# Towards a Machine-Learning Architecture for Lexical Functional Grammar Parsing

Grzegorz Chrupała

A dissertation submitted in fulfilment of the requirements for the award of

Doctor of Philosophy (Ph.D.)

to the

Dublin City University
School of Computing

Supervisor: Prof. Josef van Genabith

April 2008

# Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy (Ph.D.) is entirely my own work, that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

| | |
|---|---|
| Signed | _____ (Grzegorz Chrupała) |
| Student ID | 55130089 |
| Date | April 2008 |

# Contents

# List of Figures

# List of Tables

# Acknowledgments

The work I carried out during the 3 years of my PhD at DCU would not have been possible without the support of many colleagues and friends. First, I'd like to say many thanks to Josef van Genabith who was an enthusiastic supervisor, always interested in my ideas and ready to suggest new ones whenever I got stuck. Josef's endless optimism and positive attitude were a most welcome antidote to my doubts and skepticism.

There are two people who helped shape my thinking and my work in multiple ways: my co-authors and friends Nicolas Stroppa and Georgiana Dinu. I am grateful to Nico for sharing with me his expertise in both the technical details of, and the guiding concepts behind Machine Learning during innumerable coffee breaks. Georgiana served as a tireless sounding board: I would have never been able to fully flesh out my ideas without constantly sharing them with her and hearing what she thought. Georgiana also read parts of the thesis and helped remove many mistakes and unclear points. I would also like to thank both Nicolas and Georgiana for the effort they put in collaborating with me on joint papers: it was a pleasure to work with you.

I would also like to thank the co-members of the GramLab project: Ines Rehbein, Yuqing Guo, Masanori Oya and Natalie Schluter, as well as other researchers at NCLT: Joachim Wagner, Yvette Graham and Jeniffer Foster. Thanks for talking to me, going through the routine of weekly meetings together, listening and giving suggestions at seminar talks and dry-runs! Other researches who I would like to thank for their helpful suggestions and/or generally inspiring conversations are Aoife Cahill, John Tinsley and Augusto Jun Devegili.

Özlem Çetinoğlu helped to make this thesis better by being always ready to listen to me and offer advice. She also proof-read parts of the text and helped to clarify it.

A special round of thanks goes to two of my friends and colleagues at DCU: Bart

Mellebeek and Djamé Seddah. They were great colleagues, always ready to listen and help out with research questions. They are also my best friends, and doing a PhD in Dublin would be a less rewarding and duller experience without all the great times we had together: thanks guys!

I'd like to say a big thank you to Eva Martínez Fuentes who put up with, and even shared and enjoyed the bizarre interests and social life of a PhD student. Thank you for your support and friendship.

The final few months of a PhD program are a notoriously difficult time: they were made much more enjoyable by the endless stimulating chats about science, life and everything with Anke Dietzsch. There is nothing better to renew one's energies than the company of a smart biologist: thank you Anke.

Finally, I would like to express my gratitude to the Science Foundation Ireland who supported my research with grant 04/IN/I527.

**Abstract**

Data-driven grammar induction aims at producing wide-coverage grammars of human languages. Initial efforts in this field produced relatively shallow linguistic representations such as phrase-structure trees, which only encode constituent structure. Recent work on inducing deep grammars from treebanks addresses this shortcoming by also recovering non-local dependencies and grammatical relations. My aim is to investigate the issues arising when adapting an existing Lexical Functional Grammar (LFG) induction method to a new language and treebank, and find solutions which will generalize robustly across multiple languages.

The research hypothesis is that by exploiting machine-learning algorithms to learn morphological features, lemmatization classes and grammatical functions from treebanks we can reduce the amount of manual specification and improve robustness, accuracy and domain- and language -independence for LFG parsing systems.

Function labels can often be relatively straightforwardly mapped to LFG grammatical functions. Learning them reliably permits grammar induction to depend less on language-specific LFG annotation rules. I therefore propose ways to improve acquisition of function labels from treebanks and translate those improvements into better-quality f-structure parsing.

In a lexicalized grammatical formalism such as LFG a large amount of syntactically relevant information comes from lexical entries. It is, therefore, important to be able to perform morphological analysis in an accurate and robust way for morphologically rich languages. I propose a fully data-driven supervised method to simultaneously lemmatize and morphologically analyze text and obtain competitive or improved results on a range of typologically diverse languages.

# Chapter 1

# Introduction

Natural Language Processing (NLP) seeks to develop methods which make it possible for computers to deal with human language texts in a meaningful and useful fashion. Unstructured textual information written by and for humans is ubiquitous and being able to make sense of it in an automated fashion is highly desirable. Many NLP applications can benefit if they are able to automatically associate syntactic and/or semantic structure with natural language text, i.e. to parse it.

## 1.1 Shallow vs Deep Parsing

Traditionally, approaches to parsing within NLP fell into two types. First, parsing can be performed by having expert linguists develop a computational grammar for a given language, which can then be used by a parsing engine to assign a set of analyses to a sentence. Typically, such a grammar would be based on some sufficiently formal and explicit theory of language syntax and semantics, and would provide linguistically well-motivated and rich representations of syntactic structure.

Second, grammars, or more generally parsing models, can be extracted automatically from a large corpus annotated by expert linguists (a treebank). Typically such a grammar would tend to be a relatively simple, relatively theory-neutral, and would provide rather shallow syntactic representations.[1] However it would have access to

[1] In this context, by "shallow parsing" I mean finding a basic constituent structure for a sentence. I do not mean partial parsing, or chunking, where only a simple flat segmentation is imposed on the sentence.

frequency counts of different structures in the training corpus, which can be used for managing ambiguities pervasive in natural language syntax.

## 1.2 Deep Data-Driven Parsing

In more recent years significant effort has been put into overcoming this dichotomy and superseding the tradeoffs it imposes. A number of systems have been developed which combine the use of linguistically sophisticated, rich models of syntax and semantics with the data-driven methodology informed by probability theory and machine-learning. Such "deep data-driven parsing" approaches combine the best of both worlds: they offer wide-coverage and robustness coupled with linguistic accuracy and depth. The developments in this area come in a few flavors.

First, shallow probabilistic models have been "deepened". Many of the complexities which make natural language syntax difficult, such as long-distance dependencies, were ignored in shallow approaches early on; however, this need not be the case: treatment of wh-extraction was incorporated into the Model 3 of Collins parser (Collins, 1997).

Second, many ways have been found to "enrich" the output of shallow parsers with extra information. Examples include adding function labels (to be discussed in Chapter 5) or resolving long-distance dependencies, e.g.: (Johnson, 2001; Levy and Manning, 2004; Campbell, 2004; Gabbard et al., 2006).

Third, parsers using hand-written grammars have been equipped with probabilistic disambiguation models trained on annotated corpora (Riezler et al., 2001; Kaplan et al., 2004; Briscoe and Carroll, 2006). This does not solve the problem of limited coverage those grammars have, but does provide a principled way to rank alternative analyses. Limited coverage has been addressed in these systems by implementing robustness heuristics such as combining partial parses as described by Kaplan et al. (2004).

Finally, standard annotated corpora have been used to train data-driven parsers for deep linguistic formalisms such as Tree Adjoining Grammar (Xia, 1999), Lexical Functional Grammar (Cahill et al., 2002, 2004), Head-driven Phrase Structure Grammar (Miyao et al., 2003; Miyao and Tsujii, 2005) and Combinatory Categorial Grammar (Clark and Hockenmaier, 2002; Clark and Curran, 2004).

## 1.3 Multilingual Treebank-Based LFG

The research described in this thesis was carried out in the context of the GramLab project which aims to develop resources for wide-coverage multilingual Lexical Functional Grammar parsing.

Initial work on data-driven LFG parsing for English was done by Cahill et al. (2002, 2004) at Dublin City University (DCU). LFG has two parallel syntactic representations: constituency trees (c-structures) and representations of dependency relations (f-structures). The DCU approach develops an LFG annotation algorithm which adds information about LFG grammatical functions and other attributes to English Penn II treebank-style trees. These annotations can be used to build LFG-style representations of dependency relations (f-structures). The approach builds LFG representations in two steps: c-structures are constructed by a probabilistic parsing model trained on a treebank, then the trees are automatically annotated and the f-structures are built. It has been demonstrated that this method can successfully compete with parsing systems which use large hand-written grammars developed over many years, on their own evaluation data (Burke et al., 2004a; Cahill et al., 2008).

This empirical success provided the motivation for adapting the approach to other languages. Appropriate training resources, i.e. large, syntactically annotated treebanks are now available for many languages. However, the challenge of multilinguality is not only the availability of resources but also the variation across human languages. Languages differ along a number of dimensions, and often trade off complexity in one linguistic subsystem for simplicity in another.

Computational language processing follows the standard scientific practice of reductionism, and adopts simplifying assumptions about its object of study that may in general be untrue but enable incremental progress to be made. Such simplifications are often unstated and may be difficult to identify until our methods are stress-tested on diverse data. And multilingual processing is one scenario where our assumptions may need to be revised.

One aspect of the research described in this thesis is adapting the DCU treebank-based LFG parsing architecture to the Spanish Cast3LB treebank. This exercise, as well as work on other languages by members of the GramLab project, illuminated

a number of linguistic divergences relevant for processing. The two most relevant divergences between English and a language such as Spanish are along the dimension of configurationality and morphological richness.

While English has highly constrained constituent order, and grammatical function of constituents is highly determined by syntactic configuration, in Spanish the order of main sentence constituents is governed by soft preferences depending on multiple factors, and grammatical function is less predictable from configuration.

The syntactic rigidity of English goes hand in hand with little inflectional morphology. Spanish is morphologically much richer than English (although of course Spanish morphology is still quite limited compared to Slavic languages or to Arabic).

The syntactic flexibility of a language like Spanish makes it problematic to rely heavily on a hand-written annotation algorithm which attempts to assign LFG grammatical function annotations to constituents in a parse tree. What is needed is a method which draws information from many sources, such as local configuration, word order, morphological features, lexical items, semantic features (e.g. animacy) and combines the evidence to arrive at the final decision.

Rich morphology makes it necessary to use a step of morphological analysis more complex than simple Part-of-Speech (POS) tagging prior to syntactic analysis. Accurate morphological analysis is important for a deep lexicalized formalism like LFG where morphological features such as agreement and case are used to constrain possible syntactic analyses, and where normalized, lemmatized forms of lexical items are used to build dependency relations.

Obviously we would like to learn to perform those two tasks, namely assigning grammatical functions to nodes in parse trees and assigning morphological features and lemmas to words in context, from training data for a particular language. Treebanks are annotated with information which can be exploited to learn those tasks: they typically enrich phrase-structure annotation with some grammatical function labels and some semantic role labels. They are also typically morphologically analyzed and lemmatized (and additionally there are other morphologically analyzed corpora that can be used for training).

The driving idea in this thesis is to improve data-driven LFG parsing by making it

more data-driven: learn more, and hardcode less. Learning to reliably assign function labels from training data shifts the weight away from a hand-written LFG annotation algorithm. For a language like Spanish, an annotation algorithm without access to accurate function labels would work very poorly: in this case learning from data is a necessity rather than just an improvement. Similarly, for languages with pervasive inflectional phenomena, accurate and complete morphological analysis is a must. Even though this can, and has been, achieved by hand writing finite-state analysers, here I will adhere to the data-driven approach and determine how much and how well can be learned from annotated data.

## 1.4 Machine Learning

Machine Learning (ML) is the solution to many of the issues outlined in the previous section: supervised learning methods allow us to find in our training data correlations which can be exploited for predicting the phenomena we are interested in, such as a constituent's grammatical function, or the morphological features of a word in context. We extract such hints, or features, from the data, and learn how much and in what way they contribute to the final prediction; in other words we learn the model parameters. When we apply the learned model to new data, we obtain a prediction, possibly with an associated probability or other score indicating how confident we can be in it, which means we have a well-motivated means of predicting combinations of outcomes, such as e.g. sequences of morphological labels, using standard techniques from probability theory.

The most explored setting within supervised machine learning is classification, where the task is to use a collection of labeled training examples in order to learn a function which can predict labels for new, unseen examples. Despite its simplicity this paradigm is remarkably versatile and can be applied to a wide variety of problems. It can also be extended to learn functions with more complex codomains, such as sequences of labels.

The ML algorithms used in this thesis fall into the class of *discriminative* methods, which model the dependence between the unobserved variable $y$ (the output) on the observed variable $x$ (the input); in probabilistic terms they describe the conditional

probability distribution $p(y|x)$, rather than the joint distribution $p(x, y)$ used by generative models. Discriminative approaches allow us to define rich, fine-grained descriptions of the input objects in terms of arbitrary, possibly non-independent features. This makes discriminative modeling flexible and empirically successful in countless domains, including many NLP applications.

In the research described here I use machine learning techniques for classification and for sequence labeling to enhance the two crucial aspects of data-driven LFG parsing discussed in the previous section: function labeling, and morphological analysis.

## 1.5 The Structure of the Thesis

The presentation of my research is organized as follows:

**Chapter 2** gives a brief introduction to the aspects of Lexical Functional Grammar most relevant to parsing natural language, and proceeds to give an overview of existing work on data-driven treebank-based LFG parsing.

**Chapter 3** is a high-level overview of the main aspects of supervised machine learning. I describe feature vector representations, and introduce several commonly used learning algorithms, starting with the Perceptron and continuing with $k$-NN , Maximum Entropy and Support Vector Machines. Finally I briefly discuss approaches to sequence labeling.

**Chapter 5** presents my work on learning models for assigning function labels to parser output. I start by giving a summary of my work on adapting the LFG parsing architecture to Spanish which was the main motivation for developing a classifier-based function labeler. In Section 5.2 I then describe experiments with three ML methods on the Spanish Cast3LB treebank, report the evaluation results and error analysis; I also briefly describe experiments on the more recent AnCora Spanish treebank. In Section 5.3 I describe an improved method of learning a function labeling model by making use of parser output rather than original treebank trees for training, and report evaluation results using such a model on English and Chinese.

**Chapter 6** deals with the task of learning morphological analysis models for languages with rich inflectional morphology. I start by reviewing existing research on supervised learning of morphology. I discuss in some detail approaches based on Inductive Logic Programming (ILP) and Analogical Learning (AL), as well as a number of other methods. I introduce a classifier-based method to learn lemmatization models by means of using edit scripts between form-lemma pairs as class labels to be learned. I report on experiments using this method on data from six languages. I proceed to introduce the **Morfette** system which uses the Maximum Entropy approach to learn a morphological tagging model and a lemmatization model and combines their predictions to assign a sequence of morphological tags and lemmas to sentences. I report on experiments using this system on Spanish, Romanian and Polish. Finally, I compare the performance of the classifier-based method to morphological analysis and synthesis with an ILP implementation Clog on data from the Multext-EAST corpus.

**Chapter 7** summarizes the main contributions of this thesis and discusses ideas for refining and extending the research described in the preceding chapters.

## 1.6 Summary of Main Results

The main results described in Chapters 5 and 6 are the following:

**Spanish treebank-based LFG parsing**

- I have overhauled and substantially extended the range of phenomena treated in the Spanish annotation algorithm. I also revised and extended the gold standard which now includes 338 f-structures. This served two purposes: to identify areas where the existing LFG parsing architecture for English needed further work to make it less language dependent and more portable, and to enable the work on developing and evaluating a function labeling model for Spanish.

**Function labeling**

- I have developed a function labeler for Spanish which achieves a relative error reduction of 26.73% over the previously used method of using the c-structure

parser to obtain function-labeled trees. The use of this model in the LFG parsing pipeline also improves the f-structure quality as compared to the baseline method.

- I have described a training regime for an SVM-based function labeling model where trees output by a parser are used in combination with treebank trees in order to achieve better similarity between training and test examples. This model outperforms all previously described function labelers on the standard English Penn II treebank test set (22.73% relative error reduction over previous highest score).

**Morphological analysis**

- I have developed a method to cast lemmatization as a sequence labeling task. It relies on the notion of edit script which encodes the transformations needed to perform on the word form to convert it into the corresponding lemma. A lemmatization model can be learned from a corpus annotated only with lemmas, with no explicit part-of-speech information.

- I have built the **Morfette** system which performs morphological analysis by learning a morphological tagging model and a lemmatization model, and combines the predictions of those two models to find a globally good sequence of MSD-lemma pairs for a sentence.

- I have shown that integrating information from morphological dictionaries into the Maximum Entropy models used by **Morfette** is straightforward and can substantially reduce error, especially on words absent from training corpus data.

- I have developed an instantiation of the edit script, the Edit Tree, which improves lemmatization class induction in the case where inflectional morphology affects word beginnings in addition to word endings, and have shown that the use of this edit script version results in statistically significant error reductions on test data in Polish, Welsh and Irish.

- I compared the proposed morphology models against existing systems (Freeling and Clog): in both cases my proposed models showed superior or competitive performance

# Chapter 2

# Treebank-Based Lexical Functional Grammar Parsing

In this chapter I provide an overview of the Lexical Functional Grammar (LFG) and discuss approaches to parsing natural language within the LFG framework. I will concentrate on the aspects of LFG most relevant to computational implementations.

## 2.1 Lexical Functional Grammar

Lexical Functional Grammar is a formal theory of language introduced by Bresnan and Kaplan (1982) and further described in (Bresnan, 2001; Dalrymple, 2001). The main focus of theoretical linguistics research within LFG has been syntax. LFG syntax consists of two levels of structure.

**C-structures** The constituent structure (c-structure) is a representation of the hierarchical grouping of words into phrases. It is used to represent constraints on word order and constituency; the concept of c-structure corresponds to the notion of context-free-grammar parse-tree used in formal language theory.

**F-structures** The level of functional structure (f-structure) describes the grammatical functions of constituents in sentences, such as subject, direct object, sentential complement or adjunct. F-structures are more abstract and less variable between lan-

| Attribute | Meaning |
|---|---|
| SUBJ | subject |
| OBJ | direct object |
| $\text{OBJ}_2$ | indirect object (also $\text{OBJ}_\theta$ ) |
| OBL | oblique or prepositional object |
| COMP | sentential complement |
| XCOMP | non-finite clausal complement |
| ADJUNCT | adjunct |

Table 2.1: LFG Grammatical functions

guages than c-structures. They can be thought of as providing a syntactic level close to the semantics or the predicate-argument structure of the sentence. F-structures are represented in LFG by attribute-value matrices. The attributes are atomic symbols; their values can be atomic, they can be *semantic forms*, they can be f-structures, or they can be sets of f-structures, depending on the attribute. Formally f-structures are finite functions whose domain is the set of attributes and the codomain is the set of possible values. Table 2.1 lists the grammatical functions most commonly assumed within LFG.

Those two levels of syntactic structure are related through the so-called projection architecture. Nodes in the c-structure are mapped to f-structures via the many-to-one projection function $\phi$.

**Functional equations** An LFG grammar consists of a set of phrase structure rules and a set of lexical entries, which specify the possible c-structures. Both the phrase structure rules and the lexical entries are annotated with functional equations, which specify the mapping $\phi$. The functional equations employ two meta-variables, $\downarrow$ and $\uparrow$ which refer to the f-structure associated with the current (self) node and the f-structure associated with its mother node, respectively. The $=$ symbol in the functional equations is the standard unification operator.

$$\begin{array}{lcccc} \text{S} & \longrightarrow & \text{NP} & & \text{VP} \\ & & (\uparrow \text{SUBJ}) = \downarrow & & \uparrow = \downarrow \end{array} \tag{2.1}$$

The phrase structure rule in (2.1) is interpreted as follows: the node S has a left daughter NP and a right daughter VP, the f-structure associated with S unifies with the f-structure for VP, while the value of the SUBJ attribute of the f-structure for S unifies with the f-structure associated with the NP.

The notation ($f$ SUBJ) denotes the f-structure $f$ applied to the attribute SUBJ, i.e. the value of that attribute in $f$. Function application is left-associative so ($f$ XCOMP SUBJ) is the same as (($f$ XCOMP) SUBJ) and denotes the value of the SUBJ attribute in the f-structure ($f$ XCOMP).

Figure 2.1 shows the c-structure and the f-structure for the English sentence *But stocks kept falling*. The nodes in the c-structure are associated with functional equations. The equations on the phrasal nodes come from the phrase-structure rules; the ones on the terminals come from lexical entries. The accompanying f-structure is the minimal f-structure satisfying the set of constraints imposed by this set of equations. Two of the sub-f-structures are connected with a line; this notation is a shorthand signifying that the f-structures are identical.

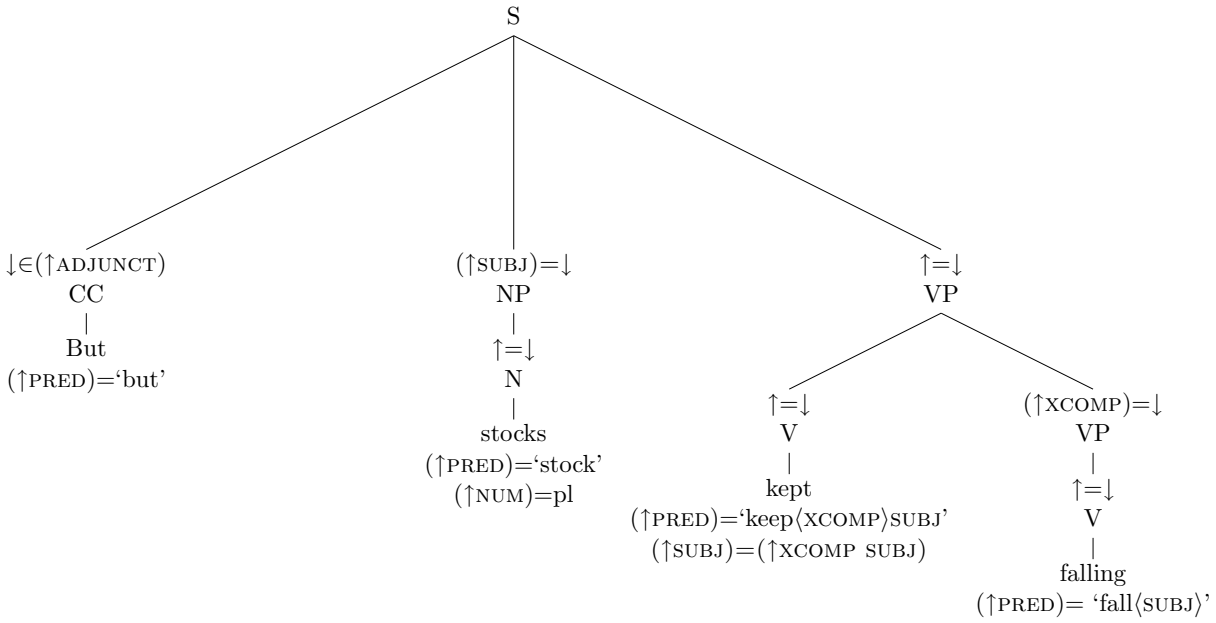
**Semantic forms** The values of the PRED attribute are so called *semantic forms*: however, rather than representing semantics they correspond to subcategorization frames for lexical items. They encode the number and the grammatical function of the syntactic arguments the lexical item requires. For example 'fall⟨SUBJ⟩'[1] means that *fall* needs one argument, with the grammatical function SUBJ. Semantic forms are uniquely instantiated, i.e. they should be understood as having an implicit index: only semantic forms with an identical index are considered equal. This ensure that semantic forms corresponding to two distinct occurrences of a lexical item in a sentence cannot be unified. For example in the f-structure for the sentence:

(2.2) The big fish devoured the little fish.

the two semantic forms 'fish'$_1$ and 'fish'$_2$ are distinct and cannot be unified.

The line connecting two f-structures to signify that they are identical also implies that the implicit indices in the semantic forms are identical.

[1] In 'keep⟨XCOMP⟩SUBJ' the SUBJ function is outside the brackets: this notation is used to indicate that the "raised" subject, which *keep* shares with its XCOMP argument; *keep* does not impose semantic selectional restrictions on this raised subject.




$$\begin{bmatrix} \text{ADJUNCT} & \left\{\begin{bmatrix}\text{PRED} & \text{'but'}\end{bmatrix}\right\} \\ \text{SUBJ} & \begin{bmatrix}\text{PRED} & \text{'stock'} \\ \text{NUM} & \text{PL}\end{bmatrix} \\ \text{PRED} & \text{'keep}\langle\text{XCOMP}\rangle\text{SUBJ'} \\ \text{XCOMP} & \begin{bmatrix}\text{SUBJ} & [\quad] \\ \text{PRED} & \text{'fall}\langle\text{SUBJ}\rangle\text{'}\end{bmatrix} \end{bmatrix}$$

Figure 2.1: LFG representation of *But stocks kept falling*

**Well-formedness of f-structures** F-structures have three general well-formedness conditions imposed on them (following Bresnan and Kaplan (1982)).

**Completeness** An f-structure is locally complete iff it contains all the governable grammatical functions that its predicate subcategorizes for. An f-structure is complete iff all its sub f-structures are locally complete. Governable grammatical functions correspond to possible types of syntactic arguments and include SUBJ, OBJ, $\text{OBJ}_2$, XCOMP, COMP, OBL.

**Coherence** An f-structure is locally coherent iff all its governable grammatical functions are subcategorized for by its local predicate. An f-structure is coherent iff all its sub f-structures are locally coherent.

**Consistency** In a given f-structure an attribute can have only one value.[2]

Together these constraints ensure that all the subcategorization requirements are satisfied and that no non-governed grammatical functions occur in an f-structure.

**Long-distance dependencies and functional uncertainty** Some phenomena in natural languages such as topicalization, relative clauses and wh-questions introduce long distance dependencies. Those are constructions where a constituent can be arbitrarily distant from its governing predicate.

(2.3) $\text{What}_1$ did she never suspect she would have to deal with $\diamond_1$?

In an LFG analysis of (2.3) the interrogative pronoun *what* has the grammatical function FOCUS in the top-level f-structure and at the same time the function OBJ in the embedded f-structure corresponding to the prepositional phrase introduced by *with* at the end of the sentence. In principle an unbounded number of tensed clauses can separate the interrogative pronoun from its governing predicate.

In order to express such constraints involving unbounded embeddings, LFG resorts to functional equations with paths through the f-structures written as regular expressions. Such equations are referred to as *functional uncertainty equations*. For example to express the constraint that the value of the FOCUS attribute is equal to the value of

[2]This constraint follows automatically if we regard f-structures as functions.

the OBJ attribute arbitrarily embedded in a number of COMPs or XCOMPs one would write

$$(f \text{ FOCUS}) = (f \{\text{COMP} \mid \text{XCOMP}\}^* \text{ OBJ}) \ .$$

The vertical bar operator | is indicates the disjunction of two expressions, while the Kleene star * operator has the standard meaning of a string of 0 or more of the preceding expressions.

## 2.2 LFG parsing

In this section I briefly review common approaches to parsing natural language with LFG grammars and then describe in some detail the wide-coverage treebank-based LFG acquisition methodology developed at DCU. This will serve as background to my own work on integrating machine learning techniques within this approach.

Computational implementations of LFG and related formalisms such as Head-Driven Phrase-Structure Grammar (HPSG) are sometimes described as *deep grammars*. This term highlights the fact that computational work within these frameworks aims at parsing natural language text into information-rich, linguistically plausible representations which account for complex phenomena such as control/raising and long-distance dependencies. They provide a level of syntax abstract and rich enough for interfacing with semantics. Until relatively recently, data-driven methods for processing language, such as parsers based on Probabilistic Context Free Grammars (PCFG), did not provide such rich structures but rather more "shallow", "surfacy" representations such as basic constituency trees.

The level of f-structures in LFG is intermediate between a basic constituency tree and a semantic representation. The higher level of abstraction as compared to c-structures can be useful for applications such as e.g. Question Answering, where we would like to have access to some approximation of argument structure. Since the f-structures abstract over surface word order they are more appropriate for this purpose: e.g. two English sentences differing only in adverb placement will receive the same f-structure representation even though their c-structures differ. This benefit is even more pronounced in languages with flexible constituent order, where e.g. core verb

arguments can appear pre- or postverbally. Additionally, at f-structure level, many dependencies between predicates and their displaced arguments, such as in questions, relative clauses or topicalization, are resolved, which further eases the task of matching similar meanings expressed by means of alternative constructions.

Initial work on parsing with deep grammars was based on hand-writing the grammars and using a parsing engine specialized to the grammatical formalism in question to process sentences. In the context of LFG, the Pargram project (Butt et al., 2002) has been developing wide-coverage hand-written grammars for a number of languages, using the XLE parser and grammar development platform (Maxwell and Kaplan, 1996). Such grammars have been subsequently coupled with stochastic disambiguation models trained on annotated treebank data which choose the most likely analysis from among the ones proposed by the parser (Riezler et al., 2001; Kaplan et al., 2004).

### 2.2.1 Treebank-based LFG parsing

Hand-written LFG grammars such as those developed for the Pargram project can offer relatively wide coverage. However, their development takes a large amount of time dedicated by expert linguists, and the coverage still falls short in comparison to that of shallower, probabilistic parsers which use treebank grammars.

This bottleneck caused by manual grammar writing has motivated an alternative approach to deep parsing, inspired by probabilistic treebank-based parsers. The idea is to exploit a treebank and automatically convert it to a deep-grammar representation. Most research in this framework has used the English Penn II treebank (Marcus et al., 1994). In addition to constituency trees this treebank employs a number of extra devices to provide information necessary for the recovery of predicate-argument-adjunct relations. The most important ones are traces coindexed with phrase structure nodes, and function labels indicating grammatical functions and semantic roles for adjuncts.

Early work on converting the Penn treebank to a deep-grammar representation and using this resource to build a data-driven deep parser was carried out within the Tree Adjoining Grammar (TAG) formalism (Xia, 1999). Subsequently, similar resources were developed for other grammar formalisms: LFG (Cahill et al., 2002, 2004), HPSG (Miyao et al., 2003; Miyao and Tsujii, 2005) and Combinatory Categorial Grammar

(CCG) (Clark and Hockenmaier, 2002; Clark and Curran, 2004).

### DCU LFG parsing architecture

The treebank-based parsing research within the HPSG and CCG frameworks follows a similar pattern: the original treebank trees are semi-automatically corrected and modified to make them more compatible with the target linguistic representations. Then a conversion algorithm is applied to the treebank trees, and produces as a results a collection of HPSG signs or CCG derivations. This transformed treebank is then used to extract a grammar and train a stochastic disambiguation model which works on packed chart representations (feature forests (Miyao and Tsujii, 2002, 2008)) and chooses the most likely parse from among the ones proposed by a dedicated HPSG or CCG parser.

The projection architecture of LFG with the two levels of syntactic representation linked via functional annotations on phrase structure rules facilitates an alternative, more modular implementation strategy. The parsing process is divided into two steps: c-structure parsing and f-structure construction.

**Treebank annotation** A key component in the DCU LFG parsing architecture is the LFG annotation algorithm. It is a procedure which walks the c-structure trees and annotates each node with functional equations. The result is an annotated c-structure tree such the one depicted in Figure 2.1. Of course the structure of the tree underdetermines the set of constraints that defines the corresponding f-structure, so the annotation algorithm uses additional sources of information to produce the equations:

**Head table.** This table specifies, for each local subtree of depth one, which constituent is the head daughter. Similar tables are used in treebank-based lexicalized probabilistic parsers, and the annotation algorithm for the English Penn treebank uses an adapted version of the head table from Magerman (1994).

**Function labels.** Function labels in the English Penn treebank annotate some nodes with their grammatical function, and label some adjuncts with semantic roles. Grammatical function labels are very useful since they can be mapped straightforwardly to LFG functional equations.

**Coindexed traces.** Traces in the English Penn treebank provide information necessary to recover predicate-argument structure, identify control/raising constructions and resolve long-distance dependencies.

**Integrated and pipeline models** There are two alternative approaches to LFG parsing within the general DCU architecture. The **integrated model** works as follows. The original treebank trees are annotated with functional equations. This collection of annotated trees is used to train a PCFG parser or a lexicalized probabilistic parser such as (Collins, 1999; Charniak, 2000; Charniak and Johnson, 2005). The functional-equation-annotated nodes are treated as atomic phrase labels and thus the parser learns to output trees with such labels. To process new text, the annotated-treebank-trained model is used to produce a tree. Then the function equations encoded on the labels are collected and evaluated using a dedicated LFG constraint solver, which produces the f-structure they define.

The **pipeline model** takes a more modular approach. The c-structure parsing model (again using some off-the-shelf data driven parsing engine) is trained on original treebank trees. When processing a new sentence, it is first parsed into a basic c-structure tree. The annotation algorithm is run on this tree, and the resulting equations are again evaluated to obtain an f-structure. The bare c-structure tree does not contain function labels or traces – the annotation algorithm will still work without those but may be less accurate. For this reason there is a module which adds function labels to the c-structure tree.

For both the integrated and pipeline models there is a non-local dependency (NLD) resolution module which deals with non-local phenomena such as raising/control contructions and long distance dependencies. Figure 2.2 illustrates the complete LFG parsing architecture in the pipeline version. In the work described in the rest of this thesis I always assume the pipeline architecture: its modular design makes it easy to improve specific components in a piecewise fashion, independently of each other. By breaking up the task it also reduces model size and permits more fine-grained control over the features used for each component.

Figure 2.2: Pipeline LFG parsing architecture

**Morphological analyzer** The first module in the pipeline is the morphological analyzer. For English, which has a reduced amount of inflectional morphology it is a simple dictionary which associates word forms with their POS tags and lemmas. POS-tagging is either integrated in the c-structure parser, or an external POS-tagger may be used. The lemmas are looked up in the dictionary while running the annotation algorithm since they are needed to construct the semantic forms. For morphologically richer languages a more sophisticated morphology module can be beneficial: Chapter 6 describes the development of such a module for use with the LFG parsing pipeline.

**C-structure parsing** A c-structure parser can be any data-driven statistical parser which can be trained on a treebank. This approach allows us to leverage advances in parsing by using state-of-the-art components such as the parser of Charniak and Johnson (2005). On the other hand, the use of c-structure parser within a pipeline means that decisions are taken early, and if this component chooses a wrong tree, this mistake will not be undone in later processing stages.

**Function labeler** C-structures labeled with function labels allow the annotation algorithm to produce more accurate functional equations. For some languages and treebanks they are even more important than for English – if c-structures are flat, most syntactic information resides in the grammatical function labels. Rich function labeling also reduces the amount of work that needs to be done within the f-structure annotation algorithm, since it can simply exploit the straightforward mapping from function labels to LFG annotations. For those reasons it is highly desirable to have an accurate data-driven function label model. Chapter 5 discusses research on developing such a model for the Spanish treebank and using it for Spanish LFG parsing. Additionally it introduces a high-performing functional labeler for English, which is also trained and evaluated on Chinese treebank data.

**NLD module** Prior to the application of the NLD module the LFG parser outputs so-called proto f-structures. The grammatical functions used to analyze wh-questions, relatives and topicalization, TOPIC and FOCUS are not resolved, i.e. they are not identified with f-structures at the level where they fulfill subcategorization requirements of their governing predicate. The NLD module resolves those dependencies; the module is described in detail in Cahill et al. (2004). In brief, the possible resolution candidates are generated, and they are ranked according to the product of two scores: the probability of the subcategorization frame given the lemma, and the probability of the path through f-structure from the source grammatical function to the target (or a finite approximation of a functional uncertainty equation), given the source grammatical function. Both conditional probabilities are obtained from f-structures generated for treebank trees, via Maximum Likelihood estimates.

## 2.3 GramLab – Treebank-Based Acquisition of Wide-Coverage LFG Resources

The approach to LFG grammar acquisition and parsing outlined above has been applied mainly to the English Penn treebank. The aim of the GramLab project is to attempt to port this architecture to other languages and treebanks. A successful adaptation of the method would provide valuable NLP resources for those languages and would also potentially enable multilingual applications such as cross-language information retrieval or data-driven transfer-based machine translation.

The challenge is to investigate to what degree the methodology developed for English will work in the context of languages with possibly quite different characteristics. The payoff is that we learn how to make language processing more robust in the face of the variety characterizing human languages. Research withing GramLab has investigated treebank-based LFG parsing for Japanese (Oya and van Genabith, 2007), Chinese (Burke et al., 2004b; Guo et al., 2007), German (Cahill et al., 2005), French (Schluter and van Genabith, 2007), Arabic (Al-Raheb et al., 2006) and Spanish (O'Donovan et al., 2005; Chrupała and van Genabith, 2006a,b).

The research described in this thesis has been carried out within the GramLab

project. I have investigated the issues arising when adapting the DCU approach to LFG parsing to the Spanish Cast3LB treebank. The insights learned from this work have informed my research on applying machine-learning methods to develop robust language-independent morphology and function labeling modules and thus minimizing the effort that has to be devoted to developing the highly language-specific LFG annotation algorithms.

# Chapter 3

# Machine Learning

## 3.1 Introduction

In this chapter I provide a brief introduction to the field of supervised machine-learning and give an overview of several machine-learning algorithms useful in Natural Language Processing. In Section 3.2 I present four algorithms used for classification: Perceptron, $k$-nearest-neighbors, MaxEnt and Support Vector Machine. In Section 3.3 I briefly discuss approaches to the sequence labeling task.

### 3.1.1 Supervised learning

In supervised learning the goal is to learn a function

$$h : \mathcal{X} \to \mathcal{Y} \tag{3.1}$$

where $x \in \mathcal{X}$ are inputs and $y \in \mathcal{Y}$ are outputs. The input objects are called instances, or examples, and they can be any kind of object, depending on the particular learning task: in NLP they could be for example documents to classify, strings of words to tag with POS-sequences or sentences in the source language to translate into the target language. Depending on the nature of the output space $\mathcal{Y}$, learning tasks can be categorized into several types:

- Binary classification: $\mathcal{Y} = \{-1, +1\}$
- Multiclass classification: $\mathcal{Y} = \{1, \ldots, K\}$ (finite set of labels)

- Regression: $\mathcal{Y} = \mathbb{R}$
- Structured prediction: here the outputs in $\mathcal{Y}$ are complex. For example, in a sequence labeling task such as POS-tagging, $\mathcal{Y} = \{1, \ldots, K\}^n$, i.e. the output is a sequence of labels of length $n$ equal to the length of the input string.

### 3.1.2 Feature representation

The prediction is based on the *feature function* $\Phi : \mathcal{X} \rightarrow \mathcal{F}$. The function $\Phi$ takes an input object and extracts features which are useful in predicting the output. Generally the feature space $\mathcal{F}$ most common in machine learning is $\mathcal{F} = \mathbb{R}^D$, i.e. $D$-dimensional real vector space. Specifically in NLP, the features will typically either be binary or will be symbols rather than numbers; the details depend on the learning algorithm.

#### Feature binarization

For algorithms which require binary features, we can extract symbolic features from instances and then *binarize* the output vectors of symbols. A common way of binarizing features involves mapping each feature-value pair to a new feature and assigning it 1 if it is active and 0 otherwise. Thus for each original symbolic feature $i$ we create as many new binary features as the number of possible values for feature $i$; one of them is set to 1, while all the others are 0. This gives rise to sparse binary vectors with few non-zero elements. For a feature vector $\mathbf{x}$ of length $d$ the corresponding binary vector $\mathbf{x}'$ is given by:

$$\mathbf{x}' = ([\![x_1 = V_{1_1}]\!], [\![x_1 = V_{1_2}]\!], ..., [\![x_1 = V_{1_n}]\!], ..., ..., [\![x_d = V_{d_1}]\!], ..., [\![x_d = V_{d_m}]\!]) \quad (3.2)$$

where the $j^{\text{th}}$ element of the set of possible values for the $i^{\text{th}}$ feature is $V_{i_j}$ and where

$$[\![p]\!] = \begin{cases} 1 \text{ if } p \text{ is true} \\ 0 \text{ otherwise .} \end{cases}$$

In the rest of this chapter I will concentrate on two learning tasks: classification, that is learning to assign a label from a (small) finite set to examples, and sequence

labeling, that is assigning sequences of such labels to examples which are typically strings of words.

## 3.2 Classification

### 3.2.1 Perceptron

One of the simplest classification algorithms is the perceptron (Rosenblatt, 1958). Like more complex algorithms presented later in this chapter (MaxEnt and SVMs), it is a linear classifier; i.e. in the case of binary classification, it learns the hyperplane separating the positive and the negative examples in a multidimensional feature space. I describe it here as a basic, easy-to-understand instance of a linear hyperplane-based classification algorithm.

The separating hyperplane is defined by a weight vector $\mathbf{w}$ of size $d$ and the bias $b$: the weights $w_0, w_1, ..., w_d$ and the bias $b$ correspond to the hyperplane equation $w_0x_0 + w_1x_1 + ... + w_dx_d + b = 0$.

The decision function assigning the example to either the positive or negative class has the following form:

$$f(x, \mathbf{w}, b) = \text{sign}(\mathbf{w} \cdot \Phi(x) + b) = \text{sign}\left(\sum_{i=1}^{d} w_i \Phi(x)_i + b\right) \tag{3.3}$$

That is, if the dot product of the weight vector and the feature vector of example $x$ (plus bias $b$) is $> 0$ the example is classified as positive, if it is $< 0$ it is classified as negative.

The learning problem thus consists in learning the parameters (the weights and the bias) from the set of training examples. The perceptron is an online learning algorithm, i.e. it processes one training example at a time. Initially the parameters are set to zero. If the current example is correctly classified by the current parameters then the algorithm proceeds to the next step. If the example is misclassified, the parameters are updated so that it is correctly classified. The algorithm iterates over the training examples until no further updates are necessary. The algorithm eventually converges to parameter settings that correctly classify the whole training set (if the data is linearly

PERCEPTRON$(x_{1:N}, y_{1:N}, I)$:

1: $\mathbf{w} \leftarrow \mathbf{0}$; $b \leftarrow 0$
2: $\mathbf{w}_a \leftarrow \mathbf{0}$; $b_a \leftarrow 0$
3: $c \leftarrow 0$
4: **for** $i = 1...I$ **do**
5:  **for** $n = 1...N$ **do**
6:  **if** $y_n(\mathbf{w} \cdot \Phi(x_n) + b) \leq 0$ **then**
7:  $\mathbf{w} \leftarrow \mathbf{w} + y_n \Phi(x_n)$; $b \leftarrow b + y_n$
8:  $\mathbf{w}_a \leftarrow \mathbf{w} + c y_n \Phi(x_n)$; $b_a \leftarrow b_a + c y_n$
9:  $c \leftarrow c + 1$
10: **return** $(\mathbf{w} - \mathbf{w}_a/c, b - b_a/c)$

Figure 3.1: Averaged Perceptron algorithm

Figure 3.2: Example separating hyperplanes in two dimensions

separable).

A modification to the basic algorithm, the AVERAGED PERCEPTRON is able to achieve better generalization to unseen examples. The final parameters which the algorithm returns are the average of all the hypothesized parameters encountered during the algorithm run. An efficient implementation of the averaged perceptron algorithm is shown in Figure 3.1 (Daumé III, 2006). It works similarly to the basic version, but in addition to current parameters $(\mathbf{w}, b)$, the averaged parameters $(\mathbf{w}_a, b_a)$ are maintained (see lines 2 and 8). When an example is incorrectly classified, in line 8 those parameters are updated, but the update is multiplied by the averaging count $c$. Finally in line 10, the algorithm returns $(\mathbf{w} - \mathbf{w}_a/c, b - b_a/c)$, which corresponds to the average of all the values the parameters $(\mathbf{w}, b)$ took.

If the data is linearly separable, there are obviously an infinite number of hyperplanes which will separate the training examples. The solution found by the perceptron algorithm depends on the order in which the examples are processed. Figure 3.2 shows three solutions to the problem of separating the positive examples (blank points) from the negative examples (filled points) found by the averaged perceptron algorithm described above. Intuitively, some lines classify better than others: for example the dashed blue line seems to be a better solution than the solid red line. This intuition can be

conceptualized as the notion of a margin: we want to find solutions which maximize the distance between the separating hyperplane and the training examples. It has been shown that a linear classifier's generalization error to unseen test data is proportional to the inverse of the margin (Vapnik, 2006; Freund and Schapire, 1998). The perceptron algorithm does not maximize margin; an online learning algorithm based on the perceptron idea which does is the Margin Infused Relaxed Algorithm (MIRA) (Crammer and Singer, 2003). The most common maximum-margin algorithm is Support Vector Machine discussed in Section 3.2.4.

The version of perceptron presented here can only deal with linear classification. This limitation can be lifted by using the algorithm in conjunction with the "kernel trick" (Aizerman et al., 1964); kernels are discussed in Section 3.2.4 in connection with Support Vector Machines.

### 3.2.2 K-NN

Another simple classification learning method is the $k$-nearest-neighbors algorithm (Fix and Hodges, 1951; Cover and Hart, 1967). The idea is to assign to a new example the class label associated with the majority of instances in its neighborhood. The neighborhood is determined by the distance in the multidimensional feature space induced by the feature vectors representing the instances. The parameter $k$ specifies how many nearest instances form the neighborhood.

In the case of real-valued features Euclidean distance is used, i.e. the distance $\Delta(x, x')$ between instances $x$ and $x'$ is:

$$\Delta(x, x') = \sqrt{\sum_{i=1}^{d} (\Phi(x)_i - \Phi(x')_i)^2}. \tag{3.4}$$

In NLP $k$-NN is frequently used with symbolic features, which may encode word forms, characters, morphological features and other non-numeric attributes. In this case the most basic distance metric is the Hamming distance, also called overlap metric or L1 metric. It defines the distance between two instances to be the sum of per-feature distances; for symbolic features the per-feature distance is 0 for an exact match and 1

for a mismatch.

$$\Delta(x, x') = \sum_i \delta(\Phi(x)_i, \Phi(x')_i) \tag{3.5}$$

where

$$\delta(\Phi(x)_i, \Phi(x')_i) = \begin{cases} 0 & \text{if } \Phi(x)_i = \Phi(x')_i \\ 1 & \text{if } \Phi(x)_i \neq \Phi(x')_i \end{cases} \tag{3.6}$$

For a vector with a mixture of symbolic and numeric values, the above definition of per feature distance is used for symbolic features, while for numeric ones we use the scaled absolute difference (Daelemans and van den Bosch, 2005):

$$\delta(\Phi(x)_i, \Phi(x')_i) = \frac{\Phi(x)_i - \Phi(x')_i}{max_i - min_i}. \tag{3.7}$$

The $k$-NN algorithm modified to use this distance metric is referred to as IB1 (Aha et al., 1991). Daelemans and van den Bosch (2005) interpret the $k$ parameter differently from the traditional meaning: instead of $k$ nearest neighbors they consider neighbors at $k$ nearest distances. This makes a difference in the case where more than one instance has the same distance to the test instance.

**Feature weighting**

It is very common to use the IB1 algorithm with some feature weighting method, where the per-feature distance is multiplied by the weight of the feature for which it is computed. That is:

$$\Delta(x, x') = \sum_i w_i \delta(\Phi(x)_i, \Phi(x')_i) \tag{3.8}$$

where $w_i$ is the weight of the $i^{\text{th}}$ feature. There are many ways to find a good weight vector $\mathbf{w}$. Daelemans and van den Bosch (2005) describe two entropy-based methods and a $\chi^2$-based method.

**Information gain** Information gain is a measure of how much knowing the value of a certain feature for an example decreases our uncertainty about its class, i.e. it is the difference in class entropy with and without information about the feature value.

$$w_i = H(Y) - \sum_{v \in V_i} P(v) \times H(Y|v) \tag{3.9}$$

where $w_i$ is the weight of the $i^{\text{th}}$ feature, $Y$ is the set of class labels, $V_i$ is the set of possible values for the $i^{\text{th}}$ feature, $P(v)$ is the probability of value $v$, and class entropy is $H(Y) = -\sum_{y \in Y} P(y) \log_2 P(y)$, while $P(Y|v)$ is the conditional class entropy given that the feature value is $v$.[1]

**Gain ratio** Information gain tends to assign excessive weight to features with a large number of values. For example if each instance in the union of the training set and test set has a unique value for a certain feature, then knowing the value of this feature gives us certainty as to the class label for the instances in the training set. However it is useless for predicting the class of a test instance as there are no training instances with the same value for this feature. To remedy this bias information gain can be normalized by the entropy of the feature values, which gives the gain ratio:

$$w_i = \frac{H(Y) - \sum_{v \in V_i} P(v) \times H(Y|v)}{H(V_i)} \tag{3.10}$$

For a feature with a unique value for each instance in the training set, the entropy of the feature values in the denominator will be maximally high, and will thus give a low weight for this feature.

**Chi-squared** Daelemans and van den Bosch (2005) adapt the $\chi^2$-based attribute selection method proposed by White and Liu (1994) as an alternative to information-theoretic methods. The following equation defines the $\chi^2$ statistic for a problem with $k$ classes and $m$ values for feature $F$:

$$\chi^2 = \sum_{i=1}^{k} \sum_{j=1}^{m} \frac{E_{ij} - O_{ij}}{E_{ij}} \tag{3.11}$$

where $O_{ij}$ is the observed number of instances with the $i^{\text{th}}$ class label and the $j^{\text{th}}$ value of feature $F$. $E_{ij}$ is the expected number of such instances in case the null hypothesis, i.e. that the feature $F$ does not predict the class, is true. The expected value is defined

[1] Numeric values need to be temporarily discretized for this to work.

as:

$$E_{ij} = \frac{n_{\cdot j} n_{i\cdot}}{n_{\cdot\cdot}} \tag{3.12}$$

where $n_{ij}$ is the frequency count of instances with the $i^{\text{th}}$ class label and the $j^{\text{th}}$ value of feature $F$, and

$$n_{\cdot j} = \sum_{i=1}^{k} n_{ij} \tag{3.13}$$

$$n_{i\cdot} = \sum_{j=1}^{m} n_{ij} \tag{3.14}$$

$$n_{\cdot\cdot} = \sum_{i=1}^{k} \sum_{j=0}^{m} n_{ij} \tag{3.15}$$

i.e. the total number of instances. Daelemans and van den Bosch (2005) propose to either use the $\chi^2$ values as feature weights in Equation 3.8, or alternatively to use the shared variance measure:

$$SV_F = \frac{\chi_F^2}{N \times (min(k, m) - 1)} \tag{3.16}$$

where $k$ is the number of classes, $m$ the number of values for feature $F$ and $N$ the number of instances.

### Distance-weighted class voting

In the basic version of the $k$-NN algorithm all the instances in the neighborhood are weighted equally for computing the majority class to be assigned to a new instance. However, we may want to treat the votes from very close neighbors as more important than votes from more distant ones. A variety of distance weighting schemes have been proposed to implement this idea; see (Daelemans and van den Bosch, 2005) for details and discussion.

The $k$-NN algorithm, unlike the perceptron, is not a linear classifier, i.e. it does not depend on the assumption that the data is linearly separable. The basic $k$-NN and its various modifications have been referred to as lazy learners or memory-based learners. During learning little "work" is done by the algorithm: the training instances are simply stored in memory in some efficient manner. It is during prediction that most of the

actual computation takes place: the test instance is compared to the training instances, the neighborhood is calculated, and the majority label assigned. In $k$-NN no abstraction is performed, the model generalizes based on directly comparing the test instance with labeled training examples. No information is discarded, all the "exceptional" and low frequency items are still available for informing the prediction.

The $k$-NN algorithm is one of the Machine Learning methods used for function labeling experiments for Spanish in Section 5.2.

### 3.2.3 Logistic Regression and MaxEnt

Maximum Entropy (or MaxEnt) models are linear probabilistic classifiers commonly used in NLP. In multiclass classification they output probability distribution over class labels. MaxEnt models correspond to logistic regression models, but are derived in an alternative way. In this section I introduce linear and logistic regression and then present the MaxEnt classifier as typically used in NLP.

#### Linear regression

In linear regression models the prediction function $h$ introduced in Equation 3.1 is instantiated as $h : \mathcal{X} \rightarrow \mathbb{R}$, i.e. we try to build models which predict outcomes in the set of real numbers based on example objects, or observations, in $\mathcal{X}$. The prediction is based on the features, or predictors, which are also typically real numbers. The feature function $\Phi$ maps observations to vectors of predictors, i.e. $\Phi : \mathcal{X} \rightarrow \mathbb{R}^d$. The model is defined by the equation:

$$y = w_0 + \sum_{i=1}^{d} w_i \Phi(x)_i \tag{3.17}$$

where $y$ is the outcome, $\Phi(x)_1..\Phi(x)_d$ are the feature values, $w_1..w_d$ are the feature weights, and $w_0$ is the intercept (or bias). We can eliminate $w_0$ by adding a special $\Phi(x)_0$ feature which is always set to 1, and reduce the above equation to the dot product between the weight vector and the feature vector:

$$y = \mathbf{w} \cdot \Phi(x) \tag{3.18}$$

Note the similarity to Equation 3.3 for the linear classifier: in the case of binary classification we use the sign of the dot product to assign the object to the class; for linear regression the value of the dot product is the outcome we are predicting.

In order to learn a linear regression model we minimize the *sum squared error* over the training set of $M$ examples:

$$\text{cost}(\mathbf{w}) = \sum_{j=0}^{M} (\mathbf{w} \cdot \Phi(x^{(j)}) - y_{obs}^{(j)})^2 \tag{3.19}$$

There is a closed-form formula for choosing the best weights $\mathbf{w}$, given by:

$$\mathbf{w} = (X^T X)^{-1} X^T \mathbf{y} \tag{3.20}$$

where the matrix $X$ contains training example features, and $\mathbf{y}$ is the vector of outcomes.

### Logistic regression

In logistic regression we use the linear model to perform classification, i.e. assign probabilities to class labels. For binary classification we want to predict the probability of the instance being in the positive class given the instance: $p(y = true|x)$. But the predictions of a linear regression model are real numbers $y \in \mathbb{R}$, whereas probabilities range between 0 and 1: $p(y = true|x) \in [0, 1]$. To ensure that the response is in the valid range we can instead predict the logit function of the probability:

$$\ln\left(\frac{p(y = true|x)}{1 - p(y = true|x)}\right) = \mathbf{w} \cdot \Phi(x) \tag{3.21}$$

$$\frac{p(y = true|x)}{1 - p(y = true|x)} = e^{\mathbf{w} \cdot \Phi(x)} \tag{3.22}$$

Solving for $p(y = true|x)$ we obtain:

$$p(y = true|x) = \frac{e^{\mathbf{w} \cdot \Phi(x)}}{1 + e^{\mathbf{w} \cdot \Phi(x)}} \tag{3.23}$$

$$= \frac{\exp\left(\sum_{i=0}^{d} w_i \Phi(x)_i\right)}{1 + \exp\left(\sum_{i=0}^{d} w_i \Phi(x)_i\right)} \tag{3.24}$$

In order to learn a logistic regression model we use *conditional likelihood estimation*. We choose the weights which make the probability of the observed outcomes ($y$) be the highest, given the observations ($x$). For a training set with $N$ examples:

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmax}} \prod_{i=0}^{N} p_{\mathbf{w}}(y^{(i)}|x^{(i)}) \tag{3.25}$$

There is no close-form solution to this equation. It is a problem in convex optimization; several special purpose and generic solutions are available to train those models, e.g.

- L-BFGS (Limited-memory Broyden-Fletcher-Goldfarb-Shanno method)
- gradient ascent
- conjugate gradient
- iterative scaling algorithms

### Maximum Entropy Models

Logistic regression with more than two classes is referred to as *multinomial logistic regression*, and also known as Maximum Entropy (MaxEnt). The MaxEnt equation generalizes Equation 3.24 above:

$$p(y|x) = \frac{\exp\left(\sum_{i=0}^{d} w_i \Phi(x,y)_i\right)}{\sum_{y' \in Y} \exp\left(\sum_{i=0}^{d} w_i \Phi(x,y')_i\right)} \tag{3.26}$$

The denominator is the normalization factor usually called $Z$ used to make the score into a proper probability distribution:

$$p(y|x) = \frac{1}{Z} \exp \sum_{i=0}^{d} w_i \Phi(x,y)_i \tag{3.27}$$

**Indicator features** Note that in the above the feature function $\Phi$ is parameterized for the class label $y$. In MaxEnt modeling typically binary indicator features are used, which depend on the class label. Thus $\Phi(x,y)_i \in \{0,1\}$. For example, in the case of part of speech tagging, if the object $x$ is the word $w_0$ in the surrounding context, and

class label $y = \text{VBG}$, then an example feature might be:

$$\Phi(x, y)_1 = \begin{cases} 1 & \text{if suffix}_3(w_0) = \textit{ing} \ \wedge \ y = \text{VBG} \\ 0 & \text{otherwise} \end{cases}$$

The model weight for this feature will indicate how strong a predictor the suffix "ing" is for the label VBG.

**Maximum Entropy and Maximum Likelihood** The name Maximum Entropy comes from the fact that solving the optimization problem for finding the multinomial logistic regression model whose weights maximize the likelihood of the training data is equivalent to finding the probability distribution $p*$ with maximum entropy among the set of distributions $C$ which are consistent with the constraints imposed by the features and the training data:

$$p* = \underset{p \in C}{\text{argmax}}\, \text{H}(p) \tag{3.28}$$

where the entropy of the distribution of discrete random variable $X$ is given by:

$$H(X) = -\sum_{x} P(X = x) \log_2 P(X = x) \tag{3.29}$$

This duality was demonstrated by Berger et al. (1996). Maximizing the entropy subject to some constraints is motivated by the well-known Occam's razor principle: our model should be as simple as possible while still predicting the data; in this case "simple" is interpreted as maximally uniform, since the uniform distribution has the highest entropy. The constraints imposed on the probability model are encoded in the features: the expected value of each one of $I$ indicator features $f_i$ under a model $p$ should be equal to the expected value under the empirical distribution $\tilde{p}$ obtained from the training data:

$$\forall i \in I, \ \text{E}_p[f_i] = \text{E}_{\tilde{p}}[f_i] \tag{3.30}$$

where $f_i(x, y) = \Phi(x, y)_i$. The expected value under the empirical distribution is given by:

$$\mathrm{E}_{\tilde{p}}[f_i] = \sum_x \sum_y \tilde{p}(x, y) f_i(x, y) = \frac{1}{N} \sum_j^N f_i(x_j, y_j) \tag{3.31}$$

The expected value according to model $p$ is:

$$\mathrm{E}_p[f_i] = \sum_x \sum_y p(x, y) f_i(x, y) \tag{3.32}$$

However, this requires summing over all possible object - class label pairs, which is in general not possible. Therefore the following standard approximation is used (Rosenfeld, 1996):

$$\mathrm{E}_p[f_i] = \sum_x \sum_y \tilde{p}(x) p(y|x) f_i(x, y) = \frac{1}{N} \sum_j^N \sum_y p(y|x_j) f_i(x_j, y) \tag{3.33}$$

where $\tilde{p}(x)$ is the relative frequency of object $x$ in the training data; this has the advantage that $\tilde{p}(x)$ for unseen events is 0. The term $p(y|x)$ is calculated according to Equation 3.26.

**Regularization** Although the Maximum Entropy principle used in MaxEnt modeling ensures that the models are maximally uniform subject to the constraints, they can still overfit the training data, resulting in poor generalization to unseen instances. There is a technique called regularization which results in relaxing the requirement that the constraints be satisfied exactly and results in models with smaller weights which may perform better on new data. Instead of solving the optimization in Equation 3.25, repeated here in log-space form:

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{i=0}^{M} \log p_{\mathbf{w}}(y^{(i)}|x^{(i)}), \tag{3.34}$$

we solve instead the following modified problem:

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{i=0}^{M} \log p_{\mathbf{w}}(y^{(i)}|x^{(i)}) + \alpha R(\mathbf{w}) \tag{3.35}$$

where $R$ is the regularizer used to penalize large weights (Jurafsky and Martin, 2008). We can use a regularizer which assumes that weight values have a Gaussian distribution centered on 0 and with variance $\sigma^2$ (Chen and Rosenfeld, 1999). By multiplying each weight by a Gaussian prior we will maximize the following equation:

$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{i=0}^{M} \log p_{\mathbf{w}}(y^{(i)}|x^{(i)}) - \sum_{j=0}^{d} \frac{w_j^2}{2\sigma_j^2} \tag{3.36}$$

where $\sigma_j^2$ are the variances of the Gaussians of feature weights. This modification corresponds to using a maximum *a posteriori* rather than maximum likelihood model estimation. In practice it is common to constrain all the weights to have the same global variance, which gives a single tunable algorithm parameter, whose optimal value can be found on held-out data or by cross-validation.

The MaxEnt algorithm is one of the Machine Learning methods used for function labeling experiments for Spanish in Section 5.2. I also employ Maximum Entropy models for the **Morfette** morphological analysis system described in Sections 6.4 and 6.5.

### 3.2.4 Support Vector Machines

Support Vector Machine (SVM) is a machine learning algorithm which exploits two two key ideas: large-margin classification, and the "kernel trick".

#### Large margin

The idea of large margin classification, mentioned briefly in Section 3.2.1, is both intuitively appealing and theoretically motivated. Intuitively, it makes sense for the decision boundary to be as far away from the training instances as possible: this improves the chance that if the position of the data points is slightly perturbed, the decision boundary will still be correct. Results from Statistical Learning Theory confirm these intuitions: maintaining large margins leads to small generalization error (Vapnik, 1995).

Formally, the functional margin of an instance $(x, y)$ with respect to some hyperplane $(\mathbf{w}, b)$ is defined to be

$$\gamma = y(\mathbf{w} \cdot \Phi(x) + b) \tag{3.37}$$

Figure 3.3: Separating hyperplane and support vectors

Some data points will have the minimum functional margin: the functional margin of the whole data set with respect to the hyperplane is then twice that quantity.

However, the functional margin can be made larger just by rescaling the weights by some constant: $(\lambda\mathbf{w}, \lambda b)$ without changing the associated hyperplane. Hence we can fix the functional margin to be 1 and minimize the norm of the weight vector (which is equivalent to maximizing the geometric margin).

This results in the following quadratic programming optimization formulation of the SVM learner: For linearly separable training instances $((x_1, y_1), ..., (x_n, y_n))$ find the hyperplane $(\mathbf{w}, b)$ that solves the optimization problem:

$$\begin{aligned} &\text{minimize}_{\mathbf{w},b} \quad \frac{1}{2}||\mathbf{w}||^2 \\ &\text{subject to} \qquad y_i(w \cdot \Phi(x_i) + b) \geq 1 \quad \forall_{i \in 1..n} \end{aligned} \tag{3.38}$$

This hyperplane separates the examples with geometric margin $2/||\mathbf{w}||$

Since SVM finds a separating hyperplane with the largest margin to the nearest instance, this has the effect of the decision boundary being fully determined by a small subset of the training examples, namely the nearest ones on both sides. Those instances are the *support vectors* which SVM is named after.

Figure 3.3 shows the same data points as Figure 3.2. The solid line is the separating hyperplane with the maximum margin with respect to the training data; the points on the dotted line are support vectors.

**Soft margin** For datasets which are not linearly separable there will be no hyperplane satisfying the constraints. To deal with such cases the version of SVM with soft margin has been proposed. It works by relaxing the requirement that all data points lie outside the margin, and introduces a penalty term which measures how much this requirement is violated. For each offending instance there is a "slack variable" $\xi_i$ which measures how much it would have to be moved to make it obey the margin constraint. This leads

to the following modified formulation:

$$\begin{aligned} \text{minimize}_{\mathbf{w},b} \quad & \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \quad & y_i(w \cdot \Phi(x_i) + b) \geq 1 - \xi_i \\ & \forall_{i \in 1..n}\xi_i > 0 \end{aligned} \tag{3.39}$$

where

$$\xi_i = \max(0, 1 - y_i(\mathbf{w} \cdot \Phi(x_i) + b))$$

The hyper-parameter $C$ is the cost of margin constraint violation, used to trade off minimizing the norm of the weight vector versus classifying correctly as many examples as possible. As the value of $C$ tends towards infinity the soft-margin SVM approximates the hard-margin version.

### Kernel-induced feature spaces

As presented so far the SVM algorithm finds decision boundaries only for linearly separable data. This limitation can be removed by exploiting the "kernel trick". The kernel technique depends on the fact that for some linear classification algorithms, including SVM, there exist dual formulations, where the weight vector can be expressed as a linear combination of training examples, and the algorithm only involves computing dot products between the test instance and training instances.

**Dual form** The dual formulation of the optimal hyperplane for SVM is in terms of support vectors, where $SV$ is the set of indices of support vectors:

$$f(x, \alpha^*, b^*) = \text{sign}\left(\sum_{i \in SV} y_i \alpha_i^* (\Phi(x_i) \cdot \Phi(x)) + b^*\right) \tag{3.40}$$

The weights in this decision function are the Lagrange multipliers $\alpha^*$. Points which are not in the support vector set have no influence on the final decision. The dual

Figure 3.4: Two dimensional classification example, non-separable in two dimensions, becomes separable when mapped to 3 dimensions by $(x_1, x_2) \mapsto (x_1^2, 2x_1x_2, x_2^2)$

optimization problem has the following form:

$$\begin{aligned} &\text{minimize} \quad W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j (\Phi(x_i) \cdot \Phi(x_j)) \\ &\text{subject to} \quad \sum_{i=1}^{n} y_i \alpha_i = 0 \qquad\qquad \forall_{i \in 1..n} \alpha_i \geq 0 \end{aligned} \tag{3.41}$$

The Lagrangian weights together with the support vectors determine the separating hyperplane parameters $(\mathbf{w}, b)$ as follows:

$$\mathbf{w} = \sum_{i \in SV} \alpha_i y_i \Phi(x_i) \tag{3.42}$$

$$b = y_k - \mathbf{w} \cdot \Phi(x_k) \text{ for any } k \text{ such that } \alpha_k \neq 0 \tag{3.43}$$

**Kernel trick** Note that both the decision function used to predict classes in Equation 3.40 and the objective function that needs to be optimized during training in Equation 3.41 only use the data points inside dot products between instances. This allows to use the "kernel trick", which implicitly maps the data to a higher dimensional space without actually needing to compute the mapped vectors, by replacing each occurrence of a dot product with a kernel function (Aizerman et al., 1964). A kernel $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a function which computes the dot product of a pair of real-valued vectors in the *kernel-induced feature space*. For example the quadratic kernel

$$K(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}')^2 \tag{3.44}$$

corresponds to mapping the vectors to a quadratic-dimensional space with the function

$$\mathbf{x} \mapsto (x_1x_1, ..., x_1x_n, x_2x_1, ..., x_2x_n, ..., x_nx_1, ..., x_nx_n)$$

and then computing the dot product between them.

Data which is not separable in the original space usually becomes (more) separable in the higher-dimensional, kernel-induced space. The geometric intuition behind this idea is illustrated in Figure 3.4. In two dimensions the blank points are surrounded by the black points, and thus there is no straight line separating the two classes. We can, however, project the points into three dimensional space by using a map such as $(x_1, x_2) \mapsto (x_1^2, 2x_1x_2, x_2^2)$: the addition of a third dimension makes the two classes easily separable by a plane.

An alternative intuition is that using kernel spaces permits us to use complex attributes which model interactions between simple features of the original low-dimensional representation. For example the quadratic kernel presented above models all conjunctions of the original features, and it does so without incurring the cost of actually explicitly computing the high-dimensional representations.

The "kernel trick" also makes it possible to work in infinite-dimensional spaces, such as those induced by Gaussian kernels. A commonly used and empirically successful Gaussian kernel used with SVM is the Gaussian Radial Basis Function (RBF), defined as:

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma|\mathbf{x} - \mathbf{x}'|^2\right) \tag{3.45}$$

where

$$\gamma = \frac{1}{2\sigma^2}$$

and $\sigma^2$ is the variance of the Gaussian. The RBF $\gamma$ parameter can found using some model selection method such as cross-validation or held out data. As the optimal value of the soft-margin SVM cost parameter $C$ depends on the choice of kernel parameters, usually the model selection has to be done jointly for $C$ and $\gamma$ (or equivalent parameters for other kernels).

### Multiclass classification

SVM is essentially a binary classifier. The most common method to perform multiclass classification with SVM, and other binary max-margin algorithms, is to train multiple binary classifiers and combine their predictions to form the final prediction. This can

be done in two ways:

- One-vs-rest (also known as one-vs-all): train $|Y|$ binary classifiers and choose the class for which the margin is the largest.
- One-vs-one: train $|Y|(|Y|-1)/2$ pairwise binary classifiers, and choose the class selected by the majority of them.

An alternative method is to make the weight vector, or the feature function $\Phi$, depend on the output $y$, and learn a single classifier which will predict the class with the highest score:

$$y = \underset{y' \in Y}{\operatorname{argmax}} \mathbf{w} \cdot \Phi(x, y') + b \tag{3.46}$$

Such multiclass extensions to the SVM algorithm have been proposed among others by (Weston and Watkins, 1999; Crammer and Singer, 2001; Tsochantaridis et al., 2005).

The SVM algorithm is the best-performing Machine Learning method used for function labeling experiments for Spanish in Section 5.2. I also employed SVM in the function-labeling experiments described in Section 5.3, and in the context-sensitive lemmatization work reported in Section 6.3.

## 3.3 Sequence Labeling

Assigning sequences of labels to sequences of some objects is a very common task not only in NLP but also in fields such as bioinformatics. In NLP sequence labeling encompasses tasks such as POS tagging, chunking (shallow parsing) and named-entity recognition. In the general case, we want to learn a function $h : \Sigma^* \rightarrow \mathcal{L}^*$ to assign some sequence of labels from the label set $\mathcal{L}$ to the sequence of input elements from the set $\Sigma$:

The most frequent, and the more easily tractable case is where each element of the input sequence receives one label, i.e. the length of the input sequence is equal to the length of the output sequence:

$$h : \Sigma^n \rightarrow \mathcal{L}^n$$

This constraint holds for POS or morphological tagging; in cases where it does not naturally hold, such as chunking, we can decompose the task in such a way that it

is satisfied. A common way of doing it using an encoding such as the so-called IOB scheme, introduced by Ramshaw and Marcus (1995), where each element receives a label indicating whether it is the initial element of a chunk X (B-X), a non-initial element of a chunk X (I-X) or is outside of any chunk (O).

In the following discussion I will restrict my attention to the more constrained case of input and output being of equal lengths. For concreteness, I will use POS tagging as a running example, where the elements of $\Sigma$ are words and the elements of $\mathcal{L}$ are part-of-speech tags.

**Local classifier** Possibly the simplest approach to sequence labeling is to simply use a regular multiclass classifier such as those discussed in Section 3.2, and make a local decision for each word; predictions for previous words can be used in predicting the current word. This straightforward strategy can give surprisingly good results.

**MaxEnt with beam search** If the classifier can output not just a hard decision, but a score such as a probability for a given label, then we can combine the local predictions in a less greedy fashion. One option is to use a probabilistic classifier in combination with a beam search strategy to find a good global sequence of tags. This method is used by (Ratnaparkhi, 1996): he proposes to train a conditional Maximum Entropy model to predict tags for words $w_i$ given their context $c_i$: the context includes the focus word as well as a window of preceding and following words, and the tags of the preceding words. The probability of the sequence of tags $(t_1, ..., t_n)$ is decomposed as:

$$P(t_1, ..., t_n | c_1, ..., c_n) = \prod_{i=1}^{n} P(t_i | c_i) \tag{3.47}$$

For each word $w_i$ the beam search algorithm maintains the $N$ (= beam size) highest scoring tag sequences for words $(w_1, ..., w_{i-1})$ up to the previous word. Each of those label sequences is combined with the current word $w_i$ to create the context $c_i$, and the Maximum Entropy Model is used to obtain the $N$ probability distributions over tags for word $w_i$. Now we find $N$ most likely sequences of tags up to and including word $w_i$ by using Equation 3.47, i.e. by multiplying the probability of the sequence of tags up to $w_{i-1}$ with the probability for the tag $t_i$ given the context formed using that sequence,

and we proceed to word $w_{i+1}$ if $i \leq n$.

Maximum Entropy in conjunction with a specialized beam search algorithm is the sequence labeling method which I chose for the **Morfette** morphological analysis system described in Section 6.4.

A similar approach which uses the Viterbi algorithm to find the globally best sequence of tags instead of the beam search is discussed in the next section.

### 3.3.1 Maximum Entropy Markov Models

The Maximum Entropy Markov Models (MEMMs) approach to sequence labeling was introduced by McCallum et al. (2000). The method is a combination of the Hidden Markov Model (HMM) and the Maximum Entropy frameworks adapted to the sequence labeling setting. A HMM is a generative model, where the probability of a tag sequence given a sequence of words, $P(T|W)$ is modeled as $P(W|T)P(T)$ (via Bayes' theorem). In an $n$-order Markov model the probability of the current tag is conditioned on $n$ previous tags. Thus in a first-order HMM model in order to find the optimal sequence of tags, we solve the following argmax:

$$\begin{aligned}\hat{T} &= \operatorname*{argmax}_T P(T|W) \\ &= \operatorname*{argmax}_T P(W|T)P(T) \\ &= \operatorname*{argmax}_T \prod_i P(w_i|t_i) \prod_i P(t_i|t_{i-1})\end{aligned} \tag{3.48}$$

In a first-order MEMM the conditional probability of the tag sequence given the word sequence is decomposed directly into the product of direct conditional probabilities of tag given word and the previous tag:

$$\begin{aligned}\hat{T} &= \operatorname*{argmax}_T P(T|W) \\ &= \operatorname*{argmax}_T \prod_i P(t_i|w_i, t_{i-1})\end{aligned} \tag{3.49}$$

Those local conditional probabilities are given by a maximum entropy model trained on the true sequence of tags. During prediction, the previous tag is unknown, so the previously predicted tag is used, and the argmax is solved using the Viterbi algorithm.

In comparison to using beam search, the advantage of using dynamic programming is that this is guaranteed to find the globally best solution. A potential problem arises if we want to use second or higher-order Markov models (i.e. we want to condition the decision about the current tag on two or more previous tags). For each additional previous label that we want to consider we need to add another dimension to the dynamic programming matrix which may make the computational cost prohibitive. The beam search approach avoids this issue.

### 3.3.2 Conditional Random Fields and other structured prediction methods

The methods described above are usually highly successful in practical applications such as POS tagging. However they suffer from some theoretical shortcomings. The greedy use of local classifiers risks suffering from error propagation, where an erroneous prediction affects the subsequent predictions for the following words. Additionally, non-local constraints on possible sequences are not enforced.

There is also the limitation to sequence labeling; ideally one would want to have more general methods able to learn more types of structured outputs.

Conditional Random Fields (CRF) introduced by Lafferty et al. (2001) were one of the first approaches of type (1); this algorithm is a generalization of the Maximum Entropy framework to structured outputs. The equation determining the probability of a structured output $y$ given the input $x$ is identical to the one used for multiclass classification in Equation 3.26:

$$p(y|x) = \frac{\exp\left(\sum_{i=0}^{d} w_i \Phi(x,y)_i\right)}{\sum_{y' \in Y} \exp\left(\sum_{i=0}^{d} w_i \Phi(x,y')_i\right)} \tag{3.50}$$

Thus, unlike for MEMMs, here MaxEnt is used to learn a single exponential model for the probability distribution over the set of possible label sequences. Since the set $Y$ contains structures such as sequences, the challenge here is to compute the sum in the denominator. Lafferty et al. (2001) and Sha and Pereira (2003) show that given certain constraints on $Y$ and on $\Phi$ (specifically $\Phi$ should obey the Markov property, i.e. no feature should depend on elements of $y$ that are more than the Markov length $l$ apart)

dynamic programming techniques can be used to compute it efficiently.

The alternative to learning global probabilistic models such as CRFs is the paradigm known as Learning and Inference, where predictions from local classifiers are taken as input to the inference procedure which finds the globally best prediction. Work within this framework includes (Roth, 2001; Carreras et al., 2002; Roth and Yih, 2004; Canisius et al., 2006).

Recently several novel approaches to structured prediction have been proposed. It is a fast-developing field of research in machine learning and a thorough discussion of those developments is outside the scope of this chapter. Some of the more important contributions include: structured and incremental Perceptron (Collins, 2002; Collins and Roark, 2004), Maximum Margin Markov Networks (Taskar et al., 2004), SVM$^{\text{struct}}$ (Tsochantaridis et al., 2005), Searn (Daumé III, 2006), as well as research within the discriminative reranking paradigm (Johnson et al., 1999; Collins, 2000; Shen et al., 2003; Collins and Koo, 2005; Charniak and Johnson, 2005).

## 3.4 Summary

In this chapter I presented an overview of well-understood machine-learning algorithms, used to solve binary and multiclass classification and sequence labeling tasks. I discussed several classification algorithms frequently used in Natural Language Processing. The Perceptron, Logistic Regression (MaxEnt) and Support Vector Machines are essentially linear classifiers which find a hyperplane separating the data in the feature space; the Perceptron and the SVM can exploit the kernel trick to deal with non-linear classification. The $k$-nearest-neighbors algorithm is non-linear, and is based on the idea of assigning points the same labels as the majority of their neighbors in the feature space.

Empirically the SVM algorithm tends to outperform the other methods for many practical tasks. This high performance comes at a price: while the other algorithms scale linearly in the number of training examples, the SVM can be $\mathcal{O}(N^2)$ or even $\mathcal{O}(N^3)$, depending on the optimization algorithm used, which can make it impractical for problems with large amounts of training data. For multiclass classification, another consideration is the number of class labels: the performance of the $k$-NN algorithm is constant with size of the class-set, while the other algorithms scale at best linearly.

I have also briefly reviewed the techniques used to perform sequence labeling. The simplest but often effective approach is to use a classifier to predict a label at each position in the sequence, possibly using previous predictions as input for decisions further on in the sequence. A better-motivated alternative is to combine scores of the local predictions to find a globally good labeling by using beam search or dynamic programming techniques. Finally, more sophisticated structured prediction approaches such as CRF models are able to rank global labelings directly.

Since in sequence labeling the structures to learn are relatively simple often good results can be obtained by using simple models, which also have the benefit of being quite efficient to train and use.

# Chapter 4

# Treebank-Based LFG Parsing Resources for Spanish

## 4.1 Introduction

In this chapter I report on the work carried out on expanding the coverage of the Spanish annotation algorithm in order to give an indication of the kind of issues involved in porting this module to a new language and treebank, and provide background for Section 5.2 in the next chapter.

One of the aims of my research is to test to what degree it is possible to achieve high portability and language independence by learning as much information as possible from annotated data. Therefore it was useful to focus on improving the LFG induction methods for a language other than English and for which little previous work had been carried out. I expanded the Spanish treebank-based grammar coverage and linguistic analyses of O'Donovan et al. (2005). The revision and reworking of the Spanish LFG induction system made it possible to experiment with learning the Cast3LB function tags from the Spanish treebank.

### 4.1.1 The Cast3LB Spanish treebank

As input to the LFG annotation algorithm I use the output of Bikel's parser (Bikel, 2002) trained on the Cast3LB treebank (Civit and Martí, 2004) (compare Section 5.2.3). Cast3LB contains around 3,500 constituency trees (100,000 words) taken from different

genres of European and Latin American Spanish. The POS tags used in Cast3LB encode morphological information in addition to Part-of-Speech information.

Due to the relatively flexible order of main sentence constituents in Spanish, Cast3LB uses a flat structure for the S node. There is no VP node, but rather all complements and adjuncts depending on a verb are sisters to the *gv* (Verb Group) node containing this verb. An example sentence (with the corresponding f-structure) is shown in Figure 4.1.

Tree nodes are additionally labelled with grammatical function labels. Civit (2004) provides Cast3LB function label guidelines. Functional labels carry some of the information that would be encoded in terms of tree configurations in languages with stricter constituent order constraints than Spanish.

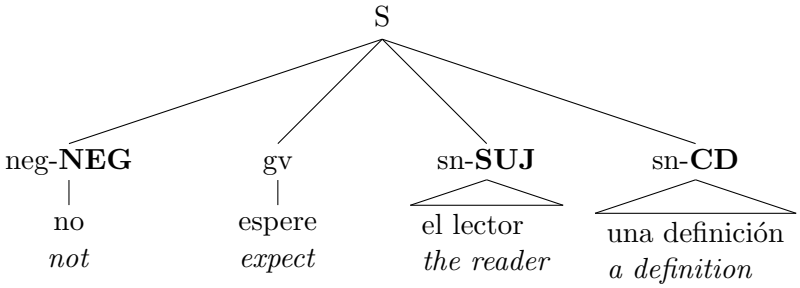
## 4.2 Comparison to Previous Work

Preliminary proof-of-concept research on Spanish LFG induction was carried out by O'Donovan et al. (2005). This work created the following resources:

- Head rules for the syntactic analyses in the Cast3LB treebank, used with the Bikel parser for c-structure parsing.
- Annotation algorithm for the Cast3LB treebank, relying heavily on function labels from the treebank.
- Hand-corrected f-structures for 100 sentences to be used as a gold standard.

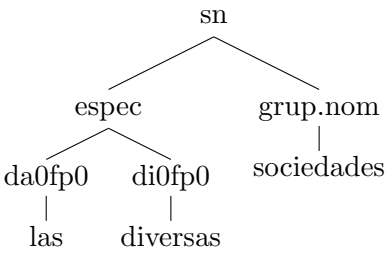
The gold standard was created by running the annotation algorithm on the treebank trees and inspecting and correcting the output. The annotation algorithm was quite basic, for most annotations relying on the function labels provided in treebank trees. Otherwise it generated f-structures which closely mimic the constituent structure encoded in the trees.

For my experiments on Spanish LFG parsing I decided to adopt f-structures that are simpler, mimic c-structures less directly, and are more abstract and thus less language and treebank specific. For example in my f-structures, both prenominal and postnominal adjectives are put in the adjunct set, whereas in O'Donovan et al. (2005) prenominal




$$\begin{bmatrix}
\text{PRED} & \text{`esperar}\langle\text{SUBJ,OBJ}\rangle\text{'} \\
\text{NEG} & + \\
\text{TENSE} & \text{PRES} \\
\text{MOOD} & \text{SUBJUNCTIVE} \\
\text{SUBJ} & \begin{bmatrix}
\text{SPEC} & \begin{bmatrix} \text{SPEC-FORM} & \text{EL} \\ \text{SPEC-TYPE} & \text{DEF} \end{bmatrix} \\
\text{PRED} & \text{`lector'} \\
\text{GEND} & \text{MASC} \\
\text{NUM} & \text{SG} \\
\text{PERS} & 3
\end{bmatrix} \\
\text{OBJ} & \begin{bmatrix}
\text{SPEC} & \begin{bmatrix} \text{SPEC-FORM} & \text{UNO} \\ \text{SPEC-TYPE} & \text{INDEF} \end{bmatrix} \\
\text{PRED} & \text{`definición'} \\
\text{GEND} & \text{FEM} \\
\text{NUM} & \text{SG}
\end{bmatrix}
\end{bmatrix}$$

Figure 4.1: On top flat structure of S. Cast3LB function labels are shown in bold. Below the corresponding (simplified) LFG f-structure. Translation: *Let the reader not expect a definition.*




Cast3LB tree

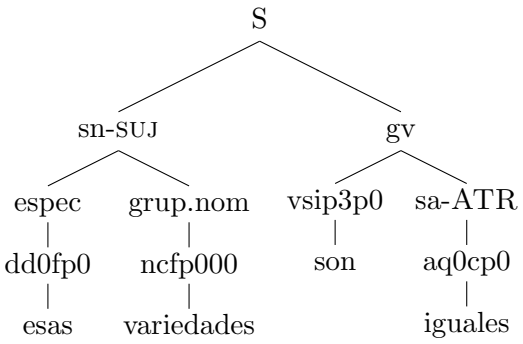
$$\begin{bmatrix} \text{SPEC} & \begin{bmatrix} \text{DET} & \begin{bmatrix} \text{SPEC} & \begin{bmatrix} \text{PRED} & \text{'el'} \\ \text{NUM} & \text{PLURAL} \\ \text{GEN} & \text{FEM} \end{bmatrix} \\ \text{PRED} & \text{'diverso'} \\ \text{NUM} & \text{PLURAL} \\ \text{GEN} & \text{FEM} \end{bmatrix} \end{bmatrix} \\ \text{PRED} & \text{'sociedad'} \\ \text{NUM} & \text{PLURAL} \\ \text{GEN} & \text{FEM} \end{bmatrix}$$

Gold f-structure in O'Donovan et al. (2005)

$$\begin{bmatrix} \text{SPEC} & \begin{bmatrix} \text{SPEC-FORM} & \text{EL} \\ \text{SPEC-TYPE} & \text{DEF} \end{bmatrix} \\ \text{PRED} & \text{'sociedad'} \\ \text{GEND} & \text{FEM} \\ \text{NUM} & \text{PL} \\ \text{ADJUNCT} & \left\{ \begin{bmatrix} \text{PRED} & \text{'diverso'} \\ \text{ATYPE} & \text{INDEF} \end{bmatrix} \right\} \end{bmatrix}$$

Gold f-structure in current thesis

Figure 4.2: Comparison of f-structure representations for NPs




Cast3LB tree

$$\begin{bmatrix} \text{SUBJ} & \begin{bmatrix} \text{SPEC} & \begin{bmatrix} \text{DET} & \begin{bmatrix} \text{PRED} & \text{`ese'} \\ \text{NUM} & \text{PLURAL} \\ \text{GEN} & \text{FEM} \end{bmatrix} \end{bmatrix} \\ \text{PRED} & \text{`variedad'} \\ \text{NUM} & \text{PLURAL} \\ \text{GEN} & \text{FEM} \\ \text{PERS} & 3 \end{bmatrix} \\ \text{PRED} & \text{`ser'} \\ \text{TENSE} & \text{PRES} \\ \text{XCOMP} & \begin{bmatrix} \text{SUBJ} & [\ \ ] \\ \text{PRED} & \text{`igual'} \\ \text{NUM} & \text{PLURAL} \end{bmatrix} \end{bmatrix}$$

Gold f-structure in O'Donovan et al. (2005)

$$\begin{bmatrix} \text{SUBJ} & \begin{bmatrix} \text{SPEC} & \begin{bmatrix} \text{SPEC-TYPE} & \text{DEF} \\ \text{SPEC-FORM} & \text{ESE} \end{bmatrix} \\ \text{PRED} & \text{`variedad'} \\ \text{NUM} & \text{PL} \\ \text{GEND} & \text{FEM} \\ \text{PERS} & 3 \end{bmatrix} \\ \text{PRED} & \text{`ser'} \\ \text{MOOD} & \text{INDICATIVE} \\ \text{TENSE} & \text{PRES} \\ \text{PREDLINK} & \begin{bmatrix} \text{PRED} & \text{`igual'} \\ \text{NUM} & \text{PL} \end{bmatrix} \end{bmatrix}$$

Gold f-structure in current thesis

Figure 4.3: Comparison of f-structure representations for copular verbs

modifiers are nested in the SPEC attribute. I have also simplified the structure of the SPEC, getting rid of redundant gender and number specifications, and extra levels of nesting due to the DET attribute. For illustration, Figure 4.2 shows the Cast3LB analysis for the NP *las diversas sociedades* (*the various societies*), and the two corresponding f-structures.

I also adopted the PREDLINK analysis for copular constructions, as opposed to the XCOMP analysis used by O'Donovan et al. (2005). Figure 4.2 compares the two analyses for the sentence *Esas variedades son iguales* (*These varieties are the same*). Additionally I implemented a variety of other more minor improvements and modifications.

In addition to the changes detailed above I also tackled three more crucial shortcomings in the O'Donovan et al. (2005)'s annotation algortihm: namely the treatment of null subjects, clitic doubling, and periphrastic constructions. These enhancements are detailed in Sections 4.3.1 and 4.3.2.

As another contribution towards LFG parsing resources for Spanish I enlarged the set of gold-standard f-structures to all the 336 sentences in the test set (i.e. 10% of the full treebank). This set of f-structures was built in the same manner as in O'Donovan et al. (2005): i.e. I first extended and enhanced the annotation algorithm, which was then run on the treebank trees. The resulting f-structures were then inspected and hand-corrected in the cases when the algorithm did not produce the desired f-structure representation.

## 4.3 Improving Spanish LFG Resources

In this section I discuss several problems which became obvious while trying to expand the coverage of Spanish grammatical constructions and phenomena and while dealing with the peculiarities of the Spanish treebank. The problems arising from adapting a grammar acquisition methodology developed for one language/treebank to another language/treebank combination fall into three broad categories:

- new phenomena and constructions, successfully treated within standard LFG: clitic doubling, null subjects;
- new phenomena and constructions, problematic within standard LFG: clitic climbing (i.e. complex predicates);
- limitations of the previous approach due to language/treebank specific assumptions which no longer hold: flexible constituent order and less configurational c-structures.

### 4.3.1 Clitic doubling and null subjects

In Spanish pronominal clitics for Direct and Indirect Object can co-occur with non-clitic (full NP) objects.Example (4.1) shows clitic doubling with Indirect Object, Example

(4.2) with Direct Object. The non-clitic Objects are in italics; the co-occurring clitics are in bold. The clitics agree with the non-clitic arguments in person, number, gender and case.

(4.1) Algo parecido **les** sucede *a los hombres.*
something similar them occurs to DEF men
Something similar happens to men.

(4.2) Cada cual **lo** comprende *eso* a su manera.
every which it understands this to POSS manner
Everyone understands this in their own way.

Clitic doubling is quite common with Indirect Objects: in our treebank data in 23% of the cases where there is a full (non-clitic) Indirect Object it co-occurs with a pronominal clitic. Clitic doubling for Direct Objects is more constrained, but still relatively common at 1% of corpus occurrences of non-pronominal Direct Objects.

In clitic doubling constructions, pronominal clitics should not introduce a PRED value, as that would clash with the one introduced by the non-clitic Object. However when clitics are not accompanied by non-clitic Objects, they should introduce PRED = 'pro', in order to satisfy the verb's subcategorization requirements.

I achieve this effect by means of optional equations (cf. (Andrews, 1990)). Example 4.3 below illustrates the equations associated with the dative *le* (Indirect Object).

(4.3) *le* **pp3csd00**

$((\uparrow \text{PRED}) = \text{'pro'})$
$((\uparrow \text{PRON-TYPE}) = \text{PERS})$
$((\uparrow \text{PRON-FORM}) = \text{el})$
$(\uparrow \text{CASE}) = \text{DAT}$
$(\uparrow \text{NUM}) = \text{SG}$
$(\uparrow \text{PERS}) = 3$

An optional equation $(e)$ is a disjunction of $e$ and *true*. In standard LFG the correct disjunct is chosen as follows: in a clitic-doubling context, the first disjunct is excluded because the PRED value it introduces clashes with the one introduced by the non-clitic

Object, and thus the *true* disjunct applies. In non-doubling contexts, the first disjunct applies successfully, while if the second one applies, the resulting f-structure does not satisfy completeness because of the missing PRED value.

In my implementation I do not check for completeness because the PRED values lack subcategorization frames,[1] so I use a slightly different definition of optionality. An optional equation works more like a default equation: the optional equation $((f\ a) = v)$ holding of f-structure $f$ is interpreted as a disjunction of the existential constraint $(f\ a)$ and the equation $(f\ a) = v$. In the clitic-doubling case the second disjunct (which introduces the PRED value) only applies if the PRED value has not been contributed by some other equation.

Another area where I use optional equations is in the treatment of null subjects (pro-drop). In Spanish explicit subjects are often absent. Subject features such as person and number are encoded in agreement morphology on the verb instead. When there is no overt subject, the PRED value that is needed to satisfy the verb's subcategorization is introduced by the inflected verb-form.

All finite verb preterminals optionally introduce a 'pro' subject. Example 4.4 below illustrates the annotation associated with the inflected verb form *vió* (see-3SG).

(4.4) *vió* **vmis3s0**

$(\uparrow \text{PRED}) = \text{'ver'}$

$((\uparrow \text{PRED SUBJ}) = \text{'pro'})$

$(\uparrow \text{SUBJ NUM}) = \text{SG}$

$(\uparrow \text{SUBJ PERS}) = 3$

$(\uparrow \text{SUBJ TENSE}) = \text{PAST}$

$(\uparrow \text{SUBJ MOOD}) = \text{INDICATIVE}$

$(\uparrow \text{LIGHT}) = -$

Currently all finite verb forms receive an optional PRED equation. This is not entirely adequate as at least one Spanish verb *haber* (existential be) can never co-occur with an overt subject, so ideally it should receive an obligatory PRED equation. Similarly, weather verbs are normally ungrammatical with explicit subjects (Example

[1] The subcat frames are acquired separately in the DCU architecture. See (O'Donovan et al., 2004).

4.5 a and b). Exceptionally they can take modified cognate subjects (Example 4.5 c).

(4.5) (a) * Llovió lluvia.
rained rain

(b) * La lluvia llovió.
the rain rained

(c) Llovió una lluvia fina pero persistente.
rained a rain light but persistent
"A light but persistent rain rained down."

Whether it is possible to learn from treebank data which verbs do not allow overt subjects and under what conditions remains an open question for future investigation.

The use of optionality in the treatment of Spanish clitic doubling and null subjects illustrates language-specific problems that arise for LFG induction, but for which there are standard solutions in the LFG framework. Those solutions can be adopted and adapted for our data-driven approach to grammar acquisition. They may require additional implementation effort (in this case adding appropriate optionality support to the constraint solver), but otherwise they can be easily accommodated within the existing methodology.

In the following section I discuss a phenomenon which is more problematic as it does not have a widely agreed-upon solution in standard LFG and thus is an issue in any computational implementation including the current one.

### 4.3.2 Periphrastic constructions

In Spanish periphrastic constructions, such as in Example (4.6 a), verbal pronominal clitics which are understood as arguments of the "lower" verb can attach to the "higher" verb. This phenomenon, called clitic climbing, is only grammatical with certain verbs. Others do not admit it, as illustrated in Example (4.6 b). The verbs that do admit clitic climbing are sometimes called *light* verbs.

(4.6) (a) **La** puedo *ver*. Puedo *ver***la**.
her can-1SG see can-1SG see-her
I can see her.

(b) * **La** insistí en *ver*. Insistí en *ver***la**.
her insisted-1SG in see insisted-1SG in see-her
I insisted on seeing her.

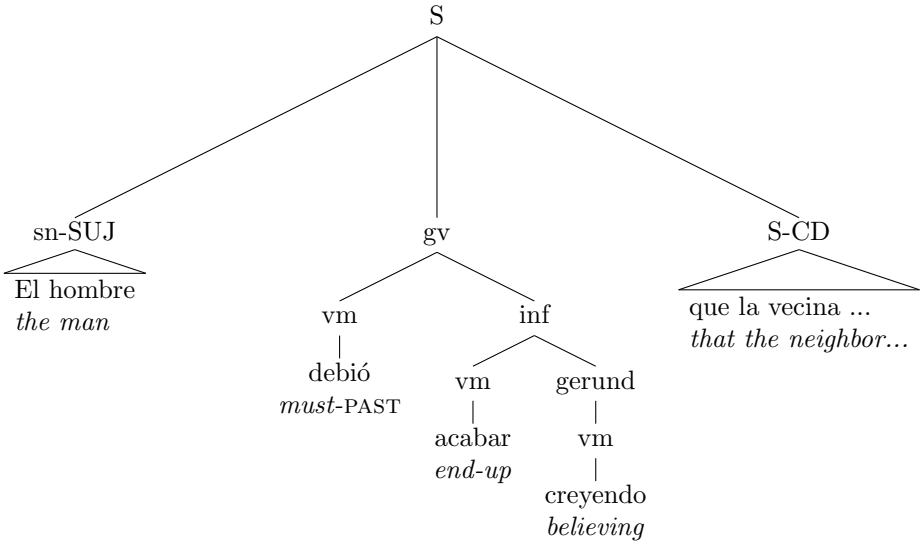
Normally only the clitic climbing versions of periphrastic constructions present difficulties for an LFG account due to the mismatch of the position of arguments in the tree and where they should end up in the f-structure. However, the c-structure configuration adopted for periphrastic constructions in Cast3LB generalizes this problematic mismatch to all contexts.

As illustrated in Figure 4.4, all verbs participating in the periphrastic construction are under the *gv* (Verb Group) node, with the argument of the lowest verb being attached as sister to the *gv*, rather than the rightmost *vm* under *gv*. This example also illustrates that periphrastic constructions can be combined with each other, so in principle the lowest non-light verb could be nested a number of levels deep.

There are several proposals of how to deal with periphrastic constructions with clitic climbing within LFG. Both Alsina (1997) and Butt (1997) propose a predicate composition analysis. As in standard LFG PRED values can never unify, this approach requires modifications to the unification operation. In (Andrews and Manning, 1999) the authors propose an even more radical departure from standard LFG and replace the projection architecture with *differential information spreading* within the f-structure.

As there seems to be no consensus as to the best treatment of Romance constructions involving light verbs, I decided in favor of a conservative approach which avoids non-standard extensions to the LFG formalism. I use functional uncertainty and a nested XCOMP configuration in the treatment of periphrastic constructions. The mechanism is illustrated in Figure 4.5. The *inf(initive)* and *gerund* daughters of the *gv* node constrain the f-structure corresponding to their mother nodes to be LIGHT +, and introduce their own f-structure as the value of XCOMP attribute.

Non-subject sisters of the *gv* are annotated with functional uncertainty equations which specify that their f-structure is the value of the GF attribute arbitrarily embedded in a series of XCOMPs. There is an off-path constraint that specifies that the f-structure containing each of the XCOMPs in the path has to be LIGHT +. Another off-path constraint on the f-structure containing the final GF restricts it to be LIGHT −. Together




$$\begin{bmatrix}
\text{SUBJ} & \left[\text{“el hombre”}\right] & & \\
\text{PRED} & \text{‘DEBER’} & & \\
\text{TENSE} & \text{PAST} & & \\
\text{LIGHT} & + & & \\
 & \begin{bmatrix}
\text{SUBJ} & [\quad] & & \\
\text{PRED} & \text{‘ACABAR’} & & \\
\text{LIGHT} & + & & \\
\text{XCOMP} & \begin{bmatrix}
\text{SUBJ} & [\quad] & \\
\text{PRED} & \text{‘CREER’} & \\
\text{LIGHT} & - & \\
\text{COMP} & \left[\text{“que la vecina ...”}\right] &
\end{bmatrix} &
\end{bmatrix} &
\end{bmatrix}$$

Figure 4.4: Periphrastic construction with two light verbs: The treebank tree, and the f-structure produced

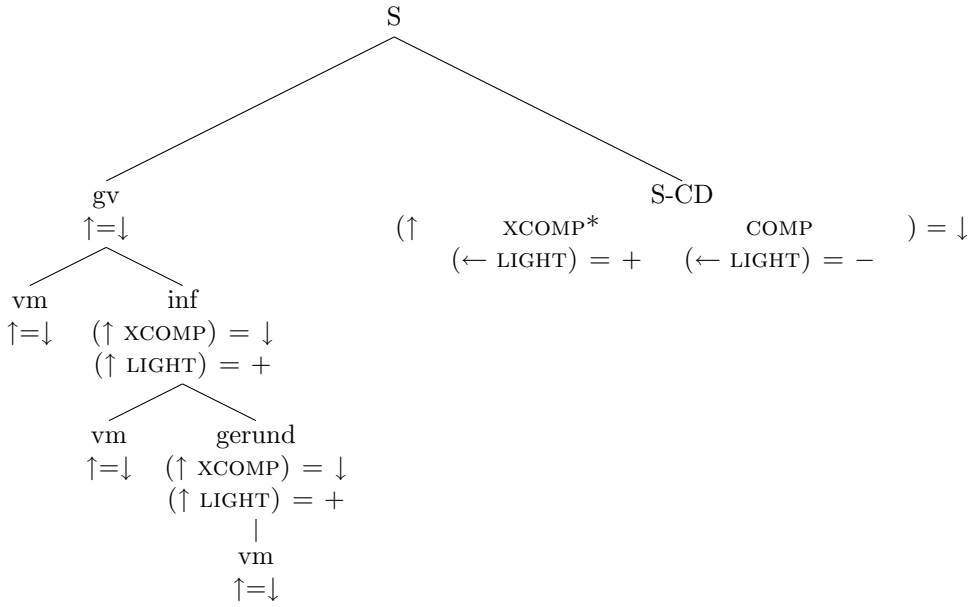



Figure 4.5: Treatment of periphrastic constructions by means of functional uncertainty equations with off-path constraints

those annotations ensure that arguments are always attached to the lowest (non-light) verb. This is the correct analysis for the majority of periphrastic constructions.[2]

This treatment of periphrastic constructions is a compromise solution: from a descriptive perspective it does not perfectly model the linguistic phenomena in question. On the other hand, it allows us to avoid implementing a solution which departs too far from the standard LFG formalism and for which there is no consensus among theoretical linguists.

For our present purposes, the XCOMP-based treatment is adequate, and has the advantage that the resulting f-structure parallels the analysis that would be used in languages with no clitic climbing (such as English) for similar sentences. This could potentially be useful if our LFG resources are to be used in multilingual applications.

[2]One exception are causative constructions, where, if one insists on an XCOMP-type treatment, the causee should be the argument of the causative verb, whereas the other arguments should depend on the verb expressing the event caused (Alsina, 1997).

## 4.4 Summary

In this chapter I described the work carried out to expand the coverage and linguistic adequacy of the Spanish annotation algorithm. This effort enabled experiments on Spanish LFG parsing described in the next chapter. I have also used the enhanced annotation algorithm to produce a set of 336 hand-corrected f-structures which were subsequently used for evaluation of the LFG parsing system for Spanish.

# Chapter 5

# Learning Function Labels

## 5.1 Introduction

As described in Section 2.2.1 the only module in the LFG parsing architecture which is not built by learning a model from labeled data is the annotation algorithm (compare Figure 2.2). It has to be developed manually, separately for each treebank and language and requires expertise in the language in question, LFG, as well as programming skills. Thus it is likely to be the main bottleneck slowing down development of multilingual data-driven LFG parsing resources using the DCU approach.

Because of this, it is desirable to minimize and simplify this component of the overall system as much as possible. In this chapter I investigate using machine-learning approaches to accurately learning syntactic and semantic function labels from treebanks. The labels often carry the same information as LFG annotations and in many cases there is a simple mapping between the two, and in many other cases access to accurately recovered function labels simplifies and eases the development of the annotation algorithms.

Section 5.2 investigates three machine learning methods to acquire function labels from the Spanish Cast3LB treebank. In Section 5.3 I describe how function labeling performance can be further improved by reducing the mismatch between training and testing instances, and I apply the method to English and Chinese data, highlighting the ease of porting of machine learning methods across languages.

I show that function labels can be reliably learned using machine-learning ap-

proaches and that all three algorithms tested substantially outperform the baseline, while the Support Vector Machine performs best for the Spanish treebank data. Furthermore, when the approach is further improved and applied to the English Penn-treebank data, it achieves the highest score on the function labeling task reported in literature to date.

Part of the research presented in this chapter has been previously published in (Chrupała and van Genabith, 2006a,b) and (Chrupała et al., 2007).

## 5.2 Learning Cast3LB Function Labels

In this section I discuss the particular features of Spanish and the Cast3LB treebank which challenge some of the assumptions made in the design of the LFG acquisition architecture initially developed using the English Penn Treebank data.

### 5.2.1 Annotation algorithm

The f-structure annotation algorithm for Spanish is implemented in a similar fashion to the original one for English, described in detail in (Burke, 2006) and briefly sketched here in Section 2.2.1. The algorithm visits every node in the c-structure tree and annotates it with LFG functional equations. The following information is used to determine the annotation:

**Head table.** This table specifies, for each local subtree of depth one, which constituent is the head daughter.

**Category labels.** Category labels of the current node and the mother node, together with position relative to head, can in many cases be used to determine the annotation.

**POS tags.** Cast3LB POS tags encode morphological features which the annotation algorithm translates into f-structure morphological attributes.

**Function labels.** Function labels in the Spanish Cast3LB treebank annotate nodes with their grammatical function. Most non-local dependencies (NLDs) are also

encoded via function labels[1]. Grammatical function labels can be mapped straightforwardly to LFG functional equations.

The Spanish annotation algorithm depends crucially on the presence of function labels in c-structure trees. Those function are a used in order to assign annotations to main sentence constituents, i.e. sisters of the *gv* (verb group) constituent. In order to determine annotations for the internal structure of constituents such as noun phrases or prepositional phrases, configurational information is exploited.

Function labels are present in treebank trees, but typically absent from parser output: thus we will need to develop a method to learn a model able to add them to parsed trees. The reasons behind the much greater weight given to functional labels in the Spanish are due to the relatively flexible constituent order and less configurational nature of this language as compared to English.

**Constituent order and configurationality in Spanish**

The method of automatic LFG induction was initially developed using the English Penn-II Treebank data. The idea behind the annotation rules is that limited configurational and categorial information should in most cases be sufficient to determine a constituent's grammatical function in the sentence: as evidenced by the good results of this approach for English, this assumption is borne out for this language. It turns out that the approach is more problematic for the Spanish Cast3LB data. Spanish allows much more variation and flexibility in major sentence constituent order than English. Partly as a consequence of this flexibility, the treebank encoding of syntactic structure also has to be different than that of the Penn Treebank.

Although the canonical word order for Spanish is SVO, in Cast3LB there are about 20% post-verbal subjects, and about 11% preverbal non-clitic direct objects. Thus the information on position relative to the verb is not a reliable predictor of grammatical function in Spanish.

Accordingly, the Spanish treebank makes extensive use of function labels to make the grammatical function of constituents more explicit. Although there are also functional labels in the Penn Treebank, their use is less necessary. In the Penn Treebank,

[1] In a small number of cases coindexations are used

configuration information alone is often sufficient to determine grammatical function: e.g.: left *NP* sister to *VP* is typically a Subject while right *NP* daughter to *V* is an Object.

While in English tree configuration alone will in most cases constrain the possible grammatical functions of constituents, this is in general not the case in Spanish. In a language like Spanish configuration is only one of several sources of interacting "soft" constraints on grammatical function. This means that an annotation algorithm using solely categorial and configuration information and a set of categorical rules cannot reliably generate LFG functional annotations for the Spanish Cast3LB treebank.

Instead we would like to extract features of the constituent trees which are plausible (but "soft") predictors of grammatical function and learn from the treebank how to use them to assign grammatical functions to tree nodes. More concretely, we will learn from training data how to assign the function labels described above to constituency trees. Such trees, with nodes tagged with function labels predicted by our model, will then be used by the annotation algorithm and mapped to LFG functional equations. Thus the Spanish annotation algorithm will rely on function labels much more heavily than is the case for English. It is therefore important to be able to enrich parser-output trees with those labels as reliably as possible.

The initial implementation described in (O'Donovan et al., 2005) relied on the parser itself to obtain function-tagged parse trees. Bikel's parser (Bikel, 2002) was trained on trees where function labels were simply part of the category label, so instead of having one non-terminal category *sn* (Noun Phrase) there are several different NP categories e.g. *sn*-SUJ, *sn*-CD, *sn*-CI, etc. I treat this simple method as a baseline in order to determine how much it could be improved on by the following alternative method: I use the parser to learn and output plain constituency trees and then a separate module learns how to add Cast3LB function labels in a postprocessing step. The intuition behind adopting this approach is that we avoid the multiplication of categories (which could potentially lead to a sparse-data-related decline in performance), and also achieve better control over the learning method and the feature set used than if we just rely on the parser.

### 5.2.2 Previous work on learning function labels

Blaheta and Charniak (2000) use a generative probabilistic model with feature dependencies encoded by means of feature trees in which nodes are features which are assumed to depend only on their ancestors. They experiment with a number of feature trees and learn models to assign Penn-II Treebank function labels to Charniak's parser output. Using the best performing model they report an f-score 88.472% on original treebank trees and 87.277% on the correctly parsed subset of tree nodes.

Jijkoun and de Rijke (2004) describe a method of enriching output of a parser with information that is included in the original Penn-II trees, such as function labels, empty nodes and coindexations. They first transform Penn trees to a dependency format and then use memory-based learning to perform various graph transformations. One of the transformations is node relabeling, which adds function labels to parser output. They report an f-score of 88.5% for the task of function labeling on correctly parsed constituents.

Musillo and Merlo (2005) and Merlo and Musillo (2005) extend the Henderson parser (Henderson, 2003) and model function labels as both expressions of the lexical semantics properties of a constituent and as syntactic elements whose distribution is subject to structural locality constraints. This improves their both parsing score and function labeling score. They do not report scores on the full set of Penn-II function labels since they only try to recover the so-called syntactic and semantic label types (see Table 5.15 in Section 5.3.1).

Gabbard et al. (2006) describe a two stage parser which builds Penn Treebank analyses including both function labels and empty categories and co-indexations. Function labeling is performed during the first stage: they modify Bikel's implementation of Collins' parsing model to enable it to output function labels. They report 88.96% f-score on correctly parsed constituents on WSJ section 23. This approach is equivalent to the one used by O'Donovan et al. (2005), and treated here as a baseline.

### 5.2.3 Assigning Cast3LB function labels to parsed Spanish text

I divided the Spanish treebank into a training set of 80%, a development set of 10%, and a test set of 10% of all trees. I randomly assigned treebank files to these sets to

| Part of Speech | Features included |
|---|---|
| Determiner | type, number |
| Noun | type, number |
| Adjective | type, number |
| Pronoun | type, number, person |
| Verb | type, number, mood |
| Adverb | type |
| Conjunction | type |

Table 5.1: Features included in POS tags. Type refers to subcategories of parts of speech such as e.g. *common* and *proper* for nouns, or *main*, *auxiliary* and *semiauxiliary* for verbs. For details see (Civit, 2000).

ensure that different textual genres are about equally represented among the training, development and test trees.

**Constituency parsing** For constituency parsing I use Bikel's (Bikel, 2002) parser for which I developed a Spanish language package adapted to the Cast3LB data. Prior to parsing, I perform one of the tree transformations described by Cowan and Collins (2005), i.e. I add CP and SBAR nodes to subordinate and relative clauses. This is undone in parser output.

The category labels in the Spanish treebank are rather fine grained and often contain redundant information. For example there are several labels for Nominal Group, such as *grup.nom.ms* (masculine singular), *grup.nom.fs* (feminine singular), *grup.nom.mp* (masculine plural) etc. This number and gender information is already encoded in the POS tags of nouns heading these constituents. I preprocess the treebank and reduce the number of category labels, only retaining distinctions that are useful for LFG parsing. The labels we retained are the following: *INC, S, S.NF, S.NF.R, S.NF, S.R, conj.subord, coord, data, espec, gerundi, grup.nom, gv, infinitiu, interjeccio, morf, neg, numero, prep, relatiu, s.a, sa, sadv, sn, sp*, and versions of those suffixed with *.co* to indicate coordination). For constituency parsing I also reduce the number of POS tags by including only selected morphological features. Table 5.1 provides the list of morphological features included for the different parts of speech. In the experiments I use gold standard POS tagged development and test-set sentences as input rather than tagging text automatically.

| | LB Precision | LB Recall | F-score |
|---|---|---|---|
| All | 84.18 | 83.74 | 83.96 |
| $\leq 70$ | 84.82 | 84.35 | 84.58 |

Table 5.2: C-structure parsing performance.

The results of the evaluation of c-structure parsing performance on the test set are shown in Table 5.2. Labelled bracketing f-score for all sentences is just below 84% for all sentences, and 84.58% for sentences of length $\leq 70$. In comparison, Cowan and Collins (2005) report an f-score of 85.1% ($\leq 70$) using a version of Collins' parser adapted for Cast3LB, and additionally using reranking to boost performance. They use a different, more reduced category label set as well as a different training-test split. Both Cowan and Collins (2005) and the present thesis report scores which ignore punctuation.

**Cast3LB function labeling** For the task of Cast3LB function label assignment I experimented with three machine learning algorithms: a memory-based learner (Daelemans and van den Bosch, 2005), a maximum entropy classifier (Berger et al., 1996) and a Support Vector Machine classifier (Vapnik, 1998). For each algorithm I used the same set of features to represent parse-tree nodes that are to be assigned one of the Cast3LB function labels. I used a special *null* label for nodes where no Cast3LB function label is present. Unlike in the case of the English Penn-II treebank, in Cast3LB a given node can only have a single function label. Thus we can train a single multiclass model, rather than a separate binary classification model for each label as in Section 5.3.

In Cast3LB only nodes in certain contexts are eligible for function labels. For this reason I only consider a subset of all nodes as candidates for function label assignment, namely those which are sisters of nodes with the category labels *gv* (Verb Group), *infinitiu* (Infinitive) and *gerundi* (Gerund). I extract the following three types of features encoding configurational, morphological and lexical information for the target node and neighboring context nodes:

- Node features:
  - position relative to head

Figure 5.1: Examples of features extracted from an example node

  - head lemma
  - alternative head lemma (i.e. the head of NP in PP)
  - head POS
  - category
  - definiteness
  - agreement with head verb
  - constituent yield length
  - human/nonhuman (according to Spanish Wordnet (Vossen, 1998))

- Local features:
  - head verb
  - verb person
  - verb number
  - parent category
- Context features: node features (except position) of the two previous and two following sister nodes (if present).

Figure 5.1 illustrates features extracted from an example node.

I used cross-validation for refining the set of features and for tuning the parameters of the machine-learning algorithms. I did not use any additional automated feature-selection procedure.

I made use of the following implementations: TiMBL (Daelemans et al., 2004) for Memory-Based Learning, the MaxEnt Toolkit (Le, 2004) for Maximum Entropy and LIBSVM (Chang and Lin, 2001) for Support Vector Machines.

Figure 5.2: Learning curves for TiMBL (t), MaxEnt (m) and SVM (s).

For TiMBL I used $k$ nearest neighbors = 7, the Jeffrey Divergence as the distance metric, the Inverse Distance for distance-weighted class voting, and the Gain Ratio metric for feature weighting. Those options were found using the Paramsearch algorithm (van den Bosch, 2004). Daelemans and van den Bosch (2005) describe the various options for TiMBL in detail. For MaxEnt, I regularized the model using a Gaussian prior with $\sigma^2 = 1$. For SVM I used the RBF kernel with the kernel parameter $\gamma = 2^{-7}$ and the cost parameter $C = 32$.

### 5.2.4 Cast3LB function label assignment evaluation

I present evaluation results on the original gold-standard trees of the test set as well as on the test-set sentences parsed by Bikel's parser. For the evaluation of Cast3LB function labeling performance on gold trees the most straightforward metric is the accuracy, or the proportion of all candidate nodes that were assigned the correct label.

However we cannot use this metric for evaluating results on the parser output. The trees output by the parser are not identical to gold standard trees due to parsing errors, and the set of candidate nodes extracted from parsed trees will not be the same as for gold trees. For this reason I use two alternative metrics which are independent of tree configuration and use only the Cast3LB function labels and positional indices of tokens in a sentence: the Node Span metric and the Headword metric.

In Node Span, for each function-labeled tree I first remove the punctuation tokens. Then I extract a set of tuples of the form $\langle \gamma, i, j \rangle$, where $\gamma$ is the Cast3LB function label and $i..j$ is the range of tokens spanned by the node annotated with this function. I use the standard measures of precision, recall and f-score to evaluate the results. For parser-output trees I apply this metric on the correctly bracketed subset of nodes, since the evaluation should not be sensitive to what function label, if any, was assigned to incorrect bracketings.

The Headword metric ignores constituent bracketing altogether and only considers

| | Acc. | Prec. | Recall | F-score |
|---|---|---|---|---|
| MBL | 87.55 | 87.00 | 82.98 | 84.94 |
| MaxEnt | 88.06 | 87.66 | 86.87 | 85.52 |
| SVM | **89.34** | 88.93 | 84.90 | **86.87** |

Table 5.3: Cast3LB function labeling performance for gold-standard trees (Node Span)

| | Precision | Recall | F-score |
|---|---|---|---|
| Baseline | 72.63 | 75.35 | 73.96 |
| MBL | 78.09 | 78.75 | 78.42 |
| MaxEnt | 78.90 | 79.44 | 79.17 |
| SVM | 80.58 | 81.27 | **80.92** |

Table 5.4: Cast3LB function labeling performance for parser output (Node Span: correctly parsed constituents)

whether constituent headwords are assigned the correct function label. For this measure we project the function label of each constituent down to its head terminal, and measure precision and recall on sets of tuples $\langle \gamma, i \rangle$ where $\gamma$ is the function label and $i$ is the index of the terminal bearing it after projection. The Headword metric does not consider constituent bracketings and can be applied directly to parser output without filtering out incorrectly parsed nodes: thus it will give lower absolute numbers than Node Span. It can be argued that it better approximates how useful the function-labeled tree would be for an application which needs to recover the basic argument structure of a sentence. In most such scenarios it is much more important to assign correct function to head words than to get all constituent boundaries right.

Results for the three algorithms on gold-standard trees are shown in Table 5.3. Precision, Recall and F-score were computed with the Node Span method. The SVM outperforms both MBL and MaxEnt, scoring 89.34% on accuracy and 86.87% on f-score. The learning curves for the three algorithms, shown in Figure 5.2, are also informative, with SVM outperforming the other two methods for all training set sizes. In particular, the last section of the plot shows SVM performing almost as well as MBL with half as much learning material.

Table 5.4 shows the performance of the three methods on parser output with the Node Span metric on correctly bracketed nodes. The baseline contains the results achieved by treating compound category-function labels as atomic during parser train-

| | Precision | Recall | F-score |
|---|---|---|---|
| Baseline | 69.19 | 70.77 | 69.97 |
| MBL | 74.85 | 73.92 | 74.38 |
| MaxEnt | 75.78 | 74.84 | 75.30 |
| SVM | 77.29 | 76.44 | **76.86** |

Table 5.5: Cast3LB function labeling performance for parser output (Headword)

| Methods | $p$-value |
|---|---|
| Baseline vs SVM | $1.169 \times 10^{-9}$ |
| Baseline vs MBL | $2.117 \times 10^{-6}$ |
| MBL vs MaxEnt | 0.0799 |
| MaxEnt vs SVM | 0.0005 |

Table 5.6: Statistical significance testing results on for the Cast3LB tag assignment on parser output.

ing so that they are included in parser output. Again the best algorithm turns out to be SVM. It outperforms the baseline by a large margin: 6.96% absolute f-score or 26.73% relative error reduction.

Table 5.5 show the results computed with the Headword metric: as expected the absolute values are lower, but the overall ranking of the four methods remains the same, with SVM performing best.

The difference in performance for gold standard trees, and the correctly parsed constituents in parser output is rather larger than what Blaheta and Charniak (2000) report. Further analysis is needed to identify the source of this difference but one contributing factor may be our use of a greater number of context features, combined with a higher parse error rate in comparison to their experiments on the Penn II Treebank. Since any mis-analysis of constituency structure in the vicinity of the target node can have a negative impact, greater reliance on context means greater susceptibility to parse errors. Another factor to consider is the fact that I trained and adjusted parameters on gold-standard trees, and the model learned may rely on features of those trees that the parser is unable to reproduce: I present research addressing this issue in Section 5.3.

For the experiments on parser output I performed a series of sign tests in order to determine to what extent the differences in performance between the different methods

| | Precision | Recall | F-score |
|---|---|---|---|
| Upper bound | 97.80 | 97.28 | 97.54 |
| Baseline | 73.95 | 70.67 | 72.27 |
| SVM | 76.90 | 74.48 | 75.67 |

Table 5.7: LFG F-structure evaluation results (preds-only) for parser output

are statistically significant. For each pair of methods I calculate the f-score (using Node Span) for each sentence in the test set. For those sentences on which the scores differ, i.e. the number of trials, the number of cases that the second method is better is the number of successes. Then I run the binomial test with the null hypothesis that the probability of success is chance (= 0.5) and the alternative hypothesis that the probability of success is greater than chance (> 0.5). The results are summarized in Table 5.6. Given that I perform 4 pairwise comparisons, I apply the Bonferroni correction and adjust the target $\alpha_\beta = \frac{\alpha}{4}$. For the confidence level 95% ($\alpha_\beta = 0.0125$) all pairs give statistically significant results, except for MBL vs MaxEnt.

### 5.2.5 Task-based LFG annotation evaluation

Finally, I also evaluated the actual f-structures obtained by running the LFG-annotation algorithm on trees produced by the parser and enriched with Cast3LB function labels assigned using SVM. For this task-based evaluation I produced a gold standard consisting of f-structures corresponding to all sentences in the test set. The LFG-annotation algorithm was run on the test set trees (which contained original Cast3LB treebank function labels), and the resulting f-structures were manually corrected.

Following Crouch et al. (2002), I convert the f-structures to triples of the form $\langle GF, P_i, P_j \rangle$, where $P_i$ is the value of the PRED attribute of the f-structure, $GF$ is an LFG grammatical function attribute, and $P_j$ is the value of the PRED attribute of the f-structure which is the value of the $GF$ attribute. This is done recursively for each level of embedding in the f-structure. Attributes with atomic values are ignored for the purposes of this evaluation: this is referred to as preds-only evaluation. The results obtained are shown in Table 5.7. The upper bound on this task is determined by the preds-only score for f-structure evaluation when the input trees are the original treebank

| | ATR | CC | CD | CI | CREG | MOD | SUJ |
|---|---|---|---|---|---|---|---|
| ATR | 136 | 2 | 0 | 0 | 0 | 0 | 5 |
| CC | 6 | 552 | 12 | 4 | 25 | 18 | 6 |
| CD | 1 | 19 | 418 | 5 | 3 | 0 | 26 |
| CI | 0 | 6 | 1 | 50 | 1 | 0 | 0 |
| CREG | 0 | 6 | 0 | 2 | 43 | 0 | 0 |
| MOD | 0 | 0 | 0 | 0 | 0 | 19 | 0 |
| SUJ | 0 | 8 | 24 | 2 | 0 | 0 | 465 |

Table 5.8: Simplified confusion matrix for SVM on test-set gold-standard trees. The gold-standard Cast3LB function labels are shown in the first row, the predicted tags in the first column. So e.g. SUJ was mistagged as CD in 26 cases. Low frequency function labels as well as those rarely mispredicted have been omitted for clarity.

trees. I also performed a statistical significance test for these results. The $p$-value given by the sign test $2.118 \times 10^{-5}$, showing that the improvement is statistically significant.

The score achieved in the LFG f-structure evaluation in (Table 5.7) is comparable to the Cast3LB tag assignment evaluation using the Headword metric (Table 5.5), but not identical: in particular the difference between the baseline score and the SVM score is smaller for the f-structure evaluation. This is can be attributed to the fact that the mapping from Cast3LB function labels to LFG grammatical functions is not one-to-one. For example three Cast3LB tags (CC, MOD and ET) are all mapped to LFG ADJUNCT. Thus mistagging a MOD as CC does not affect the f-structure score. On the other hand the Cast3LB CD tag can be mapped to OBJ, COMP, or XCOMP, and it can be easily decided which one is appropriate depending on the category label of the target node. Additionally many nodes which receive no function label in Cast3LB, such as noun modifiers, are straightforwardly mapped to LFG ADJUNCT. Similarly, objects of prepositions receive the LFG OBJ function.

### 5.2.6 Error analysis

In order to understand sources of error and determine how much room for further improvement there is, I examined the most common cases of Cast3LB function mistagging. A simplified confusion matrix with the most common Cast3LB tags is shown in Table 5.8. The most common mistakes occur between SUJ and CD, in both directions, and also many CREGs are erroneously tagged as CC.

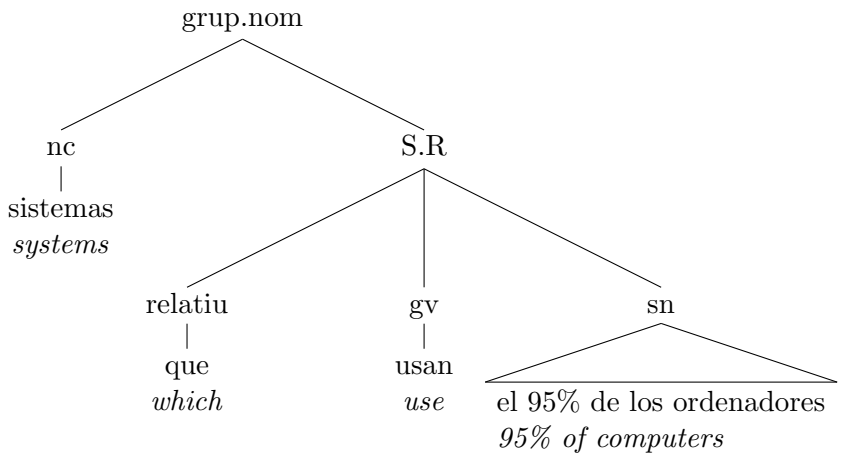



Figure 5.3: Subject - Direct Object ambiguity in a Spanish relative clause.

**Subject vs Direct Object** I noticed that in over 50% of cases when a Direct Object (CD) was misidentified as Subject (SUJ), the target node's mother was a relative clause. It turns out that in Spanish relative clauses genuine syntactic ambiguity is not uncommon. Consider the following Spanish phrase:

(5.1) *Sistemas que usan el 95% de los ordenadores.*
Systems which use DET 95% of DET computers

The c-structure tree for this phrase is shown in Figure 5.3. Its translation into English is either *Systems that use 95% of computers* or alternatively *Systems that 95% of computers use*. In Spanish, unlike in English, preverbal / post-verbal position of a constituent is not a good guide to its grammatical function in this and similar contexts. Human annotators can use their world knowledge to decide on the correct semantic role of a target constituent and use it in assigning a correct grammatical function, but such information is not explicitly encoded in the features and thus not exploited by the machine learning methods. Thus such mistakes seem likely to remain unresolvable in the current approach.

**Prepositional Object vs Adjunct** The frequent misidentification of Prepositional Objects (CREG) as Adjuncts (CC) seen in Table 5.8 can be accounted for by several factors. Firstly, Prepositional Objects are strongly dependent on specific verbs and the

comparatively small size of our training data means that there is limited opportunity for a machine-learning algorithm to learn low-frequency lexical dependencies. Here the obvious solution is to use a more adequate amount of training material when it becomes available.

A further problem with the Prepositional Object - Adjunct distinction is its inherent fuzziness. Because of this, treebank designers may fail to provide easy-to-follow, clear-cut guidelines and human annotators necessarily exercise a certain degree of arbitrariness in assigning one or the other function.

### 5.2.7 Adapting to the AnCora-ESP corpus

Recently an expanded and modified version of the Spanish Cast3LB treebank became available (Martí et al., 2007). This new treebank, the Spanish AnCora treebank[2], roughly doubles the size of Cast3LB: it has approximately 200.000 words.[3]

The annotation scheme roughly follows the Cast3LB guidelines. However, some minor modifications have been introduced. The verb group constituent label *gv* has been renamed to *grup.verb*, and a number of additional phrasal levels have been introduced: adjective group (*grup.a.ms, grup.a.fs* etc), adverb group (*grup.adv*) and participle (*participi*). Additional distinctions have been introduced for noun phrases referring to different types of named entities (*snp, sno* etc). Also a new function label AO, sentential adjunct, was introduced. Additional distinctions were introduced for verbal adjuncts: CCT stands for temporal and CCL for location adjunct. Analyses for many Cast3LB sentences have been modified to correct errors or to reflect the revised annotation guidelines.

In order to adapt the LFG grammar acquisition and parsing architecture to this new resource the following modifications were needed:

- The label reduction scheme had to be changed to collapse some of the distinctions introduced by new constituent labels. The new labels were reduced to *grup.a, grup.adv* and *sn*.

[2] Also known as CESS-ESP.

[3] It is claimed to contain 500.000 words in (Martí et al., 2007), but the currently distributed version has only 200.000.

| | LB Precision | LB Recall | F-score |
|---|---|---|---|
| All | 84.18 | 83.74 | 83.96 |
| $\leq 70$ | 84.82 | 84.35 | 84.58 |

Table 5.9: C-structure parsing performance for Cast3LB

| | LB Precision | LB Recall | F-Score |
|---|---|---|---|
| All | 83.82 | 83.35 | 83.58 |
| $\leq 70$ | 84.37 | 83.88 | 84.13 |

Table 5.10: C-structure parsing performance for AnCora

- Minor modifications were made in the head table to account for new constituent labels.
- Some modifications were necessary in the annotation algorithm because of new constituent and function labels
- The 338 f-structures corresponding to the test-set sentences needed to be revised to reflect the modifications in the analyses introduced in AnCora trees

For development and testing I used the same set of files as in the experiments with Cast3LB. All the other files in AnCora served as training set. This means that while in the Cast3LB experiments I had development and test sets which contained a random sample of files from the whole corpus, this is no longer the case for the AnCora corpus; specifically all of the additional training files seem to come from newspaper text exclusively.

| | Precision | Recall | F-score |
|---|---|---|---|
| Baseline | 72.63 | 75.35 | 73.96 |
| SVM | 80.58 | 81.27 | 80.92 |

Table 5.11: Cast3LB function labeling performance for parser output (Node Span: correctly parsed constituents)

| | Precision | Recall | F-score |
|---|---|---|---|
| Baseline | 71.68 | 73.19 | 72.42 |
| SVM | 80.48 | 79.03 | 79.75 |

Table 5.12: AnCora function labeling performance for parser output for correctly parsed constituents

| | Precision | Recall | F-score |
|---|---|---|---|
| Baseline | 73.95 | 70.67 | 72.27 |
| SVM | 76.90 | 74.48 | 75.67 |

Table 5.13: LFG F-structure evaluation results (preds-only) for parser output for Cast3LB

**Results and discussion**

Table 5.10 shows the evaluation results for c-structure parsing; Table 5.12 shows results for function labeling. Table 5.14 presents the evaluation results for f-structure parsing on the f-structures revised to reflect the revised AnCora trees. The corresponding Cast3LB results are repeated here for ease of comparison.

Because of the differences in annotation between Cast3LB and AnCora the results here are not strictly comparable to those reported in the previous section. However a broad conclusion seems warranted: it seems that doubling the training set size does not boost either the c-structure parsing or function labeling scores on the test set. This is likely related to the fact mentioned above: the test-set files are not a random selection from the AnCora corpus. Compared to the test-set, the additional training sentences are likely to diverge in domain; it is also possible that there are annotation divergences between the old and the new part of the corpus. As in the original Cast3LB evaluation, the system using the SVM function labeler performs better than the baseline system.

| | Precision | Recall | F-score |
|---|---|---|---|
| Baseline | 75.37 | 72.77 | 74.04 |
| SVM | 77.21 | 74.27 | 75.71 |

Table 5.14: LFG F-structure evaluation results (preds-only) for parser output for AnCora

## 5.3 Improving Training for Function Labeling by Using Parser Output[4]

### 5.3.1 Introduction

In this section I apply the SVM-based function labeling method developed for Spanish in the previous section to the English and Chinese treebanks, and investigate how to improve its performance by training the model on data more closely resembling the final test instances.

The function labels used in the Penn treebanks fall into several types. Grammatical labels are used to encode the grammatical function of some constituents. Form-function labels are used to indicate the semantic class of adjuncts and discrepancies between form and function. There is also a label used for topicalization, and several other miscellaneous labels. Detailed information about the label sets can be found in the annotation guidelines for the respective treebanks (Bies et al., 1995; Xue and Xia, 2000). Table 5.15 provides a summary of labels used in the English and Chinese treebanks.

In this section I present research on the impact of different training methods in a two-stage processing architecture where I use machine learning techniques to train classifiers which add function labels to bare constituent trees such as those output by Charniak's or Collins' parsers.

In a multi-stage processing pipeline the optimal training input for the downstream stages is important. Ideally the training at stage $n+1$ should be performed on input from stage $n$: e.g. a parsing model which uses automatically POS-tagged input should be trained on tags produced by the POS tagger used to preprocess the raw input, rather than gold tags. In many existing pipeline architectures this has been violated.

For example, in the case of function labeling, the two-stage models used in previous work have all used "perfect" treebank trees to train the function labeler even though the labeler operates on "imperfect" trees output by the parser (Blaheta and Charniak, 2000; Jijkoun and de Rijke, 2004; Chrupała and van Genabith, 2006b). This

[4]The research in this section has been done in collaboration with Nicolas Stroppa and Georgiana Dinu. Their main contribution was to help come up with and implement instance similarity metrics. Stroppa also proposed extending the method to use n-best parser output trees (which was implemented but is not discussed here as it did not consistently improve performance).

is presumably due to the fact that the function labels we want to learn are attached to nodes in the treebank trees. Unfortunately, those nodes do not necessarily correspond to constituents in the trees produced by the parser.

The main contribution of this section consists in presenting a theoretically sound method of training on parser output rather than treebank trees for the function labeling task and investigating the effect of versions of this approach on the results as compared against the baseline method which uses perfect treebank trees. I show that using the better-motivated method helps to improve the quality and quantity of training material available to the machine-learning algorithm.

In Section 5.3.2 I present the improved method of obtaining appropriate training material for function labeling. In Section 5.3.3 I present experimental results for English and Chinese.

### 5.3.2 Methods

There are two main approaches to obtaining parse trees with function label information:

- Two-stage systems, where "bare" parse trees are enriched with function labels in a postprocessing step (Blaheta and Charniak, 2000; Jijkoun and de Rijke, 2004; Chrupała and van Genabith, 2006b),
- Modifying the parser's internals to output function labels (O'Donovan et al., 2005; Musillo and Merlo, 2005; Merlo and Musillo, 2005; Gabbard et al., 2006).

I use the two-stage architecture, in which the first stage consists of bare constituency parsing using a statistical parsing model and the second stage decorates constituent labels with function labels. The labeler is a machine-learning classification model. My focus is to investigate ways of improving the performance of the classifier by extracting more and better quality training examples from the available resources.

Improving the quality of the training material means making it more similar to the instances that the model has to classify during prediction, i.e. we will try to better approximate the standard assumption made in most machine-learning research that instances (in training and test) are *independently and identically distributed* (i.i.d.), in particular, they should be drawn from the same probability distribution.

| Label | Meaning | ETB | CTB |
|---|---|---|---|
| **Clause types** | | | |
| IMP | imperative | | √ |
| Q | question | | √ |
| **Syntactic labels** | | | |
| LGS | logical subject | √ | √ |
| PRD | predicate | √ | √ |
| PUT | complement of put | √ | |
| SBJ | surface subject | √ | √ |
| IO | indirect object | | √ |
| OBJ | direct object | | √ |
| FOC | focus | | √ |
| **Miscellaneous labels** | | | |
| CLF | it-cleft | √ | |
| HLN | headline | √ | √ |
| TTL | title | √ | √ |
| CLR | closely related | √ | |
| APP | appositive | | √ |
| PN | proper noun | | √ |
| SHORT | short form | | √ |
| WH | WH-phrases | | √ |
| **Semantic (form-function) labels** | | | |
| ADV | adverbial | √ | √ |
| BNF | benefactive | √ | √ |
| DIR | direction | √ | √ |
| EXT | extent | √ | √ |
| LOC | locative | √ | √ |
| MNR | manner | √ | √ |
| NOM | nominal | √ | |
| PRP | purpose or reason | √ | √ |
| TMP | temporal | √ | √ |
| CND | condition | | √ |
| IJ | interjective | | √ |
| VOC | vocative | √ | √ |
| **Topicalization** | | | |
| TPC | topicalized | √ | √ |

Table 5.15: Function labels in the English and Chinese Penn Treebanks

In the previous two-stage approaches (Blaheta and Charniak, 2000; Jijkoun and de Rijke, 2004; Chrupała and van Genabith, 2006b) this assumption is not satisfied in that the training instances are extracted from nodes in the "perfect" parse trees from the treebank, whereas at prediction time the model has to classify instances extracted from nodes in imperfect parser output, which can and does contain a certain proportion of errors (incorrect bracketings or incorrect constituent labels).

I propose to alleviate this issue by using training material which is extracted from the trees obtained by reparsing the training portion of the treebank and using the (imperfect) trees output by the parser rather than the original treebank trees. We still need the original treebank trees in order to assign classes (function labels) to the training instances extracted from parser output. I do this by matching node-spans between automatically parsed trees and gold trees in the training set. I only extract training instances from those nodes in the automatically parsed tree for which there is a node with the same span in the gold tree, from which we can obtain the function label.

#### Baseline method

The baseline method uses a simple two-stage architecture: constituency parsing, followed by function labeling: this is the setup used in the experiments described in Section 5.2. The first stage is performed by the constituency parsing model, obtained by training a statistical parser on the training portion of the treebank. The output of this stage, sentences parsed into bare constituency trees, are the input to the second stage component, i.e. the function labeler. The labeler is trained, in the baseline method, on the original "perfect" trees from the training portion of the treebank.

**Features** Each node to label is represented as a vector of features encoding categorial, configurational and lexical information about the node and its context. Those features are similar to the ones used in Section 5.2.3 for Spanish. However, the highly language-specific features have been dropped ("humanness" according to Wordnet, definiteness and agreement features). Also due to the fact that Penn trees are less shallow and more hierarchical than Cast3LB trees, only one preceding and one following context

nodes are used, but the extra grandmother constituent label feature is added. The full feature set is as following:

1. Node constituent label
2. Node head word's part of speech tag
3. Node head word
4. Node's head-sister's constituent label
5. Node's head-sister's head word's part of speech
6. Node's head-sister's head word
7. Node's alternative head word's part of speech tag (alternative head is the head of the second child for PPs)
8. Node's alternative head word
9. Node's yield length
10. Node's mother's constituent label
11. Node's grandmother's constituent label
12. Offset to node's head sister

Plus the following:

- Features 1,2,3,7,8,9 for the preceding sister node
- Features 1,2,3,7,8,9 for the following sister node

There is one minor complication: in principle a node can be decorated with more than one function label (although labels belonging to the same group are (usually) mutually exclusive). Thus we could train a separate classifier for each label, or a separate classifier for each label group, or simply treat the label set on the node as an atomic class. In the experiments reported below I used the first method, i.e. I train a separate binary classifier for each function label, and combine their output to add a set of function labels to each node.

### Evaluation metrics

As noted in Section 5.2.4, evaluating the performance of a function labeling system is not entirely straightforward. Blaheta and Charniak (2000) decide to measure f-score over the *correctly parsed* subset of nodes, i.e. those nodes that subtend the correct portion of the string and have the correct constituent label. This is similar to the Node Span metric introduced in Section 5.2.4, modulo the fact that here both the bracketing and the constituent label are used to determine the set of correctly parsed nodes. I will refer to this metric as Labeled Node Span, and use it for evaluation in order to make comparison to previous work meaningful. Additionally I also present evaluation results using the Headword method.

### Training on parser output

Using Labeled Node Span described above, since we are evaluating only the correctly parsed subset of nodes, one might naively expect that the score should be the same for labeling both the parser output and the perfect treebank trees. However, the results reported in (Blaheta and Charniak, 2000) show that the performance is over 1% better for the treebank trees. The authors convincingly explain that the likely cause is that for parser output, although the focus node to be labeled is correctly parsed, the neighboring context nodes that some features depend on may be incorrect.

This fact serves as the motivation for extracting training examples from treebank sentences parsed by the same parser that is used to parse unseen test data. My hypothesis is that training instances obtained in this way are going to be more similar to test instances than the ones extracted from perfect treebank trees and thus will better approximate the i.i.d assumption. I expect that the machine learning algorithm will perform better on test instances which are more similar to those used for training; for example it might be able to weight down features which depend on incorrect characteristics of the parse trees, as such features will be less reliable as class predictors.

The improved training example extraction procedure is as follows: sentences in the training portion of the treebank are reparsed. Then we follow the algorithm presented in Figure 5.4 to extract training instances. The function INSTANCES returns training instances from a parse tree $T$ given the reference treebank gold tree $T'$ for the same

1 INSTANCES$(T, T') =$
2 $\mathcal{N} \leftarrow \{\text{NODESPEC}(n) \mid n \in \text{NODESET}(T')\}$
3 $\mathbf{I} \leftarrow [\ \text{INSTANCE}(n, T') \mid n \in \text{NODEMULTISET}(T) \wedge \text{NODESPEC}(n) \in \mathcal{N}\ ]$
4 **return** $\mathbf{I}$
5 INSTANCE$(n, T') =$
6 $\mathcal{C} \leftarrow \bigcup \{\text{FUNCLABELS}(n') \mid n' \in \text{NODESET}(T') \wedge \text{NODESPEC}(n') = \text{NODESPEC}(n)\}$
7 **return** $\langle \text{FEATURES}(n), \mathcal{C} \rangle$
8 NODESPEC$(n) = \langle \text{NODESPAN}(n), \text{NODECONSTITUENTLABEL}(n) \rangle$

Figure 5.4: Algorithm for extracting training instances from a parser tree $T$ and gold tree $T'$

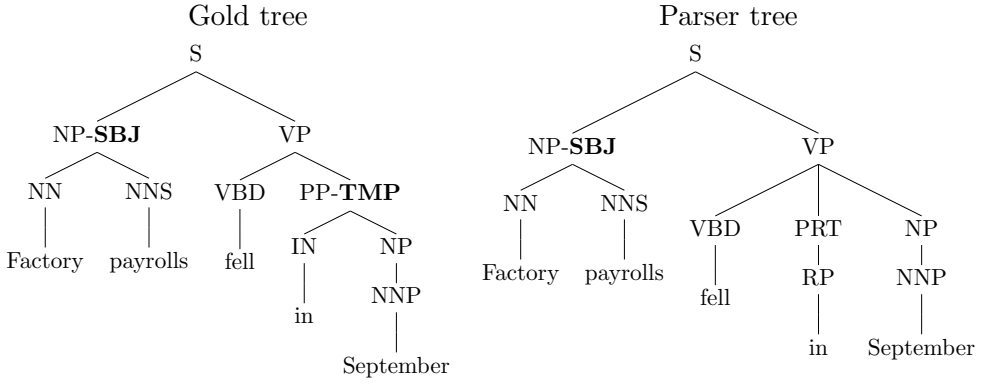



Figure 5.5: Example gold and parser tree

sentence. For each node $n$ in $T$ we check whether there exist one or more nodes with the same span and constituent label in the corresponding $T'$ (line 3)[5]. The function INSTANCE takes the union of the function label sets (FUNCLABELS$(n')$) found on the nodes in the gold tree $T'$ and returns this set (as a class $\mathcal{C}$) together with the feature vector FEATURES$(n)$ corresponding to node $n$.

Figure 5.5 illustrates this algorithm: in effect we transfer function labels from nodes in the gold tree to matching nodes in the parser tree. Matching nodes are those whose constituent label and span are the same. In the example tree the SBJ function label is transferred but TMP is not since there is no matching node in the parser tree due to a parsing error.

An issue with the method as described so far is that it uses a constituency parsing model trained on sections 2-21 of WSJ to reparse those same sections so that we can

[5]The square bracket notation denotes multisets.

extract training material from them. It is very likely that the resulting parse trees will be closer to gold trees than will be the case for test sentences taken from WSJ section 23. It would be advisable to extract input for our labeling model from the treebank trees reparsed with parsing models trained on material from which those trees are excluded. I did not do this for the experiments on the English data with Charniak's parser, due to technical difficulties encountered when attempting to retrain this parser. However, for the experiments on the Chinese data with Bikel's parser I did 10-fold-cross-training, that is I divided the training material into 10 parts and parsed each part in turn with the model trained on the remaining 9 parts. I report the results on the Chinese data in Section 5.3.3.

**Instance similarity** I tried to verify the prediction that the instances extracted using the "reparsing" method would be more similar to test instances. As a simple metric of similarity, I compare instance overlap between the training set and the test set. Instance overlap is the cardinality of the intersection of the multiset of instance feature vectors used for training and the multiset of instance feature vectors used for testing. For multisets defined as tuples $(A, f)$ with the underlying set $A$ and the multiplicity function $f : A \rightarrow \mathbb{N}$ which assigns to each element the number of times it occurs, multiset cardinality is defined as:

$$|(A, f)| = \sum_{a \in A} f(a), \tag{5.2}$$

and multiset intersection as:

$$(A, f) \cap (B, g) = (A \cap B, a \mapsto min(f(a), g(a))). \tag{5.3}$$

I use both the baseline method where examples are extracted from gold trees, and the improved "reparsing" training method to obtain training examples from sections 2-21 of the Wall Street Journal part of the English Penn Treebank and compare both against instances extracted from the parsed sentences taken from section 23. For parsing the test sentences and the training sentences I used the Charniak parser.

Table 5.16 summarizes the comparison. Even though the improved method produces

| | Instance count | Overlap |
|---|---|---|
| Test | 44,113 | — |
| Baseline | 741,833 | 9,067 |
| Reparse | 712,973 | 10,022 |

Table 5.16: Instance counts and instance overlap against test for the English Penn Treebank training set

| | Mean distance to Test |
|---|---|
| Test | 15.0999 |
| Baseline | 15.1483 |
| Reparse | 15.1283 |

Table 5.17: Mean Hamming distance scores for the English Penn Treebank training set

a lower total number of instances than the baseline (since I only extract instances from correctly spanning nodes) it still shares 955 instances more with the test set than the baseline.

To further test the conjecture about the reparsing method giving better training examples I calculated mean Hamming distance between training examples and test examples. Hamming distance counts the number of features at which two vectors differ:

$$d^h(\mathbf{v}, \mathbf{w}) = \sum_{i=1}^{|\mathbf{v}|} \mathbf{v}_i \neq \mathbf{w}_i \ . \tag{5.4}$$

We calculate the mean distance between the collection of test instances $\mathbf{T}$ and the collection of training instances $\mathbf{U}$ as:

$$\bar{d^h}(\mathbf{T}, \mathbf{U}) = \frac{1}{|\mathbf{T}| \times |\mathbf{U}|} \sum_{\mathbf{t} \in \mathbf{T}} \sum_{\mathbf{u} \in \mathbf{U}} d^h(\mathbf{t}, \mathbf{u}) \ . \tag{5.5}$$

As shown in Table 5.17, against the test set derived from section 23 of WSJ we get mean Hamming distance of 15.1483 for the baseline method and 15.1283 for the reparsing method (for comparison the mean distance of the test set against itself is 15.099). According to this metric examples obtained by my method are more similar to test examples.

### 5.3.3 Experimental results

In this section I present evaluation results on the function labeling task for two datasets:

- Section 23 of the WSJ portion of the English Penn II Treebank, with models trained on data extracted from sections 2-21. Section 22 was used for development. The Charniak parser[6] was used for constituency parsing.
- Articles 271 to 300 of the Penn Chinese Treebank 5, with models trained on data extracted from articles 26 to 270. Articles 1-25 were used for development. I follow (Levy and Manning, 2003) in adopting this test/training/development split. The Bikel parser[7] was used for constituency parsing.

For both datasets I used the LIBSVM library (Chang and Lin, 2001) which implements the Support Vector Machines algorithm (Vapnik, 1998).

#### Experiments with the English Penn Treebank

Table 5.18 summarizes evaluation results for the function labeling task on the English Penn II Treebank. I report the scores for three methods. The *baseline* consists in extracting training data from the treebank trees. *Reparse* is the training method described in Section 5.3.2 where nodes in the reparsed trees are mapped to labels in the original treebank trees. *Aux-POS* consists in extracting the training data from the treebank trees; however the terminal nodes in those trees are matched to corresponding terminals in the reparsed trees and if the reparsed tree has the AUX or AUXG part-of-speech tag, then the treebank POS tag is changed to that. This method explicitly takes care of the fact that the Charniak parser's POS tagger assigns AUX and AUXG tags to some words which have verb POS tags in the original treebank.

Additionally I tried an alternative reparsing method, *Reparse-HW*. This is a variation on the *Reparse* method: the difference is the way nodes are matched between the gold trees and the parser trees in order to add function labels to the parser trees: in *Reparse* we match on constituent label and span; in *Reparse-HW* we match just on the constituent headword index. This more lax way of node matching allows us to

[6] Available at `ftp://ftp.cs.brown.edu/pub/nlparser/`
[7] Available at `http://www.cis.upenn.edu/~dbikel/software.html#stat-parser`

use more of the gold function labels from treebank trees and increase the amount of training data. Since in training constituent bracketing is ignored, I use the Headword metric for evaluating the performance of *Reparse-HW* performance since Headword also ignores the constituent boundaries.

Table 5.19 presents the scores for all four methods using Headword: *Reparse-HW* is slightly worse than *Reparse*: the increase in the number of training examples does not in this case translate to improved performance.

Table 5.18 shows the results for the first three methods, using Labeled Node Span to compute precision, recall and f-score. There is a clear increase in f-score over the baseline for the reparsing method, which gives a relative error reduction of almost 8.5% over the baseline. The approximate randomization test (Noreen, 1989) with $10^6$ shuffles obtained a $p$-value of $10^{-7}$ for the baseline versus the reparsing method, showing that the improvement is statistically significant.

The score of the *Aux-POS* method shows that a large part of the improvement obtained with *reparse* can be attributed to the correction of mismatches between the training and test data at the level of POS tag, namely to the presence of AUX and AUXG tags in the output of Charniak's parser. The *Reparse* method corrects this discrepancy between training and test trees automatically, together with other possible divergences.

The results (91.47% f-score) are the best scores published to date on the function labeling task evaluated on parser output on the section 23 of WSJ: 87.27% in (Blaheta and Charniak, 2000), 88.5% in (Jijkoun and de Rijke, 2004) and 88.96% in (Gabbard et al., 2006)[8]

Table 5.20 shows the performance broken down per function label. Although performance on three labels (LOC, LGS and PRP) drops, the rest of the labels show the same score or benefit from the enhanced training method.

[8]Not all of those scores are exactly comparable to ours or to each other. The score in (Jijkoun and de Rijke, 2004) is on trees transformed into dependencies. Gabbard et al. (2006) use Bikel's parser to produce the trees whereas we use Charniak's.

| | Precision | Recall | F-score |
|---|---|---|---|
| Baseline | 92.28 | 89.14 | 90.68 |
| Aux-POS | 92.67 | 89.98 | 91.31 |
| Reparse | 93.07 | 89.92 | **91.47** |

Table 5.18: Function labeling evaluation on parser output for WSJ section 23 - Labeled Node Span

| | Precision | Recall | F-score |
|---|---|---|---|
| Baseline | 88.25 | 84.23 | 86.19 |
| Aux-POS | 88.16 | 84.74 | 86.42 |
| Reparse | 88.53 | 84.62 | **86.53** |
| Reparse-HW | 88.37 | 84.69 | 86.49 |

Table 5.19: Function labeling evaluation on parser output for WSJ section 23 - Headword

| Label | Freq. in test | Baseline | Reparse |
|---|---|---|---|
| SBJ | 4148 | 98.27 | 98.27 |
| TMP | 1303 | 91.19 | 91.52 |
| PRD | 1025 | 68.35 | 91.26 |
| LOC | 1024 | 89.45 | 89.06 |
| CLR | 635 | 68.98 | 68.93 |
| ADV | 419 | 85.98 | 89.36 |
| DIR | 293 | 68.98 | 71.20 |
| TPC | 267 | 86.50 | 96.02 |
| PRP | 207 | 68.35 | 67.95 |
| NOM | 199 | 95.02 | 95.58 |
| MNR | 178 | 76.12 | 77.62 |
| LGS | 166 | 88.10 | 88.10 |
| EXT | 105 | 87.72 | 88.24 |
| TTL | 61 | 74.42 | 74.42 |
| HLN | 52 | 18.18 | 26.23 |
| DTV | 19 | 66.67 | 66.67 |
| PUT | 10 | 66.67 | 66.67 |
| CLF | 3 | — | — |
| BNF | 2 | — | — |
| VOC | 1 | — | — |

Table 5.20: Per-tag performance of baseline and when training on reparsed trees - Labeled Node Span

| | Precision | Recall | F-score |
|---|---|---|---|
| Baseline | 88.35 | 84.64 | 86.46 |
| Reparse | 88.54 | 84.82 | 86.64 |
| Reparse + x-train | 89.11 | 84.88 | 86.94 |

Table 5.21: Function labeling evaluation for the CTB on the parser output for the development set

**Experiments with the Penn Chinese Treebank**

For the Chinese Treebank I performed experiments evaluating the impact of using the *Reparse* method and also the variation with cross-training on the function labeling task.

The results obtained are somewhat contradictory: there is an improvement in performance using both on the development set (articles 1-25), but on the test set (articles 271-300) the basic method shows practically no improvement whereas cross-training actually leads to results worse than for the baseline.

Table 5.21 shows the results for the development set which are consistent with the findings so far: the reparsing method outperforms the baseline by 0.18%. Additionally, adding cross-training produces a further increase in the f-score of 0.3%.

However, as can be seen in Table 5.22, for the test set the predictions are not borne out: with cross-training I actually obtain a lower score than the baseline ($-0.32\%$); without cross-training the score is only marginally better than the baseline ($+0.01\%$).

I performed an approximate randomization test for both the development set and the set, testing the baseline against the reparsing method with cross-training. For the development set I obtained a $p$-value of 0.13; for the test set the $p$-value was 0.08 – thus neither the improvement for the development set nor the decrease in f-score for the test set are statistically significant.

It would be interesting to repeat the experiments for Chinese using larger data sets. There are two reasons to want to do that. First, testing on a larger test set would offer a higher confidence in the significance of the observed performance scores. Second, I suspect that one reason that my approach did not show consistent improvement

| | Precision | Recall | F-score |
|---|---|---|---|
| Baseline | 91.46 | 90.13 | 90.79 |
| Reparse | 91.39 | 90.23 | 90.80 |
| Reparse + x-train | 91.53 | 89.43 | 90.47 |

Table 5.22: Function labeling evaluation for the CTB on the parser output for the test set

across both the development set and the test set might be related to the relatively small amount of training material used, for both training the parser and the function labeling model. Thus parse quality is rather low, and since we only exploit correctly parsed nodes in extracting training instances for labeling, the amount of training data available decreases even further. I also suspect that parse quality for Chinese may be lower than for English even while holding training set size constant, reflecting the smaller amount of work which has gone into research on parsing Chinese.

Testing those conjectures remains an area for future investigation. It remains to be seen whether using my approach with training sets comparable in size with the one we used for English would more show more consistent benefits for Chinese.

Following up on the research described is this section I would like to better understand what factors influence the effect of the proposed training methods on function labeling performance. It should also be possible to apply the findings reported in this section to other tasks where training examples are typically extracted from perfect trees whereas the test data is produced automatically and contains errors. Training on parser output instead could be beneficial in those situations.

## 5.4 Summary

In this chapter I showed how to reliably learn function labels from treebanks for several languages using classification techniques from machine-learning. In the machine-learning setting various interacting parse-tree features can be used to predict the correct function label. This is more robust and reliable than trying to use categorical rules to map configurations to grammatical functions in the LFG annotation algorithm. Au-

tomatically adding function labels to parse trees, and then using them in a simplified annotation algorithm makes the DCU LFG parsing architecture easier to adapt to new languages and treebanks.

I have shown how to successfully learn function labels from the Spanish Cast3LB treebank and how the use of this methods substantially boosts the f-structure parsing scores for this language. I have also shown how to improve the performance of the function labeler on English Penn treebank data by extracting training material from reparsed sentences rather than from original treebank trees. My approach improves the similarity of the training material to the test instances as measured by instance overlap and mean Hamming distance. I have consistently found statistically significant improvements on the English Penn Treebank data, and a more mixed picture for the Chinese Penn Treebank sentences.

# Chapter 6

# Learning Morphology and Lemmatization

## 6.1 Introduction

In a lexicalized grammatical formalism such as LFG a large amount of syntactically relevant information comes from lexical entries. It is, therefore, important to be able to perform morphological analysis in an accurate and robust way for morphologically rich languages. It would also be desirable to treat morphological analysis in the same way as other aspects of building LFG representations: i.e. learn as much as possible from treebank data, and minimize the amount of language-dependent manual specification. In this chapter I first present existing work on supervised learning of morphological structure. Then I present a novel method of learning to lemmatize running text, and follow on with the description of **Morfette** - a probabilistic model of morphological tagging combined with lemmatization.

### 6.1.1 Main results obtained

This chapter shows how lemmatization can be seen as a classification where the class label is the representation of the operations needed to map a wordform to the corresponding lemma. Two such representations (or edit scripts) are proposed: REVERSE EDIT LIST which work best for the most common case of suffixation, and EDIT TREE which is more general and also gives good class labels for prefixation. I show that the

classifier approach to lemmatization can be used to lemmatize corpus text, i.e. wordforms in context, as well as pairs of wordform and morphological tag. In both scenarios the approach is competitive with state-of-the-art alternative methods.

Furthermore I propose a factored joint model – **Morfette** – for performing morphological tagging and lemmatization. This model can be used to assign sequences of lemma - morphological tag pairs to sentences. The model is trained on annotated corpus data; however it also makes it easy to exploit alternative sources of annotated data such as morphological dictionaries. The complete system shows good performance on a number of datasets for inflectional languages.

Part of the research presented in this chapter has been previously published in (Chrupała, 2006) and (Chrupała et al., 2008).

## 6.2 Previous Work

In this section I summarize existing work on supervised learning of morphological structure. I will not discuss unsupervised learning or non-data-driven finite-state approaches, since the literature in those fields is vast and is not directly relevant to the focus of this chapter.

### 6.2.1 Inductive Logic Programming

Early work on supervised learning of morphological structure was done within the Inductive Logic Programming (ILP) framework. ILP is a machine-learning methodology for inducing general rules from examples and background knowledge (Muggleton, 1991; Lavrač and Džeroski, 1994). First-order logic programming is used as a uniform representation of the background knowledge, examples and hypotheses. Training examples are typically represented as ground facts of the target predicate (i.e. the relation to induce). Background knowledge is also typically represented as ground terms.

Two ILP systems capable of learning first-order decision lists have been used to learn morphology: initial work made use of FOILD (Mooney and Califf, 1995) to learn past tense of English verbs; subsequently the more efficient ILP system CLOG (Manandhar et al., 1998) was shown to scale to large datasets and to accurately learn complex

inflectional morphology. Both systems use a hill-climbing strategy to find a locally best solution.

Mooney and Califf (1995) present the FOIDL system which extends the ILP paradigm with three features which make it suitable for NLP tasks such as learning morphology:

- Background knowledge can be expressed intensionally, i.e. as logic rules rather than extensionally as a collections of ground terms (facts).
- Negative examples are implicit (using the *output completeness* assumption)
- The target predicate that is learned can use the extralogical cut operator, i.e. the clauses in the definition learned are ordered, and thus correspond to first-order decision lists.

In order to learn the rules for past-tense formation for English verbs they provide the algorithm with the background predicate `split/3` which non-deterministically splits a non-empty list into a prefix and a suffix. It is defined in Prolog as follows:

```
split([X,Y | Z], [X], [Y | Z]).
split([X | Y],[X | W],Z) :- split(Y,W,Z).
```

Since FOILD target predicates can use the cut operator it can learn compact ordered decision lists where the initial clauses capture the most specific cases (exceptions) whereas clauses further down the list deal with increasingly general rules. For example the following target predicate can be learned:

```
past(A,B) :- split(A,C,[e,e,p]), split(B,C,[e,p,t]),!.
past(A,B) :- split(A,C,[y]), split(B,C,[i,e,d]),!.
past(A,B) :- split(A,C,[e]), split(B,C,[d]),!.
past(A,B) :- split(B,A,[e,d]).
```

This encodes the following decision list:

- If the word ends in *eep*, replace *eep* with *ept* (e.g. *sleep* → *slept*)
- If the word ends in *y*, replace *y* with *ied* (e.g. *try* → *tried*)
- If the word ends in *e*, replace *e* with *ed* (e.g. *bake* → *baked*)

- Otherwise, append *ed* (e.g. *cook* → *cooked*)

The authors show that FOILD learns rules for predicting the past test of English verbs with fewer examples and obtains better accuracy than previous ILP systems, and other approaches such as neural networks or decision-tree forests (Ling, 1994).

Subsequently FOILD was also used to learn the synthesis and analysis of Slovene nouns (Džeroski and Erjavec, 1997).

As discussed by Manandhar et al. (1998) FOILD has efficiency issues which make it difficult to scale to datasets significantly larger than the English past-tense task. The work on Slovene nominal morphology (Džeroski and Erjavec, 1997) was hampered by the fact that it could not be trained on sufficiently large datasets. The CLOG ILP system described by Manandhar et al. (1998) remedies those efficiency issues and demonstrates that the ILP approach can be extended to deal with realistic NLP tasks such as supervised learning of complex morphology. Manandhar et al. (1998) perform experiments on learning morphology for English, Romanian, Czech, Slovene and Estonian. The datasets they use come from the MULTEXT-EAST corpus (Erjavec, 2004), which consists of the text of George Orwell's novel *1984* in English and its translation into several Central and East-European languages. The text is tokenized, and the tokens are annotated with lemma and a *morphosyntactic description tag* (MSD). Those tags encode the part of speech and language-specific morphosyntactic features of the word. The tags in the corpus text are manually disambiguated. The resource comes with morphological lexicons which include all the word forms in the corpora, with all their possible analyses.

For the experiments the authors used word-lemma-MSD triples for wordforms in the corpus, in their non-disambiguated form. Only nouns and adjectives were used. From each triple two examples were created, one for morphological analysis and one for synthesis. Some features, irrelevant for morphological behavior, such as proper/common noun distinction were collapsed to a unspecified value. They used the background knowledge predicate `mate(W1,W2,P1,P2,S1,S2)` which is true if `P1` and `S1` are the prefix and suffix of `W1` and analogously with `P2`, `S2` and `W2`. This is sufficient to handle concatenative morphology such as in the languages the paper deals with.

The first three parts of *1984* were used for training and the appendix for testing.

| | Analysis | Synthesis |
|---|---|---|
| English | 96.05 | 98.02 |
| Romanian | 92.56 | 94.66 |
| Czech | 97.08 | 97.34 |
| Slovene | 96.95 | 91.56 |
| Estonian | 97.21 | 83.64 |

Table 6.1: Morphological synthesis and analysis performance in (Manandhar et al., 1998)

There were 288 MSDs between the five languages and thus the system had to learn 288 programs for synthesis and analysis each. The accuracies reported by the authors for Clog trained on the full training set are shown in Table 6.1.

Erjavec and Džeroski (2004) present a system for lemmatizing Slovene open-class words. Unlike in (Manandhar et al., 1998) the system is able to lemmatize raw word forms; this is achieved by decomposing the task into two stages: first raw text is MSD-tagged using a part-of-speech tagger, then the surface wordform-MSD tuples are lemmatized. The tagger used is the trigram TnT tagger described in (Brants, 2000). It is trained on 100,000 word corpus of MSD-tagged Slovene from Multext-EAST and backed up by a lexicon and various heuristics.

The Clog system is used to learn first order decision lists to predict word lemma given the wordform and the MSD tag determined by the MSD tagging module. Lemmatization is only learned for nouns, adjectives, and verbs. Lemmas for closed-class words, such as pronouns or auxiliary verbs, are retrieved from a lexicon, and for other open-class words such as adverbs, lemmas are always equal to wordforms. For training the lemmatizer the Multext-EAST lexicon of approximately 15,000 lemmas and the corresponding inflected forms was used.

The system was evaluated on the IJS-ELAN Slovene corpus (Erjavec, 2002). Since the TnT tagger trained on Multext-EAST performed poorly on the test corpus the authors tried to improve its accuracy by adding to the training set 1% of the test corpus sentences. They also used a lexicon covering 97% of the test corpus tokens and certain other heuristics to correct systematic tagging errors.

The improved MSD tagger achieves an accuracy of 92.9% on all tokens in the test set, and 90.0% over nouns, adjectives, and verbs only. The authors tested the performance of the whole system with lemmatization in the following way: they extracted a test lexicon from the tagged corpus consisting of words tagged as nouns, adjectives or verbs, and which have not been assigned a lemma during tagging based on lexicon lookup. From this they filtered out wordforms which contain non-alphanumeric characters (except hyphen), are less than four characters long, and appear only once in the corpus, as well as English words. The resulting lexicon consists of 763 distinct pairs of surface wordform and automatically assigned MSD. On this test set 92% of lemmas were correctly assigned; most errors were due to MSD-tagging mistakes.

### 6.2.2 Memory-based learning

An approach to morphological analysis of Dutch wordforms based on the nearest-neighbor classifier is described in van den Bosch and Daelemans (1999). It deals with both inflectional and derivational morphology.

For each position in a wordform the system predicts a label which encodes whether there is a morpheme boundary at that point, its part of speech, and what spelling changes were present. For example for the wordform *abnormaliteiten* (abnormalities) the first character receives the label **A+Da**, which means the morpheme beginning at that position is an adjective, and the spelling change is the deletion of the letter *a* (i.e. the original adjective morpheme is *abnormaal* (abnormal)). The character *i* which starts the morpheme *iteit* receives the label **N_A***, which encodes the fact that the morpheme attaches to the right of an adjective and produces a noun. Finally the character *e* at the start of the morpheme *en* receives the label **m**, which stands for plural inflection.

All the other characters receive the **0** label, meaning that no morphological boundary is present at that position. The wordforms for training and testing were taken from the CELEX (Baayen et al., 1993) lexical database. The features used were the character at the current position plus the preceding and following five characters. The overlap distance metric was used with features weighted by information gain. The authors performed a 10-fold cross-validation. They report the following generalization

accuracies: 95.88% instances (i.e. positions in wordforms) are classified correctly. This translates into 64.63% correctly analyzed words.

### 6.2.3 Analogical learning

Stroppa and Yvon (2005) present yet another take on supervised learning of morphology: analogical learning. Supervised learning using analogical reasoning is a type of instance-based learning model. Each training instance is a vector of $m$ features. Given the set of training instances $S = \{X_1, ..., X_n\}$ the task is to predict the missing features of new instances: thus this setting is more general than the classification task described in Section 3.2. The inference proceeds as follows: training instances are stored and no abstraction is performed. Given a new instance $X$, analogical proportions involving $X$ are identified: the features of objects involved in these relations are used to infer the missing features of the instance $X$. An analogical proportion is a relation involving four objects, and denoted as $A : B :: C : D$, i.e. $A$ is to $B$ as $C$ is to $D$. Let $I(X)$ be the set of known features of $X$ (projection to the input space) and $O(X)$ for the set of unknown features of $X$ (projection to the output space). Then the inference process can be formalized as:

1. Construct the set $T(X) \in S^3$ such that
   $T(X) = \{(A, B, C) \in S^3 \mid I(A) : I(B) :: I(C) : I(X)\}$
2. For each $(A, B, C) \in T(X)$ compute hypotheses $\widehat{O(X)}$ by solving the analogical equation: $\widehat{O(X)} = O(A) : O(B) :: O(C) :?$

Various methods are used to optimize search in $S^3$ to make it tractable.

The analogical learning model requires the availability of a method to compute analogical proportions on feature vectors. Stroppa and Yvon (2005) present a general definition of analogical proportion for semigroups and two concrete instantiations: for words over finite alphabets and for labeled trees. An element $u$ of the semigroup $(U, \oplus)$ where $\oplus$ is an associative composition operator on $U$, can be factorized into factors $u_1...u_n$, such that each factor $u_i \in U$ and $u_1 \oplus ... \oplus u_n = u$. Then $(x, y, z, t) \in (U, \oplus)$ form an analogical proportion $x : y :: z : t$ if and only if there exists some factorizations of $x_1 \oplus ... \oplus x_d = x$ (and similar for $y, z$ and $t$) such that $\forall i, (y_i, z_i) \in \{(x_i, t_i), (t_i, x_i)\}$.

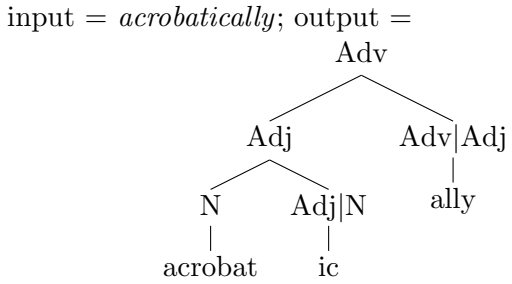



Figure 6.1: Instance for task 2 in Stroppa and Yvon (2005)

This definition can be instantiated to analogical proportions on strings over finite alphabets, where $U$ becomes the alphabet and $\oplus$ the concatenation operator. Yvon (2003) describes an efficient solver for analogical equations on strings based on finite-state transducers.

Similarly for labeled trees, analogical proportions can be defined if a binary associative operator on trees is defined such as substitution. In order to solve analogical proportions on trees Stroppa and Yvon (2005) propose to linearize trees into parenthesized strings and use the solver for strings.

Stroppa and Yvon (2005) describe an application of analogical learning to two morphological tasks. The first task is to predict vectors of features associated with isolated wordforms. Each vector consists of the lemma, part of speech and various part-of-speech-dependent morphological features such number, gender, case, tense, mood etc. A training example for English would be: input=*replying*, output={*reply,V-pp--*}.

The second task is similar to that undertaken in (van den Bosch and Daelemans, 1999) above, i.e. morphological segmentation. An example input-output pair for English for this task is shown in Figure 6.1.

The German, English and Dutch CELEX database was used for task 1. For task 2 only the English data was used. For each experiment they performed 10 runs and tested on 1,000 randomly picked instances. Generalization performance was measured as follows: per instance precision was computed as the proportion of correct hypothesis. Per instance recall was the proportion of correct solutions that were predicted by the system. Those two scores were averaged over over the test set, and averaged over the

| English | | |
|---|---|---|
| | Recall | Precision |
| Nouns | 75.26 | 95.37 |
| Verbs | 94.79 | 97.37 |
| Adjectives | 27.89 | 87.67 |
| Dutch | | |
| Nouns | 54.59 | 74.75 |
| Verbs | 93.26 | 94.36 |
| Adjectives | 90.02 | 95.33 |
| German | | |
| Nouns | 77.32 | 81.70 |
| Verbs | 90.50 | 90.63 |
| Adjectives | 99.01 | 99.15 |

Table 6.2: Results for task 1 in Stroppa and Yvon (2005)

| | Recall | Precision |
|---|---|---|
| Morphologically complex | 46.71 | 70.92 |
| Other | 17.00 | 46.86 |

Table 6.3: Results for task 2 in Stroppa and Yvon (2005)

10 runs. The results are summarized in Tables 6.2 and 6.3.

The authors note that high generalization performance is manifest for data with rich inflectional paradigms such as all German data. On the other hand for categories such as English adjectives the results are poor. They attribute this effect to the fact that in order to make search tractable instances are divided into bins based on inflectional categories – the performance could be improved for English adjectives by using bins based on derivational rather than inflectional families.

The methods for supervised learning of morphology described above all have in common the fact that for a given word form they are able to generate the corresponding lemma (or root or stem) for unseen words.

Another common method of performing data-driven morphological analysis does not have this property. In this approach morphological analysis is treated as a sequence labeling task, i.e. a generalization of part of speech tagging. The labels encode morphosyntactic features. A lexicon which maps wordform-label pairs to lemmas is then

used to perform lemmatization, which has the disadvantage that words not present in the lexicon cannot be lemmatized. This problem is solved by (Erjavec and Džeroski, 2004) by learning morphological analysis in two steps, first learning a morphological labeling model from running corpus text, and then learning a lemmatization model from a full form lexicon. The output of the first model is used by the second model.

In the following section I briefly review research on morphological tagging and on disambiguation of morphological analyses; this is relevant to Section 6.4 which presents a novel method of combining data driven morphological analysis with lemmatization, different to the approach used by (Erjavec and Džeroski, 2004).

### 6.2.4 Morphological tagging and disambiguation

Morphological tagging is a data-driven approach to morphological analysis which treats the task in terms of sequence labeling: i.e. it is modeled on the POS tagging framework. Morphological information is encoded in a tag, (sometimes called a morphosyntactic description, MSD) associated with the word form, and the system learns to predict those sequences of MSDs from annotated data. The main challenge here is dealing with morphologically rich languages where the number of unique tags is in the hundreds or thousands. This creates scaling issues for many machine learning approaches, and data-sparseness problems for those that do scale.

In order to alleviate those issues it is common to restrict MSD tagging to disambiguating candidate analyses proposed by a rule-based morphological analyzer, or a large morphological lexicon. It has also proved useful to predict the features encoded in the tags separately and combine those factored decisions to choose the full MSD tag.

Research using such approaches has been carried out initially mostly for East European languages. Hajič and Hladká (1998) describe a disambiguating tagger for Czech which uses an efficient exponential probabilistic model. Hajič (2000) adapts this model to other Easter European languages and investigates the issue of the relative usefulness of morphological dictionaries versus annotated corpora (cf. Section 6.4.5).

Tufiş (1999); Tufiş and Dragomirescu (2004); Ceauşu (2006) propose the approach of tiered tagging and apply it to Romanian. The idea is to reduce the original rich tagset to a smaller one, train the tagger on the reduced tagset, and recover the original

tags in a second-stage postprocessing step, which involves either application of hand written rules and dictionary lookup, or in the most recent paper, a second-stage tagger which uses the predicted reduced tags as input to recover the original ones.

More recently the tag disambiguation approach has been applied to Korean (Han and Palmer, 2004), Arabic (Habash and Rambow, 2005), and Turkish (Hakkani-Tür et al., 2002; Yuret and Türe, 2006). Due to the morphological complexity of those languages, those systems use an online morphological analyzer rather than a precompiled dictionary as the source of analysis candidates.

There are varying ways of dealing with lemmatization associated with these approaches: either ignore it altogether (Hajič and Hladká, 1998), rely on the morphological analyzer or dictionary to provide it (this leaves the problem of unknown words) or, as discussed in the previous section, treat it in a second stage (Erjavec and Džeroski, 2004).

In Section 6.3 I first present a method of learning lemmatization using only running text annotated with lemmas. This model learns some morphosyntactic features implicitly in order to predict lemmas, although such features are not explicitly present in the training data and are not output by the model. This is a useful mode of operation when disambiguated morphosyntactic labels are not available and it serves as a proof of concept for learning lemmatization in the classification setting.

In Section 6.4 I then show how to combine this lemmatization method with morphological tagging in an integrated model which predicts probability distributions over sequences of morphological tag-lemma pairs. This can be seen as an alternative, arguably simpler and more general approach to the two step method proposed by Erjavec and Džeroski (2004) for Slovene.

## 6.3 Simple Data-Driven Context-Sensitive Lemmatization

### 6.3.1 Lemmatization as a classification task

Many successful machine-learning methods require that the task to be performed be cast as classification. The training data should consist of a collection of examples with assigned class labels. The algorithms learn to assign those labels to new examples. Here

I show how lemmatization can be easily adapted to the classification setting, given some reasonable assumptions about the data.

It is not immediately obvious what the class labels should be for the task of lemmatization. In principle, lemma classes could be specified manually, on the basis of analysis of inflectional or derivational paradigms for a given language. This works but is labor-intensive.

The approach I propose is to derive the classes automatically from training data. Instead of inspecting data to identify and specify paradigms we try to automatically discover recurring patterns in the mappings form word forms to lemmas.

I present a very simple class-inference mechanism based on the idea of the edit script between two strings . An edit script specifies what transformations should be applied to the input string in order to obtain the output string. There are many possible string operations and many ways of specifying them in an edit script. Here I will concentrate on one simple instantiation of this idea: an EDIT-LIST (Myers, 1986; Aho et al., 1976; Hirschberg, 1977). An EDIT-LIST of sequences $w$ and $w'$ is a list of instructions (insertions and deletions) which, when applied to sequence $w$, transform it into sequence $w'$. An instruction specifies whether an insertion or a deletion should be performed, at which position in sequence $w$, and which element is to be inserted or deleted. As an example consider the strings $w = pidieron$ and $w' = pedir$[1]. An EDIT-LIST which transforms $w$ into $w'$ is $(\langle \mathrm{D}, i, 2\rangle, \langle \mathrm{I}, e, 3\rangle, \langle \mathrm{D}, e, 5\rangle, \langle \mathrm{D}, o, 7\rangle, \langle \mathrm{D}, n, 8\rangle)$. This is interpreted as

- delete character $i$ at position 2
- insert character $e$ before position 3
- delete character $e$ at position 5
- delete character $o$ at position 7
- delete character $n$ at position 8

I use the EDIT-LIST between word forms and their lemmas as class labels.

[1] *pidieron* is the 3rd person plural preterite form of the verb *pedir*, ASK in Spanish.

One nuance is that in the majority of languages inflectional morphology is mostly suffixal, i.e. it affects the endings of words, or occasionally material in word roots, rather than the beginning.[2] This means that EDIT-LISTs will work better as classes if we index characters starting at the end of the string rather than at the beginning, or equivalently if we compute the EDIT-LIST on reversed strings. For example if we compute *editlist*(*repitieron*, *repetir*)[3] we will not get the same EDIT-LIST as for the example above (as all indices will be incremented by 2). However on reversed strings, *editlist*(*noreidip*, *ridep*) and *editlist*(*noreitiper*, *riteper*) give the same result $\{\langle \mathrm{D}, n, 1\rangle, \langle \mathrm{D}, o, 2\rangle, \langle \mathrm{D}, e, 4\rangle, \langle \mathrm{D}, i, 7\rangle, \langle \mathrm{I}, e, 8\rangle\}$. This accords with the linguistic notion that the strings *pedir* and *repetir* are forms of Spanish verbs which occupy the same position in the verb inflection paradigm of the same conjugation class. If our assumptions about inflectional morphology hold, i.e. if it is predominantly suffixal, such agreement should happen frequently. I will refer to this adjusted version of the edit list as REVERSE-EDIT-LIST.

### 6.3.2 Experiments

I have performed a series of experiments on a range of languages and data sets to evaluate how this idea works in practice.

#### Data

I have used lemma-annotated corpora in eight languages.

- Spanish, Cast3LB Civit and Martí (2004)
- Catalan, Cat3LB Civit et al. (2004)
- Portuguese, Bosque 7.3, Afonso et al. (2002)
- French, Paris-7 Treebank, Abeillé et al. (2003)
- Polish, Polish Frequency Corpus, Section B - Press, Bień and Woliński (2003)
- Dutch, Alpino Treebank, van der Beek et al. (2002)

[2] Celtic languages such as Irish or Welsh are a well-known exception.

[3] *repitieron* is the 3rd person plural preterite form of the verb *repetir*, REPEAT in Spanish.

| Feature notation | Description |
|---|---|
| $f_{0_n},\ n = \|f_0\| - 12 \cdots \|f_0\| - 1$ | The last 12 characters of the word form |
| $f_0$ | The target token word form (treated atomically) |
| $f_n,\ n \in \{-3, -2, -1, 1, 2, 3\}$ | Word forms of preceding and following 3 tokens |

Table 6.4: Feature notation and description for lemmatization

- German, Tiger Treebank, Brants et al. (2002)
- Japanese, Kyoto Text Corpus, Kurohashi and Nagao (2003)

For each corpus I took 10,000 tokens as the test test, another 10,000 tokens as development set, and 70,000 as training set.

In the Japanese corpus the word forms appear in Kanji and the lemmas in Hiragana. Since the method needs data written in the same script, and preferably an alphabetic one, I convert both Kanji and Hiragana to Romaji using the Kakasi software package.[4] I have not evaluated the accuracy of the conversion so there may be some noise in our Japanese results.

### Methodology

For each language I use the same set of features, the presented in Table 6.3.2. Example features extracted from a Spanish sentence are shown in Table 6.3.2.

In the experiments I use the LIBSVM implementation (Chang and Lin, 2001) of Support Vector Machines (Boser et al., 1992; Vapnik, 1998), which implements the one-against-one strategy for non-binary (multi-class) classification. I binarize the features described above for use with the SVM: i.e. each original feature-value combination is mapped to a new binary feature.

I use the Radial Basis Function kernel. The parameters $C$ (32768) and $\gamma$ ($3.05 \times 10^{-5}$) were chosen by cross-validation on the Spanish development set. Because I did not repeat feature selection and parameter tuning separately for each language, my results may underestimate the potential performance of our method for languages other than

[4] Available for download at http://kakasi.namazu.org/. I would like to thank Masanori Oya for pointing out Kakasi to me and for help with the conversion.

| Left context | | | Focus | Focus chars | Right context | | | Class |
|---|---|---|---|---|---|---|---|---|
| $f_{-3}$ | $f_{-2}$ | $f_{-1}$ | $f_0$ | $f_{0_{\lvert f_0\rvert-12}} \cdots f_{0_{\lvert f_0\rvert-1}}$ | $f_1$ | $f_2$ | $f_3$ | |
| que | el | presidente | recibiera | `---recibiera` | cientos | de | miles | $(Da0, De2)$ |
| el | presidente | recibiera | cientos | `-----cientos` | de | miles | de | $(Ds0)$ |
| presidente | recibiera | cientos | de | `---------de` | miles | de | dólares | $\emptyset$ |
| recibiera | cientos | de | miles | `-------miles` | de | dólares | de | $(Ds0, De1)$ |
| cientos | de | miles | de | `---------de` | dólares | de | regalo | $\emptyset$ |
| de | miles | de | dólares | `-----dólares` | de | regalo | de | $(Ds0, De1)$ |

Table 6.5: Example features for lemmatization extracted from a Spanish sentence

| | Baseline Acc. | Accuracy | Precision | Recall | F-score |
|---|---|---|---|---|---|
| Catalan | 66.33 | 97.27 | 95.91 | 93.40 | 94.64 |
| German | 52.64 | 95.11 | 94.61 | 92.03 | 93.31 |
| Polish | 48.61 | 95.06 | 93.75 | 91.96 | 92.84 |
| Spanish | 69.50 | 96.44 | 92.32 | 92.65 | 92.48 |
| Japanese | 88.42 | 98.36 | 94.05 | 88.77 | 91.33 |
| Portuguese | 71.74 | 96.38 | 91.85 | 90.58 | 91.21 |
| French | 61.88 | 94.36 | 92.16 | 87.93 | 89.99 |
| Dutch | 78.80 | 94.15 | 85.38 | 79.62 | 82.40 |

Table 6.6: Lemmatization evaluation for eight languages

Spanish.

When calculating the REVERSE-EDIT-LIST for word form - lemma pairs, both strings were lowercased and embedded quotes (occasionally found in German compound words) were removed. These simplifications reduce the number of classes and make the learning task easier. Also, lemma capitalization is ignored for evaluation.

### 6.3.3 Evaluation results and error analysis

Table 6.6 presents the results of evaluation for all the languages on the test sets. The most straightforward performance metric is token accuracy: i.e. what proportion of tokens were correctly lemmatized (shown in the second column). Depending on the data set, high accuracy can be achieved by simply returning the word form (the baseline method). E.g. for Japanese, where the only open-class words which inflect are verbs, the baseline method give 88.42% accuracy on the test set. Baseline accuracies are shown in the first column. The second column shows accuracies: for all the data sets they are above 94%, with the highest score for Japanese at 98.36%.

To give a more informative indication of the performance I also calculate precision, recall, and the harmonic mean of those two, the f-score. For those metrics I consider the empty REVERSE-EDIT-LIST, i.e. when the lemma is equal to the word form, as the null class. The number of correct lemmas, excluding the nulls, are the true positives. Recall is then calculated by dividing the number of true positives by the number of non-null lemmas in the gold standard, whereas precision is the number of true positives

| | Baseline Acc. | Accuracy | Precision | Recall | F-score |
|---|---|---|---|---|---|
| Polish | 26.31 | 80.29 | 81.73 | 77.53 | 79.58 |
| Spanish | 58.75 | 86.35 | 77.97 | 79.39 | 78.67 |
| Portuguese | 58.58 | 85.17 | 76.35 | 70.32 | 73.21 |
| Catalan | 60.16 | 82.99 | 76.11 | 66.05 | 70.72 |
| German | 55.95 | 78.88 | 72.02 | 62.80 | 67.09 |
| Japanese | 78.96 | 89.54 | 74.62 | 59.88 | 66.44 |
| French | 55.80 | 76.83 | 71.42 | 55.71 | 62.60 |
| Dutch | 66.42 | 72.40 | 46.82 | 31.33 | 37.54 |

Table 6.7: Lemmatization evaluation for eight languages – unseen word forms only

divided by the number of non-nulls among the predicted lemmas.

Except for one case, the f-scores cluster between 90% and 95%, even though baseline accuracies range from under 50% to almost 90%. Thus even though the languages represent varying degrees of inflectional richness, this has limited impact on the performance of the REVERSE-EDIT-LIST-based lemmatization method.

There is one outlier, however: for Dutch the f-score is over 7% worse than the next best result. I suspect that this is due to the fact that for this dataset the assumption of predominantly suffixal inflection does not hold. It turns out that there are many tokens in the Dutch corpus where mapping the wordform to lemma involves changes to the beginning of the string, often involving moving an initial part of the wordform to the end. This happens in the case of separable prefix verbs such as: *lesgegeven* → *geef_les* (teach class) or *meelopen* → *loop_mee* (run with). In those two examples the lemma is the verb inflected for present first-person singular, with the separable particle following it. Even more problematic are cases where the separable prefix that appears at another point in the utterance is appended at the end of the lemma, e.g. the sequence of word forms *we trokken erop uit* (we went out) is lemmatized as follows: $\langle we, trek\_uit, erop, uit \rangle$. Another non-final transformation involves compound words, where the compounding morpheme *-s* is replaced by an underscore in the lemma: *verbrandingsmotor* → *verbranding_motor* (internal combustion engine). It is clear that for such transformations classifying examples by REVERSE-EDIT-LIST is not sufficient.

It is also evident, however, that for many datasets such cases are rare and the

| | | Base Acc. | Accuracy | Precision | Recall | F-score |
|---|---|---|---|---|---|---|
| Freeling | all | 69.50 | 95.05 | 92.78 | 88.13 | 90.39 |
| | unseen | 58.75 | 82.05 | 82.58 | 64.36 | 72.34 |
| REL+SVM | all | 69.50 | 96.44 | 92.32 | 92.65 | 92.48 |
| | unseen | 58.75 | 86.35 | 77.97 | 79.39 | 78.67 |

Table 6.8: Comparison of REVERSE-EDIT-LIST+SVM to Freeling on the lemmatization task for Spanish

simple method shows remarkable high performance. The problems with lemmatizing the Dutch corpus are only partly caused by the features of the language itself. Equally important are the choices made by corpus designers. This can be seen by comparing the results for Dutch to those on the closely related language German, which also has verbs with separable prefixes and the compounding morpheme *-s-*. However in the German Tiger treebank separable prefixes which appear elsewhere in the text are not attached to the verb lemma, and the morpheme *-s-* does not get replaced by an underscore in lemmas for compound nouns. For example the sentence *Konzernchefs lehnen den Milliardär als US-Präsidenten ab* (Company bosses reject a billionaire for US president) is lemmatized as ⟨*Konzernchef, lehnen, der, Milliardär, als, US-Präsident, ab*⟩ even though it contains the verb *ablehnen* with the separable prefix *ab*. It could be plausibly argued that in the common "pipeline" approach to language processing finding such non-local dependencies is best left to the syntactic level of analysis.

Table 6.7 shows the same statistics as Table 6.6 for the subset of word forms not seen in the training set. There is more variance in these results than for the all-tokens evaluation and the relative ranking of languages is also different. Understandably, for all test-sets there is a significant drop in the f-score for the unseen subset. There is a group of languages where the difference between the f-scores is below 20% (Polish, Spanish and Portuguese), another group (Catalan, German, Japanese and French) between 20% and 30%, and again the outlier datapoint of Dutch, where the difference is of 44%. It remains to be investigated to what degree these differences are a function of the morphological features of the languages in question and to what extent they reflect the nature of the particular datasets or treebanks used in this evaluation.

| | | Base Acc. | Accuracy | Precision | Recall | F-score |
|---|---|---|---|---|---|---|
| Freeling | all | 66.33 | 93.32 | 93.08 | 83.93 | 88.27 |
| | unseen | 60.16 | 77.16 | 86.31 | 46.13 | 60.13 |
| REL+SVM | all | 66.33 | 97.27 | 95.91 | 93.40 | 94.64 |
| | unseen | 60.16 | 82.99 | 76.11 | 66.05 | 70.72 |

Table 6.9: Comparison of REVERSE-EDIT-LIST+SVM to Freeling on the lemmatization task for Catalan

**Comparison to Freeling**

In order to determine how the REVERSE-EDIT-LIST-based machine-learning approach to lemmatization compares to more traditional methods, I compare the results of my method to the performance of a popular analyzer Freeling (version 1.2) (Carreras et al., 2004)[5]. Freeling performs a range of language-processing tasks (tokenization, morphological analysis, named-entity recognition, chunking etc.) for several languages. Below I compare the two systems on the lemmatization task on the Spanish and Catalan test sets. Lemmatization in Freeling is based on lexicon lookup combined with disambiguation based on the part of speech tag in cases where the same word form can correspond to different lemmas. The Spanish lexicon size is about 71,000 word forms. The Catalan lexicon contains around 46,000 word forms.

In the input to Freeling I keep the original tokenization and sentence splits present in the corpus data. In the Spanish and Catalan treebanks multi-word expressions, named-entities, dates and quantities are treated as single tokens – I also keep those tokens, and consequently deactivate named-entity, multi-word, date and quantity handling by Freeling.

Tables 6.8 (Spanish) and 6.9 (Catalan) show the results for Freeling and for the REVERSE-EDIT-LIST-based method using the SVM classifier. I report results on all tokens and also results on tokens not seen in our training set. For Spanish, my method outperforms Freeling by about +2% on all tokens, which corresponds to a relative error reduction of 28%. The +6.33% difference between the systems on the unseen subset of

[5]At the time these experiments were performed this was the latest version of Freeling. In Section 6.4.5 the current version, Freeling 2.0, is compared against the complete morphological analysis system **Morfette**, which uses the classifier-based lemmatization approach proposed here.

| | Different | EDIT-LIST better | $p$-value |
|---|---|---|---|
| Spanish | 581 | 360 | $4.414 \times 10^{-9}$ |
| Catalan | 705 | 550 | $2.2 \times 10^{-16}$ |

Table 6.10: Statistical significance test

tokens, gives a relative error reduction of 24%. For Catalan the differences are larger: +6.37% (50% error reduction) on all tokens and +10.58% (25% error reduction) on the unseen token subset. The poorer performance of Freeling on the Catalan data probably reflects its small lexicon size for that language.

Freeling's lemmatization is not data-driven, and does not use the training data. The sharp drop in performance it shows for the subset of tokens unseen in the training material is probably due to the fact that such tokens are relatively uncommon words, in many cases probably absent from Freeling's word form lexicon. In those cases, Freeling simply returns the word form as the lemma, whereas the machine-learning model generalizes to unseen data and in most cases outputs the correct answer.

To determine how statistically significant the difference between the systems' performance is, I ran a binomial test on the results for the all tokens comparison. I adopt a confidence level of 99% ($\alpha = 0.01$) for these tests.

For each token we check whether the methods give different answers – for these cases (i.e. the number of trials) we calculate in how many cases the second method is better than the first (i.e. the number of successes). I then perform the binomial test with the null hypothesis that the probability of success is chance ($= 0.5$) and the alternative hypothesis that the probability of success is greater than chance ($> 0.5$). The results are summarized in Table 6.10. For both languages the $p$-values are much below $\alpha$, and statistically significant at that confidence level.

### 6.3.4 Conclusion

The edit script approach to learning to lemmatize running text is appealingly simple and manages to combine good performace with a high degree of language independence. Other methods often rely on a large full-paradigm inflectional lexicon, either to perform

word form lookup, or as a training resource. To train the system presented here only a lemmatized corpus is needed. The system is context-sensitive: it incorporates features of context words surrounding the target word form to combine lemmatization with disambiguation.

Though certainly useful, lemmatization without accompanying morphological analysis is often insufficient. In order to perform lemmatization, the system already has to learn some implicit morphological classes. For example in order to decide whether the correct lemma class should map the Spanish word form *bases* to *base* (basis) or to *basar* (to base) the system has to decide whether the token is more likely to be a verb or a noun. In Section 6.4 I factor out the learning of morphological features and lemma classes and investigate how to best reintegrate them in a modular fashion.

As evident from the Dutch results, the REVERSE-EDIT-LIST approach is inadequate when the assumption of suffixal inflection does not hold. A partial solution to this issue is offered in Section 6.4.6.

## 6.4 Morfette – a combined probabilistic model for morphological tagging and lemmatization[6]

### 6.4.1 Introduction

In this section I describe and evaluate the **Morfette** system for data-driven morphological analysis. The approach follows the decomposition of the task of morphological analysis into two subtasks: the assignment of morphological features to the word form, and lemmatization.

Many data-driven approaches to morphology involve encoding morphological features as tags (MSDs), and use some sequence labeling method to assign MSD sequences to sentences. In the case of morphologically rich inflectional or agglutinative languages, the classification decision is often constrained by the use of an MSD lexicon, or a finite-state morphological analyzer: in such systems the data-driven component is limited to performing morphological disambiguation rather than morphological analysis

[6]The research presented in this section was done in collaboration with Georgiana Dinu, who helped develop the EDIT-TREE lemma induction scheme, proposed the prepruning criterion, prepared software for error analysis and did the error analysis for Romanian data.

itself (Hajič and Hladká, 1998; Hajič, 2000; Tufiş, 1999; Tufiş and Dragomirescu, 2004; Ceauşu, 2006; Han and Palmer, 2004; Habash and Rambow, 2005; Hakkani-Tür et al., 2002; Yuret and Türe, 2006).

In an MSD disambiguation setting, lemmatization is simple: either the lexicon or the morphological analyzer already returns the correct lemma corresponding to each of the candidate analyses. The problematic cases are unknown words: most systems are able to guess the MSD of an unknown word, but not the corresponding lemma.

As described in Section 6.2.1, Erjavec and Džeroski (2004) solve the problem of lemmatizing unknown words by using a two stage architecture, first sentences are assigned MSD sequences by a POS-tagger, and then an ILP system assigns lemmas to unknown wordform-MSD pairs.

In Section 6.3 I described an alternative approach to lemmatization. This method automatically induces lemma-classes: they correspond to the edit script between word forms and the corresponding lemmas. Then a standard classifier is used to "tag" words with their lemma-classes, from which the words' lemmas can be obtained by "executing" the edit script on the word forms. Thus in this approach lemmatization becomes just another instantiation of sequence labeling.

In this section I present a modular, data-driven model which performs both morphological tagging and lemmatization, i.e. it maps a sequence of word forms of length $n$ to the sequence of MSD - lemma pairs:

$$M : \mathcal{W}^n \rightarrow (\mathcal{M} \times \Lambda)^n \tag{6.1}$$

I use a generic, language-independent feature-set in the models and investigate how well such an approach generalizes to three morphologically rich languages.

In Section 6.4.2 I present the architecture of the model, the features used and the search algorithm. In Section 6.4.3 I present experimental evaluation results for three languages and corpora. Section 6.4.4 contains a brief error analysis. Section 6.4.5 experiments with integrating lexical resources in the system while Section 6.4.6 proposes an alternative version of edit script to use for lemma class induction. Finally Section 6.4.7 presents conclusions and ideas for further improvements in data-driven morphological analysis.

### 6.4.2 The Morfette system

#### Architecture

The **Morfette** system is composed of two learning modules, one for morphological tagging and one for lemmatization, and one decoding module which searches for the best sequence of pairs of morphological tags and lemmas for an input sequence of word forms. Both modules learn Maximum Entropy classifiers such as that described for POS tagging by Ratnaparkhi (1996). For the lemmatization model I use the method of inducing lemma classes described in Section 6.3. I do not, however, use the features or the SVM classifier, as that configuration turned out to be impractically slow in practice and to scale poorly. The primary reason for that is that a SVM is a binary classifier and to perform multiclass classification one has to decompose that task into a series of binary classifications and combine them using a method such as one-against-all. For the large number of classes involved in lemmatizing and MSD-tagging inflectional languages this is computationally quite expensive.

#### Features

In the **Morfette** architecture one can use arbitrary features of the focus word and the context sentence. I use a rather minimalistic and language-independent feature set in the experiments reported in Section 6.4.3. This has the advantage of being very general and using very little domain expertise but obviously for maximum performance it is desirable to extend and refine it using language and domain specific features. Initially this BASIC feature configuration was tested on three languages for which we had sufficient expertise to perform meaningful error analysis, i.e. Spanish, Polish and Romanian. The features are described in Table 6.11. Figure 6.2 shows the values of those features extracted from the following example sentence in Romanian:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Wordform | În | pereţii | **boxei** | erau | trei | orificii | (6.2) |
| Lemma | în | perete | **boxă** | fi | trei | orificiu | |
| MSD | Spsa | Ncmpry | **Ncfsoy** | Vmii3p | Mc-p-l | Ncfp-n | |
| Gloss | *In* | *the walls of the cubicle* | | *there were* | *three* | *orifices* | |

| Feature notation | Description |
|---|---|
| | MSD-tagging model |
| $f_0$ | Lowercased wordform of the focus token |
| $s_n(f_0),\ n = 1 \cdots 7$ | Suffixes of length $n$ |
| $p_n(f_0),\ n = 1 \cdots 5$ | Prefixes of length $n$ |
| $sp(F_0)$ | Spelling pattern of the (non-lowercased) wordform |
| $s_1(m_{-2}) \oplus s_1(m_{-1})$ | Concatenation of the first element of the two previous MSDs |
| $f_{-2}, f_{-1}$ | Lowercased wordform of two previous tokens |
| $m_{-2}, m_{-1}$ | (Predicted) MSD of two previous tokens |
| $l_{-2}, l_{-1}$ | (Predicted) Lemma of two previous tokens |
| $m_{train_1}$ | Set of MSDs seen in training data for wordform of next token |
| | Lemmatization model |
| $f_0$ | Lowercased wordform of the focus token |
| $s_n(f_0),\ n = 1 \cdots 7$ | Suffixes of length $n$ |
| $p_n(f_0),\ n = 1 \cdots 5$ | Prefixes of length $n$ |
| $m_0$ | (Predicted) MSD tag |
| $sp(F_0)$ | Spelling pattern[7] of the (non-lowercased) wordform |

Table 6.11: Feature notation and description for the BASIC configuration

In comparison with the features used in Section 6.3, the main difference is that here word suffixes of length 1 to 7 are represented explicitly whereas before only single characters were used. This is due to the fact that the SVM classifier when used with a kernel can automatically model feature conjunctions by implicitly mapping feature vectors to higher dimensional spaces whereas in the MaxEnt framework they have to be represented explicitly.

### Search

Maximum entropy models predict probability distributions over classes (i.e. MSD-tags or lemma-classes) for the current focus word form given its context as encoded in the features. That is for a focus word $w_i$ with the context $c \in \mathcal{C}$, for each possible MSD-tag $m \in \mathcal{M}$ the MSD-tagging model gives $p(m|c)$, and for each possible lemma-class $l \in \mathcal{L}$ the lemmatization model gives $p(l|c, m)$. The context includes the focus wordform as well as the preceding and following wordforms in the same sentence.

The algorithm is a beam search which maintains a list of $n$-best sequences of $(m, l) \in \mathcal{M} \times \mathcal{L}$ (MSD-tag - lemma) pairs up to the current position in the input word sequence.

| $f_0$ | $sp(F_0)$ | $p_1(f_0)$ | $p_2(f_0)$ | $p_3(f_0)$ | $p_4(f_0)$ | $p_5(f_0)$ | $s_1(f_0)$ | $s_2(f_0)$ | $s_3(f_0)$ | $s_4(f_0)$ | $s_5(f_0)$ | $s_6(f_0)$ | $s_7(f_0)$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| în | Xx | î | în | - | - | - | n | în | - | - | - | - | - |
| pereţii | x | p | pe | per | pere | pereţ | i | ii | ţii | eţii | reţii | ereţii | pereţii |
| boxei | x | b | bo | box | boxe | boxei | i | ei | xei | oxei | boxei | - | - |
| erau | x | e | er | era | erau | - | u | au | rau | erau | - | - | - |
| trei | x | t | tr | tre | trei | - | i | ei | rei | trei | - | - | - |
| orificii | x | o | or | ori | orif | orifi | i | ii | cii | icii | ficii | ificii | - |

| $f_{-2}$ | $l_{-2}$ | $t_{-2}$ | $f_{-1}$ | $l_{-1}$ | $m_{-1}$ | $s_1(m_{-2}) \oplus s_2(m_{-1})$ | $f_{+1}$ | $m_{train_{+1}}$ |
|---|---|---|---|---|---|---|---|---|
| - | - | - | - | - | - | | pereţii | { Ncmpry} |
| - | - | - | în | în | Spsa | S | boxei | {} |
| în | în | Spsa | pereţii | perete | Ncmpry | S+N | erau | {Vmii3p} |
| pereţii | perete | Ncmpry | boxei | boxă | Ncfsoy | N+N | trei | {Mc-p-l} |
| boxei | boxă | Ncfsoy | erau | fi | Vmii3p | N+V | - | - |
| erau | fi | Vmii3p | trei | trei | Mc-p-l | V+M | - | - |

Figure 6.2: Features extracted for the MSD-tagging model from an example Romanian phrase: *În pereţii boxei erau trei orificii.*

The conditional probability of a candidate sequence of words $w_0..w_i$ is given by

$$P(m_0..m_i, l_0..l_i|w_0..w_i) = p(l_i|c_i, m_i)p(m_i|c_i)P(m_0..m_{i-1}, l_0..l_{i-1}|w_0..w_{i-1}) \quad (6.3)$$

The search proceeds as follows: for focus word $w_i$ there are $n$ ($n$ being the beam size) highest probability sequences $((m_0, l_0)..(m_{i-1}..l_{i-1}))$. For each of those sequences we obtain a MSD-tag probability distribution from the MSD-tagging model. For efficiency reasons we pre-prune this set of tags: given the list of tag probabilities $(m_0, p_0)..(m_j, p_j)$ sorted in decreasing order, we keep all the tags $m_0..m_i$ where $p_i$ satisfies the condition:

$$p_i / \sum_{k=0}^{i} p_k < T,$$

where $T$ is a threshold parameter.

Each of the retained morpho-tags for word $w_i$ is added to each candidate sequence and for each of those combinations we obtain lemma-class probability distribution from the lemmatization model. The lemma-class set is pruned according to the same method as for MSD-tags. The probability of candidate sequences is updated according to Equation 6.3, the $n$ highest ranking candidate sequences for $w_0..w_1$ are retained and the algorithm proceeds to word $w_{i+1}$.

### 6.4.3 Evaluation

For evaluation I chose three morphologically rich languages for which we had expertise to perform error analysis. I have not tuned the features or parameters of our system to any particular dataset. At this stage the focus is not necessarily on improving on the best published results for a particular language; rather the objective is to see how well the system performs with a minimalistic feature set and no language-dependent engineering effort and identify the main source of mistakes for each language.

I use the following data sets:

- Romanian: MULTEXT-EAST corpus (Erjavec, 2004), approx. 13,500 tokens (chapters 1-3) as a test set, approx. 11,800 tokens (chapters 5 and 6) for development and 88,000 tokens (chapters 7-23) for training.

- Spanish: AnCora treebank (Martí et al., 2007), approx. 10,000 tokens each for test and development set, and approx. 168,000 tokens for the FULL training set, and approx. 70,000 for the SMALL training set.

- Polish: Korpus Słownika Frekwencyjnego (IPI PAN)[8], 10,000 tokens each for test and development sets, and approx. 219,000 for FULL training set, and approx. 70,000 for the SMALL training set.

The SMALL training set was used in order to be able to have similar training set sizes across the three languages. Additionally for Polish and Spanish the FULL set contains all the available data. No more training data is available in the Romanian corpus.

The Polish data set contains some tokens which have not been disambiguated: I filtered out all sentences containing such tokens.

For all the experiments reported in the following sections a beam size of 3 was used, with the prepruning threshold set to 0.3: validation on the development sets showed that those settings give good results for all the languages.

Table 6.12 shows the evaluation results for the SMALL training set for all three languages. Table 6.13 shows the results for Spanish and Polish, for which there is a larger training set available. Note that for seen and unseen tokens in Table 6.13, if we subtract the improvement in brackets from the scores, we do not get the score in Table 6.12: this is because in Table 6.13 the set of seen and unseen tokens is computed with reference to the FULL training set, while in Table 6.12 it is in reference to the SMALL training set.

More data is clearly beneficial: the scores improve substantially for both languages. Both the morphological tagging and lemmatization score for Polish is lower than for the other two languages: this is to be expected for a Slavic language with a rich inflection and high ambiguity. In the following Section I present a summary of the most common errors detected on the development set using the models trained with the BASIC configuration.

[8] Available at http://korpus.pl/index.php?page=download

| Unseen word ratio | |
|---|---|
| Spanish | 13.05 |
| Romanian | 8.77 |
| Polish | 20.39 |

| All words | | | |
|---|---|---|---|
| | MSD-tagging | Lemmatization | Joint |
| Spanish | 94.33 | 97.84 | 93.83 |
| Romanian | 96.83 | 97.78 | 96.08 |
| Polish | 81.87 | 93.29 | 81.19 |

| Seen words | | | |
|---|---|---|---|
| | MSD-tagging | Lemmatization | Joint |
| Spanish | 97.26 | 99.14 | 97.22 |
| Romanian | 97.81 | 99.21 | 97.77 |
| Polish | 86.96 | 97.48 | 86.81 |

| Unseen words | | | |
|---|---|---|---|
| | MSD-tagging | Lemmatization | Joint |
| Spanish | 74.79 | 89.20 | 71.26 |
| Romanian | 86.68 | 82.88 | 78.50 |
| Polish | 61.93 | 76.88 | 59.17 |

Table 6.12: Evaluation results with the BASIC model with SMALL training set for Spanish, Romanian and Polish

| Unseen word ratio | |
|---|---|
| Spanish | 8.77 |
| Polish | 12.92 |

| All words | | | |
|---|---|---|---|
| | MSD-tagging | Lemmatization | Joint |
| Spanish | 95.40 (+1.07) | 98.52 (+0.68) | 95.02 (+1.19) |
| Polish | 84.91 (+3.04) | 95.55 (+2.26) | 84.44 (+3.25) |

| Seen words | | | |
|---|---|---|---|
| | MSD-tagging | Lemmatization | Joint |
| Spanish | 97.29 (+0.77) | 99.22 (+0.48) | 97.25 (+0.92) |
| Polish | 87.74 (+2.85) | 97.69 (+1.93) | 87.60 (+3.09) |

| Unseen words | | | |
|---|---|---|---|
| | MSD-tagging | Lemmatization | Joint |
| Spanish | 75.71 (+4.22) | 91.22 (+2.74) | 71.84 (+3.99) |
| Polish | 65.87 (+4.33) | 81.11 (+4.49) | 63.16 (+4.33) |

Table 6.13: Evaluation results with a FULL training set for Spanish and Polish. Numbers in brackets indicate accuracy improvement over the same model trained on the SMALL training set

### 6.4.4 Error analysis

We performed detailed error analysis for morphological tagging and lemmatization for Spanish, Romanian and Polish. In this section I summarize the results of this analysis and suggest possible ways of dealing with some of the common errors our systems makes.

Errors in morphological tagging and lemmatization tend to co-occur: often an incorrectly assigned morphological category triggers lemmatization which is consistent with this category but incorrect given the gold MSD. I will therefore discuss the issues related to both morphological tags and lemma-class tags jointly.

**Named entities** A common source of errors in Spanish and Romanian is failure to detect proper names (the tagset used in Polish does not have a separate tag for proper nouns). This results in the assignment of the wrong morphological tags and sometimes also the wrong lemma-class. For example in Spanish certain person or place names, such as *Reyes* or *Chiapas* have the plural suffix but, unlike for common nouns, their correct lemma-class should not delete it. Poor performance in this area is to be expected as my focus here is on learning morphological structure and not on detecting and classifying named entities. The only feature designed to capture some characteristics of those is $sp(F_0)$, the spelling pattern feature, which is clearly very rudimentary. In order to deal with named entities properly a dedicated module would be probably the best solution.

**Suffix ambiguity** A problem for all the three languages is suffix ambiguity, i.e. certain word endings can be indicative of more than one morphological category. In Spanish and Romanian, nouns and adjectives are difficult to distinguish based only on word endings and are sometimes mistagged and mislemmatized. This tends to happen mostly in constructions with adjectives preceding nouns, e.g. Spanish *cruenta lucha* "bloody battle", which are rare and marked in comparison to adjectives post-modifying the noun.

In Romanian third person singular verbs in the *imperfect* tense have the same ending as nouns marked with a definite feminine article, and are also sometimes misclassified.[9]

[9]This affects only the written language as in speech those two forms differ in stress, which is not represented in the spelling.

**Syncretism** This is an especially frequent error type for Polish. Sometimes different grammatical cases of the same lexical item have the same form, e.g. feminine genitive singular noun forms and feminine genitive plural forms or masculine singular nominative and accusative.

There is sometimes genuine semantic ambiguity in the sentence but in many other cases, especially for number ambiguity, the correct morphological tag can be determined from context, but the system fails to do so. The determination of the right grammatical case is more difficult as it often involves non-local dependencies on the head verb or preposition and is unlikely to be solved completely by examining local context only.

**Ambiguous function words** Some high frequency function words are ambiguous: Spanish *que* (coordinating conjunction or relative pronoun), *se* (third person pronoun or impersonal pronoun); Romanian *a* (infinitive particle or a form of auxiliary *avea*, "have"), *lui* and *o* (article or pronoun); Polish *na* (locative or directional preposition). These distinctions are based on function rather than form and can be difficult to determine locally.

**Annotation problems** A nonnegligible number of errors in both morphological tagging and lemmatization are actually mistakes or inconsistencies in the training and test data. In the Polish dataset de-verbal nouns such as *działanie* are sometimes tagged as nouns and sometimes as "gerunds" (where the corresponding lemma is the verb infinitive). There seems to be no consistent pattern to which tag is used when. Some Spanish plurals are assigned incorrect lemmas in the corpus.

**Prefixal morphology** Even though in the languages we examined inflectional morphology is almost exclusively suffixal, Polish offers one isolated but important exception. The superlative form of adjectives is formed by attaching the prefix *naj-* to the (already inflected) comparative form. Thus the comparative of *wysoki*, "tall", is *wyższy*, and the superlative is *najwyższy*. Since lemma-classes are computed by the REVERSE-EDIT-LIST, this class induction method fails to generalize over word initial transformations. As a result, lemmas for superlatives are correct only in the case of very frequent words, and in general are not predicted correctly.

From the evaluation and error analysis performed for three languages I have found that some error categories occur in all three languages; others are language and corpus specific. I suspect that the error classes which mostly affect unknown words could be dealt with successfully by (i) providing more training data, (ii) exploiting additional resources in conjunction with annotated corpora, such as lexicons – I explore this option in Section 6.4.5

Other problems such as nominal/accusative syncretism or some ambiguous function words are more of a challenge, and although some improvement may be obtained by using more context and smarter features, it may be necessary to defer ambiguity resolution until a full syntactic structure is built.

Finally, the lemma-class induction mechanism is biased to dealing with suffixal morphology exclusively. In Section 6.4.6 I explore an alternative edit-script instantiation which is not as heavily biased to suffixal morphology as the REVERSE-EDIT-LIST used so far.

### 6.4.5 Integrating lexicons

We have seen that several of the common mistakes the system makes are due to unknown words in ambiguous contexts: given only the local context and features of the wordform sometimes there are more than one roughly equally plausible analysis for a certain token. The most obvious solution to this is to use more annotated data; however there are two issues here. First it is not always easy to obtain or produce large, high-quality morphologically annotated corpora. The second issue is the Zipfian nature of word frequency distribution in natural language, which means that even given a very large corpus many wordforms will be not appear or will appear only once.

The second most obvious solution is to try to leverage an alternative source of morphologically annotated data, namely morphological lexicons. In comparison to corpus data, morphological lexicons are a more impoverished source of information: they associate wordforms to the set of possible analyses, without taking context or relative frequency into account.

On the other hand it is easier to provide coverage of uncommon words by means of including them in a lexicon than by annotating very large corpora with their rich

contextual and frequency information. Hajič (2000) shows how for five languages the use of dictionaries is more helpful than more annotated corpus data in alleviating data sparseness. In the title of his paper he contrasts "data" vs "dictionaries", but obviously dictionaries are just another kind of annotated data that we can exploit in a supervised learning paradigm: those two kinds of data offer different trade-offs in terms of coverage vs. richness of information. In this section I investigate whether exploiting them both in an integrated fashion would let us take advantage of the strong points of both and improve overall morphological analysis results.

### Dictionary features

There are several options to exploit morphological dictionaries in a MSD tagging model. Perhaps the most common one is to treat the dictionary as a source of possible analyses and let the model simply choose one of them, or disambiguate. This assumes that the dictionary has very large coverage and is of high quality – also some provision needs to be made for the unavoidable cases of unknown words. In this approach the dictionary is a trusted, primary resource, whereas the annotated corpus provides frequency statistics which help disambiguate ambiguous wordforms. This is a well-known and well understood approach; here I use an alternative method: the dictionary information is incorporated into an overall MaxEnt model by means of dedicated features. Specifically, I take the BASIC model and modify it to incorporate dictionary information; the resulting model is called BASIC+DICT:

- MSD model:
  - The feature $m_{dict_0}$ is added: this is the set of MSD tags which the focus wordform occurs with in the dictionary.
  - The feature $m_{train_1}$ is replaced with $m_{dict_1}$: this is the set of MSD tags which the wordform of the following token occurs with in the dictionary. The new feature with MSDs extracted from the lexicon rather than from training tokens should more completely encode the ambiguity class of the following wordform.
- Lemmatization model:

– The feature $e_{dict_0}$ is added: this is the set of lemma classes (edit scripts) computed between the word form and the set of corresponding lemmas in the dictionary

This approach has the advantage that it automatically adjusts to the size, nature and quality of the available lexical resource: as the dictionary features are just one of several kinds of sources of information, the classifier learns to to assign them more or less importance depending on how useful they are in predicting the class label. An additional advantage is that the MSD tagset used, and the choices made in lemmatization, need not be the same for the annotated corpus and the dictionary: as long as they are correlated, the inclusion of dictionary features can be helpful. On the other hand this integration means that the model might sometimes make decisions inconsistent with the set of possibilities in the dictionary, which may be undesirable in the case when it gives very high coverage and high quality analyses.

**Experiments with dictionaries**

Here I experiment with the following two dictionaries for Spanish

- DICT-SMALL: Dictionary included in Freeling 1.2. It contains over 71,000 word forms
- DICT-LARGE: Dictionary extracted from the Spanish Resource Grammar project.[10] It contains over 556,000 word forms (it is included in the Freeling 2.0 distribution)

The dictionaries do not include proper names. Both dictionaries use the same tagset, which is also the same as the one used to annotate the AnCora corpus.

Tables 6.14 and 6.15 summarize the evaluation results.

For the SMALL training set the use of the lexicon gives a large relative error reduction for lemmatization (31.02% and 35.65% for DICT-SMALL and DICT-LARGE respectively), and a smaller but still respectable reduction for MSD tagging (10.58% and 16.23%) and joint analysis (9.72% and 15.07%). It seems that the additional benefit from using an over 7 times larger lexicon is relatively small – most improvement seems to come from information covered in the smaller resource. As expected, the dictionary features

[10] http://www.upf.edu/pdi/iula/montserrat.marimon/srg.html

| All words | | | |
|---|---|---|---|
| | MSD-tagging | Lemmatization | Joint |
| DICT-SMALL | 94.93 (+0.60) | 98.51 (+0.67) | 94.43 (+0.60) |
| DICT-LARGE | 95.25 (+0.92) | 98.61 (+0.77) | 94.76 (+0.93) |
| Seen words (in training) | | | |
| | MSD-tagging | Lemmatization | Joint |
| DICT-SMALL | 97.45 (+0.18) | 99.42 (+0.29) | 97.40 (+0.18) |
| DICT-LARGE | 97.49 (+0.23) | 99.38 (+0.24) | 97.47 (+0.25) |
| Unseen words (in training) | | | |
| | MSD-tagging | Lemmatization | Joint |
| DICT-SMALL ($n$=1305) | 78.16 (+3.37) | 92.41 (+3.22) | 74.64 (+3.37) |
| DICT-LARGE ($n$=1305) | 80.31 (+5.52) | 93.49 (+4.29) | 76.70 (+5.44) |
| Unseen words (in training or dictionary) | | | |
| | MSD-tagging | Lemmatization | Joint |
| DICT-SMALL ($n$=831) | 72.92 (+0.12) | 92.90 (+0.24) | 70.04 (+0.48) |
| DICT-LARGE ($n$=509) | 64.83 (+1.18) | 92.34 (+1.57) | 60.31 (+1.57) |

Table 6.14: Evaluation results of the BASIC+DICT model with the SMALL training set with lexicons of various sizes for Spanish. Numbers in brackets indicate accuracy improvement over the BASIC model with the same training set

| All words | | | |
|---|---|---|---|
| | MSD-tagging | Lemmatization | Joint |
| DICT-SMALL | 95.69 (+0.29) | 98.81 (+0.29) | 95.32 (+0.30) |
| DICT-LARGE | 95.67 (+0.27) | 98.98 (+0.46) | 95.33 (+0.31) |
| Seen words (in training) | | | |
| | MSD-tagging | Lemmatization | Joint |
| DICT-SMALL | 97.38 (+0.09) | 99.41 (+0.19) | 97.36 (+0.11) |
| DICT-LARGE | 97.16 (−0.13) | 99.41 (+0.19) | 97.14 (−0.11) |
| Unseen words (in training) | | | |
| | MSD-tagging | Lemmatization | Joint |
| DICT-SMALL ($n = 877$) | 78.11 (+2.39) | 92.59 (+1.37) | 74.12 (+2.28) |
| DICT-LARGE ($n = 877$) | 80.16 (+4.45) | 94.53 (+3.31) | 76.51 (+4.68) |
| Unseen words (in training or dictionary) | | | |
| | MSD-tagging | Lemmatization | Joint |
| DICT-SMALL ($n = 666$) | 74.92 (+0.15) | 92.49 (−0.75) | 71.02 (+0.00) |
| DICT-LARGE ($n = 433$) | 68.59 (+1.62) | 92.61 (+0.69) | 63.28 (+1.39) |

Table 6.15: Evaluation results of the BASIC+DICT model with the FULL training set with lexicons of various sizes for Spanish. Numbers in brackets indicate accuracy improvement over the BASIC model with the same training set

help predict the analysis for words unseen in the training corpus, but have little effect on words unseen in either the corpus or the lexicon (the small improvements for those words are not statistically significant, with $p$-values $> 0.5$ according to binomial test).

For the FULL training set there are still large relative error reductions for lemmatization (19.59% and 31.08%), especially for DICT-LARGE. The improvements for MSD tagging and joint analysis for all words are smaller, around 6% relative error reduction, but statistically significant with $p$-values below 0.05. It is noteworthy that for all words, for MSD tagging and for joint analysis the bigger dictionary does not seem to lead to any further improvement over the smaller one. For words unseen in training set, all improvements are statistically significant; for those unseen in both corpus and dictionary the differences are not statistically significant.

Thus, as expected, using annotated data in the form of morphological lexicons in addition to annotated corpora is beneficial, especially so when using a relatively small training corpus. Dictionaries, although lacking contextual and frequency information help alleviate data sparseness in small training sets.

In the next section I compare the performance of the models described here for Spanish to the morphological analyzer included in Freeling to gain an insight to how the approach proposed here compares to the more common architecture used by Freeling.

### Morfette vs Freeling

I evaluated **Morfette** against two different Freeling configurations: one using the DICT-SMALL dictionary and one with DICT-LARGE. In both cases I use the latest Freeling version 2.0. As for the evaluations described in Section 6.3 I had to ensure that Freeling does not retokenize the input in order to be able to compute accuracy: this means that the Freeling modules for recognition of quantities, numbers, locations and named entities need to be disabled. Version 2.0 also uses a different analysis for contractions such as *al* and *del* as well as word-final clitics such as in *desarrollar**lo***, which is incompatible with our training and test set: I postprocessed Freeling output to adjust it. Since Freeling's treatment of named entities and numbers was disabled, in the evaluation proper names, numbers, dates and also punctuation were ignored: i.e. all tokens whose gold MSD starts with one of *np, w, z, ao, f* were filtered out prior to evaluation.

Freeling - All words

| | MSD-tagging | Lemmatization | Joint |
|---|---|---|---|
| DICT-SMALL | 91.51 | 94.73 | 89.44 |
| DICT-LARGE | 92.58 (+1.07) | 96.55 (+1.81) | 91.86 (+2.42) |

Table 6.16: Evaluation results for Freeling with two different dictionaries

Morfette - All words

| | MSD-tagging | Lemmatization | Joint |
|---|---|---|---|
| BASIC SMALL | 95.88 (+3.30) | 97.78 (+1.23) | 95.52 (+3.66) |
| BASIC+DICT-LARGE FULL | 97.32 (+4.75) | 99.10 (+2.55) | 97.17 (+5.31) |

Table 6.17: Evaluation results for **Morfette** in two configurations. The numbers in brackets indicate improvement over Freeling with DICT-LARGE

Table 6.16 shows the Freeling scores with the two dictionaries described previously. Table 6.17 shows the scores obtained in the same way as for Freeling, i.e. ignoring the MSDs described above, for two **Morfette** configurations: BASIC trained on the SMALL corpus, and BASIC+DICT with DICT-LARGE trained on the FULL corpus. The scores are given for all words only, and not for unseen and seen words. This is because Freeling does not use the same training set as **Morfette**; rather the default POS tagging model which comes with Freeling is used.[11]

The first observation is that in the case of Freeling the difference in the dictionary size between DICT-SMALL and DICT-LARGE translates into more pronounced score improvements than what we saw for **Morfette**. The second is that even the resource-light **Morfette** configuration gives substantial error rate reductions over Freeling with the large lexicon: 44.44% and 35.66% for MSD tagging and lemmatization respectively. The differences are even more pronounced between the resource rich **Morfette** configuration and the best Freeling numbers: 63.93% and 73.90% error reduction for MSD tagging and lemmatization respectively.

Freeling is a mature and widely used system which efficiently performs a large array of useful language processing tasks, and it has not been tuned specifically to joint MSD

[11] I tried retraining Freeling's POS tagging model on the same data as used to train **Morfette**, but the results turned out to be slightly worse.

tagging and lemmatization. It also has a very simple lookup approach to lemmatization. Given that, the fact that my dedicated approach outperforms it is hardly surprising. On the other hand **Morfette**'s feature set is quite generic, the annotated resources it was trained on are relatively small in size, and relatively little effort was expended for tuning the system. Thus the large gains over Freeling inspire confidence that the approach is on the right track.

### 6.4.6 Improving lemma class discovery

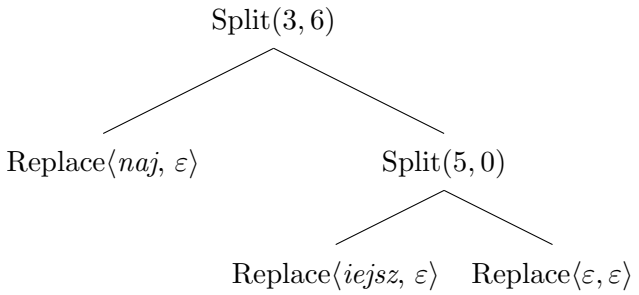
As noticed in Section 6.4.4 the REVERSE-EDIT-LIST version of edit script makes it is difficult to induce sufficiently general lemma classes for cases where morphology affects word beginnings rather than, or in addition to, word endings. In this Section I investigate whether another instantiation of edit script which makes less assumptions about where in the string morphological changes take place can improve lemma class induction and help boost lemmatization results.

#### Edit tree

The problem with REVERSE-EDIT-LIST is that it always indexes edits starting from the end of the string: this means that lemma classes involving prefixation are not general: they unnecessarily depend on the word length. EDIT-TREE is a variation of edit script which indexes prefixes relative to the beginning of the string and suffixes relative to the end.

The idea is to find the longest common substring (LCS) between the form $w$ and the lemma $w'$. We know that the portions of the string in the lemma before (prefix) and after (suffix) the LCS need to be modified in some way, while the LCS (stem) stays the same. If there is no LCS, then we simply record that we need to replace $w$ with $w'$. As for the modifications to the prefix and the suffix, we apply the same procedure recursively: we try to find the LCS between the prefix of $w$ and the prefix of $w'$: if we find one, we recurse again; if we do not, we record the replacement; we do the same for the suffix. So for example the EDIT-TREE of the Polish form-lemma pair *najtrudniejszy* and *trudny*[12] is the following:

[12] *hardest* and *hard*




It encodes the following operations:

- Split *najtrudniejszy* at position 3 and $length(najtrudniejszy) - 6$ to get the prefix *naj* the stem *trudn* and the suffix *iejszy*.

  Concatenate:

  - Replace *naj* with $\varepsilon$.
  - Stem *trudn*
  - Split *iejszy* at position 5 and $length(iejszy) - 0$ to get the prefix *iejsz*, the stem $y$ and the suffix $\varepsilon$.

    Concatenate:

    * Replace *iejsz* with $\varepsilon$
    * Stem $y$
    * Replace $\varepsilon$ with $\varepsilon$

More formally the EDIT-TREE can be defined as follows. Let the function *lcs* : $(\Sigma^* \times \Sigma^*) \to (\mathbb{N} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N})$ take two strings $w_{1..n}$ and $w'_{1..m}$ and return the 4-tuple $(i, j, k, l)$ where $w_{i..n-j} = w'_{k..m-l}$ is the (first) longest common substring of strings $w$ and $w'$.

Let the function *split* : $(\Sigma^* \times \mathbb{N} \times \mathbb{N}) \to (\Sigma^* \times \Sigma^* \times \Sigma^*)$ take the string $w_{1..n}$, and indices $i$ and $j$ and return the triple $(w_1..w_i, w_{i+1..n-j}, w_{n-j+1..n})$.

The EDIT-TREE is represented by the following recursive sum-of-products type:

$$\text{EditTree} = (\text{Replace} : (\Sigma^* \times \Sigma^*)) + (\text{Split} : (\mathbb{N} \times \mathbb{N}) \times \text{EditTree} \times \text{EditTree}) \quad (6.4)$$

That is an EditTree is either a leaf node labeled Replace, which stores a tuple of strings, or it is an internal node labeled Split which stores a tuple of natural numbers and has an EditTree as the left child and an EditTree as the right child.

The function *et* builds this tree structure given two strings $w$ and $w'$:

$$et(w, w') = \begin{cases} \text{Replace}\langle w, w' \rangle & \text{if } w \text{ and } w' \text{ have no LCS} \\ \text{Split}\langle (i_w, j_w), et(w_{prefix}, w'_{prefix}), et(w_{suffix}, w'_{suffix}) \rangle & \text{otherwise} \end{cases} \tag{6.5}$$

where

$$\begin{aligned} (i_w, j_w, i_{w'}, j_{w'}) &= lcs(w, w') \\ (w_{prefix}, w_{stem}, w_{suffix}) &= split(w, i_w, j_w) \\ (w'_{prefix}, w'_{stem}, w'_{suffix}) &= split(w', i_{w'}, j_{w'}) \end{aligned} \tag{6.6}$$

EditTree encodes the operations to perform on an input string $w$ in order to transform it into the output string $w'$. The operations are applied recursively as follows:

$$\begin{aligned} apply(\text{Replace}\langle v, v' \rangle, w) &= v' \\ apply(\text{Split}\langle (i, j), l, r \rangle, w) &= apply(l, w_{prefix}) \oplus w_{stem} \oplus apply(r, w_{suffix}) \end{aligned} \tag{6.7}$$

where

$$(w_{prefix}, w_{stem}, w_{suffix}) = split(w, i, j)$$

and $\oplus$ is string concatenation.

Using EDIT-TREE as the edit script (unlike REVERSE-EDIT-LIST) preserves analogies like the one holding between the two Polish form-lemma pairs[13]:

*najładniejszy : ładny :: najtrudniejszy : trudny*

since

*apply(et(najładniejszy, ładny), najtrudniejszy) = trudny*

The better generalization of EDIT-TREE as compared to REVERSE-EDIT-LIST is suggested by the smaller number of unique lemma classes discovered in the SMALL training set for Polish by the former compared to the latter method: 1110 vs. 1209.

Table 6.18 shows evaluation results for the SMALL Polish training set when using EDIT-TREE for lemma class induction. There is a modest (relative error reduction of 2.24%) but statistically significant ($p$-value $< 0.005$) improvement for lemmatization

[13] *prettiest, pretty* and *hardest, hard*

| All words | | |
|---|---|---|
| MSD-tagging | Lemmatization | Joint |
| 81.94 (+0.07) | 93.44 (+0.15) | 81.28 (+0.09) |
| Seen words | | |
| 86.97 (+0.01) | 97.53 (+0.05) | 86.83 (+0.03) |
| Unseen words | | |
| MSD-tagging | Lemmatization | Joint |
| 62.22 (+0.30) | 77.42 (+0.54) | 59.52 (+0.34) |

Table 6.18: Results for the BASIC feature set on SMALL training set, using the EDIT-TREE as lemma class for Polish. Numbers in brackets indicate improvement over the same configuration with REVERSE-EDIT-LIST

for all tokens. The small magnitude of the improvement is most probably due to the relatively infrequent occurence of prefixal morphological phenomena in Polish. In Spanish and Romanian they are even less frequent, and differences between EDIT-TREE and REVERSE-EDIT-LIST for them are very small and not statistically significant.

In order to test the EDIT-TREE lemma class induction on more interesting and more challenging data I decided to try to run **Morfette** data taken from the Celtic languages Welsh and Irish. Celtic features word initial consonant mutations: i.e. the first consonant of a word may change depending on the word's grammatical context. Those mutations can co-occur with the more common suffixal morphology.

The Welsh data comes form the Cronfa Electroneg o Gymraeg corpus (Ellis et al., 2001). I used a training set of 70,000 tokens and a test set of 10,000 tokens. The Irish data was provided by Elaine Uí Dhonnchadha. I used 10,000 tokens for the test set and approximately 60,000 for training.

Table 6.19 shows the results for the EDIT-TREE on this data together with improvements over the baseline using REVERSE-EDIT-LIST. The difference for Welsh is somewhat more pronounced than for Polish (4.58% relative error reduction for lemmatization, all words, $p$-value $< 0.005$) but still rather modest. For Irish however there are larger gains: 12.14% relative error reduction for lemmatization for all words, with $p$-value of $3\times10^{14}$

It seems that the more general edit script instantiation does permit us to find

| Unseen word ratio | | | |
|---|---|---|---|
| Welsh | 11.25 | | |
| Irish | 12.61 | | |
| All words | | | |
| | MSD-tagging | Lemmatization | Joint |
| Welsh | 87.72 (+0.18) | 92.92 (+0.34) | 84.86 (+0.35) |
| Irish | 79.64 (+0.12) | 94.21 (+0.80) | 78.81 (+0.40) |
| Seen words | | | |
| | MSD-tagging | Lemmatization | Joint |
| Welsh | 92.11 (+0.03) | 96.12 (−0.16) | 90.35 (−0.05) |
| Irish | 85.43 (+0.06) | 98.00 (+0.03) | 85.07 (+0.05) |
| Unseen words | | | |
| | MSD-tagging | Lemmatization | Joint |
| Welsh | 53.07 (+1.33) | 67.64 (+4.27) | 41.51 (+3.47) |
| Irish | 39.49 (+0.56) | 67.96 (+6.11) | 35.45 (+2.85) |

Table 6.19: Results for the BASIC feature set, using the EDIT-TREE as lemma class for Welsh and Irish. Numbers in brackets indicate improvement over the same configuration with REVERSE-EDIT-LIST.

better lemma classes, but the magnitude of the improvement varies depending on the language. In the case of Irish the improvements are robust and the richer lemma class representation is clearly beneficial. In other cases, such as Polish and also Welsh, in spite of the very strong bias towards suffixation that REVERSE-EDIT-LIST encodes, it is not trivial to substantially improve on the scores achieved with it.

### 6.4.7 Conclusion

**Morfette** has two important features. Firstly, it is modular in the sense that the morphological-tagging and lemmatization models can use different features, can be trained separately, and even use different classifiers. Secondly, in spite of such modularity, the way the search algorithm combines MSD and lemma-class conditional probabilities means that the outputs of the two models are integrated at decoding time and their predictions are combined into an overall scoring over MSD-tag-lemma-class pair sequences.

This data-driven approach can be adapted to scenarios with different types of resources available: it can be trained on nothing more than a modest-sized corpus an-

notated with lemmas and morphological tags, or it can additionally make use of morphological dictionaries if such are available. The dictionary information is exploited by including it in the log-linear models as features. This means that the tag sets and annotations in the training corpus and the lexicon resources do not have to be identical.

**Morfette** substantially improves on the scores on the Spanish test set obtained by a popular and mature language analyzer Freeling: even in the resource-poor configuration, using a small training set and no lexical resources. Similar comparisons with other systems and on other languages would be desirable to strengthen the conclusion that the **Morfette** approach is competitive with state-of-the-art morphological analyzers.

The novel lemma class induction method depends on the abstract notion of edit script or a specification of how to convert an inflected word form to the corresponding lemma. The current default edit script type used in the system, REVERSE-EDIT-LIST assumes that inflectional morphology affects word endings and does not produce good general lemma classes when this assumption is violated. Experiments with a less biased alternative edit script type EDIT-TREE showed improvement in lemmatization score; however there is still scope for further investigation in lemma class induction.

## 6.5 Morphological Analysis and Synthesis: ILP and Classifier-Based Approaches

In this section I compare two machine learning approaches to building models of morphological analysis and synthesis: the Inductive Logic Programming (ILP) approach used in (Manandhar et al., 1998) and the classifier-based approach used here.

The lemmatization method described in Sections 6.3 and 6.4 differs in several ways from the ILP method using CLOG and thus comparative performance is difficult to assess.

In what follows I take two basic ideas from Section 6.3, namely

- treating lemmatization as a classification task,
- using a version of edit script to induce lemma classes.

However, I use an experimental setup similar to that described in (Manandhar et al.,

1998) and perform evaluation of both systems on the same data, which makes a meaningful comparison possible.

More specifically, the classifier-based lemmatization method introduced in 6.3 works on running text, that is the examples to classify are word tokens in context. The classifier uses features extracted from the word forms and *also from the surrounding context*, i.e. the preceding and following word forms. The output to learn to predict is always the edit script which takes the word-form to the corresponding lemma.

In contrast, the training examples in the ILP method are isolated wordform-MSD-lemma triples, i.e. the learner works with word-types, rather than word tokens. During prediction, either the wordform or the lemma are output, depending on whether we are performing synthesis or analysis. That is both the training and test examples come from a lexicon rather than being from a corpus of sentences. However, it is easy to adapt the classifier method to the setting analogous to that used in (Manandhar et al., 1998). We need to remove all the context features since they are no longer available. They are effectively replaced by the MSD-tag associated with the word type or lemma to classify. In order to perform synthesis rather than analysis, we compute the edit script from the lemma to word form rather than the other way round.

Those modifications to the classifier-based lemmatization method make it possible to have a fair and informative comparison to the ILP approach to learning morphology.

### 6.5.1 Data

The data I use for this experiment are almost the same as those used in (Manandhar et al., 1998), i.e. they come from the Multext-EAST corpus. However, I use the most recent Version 3 of that resource. Also I do not experiment with Estonian data, since the morphologically analyzed Appendix for this language is missing from the resource. I use data from English, Romanian, Czech and Slovene.

The training data comes from parts 1 through 3 of the corpus, while the test set comes from part 4, the Appendix. The training and testing data is generated as follows, to mimic the setup used in (Manandhar et al., 1998):

For each token in the text, all its morphological analyses (lemma-MSD pairs) are extracted from the accompanying lexicon. Then all analyses with MSDs for nouns and

| Feature notation | Description |
|---|---|
| $f$ | Lowercased wordform |
| $s_n(f),\ n = 1 \cdots 7$ | Suffixes of length $n$ |
| $p_n(f),\ n = 1 \cdots 3$ | Prefixes of length $n$ |
| $m$ | MSD tag |

Table 6.20: Features for lexical analysis model

| Feature notation | Description |
|---|---|
| $l$ | Lowercased lemma |
| $s_n(l),\ n = 1 \cdots 7$ | Suffixes of length $n$ |
| $p_n(l),\ n = 1 \cdots 3$ | Prefixes of length $n$ |
| $m$ | MSD tag |

Table 6.21: Features for lexical synthesis model

adjectives are kept while other tokens are discarded. Finally duplicates are removed, so that the training and testing data consists of morphologically analyzed word types, not tokens. Duplicates are removed separately for training set and test set, which means that there are some examples in the test set that also appear in the training set.

### 6.5.2 Model and features

Here, as in Section 6.4, I use the Maximum Entropy algorithm for classification.

In order to match the original ILP work, I trained two sets of models: one for analysis, mapping wordform-MSD pairs to lemmas, and one for synthesis, mapping lemma-MSD pairs to wordforms. The only difference is which of the two strings is given as input and which is produced as output – no other changes were made to the overall method. Both model sets used EDIT-TREE for class induction (with the arguments swapped for the synthesis model).

As already mentioned, the features have to be adapted to the new setting: none of the context word features can be used, while the MSD-tags provide an important new feature. Tables 6.20 and 6.21 describe the features sets used: they are analogous for both sets of models, identical for all the languages, and were chosen based on cross-validation on the training set. The MaxEnt smoothing parameter $\sigma^2 = 10$ was also

```
split([X,Y|Z],[X],[Y|Z]).
split([X|Y],[X|Z],W) :- split(Y,Z,W).

mate(W1,W2,[],[],Y1,[]):-
    split(W1,W2,Y1).

mate(W1,W2,[],[],[],Y2):-
    split(W2,W1,Y2).

mate(W1,W2,[],[],Y1,Y2):-
    split(W1,X,Y1),
    split(W2,X,Y2).

mate(W1,W2,P1,P2,Y1,Y2):-
    split(W1,P1,W11),
    split(W2,P2,W22),
    split(W11,X,Y1),
    split(W22,X,Y2).

% total suppletion
mate(W1,W2,[],[],W1,W2).
```

Figure 6.3: Background predicate `mate/6`

chosen in the same way.

For experiments with CLOG[14], the background predicate `mate/6` was used, shown in Table 6.3.

Following (Manandhar et al., 1998), a separate program was learned for each MSD (I did not collapse any MSDs together). When an MSD in the test set is absent from the training data, the system backs off to outputting the input string. The same backoff was used in case of the predicate failing.

### 6.5.3 Results and error analysis

Tables 6.22 and 6.23 summarize the results for analysis and synthesis respectively, for all input tuples. In most cases the differences between the two systems are relatively small, with higher MaxEnt+EDIT-TREE scores, giving relative error reductions between 2.67% and 18.16%. The only exception is Czech synthesis where CLOG is better with

[14]I would like to thank Tomaž Erjavec for providing me with CLOG software for the purpose of this research

| | Clog | MaxEnt+et | RER |
|---|---|---|---|
| English | 98.93 | 99.04 | 10.28 |
| Romanian | 96.67 | 97.10 | 12.91 |
| Czech | 95.15 | 95.70 | 11.34 |
| Slovene | 97.34 | 97.66 | 12.03 |

Table 6.22: Morphological analysis results - all

| | Clog | MaxEnt+et | RER |
|---|---|---|---|
| English | 99.25 | 99.36 | 14.67 |
| Romanian | 97.38 | 97.45 | 2.67 |
| Czech | 92.01 | 91.17 | −9.51 |
| Slovene | 95.98 | 96.71 | 18.16 |

Table 6.23: Morphological synthesis results - all

| | Clog | MaxEnt+et | RER |
|---|---|---|---|
| English | 99.71 | 99.71 | 0.00 |
| Romanian | 99.88 | 99.88 | 0.00 |
| Czech | 99.51 | 99.40 | −18.33 |
| Slovene | 99.44 | 99.44 | 0.00 |

Table 6.24: Morphological analysis results - seen

| | Unseen ratio | Clog | MaxEnt+et | RER |
|---|---|---|---|---|
| English | 25.91 | 96.69 | 97.11 | 12.69 |
| Romanian | 40.65 | 91.99 | 93.03 | 12.98 |
| Czech | 42.95 | 89.37 | 90.77 | 13.17 |
| Slovene | 48.26 | 95.10 | 95.75 | 13.27 |

Table 6.25: Morphological analysis results - unseen

| | Clog | MaxEnt+et | RER |
|---|---|---|---|
| English | 99.42 | 99.57 | 25.86 |
| Romanian | 99.76 | 99.64 | −33.33 |
| Czech | 91.19 | 91.69 | 5.68 |
| Slovene | 98.54 | 98.74 | 13.70 |

Table 6.26: Morphological synthesis results - seen

| | Unseen ratio | CLOG | MaxEnt+ET | RER |
|---|---|---|---|---|
| English | 25.59 | 98.74 | 98.74 | 0.00 |
| Romanian | 40.51 | 93.88 | 94.23 | 5.72 |
| Czech | 39.49 | 93.28 | 90.37 | −30.22 |
| Slovene | 47.74 | 93.17 | 94.49 | 19.33 |

Table 6.27: Morphological synthesis results - unseen

a relative error reduction of 9.51%.

Tables 6.24 and 6.26 show results for the subset of input tuples seen in the training data, while tables 6.25 and 6.27 show unknown tuple ratio and results for unseen inputs.

For both systems the performance on Czech data looked surprising bad, especially for synthesis, so I inspected the training data more closely. It seems that the major source of error are negated adjectives such as *nebezpečný* (dangerous). In the Czech lexicon both *nebezpečný* and *bezpečný* (safe), are assigned the same lemma-MSD pair. Thus in the case of synthesis, in principle each adjectival lemma-MSD pair has two possible solutions, one non-negated and one negated form. In the case of analysis it simply creates more cases where both the suffix and prefix have to be modified in the input form, which seems more difficult to learn than the simpler cases of suffix modification. This treatment of negated adjectives is questionable since it makes synthesis non-deterministic, and the analysis does not preserve all the information present in the wordform. Arguably if the negative prefix is treated as an inflection, then there should be a corresponding feature in the MSD recording its presence. It is not clear how Manandhar et al. (1998) deal with this issue or if it even arises in their version of the Multext-EAST data.

This interpretation of the poor performance on Czech data is confirmed by rerunning the systems on data with negated adjectives removed. I removed from both training and test data all examples where the wordform starts with *ne* whose lemma does not start with*ne*, except for the superlative adjectives which start with *nej*. On this modified Czech data set CLOG scores 97.03% and 96.63% on analysis and synthesis respectively, while MaxEnt+ET scores 97.28% and 96.16%

From these results it is evident that the classifier + edit script approach to morpho-

logical analysis is competitive with, and in most cases improves on the ILP method. The advantages of ILP are its greater generality, as well as the fact that the learned decision lists can be easily interpreted by humans.

On the other hand the classifier approach offers better performance and easier scalability. Since the classification paradigm is the most common one in machine learning, it comes with many well-understood, efficient and well-performing algorithms. If the classifier used is a probabilistic one such as MaxEnt, there is the considerable additional advantage of being able to easily integrate it in larger probabilistic models, as in Section 6.4.

## 6.6 Summary

In this chapter I have reviewed the most common existing approaches to data driven morphological analysis and lemmatization, and I have proposed a novel perspective on lemmatization as a classification task. I showed that lemmatization classes can be induced automatically from annotated data using the idea of edit script which is a representation of the transformation which maps an input string (e.g. wordform) to the corresponding output string (e.g. the lemma).

I described successfully lemmatizing running text using REVERSE-EDIT-LISTS as lemma classes and an SVM classifier for data from eight different languages. I then proposed a method which performs morphological analysis and lemmatization in a joint fashion, by learning two models, one for morphological tagging and one for lemmatization, and by integrating their predictions in an integrated probabilistic model. I also showed how to exploit information from morphological lexicons within this framework, and how to improve lemma class induction by using a more general version of edit script, the EDIT-TREE.

Finally I contrasted the edit script- and classifier-based approach to learning morphology with the Inductive Logic Programming method, and showed that the former gives competitive or improved performance while being scalable, well understood, and easy to use.

# Chapter 7

# Conclusion

The approach to treebank-based LFG parsing developed by Cahill et al. (2002, 2004) has been shown to be highly competitive. Part of that success is due to its modular, pipeline-like architecture which makes it easy to swap components, and experiment with different combinations. In this way a system can be built which leverages state-of-the-art models for the different subparts of the parsing system.

In this thesis I have described the work on the design, integration and evaluation of two of the components for the DCU LFG parsing architecture. Even though this work has been motivated and driven by this particular parsing approach, its usefulness transcends this context. This is again the result of the modular and flexible nature of the system: most of the submodules are general-purpose processing engines while the LFG-specific part is localized in the annotation algorithm, and to some degree in the NLD module. This means that my enhancements to LFG parsing can be easily reused in other applications.

## 7.1 Summary of Main Contributions

The main achievements described in this thesis are the following:

**Spanish treebank-based LFG parsing**

- I have overhauled and substantially extended the range of phenomena treated in the Spanish annotation algorithm. I also revised and extended the gold standard

which now includes 338 sentences with their corresponding f-structures. This exercise served two purposes: first it helped to identify areas where the existing LFG parsing architecture for English needed further work to make it less language dependent and more portable. Second, it enabled the work on developing and evaluating a function labeling model for Spanish.

**Function labeling**

- I have developed a function labeler for Spanish which substantially outperforms the previously used method of using the c-structure parser to obtain function-labeled trees. The use of this model in the LFG parsing pipeline also improves the f-structure quality as compared to the baseline method.

- I have described a training regime for an SVM-based function labeling model where trees output by a parser are used in combination with treebank trees in order to achieve better similarity between training and test examples. This model outperforms all previously described function labelers on the standard English Penn II treebank test set.

**Morphological analysis**

- I have developed a method to cast lemmatization as a sequence labeling task. It relies on the notion of edit script which encodes the transformations on the word form which will convert it into the corresponding lemma. Edit scripts can be automatically computed from a corpus of word-lemma pairs and used as lemmatization classes: thus class labels themselves are induced from data and need not be manually predefined. A lemmatization model can be learned from a corpus annotated only with lemmas, with no explicit part-of-speech information.

- I have built the **Morfette** system which performs morphological analysis by learning a morphological tagging model and a lemmatization model, and combines the predictions of those two models to find a globally good sequence of MSD-lemma pairs for a sentence.

- I have shown that integrating information from morphological dictionaries into the Maximum Entropy models used by **Morfette** is straightforward and can substantially reduce error, especially on words absent from training corpus data.
- I have developed an instantiation of the edit script, the Edit Tree, which improves lemmatization class induction in the case where inflectional morphology affects word beginnings in addition to word endings, and have shown that the use of this edit script version results in statistically significant error reductions on test data in Polish, Welsh and Irish.
- I compared the proposed morphology models against existing systems (Freeling and Clog): in both cases my proposed models showed competitive or superior performance

## 7.2 Directions for Future Research

Research extending and following from the work reported in this thesis falls into two broad categories. Firstly, there are multiple interesting problems related to extending and improving the specific models discussed. Secondly, more work is needed to better integrate the proposed models into the overall LFG parsing architecture, and port, train and tune them on data for other languages, both within the GramLab project and beyond.

### 7.2.1 Grammatical functions

I have shown in Chapter 5 that using the simple local classifier approach to learning a function labeling model gives good results. However, in reality labeling decisions at different nodes of the parse tree are dependent on each other. As a simple example consider the cross-linguistically wide-spread constraint which ensures that a given verb cannot subcategorize for more than one argument with the same grammatical function, i.e. a verb can only govern a single subject, or a single direct object etc. In the scenario where a local classification decision is taken at each node this very strong constraint is not enforced in any way. For instance for a verb each of its neighboring nodes may be

independently a good candidate for a subject label; however if one of them is labeled as a subject, the others should not.

A simple resolution to this inadequacy would be to partition nodes in the parse tree into sets which are governed by the same verb – for example for the Spanish Cast3LB treebank it would be sufficient to group together all sisters of the same *gv* (verb group) constituent – and label those groups jointly. The simplest such joint labeling could use a sequence labeling approach, make labeling decision dependent on previous labelings, and choose a label sequence which is globally optimal. More complex structured prediction schemes could also be explored.

As currently used, the function labeler is a module which is independent of the overall LFG parsing system: the annotation algorithm uses its output, but the module itself is unaware of the context in which it is used. This makes is easier to reuse in scenarios other than LFG parsing, but to achieve best performance a tighter integration might be helpful. Currently model parameters and algorithm parameters (hyperparameters) are tuned to optimize the accuracy on the original treebank function labels: however some of these labels are not used by the LFG annotation algorithm; also some distinctions encoded in the labels are collapsed in the f-structures. Additionally some labels may influence larger portions of the f-structure than others. Thus it might be beneficial to optimize more directly the score on the f-structures rather than the raw treebank function labeling accuracy: to some degree this can be achieved by simply dropping LFG-irrelevant labels and collapsing LFG-irrelevant distinctions from the training data, and training as usual. For model selection, including feature selection and hyperparameter tuning, it would also be relatively straightforward to optimize directly on f-structure f-scores.

### 7.2.2 Morphology and Morfette

As stressed in Chapter 6, the morphology models investigated in this thesis are quite generic. Although they are usable in realistic scenarios as they are, for best performance some amount of work would need to be dedicated to model selection: the feature sets and the parameter settings should be tuned to specific languages and specific processing tasks. There are extensions to Maximum Entropy optimization algorithms

which incorporate feature selection in the training step (Berger et al., 1996): it would be interesting to implement such a scheme for **Morfette** in order to automate model selection to some degree and make the system better adapt to characteristics of the training data.

The **Morfette** system as described in Section 6.4 learns two separate MaxEnt models, one for MSD tagging and one for lemmatization. This has the virtue of simplicity, and, as we have seen, gives good performance. However, it would be possible to envision other more fine-grained decompositions of the overall morphological analysis model. As mentioned in Section 6.2.4, one approach to reducing data-sparseness and computational load for languages with highly complex morphology is to predict morphological features encoded in MSDs independently and then combine those predictions for the full MSD. It would be worthwhile to adopt such a scheme since it could potentially offer improved scalability of the simple approach of learning each MSD in one go.

Another decomposition of the MSD tagging task has been explored in (Tufiş, 1999; Tufiş and Dragomirescu, 2004; Ceauşu, 2006). They propose to learn the target MSD set in two steps: first learn a reduced tagset and then use the predictions of the model trained on such a reduced set to learn a model which predicts the full tagset. It would be straightforward and could be beneficial to implement a similar approach for **Morfette**.

The lemmatization model could also be easily factored into a number of smaller models, one for each MSD tag: currently the MSD tag of the focus word is one of the features in the lemmatization model. It is an important feature, which stands proxy for the local context: its importance would be made more explicit by conditioning the whole lemmatization model on the predicted MSD tag. One potential benefit would be the possibility of more fine-grained model selection which could be done separately for each MSD-conditioned model.

The research on data-driven morphological analysis reported in this thesis has concerned itself almost exclusively with the supervised learning setting: the only unsupervised aspect is the fact that lemma-classes are not present in the training data but rather are automatically induced from it. However, there is a large body of research on learning morphology from unannotated data: some of this research could

be adapted to leverage the huge amounts of unlabeled data which are nowadays available for many languages. Just as I have shown that using dictionaries improves the analysis of lexical items not encountered in the training corpus, one would hope that exploiting co-occurrence statistics in large bodies of unlabeled text would enable further improvements. Some very preliminary experiments in this area have been encouraging.

### 7.2.3 Other aspects of LFG parsing

One module in the LFG parsing architecture which has not received much attention in this thesis is the Non-Local Dependency (NLD) resolver. Currently its resolution-ranking model relies on the product of two conditional probability scores: the probability of the subcategorization frame given lemma, and the probability of NLD path given the source grammatical function (GF). It is likely that this simple model could be improved on by using a classifier-based approach, where the resolution candidates could have a rich feature representation, subsuming the lemma, subcat frame, source GF and NLD path "features" used in the current model, and adding other features such as target GF, attributes of the f-structure which is the value of source/target GFs, NLD path length and other sources of information that could contribute to the ranking.

Finally, given the linguistically rich two-level syntactic representations given by the DCU LFG parsing architecture, it would be worthwhile to explore how to leverage them in a system which computes predicate-argument structures. A large body of work exists on the task of semantic role labeling (SRL) especially using the PropBank (Palmer et al., 2005) as the target representation and training resource.

The majority of this research has relied on using machine-learning approaches working with relatively shallow syntactic representations such as chunks or basic phrase-structure trees (see e.g. the CoNLL Shared Tasks in (Carreras and Màrquez, 2004, 2005)). On the other hand, Miyao and Tsujii (2004) and Burke et al. (2005) attempt to directly map deep syntactic representations (HPSG and LFG respectively) to PropBank roles.

Gildea and Hockenmaier (2003) combine the use of deep syntactic representations (CCG) with machine-learning techniques: this is the approach which is most in the spirit of the ideas behind this thesis. As such, it would be worthwhile trying to use

f-structures as input to a machine-learning-based SRL system, thus providing the LFG parsing system with a semantic component.

# Bibliography

Abeillé, A., Clément, L., and Toussenel, F. (2003). Building a treebank for French. In Abeillé, A., editor, *Treebanks: Building and Using Parsed Corpora*, pages 165–188. Kluwer Academic Publishers, Dordrecht, The Netherlands.

Afonso, S., Bick, E., Haber, R., and Santos, D. (2002). "Floresta sintá(c)tica": a treebank for Portuguese. In *LREC 2002: Proceedings of the Third International Language Resources and Evaluation Conference*, pages 1698–1703.

Aha, D., Kibler, D., and Albert, M. (1991). Instance-based learning algorithms. *Machine Learning*, 6(1):37–66.

Aho, A. V., Hirschberg, D. S., and Ullman, J. D. (1976). Bounds on the complexity of the longest common subsequence problem. *Journal of the ACM*, 23(1):1–12.

Aizerman, M., Braverman, E., and Rozonoer, L. (1964). Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25(6):821–837.

Al-Raheb, Y., Akrout, A., van Genabith, J., and Dichy, J. (2006). DCU250 Arabic Dependency Bank: An LFG gold standard resource for the Arabic Penn Treebank. In *Proceedings of the Arabic NLP/MT Conference*, pages 105–116.

Alsina, A. (1997). A theory of complex predicates: evidence from causatives in Bantu and Romance. In Alsina, A., Bresnan, J., and Sells, P., editors, *Complex Predicates*, pages 203–246. Center for the Study of Language and Information, Stanford, CA, USA.

Andrews, A. (1990). Unification and morphological blocking. *Natural Language & Linguistic Theory*, 8(4):507–557.

Andrews, A. D. and Manning, C. D. (1999). *Complex Predicates and Information Spreading in LFG*. Center for the Study of Language and Information, Stanford, CA, USA.

Baayen, R., Piepenbrock, R., and van Rijn, H. (1993). The CELEX lexical database (CD-ROM). Technical report.

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

Bień, J. S. and Woliński, M. (2003). Wzbogacony korpus słownika frekwencyjnego polszczyzny współczesnej. In Linde-Usiekniewicz, J., editor, *Prace lingwistyczne dedykowane prof. Jadwidze Sambor*, pages 6–10. Wydział Polonistyki, Uniwersytet Warszawski, Warsaw, Poland.

Bies, A., Ferguson, M., Katz, K., and MacIntyre, R. (1995). Bracketing guidelines for Treebank II style Penn Treebank project. Technical report, University of Pennsylvania.

Bikel, D. M. (2002). Design of a multi-lingual, parallel-processing statistical parsing engine. In *HLT 2002: Proceedings of the Second International Conference on Human Language Technology Research*, pages 178–182.

Blaheta, D. and Charniak, E. (2000). Assigning function tags to parsed text. In *NAACL 2000: Proceedings of the First Conference of the North American Chapter of the Association for Computational Linguistics*, pages 234–240.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *COLT '92: Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 144–152.

Brants, S., Dipper, S., Hansen, S., Lezius, W., and Smith, G. (2002). The TIGER treebank. In *TLT 2002: Proceedings of the Workshop on Treebanks and Linguistic Theories*, pages 24–41.

Brants, T. (2000). TnT: a statistical part-of-speech tagger. In *Proceedings of the Sixth Conference on Applied Natural Language Processing*, pages 224–231.

Bresnan, J. (2001). *Lexical-Functional Syntax*. Blackwell Publishers, Oxford, UK.

Bresnan, J. and Kaplan, R. (1982). Lexical-functional grammar: A formal system for grammatical representation. In *The Mental Representation of Grammatical Relations*, pages 173–281.

Briscoe, T. and Carroll, J. (2006). Evaluating the accuracy of an unlexicalized statistical parser on the PARC DepBank. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 41–48.

Burke, M. (2006). *Automatic Treebank Annotation for the Acquisition of LFG Resources*. PhD thesis, Dublin City University.

Burke, M., Cahill, A., O'Donovan, R., van Genabith, J., and Way, A. (2004a). Evaluation of an Automatic Annotation Algorithm against the PARC 700 Dependency Bank. In *LFG 2004: Proceedings of the Ninth International Conference on Lexical Functional Grammar*, pages 101–121.

Burke, M., Cahill, A., van Genabith, J., and Way, A. (2005). Evaluating automatically acquired f-structures against PropBank. In *LFG 2005: Proceedings of the 10th International Lexical Functional Grammar Conference*, pages 84–99.

Burke, M., Lam, O., Cahill, A., Chan, R., O'Donovan, R., Bodomo, A., van Genabith, J., and Way, A. (2004b). Treebank-Based Acquisition of a Chinese Lexical-Functional Grammar. In *PACLIC 2004: Proceedings of the 18th Pacific Asia Conference on Language, Information and Computation*, pages 161–172.

Butt, M. (1997). Complex predicates in Urdu. In Alsina, A., Bresnan, J., and Sells, P., editors, *Complex Predicates*. Center for the Study of Language and Information, Stanford, CA, USA.

Butt, M., Dyvik, H., King, T. H., Masuichi, H., and Rohrer, C. (2002). The Parallel Grammar project. In *Proceedings of the Workshop on Grammar Engineering and Evaluation (COLING02)*, pages 1–7.

Cahill, A., Burke, M., O'Donovan, R., Riezler, S., van Genabith, J., and Way, A. (2008). Wide-Coverage Deep Statistical Parsing Using Automatic Dependency Structure Annotation. *Computational Linguistics*, 34(1):81–124.

Cahill, A., Burke, M., O'Donovan, R., van Genabith, J., and Way, A. (2004). Long-Distance Dependency Resolution in Automatically Acquired Wide-Coverage PCFG-Based LFG Approximations. In *ACL 2004: Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, pages 320–327.

Cahill, A., Forst, M., Burke, M., McCarthy, M., O'Donovan, R., Rohrer, C., van Genabith, J., and Way, A. (2005). Treebank-Based Acquisition of Multilingual Unification Grammar Resources. *Journal of Research on Language and Computation; Special Issue on "Shared Representations in Multilingual Grammar Engineering"*, pages 247–279.

Cahill, A., McCarthy, M., van Genabith, J., and Way, A. (2002). Parsing with PCFGs and Automatic F-Structure Annotation. In *LFG 2002: Proceedings of the Seventh International Conference on Lexical Functional Grammar*, pages 76–95.

Campbell, R. (2004). Using linguistic principles to recover empty categories. In *ACL 2004: Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, pages 645–653.

Canisius, S., Van den Bosch, A., and Daelemans, W. (2006). Constraint satisfaction inference: Non-probabilistic global inference for sequence labelling. In *Proceedings of the EACL 2006 Workshop on Learning Structured Information in Natural Language Applications*, pages 9–16.

Carreras, X., Chao, I., Padró, L., and Padró, M. (2004). Freeling: An open-source suite of language analyzers. In *LREC 2004: Proceedings of the 4th Conference on on Language Resources and Evaluation*, pages 239–242.

Carreras, X. and Màrquez, L. (2004). Introduction to the CoNLL-2004 Shared Task: Semantic Role Labeling. In *CoNLL-2004: Eighth Conference on Computational Natural Language Learning*, pages 89–97.

Carreras, X. and Màrquez, L. (2005). Introduction to the CoNLL-2005 Shared Task: Semantic Role Labeling. In *CoNLL 2005: Ninth Conference on Computational Natural Language Learning*, pages 152–164.

Carreras, X., Màrquez, L., Punyakanok, V., and Roth, D. (2002). Learning and inference for clause identification. In *EMCL 2002: Proceedings of the 13th European Conference on Machine Learning*, pages 35–47.

Ceauşu, A. (2006). Maximum entropy tiered tagging. In *ESSLLI Student Session*.

Chang, C.-C. and Lin, C.-J. (2001). LIBSVM: a library for Support Vector Machines (version 2.31).

Charniak, E. (2000). A maximum-entropy-inspired parser. In *NAACL 2000: Proceedings of the First Conference of the North American Chapter of the Association for Computational Linguistics*, pages 132–139.

Charniak, E. and Johnson, M. (2005). Course-to-fine n-best parsing and MaxEnt discriminative reranking . In *ACL 2005: Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, pages 173–180.

Chen, S. and Rosenfeld, R. (1999). A Gaussian Prior for Smoothing Maximum Entropy Models. Technical Report CMU-CS-99-108, Carnegie-Mellon University, School of Computer Science.

Chrupała, G. (2006). Simple data-driven context-sensitive lemmatization. *Procesamiento del Lenguaje Natural*, (37):121–127.

Chrupała, G., Dinu, G., and van Genabith, J. (2008). Learning morphology with Morfette. In *LREC 2008: Proceedings of The Sixth International Conference on Language Resources and Evaluation*. To appear.

Chrupała, G., Stroppa, N., van Genabith, J., and Dinu, G. (2007). Better training for function labeling. In *RANLP 2007: Proceedings of the Conference on Recent Advances in Natural Language Processing*, pages 133–138.

Chrupała, G. and van Genabith, J. (2006a). Improving treebank-based automatic LFG induction for Spanish. In *LFG 2006: The 11th International Lexical Functional Grammar Conference*, pages 91–106.

Chrupała, G. and van Genabith, J. (2006b). Using machine-learning to assign function labels to parser output for Spanish. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 136–143.

Civit, M. (2000). Guía para la anotación morfosintáctica del corpus CLiC-TALP, X-TRACT Working Paper. Technical report. Available at `http://clic.fil.ub.es/personal/civit/PUBLICA/guia_morfol.ps`.

Civit, M. (2004). Guía para la anotación de las funciones sintácticas de Cast3LB. Technical Report 3LB-WP 03-02. Available at `http://clic.fil.ub.es/personal/civit/PUBLICA/funcions.pdf`.

Civit, M., Bufí, N., and Valverde, P. (2004). Building Cat3LB: a treebank for Catalan. In *Proceedings of the SALTMIL Workshop at LREC 2004*, pages 48–51.

Civit, M. and Martí, M. A. (2004). Building Cast3LB: A Spanish treebank. *Research on Language and Computation*, 2(4):549–574.

Clark, S. and Curran, J. R. (2004). Parsing the WSJ Using CCG and Log-Linear Models. In *ACL 2004: Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*.

Clark, S. and Hockenmaier, J. (2002). Evaluating a Wide-Coverage CCG Parser. In *Proceedings of the LREC 2002 Beyond Parseval Workshop*, pages 60–66.

Collins, M. (1997). Three Generative, Lexicalized Models for Statistical Parsing. In *ACL 1997: Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, pages 16–23.

Collins, M. (1999). *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, Philadelphia, PA.

Collins, M. (2000). Discriminative Reranking for Natural Language Parsing. In *ICML 2000: Proceedings of the 17th International Conference on Machine Learning*, pages 175–182.

Collins, M. (2002). Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In *EMNLP 2002: Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing*, pages 1–8.

Collins, M. and Koo, T. (2005). Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–69.

Collins, M. and Roark, B. (2004). Incremental parsing with the perceptron algorithm. In *ACL 2004: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, pages 111–119.

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27.

Cowan, B. and Collins, M. (2005). Morphology and reranking for the statistical parsing of spanish. In *HLT-EMNLP 2005: Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 795–802.

Crammer, K. and Singer, Y. (2001). On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2(5):265–292.

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

Crouch, R., Kaplan, R. M., King, T. H., and Riezler, S. (2002). A comparison of evaluation metrics for a broad-coverage stochastic parser. In *Proceedings of the LREC 2002 Beyond Parseval Workshop*, pages 67–74.

Daelemans, W. and van den Bosch, A. (2005). *Memory-Based Language Processing*. Cambridge University Press.

Daelemans, W., Zavrel, J., van der Sloot, K., and van den Bosch, A. (2004). TiMBL: Tilburg Memory Based Learner, version 5.1, Reference Guide. Technical report. Available from `http://ilk.uvt.nl/downloads/pub/papers/ilk0402.pdf`.

Dalrymple, M. (2001). *Lexical functional grammar.* Academic Press, San Diego, USA.

Daumé III, H. (2006). *Practical Structured Learning Techniques for Natural Language Processing.* PhD thesis, University of Southern California, Los Angeles, CA.

Džeroski, S. and Erjavec, T. (1997). Induction of Slovene nominal paradigms. In *ILP 1997: Proceedings of the 7th International Workshop on Inductive Logic Programming*, pages 141–148.

Ellis, N. C., O'Dochartaigh, C., Hicks, W., Morgan, M., and Laporte, N. (2001). Cronfa Electroneg o Gymraeg (CEG): A 1 million word lexical database and frequency count for Welsh. On-line. Available at: `http://www.bangor.ac.uk/ar/cb/ceg.php.en`.

Erjavec, T. (2002). The IJS-ELAN Slovene-English parallel corpus. *International Journal of Corpus Linguistics*, 7(1):1–20.

Erjavec, T. (2004). MULTEXT-East version 3: Multilingual morphosyntactic specifications, lexicons and corpora. In *LREC 2004: Proceedings of the Fourth International Conference on Language Resources and Evaluation*, pages 1535–1538.

Erjavec, T. and Džeroski, S. (2004). Machine learning of morphosyntactic structure: Lemmatizing unknown Slovene words. *Applied Artificial Intelligence*, 18(1):17–41.

Fix, E. and Hodges, J. (1951). Discriminatory analysis, nonparametric discrimination. Technical report, USAF School of Aviation Medicine.

Freund, Y. and Schapire, R. E. (1998). Large margin classification using the perceptron algorithm. In *COLT 1998: Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pages 209–217.

Gabbard, R., Kulick, S., and Marcus, M. (2006). Fully parsing the Penn Treebank. In *HLT-NAACL 2006: Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 184–191.

Gildea, D. and Hockenmaier, J. (2003). Identifying semantic roles using Combinatory Categorial Grammar. In *EMNLP 2003: Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing*, pages 57–64.

Guo, Y., Wang, H., and van Genabith, J. (2007). Recovering non-local dependencies for Chinese. In *EMNLP/CoNLL 2007: Proceedings of the 2007 Joint Meeting of the Conference on Empirical Methods in Natural Language Processing and the Conference on Computational Natural Language Learning*, pages 257–266.

Habash, N. and Rambow, O. (2005). Arabic tokenization, part-of-speech tagging and morphological disambiguation in one fell swoop. *ACL 2005: Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, pages 573–580.

Hajič, J. (2000). Morphological tagging: data vs. dictionaries. In *NAACL 2000: Proceedings of the First Conference of the North American Chapter of the Association for Computational Linguistics*, pages 94–101.

Hajič, J. and Hladká, B. (1998). Tagging inflective languages: prediction of morphological categories for a rich, structured tagset. In *COLING-ACL 1998: Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and the 17th International Conference on Computational Linguistics*, pages 483–490.

Hakkani-Tür, D., Oflazer, K., and Tür, G. (2002). Statistical Morphological Disambiguation for Agglutinative Languages. *Computers and the Humanities*, 36(4):381–410.

Han, C.-H. and Palmer, M. (2004). A morphological tagger for Korean: Statistical tagging combined with corpus-based morphological rule application. *Machine Translation*, 18(4):275–297.

Henderson, J. (2003). Inducing history representations for broad coverage statistical parsing. In *NAACL 2003: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pages 24–31.

Hirschberg, D. S. (1977). Algorithms for the longest common subsequence problem. *Journal of the ACM*, 24(4):664–675.

Jijkoun, V. and de Rijke, M. (2004). Enriching the output of a parser using memory-

based learning. In *ACL 2004: Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, pages 311–318.

Johnson, M. (2001). A simple pattern-matching algorithm for recovering empty nodes and their antecedents. In *ACL 2002: Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 136–143.

Johnson, M., Geman, S., Canon, S., Chi, Z., and Riezler, S. (1999). Estimators for stochastic "Unification-Based" grammars. In *ACL 1999: Proceedings of the 37th Annual Conference of the Association for Computational Linguistics*, pages 535–541.

Jurafsky, D. and Martin, J. H. (2008). *Speech and Language Processing*. Prentice Hall, 2 edition.

Kaplan, R., Riezler, S., King, T., Maxwell, J., Vasserman, A., and Crouch, R. (2004). Speed and accuracy in shallow and deep stochastic parsing. In *HLT-NAACL 2004: Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*.

Kurohashi, S. and Nagao, M. (2003). Building a Japanese parsed corpus while improving the parsing system. In Abeillé, A., editor, *Treebanks: Building and Using Parsed Corpora*, pages 249–260. Kluwer Academic Publishers, Dordrecht.

Lafferty, J. D., McCallum, A., and Pereira, F. C. N. (2001). Conditional Random Fields: Probabilistic models for segmenting and labeling sequence data. In *ICML 2001: Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289.

Lavrač, N. and Džeroski, S. (1994). *Inductive logic programming*. E. Horwood New York.

Le, Z. (2004). *Maximum Entropy Modeling Toolkit for Python and C++*. Available at `http://homepages.inf.ed.ac.uk/s0450736/software/maxent/manual.pdf`.

Levy, R. and Manning, C. (2003). Is it harder to parse Chinese, or the Chinese treebank? In *ACL 2003: Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 439–446.

Levy, R. and Manning, C. (2004). Deep dependencies from context-free statistical parsers: correcting the surface dependency approximation. In *ACL 2004: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, pages 328–335.

Ling, C. X. (1994). Learning the Past Tense of English Verbs: The Symbolic Pattern Associator vs. Connectionist Models. *Journal of Artificial Intelligence Research*, 1:209–229.

Magerman, D. (1994). *Natural Language Parsing as Statistical Pattern Recognition*. PhD thesis, Department of Computer Science, Stanford University, CA.

Manandhar, S., Džeroski, S., and Erjavec, T. (1998). Learning multilingual morphology with CLOG. In *Proceedings of the 8th International Conference on Inductive Logic Programming*, pages 135–144.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1994). Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Martí, M. A., Taulé, M., Bertran, M., and Màrquez, L. (2007). AnCora: Multilingual and multilevel annotated corpora. Unpublished draft available at `http://clic.ub.edu/ancora/ancora-corpus.pdf`.

Maxwell, J. T. and Kaplan, R. M. (1996). Unification-based parsers that automatically take advantage of context freeness. In *LFG 1996: Proceedings of the Lexical Functional Grammar Conference*.

McCallum, A., Freitag, D., and Pereira, F. (2000). Maximum Entropy Markov models for information extraction and segmentation. In *ICML 2000: Proceedings of the International Conference on Machine Learning*, pages 591–598.

Merlo, P. and Musillo, G. (2005). Accurate function parsing. In *HLT-EMNLP 2005: Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 620–627.

Miyao, Y., Ninomiya, T., and Tsujii, J. (2003). Probabilistic modeling of argument structures including non-local dependencies. In *RANLP 2003: Proceedings of the Conference on Recent Advances in Natural Language Processing*, pages 285–291.

Miyao, Y. and Tsujii, J. (2002). Maximum Entropy Estimation for Feature Forests. In *HLT 2002: Proceedings of Human Language Technology Conference*, pages 292–297.

Miyao, Y. and Tsujii, J. (2004). Deep linguistic analysis for the accurate identification of predicate-argument relations. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, pages 1392–1397.

Miyao, Y. and Tsujii, J. (2005). Probabilistic disambiguation models for wide-coverage HPSG parsing. In *ACL 2005: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 83–90.

Miyao, Y. and Tsujii, J. (2008). Feature forest models for probabilistic HPSG parsing. *Computational Linguistics*, 34(1):35–80.

Mooney, R. J. and Califf, M. E. (1995). Induction of first-order decision lists: Results on learning the past tense of English verbs. In *Proceedings of the 5th International Workshop on Inductive Logic Programming*, pages 145–146.

Muggleton, S. (1991). Inductive logic programming. *New Generation Computing*, 8(4):295–318.

Musillo, G. and Merlo, P. (2005). Lexical and structural biases for function parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 83–92.

Myers, E. W. (1986). An O(ND) difference algorithm and its variations. *Algorithmica*, 1(1):251–266.

Noreen, E. W. (1989). *Computer intensive methods for testing hypotheses.* A Wiley-Interscience Publication, New York.

O'Donovan, R., Burke, M., Cahill, A., van Genabith, J., and Way, A. (2004). Large-scale induction and evaluation of lexical resources from the Penn-II treebank. In *ACL*

*2004: Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, pages 367–374.

O'Donovan, R., Cahill, A., van Genabith, J., and Way, A. (2005). Automatic acquisition of Spanish LFG resources from the CAST3LB treebank. In *LFG 2005: Proceedings of the Tenth International Conference on Lexical Functional Grammar*, pages 334–352.

Oya, M. and van Genabith, J. (2007). Automatic acquisition of Lexical-Functional Grammar resources from a Japanese dependency corpus. In *PACLIC 2007: Proceedings of the 21st Pacific Asia Conference on Language, Information and Computation.*

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

Ramshaw, L. and Marcus, M. (1995). Text chunking using transformation-based learning. In *Proceedings of the Third ACL Workshop on Very Large Corpora*, pages 82–94. Cambridge MA, USA.

Ratnaparkhi, A. (1996). A maximum entropy model for part-of-speech tagging. In *EMNLP 1996: Proceedings of the 1st Conference on Empirical Methods in Natural Language Processing*, pages 133–142.

Riezler, S., King, T. H., Kaplan, R. M., Crouch, R., John T. Maxwell, I., and Johnson, M. (2001). Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *ACL 2002: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 271–278.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, (65):386–408. Reprinted in *Neurocomputing* (MIT Press, 1998).

Rosenfeld, R. (1996). Maximum entropy approach to adaptive statistical language modelling. *Computer Speech & Language*, 10(3):187–228.

Roth, D. (2001). Reasoning with classifiers. In *ECML 2001: Proceedings of the European Conference on Machine Learning*, pages 506–510.

Roth, D. and Yih, W. (2004). A linear programming formulation for global inference in natural language tasks. In *CONLL 2004: Eighth Conference on Computational Natural Language Learning*, pages 1–8.

Schluter, N. and van Genabith, J. (2007). Preparing, restructuring and augmenting a French treebank: Lexicalised parsing or coherent treebanks? In *PACLING 2007: Proceedings of The 10th Conference of the Pacific Association of Computational Linguistics*, pages 200–209.

Sha, F. and Pereira, F. (2003). Shallow parsing with Conditional Random Fields. In *NAACL 2003: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pages 134–141.

Shen, L., Sarkar, A., and Joshi, A. (2003). Using LTAG based features in parse reranking. In *EMNLP 2003: Proceedings of the ACL-03 Conference on Empirical Methods in Natural Language Processing*, pages 89–96.

Stroppa, N. and Yvon, F. (2005). An analogical learner for morphological analysis. In *CoNNL 2005: Proceedings of the 9th Conference on Computational Natural Language Learning*, pages 120–127.

Taskar, B., Guestrin, C., and Koller, D. (2004). Max-margin Markov networks. In Thrun, S., Saul, L., and Schölkopf, B., editors, *Advances in Neural Information Processing Systems*. MIT Press.

Tsochantaridis, I., Joachims, T., Hofmann, T., and Altun, Y. (2005). Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484.

Tufiş, D. (1999). Tiered tagging and combined language models classifiers. In *TSD 1999: Proceedings of the Second International Workshop on Text, Speech and Dialogue*, pages 28–33.

Tufiş, D. and Dragomirescu, L. (2004). Tiered tagging revisited. In *LREC 2004: Pro-*

*ceedings of the Fourth International Language Resources and Evaluation Conference*, pages 39–42.

van den Bosch, A. (2004). Wrapped progressive sampling search for optimizing learning algorithm parameters. In *Proceedings of the 16th Belgian-Dutch Conference on Artificial Intelligence*, pages 219–226.

van den Bosch, A. and Daelemans, W. (1999). Memory-based morphological analysis. In *ACL 1999: Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 285–292.

van der Beek, L., Bouma, G., Malouf, R., and van Noord, G. (2002). The Alpino dependency treebank. In *CLIN 2001: Computational Linguistics in the Netherlands*, pages 8–22.

Vapnik, V. (2006). *Estimation of Dependences Based on Empirical Data*. Springer.

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

Vapnik, V. N. (1998). *Statistical Learning Theory*. Wiley-Interscience, New York, NY, USA.

Vossen, P., editor (1998). *EuroWordNet: A Multilingual Database with Lexical Semantic Networks*. Springer.

Weston, J. and Watkins, C. (1999). Support vector machines for multiclass pattern recognition. In *Proceedings of the Seventh European Symposium On Artificial Neural Networks*.

White, A. and Liu, W. (1994). Bias in information-based measures in decision tree induction. *Machine Learning*, 15(3):321–329.

Xia, F. (1999). Extracting Tree Adjoining Grammars from Bracketed Corpora. In *Proceedings of the 5th Natural Language Processing Pacific Rim Symposium*, pages 398–403.

Xue, N. and Xia, F. (2000). The bracketing guidelines for the Penn Chinese treebank. Technical report, University of Pennsylvania.

Yuret, D. and Türe, F. (2006). Learning morphological disambiguation rules for Turkish. In *HLT-NAACL 2006: Proceedings of the Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 328–334.

Yvon, F. (2003). Finite-state machines solving analogies on words. Technical report, ENST.