the optimal subset and saved for later use, along with the filtering settings.

   (d) The remaining (possibly merged) predicate data are filtered and the filtering settings stored. All feature ranking algorithms (see Section 5.5) are applied and the top $10, 15, 20, 25$ or 30 according to each ranking are tested, the best result of which is saved as the selected feature set. Due to the high computational complexity of the task, only such limited feature selection algorithm is employed (cf. also Section 7.4).

3. Adverbial modifiers and references classification:

   (a) The training data is split into 22 parts, 250000 instances each (the reasons are given in Section 7.7).

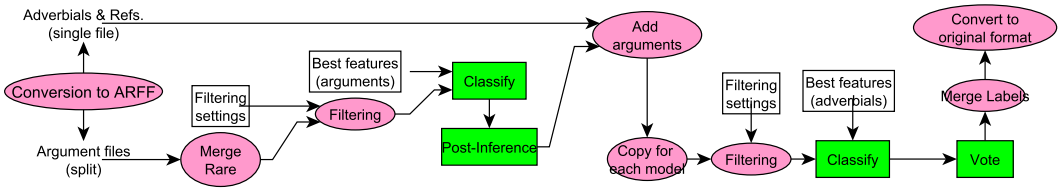
   (b) Each part is then filtered (with the filtering settings saved for later use) and undergoes all feature rankings described in Section 5.5.

   (c) The same simple feature selection technique as for the valency arguments is applied; its results are then stored for the evaluation phase.

[9] The corresponding configuration files for our software framework are included on the enclosed CD, see Appendix B.

[10] There are 212 predicates that have 31 or more training sentences, but no development senteces.

Figure 7.2: A chart of our AC system, evaluation phase
This schema has the same form as the ones in Figures 6.1 and 7.1.



The setting is analogous to the PD setup insofar as the filtering settings and feature sets are the inputs from the training stage into the evaluation stage (the evaluation models are trained anew using these data, see Section 6.4 for explanation). The latter then proceeds as follows (see also Figure 7.2 for the detailed data flow):

1. The evaluation data with predicates disambiguated (as output by the PD evaluation phase, see Section 6.4) are converted to the ARFF format, in the same two variants as in the training phase. The argument values are now missing.

2. Valency argument classification:

   (a) All the rare predicates are merged. Since the merging criterion is the number of occurrences in the training set, this also guarantees that there are no unknown separate predicates.

   (b) The data are filtered according to the settings from the training stage.

   (c) The data are classified, using the best feature sets saved (creating probability estimates for each one of the roles, if post-inference will be used).

   (d) (Optionally) a post-inference algorithm is applied.

3. The results of the valency argument classification are written into the adverbial modifiers evaluation file, so that they can be used by the classifier in this stage.

4. Adverbial modifiers and references classification:

   (a) The evaluation set is copied 22 times and filtered each time according to the different settings.

   (b) The classification proceeds using the individual feature sets selected in the training stage.

   (c) The classification results are unified, using voting.

5. All the semantic roles are merged to a single data column. The results are written back into the original CoNLL corpus format.

The output of our system is thus the original evaluation data file, with all the predicates and semantic arguments filled.

8

The Deep Analysis System: Structure and Performance

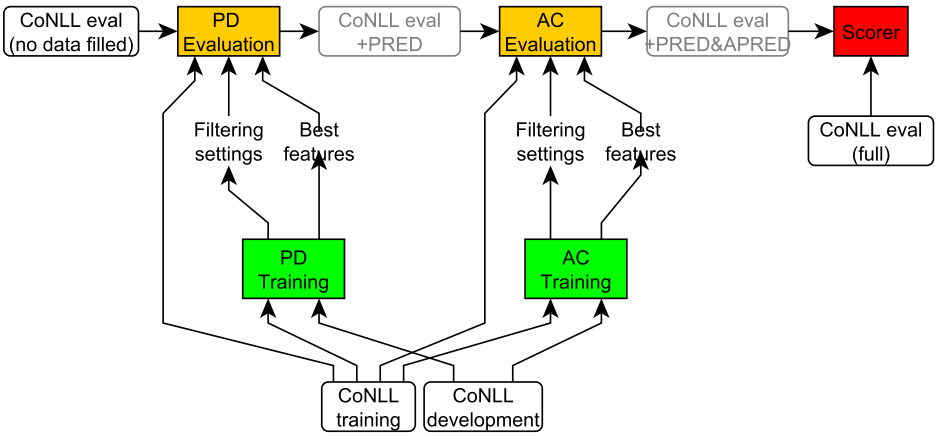
After we discussed the two main parts of our own SRL system in particular in Chapters 6 and 7, we will now provide a brief summary of its overall implementation, connecting the PD and AC subtasks together (in Section 8.1). We include an insight into the feature sets selected in the training phase of both subtasks (Section 8.2) and a detailed analysis of the results of our system on the CoNLL 2009 evaluation data set (see Section 8.3). We attach a short comparison to the other systems competing in the last year's contest (as Section 8.4) and attempt to identify the most probable error sources (Section 8.5).

8.1 Overall Organization

The two main units of our setup, the PD and AC systems, are completely independent of each other in their respective training stages, which allows for parallel computation. They both use the training and development data sets of the CoNLL 2009 corpus, converting them to the ARFF data format (see Section 4.4) with different settings for each unit. The results of the both training stages are the feature filtering settings, along with a set of selected features (see also Section 5.5 for basic details). These are then used to train the models anew in the evaluation stage, thus being its only inputs from the training (please refer to Section 6.4 for explanation). The evaluation phase therefore uses the training data as one of its inputs as well.

By contrast to the training, the evaluation stages of both subtasks depend directly on each other, forming a pipeline. The CoNLL corpus data without `PRED` and `APRED`*i* columns (predicates and arguments specification, see Section 3.4) is first input to the PD system, along with the best features and filtering settings from the PD training stage. The PD evaluation output is then the same

Figure 8.1: The overall organization of our SRL system
The data in CoNLL format are displayed in rectangles with round borders. The system output is marked grey.



file in the original data format, but with all the `PRED` values filled in. We use the conversion into the original format at this point, since it is easier to implement adding the predicates back to the single original file than to append the predicate information into the native AC data format split for different predicates. Since the AC task uses conversion settings different to the PD solution, it is also not possible to use the PD output data directly. The AC evaluation stage then outputs the full data (containing the predicted arguments) in the original format.

The whole configuration is depicted in Figure 8.1, including the final system evaluation, which is done using the original CoNLL 2009 Scorer, so that the result is comparable to other systems enrolled to this contest (see Section 8.4). We will concentrate on the labeled variants of all metrics, since they reflect the performance more accurately, indicating problems with incorrect labeling, too. We include the non-labeled measure outputs only for comparison.

8.2 Features Evaluation

As the output analysis of the training phase, we include an overall account of the most beneficial features chosen for both PD and AC subtasks, since they might provide an insight useful for further enhancements of the SRL system, even if the results are partly influenced by the filtering and selection techniques and a more individual inquiry would also be desirable. Our expectation was that the list of frequently selected features would differ greatly for the both tasks, which proved to be true.

We have excluded the predicate lemmas that did not undergo any tuning on the development data (the "orphans"), because the features selected rest solely on the ranking output. The number of features used in the remaining 525 PD models is very small, with an average of 5.43. This is probably due to the small

Table 8.1: Top 10 most frequently used features in predicate disambiguation

Feature	*U*
Children Types (prepositions)	120
1st word to the right — word form	116
1st word to the right — lemma	78
Form	75
Children Types (particles)	63
Children Types (nominal POSs)	63
Children Types (number of particles)	60
Children lemmas	59
Voice	53
Cluster (form + POS) of the 1st word to the right	52
CPOS of the 1st+2nd word to the right	52
Cluster (lemma + POS) of the 1st word to the right	52
Children types (number of prepositions)	52

The value shown is the total number of classification models that use the given feature, out of 525, i.e. the most often used feature was selected in approx. 38 % of cases.

number of training instances in most cases. The top ten list of popular features is given in Table 8.1. It shows that feature types newly introduced in our system (see Section 5.4) have been rather successful, especially the Children Types. The utility of the features based on word clustering varies greatly with their particular type, with the semantic clusters of the word directly to the right of the predicate being chosen most often. The exact word form of the predicate and of the following word is also chosen very frequently, i.e. some very straightforward features proved to be quite useful.

The AC models favored different groups of features, which varied further for the classification of arguments and adverbials. Some of these features could not be applied to PD at all, since they reflect relations between the predicate and argument candidates. Their number was greater than in the PD task with 11.27 for arguments and 13.41 for modifiers on average, which is probably partly caused by the coarser feature selection techniques. Table 8.2 displays the list of top features for both AC subtasks.

The most useful features for the classification of the valency arguments are thus the ones describing the dependency relation between the predicate and the argument candidate, with the Sibling or Child indication at the top. The adverbial modifiers and references show a somewhat different picture, with several morphology-related features used in all of the partial models (cf. Sections 7.7 and 7.8 for details). The difference reflects the typical real language use relatively closely, as adverbial modifiers tend to occur as distinct adverbs (e.g. “Friday”) or prepositional phrases (e.g. “in Chicago”), which are denoted by their syntactic head — the preposition. The valency arguments, however, are recognized rather by their syntactic relation to the predicate. The both above statements hold in a typical case, which corresponds well to the expected output of a statistical learning method: The classification is trained to be applicable mainly to the most frequent phenomena.

Table 8.2: Top 10 most frequently used features in argument classification

Valency arguments		Other	
Feature	U	**Feature**	U
Sibling or Child (of the predicate)	827	Lemma (predicted)	22
Dependency Path (POS)	520	Dependency Path (dep. relation)	22
Dependency Path (dep. relation)	457	Dependency relation (predicted)	22
Dependency Path (CPOS)	457	CPOS of predicate + current word	22
Head Position (before or after the word)	412	Dependency relation	22
Syntactic Dependence on Predicate	411	POS	22
Head word CPOS	377	Lemma	22
Dependency relation	347	Dependency path (direction)	22
Position (relative to predicate)	271	POS (predicted)	22
Dependency relation (predicted)	258	Form	21

The value shown is the total number of classification models that use the given feature, out of 993 for arguments and 22 for adverbial modifiers and references.

8.3 System Performance on the CoNLL 2009 Evaluation Data

The final results of our SRL system on the CoNLL evaluation data set for the SRL-only task[1] are displayed in Table 8.3. The numbers indicate only a limited success of our solution, with our simple post-inference algorithm ranking best. It shows that the ILP post-inference technique strongly improves precision, while lowering recall, but still increasing F_1 in total. This behavior was also reported by Che et al. (2009). The simple post-inference method on the other hand brings smaller gains in both precision and recall, which add up to a better overall F-measure score. We will therefore focus on the results of the latter in the following, but since the gap between the individual versions is not very wide, the detailed figures are very similar for all of them.

Table 8.3: The results of our SRL solution on the CoNLL evaluation data

System variant	Labeled			Unlabeled		
	P	R	F_1	P	R	F_1
No post-inference (base)	82.91	66.31	73.73	91.61	73.34	81.47
ILP post-inference	**85.19**	65.90	74.32	**93.30**	72.18	81.40
Simple post-inference	84.46	**67.46**	**75.00**	92.47	**73.85**	**82.12**

The listed values are: labeled and unlabelled precision, recall and F_1 percentage, respectively.

As the scorer figures include the performance in both predicate disambiguation and argument classification, we have created several scripts for further automatic analysis, both a direct one of the system output and an indirect one of the results given by the CoNLL scorer. We present the relevant figures in Tables 8.4, 8.5 and 8.6.

[1] This evaluation data set contained predicted syntactic relations, which we copied to be used as the golden ones for the generation of other features, as well as for classification.

Table 8.4: Detailed performance of the PD subtask

Predicate lemmas	*N*	*A*
All lemmas	10498	95.06
Multiple senses	3834	86.86
"Simple" (2-3 senses)	2420	89.67
"Harder" (4-8 senses)	943	85.37
"Tough" (more than 8 senses)	480	75.63
Nominal lemmas with multiple senses	1452	86.98
Verbal lemmas with multiple senses	2391	86.78

The listed values are: the total number of predicates of the given type and the total accuracy percentage on these predicates (see Section 6.2 for explanation). The PD part of the system is identical for all variants.

It is now apparent that the PD subtask is able to achieve considerably high accuracy, but partly due to the fact that many lemmas are not ambiguous. There is still a margin for possible further improvement (see Table 8.4 for details). With highly ambiguous lemmas, the performance is markedly weaker than for the simpler problems, even if much broader range of candidate feature subsets has been explored. The classification of nominal and verbal predicate lemmas is almost equally successful.

The output of the AC solution is much more problematic. The semantic labeling precision still remains relatively high in all variant, but the recall figures show the major difficulty — the system does not recognize a considerably large portion of the arguments. If we observe the more detailed numbers in Table 8.5, we can see that this problem pertains to all types of labels: valency arguments as well as adverbial modifiers and references. However, the arguments (`A0...A5`) feature considerably better results than the rest of the labels. The lower figures for adverbial modifiers are caused mainly by an inferior performance in nominal predicates — most of their modifiers are not identified at all. The classification of verb modifiers is significantly more successful, while still not as accurate as with valency arguments. On the other hand, the attainment in negation particles and modal verbs as arguments of the verbal predicates exceeds 90 % F-measure, which is probably thanks to the fact that these modifier types are mostly identified by the word form itself.

Our results also indicate a slightly greater efficiency in verbal predicates (cf. Table 8.6), which is possibly also caused by the problems with adverbial modifiers of nouns. Further, if we look at the performance for different predicates shown in Table 8.6, it becomes apparent that in our setting, the correct output of the PD subtask seriously influences the AC performance. This only reflects the fact that disparate word senses often bind unequal argument types.

We also examined how the system performance depends on the frequency of the predicates. It appears that the overall efficiency in terms of F-measure grows slightly for more frequent predicates, but stagnates with very common ones, which could be expected. The higher value for the most plentiful predicates is caused by a very successful classification of two of them, "say.01.v" and "%.01.n". The F-measure growth with more common predicates is mainly brought about

Table 8.5: Detailed performance of the AC subtask by argument type

The values listed are: the total numbers of instances of the given type (in the gold standard data) and labeled precision, recall and F-measure percentages of the simple post-inference version, respectively. Other POSs as predicates and very rare kinds of adverbial modifiers are omitted in particular, but included in the total numbers.

Subset	**Size**	P	R	F_1
All predicates	23286	77.71	55.01	64.42
Valency arguments	17760	79.65	59.30	67.98
Adverbial modifiers	4838	70.79	42.97	53.48
References	205	100.00	1.95	3.83
Co-references	483	60.75	40.37	48.51
Nominal pred. + A0	2339	79.20	50.79	61.89
Nominal pred. + A1	3757	79.38	55.74	65.49
Nominal pred. + A2	1537	72.93	53.29	61.58
Nominal pred. + A3	349	67.10	59.03	62.81
Nominal pred. + A4	18	62.50	55.56	58.83
Nominal pred. + A5	1	-	0.00	-
Nominal pred. + AM*	1186	62.79	4.55	8.49
Nominal pred. + C-*	2	-	0.00	-
Nominal pred. + R-*	2	-	0.00	-
Verbal pred. + A0	3509	81.02	61.44	69.88
Verbal pred. + A1	4844	86.40	66.78	75.33
Verbal pred. + A2	1085	68.55	59.26	63.57
Verbal pred. + A3	169	54.14	57.99	56.00
Verbal pred. + A4	99	72.28	73.74	73.00
Verbal pred. + A5	5	100.00	80.00	88.89
Verbal pred. + AA	0	0.00	-	-
Verbal pred. + AM*	3630	71.07	55.76	62.49
Verbal pred. + C-*	202	100.00	1.98	3.88
Verbal pred. + R-*	481	60.75	40.54	48.63
Nominal pred. + AM-ADV	32	100.00	3.12	6.05
Nominal pred. + AM-CAU	2	100.00	50.00	66.67
Nominal pred. + AM-EXT	33	-	0.00	-
Nominal pred. + AM-LOC	232	40.00	0.86	1.68
Nominal pred. + AM-MNR	344	-	0.00	-
Nominal pred. + AM-MOD	7	100.00	28.57	44.44
Nominal pred. + AM-NEG	35	100.00	22.86	37.21
Nominal pred. + AM-TMP	492	57.97	8.13	14.26
Verbal pred. + AM-ADV	488	43.34	34.02	38.12
Verbal pred. + AM-CAU	70	67.50	38.57	49.09
Verbal pred. + AM-DIR	81	66.67	2.47	4.76
Verbal pred. + AM-DIS	315	63.35	44.44	52.24
Verbal pred. + AM-EXT	32	54.55	18.75	27.91
Verbal pred. + AM-LOC	355	53.70	49.01	51.25
Verbal pred. + AM-MNR	335	60.65	28.06	38.37
Verbal pred. + AM-MOD	539	94.15	89.61	91.82
Verbal pred. + AM-NEG	227	93.56	96.04	94.78
Verbal pred. + AM-PNC	113	52.24	30.97	38.89
Verbal pred. + AM-TMP	1068	75.61	63.58	69.08

Table 8.6: Detailed performance of the AC subtask by predicate type

Subset	Size	P	R	F_1
All predicates	23286	77.71	55.01	64.42
Nominal predicates	9191	77.09	47.56	58.83
Verbal predicates	14024	78.05	60.13	67.93
Correctly disambiguated predicates	22017	78.69	56.23	65.59
Wrong predicate sense	1269	57.10	33.88	42.53
Merged predicates	5054	68.21	18.34	28.91
Predicates classified individually	18232	78.56	65.18	71.25
Occurence count in training data = 0-10	2438	64.53	15.67	25.22
Occurence count in training data = 11-20	1406	67.53	18.49	29.03
Occurence count in training data = 21-30	1090	72.39	19.72	31.00
Occurence count in training data = 31-40	1038	79.05	61.08	68.91
Occurence count in training data = 41-50	830	73.56	53.98	62.27
Occurence count in training data = 51-100	3603	77.75	60.92	68.31
Occurence count in training data = 101-200	3443	78.04	64.71	70.75
Occurence count in training data = 201-500	5013	77.79	65.97	71.39
Occurence count in training data = 501-1000	2314	76.57	64.69	70.13
Occurence count in training data > 1000	2111	85.80	77.88	81.65

The format of this table is the same as in Table 8.5.

by the increase in recall. A very problematic finding is that recall stays very low especially for the predicates that were merged according to their frames due to lack of training data (as described in Section 7.5).

8.4 Comparison to CoNLL 2009 Shared Task Systems

Since the outputs of all setups competing in the CoNLL 2009 Shared Task are publicly available[2], we may analyze them and compare them to our results. If we look at the final overall SRL results for English (cf. tables in Hajič et al., 2009), our system performance is average; it would rank fourth out of eight SRL-only solutions (including our system). Compared to all competing setups, including the ones that solve both the syntactic and semantic subtasks, it would receive the 15th place out of 21.

A more detailed examination of all competing systems results has shown that our PD solution ranks fourth best, after the setups of Täckström (2009), Björkelund et al. (2009) and Brown[3], which reached 95.62 %, 95.59 % and 95.32 % accuracy, respectively. Several other systems have passed the barrier of 94 %, but many of them classify predicate senses with around or less than 90 % confidence. This shows that several setups, including the best one for English by Dai et al. (2009), perform better on the whole, even if their PD part is less accurate than our solution. The influence of the PD part on the accuracy of the AC part varies greatly, from only 3 % drop in F-measure experienced by the setup of Li et al. (2009), up to 20 % or 28 % decrease that may be observed in the classification

[2]Downloadable at `http://ufal.mff.cuni.cz/conll2009-st/results/results.php`.

[3]The authors of this particular SRL system did not provide a description paper.

outputs of Björkelund et al. (2009) and Lin[4], respectively. Most solutions display a drop of approximately 10 %. Our findings show a comparatively greater dependence between the two subtasks, given by the separate classification of different predicates.

If we compare the AC subtask results, it shows that while the best ranking setups (Björkelund et al., 2009; Zhao et al., 2009) produce a very balanced output, with a comparable performance for nominal and verbal predicates and also for valency arguments, adverbial modifiers and references, most solutions achieve a better F-measure score with verbs and their arguments, which apparently follows from the fact that verbal arguments are often more closely syntactically bound than the nominal ones. Possibly due to the same cause, most systems classify adverbial modifiers and references with less confidence than the valency arguments. These findings are very similar to our experience.

A majority of the setups also behaves very similar to ours with respect to the precision-recall ratio: it is very usual that a classifier attains higher precision than recall in this particular task, mostly by a few percent points, with the top systems being better balanced, but even 10 % in the case of Meza-Ruiz and Riedel (2009). There are also some exceptions from this rule (Morante et al., 2009). However, the gap between precision and recall figures reached by our system, which stems both from generally lower recall figures and extremely low recall in adverbial modifiers of nouns, is even wider.

8.5 Possible Sources of Errors

The performance analysis and comparison given in Sections 8.3 and 8.4 have identified the weaknesses of our SRL setup. We will now attempt to provide their most probable causes for both PD and AC subtask and suggest possible improvements.

Concerning the former, the data sparsity is most probably among the main error sources: For many predicate lemmas, there are only very few or none training data examples or none at all, which leads to a trivial model assigning the most frequent sense. Apart from using more sophisticated features and more detailed filtering, which could lead to a greater efficiency of simple modes with only few variables, there are not many remedies to this problem — since the number and frequency of the individual senses is specific to each lemma, there is no straightforward way of grouping the rare ones. In more common predicates, which also tend to be the more ambiguous ones, we believe that employing a more exhaustive feature search, e.g. applying the greedy algorithm to a wider set of predicates or using beam search (Aha and Bankert, 1995), could contribute to at least partial rectification of the classification errors.

The PD subtask results pose a more significant problem, especially the low recall figures in some cases. They indicate that particularly for very large and inhomogeneous training sets, such as the adverbial modifiers classification (see Section 7.7), but also some of the merged problems for rare predicates (delineated

[4]There is also no paper describing this CoNLL 2009 contest system.

in Section 7.5), the feature filtering and selection techniques are not powerful enough to suggest arguments reliably and show themselves too coarse for divergent information sources. Therefore, a finer feature filtering and ranking, and also a more exhaustive feature space search in the selection phase should most probably improve the system performance regarding such phenomena. More sophisticated features could also contribute to a greater accuracy in argument identification.

As the adverbial modifiers of verbs were classified more successfully than the ones of nouns by our system, we believe that additional features describing the syntactic relations independent of POS could also yield further progress. Another option regarding adverbial modifiers is to separate their classification model for verbs and nouns. Since the recall figures for (co-)references are also very low, it could (although the proportion of such events is not very high in the CoNLL 2009 corpus) prove beneficial to solve the reference classification individually and use special features tailored for this task, building upon usual anaphora resolution approaches (Soon et al., 2001; Ng and Cardie, 2002).

9 Conclusions

We will conclude the thesis with a brief summary of our SRL system results and an account of our contributions to the deep analysis (Section 9.1). We will also suggest some possibilities of further performance improvement and introduce additional applications of our setup in Section 9.2.

9.1 Results

Using our original software framework for effective batch parallel processing of machine learning (ML) tasks, which features a space-saving description of multiple analogous subtasks and our own implementations of data conversion, filtering and feature selection tools, combined with third-party classifier, ranking and data storage libraries, we have developed a deep analysis solution for English, applying the predicate-argument structure approach to semantic description used in the CoNLL 2009 Shared Task contest (Hajič et al., 2009, introduced in Chapter 3). Being very abstract and extensible, our ML framework is also applicable to many other problems consisting of subtasks depending on each other.

We conducted and evaluated series of experiments with various semantic analysis subproblems within our framework in order to find the optimal setting. Our setup, consisting of the predicate disambiguation (PD) and argument classification (AC) subtasks, therefore includes several novel improvements to the present deep analysis solution: new data feature types, different approach to PD problems depending on their complexity, using different classification models for individual predicates in the AC subtask in combination with rare predicate clustering based on their valency frames, a separate global solution for adverbial modifiers and the application of a new simple post-inference algorithm guaranteeing the uniqueness of valency arguments.

We have evaluated our apporach using the CoNLL 2009 Shared Task English corpus, achieving the overall F-measure of 75.00 %, with PD accuracy of 95.06 %

and AC F_1 of 64.42 %. A detailed analysis and a comparison of our results to the findings of the CoNLL 2009 competition show that our semantic classification solution can compete with other contestants' systems and is able to achieve top performance in PD and relatively high precision for the AC task in most cases, but shows lower recall values, especially for valency arguments of rare predicates and adverbial modifiers of nouns. We have identified the most probable error sources and propose possible paths leading to their compensation.

The evaluation has further shown that for the PD task, our newly introduced data features are one of the most often selected. Our simple post-inference algorithm in the AC part of the setup also proved to be a contribution to the overall better performance. Our experiments with separate adverbial modifier classification suggest possible improvements using this approach, but the results of our system do not yet match the top ranking approaches in this respect.

9.2 Further Research

As our error analysis has shown comparably lower recall values for the PC subtask, especially in large inhomogeneous classification models, we assume that better feature filtering and selection will improve the results. Therefore, we propose examining more individual approaches to feature filtering, depending on the size of the given classification model, and analyzing the possibility of using more exhaustive feature space search. We also consider applying a two-stage feature selection, before and after feature binarization, which would lead to much higher computational complexity, but promises certain improvements. Inferring the most viable feature subset based on the outputs of various feature rankings should also be tested.

Adding a wider variety of generated features, possibly including feature bigrams and global features (Björkelund et al., 2009) or special features for different argument types, also invites further research. We are also interested in separate classification models for adverbial modifiers of nouns and verbs and a different approach to (co-)references. Another improvement possibility could be tuning the classifier parameters more precisely for different variants of the AC problem.

Since our system is based on statistical learning methods, which are highly language-independent, and the CoNLL 2009 contest included semantically annotated corpora for seven different languages, all using predicate-argument structure and the same basic input format, our system can be easily modified to be applicable to the other data sets provided. Several details in our solution are specifically tailored for our English corpus (e.g. the generated features make use of its morphological tag set), but a modification for another language or annotation is not very complicated.

We have already mentioned the theoretical possibility of modifying the system designed for the CoNLL 2009 English semantic description in order to use it with a different English corpus — the Prague English Dependency Treebank (Cinková et al., 2009). The conversion of the annotation format remains the most difficult prerequisite task, but its solution may be analogous to the CoNLL 2009 Shared

Task conversion of the Prague Dependency Treebank (Hajič et al., 2006, 2009), which uses a similar annotation schema.

List of Abbreviations

AC	Argument Classification
AI	Argument Identification
API	Application Programming Interface
ARFF	Attribute-Relation File Format
CoNLL	Conference on Computational Natural Language Learning
CPOS	Coarse Part-of-Speech (first letter of the part-of-speech tag)
FGD	Functional Generative Description
ILP	Integer Linear Programming
JNI	Java Native Interface
MaxEnt	Maximum Entropy model, Maximum Entropy classifier
MI	Mutual Information
MIRA	Margin-Infused Relaxed Algorithm
mRMR	Minimum Redundancy-Maximum Relevance
NLP	Natural Language Processing
PD	Predicate Disambiguation
PDT	Prague Dependency Treebank
PEDT	Prague English Dependency Treebank
PI	Predicate Identification
POS	Part-of-Speech
PropBank	The Proposition Bank
PTB	Penn Treebank
SRL	Semantic Role Labeling
SVM	Support Vector Machine
WEKA	Waikato Environment for Knowledge Analysis

Bibliography

D. W Aha and R. L Bankert. A comparative evaluation of sequential feature selection algorithms. *Learning from Data: Artificial Intelligence and Statistics V*, page 199–206, 1995.

A. Ahmad and L. Dey. A feature selection technique for classificatory analysis. *Pattern Recognition Letters*, 26(1):43–56, 2005.

C. F Baker and J. Ruppenhofer. FrameNet's frames vs. Levin's verb classes. In *Proceedings of the 28th Annual Meeting of the Berkeley Linguistics Society*, page 27–38, 2002.

C. F Baker, C. J Fillmore, and J. B Lowe. The Berkeley Framenet project. In *Proceedings of the 17th international conference on Computational linguistics-Volume 1*, page 86–90, 1998.

A. L Berger, V. J.D Pietra, and S. A.D Pietra. A maximum entropy approach to natural language processing. *Computational linguistics*, 22(1):39–71, 1996.

A. Björkelund, L. Hafdell, and P. Nugues. Multilingual semantic role labeling. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, page 43–48, 2009.

D. J Blower. An easy derivation of logistic regression from the Bayesian and maximum entropy perspective. In *AIP Conference Proceedings*, volume 707, page 30, 2004.

B. Bohnet. Efficient parsing of syntactic and semantic dependency structures. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, page 67–72, 2009.

B. E Boser, I. M Guyon, and V. N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, page 144–152, 1992.

X. Carreras and L. Marquez. Introduction to the CoNLL-2004 shared task: Semantic role labeling. In *Proceedings of CoNLL-2004*, pages 89—97, 2004.

X. Carreras and L. Marquez. Introduction to the CoNLL-2005 shared task: Semantic role labeling. In *Proceedings of the Ninth Conference on Computational Natural Language Learning*, page 152–164, 2005.

R. Caruana and D. Freitag. Greedy attribute selection. In *Proceedings of the Eleventh International Conference on Machine Learning*, page 28–36, 1994.

C. C Chang and C. J Lin. LIBSVM: a library for support vector machines. 2001.

W. Che, Z. Li, Y. Hu, Y. Li, B. Qin, T. Liu, and S. Li. A cascaded syntactic and semantic dependency parsing system. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, page 238–242, 2008.

W. Che, Z. Li, Y. Li, Y. Guo, B. Qin, and T. Liu. Multilingual dependency-based syntactic and semantic parsing. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, page 49–54, 2009.

E. Chen, L. Shi, and D. Hu. Probabilistic model for syntactic and semantic dependency parsing. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, page 263–267, 2008.

N. Chomsky. *Government and binding*. Foris, Dordrecht, 1981.

S. Cinková, J. Toman, J. Hajič, K. Čermáková, V. Klimeš, L. Mladová, J. Šindlerová, K. Tomšů, and Z. Žabokrtský. Tectogrammatical annotation of the Wall Street Journal. *The Prague Bulletin of Mathematical Linguistics*, (92), 2009.

C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3): 273–297, 1995.

K. Crammer and Y. Singer. Ultraconservative online algorithms for multiclass problems. *The Journal of Machine Learning Research*, 3:951–991, 2003.

N. Cristianini and J. Shawe-Taylor. *An introduction to support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.

Q. Dai, E. Chen, and L. Shi. An iterative approach for joint dependency parsing and semantic role labeling. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, page 19–24, 2009.

B. Efron. Bootstrap methods: another look at the jackknife. *The annals of statistics*, 7(1):1–26, 1979.

R. E Fan, K. W Chang, C. J Hsieh, X. R Wang, and C. J Lin. LIBLINEAR: a library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008.

S. R Garner. WEKA: The waikato environment for knowledge analysis. In *Proceedings of the New Zealand Computer Science Research Students Conference*, page 57–64, 1995.

A. Gesmundo, J. Henderson, P. Merlo, and I. Titov. A latent variable model of synchronous syntactic-semantic parsing for multiple languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, page 37–42, 2009.

D. Gildea and D. Jurafsky. Automatic labeling of semantic roles. *Computational Linguistics*, 28(3):245–288, 2002.

A. M Giuglea and A. Moschitti. Semantic role labeling via FrameNet, VerbNet and PropBank. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, page 936, 2006.

J. Hajič, J. Panevová, E. Hajičová, P. Sgall, P. Pajas, J. Štepánek, J. Havelka, M. Mikulová, Z. Žabokrtský, and M. Š Razímová. Prague Dependency Treebank 2.0. *LDC2006T01*, page 1–58563, 2006.

J. Hajič, M. Ciaramita, R. Johansson, D. Kawahara, M. A Martí, L. Marquez, A. Meyers, J. Nivre, S. Padó, J. Štěpánek, et al. The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, page 1–18, 2009.

T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin. *The elements of statistical learning: data mining, inference and prediction*. Springer, 2nd edition, 2009.

D. W Hosmer and S. Lemeshow. *Applied logistic regression*. Wiley-Interscience, 2000.

C. W Hsu, C. C Chang, C. J Lin, et al. A practical guide to support vector classification. 2003.

F. Jelinek. *Statistical methods for speech recognition*. MIT Press, 1997.

Z. P Jiang and H. T Ng. Semantic role labeling of NomBank: a maximum entropy approach. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, page 138–145, 2006.

R. Johansson and P. Nugues. Extended constituent-to-dependency conversion for English. In *Proc. of the 16th Nordic Conference on Computational Linguistics (NODALIDA)*, 2007.

G. H John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In *Proceedings of the eleventh international conference on machine learning*, volume 129, 1994.

A. B. Kahn. Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562, 1962.

P. Kingsbury and M. Palmer. From treebank to PropBank. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC-2002)*, page 1989–1993, 2002.

P. Kingsbury and M. Palmer. PropBank: the next level of treebank. In *Proceedings of Treebanks and lexical Theories*, 2003.

K. Kipper-Schuler. *VerbNet: A broad-coverage, comprehensive verb lexicon.* PhD thesis, University of Pennsylvania, 2005.

K. Kira and L. A Rendell. A practical approach to feature selection. In *Proceedings of the ninth international workshop on Machine learning*, page 249–256, 1992.

I. Kononenko. Estimating attributes: Analysis and extensions of RELIEF. In *Machine Learning: ECML-94*, page 171–182, 1994.

A. Kulkarni and T. Pedersen. SenseClusters: Unsupervised clustering and labeling of similar contexts. In *Proceedings of the ACL 2005 on Interactive poster and demonstration sessions*, page 108, 2005.

S. I Lee, H. Lee, P. Abbeel, and A. Y Ng. Efficient L1-regularized logistic regression. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 401, 2006.

Beth Levin. *English verb classes and alternations: A preliminary investigation.* Univeristy of Chicago Press, Chicago, 1993.

B. Li, M. Emms, S. Luz, and C. Vogel. Exploring multilingual semantic role labeling. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, page 73–78, 2009.

Edward Loper, Szu ting Yi, and Martha Palmer. Combining lexical resources: Mapping between PropBank and VerbNet. In *In Proceedings of the 7th International Workshop on Computational Linguistics*, 2007.

P. J MacCullagh and J. A Nelder. *Generalized linear models.* Chapman & Hall, 2nd edition, 1991.

R. Malouf et al. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the Sixth Conference on Natural Language Learning (CoNLL-2002)*, page 49–55, 2002.

C. D. Manning and H. Schütze. *Foundations of statistical natural language processing.* MIT Press, 2000.

C. D Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval.* Cambridge University Press, New York, 2008.

M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics*, 19(2):330, 1993.

R. McDonald, F. Pereira, K. Ribarov, and J. Hajič. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, page 523–530, 2005.

A. Meyers, R. Reeves, C. Macleod, R. Szekely, V. Zielinska, B. Young, and R. Grishman. The NomBank project: An interim report. In A. Meyers, editor, *HLT-NAACL 2004 Workshop: Frontiers in Corpus Annotation*, page 24–31, Boston, Massachusetts, USA, May 2004. Association for Computational Linguistics.

I. Meza-Ruiz and S. Riedel. Multilingual semantic role labelling with markov logic. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, page 85–90, 2009.

R. Morante, V. Van Asch, and A. Van den Bosch. Joint memory-based learning of syntactic and semantic dependencies in multiple languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, page 25–30, 2009.

E. Moreau and I. Tellier. The crotal SRL system: a generic tool based on tree-structured CRF. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, page 91–96, 2009.

P. Moreda and M. Palomar. The role of verb sense disambiguation in semantic role labeling. *Advances in Natural Language Processing*, page 684–695, 2006.

A. Neumaier. Solving ill-conditioned and singular linear systems: A tutorial on regularization. *SIAM Review*, 40:636–666, 1998.

A. Y Ng. Feature selection, L1 vs. L2-regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78, 2004.

V. Ng and C. Cardie. Improving machine learning approaches to coreference resolution. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, page 111, 2002.

D. Opitz and R. Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11(169-198):12, 1999.

M. Palmer, P. Kingsbury, and D. Gildea. The Proposition Bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106, 2005.

H. Peng, F. Long, and C. Ding. Feature selection based on mutual information: criteria of Max-Dependency, Max-Relevance, and Min-Redundancy. *IEEE Transactions on pattern analysis and machine intelligence*, page 1226–1238, 2005.

F. Pereira, N. Tishby, and L. Lee. Distributional clustering of English words. In *Proceedings of the 31st annual meeting on Association for Computational Linguistics*, page 183–190, 1993.

V. Punyakanok, D. Roth, W. Yih, and D. Zimak. Semantic role labeling via integer linear programming inference. In *Proceedings of the 20th international conference on Computational Linguistics*, page 1346, 2004.

O. Rambow, B. Dorr, I. Kučerová, and M. Palmer. Automatically deriving tectogrammatical labels from other resources: A comparison of semantic labels across frameworks. *The Prague Bulletin of Mathematical Linguistics*, 80, 2003.

B. Santorini. *Part-of-speech tagging guidelines for the Penn Treebank Project.* University of Pennsylvania, 3rd edition, 1990.

J. Semecký and S. Cinková. Constructing an English valency lexicon. In *Proceedings of the Workshop on Frontiers in Linguistically Annotated Corpora 2006*, page 94–97, 2006.

P. Sgall, E. Hajičová, and J. Panevová. *The meaning of the sentence in its semantic and pragmatic aspects.* D. Reidel, Dordrecht, 1986.

W. M Soon, H. T Ng, and D. C.Y Lim. A machine learning approach to coreference resolution of noun phrases. *Computational Linguistics*, 27(4):521–544, 2001.

M. Surdeanu, R. Johansson, A. Meyers, L. Marquez, and J. Nivre. The CoNLL-2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, page 159–177, 2008.

O. Täckström. Multilingual semantic parsing with a pipeline of linear classifiers. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, page 103–108, 2009.

Y. Watanabe, M. Asahara, and Y. Matsumoto. Multilingual syntactic-semantic dependency parsing with three-stage approximate max-margin linear models. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, page 114–119, 2009.

R. Weischedel and A. Brunstein. BBN pronoun coreference and entity type corpus. *LDC2005T33, Linguistic Data Consortium, Philadelphia*, 2005.

I. H Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann Pub, 2nd edition, 2005.

D. Zeman. A simple generative pipeline approach to dependency parsing and semantic role labeling. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, page 120–125, 2009.

Y. Zhang, C. Ding, and T. Li. Gene selection algorithm by combining ReliefF and mRMR. *BMC genomics*, 9(Suppl 2):S27, 2008.

H. Zhao, W. Chen, C. Kit, and G. Zhou. Multilingual dependency learning: a huge feature engineering method to semantic dependency parsing. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, page 55–60, 2009.

A

List of Machine Learning Tasks Implemented

Each of the subtasks implemented corresponds to one Java class, as described in Chapter 4. The tasks are divided into groups according to their purpose; their ordering is alphabetical. Detailed technical descriptions, including lists of required or optional parameters and the formats of inputs and outputs, are a part of the Javadoc documentation provided on the enclosed CD (cf. Appendix B).

Computation Tasks

AverageAttributeRanks	Makes an average from several attribute rankings (it averages the attributes' positions, if the resulting numbers are the same, it orders them according to the first ranking given), as described in Section 5.5.
ConcatClassifier	A dummy classifier for the files where no training data are present — concatenates given input fields and a pre-set text (such as the lemma and ".01", see Section 6.4).
DummyClassifier	A dummy classifier which always assigns the most common value of the target attribute in the training data. We use it with unary predicates (see Section 6.1).
GreedyAttributeSearch	This class applies the greedy attribute selection algorithm (see Sections 5.5 and 6.3) with the given WEKA classifier configuration, capable of starting with a predefined number of attributes or all subsets of a given size, possibly confined to a set of best attributes as output by a ranking algorithm. Each round may be parallelized.

ILPSemanticResolver Using the LP_Solve library, this creates an ILP task which optimizes the probability of all semantic labels under the constraint that valency arguments must not repeat (see Section 7.6)

mRMR Our own implementation of the minimum Redundancy – Maximum Relevance feature ranking algorithm (see Section 5.5).

MutualInformation Attribute ranking based on mutual information (see Section 5.5); needed for our mRMR implementation.

SettingSelector This class tries to run the given WEKA classifier on the same data with several different sets of settings and then selects the one with the best results. We used it in our search for the best classifier parameters (see Sections 6.2 and 7.4).

SimpleSemanticResolver Given the probability distributions of the individual classifications of words in a sentence, this employs our own simple procedure for assigning the most likely semantic roles, while preserving the uniqueness of valency arguments (see Section 7.6).

SubsequentAttributeAdder A Search for the best performing number of attributes (within a specified range, with a specified stepping), given an attribute ranking. All trials may run in parallel. See Section 6.3 for details on use in our system.

WekaAttributeRanker This applies a WEKA feature ranking (or our own MI, mRMR) to evaluate all features in a data set.

WekaClassifier This task either trains a specified WEKA library model with the given parameters and data set, or classifies given data sets using a pre-loaded model, or both. It is used by the feature selection searches, or standalone (for usages in our setup, see Sections 6.4 and 7.8). It is capable of selecting only the given features and binarizing them.

Data Manipulating Tasks

AttributeAdder This allows re-adding a feature to a data set (such as sentence ID) that was previously deleted because of feature selection.

AttributeDivider This divides an attribute in two, filling the remaining values with an empty one, signified by "_" (used for adverbial modifiers split classification, as described in Section 7.7).

AttributeFilter This creates new attributes that contain only the most common values of the old ones, optionally deleting the old ones (see Section 5.5).

AttributeMerger This merges two attributes into one, replacing "_" values (cf. Section 7.7).

AttributeSelector This class filters out some features from the data set.

BigDataSplitter This will split a large data file into chunks based on the number of input instances (cf. Section 7.7).

ClassificationMerger	This tries to merge the classification of several classifiers, using voting (cf. Section 7.7).
CollectingAdder	This gathers values of a features in a large data set from several smaller data sets.
ConditionalSelector	This class sorts the input files to several groups, according to a given condition, i.e. number of possible values of a feature, existence of development data etc. (used for selecting "orphans" and splitting the different types of predicates, see Sections 6.4 and 7.8).
CopyInput	A dummy task that just copies its selected inputs to outputs, used for debugging.
DataMerger	This class merges several data sets into one, if they have the same format (used in predicate merging).
FileGroupsMerger	This will just merge the files that originate from different outputs and give them compatible names (and issue errors if their names collide).
FileMerger	This class merges several (text) files into one by writing them sequentially one after another.
IrrelevantAttributes-Removal	This removes all the attributes that are irrelevant for the classification (described in Section 5.5).
MergedDataSplitter	This splits the given files, if they've been previously merged and contain an attribute indicating the original file name, while preserving the non-merged files intact (cf. Section 7.5).
PredicateMerger	This merges rare predicates according to their frames (see Section 7.5).
RandomSampleSelector	This selects a random sample from the given input files and passes it to the output (used for our AC experiments, see Section 7.1).
ResultsToSt	This adds the results from the classification files back to the corpus file in the original format, possibly rewriting some fields (applied in the evaluation stage of both AC and PD systems, see Sections 7.8 and 6.4).
SelectBest	This selects the best performing set of features from several given classifier results (see Section 6.3).
SimpleDataSplitter	This class splits the given data set, according to the number of instances or different values of a feature.
StToArff	This class converts the original corpus file format of the CoNLL Shared Task to ARFF file format in the given settings (see Section 5.3), creating data features (see Section 5.4).
WekaFilter	This applies any WEKA filter to the given data set.

Evaluation Tasks

AttributeStats	This creates a summary about the number of values of a feature in different data sets.

BootstrapTest	Our own implementation of a bootstrap confidence intervals estimation (see Section 5.6).
EvalClassification	This computes accuracy, precision, recall and F_1 (labeled and unlabeled) for the given target feature (see Section 5.6).
SumEval	This makes a sum of all the evaluations from various files and computes the overall scores, taking the number of instances into account.

B

Contents of the Enclosed CD

The enclosed CD contains the source files and Java binary of our batch processing NLP framework (see Chapter 4), along with several helper scripts used in our analyses. Also included are the configuration files for both the PD and AC subtasks of our SRL setup, training and evaluation phases (see Chapters 6 and 7) and a detailed evaluation of our setup. The CoNLL 2009 corpora are not provided, as we do not own the redistribution rights.

The CD directory structure looks as follows:

`ml-process/`	Our Java batch processing NLP framework:
`ml-process/bin/`	The Java binary, along with installation and usage information
`ml-process/javadoc/`	The documentation for the framework base, as well as all implemented subtasks (cf. also Appendix A)
`ml-process/src/`	The complete Java source files
`ml-process/template/`	A Java class template for subtask implementation
`output/`	Performance statistics of all three variants of our SRL setup, as output by the CoNLL 2009 scorer, and more detailed statistics for the simple post-inference version (see Chapter 8 for explanation)
`scenario/`	Scenario files for AC and PD subtasks, training and evaluation phase
`scripts/`	Various helper scripts:
`scripts/data-inspection/`	Various analysis and statistics functions used to obtain an insight into the nature of the data
`scripts/evaluation/`	Detailed analysis of the SRL system outputs
`scripts/experiments/`	Helper scripts for the execution of experiments
`scripts/predicate-merging/`	Predicate statistics needed for the merging of rare predicates (see Section 7.5)

`scripts/word-clustering/`	A helper script used to obtain a plain-text version of the CoNLL corpora to create the Clusters features using SenseClusters, see Section 5.5
`thesis.pdf`	The complete text of this thesis