Learning and Inference in Phrase Recognition: A Filtering-Ranking Architecture using Perceptron

Tesi Doctoral
per a optar al grau de
Doctor en Informàtica

per
Xavier Carreras Pérez

sota la direcció del doctor
Lluís Màrquez Villodre

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

Barcelona, Juliol de 2005

Abstract

This thesis takes a machine learning approach to the general problem of recognizing phrases in a sentence. This general problem instantiates in many disambiguation tasks of Natural Language Processing, such as Shallow Syntactic Parsing, Clause Identification, Named Entity Extraction or Semantic Role Labeling. In all of them, a sentence has to be segmented into many labeled phrases, that form a sequence or hierarchy of phrases.

We study such problems under a unifying framework for recognizing a structure of phrases in a sentence. The methodology combines learning and inference techniques, and consists of decomposing the problem of recognizing a complex structure into many intermediate steps or local decisions, each recognizing a simple piece of the structure. Such decisions are solved with supervised learning, by training functions from data that predict outcomes for the decisions. Inference combines the outcomes of learning functions applied to different parts of a given sentence to build a phrase structure for it.

In a phrase recognition architecture, two issues are of special interest: efficiency and learnability. By decomposing the general problem into lower-level problems, both properties can be achieved. On the one hand, the type of local decisions we deal with are simple enough to be learned with reasonable accuracy. On the other hand, the type of representations of a decomposed structure allows efficient inference algorithms that build a structure by combining many different pieces.

Within this framework, we discuss a modeling choice related to the granularity at which the problem is decomposed: word-level or phrase-level. Word-level decompositions, used commonly in shallow parsing tasks, reduce the phrase recognition problem into a sequential tagging problem, for which many techniques exist. In this thesis, we concentrate on phrase-based models, that put learning in a context more expressive than word-based models, at the cost of increasing the complexity of learning and inference processes. We describe incremental inference strategies for both type of models that go from greedy to robust, with respect to their ability to trade off local predictions to form a coherent phase structure. Finally, we describe discriminative learning strategies for training the components of a phrase recognition architecture. We focus on large margin learning algorithms, and discuss the difference between training each predictor locally and independently, and training globally and dependently all predictors.

As a main contribution, we propose a phrase recognition architecture that we name *Filtering-Ranking*. Here, a filtering component is first used to substantially reduce the space of possible solutions, by applying learning at word level. On the top of it, a ranking component applies learning at phrase level to discriminate the best structure among those that pass the filter. We also present a global learning algorithm based on Perceptron, that we name *FR-Perceptron*. The algorithm trains the filters and rankers of the architecture at the same time, and benefits from the interactions that these predictors exhibit within the architecture.

We present exhaustive experimentation with FR-Perceptron in the context of several partial parsing problems proposed in the CoNLL Shared Tasks. We provide empirical evidence that our global learning algorithm is advantageous over a local learning strategy. Furthermore, the results we obtain are among the best results published on the tasks, and in some cases they improve the state-of-the-art.

Agraïments

Em sento afortunat de que en Lluís Màrquez m'hagi dirigit aquest treball. Fer recerca sota la seva direcció ha sigut entusiasmador, apassionant i divertit, com un bon joc. En els moments borrosos m'ha donat consells i suggerències que, per la via fàcil i simple, han il·luminat els següents passos. Li dono les gràcies més profundes per la confiança, paciència i treball que ha dipositat en mi.

Hi ha altres persones directament implicades en la qüestió. En Jordi Turmo i l'Horacio Rodríguez —el savi— em van iniciar en l'apassionant món del Processament del Llenguatge Natural, i en Lluís Padró i el German Rigau m'han donat moltes mostres de com cal fer les coses.

A més, tinc la sort d'haver caigut en un grup de recerca on, sense cap dubte, hi regna el bon ambient. Pel seu companyerisme, dono les gràcies a la gent del Grup de Recerca en Processament del Llenguatge Natural i col·laboradors habituals. A part dels Lluïsos, l'Horacio, en Jordi i el German, aquí ve una colla de bons col·legues: Alicia Ageno, Laura Alonso, Jordi Atserias, Victòria Arranz, Manu Bertran, Neus Català, Bernardino Casas, Núria Castell, Irene Castellón, Isaac Chao, Grzegorz Chrupała, Montserrat Civit, Pere R. Comas, Eli Comelles, Montse Cuadros, Jordi Daudé, Gerard Escudero, Javi Farreres, David Farwell, Dani Ferrés, Maria Fuentes, Marta Gatius, Jesús Giménez, Edgar González, Meritxell González, Àngels Hernández, Patrik Lambert, Toni Martí, Muntsa Padró, F.J. Raya, Francis Real, Enrique Romero, Mihai Surdeanu, Lluís Villarejo, *et al.* (per si les mosques).

També vull donar agraïments a les persones del Departament de LSI de la UPC que me n'han fet un bon lloc de treball, molt especialment a la gent del laboratori de càlcul i a les secretàries, tots ells tant generosos.

En la part personal, vull donar les gràcies a la meva família, pel seu amor i suport. L'esforç que he posat en aquest treball va dedicat a ells.

Acknowledgements

In my doctoral research, I had the amazing opportunity of visiting research centers abroad. In 2002, I visited Dan Roth at the University of Illinois at Urbana-Champaign. An important part of my research started there, and many ideas in this thesis result from discussions with Dan and his students, especially Vasin Punyakanok. More recently, during the spring of 2004 I could visit Michael

Collins at the Massachussets Institute of Technology, in the Boston area. Discussing machine learning topics with him was great, and gave me new interesting perspectives that are very valuable to me. I am very grateful to them and to the people I met there, so friendly.

I express my gratitude to the co-authors of the papers I've been involved in during this thesis research. Here's the list: Lluís Màrquez, Lluís Padró, Jorge Castro, Enrique Romero, Vasin Punyakanok, Dan Roth, Grzegorz Chrupała, Adrià de Gispert, Toni Martí, Montse Arévalo and Maria José Simon. Also, I thank the two anonymous referees of this thesis for helpful and encouraging comments on a preliminary version of this document.

From 2001 to 2004, the author was supported by a pre-doctoral grant from *DURSI*, the *Ministry of Universities, Research and Information Society* of the *Catalan Government* (grant reference: *2001FI 00663*).

The research in this thesis was developed in the context of several research projects and other initiatives, funded by the following institutions: the *Catalan Ministry of Universities, Research and Information Society* (Research Group of Quality, 2001 SGR 00254), the *Spanish Ministry of Science and Technology* (Hermes, TIC2000-0335-C03-02; Petra, TIC2000-1735-C02-02; Aliado, TIC2002-04447-C02), and the *European Union Commission* (NAMIC, IST-1999-12392; Meaning, IST-2001-34460; Chil, IP 506909; PASCAL Network of Excellence, IST-2002-506778).

Contents

Chapter 1

Introduction

Natural Language (NL) processing and understanding requires the general task of revealing the language structures of text, at morphologic, syntactic and semantic levels [Jurafsky and Martin, 2000]. Broadly speaking, analyzing a sentence consists of segmenting it into words, recognize syntactic elements and their relationships within a structure, and induce a syntactico-semantic representation of the many concepts the sentence may express. In this line, a number of central NL tasks consist of recognizing some type of structure which represents linguistic elements of the analysis and their relations. Many language applications, such as Question-Answering, Information Extraction, Machine Translation, Summarization, etc. build on general tools which perform such tasks.

A major problem is that natural language is ambiguous at all levels. Thus, the main concern of language analyzers is how to disambiguate the correct structure of a sentence from all possible structures for it.

This thesis focuses on language structures based on phrases, at the syntactico-semantic level. In a sentence, phrases group words that together represent a linguistic element of some nature. For example, the sentence *the cat eats fresh fish* can be segmented into three basic syntactic phrases: a noun phrase (*the cat*), a verb phrase (*eats*), and another noun phrase (*fresh fish*). In the NL domain, several fundamental tasks consist of recognizing phrases of some type. Examples of these tasks include Named Entity Extraction, Syntactic Analysis, or Semantic Role Labeling, among others. Generally, in all cases the task consists of recognizing a set of phrases in a sentence, organized in a structure. The difference between tasks concerns the nature of the phrases and the relations they exhibit in a sentence.

The aim of this thesis is to develop machine learning systems for these problems under a unified framework. Machine learning techniques are used to predict whether some words of a sentence form a phrase or not. In doing so, it is assumed that there exist some patterns that naturally explain the phrases that are to be recognized –which certainly exist in natural language. Thus, a crucial point is to choose a representation of linguistic elements that gives the learning components enough expressivity to capture the natural patterns.

An important aspect of our framework are the structural properties of the problems we are dealing with, with two main issues. The first is computational. Structurally, a sentence is a sequence of words. The number of possible phrases in a sentence grows quadratically with the length of the sentence, and the space of possible phrase structures is of exponential size. In this scenario, an algorithmic scheme is required to efficiently explore the sentence to recognize phrases.

The second issue concerns the relations that different phrases in a structure exhibit. For example, in the syntactic domain it is commonly observed that a noun phrase is followed by a verb phrase, and that the former is the agent of the predicate expressed in the verb phrase, as in *the cat eats*. A more complex level of dependencies is found when phrases appear under a recursive pattern in a structure. For example, syntactic clauses appear in a sentence an arbitrary number of times through coordination, subordination and other patterns. Thus, an important aspect is to put learning in a context in which these dependencies can be captured, so that a prediction can benefit from them.

These observations motivate approaches which combine learning and exploration processes, each supporting the other. In this thesis, we refer to this general approach as *learning and inference* paradigm. The role of learning is to capture statistics from training data that serve to accurately and robustly make predictions about which words form phrases on unseen data. The role of inference is to efficiently explore the sentence so as to recognize phrases with learning predictors, ensuring that the recognized phrases form a coherent phrase structure for the sentence. The research presented in this thesis proposes techniques that follow this paradigm, in the context of several phrase recognition problems that arise in natural language disambiguation.

The rest of the chapter is organized as follows. The next section describes the phrase recognition problem and identifies some goals of this thesis. Then, Section 1.2 presents this thesis contents, starting by over-viewing the major contributions of this research, and then outlining the organization of the thesis into chapters.

1.1 The Phrase Recognition Problem

In this section we describe the phrase recognition problem. We first motivate the type of problems that we regard as phrase recognition, and introduce the family of techniques used to solve them. Next, in subsection 1.1.2 we present a number of natural language tasks that are addressed in this thesis, all of them belonging to the CoNLL Shared Task Series, and all of them being particular instances of the the general problem we address. In subsection 1.1.3, we provide a formal definition of the phrase recognition problem and the standard method to evaluate a system. After that, in subsection 1.1.4 we discuss some general characteristics of the tasks in NL that instantiate the phrase recognition problem. Finally, we introduce the machine learning approach for these type of tasks.

1.1.1 From Full to Partial Syntactic Parsing

During the 90's, a number of statistical approaches applied to natural language demonstrated that, to some extent, non-restricted automatic syntactic analysis of language is possible. Part of the success can be attributed to the availability of large-scale treebanks, that is, resources containing a large collection of sentences annotated with syntactic structure (e.g., Brown Corpus and Penn Treebank [Marcus et al., 1993], 3LB [Palomar et al., 2003], PropBank [Palmer et al., 2005]). The annotation scheme of such resources, defining which linguistic elements and syntactic relations and dependencies are annotated, is usually fine-grained. To some extent, the annotations unambiguously and consistently represent all behaviors that syntactic elements of natural language exhibit in the treebank. With this, many statistical methods have been developed for the *full parsing* task, the problem of associating sentences with their syntactic tree. Broadly, probabilistic full parsers consist of fine-grained parametric language models which associate probabilities to trees, relying on a grammar of the language that is assumed to generate such syntactic trees and sentences. The parameters of the model are estimated from data, in particular from the large collection of sentence-tree pairs of the treebank. To date, the best state-of-the-art statistical full parsers working on the *standard* Wall Street Journal (WSJ) data achieve results slightly below 90% of accuracy [Collins, 1999; Charniak, 2000].

Current natural language applications make use of much less analysis than the one annotated in treebanks. For instance, a question-answering application —as a complex application of interest in the NL community– makes use of many different types of analysis, each of which is much simpler than a full syntactic tree. Typical modules are syntactic part-of-speech taggers and chunkers, named entity recognizers, semantic disambiguators, analyzers of the relations expressed in verbs, etc. The question-answering system makes use of the output of each one to make inferences that answer a posed question.

Therefore, in recent years there has been a lot of interest on the design of learning systems which perform only a partial analysis of the sentence [Abney,

1991, 1996b; Hammerton et al., 2002]. In contrast with tasks that reveal a full analysis, partial tasks are characterized by much coarser annotation schemes. The corresponding analysis does not identify and disambiguate all linguistic elements of a sentence, but only those specific to the task. Thus, partial tasks are much simpler than full tasks (for example, the cardinality of the output space is much lower) and, consequently, the resulting systems are much simpler and faster than those performing a full analysis of the sentence. This property has facilitated the use of general machine learning for partial tasks. Here, instead of designing a fine-grained, linguistically-motivated statistical model, the approach relies on reducing the problem to classification subproblems, and learn powerful classifiers that assign labels to the linguistic elements of interest. To predict such labels, classifiers operate in high-dimensional spaces of propositional features that represent linguistic elements and their relations, possibly with many different sources of information. This flexibility in representation constitutes an attractive property of classification-based techniques, in contrast to traditional probabilistic approaches that impose strong conditions on the type of representations.

So, the general problem of analyzing natural language presents a trade-off on how to approach it. Focusing on syntactic analysis, one can think of a single complex task which disambiguates all syntactic elements of a sentence. On the other hand, one can break a full analysis into many intermediate steps or layers, define and learn a partial analyzer for each layer, and chain partial analyzers in a pipeline to obtain incrementally the full analysis. An advantage of the full approach is that the dependencies between syntactic elements at different levels can be exploited within the same task. However, it is generally accepted that a complete syntactic disambiguation cannot be performed isolatedly of other linguistic knowledge, such as the semantics of syntactic constituents and of their relations, but rather at the same time. In contrast, the partial approach allows to build the structure at each layer taking into account different annotations of earlier analysis, and incorporating knowledge resources specific to each layer (e.g., dictionaries, semantic taxonomies, etc.). A crucial aspect, then, is how to break down the whole analysis process into many partial tasks, and which dependencies are established between tasks. To succeed, each layer of partial analysis must be simple enough to be resolved independently of higher-levels of analysis, and accurate enough to permit further processing depending on it. If this is possible, the general scheme for language analysis can be resolved with a simple pipeline which chains the partial processors from lower to higher levels, each benefiting from the structures recognized at lower levels. In practice, it is not possible to resolve any natural language task with no error, and it has been shown in many experimentations that errors committed in early layers substantially degrade the performance of later layers. Thus, to overcome error propagation through a pipeline of processors, one can think of a flexible architecture for analyzing language. Here, several different types of analyzers detect partial structures on a sentence, and assign meanings to them. Above them, a reasoning process induces a global analysis for the sentence, taking into account dependencies and constraints over different types and levels of partial structures

to form a global one.

In the context of syntactic parsing, up to date full approaches show superiority over partial approaches. First, a full analysis is typically richer than the result of several layers of partial processing, since partial approaches often consider only the main elements of a syntactic structure, and discard intermediate constituents of the structure to simplify the problem. Second, in terms of results, full parsers perform better in the context of general syntactic analysis. So, a full analysis is richer and more accurate than a few layers of partial analysis. However, empirical evidence has been found in favor of machine learning approaches detecting the basic syntactic chunks of a sentence. Their performance, focusing only on the basic analysis, is better than that of statistical full parsers [Li and Roth, 2001]. Nowadays, the study of learning algorithms for higher levels of analysis constitutes an active direction of research in natural language tagging and parsing, and machine learning. A complementary research direction concerns the semantic analysis of a sentence. The complexity of the semantic meanings of words and, more importantly, the relations between them, constitutes a challenging aspect of language understanding in which, clearly, learning takes an important role. Currently, semantic analyzers are not yet accurate and stable enough to take part in a global interdependent analysis.

This thesis takes a machine learning approach to resolve partial analysis tasks, at different levels of analysis. The aim is to study mechanisms for recognizing phrase structures in a sentence. We concentrate on techniques for solving a single problem. In particular, in Chapter 3 we propose a framework to recognize phrases in a sentence, based on learning and incremental inference techniques. We study strategies that go from simple to complex, and experiment with partial parsing tasks of different characteristics. Out of the scope of this thesis are the mechanisms to integrate many different partial analyzers into a global analysis process. However, the tasks we experiment with can be pipelined to resolve a coarse-grained syntactic analysis of the sentence. Next section explains these tasks.

1.1.2 Phrase Recognition in CoNLL Shared Task Series

Since 1999, the Conference on Natural Language Learning (CoNLL) organizes each year a Shared Task [1]. Each edition proposes a natural language problem to be solved with learning techniques, with the aim of comparing different learning approaches in a common problem setting. For a problem, training and test data derived from existing corpora are prepared and made public to develop systems. Then, systems can be compared by contrasting their approach with the evaluation measures they obtain on the test set.

An attractive aspect of the CoNLL Shared Task series is that the addressed problems can be thought as a decomposition of the basic syntactico-semantic analysis of language into many separate tasks, each building on the analysis achieved in the previous task. Thus, by chaining in a pipeline a processor of

[1] See CoNLL website at: `http://www.cnts.ua.ac.be/conll`

each task, one obtains a basic language analyzer that is useful for building language applications.

In this thesis, we concentrate on the following tasks for the experimental evaluation. Chapter 5 is devoted to describe systems these tasks. It is assumed that the pipeline starts with a part-of-speech tagging process, that assigns to each word of a given sentence its part-of-speech tag. Then, three phrase recognition tasks are applied, layered in the following order:

Syntactic Chunking

Shallow Syntactic Parsing is the problem of recognizing syntactic base phrases in a sentence. Base phrases are syntactic phrases which do not contain any other phrase in within, that is, they are non-recursive. Such base phrases are also known as *chunks*, so the task is often referred to as Syntactic Chunking. Syntactically, words within a chunk act as a unit in a syntactic relation of a sentence. The chunking of a sentence (i.e., the set of chunks) constitutes the first syntactic generalization of the sentence. In the following examples, chunks are represented in brackets, with the label of the chunk as a subscript of the closing bracket:

(The San Francisco Examiner)$_{\texttt{NP}}$ (issued)$_{\texttt{VP}}$ (a special edition)$_{\texttt{NP}}$ (around)$_{\texttt{PP}}$ (noon)$_{\texttt{NP}}$ (yesterday)$_{\texttt{NP}}$ (that)$_{\texttt{NP}}$ (was filled)$_{\texttt{VP}}$ (entirely)$_{\texttt{ADVP}}$ (with)$_{\texttt{PP}}$ (earthquake news and information)$_{\texttt{NP}}$.

Here, `NP` stands for noun phrase, `VP` for verb phrase, `PP` for prepositional phrase, and `ADVP` for adverbial phrase.

This task was addressed in CoNLL-2000 [Tjong Kim Sang and Buchholz, 2000]. Eleven types of syntactic chunks were considered, and data was derived from the Wall Street Journal portion of the Penn Treebank (WSJ) [Marcus et al., 1993]. The task generalizes the one of the 1999 edition, concerning the recognition of basic noun phrases, that in turn reproduced the problem proposed by Ramshaw and Marcus [1995].

The particular setting of this task, together with the public datasets, has become a benchmark for evaluation of shallow parsing systems. Within the machine learning community, this task is also an attractive problem for testing sequential learning algorithms.

Clause Identification

This task consists of recognizing the syntactic clauses of a sentence. Clauses can be roughly defined as sequences of words with a verb and a subject, possibly implicit. The main challenge of this task is that clauses form a hierarchical structure in a sentence, thus the task cannot be directly approached with sequential learning techniques, as in chunking. In a sentence, the hierarchical clause structure forms the skeleton of the full syntactic tree. In the example below, clauses are enclosed within brackets:

(The San Francisco Examiner issued a special edition around noon yesterday
(that (was filled entirely with earthquake news and information)$_{S}$)$_{S}$.)$_{S}$

This task was addressed in CoNLL-2001 [Tjong Kim Sang and Déjean, 2001], and data was derived from the WSJ corpus, as in syntactic chunking. All clauses were labeled the same type (S), so in the data there is no differentiation among different types of clauses (e.g., main clause, relative clause, etc.).

Semantic Role Labeling

In this problem, the goal is to recognize the arguments of the predicates in a sentence. Arguments are phrases in the sentence which play a relation with a predicate of the sentence. This relation is called a *semantic role*. The PropBank corpus [Palmer et al., 2005] defines what semantic roles accepts each verb in English, and annotates the predicate-argument relations of the verbs in the WSJ corpus with their semantic role. In a sentence, each verb has a set of labeled arguments. For the example sentence, the arguments of the verb "issue" are:

(The San Francisco Examiner)$_{A0}$ (issued)$_{V}$ (a special edition)$_{A1}$ (around noon)$_{TMP}$ (yesterday)$_{TMP}$ (that was filled entirely with earthquake news and information)$_{C-A1}$.

According to PropBank, A0 is the issuer of the verb predicate "issue", and A1 is the thing issued (in the example, this argument is broken down into two pieces, the second annotated as C-A1). V stands for the verb, and TMP is a general modifier expressing a temporal relation. For the verb "fill" the arguments are:

The San Francisco Examiner issued (a special edition)$_{A1}$ around noon yesterday (that)$_{R-A1}$ was (filled)$_{V}$ (entirely)$_{MNR}$ with (earthquake news and information)$_{A2}$.

A1 is the destination, R-A1 is a referent to A1, and A2 is the theme. MNR stands for a general modifier expressing a manner relation.

In this thesis, we frame all these tasks under the general problem of recognizing phrases in a sentence. The difference between tasks relies then on the nature of the phrases, the scheme of phrase labels and the type of structures that phrases form in a sentence (i.e., shallow phrases or phrase hierarchies).

Apart from these tasks, another task that can be directly casted as a phrase recognition problem is Named Entity Extraction. This task consists of recognizing the named entities in a sentence and classify them according to their type. For instance, in the above example sentence, "San Francisco Examiner" is a named entity, denoting an organization. This problem was addressed in two editions of the CoNLL Shared Task, namely for Spanish and Dutch in 2002 [Tjong Kim Sang, 2002a], and for English and German in 2003 [Tjong Kim Sang and De Meulder, 2003]. While throughout the thesis we mention this problem, for conciseness we do not present experiments on it.

1.1.3 Problem Definition and Evaluation

In this section we give a definition of the general problem discussed in this thesis, in the context of a supervised learning problem, and we describe the standard evaluation method of a system.

Let $\mathcal{X}$ be the space of sentences of a language. Let $\mathcal{Y}$ be the space of phrase structures for sentences. In particular, an element $y \in \mathcal{Y}$ is a *set* of phrases that constitute a well-formed phrase structure (i.e., the phrases in y form a sequence or a hierarchy of phrases).

Given a training set $S = \{(x^1, y^1), \ldots, (x^m, y^m)\}$, where x^i are sentences in $\mathcal{X}$ and y^i are phrase structures in $\mathcal{Y}$, the goal is to learn a function $\mathrm{R} : \mathcal{X} \to \mathcal{Y}$ which correctly recognizes phrases on unseen sentences.

In order to evaluate a phrase recognition system, the standard measures for recognition tasks are used: precision, recall and $\mathrm{F}_{\beta=1}$. Precision (p) is the proportion of recognized phrases that are correct. Recall (r) is the proportion of solution phrases that are correctly recognized. Finally, their harmonic mean, $\mathrm{F}_{\beta=1}$, is taken as the standard performance measure to compare systems.

We consider that a predicted phrase is correctly recognized if it matches exactly a correct phrase. That is, the words that the phrase spans and the phrase label have to be correct. This criterion makes the evaluation of a system strict. In contrast, there are softer criterions that consider that a phrase is correctly recognized if the important words of it match the important words of a correct phrase (e.g., in standard evaluation of full parsers, punctuation tokens are not considered).

Recall that a solution y is a set, so the intersection with another solution y' gives the set of phrases that are in both sets, with exact matching. Let $|\cdot|$ be the number of elements in a set. The computation of the evaluation measures in a test set $\{(x^i, y^i)\}_1^l$ can be expressed as follows:

$$p = \frac{\sum_{i=1}^{l} |y^i \cap \mathrm{R}(x^i)|}{\sum_{i=1}^{l} |\mathrm{R}(x^i)|} \qquad r = \frac{\sum_{i=1}^{l} |y^i \cap \mathrm{R}(x^i)|}{\sum_{i=1}^{l} |y^i|} \qquad \mathrm{F}_{\beta=1} = \frac{2\,p\,r}{p + r}$$

1.1.4 Generalities

In general, the goal of the tasks we focus on is to recognize a set of phrases in a sentence, organized in a structure. Here we characterize different tasks of phrase recognition.

Origin

Phrase recognition tasks arise in the NLP area with two different motivations. First, the syntactic analysis problem can be approached as a full task or be broken down into many separate tasks, each focusing on a part of the general structure. Full parsing can be seen as a phrase recognition task, in which phrases correspond to nodes of the full syntactic tree. However, this thesis focuses on partial parsing tasks, where the structures are much simpler than syntactic trees.

Typical partial syntactic tasks are noun phrase recognition, shallow parsing, prepositional phrase attachment or clause identification. In a sentence, each of the phrase structures detected in these tasks corresponds to a piece of the full syntactic tree.

Second, language applications often look for specific information in text, and consequently define tasks in which the goal is to recognize phrase structures containing such information. For instance, current text-retrieval, question-answering or summarization systems make use of named entity extractors, which detect and classify phrases in text denoting a named entity. In the most general case, a named entity can be a complex structure formed of basic named entities and other words. For example, *Goldsboro Express* is a named entity denoting a particular train related to a place named *Goldsboro*. Or, *The Complete Prestige Recordings of John Coltrane* denotes the title of a music compilation, and there are in within two related named entities, namely a company (*Prestige*) and a person (*John Coltrane*). Regardless of their complexity, named entity structures appear in specific parts of a sentence, and constitute only a piece of the global structure representing syntactico-semantically a sentence.

Complexity of Phrase Structures

Different tasks can be grouped according to properties of the phrase structures that are to be recognized. Structurally, the most simple task is to recognize basic phrases that do not contain other phrases in within. This problem is usually known as *shallow analysis* or *chunking*, since non-recursive phrases are often called *chunks*. The resulting phrase structure in a sentence is a sequence of phrases. Tasks in this category include syntactic chunking, or named entity extraction in its simpler and most typical setting —where the goal is to recognize maximal named entities, without considering their internal structure. Also, recognizing arguments of a predicate in a sentence can be modeled as a chunking task, where each chunk is an argument that plays some semantic role in the predicate.

A more complex level of tasks is found when phrases admit embedding, usually under a recursive pattern. In this case, related phrases form trees and, in general, the phrase structure of a sentence is a forest of phrase hierarchies. A particular case is that of full parsing, where the structure is the syntactic tree of the sentence. Other hierarchical phrase recognition tasks are syntactic clause identification, or general versions of noun phrase recognition –recognizing structures of base noun phrases together with their modifiers, as in *(a cup (of coffee))*– or named entity extraction –unraveling the internal structure of named entities, as in *((United Nations) Headquarters)*.

Sparseness of Phrase Structures

A different dimension for characterizing phrase structures –and their corresponding problems– is the sparseness of phrases in a sentence, that is, whether most of the words are covered by the phrase structure or not. For example, in shallow

syntactic parsing the task may be to recognize a particular base chunk, such as noun or verb phrases, resulting in specific phrase structures which do not cover completely the level of analysis. On the other side, the task may be to recognize all base syntactic chunks of the sentence. In this case, except for punctuation and other functional tokens, all words in the sentence belong to some base chunk, and the resulting phrase structure completes the shallow level of syntactic analysis. Recognizing all chunks together will allow to take advantage of the dependencies between chunks and to ensure coherence of the shallow analysis. In the syntactic domain, clause hierarchies or predicate-argument constructs are also partial syntactic structures which complete a piece of the global sentence analysis. In contrast, named entities appear in text sparsely. A sentence might contain no named entities or a few of them. And, in the context of a sentence, two named entities cannot be related without looking at some other piece of the sentence analysis, such as a verb expressing a relation between them, as in *Peter goes to Japan.*

1.1.5 The Machine Learning Approach

Simple phrase recognition tasks, where the goal is to recognize non-recursive phrases with little dependencies among them, facilitate the use of general machine learning algorithms. The approach consists of reducing the global task of recognizing phrases to local classification subproblems, and learn a classifier for each of the subproblems using machine learning. Typical local subproblems include learning whether a word *opens*, *closes*, or is *inside* a phrase of some type. Recognizing phrases, then, consists of exploring the words of a sentence and apply classifications to them, while combining the outcome of the classifications to build the phrases of the sentence. Usually, not all combinations of the classifiers' outcomes are possible. For example, a word cannot be inside a certain phrase if such phrase has not been opened in some preceding word. Thus, the exploration strategy not only visits words of a sentence to classify them, but also checks that the classifications form coherent phrases in a sentence. In simple phrase recognition tasks, many state-of-the-art systems rely on powerful learning algorithms which provide accurate classifiers. Then, the exploration strategy is usually greedy at ensuring coherence, in that a classification in a certain word is assumed to be correct, and then future classifications which would produce incoherence are not considered.

As the complexity of phrase structures increases, and it is accepted that learned predictors are not error-free, it seems adequate to design exploration strategies that are more robust to local prediction errors than simple greedy strategies. In general, the learning components of a phrase recognition system compute different local predictions at different parts of the sentence. It is always possible to express that a combination of predictions forms a coherent phrase structure with constraints over different predictions. Thus, given a phrase recognition task decomposed into many local learning problems, and given learned local predictors, recognizing phrases in a sentence consists of finding the best combination of local predictions that satisfies the constraints. This way, the

general approach is not only an exploration of a sentence to compute predictions, but rather a powerful inference process to find the best global coherent assignment given the local predictions [Roth and Yih, 2004]. The phrase recognition architecture can be divided into two layers: first, a learning layer, where local predictions are computed independently; then, an inference layer, which selects the best phrase structure considering both the local predictions and the structural properties of the solution, expressed in the constraints.

Still, in some tasks the phrases in a structure exhibit many dependencies between them. For example, in the hierarchy of syntactic clauses of a sentence, top-level clauses group low-level simple clauses to form a complex sentence. In these domains, thus, it seems desirable that some predictions take into account part of the phrase structure that is being recognized. This idea motivates a third paradigm for phrase recognition task, that of combining learning and inference in both directions. While in the previous paradigm inference was on the top of learners, here learners will depend on inference, and vice-versa, since a certain prediction in a part of the sentence will benefit of the structure recognized in other parts of the sentence. As it will be shown, the type of learning here is global, in that it considers the global inference strategy to train the prediction functions. Actually, this paradigm corresponds to the standard approach in generative probabilistic models, such as HMM or PCFG. In these approaches, a certain model makes use of several interdependent probabilistic distributions estimated from data, corresponding to learners. Inference consists of finding the solution with highest probability, and, when searching for it, dependencies of the model are taken into account. However, discriminative learning seems to be advantageous over generative learning, because it offers flexibility in terms of feature representations and theoretical guarantees on generalization. In this direction, the paradigm of globally training learners with respect to inference has been recently proposed in the literature for discriminative learners [Collins, 2004].

This thesis studies phrase recognition architectures, focusing on discriminative learning methods, inference strategies, and their interaction. We present different architectures and experiment with them in the context of the tasks presented in Section 1.1.2.

1.2 This Thesis

1.2.1 Contributions

The research work presented in this thesis can be framed in the field of Natural Language Learning, which is the field devoted to study approaches that place machine learning as the central mechanism to understand and process natural language. As the name suggests, the field of Natural Language Learning borrows theories and problems from the area of Natural Language Processing, while concentrates on the results and techniques of the Machine Learning community to make them applicable to the Natural Language domain.

The specific field we focus on is that of designing supervised learning systems for complex problems that arise when analyzing natural language. In particular, we study and propose methods that learn to recognize phrases in a sentence. Below, we overview the three major contributions of this thesis.

A Framework for Phrase Recognition

This thesis develops a framework for phrase recognition problems oriented to approaches based on discriminative learning, and inference mechanisms to deal with structured data. The starting point is the formalization of the general problem. In particular, the goal in a phrase recognition problem is to recognize in a sentence a structure of labeled phrases. Here, the basic element is a phrase, defined as a sequence of contiguous words which, together, assume some functionality in the sentence analysis. Such functionality is expressed by the label of the phrase. Then, the framework considers two types of phrase structures, which result in two particularizations of the general problem. The first is that of chunking, where phrases form a non-overlapping sequential structure in the sentence. Instances of this problem include shallow syntactic analysis, named entity extraction, or recognition of the semantic roles of a sentence predicate. The second problem, more general, consists of recognizing hierarchical phrase structures, that is, structures of phrases that form a tree or a forest in the sentence. This problem instantiates on tasks such as the identification of syntactic clauses, more general versions of the above-mentioned problems, and, in general, syntactic parsing tasks.

The framework develops techniques for phrase recognition architectures that make use of two central mechanisms: *learning* and *inference*. Learning serves to predict which words of a sentence form phrases. Inference combines the outcome of learning predictors to form a structure of phrases for the sentence, with two main concerns: efficiency and coherence of the phrase structure. In the framework, a phrase recognition architecture consists of three main components:

- *Model.* We discuss models that decompose the global problem into many decisions, at two different granularities, which are directly solved with learning techniques. The first is at word-level, which is the traditional approach of phrase recognition systems. The second is at phrase level, that puts learning in a context more expressive than word-level models, but also computationally more expensive.

- *Inference strategy.* We comment on inference algorithms for phrase recognition models at word and phrase levels. In all cases, the type of inference is incremental, that is, as the sentence is processed in some order, the plausible solutions are built, incremented and contrasted between them, so that when the final of the sentence is reached the best solution is ready. We present instantiations of this general processing scheme that go from approximate to exact strategies according to the model's optimality criterion.

- *Learning strategy.* We discuss discriminative strategies for learning the predictors of a phrase recognition architecture. We focus on large margin algorithms for training linear separators, and discuss the difference between learning each predictor locally, or learning globally all predictors of the architecture.

Perceptron Learning for a Filtering-Ranking Architecture

As a main contribution, this thesis proposes a phrase recognition architecture, which we name *Filtering-Ranking*, and a global learning strategy for it based on Perceptron. This work is published in [Carreras et al., 2005] and, in turn, builds on previous work [Carreras and Màrquez, 2003a,b; Carreras et al., 2002b].

The Filtering-Ranking architecture faces the general problem of recognizing hierarchies of phrases, and can be particularized to look for sequential phrase structures. The strategy for recognizing phrases in a sentence can be sketched as follows. Given a sentence, learning is first applied at word level to identify phrase candidates of the solution. Then, learning is applied at phrase level to score phrase candidates and discriminate among competing ones. These two layers of learning predictors are controlled by the inference strategy, that explores the sentence and computes predictions at different parts of it, with the goal of building the optimal phrase structure for the sentence, according to the predictions and other coherency criteria. In the architecture, the phrase–level layer of predictors allows to deal with complete candidate phrases and partially constructed phrase structures. An advantage of working at this level is that rich and informed feature representations can be used to exploit structural properties of the examples, possibly through the use of kernel functions. However, a disadvantage of working with high level constructs is that the number of candidates to explore increases and the search space may become prohibitively expensive to explore. For this reason, the word–level layer of our architecture plays a crucial role by filtering out non plausible phrase candidates and thus reducing the search space in which the high–level layer operates.

We then propose the FR-Perceptron learning algorithm to globally train the learning functions of the system as linear separators, all in one go. The learning strategy is a generalization of Perceptron that follows [Collins, 2002] in two main aspects. First, it works online at sentence level: when visiting a sentence, the functions being trained are first used to recognize the set of phrases in it, and then updated according to the correctness of the solution. Second, the type of update is conservative since it operates only on the mistakes of the global solution. The proposed algorithm extends Collins' in that not only it trains a score function for ranking potential output labelings, but also trains the filtering component which provides candidates to the ranker. The extension is in terms of the feedback rule for the Perceptron, which reflects to each individual function its committed errors when recognizing a set of phrases. As a result, the learned functions are automatically approximated to behave as word filters and phrase rankers, and thus, become adapted to the recognition strategy.

Regarding the analysis of the algorithm a convergence proof is presented.

Regarding the empirical study, we provide extensive experimentation on two relevant problems of the Partial Parsing domain, namely base Chunking and Clause Identification. Moreover, we incorporate, in the experimental architecture, the results of Freund and Schapire [1999] on Voted Perceptrons to produce robust predictions and allow the use of kernel functions. The performance achieved by the presented learning architecture is comparable to the state–of–the–art systems on base chunking and substantially better in the recognition of clauses. Besides, the experiments presented help understanding the behavior of the different components of the architecture and give evidence about its advantages compared to other standard alternative learning strategies.

State-of-the-art results in CoNLL Shared Tasks

A central goal of this thesis research is to build competitive systems for the yearly-organized Shared Tasks of the CoNLL conference, which we view as phrase recognition problems. Namely, these tasks include Shallow Syntactic Parsing [Tjong Kim Sang and Buchholz, 2000], Clause Identification [Tjong Kim Sang and Déjean, 2001], Named Entity Extraction [Tjong Kim Sang, 2002a; Tjong Kim Sang and De Meulder, 2003] and Semantic Role Labeling [Carreras and Màrquez, 2004]. The series of CoNLL Shared Tasks are an interesting and motivating initiative of the Natural Language Learning community, since they propose relevant, real-sized Natural Language problems, and establish experimental settings that permit fair comparisons of systems implementing different learning approaches. In particular, all systems are developed under the same task specification and training data, and evaluated with standard performance measures on the same test data. Thus, systems can be ranked according to their final performance, and conclusions can be drawn about what elements influenced the good or bad performance of different approaches. In general, building a competitive system concerns issues related to: (1) the type of architecture and model for recognizing phrases; (2) the learning algorithm used to train the learning functions of the architecture; (3) the type of features for representing the data; and (4) practical techniques and tricks to develop and tune a learning system for a real-sized natural language problem.

Under an unified framework, we have developed systems that perform among the best at each edition, and thus can be considered in the state–of–the–art. For Shallow Syntactic Parsing, the task with more evaluated systems, our system obtains results that are very close to the top-performing system. On Clause Identification, our system is substantially better than any other system. For Named Entity Extraction (on 4 languages) and Semantic Role Labeling, our systems are among the top-performing ones.

1.2.2 Organization

The rest of the thesis is organized in the following chapters.

- **Chapter 2. A Review of Supervised Natural Language Learning** We review the major approaches to natural language disambiguation prob-

lems with supervised machine learning, focusing on discriminative learning and large margin learning algorithms, as they are the techniques used in this thesis. We also review the relevant literature on shallow and partial parsing systems.

- **Chapter 3. A Framework for Phrase Recognition**
  This chapter describes the framework to design phrase recognition systems. We comment on the major components of a learning-based phrase recognition architecture, namely the model, the inference algorithm, and the learning strategy. We propose a variety of options for each one, and discuss instantiations of phrase recognition architectures that go from simple to more complex.

- **Chapter 4. A Filtering-Ranking Learning Architecture**
  This chapter presents the main contribution of this thesis, namely a filtering-ranking learning architecture for general phrase recognition problems. First, we describe the architecture. Then, we propose a Perceptron algorithm to train it globally, with analysis on its convergence, and extensive empirical experimentation that evinces its good performance.

- **Chapter 5. A Pipeline of Systems for Syntactic-Semantic Parsing**
  In this chapter, we develop Natural Language analyzers for three CoNLL Shared Tasks, using the filtering-ranking learning architecture. We contrast our systems, in terms of results, with other systems in the literature developed for the same tasks, and show that the performance of our systems is among the best in the state-of-the-art.

- **Chapter 6. Conclusion**
  This chapter gives conclusions and outlines future research directions.

How to read this document. The main contribution of this thesis is found in Chapter 4, that presents the filtering-ranking learning architecture and the Perceptron-based global algorithm for it. This chapter is mostly self-contained and, in fact, it is essentially the main part of the article published in [Carreras et al., 2005]. Readers who are familiar with machine learning and partial parsing techniques should be able to read straightforwardly the chapter.

For readers not familiar with these topics, Chapter 2 introduces machine learning techniques for disambiguating language. We contextualize and discuss the type of learning that we use. We also comment on partial parsing techniques and systems of the literature that are relevant to the area and influence our work.

Chapter 3 presents a framework for partial parsing and other phrase recognition problems. Essentially, the phrase recognition problem is studied in formal way, identifying the major computational difficulties and discussing a family of techniques to solve it with discriminative learning. The filtering-ranking architecture proposed later in Chapter 4 can be thought as a particular choice within this family of phrase recognition methods.

Chapter 5 provides a practical description about building partial parsers for natural language processing, for three different tasks. We comment on final adaptations of the general model, features used by the learners, training processes, and evaluation results.

Chapter 2

A Review of Supervised Natural Language Learning

This chapter reviews the main concepts and approaches of the areas of Machine Learning (ML) and Natural Language (NL) processing that serve as basis of the work presented in this thesis.

The chapter is organized in three sections. The first section introduces supervised machine learning techniques which apply to natural language disambiguation problems, focusing on approaches for dealing with structured domains, as is the case for the problems discussed in this thesis. The second section reviews the specific type of learning algorithms that are used, a family known as *large margin* methods. We introduce the main theoretical concepts that serve as starting point for the design and justification of the methods, and review the main algorithms. Finally, the last section reviews techniques and systems for shallow and partial parsing, which is the specific type of NL problems we concentrate on.

2.1 Learning to Disambiguate in Natural Language

In this section we review machine learning techniques for natural language disambiguation tasks, focusing on supervised methods which apply to phrase recognition tasks and other more general problems.

A natural language disambiguation problem can be thought as inducing a function $\mathrm{h} : \mathcal{X} \rightarrow \mathcal{Y}$. In the problems discussed in this thesis (and in most NL disambiguation problems), $\mathcal{X}$ is the space of all sentences of language. But in general $\mathcal{X}$ could be a space representing words, sentences, documents, or other textual domains. As for the output space, we can differentiate between classification problems and structured problems. In classification problems, $\mathcal{Y}$ is fixed set of labels. For example, $\mathcal{Y}$ might be the set of possible part-of-speech

tags of a language, and the associated task is to determine the tag in $\mathcal{Y}$ that a given word assumes in a sentence. Other examples of pure NL classification tasks include Word Sense Disambiguation —consisting of assigning the appropriate semantic label for a word in a sentence—, Spelling Correction —the task of fixing spelling errors, by classifying a certain string into the possible legitimate words of the language— or Text Categorization —where the goal is to identify the topic categories of a document.

In structured problems, the elements of the output space $\mathcal{Y}$ are structures such as sequences, trees or, in general, graphs. Such structures are labeled, in that each of the nodes forming the structure have a label that denotes the nature of the constituent represented by the node. In most of these problems, $\mathcal{X}$ is the space of sentences, so the problem is to induce a function mapping sequences of words (i.e., sentences) to labeled sequences or trees. For example, let $\mathcal{T}$ be the set of part-of-speech tags of a language, and $\mathcal{Y}$ be the space of sequences formed with tags in $\mathcal{T}$. A sequential classification problem is that of determining the sequence of part-of-speech tags for each word of a given sentence —which corresponds to the standard problem known as Part-of-Speech (PoS) tagging. Another instance of structured-output problems is that of syntactic parsing, where the goal is to disambiguate the syntactic tree of a sentence (i.e., $\mathcal{Y}$ the space of all possible syntactic trees). Closely related, in the problems discussed in this thesis $\mathcal{Y}$ is the set of all possible phrase structures for sentences, either sequential or hierarchical. Structured output domains exhibit two particular properties. First, the cardinality of the output spaces is of exponential size with respect to the length of an input sentence. Second, two different structures of $\mathcal{Y}$ can be very similar, while others can be radically different. For example, two trees can differ only in one node's label. These properties are crucial for the design of appropriate disambiguation learning techniques.

In supervised learning, the learning protocol is the following. It is assumed the existence of a distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$ that is unknown, but generates a collection of training examples, independently and identically distributed.[1] Each example is of the form (x_i, y_i), with $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$.

In this framework, the task of a *learning algorithm* is to induce, from the training examples, a *hypothesis* h which predicts accurately the correct values $y \in \mathcal{Y}$ for an instance $x \in \mathcal{X}$. In order to evaluate the quality of a learned function, an *error function* or *loss* function, denoted $L(y, \hat{y})$, measures the *error* or cost of proposing the output $\hat{y}$ when the correct output is y. In classification, the most common error function is the *0-1 loss*, which assigns a penalty of 1 when the output is not correct ($y \neq \hat{y}$) and 0 when correct ($y = \hat{y}$). When dealing with structures, however, it is more appropriate to consider error functions based on the number of different nodes of the correct and predicted structures, which yields error functions closely related to precision and recall measures.

[1] Note that since $\mathcal{D}$ is a joint distribution, it is possible to observe examples with different y's for the same x. While we assume that only one value of $\mathcal{Y}$ is correct for each instance of $\mathcal{X}$, a joint distribution of examples is useful to study a number of interesting issues out of the scope of this thesis. For instance, noisy sources of examples, or inherently ambiguous instances that have several correct solutions —the later specially suitable for processing language.

Conceptually, the ultimate goal is to learn the best hypothesis, that minimizing the true error over the complete distribution $\mathcal{D}$. However, $\mathcal{D}$ is unknown, so, in practice, a separate test set, assumed also to be drawn from $\mathcal{D}$, is used to measure the error of the learned function on the test examples.

In the following sections, we review three main families of machine learning techniques for resolving NL ambiguities. First, we comment on probabilistic learning, where hypothesis are probabilistic distributions of the data, trying to estimate the densities of $\mathcal{D}$ in some way. The focus is to introduce the approach of generative models. Then, we review discriminative learning, the type of learning that, up to date, provides the most powerful and general learning algorithms. Finally, we discuss the learning and inference paradigm, a family of methods for disambiguating in structured domains. As it will be clear, the reviewed methods do not belong exclusively to the family of methods in which we present them.

2.1.1 Probabilistic Learning

Probabilistic methods associate probabilities to input-output pairs of language [Charniak, 1993]. If $x \in \mathcal{X}$ is an instance, and $\mathcal{Y}(x)$ are the possible output values for x, probabilistic methods choose the value of $\mathcal{Y}(x)$ maximizing the probability of it given x, that is $\mathrm{h}(x) = \arg\max_{\mathcal{Y}(x)} p(y|x)$. Generative models estimate a joint probability distribution $p(x, y)$ of the data, and are later used for disambiguation via the Bayes rule to compute $p(y|x)$. Modeling alternatives exist to estimate directly the conditional distribution $p(y|x)$.

Generative Models

Generative models define a probability distribution of the data parametrized by a stochastic generation mechanism of the data. Under this mechanism, a pair (x, y) is uniquely defined by its *derivation*, that is, the sequence of steps or decisions taken for generating (x, y) in some canonical order. The probability of a pair (x, y) is then the probability of chaining the corresponding steps in the derivation. The model defines which steps are possible, and makes assumptions on which elements of the x and y variables take part in a generation step. With this, a probability distribution can be defined parametrically by associating parameters to the decisions of a derivation. Broadly, parameters determine the probability of each possible instantiation of each decision (i.e., an event), and are estimated from data. In particular, each parameter is the conditional probability the part of (x, y) generated in the step given a factored history of the generation.

We now describe two generative models, Hidden Markov Models and Probabilistic Context Free Grammars, which are standard models for tagging and parsing problems.

Hidden Markov Models (HMM). A hidden Markov model is a standard probabilistic model for language modeling and sequential disambiguation tasks

(see [Rabiner, 1989] for a classic tutorial oriented to speech recognition). In a tagging task, $\mathbf{x}$ is a sequence of tokens $x_1 \cdots x_n$ and $\mathbf{y}$ is a sequence of tags $y_1 \cdots y_n$ attached to $\mathbf{x}$. An HMM is a stochastic finite-state automaton, generating sequences of tokens. To generate a sequence $\mathbf{x}$, the process first chooses an initial state, from which it emits the first token x_1. Then, it transitions to a new state, emitting the second token, and so on until a designated final state is reached. Three probability distributions are involved in this process, namely the initial state distribution $p_0(s)$, and two conditional distributions, the state transition probability $p(s'|s)$ from state s to s', and the emission probability $p(x|s)$ of a token x_i from a state s. There is a correspondence between the states of the automaton and the possible labels, so knowing the states which generated $\mathbf{x}$ determines $\mathbf{y}$. A HMM defines a joint probability distribution of input-output sequences, $p(\mathbf{x}, \mathbf{y})$, by means of the transition, emission and initial state probabilities, estimated from data.

Thus, the assumptions in HMM dictate that a token depends only on the state that generated it, and that a state is determined only by its previous state in the process. For example, a trigram HMM part-of-speech tagger defines a state for each combination of two part-of-speech tags $\langle t^-, t\rangle$, where t is the tag of the current word and t^- is the tag of the previous word. The state-transition distribution controls the probability of assigning the tag t^+ to the following word when the previous and current word are tagged with t^- and t respectively, with a parameter for every trigram $\langle t^-, t, t^+\rangle$ —hence the name of the tagger. The emission distribution controls the probability of generating a word w given the current and previous tags, with a parameter for every triple of two tags and a word $\langle t^-, t, w\rangle$. To disambiguate the labels of a given sequence x with a trained HMM, the Viterbi algorithm finds the most probable sequence of states (associated to labels) that generates $\mathbf{x}$.

Probabilistic Context-Free Grammars (PCFG). PCFG models associate probabilities to sentence-tree pairs (x, y). The sequence of decisions of the generative mechanism is defined as the expansions of the grammar rules in a derivation of the tree, under a fixed parsing strategy (e.g., top-down, left-most derivations). Each rule has an associated probability, and the probability of a (x, y) pair is the product of rule probabilities in the derivation of (x, y). In doing so, the assumption in PCFG models is that generating the right-hand side of a rule depends only on the left-hand side of the rule, that is, the non-terminal being expanded. Given a PCFG, finding the most probable tree for a sentence can be done in polynomial time with, e.g., the CKY algorithm [Charniak, 1993]. Modern statistical parsers, such as those with best results on the WSJ corpus [Collins, 1999; Charniak, 2000], are PCFG models where the rules are lexicalized. That is, the non-terminals of the grammar are augmented with lexical items and other information which can be considered linguistically relevant to the generation process. For example, Collins [1999] proposes head-based PCFG models, where the non-terminals are augmented with the lexical head governing the non-terminal constituent. The expansion of a rule (i.e., generating the right

part given the left part) is broken down into a number of steps. Broadly, first the head non-terminal is chosen; then the head modifiers, to the left and to the right of it, are selected sequentially, given the lexical head of the rule and other information generated during the rule expansion. Lexicalization leads to fine-grained PCFG models with a very large number of rules, with the advantage that now the model is much more expressive, since the model parameters capture dependencies between the lexical information of the left and right parts of a rule.

Maximum Likelihood Estimates

A standard technique in parametric joint probabilistic models is to set the parameters to optimize the joint likelihood of the data. In this framework, the goal is to estimate the unknown distribution $\mathcal{D}$ that generates the training and test examples. A crucial assumption of the framework is that $\mathcal{D}$ belongs to the class of distributions considered by the probabilistic model. That is, there is an assignment of weights α to the model parameters for which the probability distribution is $\mathcal{D}$, $p_\alpha(x, y) = \mathcal{D}$. Under this assumption, learning translates to finding such optimal assignment. Bayesian learning suggests to fit parameters to maximize the joint likelihood of the training data, $\alpha_{\text{jMLE}} = \arg\max_\alpha \sum_i p_\alpha(x_i, y, i)$. It is well-known that under the assumption that $\mathcal{D}$ belongs to the class of parametric distributions of the model, maximum-likelihood values make the estimated distribution converge to the natural distribution $\mathcal{D}$, as the amount of data goes to infinity.

It turns out that in history-based processes, such as HMM, PCFG and other models, the maximum-likelihood estimates correspond to simple relative frequencies of the decisions observed in the derivations of the training collection. In particular, the weight for a parameter associated to an observation o given a history h, $p(o|h)$, is the fraction of training decisions with history h where o is observed. Furthermore, there exist many smoothing techniques to robustly estimate parameters in the case data sparseness, that is, when some observations occur infrequently in the training data and produce unreliable frequencies (e.g., see the empirical study of Chen and Goodman [1996] and their references).

A major limitation of maximum-likelihood estimation techniques (pointed out by many authors [Johnson et al., 1999; Collins, 2004]), which apply either to joint or conditional estimation, comes from the assumption that the true distribution $\mathcal{D}$ is actually an instantiation of the class of probability distributions parametrized by the model. Typical language disambiguation models, such as generative models, make strong independence assumptions which clearly do not hold in the natural data. In other words, the representation of input-output pairs adopted by the model (in terms of features, in direct relation with the model parameters) is very poor. This limitation conflicts with the ability of linguists to describe language units in many different ways and sources of information. Enriching a generative model to include more features in the representation is a complex task, since the steps of a derivation have to be redefined to generate the included features at some point. Indeed, features not

tied to the derivations of solutions add dependencies in the model, and usually make maximum likelihood estimation intractable.

Furthermore, apart from restricting to the model class of distributions, the theoretical guarantees that maximum likelihood estimation converges to the true distribution are asymptotic as the training size goes to infinity. Collins [2004] gives more formal arguments to these limitations, and discusses the advantages of distribution-free learning methods, that, as the name indicates, do not make assumptions on the underlying real distribution of the data. Also, the theory behind them provides strategies that benefit generalization to unseen data given that the amount of actual training data is limited. Section 2.2.1 looks closer to distribution-free learning.

Alternatives exist to model the conditional probability distribution of the data, and choose the parameters that maximize the conditional likelihood of the data [Johnson, 2001; Klein and Manning, 2002]. Here, estimation methods, such as probability-based decision trees [Magerman, 1996], Maximum Entropy estimation [Ratnaparkhi, 1998; McCallum et al., 2000] or Conditional Random Fields [Lafferty et al., 2001], do not make strong independence assumptions, and thus allow flexible representations where arbitrary features can be coded. These estimation techniques are reviewed in the following section, as general discriminative learning techniques.

2.1.2 Direct, Discriminative Learning

Discriminative learning solves the disambiguation learning problem directly, learning a map from inputs in $\mathcal{X}$ to outputs in $\mathcal{Y}$. This contrasts with generative or informative learning (see [Rubinstein and Hastie, 1997] for a comparison between generative and discriminative paradigms), where the approach is to learn the distribution generating $\mathcal{X}$ and $\mathcal{Y}$, and then use it to disambiguate by examining the value of $\mathcal{Y}$ that most probably was generated with a given input $x \in \mathcal{X}$. In probabilistic learning, generative methods estimate a joint probability distribution of the data, while discriminative methods estimate directly a conditional probability distribution. The family of discriminative algorithms we focus on moves away from estimating a (conditional) probability distribution. Rather, the algorithms concentrate on learning the boundaries of the possible disambiguation outputs.

The main concern in discriminative techniques is to learn accurate hypotheses, considering the error function as the ultimate practical measure to optimize when testing the hypothesis. In this framework, training data serves to discover regularities in the data that might be useful in predicting the output variables given the input variables. Since training and test examples are assumed to be drawn from the same distribution, one hopes that a good predictor learned from training data will perform also well on test data.

In the 90s, many general machine learning methods of the Artificial Intelligence (AI) community [Mitchell, 1997] were successfully applied to natural language problems that can be framed into classification tasks [Roth, 1998; Daelemans et al., 1997]. Typical problems here include Word Sense Disam-

biguation, Spelling Correction, Text Categorization, etc. In all cases, there is a pre-specified scheme of classes or categories, and the problem consists of assigning the most suitable class to a given instance (i.e., a word in context, sentence, or document).

Broadly, the design of a learning system for such problems involves issues concerning the representation of the input instances, the type of policy adopted to predict the most suitable class, and the algorithm that learns a predictor given a collection of training examples.

In terms of representation, the most common standard way is to represent an instance with a collection of propositional features. These features capture properties, attributes or characteristics of the instance, and their design depends both on the domain of the data –in our case natural language– and the specific task we are willing to resolve. From this point of view, x is represented as a vector of features. Each coordinate in the vector corresponds to one feature, and the value of the coordinate is the value of the feature for the instance. In general, a feature set has to be expressive enough to characterize properly any of the instances of the input space $\mathcal{X}$, so that it is possible to learn a predictor that discriminates correctly the output value of an instance $x \in \mathcal{X}$ by looking only at the feature values of x.

As for the type of predictors and learners, there exist many options. Some of the most popular classical AI methods include Decision Tree Learning, Decision Lists, Memory-Based Learning and Nearest Neighbor classifiers, or Transformation Based Learning, among others. In the probabilistic family, the two most common techniques are Maximum Entropy Estimation and Naive Bayes[2] Finally, Artificial Neural Networks are also a common choice in NLP, although as we explain below, recent research has populated the use of simpler learners based on linear separators that optimize margins, resulting in algorithms such as Perceptron, Winnow, Support Vector Machines, or related algorithms such as AdaBoost.

With so many choices, it is desirable to understand the properties of the different problems and algorithms. From a learning point of view, natural language problems are characterized by feature spaces of very large dimensionality. For example, to achieve reasonable expressiveness, it is common to consider one or several dimensions for *each* word of the language, or even for combinations of words. In such spaces, the representation of instances is very sparse, that is, most of the features in the vector have a null value. For example, a representation where features are binary indicators usually have tens of thousands of dimensions, but only a few hundred of them have a non-trivial value in a particular instance. On the other side, the size of a training collection varies from problem to problem, and goes from just hundreds of examples, to tens or even hundreds of thousands of examples. With these properties, a learning algorithm for NLP has to be computationally efficient at dealing with big training sets of

[2]Naive Bayes is a generative classifier, not discriminative. In practice, however, it is successfully used with representations in which features are not independent, and thus violate the assumptions of the learner. In this sense, the learner is used in a direct discriminative way, rather than aiming at estimating a generative model of the data. See Roth [1999].

huge dimensionality, where instances occur sparsely. It has to be also robust at dealing with irrelevant features.

It turns out that many of the cited learning algorithms, although being designed from different ideas, operate with linear decision surfaces [Roth, 1998, 1999]. That is, the shape of a learned model is a linear separator in the representation space that discriminates between the positive and the negative instances of the target concept. In other words, with linear separators the problem of not producing errors translates to the problem of separating well between positive and negative instances. Therefore, the difference between algorithms learning a linear separator relies on the criteria for choosing a particular separator among all possible separators. This observation facilitates the analysis of algorithms under a unified framework, that of learning linear separators. In section 2.2.1 we overview some theoretical concepts and results of this framework. Importantly, these results translate into clues about the relevant quantities to optimize during learning and, in turn, motivate algorithms such as Support Vector Machines, AdaBoost, or the well-known Perceptron, the latter being the core algorithm used in the systems that this thesis proposes.

2.1.3 Learning and Inference Paradigm

So far, we have over-viewed two families of supervised learning methods: generative learning –willing to explain the data– and discriminative learning –focusing directly on differentiating what is correct and what is not.

The generative approach offers well-understood techniques for problems with structured output –which is the case of the problem of recognizing phrase structures in sentences or, more generally, tagging and parsing problems. Examples of such techniques are HMM or PCFG, discussed in Section 2.1.1, as well as other more general *graphical* models [Jordan, 2004]. However, as it has been pointed out, a major limitation of generative models comes from the fact that features are tied to the derivations assumed to generate the data, which makes it difficult to incorporate arbitrary and dependent features for supporting predictions.

On the other hand, discriminative learning imposes no conditions on the features representing learning instances, thus offers flexibility at combining different, possibly dependent knowledge sources and characteristics of the data. However, discriminative learning techniques are designed and analyzed as general algorithms for classification problems. That is, the learning algorithms of the machine learning community are developed to discriminate among a limited, fixed number of classes. When the nature of the problem has a structured, exponential-sized output space, such as the sequential or hierarchical solutions of tagging and parsing problems, these algorithms shall not be used "out of the box", for several related reasons. First, because most of the discriminative methods rely on enumerating exhaustively all possible output values during training and prediction. Obviously, this is not feasible for exponential-sized output spaces. Even when the learning method is not sensitive to the number of output classes (such as, e.g., nearest neighbor learners), the learner will

suffer from data sparseness, and will not be able to generalize. But, most importantly, treating each possible output structure as an atomic class does not reflect appropriately the nature of the structures: two different structures may have a lot of substructure in common, and differ only in specific parts. Thus, a structured-output space has to be compactly modeled, in a way that shared substructure is factored in the model.

For these reasons, there is a need to combine the strength of discriminative learning at arbitrarily representing instances with the strength of generative learning at compactly representing solutions with factored models.

We refer to *learning and inference* as the paradigm of to study learning techniques that apply to complex domains where: (1) a *global* complex prediction is decomposed into many *local* simple predictions, and; (2) there are dependencies and constraints that influence what combinations of local predictions form a correct global prediction. As a simple example, consider the Part-of-Speech (PoS) tagging problem, where the goal is to assign the correct PoS tag to each word of a given sentence. Here, the global solution is the sequence of PoS tags of a sentence, which can be computed by locally predicting a tag for each word. Clearly, there exist dependencies between the tags of a sentence, such as that a determiner often precedes a noun, or that usually a sentence contains at least one verb. In such scenarios, we differentiate two processes. First, a learning-based process, with learning functions as main components, that predicts values to local parts of a sentence. Second, an inference process, with an exploration algorithm as a main component, that combines local predictions to form a global solution. When dependencies and constraints exist, the role of the inference is to trade off values predicted at different positions to obtain a globally optimal solution.

In fact, traditional generative models make use of such learning and inference processes, the former to estimate the distribution generating the data, and the later used to disambiguate the structure of a given sentence. As pointed out, though, discriminatory models are currently preferred, and their design and development constitutes nowadays an active research direction in areas that deal with complex data, such as natural language processing, information retrieval, computer vision or computational biology.

In the literature, Dietterich [2002] reviews a specific scenario of learning and inference, that of supervised sequential learning, or tagging. The general problem consists of assigning labels to the components of a sequence, such as in PoS tagging. Below we describe in more detail families of inference algorithms and the role of learning in such systems.

Chained Classifiers. A simple approach to apply general discriminative learning to structured domains consists of learning classifiers which assign values to local parts. To predict the global structure of a sentence, the different parts of a sentence are explored in some canonical order (e.g., from left to right in sequences) and, at each part, the classifiers are used to assign the most plausible value. In the recurrent version of the strategy –the most common one– a

prediction in one part makes use of the values assigned in previous parts, thus exploiting dependencies between neighboring output values.

Such a simple strategy is not able to trade off decisions at different points. That is, the local predictions at one part are final in the solution, with no attempt to globally optimize the outcome. Hence, the technique is merely an exploration, rather than an inference process. Despite this limitation, and maybe for its simplicity, many systems in the literature make use of this strategy, specially for problems with simple constraints and few dependencies and often combined with powerful learning algorithms that commit few errors.

Probabilistic Inference. Probabilistic learning provides models for dealing with structured domains that exploit dependencies between different parts of a structure. In such models, the global (joint or conditional) probability distribution is decomposed into several local conditional probability distributions that are estimated from data with Maximum Likelihood techniques. Once the model is learned, there exist well-known inference algorithms for disambiguating the structure of a given sentence. Several authors have proposed to estimate the local conditional probabilities with probability-based machine learning algorithms, which in general are more powerful than Maximum Likelihood estimates. Some of the algorithms found in the literature are decision tree learning [Magerman, 1996; Màrquez, 1999], maximum entropy estimation [Ratnaparkhi, 1998; McCallum et al., 2000], or Winnow-based learning [Punyakanok and Roth, 2001]. Assuming that the estimated probabilities constitute a well-formed distribution, the global model is a consistent probability distribution too, and therefore the same inference algorithms can be used. For example, in the context of HMM, Punyakanok and Roth [2001] use Winnow classifiers to learn the conditional probabilities of states given observations, $p(s|x)$. Via Bayes rule and other transformations, this probability is transformed into $p(x|s)$, which is the distribution required by the HMM model to define the global distribution $p(\mathbf{x}, \mathbf{s})$. The Viterbi algorithm can be applied to find the most probable state sequence $\mathbf{s}$ given a sentence $\mathbf{x}$. Due to the limitations in representation of a joint probability model such as HMM, both McCallum et al. [2000] and Punyakanok and Roth [2001] proposed to change the topology of the model to be a conditional model $p(\mathbf{s}|\mathbf{x})$, which allows to make use of arbitrary representations of the input sentence $\mathbf{x}$. In both works, the equations of the model are essentially the same, the difference being that McCallum et al. [2000] estimated the local distribution $p(s|\mathbf{x})$ with maximum entropy, while Punyakanok and Roth [2001] used Winnow-based learning. Other variations of Markov models with classifiers exist, which can be found in e.g., [Dietterich, 2002]. Conditional Random Fields [Lafferty et al., 2001] fall also into the category of probabilistic models making use of inference to disambiguate. Below we give more details on it.

Constraint Satisfaction Inference. From the traditional Artificial Intelligence perspective, the learning and inference problem can be framed as a constraint satisfaction problem. Here, the problem consists of making a global

assignment to a related set of individual local variables, and there are constraints that restrict what combinations of local assignments are possible. In the *soft* version of the problem, the local assignments have a cost, and there is a global optimality criterion which reflects which global assignment satisfying the constraints is better. In this context, learning serves to predict the cost of local assignments and, after appropriately modeling the constraints and the objective function for a problem, constraint satisfaction schemes can be used to infer the optimal global solution. Yet, it is well-known that the general constraint satisfaction problem is NP-hard. However, depending on the type of constraints there exist efficient polynomial-time algorithms, that provide approximate or even optimal solutions. In this direction, Padró [1997] used the relaxation labeling algorithm for PoS tagging, with the aim of combining several knowledge-based and statistical language models. Punyakanok and Roth [2001] reduced the sequential phrase recognition problem to the shortest path algorithm for directed graphs. In a more general framework, Roth and Yih [2004] propose a constraint satisfaction inference scheme based on integer linear programming, which allows general constraints that cannot be expressed in other inference paradigms such as that of probabilistic models.

Local *vs.* Global Learning. Systems based on inference with learned predictors can be categorized in two frameworks, namely *local* or *global* learning. In the local approach, each predictor is trained independently of the others, without considering the constraints and the interactions that take place in the inference process to disambiguate the structure of a sentence. In contrast, a global learning strategy trains the whole system as one. That is, all the predictors of the system are trained in one go, dependently, taking into account that a local prediction is not final, but depends on an inference process that trades off local predictions to produce an optimal global outcome. To make it clear, a direct consequence of this difference is found at the granularity of the training examples. In the global approach, the training collection consists of sentences, each accompanied with its correct structure, and the goal of the learner is to obtain an accurate hypothesis mapping sentences to structures, under some learning criteria and a global loss function. On the other hand, the local approach breaks each example into many lower-level examples. For example, a sentence/tagging pair is broken into many word/tag pairs, and the learner attempts to learn a function mapping words to tags, under some local loss function. Obviously, the global loss function is usually in more direct relation to the error function of the problem than the local loss function. In the context of sequential learning, Lafferty et al. [2001] identify a potential problem for locally learned discriminative models based on state-transition automata, that they call *label bias problem*. They claim that transitions leaving a state compete only against each other, rather than against all other transitions. This implies that inference cannot effectively trade off decisions at different positions, and may produce severe consequences for sparsely connected automata. To overcome the problem, they propose Conditional Random Fields (CRFs), a

global conditional model that estimates the probability of an entire sequence of labels given the input sequence. That is, CRFs do not decompose the global conditional probability distribution into several local, per-state distributions trained independently. In a different line, Collins [2002] proposes a Perceptron algorithm to train a global discriminative model. This technique is central to this thesis, since it is the selected learning algorithm for our systems and experiments. Next section discusses margin-based algorithms, to which Collins' Perceptron belongs.

2.2 Learning Linear Separators: A Margin Based Approach

This section reviews the family of machine learning algorithms used in the systems of this thesis, known as *large margin* methods. First we introduce the main concepts that motivate and justify the algorithms. Then, we discuss particular large margin learning algorithms for classification and structured-output problems.

2.2.1 Theoretical Aspects of Distribution Free Learning

In this section we overview some concepts and results of Computational Learning Theory (COLT), the research area devoted to analyze theoretically learning paradigms and methods. The paradigm we are interested on is *PAC learning* [Valiant, 1984; Kearns and Vazirani, 1994], that studies under which conditions it is possible to learn accurately a given concept. Under this paradigm, we focus on Statistical Learning methods, defined by Roth [1998] as those that make inductive generalizations from observed data, and then use them to make inferences with respect to previously unseen data.[3] In particular, we describe the main concepts of the Structural Risk Minimization principle of Vapnik [1998], which provides theoretical grounds for margin-based algorithms such as Support Vector Machines. The ideas and concepts of this section are described with more detail in the book of Cristianini and Shawe-Taylor [2000], in the tutorial by Burges [1998], or in the overview of Statistical Learning Theory by Vapnik [1999].

In PAC learning, the only key assumption is that there exists some unknown probability distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$. This distribution is used to generate a training set of m examples of the form $\{(x_i, y_i)\}_{i=1}^{m}$, which are independently drawn according to $\mathcal{D}$ and identically distributed. This setting includes the particular case where it is assumed the existence of a target function f that associates a fixed y for every x, that is, $y = f(x)$. Since, in general, the only assumption is the existence of the unknown distribution, methods derived under this paradigm are known as *distribution-free* methods. In this situation, the only data available to the learner is the training set.

[3] As Roth [1998] points out, this family of methods includes both probabilistic, symbolic and sub-symbolic methods, as they all make use of statistics of the training data to learn.

The learning algorithm can choose among a set of possible functions $\mathcal{H}$, called *hypothesis space*. It is common to define $\mathcal{H}$ as a class of functions parametrized by a set of adjustable parameters α, so that a particular choice of α determines a particular function $h_\alpha \in \mathcal{H}$. From this point of view, learning consists of finding a hypothesis h_α by giving value to the parameters α. In our case, $\mathcal{H}$ will be the space of hyperplanes or linear separators, with α being the normal vector uniquely identifying a hyperplane.

In order to evaluate the quality of a learned function, a *loss function*, denoted $L(y, \hat{y})$, measures the *error* or cost of proposing the output $\hat{y}$ when the correct output is y. For instance, in classification problems it is common to consider the *0-1 loss* that, for a given example (x, y) and hypothesis h, assigns a penalty of 1 when $y \neq h(x)$ and 0 otherwise. The expectation of the error of a learned hypothesis h from $\mathcal{X}$ to $\mathcal{Y}$, called the *expected loss* or *generalization error*, is defined as the mean of losses over the $\mathcal{X} \times \mathcal{Y}$ space as follows

$$Er(h) = \int L(y, h(x)) \, \mathrm{d}\mathcal{D}(x, y)$$

Note that, for 0-1 loss, the above expression corresponds to the probability of classification error. The goal of a learning algorithm is to choose the hypothesis that minimizes the expected loss, in the situation where $\mathcal{D}$ is unknown and the only available data is the training set. The *empirical loss*, or *training error*, is defined to be the mean of losses on the training set, and is computed as:

$$Er_{emp}(h) = \frac{1}{m} \sum_{i=1}^{m} L(y_i, h(x_i))$$

This quantity establishes an inductive principle for a learning algorithm, called *Empirical Risk Minimization* (ERM), consisting of choosing the hypothesis which minimizes the training error on the given training set.

The question which arises is whether low training error implies low generalization error. Thus, a fundamental problem in learning is that of *generalization*, or ability of a hypothesis to correctly classify data not in the training set. Computational Learning Theory aims to answer this question by giving upper bounds on the generalization error. Broadly, these bounds relate the deviation between the training error and the generalization error. For instance, it is possible to give conditions which ensure that by increasing the size of the training set the training error will asymptotically converge toward the generalization error. However, in practice training sets are fixed, and for realistic sizes large deviations between empirical and true error are possible. Alternatively, generalization properties can be studied through a notion of complexity of the functions in the hypothesis space, so that the generalization error is upper bounded by the training error, the size of the training set and some quantities related to the space of hypotheses. A complementary question of the theory is that of *convergence*, or how quickly a learning method minimizing the training error achieves low generalization error. The results of this theory provide a principled method for

designing learning algorithms: methods are designed to optimize the quantities that have influence on the generalization and convergence rates.

An important issue, therefore, is the characterization of hypothesis spaces. Generally, hypotheses that become too complex to achieve no error on the training set are likely to *overfit* the data, and may not have good generalization properties. When dealing with a finite hypothesis class, its complexity depends on its size. For an infinite hypothesis space –which is the case of linear separators– a measure called *VC-dimension* accounts for the complexity of the space. Roughly speaking, this measure refers to the capacity of a hypothesis space to produce a hypothesis with zero empirical error for *any* training set. With this notion of complexity, the *Structural Risk Minimization* (SRM) inductive principle of Vapnik [1998] states that the best generalization for a learning task, given training data, is achieved by assessing the right balance between good accuracy on the training data, in terms of empirical error, and low complexity of the selected hypothesis, in terms of VC-dimension. The underlying generalization bound of a hypothesis $h \in \mathcal{H}$ is, with probability $1 - \eta$,

$$Er(h) \leq Er_{emp}(h) + \sqrt{\left(\frac{d(\log(2m/d) + 1) - \log(\eta/4)}{m}\right)}$$

where m is the size of the training sample and d is the VC-dimension of $\mathcal{H}$. Note that both low VC-dimension and big training samples have a positive influence on achieving low generalization error.

Linear Separators and Margin

We focus now on the case that the hypothesis space is the set of n-dimensional linear separators, in the setting of binary classification. That is, input instances are vectors x in $I\!R^n$, and $\mathcal{Y} = \{+1, -1\}$. The space of (normalized) hyperplanes is:

$$\mathcal{H} = \{\ h_{\mathbf{w},b} \mid \mathbf{w} \in I\!R^n,\ b \in I\!R,\ \|\mathbf{w}\| = 1\ \}$$

A hyperplane in $h_{\mathbf{w},b} \in \mathcal{H}$ induces the following classification rule:

$$h_{\mathbf{w},b}(\mathbf{x}) = \text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle + b) = \begin{cases} +1 & \text{if } \langle \mathbf{w}, \mathbf{x} \rangle + b > 0 \\ -1 & \text{otherwise} \end{cases}$$

Note that, geometrically, a hyperplane $h_{\mathbf{w},b}$ is formed by the set of points $\mathbf{z}$ in the Euclidean space $I\!R^n$ that satisfy $\langle \mathbf{w}, \mathbf{z} \rangle = 0$. Thus, $h_{\mathbf{w},b}$ divides the $I\!R^n$ space into two regions or half-spaces: the +1 region, for points falling on the positive side of the hyperplane; and the -1 region, for points falling on the negative side. A hyperplane is defined to be a γ-margin separating hyperplane if it classifies instances as follows:

$$h_{\mathbf{w},b,\gamma}(\mathbf{x}) = \begin{cases} +1 & \text{if } \langle \mathbf{w}, \mathbf{x} \rangle + b \geq \gamma \\ -1 & \text{if } \langle \mathbf{w}, \mathbf{x} \rangle + b \leq -\gamma \end{cases}$$

That is, a classification is only produced when the distance from the hyperplane to $\mathbf{x}$ is greater than γ, and for points falling into the margin region $(-\gamma, \gamma)$ the classifications are undefined.

Now, it can be proved that the VC-dimension d of the set of γ-margin separating hyperplanes is bounded by the following inequality (see [Vapnik, 1999]):

$$d \leq \min\left(\frac{R^2}{\gamma^2}, n\right) + 1$$

where R is the radius of a sphere containing vectors x. That is, in general the VC-dimension of a hyperplane is $n+1$, where n is the dimensionality of the input space. However, for hyperplanes separating the input space with large margin γ the VC-dimension can be less than $n + 1$. This observation is crucial for the family of methods explicitly looking for large-margin hyperplanes, such as Support Vector Machines, since it connects the learning strategy of the methods with the theoretical grounds of the SRM principle.

2.2.2 Learning Algorithms: From Classification to Discrimination of Structures

In the 90s, the theoretical results of Statistical Learning Theory of Vapnik [1998], over-viewed in the previous section, translated into novel learning algorithms looking for separating hyperplanes with large margin, with Support Vector Machines (SVMs) as the most prominent method in the family. In this section, we describe some of these algorithms in the context of function approximation by increasing complexity of the output space: from binary classification, to multiclass classification, to discrimination of structures belonging to an exponential-sized space.

Binary Classification

Cortes and Vapnik [1995] proposed soft-margin SVMs for binary classification, the first practical learning algorithm motivated by the SRM inductive principle. Following Vapnik's theory, this algorithm looks for the optimal hyperplane, defined as the linear separator with maximal margin between the training points of the two classes. One of the key observations is that such optimal hyperplane can be expressed as a linear combination of a small amount of training points, the so called *support vectors*, with two main results. The first is that the support vectors determine the margin of the hyperplane, which has direct influence on the confidence term of generalization bounds. Importantly, the quantities bounding the error are independent of the dimensionality of the input space, which makes the method attractive for dealing with high-dimensional spaces —which is the case of natural language learning problems. The second result is that the operations for constructing and testing the optimal hyperplane admit a non-linear mapping of the input instances into a high-dimensional feature space. In such space, the learning algorithm makes use only of inner products between

an instance and the support vectors, all of them transformed. The function implicitly computing the inner product in the transformed space, chosen a priori, is known as the *kernel* function. Thus, through a kernel function, the support vector method is able to construct rich classes of decision surfaces.

The soft-margin SVM [Cortes and Vapnik, 1995] formulates this method for the general case of training data that may not be separable. In the formulation, the problem of finding the optimal hyperplane results in a constrained quadratic optimization program, in which there is one constraint for each training point that controls the prediction on that point. In this setting, the objective of the program is to maximize the margin of the hyperplane, paying a penalty for those training points which are misclassified. Thus, the soft-margin SVM follows the SRM principle since it allows to trade off the empirical error and the capacity of the classifier. In particular, by allowing misclassified training points the margin of the separating hyperplane increases. Moreover, the capacity can be varied by changing the kernel function which induces the feature space where the hyperplane is learned. Roughly speaking, more complex kernels yield better separation, and, thus, more margin and less training error. However, increasing the dimensionality of the feature space also makes the radius of the sphere containing instances larger, which has a negative influence on the confidence of the learned classifier.

In their paper, Cortes and Vapnik [1995] considered the SVM method, named there *support vector networks*, "as powerful and universal as neural networks". In fact, from the first works in the early 90s, the research related to SVMs has witnessed a tremendous impact on the machine learning community, which has populated, on the one hand, the study of large margin methods [Smola et al., 2000] and, on the other hand, the research related to kernel methods and functions [Schölkopf and Smola, 2002].

Concerning the first direction, number of algorithms have been analyzed in terms of their ability to produce large margin predictions on the training sample. For instance, Schapire et al. [1998] showed that the AdaBoost algorithm, initially designed as a voting method to "boost" the performance of weak classifiers, was particularly aggressive at inducing large margins. This explained the ability of AdaBoost to improve the test error as more base classifiers were added to the ensemble, after the training sample was perfectly learned. Also, the traditional Perceptron algorithm of Rosenblatt [1958] has been redefined as a large-margin method [Freund and Schapire, 1999; Cristianini and Shawe-Taylor, 2000]. Freund and Schapire [1999] proposed the *Voted Perceptron* algorithm, a modification of the original Perceptron that includes voting in the prediction, admits kernels, and has theoretical guarantees similar to those of the SVM method, while it is much easier and simpler. Due to its simplicity and good properties, the Voted Perceptron algorithm is used in the systems proposed in this thesis.

Multiclass Classification

While SVMs were originally designed for binary classification, there have been a number of works extending the maximum-margin strategy to classification into a fixed number of labels (e.g., Weston and Watkins [1999], Crammer and Singer [2001]).

The traditional approach for solving multiclass problems relies on the *one-versus-all* assumption, which reduces the multiclass problem to many binary classification problems. Here, a set of binary classifiers is built, each discriminating between one of the labels and the rest. Such classifiers are trained independently with a binary classification algorithm. In doing so, it is assumed that each class can be separated from the rest. To classify a given instance, a *winner-take-all* scheme is followed: the class associated to the binary classifier with *highest* prediction is selected.

In multiclass margin-based learning, the approach is more direct. As in the traditional approach, there is a hyperplane for each class, that given an instance predicts a confidence score for the instance belonging to the associated class, and the multiclass classification is produced under winner-take-all. The key point of the approach consists of a generalized definition of the margin for multiclass problems. In particular, the margin is defined as the difference between the confidence score of the correct class and the highest score of the competing incorrect classes. Note that, as in binary classification, a positive margin implies correct classification, and that the larger the margin, the more confident a classification is. On this definition of margin, Crammer and Singer [2001] formulate the problem of finding the maximal margin multiclass hypothesis. The resulting problem is a single constrained optimization problem, where for each example and class there is a constraint controlling the associated confidence value predicted by the hypothesis. The idea of approaching multiclass learning directly as a single constrained problem is not exclusive of SVMs. Har-Peled et al. [2002] describe a general framework for formulating constrained learning problems concerning a fixed number of classes, which include binary, multiclass and multilabel classification, and label-ranking problems. Extensions for these problems have been proposed for margin based algorithms such as AdaBoost [Schapire and Singer, 1999] or Perceptron [Crammer and Singer, 2003b,a].

Discrimination of Structures

Learning problems in which the input and output space are structured introduce a major challenge: the number of possible structures for an instance is of exponential cardinality with respect to the size of the instance. Throughout this chapter we have reviewed techniques to deal with these problems. The underlying models compactly represent decomposed solutions, and the whole approach relies on learning and inference processes that decide efficiently for the optimal structure of an instance, given a trained model. In this section, we comment on large margin learning approaches for such problems.

To our knowledge, Collins [2000] was the first to look at the margin of pre-

dictions for structured problems, in the context of syntactic parsing. There, the model is a ranking function that predicts a confidence score for an instance-structure pair. Then, discriminating the structure of a given instance corresponds to select the top-ranked structure, that is, the one with highest confidence score. Hence, as in the multiclass setting, the margin of an instance is the difference between the score of the correct structure and the highest score of the incorrect structures. However, the approach in that work does not deal with the exponentiality of the output space, since the motivation was to take advantage of the globality of solutions. Instead, a baseline parser was used to select a tractable set of candidate solutions that presumably contained the correct solution. Since the baseline parser already ranks the possible candidates, the overall two-stage approach is known as *re-ranking*. Collins [2000] proposed a boosting algorithm to solve the re-ranking learning problem, using his probabilistic state-of-the-art parser for proposing candidates.

Later, Collins [2002, 2004] showed that learning directly a full ranker looking at margins is possible. The algorithm he proposed is a generalization of Perceptron, and relies on properties of the ranking model. In particular, learning is possible if there exists an inference algorithm that efficiently retrieves the top-ranked structure for an instance (e.g., dynamic programming algorithms). Then, when visiting an example the Perceptron algorithm uses the inference algorithm to predict the top-ranked structure of the example. If it is correct, the ranking function is left unchanged. If it is incorrect, the appropriate parameters of the function are updated to overcome the error. In other words, Perceptron looks for a hyperplane that achieves positive margins on the training sample. Moreover, it allows to use kernels and voting schemes that help achieving large margins. The representation of solutions and the type of ranking model are crucial in this approach because they determine the existence of efficient inference algorithms, which is the key point for dealing with the exponentiality of solutions. With appropriate ones, the strategy that Perceptron follows to learn a binary classifier is also valid for discriminating structures. This algorithm has been used in the context of part-of-speech tagging and noun phrase chunking [Collins, 2002], and full parsing [Collins and Roark, 2004], in all cases yielding results comparable to the best systems of the state-of-the-art. In the following chapter, Section 3.4.2 discusses the algorithm in detail, since it is used in the systems proposed in this thesis.

Recently, a formulation for the maximal margin separating hyperplane in structured domains has been proposed [Taskar et al., 2003; Tsochantaridis et al., 2004]. Again, the resulting problem is a constrained quadratic program. Here the constraints control the loss suffered for each training instance and possible output structure. Unlike the classification formulation, which is tailored to the standard 0-1 loss, the optimization program in this setting admits general loss functions that contemplate the degree of correctness of a solution. However, the main challenge in this setting is that there is an exponential number of constraints in the program, which makes it intractable in general. To overcome this problem, Taskar et al. [2003] restrict to Markov-based hyperplanes that represent compactly the necessary parameters to discriminate the structured

space. They show that with these representations the optimization program can be expressed with a polynomial number of constraints, which makes learning possible. In the same direction, Taskar et al. [2004] extend the maximal margin formulation for PCFG models. In a different line, Tsochantaridis et al. [2004] propose a method for identifying a polynomially-sized subset of active constraints. The optimal hyperplane considering only the active constraints is guaranteed to be a close approximation of the hyperplane optimizing the program with the full set of constraints. This formulation allows arbitrary models and loss functions, as long as it is provided an efficient inference procedure for retrieving the structure with maximum loss for a given instance. The method builds on previous works on discriminative learning for sequence labeling [Altun et al., 2002, 2003].

2.3 Learning Systems in Partial Parsing

Abney [1996b] defines Partial Parsing as the range of different techniques for recovering some but not all the information contained in a traditional syntactic analysis. Partial parsing techniques were motivated in the late 80s by the success of part-of-speech tagging systems on unrestricted text, e.g. [Church, 1988]. Although, linguistically, it is well-known that part-of-speech disambiguation cannot be solved without solving the rest of the natural language disambiguation problem, reasonable accuracies where achieved by those systems with fairly simple statistical techniques. Therefore, the question was whether, by extending such techniques, more layers of partial analysis could be efficiently solved with reasonable accuracy and robustness. Initial partial parsing systems concentrated at recovering just the structure that can be reliably recovered with a minimal set of manually given rules and regular expressions. The underlying techniques were mainly based on finite-state automata or simple grammar formalisms, and probabilistic versions of them, and resulted in very fast and quite reliable recognizers of the basic, non-recursive *chunks* of a sentence [Church, 1988; Ejerhed, 1988; Abney, 1990, 1991, 1996a; Grefenstette, 1996].

During the 90s, the impact of statistical and machine learning methods on natural language processing gave new challenges and directions in partial parsing [Hammerton et al., 2002]. The general approach, pioneered by Ramshaw and Marcus [1995], consists of formulating a partial parsing task as a general classification problem, such a sequential tagging task. Then, a broad range of machine learning algorithms can be used to automatically learn the classifiers involved in the partial parsing task. The machine learning approach is superior to initial partial parsers in two aspects: first, there is no need to hand-craft the rules of the parser; second, the accuracy of the learned parsers is generally much better. These results permit scaling up partial parsing techniques, from the initial goal of recognizing simple chunks of some specific types, to considering the whole range of syntactic categories, or phrases that admit some recursivity.

In general, partial parsing tasks and techniques lie between part-of-speech tagging and full parsing. In what follows, we first describe typical learning

architectures of partial parsing systems found in the literature. Due to the structured nature of phrases, all of them combine learning and inference processes, mostly sequential. Then we review particular systems of the literature that have influenced this thesis.

2.3.1 Typical Architectures

In the literature, we find different general architectures for partial parsing. Broadly, there are two related main difficulties in a partial parsing task, or, more generally, a phrase recognition task. The first concerns the structural nature of the phrases in a sentence, with implications on the complexity of algorithms. Relative to the length of a sentence, there is a quadratic number of possible phrases, and an exponential number of structures made with phrases. Partial parsing tasks, by definition, are tasks in which the structures are not too complex. Thus, a common approach in partial parsing is based on models at word level, that is, models that associate parameters to features of a word and its local context. These family of models is known as taggers. Beyond taggers, other models operate at phrase and sentence level, with the advantage of exploiting richer patterns and constraints in the representation.

The second difficulty concerns the linguistic scheme with which we are willing to annotate the structures. In other words, what are the labels of the phrases appearing in a sentence and their relationships. In Natural Language tasks, label sets go from syntactic categories to semantic meanings (named entity categories, roles in a predicate of syntactic arguments), and can be defined at different granularity levels. While current systems dealing with syntax achieve reasonable accuracies, semantic disambiguation still constitutes a major challenge, with accuracies that are far from acceptable.

Accordingly, in a solution we distinguish between the bracketing, as the unlabeled structure that segments a sentence into phrases, and the labeling, as the assignment of linguistic labels to the phrases and relations of the bracketing.

In what follows, we review four main approaches to determine a labeled bracketing of phrases in a sentence.

Taggers

A common approach for recognizing phrases in a sentence consists of performing a tagging along the tokens of the sentence. The underlying idea is to represent phrase structures with a limited set of tags, and use essentially the same machinery developed for lower-level problems such as part-of-speech tagging. A tag is assigned to each token or word, or in between of each contiguous pair of tokens. The tags assigned to a sentence encode both the phrasal bracketing and the labels of the phrases and their relationships. The phrase structures must be relatively simple, otherwise the tag scheme becomes too complex. Because of this, under this approach only shallow or limited-depth phrases are considered.

To recognize structure, a learning model is trained to predict the correct tag for each word. The type of learning found in the literature goes from probabilis-

tic methods [Church, 1988; Skut and Brants, 1998] to classification and other distribution-free learners [Ramshaw and Marcus, 1995; Kudo and Matsumoto, 2000; Tjong Kim Sang, 2002b; Collins, 2002]. In all cases, the parameters of the learning model are associated to local patterns of the target word and its neighboring words and tags. Thus, learning components operate at word level. We comment on the two most common tagging strategies:

Open-Close tagging. Church [1988] introduced the idea of parsing simple Noun Phrases by means of performing a tagging along tokens. The approach is to capture statistics for assigning an open or close bracket between two adjacent words, assumed to be PoS-tagged. Then, recognizing NPs in a PoS-tagged sentence consists of computing the most probable bracketing, checking that open and close tags of a bracketing are balanced. Skut and Brants [1998] augmented the tags for recognizing labeled phrases organized in structures of limited depth, set to 3.

BIO tagging. Ramshaw and Marcus [1995] proposed the IOB tagging scheme to represent shallow phrases, alternative to Church's *open-close* brackets. Tags are assigned to words, rather than positions in between, and encode that a word is *Inside* or *Outside* a phrase. A third tag, *Begin*, is used to encode words which begin a noun phrase immediately after another noun phrase. Unlike open-close representations, this tagging scheme is very simple at checking that a certain tagging represents a well-formed phrase structure. Phrase labels can be encoded by augmenting the begin-inside tags with corresponding labels. Tjong Kim Sang and Veenstra [1999] proposed variations of begin-inside-end-outside tags (see also [Tjong Kim Sang, 2002b]). They exploited the difference in expressivity of different tagging schemes. For each one, they learned a set of classifiers, which were then used in combination to improve predictions. In the literature, a standard setting is to use the Begin tag for all words that begin a phrase, together with inside and outside tags. We refer to this representation as BIO tagging.[4]

Recognition + Labeling

In this approach, the phrase recognition task is decoupled into two cascaded subtasks. The first subtask decides for the best well-formed bracketing of the sentence. Then, the second stage decides the appropriate labels for the phrases in the bracketing and their relationships.

Deciding for the best bracketing can be done by tagging the tokens of the sentence with structural tags, which encode brackets of the phrase structure. This tagging is relatively simple, since tags do not encode the labels of linguistic elements.

[4] Tjong Kim Sang refers to this tagging as IOB2; we prefer BIO, since we think it is more intuitive.

After the first stage, the brackets segment the sentence into a number of phrases without labels. In the most simpler version, assigning labels to such phrases corresponds to a multiclass classification problem: for each phrase, a classifier decides for the best label among all possible labels considered in the task. Alternatively, the assignment of labels to phrases can be done dependently. An advantage of this approach is that a phrase is already constructed when selecting the best label for it, and information about the complete phrase can be extracted to support the classification. Thus, in the second stage learning operates at phrase level.

A disadvantage is that the bracketing predicted in the first stage is final, and it has been recognized without considering the labels of the elements in the structure. In some complex domains, it might be argued that this assumption is quite strong, and that it would be better to predict the bracketing dependently of the phrase types.

This two-stage approach is commonly used in problems such as Named Entity Extraction or Semantic Role Labeling. In these tasks, phrases –named entities or propositional arguments, respectively– have a syntactic structure, while their labels denote the meaning of them, according to some predefined semantic scheme. To some extent, it is assumed that phrase boundaries are independent of the semantics of phrases, thus permitting the two-stage approach.

Filtering + Ranking

This approach aims to combine the advantages of the two previous approaches. It is organized in two stages. The first stage, which we refer to as filtering, identifies a set of candidate phrases for the sentence, either labeled or not, which do not necessarily constitute a well-formed solution (e.g., two candidate phrases may overlap, which is never permitted in a phrase structure). The second stage considers well-formed solutions made of candidate phrases and, under a criterion for ranking solutions, selects the top-ranked one.

Filtering is intended to filter out implausible phrases, which constitute most of the possible phrases. It is usually approached as a tagging along the words of the sentence, with learners operating at word level. However, here the tags of the sentence do not represent a well-formed solution, but only a set of possible phrases for the sentence. The goal of this stage is to substantially reduce the output space of the task efficiently (i.e., linearly on the number of words), leaving complex ambiguities for the second stage.

In the second stage, the goal is to predict the best well-formed structure for the sentence, selecting the best candidate phrases to build it. Because of the filtering stage, it can be assumed that the number of candidate phrases is not very high. Then, higher-order processing techniques can be used. In this stage, the candidate phrases are scored by some learning functions. Learners operate at phrase level and, to support the scoring process, information about complete phrases can be extracted. Then, deciding for the best structure is done by selecting the structure made of the best scored phrases. In doing so, it must be taken into account some constraints dictating whether a certain phrase

is compatible with some other phrases to form a valid structure.

This approach is central to this thesis, since Chapter 4 proposes a particular learning architecture based on filters and rankers [Carreras et al., 2005, 2004].

Punyakanok and Roth [2001, 2004] applied this approach to the problem of shallow syntactic parsing, making use of SNoW learners. The phrase structures made of candidate phrases were compactly represented in a directed graph, in which paths connecting two special start and end nodes represent well-formed structures for the sentence. Edges in the graph were weighted according to the classifiers' predictions, and the shortest-path algorithm was used to infer the best structure.

Re-Ranking

Re-ranking methods rely on a baseline system which generates a list of candidate solutions for the problem. Then, the re-ranker system selects the best solution from the list of candidate solutions. Typically, baseline systems are probabilistic systems that select the N most probable solutions for a sentence, where N is a constant of the architecture. Then, the list of candidate solutions is re-ranked so that the correct one is at the top of the list.

The aim of this approach is to improve the accuracy of existing probabilistic systems, motivated by the limitation of probabilistic models to include arbitrary features. On the one hand, the baseline system discards most of the possible structures. In particular, it simplifies the problem from having an exponential number of solutions to just N possible solutions. On the other hand, the re-ranker deals with global solutions, and has to discriminate the correct solution from the list of candidates. To do so, it can incorporate an arbitrary number of features, each encoding information possibly related to an entire solution. Thus, the expressivity of the re-ranker is potentially very rich, compared to the representations adopted in probabilistic models, where only features related to the derivation of a solution are possible. Clearly, to achieve success, the probabilistic model has to be accurate enough to place the correct solution among the list of N candidates. Yet, N cannot be very large, since the re-ranker will exhaustively visit each candidate to score it.

Collins [2000] experimented with a re-ranking model for full-parsing (see also the excellent extended version in [Collins and Koo, 2005]). In that system, the baseline model consists of his head-driven statistical parser [Collins, 1999], while the re-ranker is a boosting learning algorithm, working with hundreds of thousands of global features. Experimentally, the re-ranker achieved a 13% error reduction relative to the accuracy of the baseline parser. Later, Collins and Duffy [2002] experimented with a re-ranker based on the Voted Perceptron with kernels for sequences and trees, in the context of full parsing and named entity recognition.

2.3.2 A Review of Partial Parsing Systems

In this section we briefly review some systems in the literature that deserve special mention. We list them chronologically:

[Church, 1988] Develops a Noun Phrase recognizer that runs after a Part-of-Speech tagger, proposed in the same paper. It performs stochastic Open-Close tagging, with probabilities over open-close brackets between every pair of adjacent PoS tags. The probabilities are estimated as in the PoS tagger, by Maximum Likelihood.

[Abney, 1991] Studies linguistic phenomena related to partial parsing tasks. He provides psycholinguistic evidence in favor of partial analysis, and defines a practical decomposition of a global analysis into partial tasks designed to avoid attachment ambiguities. The architecture is formed of chunkers, that segment the sentence into basic linguistic elements (i.e., chunks), and attachers, that form higher level constituents and detect relations among them. The disambiguation approach is knowledge-based, although the type of tasks and decisions that are considered constitute a source of inspiration for learning-based systems.

[Ramshaw and Marcus, 1995] Introduce the IOB representation for Noun Phrase recognition and a more general shallow parsing task. With IOB representation, the task is formulated as a tagging task involving inside, outside and begin tags. They use Transformation-based learning as a general machine learning algorithm to obtain a classifier for each tag. The concrete definition of Noun Phrase task and the experimental datasets has become a benchmark for evaluating other systems developed later.

[Skut and Brants, 1998] Augment Church's open-close brackets with information on phrase labels and fixed-depth recursions in the phrase structure. They use probabilistic Markov modeling to obtain sequence taggers.

[Tjong Kim Sang and Veenstra, 1999] Explore several representations of a chunk into tags. In particular, making use of chunk *begin-inside-end-outside* tags, they define several tagging schemes, each capturing different particularities of the words that form phrases. They use Memory-based learning to obtain a classifier for each tag. Tjong Kim Sang [2000] applied system combination of different redundant tagging schemes, improving the performance in the context of syntactic chunking. Later, Tjong Kim Sang [2002b] applied similar learning techniques to Clause Identification and other partial parsing tasks, which are cascaded to build a complex output structure.

[Argamon, Dagan, and Krymolowski, 1999] Propose a memory-based sequential learning algorithm to predict labeled bracketings. The training phrase structures are represented as sequences of PoS tags (the input) and phrase

brackets (the output). The memory stores all subsequences of input-output tags found in training, called *tiles*. Then, to recognize a bracketing for a sentence, the training tiles are combined to form coherent bracketings that match the PoS tags of the input sentence. The scoring function of a solution takes into account tile frequencies obtained from training data, and other features regarding the tags in the tiles that form a solution. Greedy inference is performed to find the best solution. The method was first proposed for shallow phrase patterns, and extended later in [Dagan and Krymolowski, 2001] to deal with recursivity.

[Kudo and Matsumoto, 2001] Apply Support Vector Machines (SVM) to syntactic chunking, in the context of the CoNLL-2000 Shared Task. The chunking problem is approached as a sequential tagging, and reduced to multiclass learning. That is, a multiclass classifier decides the appropriate tag for each word of the sentence. The multiclass classifier is implemented via pairwise binary classifiers, each learned independently as a SVM. Classifiers take into account the predictions of neighboring words, and sequential inference is performed using beam search. They experiment with many different tag schemes, and apply system combination to improve the performance. This approach leads the best results on the CoNLL-2000 data set.

[Punyakanok and Roth, 2001] Present two families of sequential inference strategies to perform phrase recognition with word-based classifiers. The first approach is based on probabilistic inference under Markov assumptions. They adapt joint and conditional probabilistic sequential models to work with classifiers. The second scheme for sequential inference is based on constraint satisfaction. Constraints reflect which classifications on different words yield to incoherent phrase structures. The inference algorithm, tailored for shallow phrase structures, selects the optimal global assignment that satisfies the constraints. See [Punyakanok and Roth, 2004] for an extended version.

[Zhang, Damereau, and Johnson, 2002] Use a large-margin version of Winnow to learn word-based classifiers, that are plugged into a Markov conditional model that recognizes shallow phrase structures by tagging words. In syntactic chunking, they obtain state-of-the-art results on the CoNLL-2000 dataset, very close to the leading result. Moreover, by using grammatical information obtained from a resource, they substantially improve the results on that dataset (yielding the best result, although this one is not comparable to others because it uses external information).

[Collins, 2002] Presents a global Perceptron algorithm to train a conditional Markov model for sequential tagging. He experiments with two problems, part-of-speech tagging and Noun Phrase chunking. In both cases, the obtained results are comparable to the top-performing methods in the literature. This global Perceptron algorithm is central for our work, since we propose an extension of

it for more general phrase recognition problems. See also the application of the global algorithm to full parsing in [Collins and Roark, 2004].

[Sha and Pereira, 2003] Apply Conditional Random Fields [Lafferty et al., 2001] to Noun Phrase chunking, performed as sequential tagging. They obtain one of the top-performing results for the specific chunk, in the context of the CoNLL-2000 data.

[Sutton, Rohanimanesh, and McCallum, 2004] Propose dynamic Conditional Random Fields, a global conditional model for recognizing several layers of dependent output labeled sequences. They experiment with the task of jointly predicting part-of-speech tags and noun phrases, showing that the joint model performs better than two separate models applied in cascade.

Chapter 3

A Framework for Phrase Recognition

This chapter discusses a particular framework, and a variety of techniques within it, to recognize a set of phrases in a sentence. The following chapters present particular phrase recognition architectures that make use of the concepts and techniques explained here. Many of the concepts and formalizations in this chapter are based on statistical approaches to tagging and parsing of language [Charniak, 1993] and margin-based learning algorithms [Cristianini and Shawe-Taylor, 2000; Crammer and Singer, 2003b], as well as recent material on parsing with margin-based algorithms [Collins, 2004].

The general scheme consists of a divide-and-conquer strategy. The problem of recognizing phrase structures in a sentence is decomposed into smaller subproblems. This decomposition involves issues concerning the representation of phrase structures in a sentence, the granularity at which the learning functions –as basic recognition functions– are devised and its learnability, and the exploration of the solution space.

The components of a phrase recognition architecture are :

- The **model**, which defines the representation of a solution and the components of the recognition (i.e., the learning functions).
- The **inference algorithm** (a.k.a. tagger, parser, decoding algorithm), which defines the strategy to efficiently search for the best solution.
- The **learning strategy**, for training the learning components of the architecture from data.

The rest of the chapter is organized as follows. The next section gives a formal definition of the phrase structures and the problem of recognizing them in a sentence. Then, the following sections discuss the design of the three main components of a phrase recognition architecture; namely models, inference algorithms, and discriminative learning strategies.

3.1 A Formal Definition of Phrase Structures

Let x be a sentence consisting of a sequence of n words $[x_1, x_2, \ldots, x_n]$, belonging to the sentence space $\mathcal{X}$. Let $\mathcal{K}$ be a predefined set of phrase categories. For example, in the syntactic parsing task $\mathcal{K}$ may include, among others, noun phrases, verb phrases, prepositional phrases, and clauses. A *phrase*, denoted as $(s, e)_k$, is the sequence of consecutive words spanning from word x_s to word x_e, having $s \leq e$, with category $k \in \mathcal{K}$.

Let $p_1 = (s_1, e_1)_{k_1}$ and $p_2 = (s_2, e_2)_{k_2}$ be two different phrases. We define that p_1 and p_2 *overlap* iff $s_1 < s_2 \leq e_1 < e_2$ or $s_2 < s_1 \leq e_2 < e_1$, and we note it as $p_1 \sim p_2$. Also, we define that p_1 is *embedded* in p_2 iff $s_2 \leq s_1 \leq e_1 \leq e_2$, and we note it as $p_1 \prec p_2$.

Let $\mathcal{P}$ be the set of all possible phrases, expressed as:

$$\mathcal{P} = \{(s, e)_k \mid 1 \leq s \leq e,\ k \in \mathcal{K}\}$$

In a phrase recognition problem, a *solution* for an input sentence x is a finite set y of phrases which is *coherent* with respect to some *constraints*. One of the constraints is that phrases of a solution are not allowed to overlap. Then, we differentiate two types of problems depending on the embedding constraint.

In a **sequential phrase recognition problem**, often referred to as *chunking*, phrases of a sentence are not allowed to embed. Thus, the solution space can be formally expressed as:

$$\mathcal{Y} = \{y \subseteq \mathcal{P} \mid \forall\, p_1, p_2 \in y \quad p_1 \not\sim p_2 \wedge p_1 \not\prec p_2\}$$

In the literature, it is common to refer to a phrase that does not accept embedded phrases as a *chunk* or *base phrase*.

In a **hierarchical phrase recognition problem**, a solution is a set of phrases which may be embedded. Formally, the solution space is:

$$\mathcal{Y} = \{y \subseteq \mathcal{P} \mid \forall\, p_1, p_2 \in y \quad p_1 \not\sim p_2\}$$

Note that the embedding of phrases is not a requirement. Either a set of chunks, a tree of phrases, or a forest of phrases are valid phrase structures for a sentence. Thus, the output space here is general but also complex. It is common to refer as *recursive phrase* to a phrase that contains embedded phrases, these being either recursive or base phrases.

Note that, for simplicity and readability, we have defined the $\mathcal{P}$ and $\mathcal{Y}$ sets independently from concrete word sequences. However, whenever we are referring to the $\mathcal{P}$ and $\mathcal{Y}$ sets in the context of a concrete input sequence $[x_i]_{i=1}^n$ we implicitly restrict to phrases $(s, e)_k$ which span actual word sequences in x (i.e., $1 \leq s \leq e \leq n$), and to solutions formed with these phrases.

3.2 Models

The models for phrase recognition studied in this thesis are described as a function $\mathrm{R} : \mathcal{X} \rightarrow \mathcal{Y}$, where $\mathcal{X}$ is the space of sentences and $\mathcal{Y}$ are all possible phrase structures. The R function follows the general form:

$$\mathrm{R}(x) = \arg\max_{y \in \mathcal{Y}} \mathrm{score}(x, y) \tag{3.1}$$

Here, the score function produces a plausibility score for a solution y being the correct structure of the input sentence x. Potentially, the R function scores all possible structures and selects the one with maximum score. In practice, the scoring function at sentence level is a composition of several lower level decisions, so that there exists a mechanism to efficiently explore the output space. We call this mechanism the inference strategy, and will be discussed in the next section. The difference in the models relies on the decomposition of the scoring function. Defining a particular model concerns defining the following issues:

- A **decomposition scheme** to break down a global solution –a set of phrases– into a number of parts which compose the solution. Such parts can be, for example, the phrases which compose the solution, or some tags assigned to words which allow to construct the solution. A particular decomposition scheme implies constraints that dictate how local parts can be combined to form a coherent global structure. In other words, out of all possible combinations of local parts, those which satisfy the constraints are those which correspond to valid phrase structures. Equivalently, the model can be associated a generative mechanism to incrementally produce a global structure by generating, in a time-line of decisions, a local part at each step. Again, the generative mechanism must be correct –in that the generated solutions are always valid phrase structures– and complete –in that for each valid phrase structure there exists a (unique) sequence of decisions which generates it.

- A set of **learning functions** to score parts, in accordance with the granularity of the decomposition scheme. These functions are trained from data to predict a confidence score for a certain decision. Some functions predict a plausibility score of assigning a phrase of some type to some part of the sentence. Others decide whether a word starts or ends a certain phrase. In all cases, a learning function receives some parameters describing the context of the decision, and outputs a real-valued score indicating the plausibility of the decision: positive numbers indicate a plausible decision, while negatives indicate non-plausible, and the magnitude of the prediction indicates the confidence of it.

- An **optimality criterion** to select a solution. Such criterion follows the form of the above equation. In particular, as we show below, a global solution is scored by accumulating the predicted scores of the parts it

contains. Then, the selected solution is the one which receives higher score.

3.2.1 Models at Word-Level

The idea behind word-based models is to represent a solution with a sequence of tags, each of which is assigned to a word in the input sentence. Thus, the phrase recognition problem is reduced to a sequence tagging problem, and sequential learning techniques can be applied. This approach is specially suitable when dealing with sequential phrase structures, since the underlying representations into tags are very simple.

Formally, given a sentence $x = [x_i]_{i=1}^n$, there is an output variable y_i associated with each word x_i. The model defines a set of tags $\mathcal{T}$, and each variable is assigned one of the tags, $y_i \in \mathcal{T}$. Finally, the model defines a set of constraints forcing a sequence of output variables to form a valid global structure. Basically, such constraints correspond to the structural constraints of a solution –non-overlapping or non-embedding of phrases– expressed in terms of tag assignments.

Then, a learning function is trained to recognize tags on words. This function, named score_w, predicts a confidence score for the assignment of a certain tag to a certain word. Recognizing a phrase structure for a sentence corresponds to find the best tagging along it, ensuring that the tagging is a well-formed phrase set. A word-based model computes the score of a solution as the summation of scores of word-tag assignments:

$$\text{score}(x, y) = \sum_{i=1}^{n} \text{score}_\text{w}(x, y, i, y_i) \tag{3.2}$$

In the expression, the score_w function predicts a score for assigning the tag y_i to the i-th word, in the context of a sentence x and a solution y.

We now discuss two well-known word-based models for recognizing sequential phrase structures as a tagging task.

Begin-Inside-Outside In this reduction, each word in the sentence is tagged either as the *beginning* of a phrase (`B` tag), a word *inside* a phrase (`I` tag), or a word *outside* a phrase (`O` tag). Considering a phrase scheme of types in $\mathcal{K}$, a BIO model defines begin and inside tags for each type in $\mathcal{K}$, and an outside tag independent of the categories. Hence, there are $2|\mathcal{K}| + 1$ tags. Figure 3.1 shows a representation of the syntactic chunks of a sentence with `BIO` tags. Note that two tags are needed for every type to differentiate a certain phrase in a span from a sequence of contiguous phrases at the same span, all of the same type –in the example, "noon" and "yesterday" are two contiguous `NP` chunks.

In this representation, a sequence of tags is valid if all maximal subsequences of `I-`k tags are preceded by a `B-`k tag, where k is a phrase type. This constraint, however, is not critical at interpreting an ill-formed sequence of tags: an `I-`k tag not preceded by another `I-`k tag may be considered a `B-`k tag. A good property

Original Sentence:

(The San Francisco Examiner)$_{\texttt{NP}}$ (issued)$_{\texttt{VP}}$
(a special edition)$_{\texttt{NP}}$ (around)$_{\texttt{PP}}$ (noon)$_{\texttt{NP}}$ (yesterday)$_{\texttt{NP}}$.

BIO Representation:

The$_{\texttt{B-NP}}$ San$_{\texttt{I-NP}}$ Francisco$_{\texttt{I-NP}}$ Examiner$_{\texttt{I-NP}}$ issued$_{\texttt{B-VP}}$
a$_{\texttt{B-NP}}$ special$_{\texttt{B-NP}}$ edition$_{\texttt{I-NP}}$ around$_{\texttt{B-PP}}$ noon$_{\texttt{B-NP}}$ yesterday$_{\texttt{B-NP}}$.

Start-End Representation:

$_{\texttt{S-NP}}$The$_{\lambda}$ $_{\lambda}$San$_{\lambda}$ $_{\lambda}$Francisco$_{\lambda}$ $_{\lambda}$Examiner$_{\texttt{E-NP}}$ $_{\texttt{S-VP}}$issued$_{\texttt{E-VP}}$
$_{\texttt{S-NP}}$a$_{\lambda}$ $_{\lambda}$special$_{\lambda}$ $_{\lambda}$edition$_{\texttt{E-NP}}$ $_{\texttt{S-PP}}$around$_{\texttt{E-PP}}$ $_{\texttt{S-NP}}$noon$_{\texttt{E-NP}}$
$_{\texttt{S-NP}}$yesterday$_{\texttt{E-NP}}$.

Figure 3.1: Example of `BIO` and `SE` taggings representing a sequential phrase structure, corresponding to the syntactic chunks of a sentence. In `SE` tagging, every word is associated two variables, one for start tags (preceding the word) and another for end tags (following the word). λ represents a null assignment.

of this constraint –which is the only one of the model– is that it can be verified looking only at every tag bigram: a tag `I-`k assigned to a word i, for some k, is valid only if the tag assigned to word $i-1$ is `B-`k or `I-`k. Thus, structurally, a certain tag only depends on its predecessor in the sequence, which directly allows the use of Markov modeling –as it will be shown later in Section 3.3.1.

There exist many variations of the `BIO` tagging scheme. Originally, this representation was introduced as `IOB` tagging by Ramshaw and Marcus [1995], but there the `B` tag is used strictly when two phrases are contiguous. Symmetrically to `BIO`, one can use `IEO` tags to mark inside-end-outside phrase words. In general, one can use any combination of begin-inside-end-outside tags which, in nature, better represents a phrase structure. Tjong Kim Sang and Veenstra [1999] and Kudo and Matsumoto [2001] experimented with several types of taggings, and combinations of them, in the context of shallow syntactic parsing.

Start-End A start-end model defines *start* and *end* tags for each type in $\mathcal{K}$, resulting in a scheme of $2|\mathcal{K}|$ tags. In this representation, a phrase of type k receives the start tag (`S-`k) at the first word and the end tag (`E-`k) at the last word, and no tags in the middle words. Note that when a phrase consists of a single word, that word receives both tags, and that when a word is strictly within a phrase, it receives no tag. Formally, the model can be thought as having two variables per word: the first receives one of the start tags and the second receives one of the end tags, and in either cases they can be assigned a null tag. Figure 3.1 shows a sequential phrase structure represented with start-end tags.

This model imposes constraints for a sequence of tags to represent a valid, well-formed phrase structure. The two enclosing tags of a phrase have to be of the same type. When dealing with sequential structures, no tags can appear

between the start-end tags of a phrase. When embedding is allowed, the tags enclosed within a phrase boundary structure must form a valid structure of start-end tags. As opposed to BIO models, the structural constraints cannot be expressed as relations between a certain tag and a fixed-length window of previous tags. Thus, Markov modeling cannot be directly used. To overcome this limitation, the models for chunking by Punyakanok and Roth [2004] augment the start-end tags (noted there as *open-close*) with two empty tags: one for non-boundaries inside a phrase, and the other for non-boundaries outside the phrase. With the augmented tag set, the constraints can be expressed as Markovian dependencies.

3.2.2 Models at Phrase-Level

Working at phrase level provides a natural granularity for phrase recognition models. Compared to word level, there is more flexibility at defining constraints, since these can be expressed as relations between two phrases. Also, as we will see, it increases the expressivity of the learning functions, in terms of the possible features that can be extracted to represent learning instances. On the other hand, moving from word to phrase levels increases the computational cost of the phrase recognition strategy.

In phrase-based models, a global solution is decomposed into its phrases. Figure 3.2 shows the collection of phrases of an example sentence. The phrases of a solution are scored by a function, and the global score is defined as the summation of scores of the phrases in the solution:

$$\text{score}(x, y) = \sum_{(s,e)_k \in y} \text{score}_{\text{p}}(x, y, (s, e)_k) \tag{3.3}$$

The score_{p} function, at phrase level, produces a confidence score for assigning a phrase of type k to the span (s, e), in the context of a sentence x and a candidate solution y.

The model can be formalized as follows. Given a sentence x of length n, there is an output variable $y_{s,e}$ for each phrase span, that is, $1 \leq s \leq e \leq n$. Let $\mathcal{K}'$ be the set of possible phrase types $\mathcal{K}$ augmented with a null phrase type λ, used as a special value indicating that certain span does not correspond with any phrase. In phrase-based models, the phrase recognition task can be seen as assigning values of $\mathcal{K}'$ to variables $y_{s,e}$. Note that with this formalization a particular span can be assigned only one phrase. This might be not general enough for phrase recognition tasks, where in some cases there several phrases, of different types, at the same span. However, it is always possible to engineer a fine-grained assignment in a span, having a fixed number of variables in it and establishing some order for giving them values. This is the approach followed in start-end tagging models, where each word has two variables. Also, in syntactic parsing it is common to limit the recursivity of unary productions to a fixed number. For simplicity of the framework, we assume tasks where at most one phrase per span is possible.

((The San Francisco Examiner)$_{\texttt{NP}}$ (issued)$_{\texttt{VP}}$ (a special edition)$_{\texttt{NP}}$ (around)$_{\texttt{PP}}$ (noon)$_{\texttt{NP}}$ (yesterday)$_{\texttt{NP}}$.)$_{\texttt{S}}$

$$y = \{\ (1,12)_{\texttt{S}}, (1,4)_{\texttt{NP}}, (5,5)_{\texttt{VP}}, (6,8)_{\texttt{NP}}, (9,9)_{\texttt{PP}}, (10,10)_{\texttt{NP}}, (11,11)_{\texttt{NP}}\ \}$$

Figure 3.2: Example of a syntactic phrase structure represented as a collection of phrases. Note that the clause, $(1,11)_{\texttt{S}}$, embeds all other phrases, which are syntactic chunks.

Importantly, a set of constraints over an assignment dictates whether the assignment constitutes a valid phrase structure. The constraints encode structural properties of the solution, and can be expressed as relations over every two pair of phrases of a solution. In particular, in a valid solution two phrases cannot overlap, and, for chunking tasks, phrases cannot be embedded. Note that here the constraints are expressed as we did when formalizing phrase structures in Section 3.1.

Thus, the general task is to find the best global assignment, in terms of confidence score of phrase variables, which satisfies the set of constraints.

Phrase-Based Models with Start-End Filtering

In a sentence of length n, a phrase-based model considers a quadratic number of variables, one for each phrase span. In particular, there are $\frac{n^2+n}{2}$ phrase spans, each of which may be assigned a phrase label in $\mathcal{K}$, denoting a phrase in that span, or a null label, denoting no phrase.

Evaluating straightforwardly the score of each phrase type at each span might be computationally expensive. In that case, it might be desirable to apply a filtering process which blocks some of the combinations of the scoring process.

A phrase recognition model working at phrase level and extended with a filtering component is expressed as follows:

$$\mathrm{R}(x) = \underset{y \in \mathcal{Y} \ \mid \ y \subseteq \mathrm{F}(x)}{\arg\max} \sum_{(s,e)_k \in y} \mathrm{score}_{\mathrm{p}}(x, y, (s,e)_k)$$

Let $\mathcal{P}$ be the set of all possible phrases for a sentence, as defined in Section 3.1, and recall that solutions are subsets of $\mathcal{P}$ satisfying the structural constraints. The function F is intended to filter out phrase candidates from $\mathcal{P}$ by applying decisions at word level. A simple setting for this function is a *start-end* classification for each phrase type: each word of the sentence is considered as *k-start* —if it is likely to start a type-k phrase— and as *k-end* —if it is likely to end a type-k phrase. Each *k-start* word x_s with each *k-end* word x_e, having $s \leq e$, form the phrase candidate $(s,e)_k$. Assuming *start* and *end* binary classification functions, start^k and end^k, for each type $k \in \mathcal{K}$, the filtering function

is expressed as:

$$\mathrm{F}(x) = \{\, (s,e)_k \in \mathcal{P} \mid \mathrm{start}^k(x,s) \wedge \mathrm{end}^k(x,e) \,\}$$

Note that the start-end functions work at word level and are independent of any solution y. Thus, this model can be organized in two levels of processing: first, the filtering layer –which disables phrase candidates– and then the ranking layer –which scores global solutions made of phrases that pass the filter and selects the top-ranked.

In Chapter 4 we discuss in detail this filtering-ranking model, and propose a Perceptron learning strategy to train the model learning functions.

3.2.3 Models in a Phrase Recognition Architecture

We now discuss the relation of models with the inference and learning components of a phrase recognition architecture.

The score learning functions, as well as other learning functions the model may define, make use of the input parameters to extract features from them. Such features represent the context of the decision, and have to be expressive enough so that it is possible to learn an accurate prediction rule for the function's decision. The actual extracted features depend on the particular task and the nature of phrases to be recognized. As it will be shown, we will assume a feature extraction function ϕ (*a.k.a.* representation function) that will represent a decision and its context with a set of features. The design of the ϕ function is related to domain and task knowledge. Therefore, an important aspect is that the granularity at which the learning functions are defined allows to extract features considered relevant for the task. For example, in phrase-based models features can represent patterns of a complete phrase, whereas in word-based models only features representing a word and its nearby context are possible.

A key issue related to the inference concerns the global solution y appearing as a parameter in the input of the scoring functions. In practice, the inference component will make use of the scoring functions while building incrementally y. Thus, when predicting a confidence score, y will actually be a partial solution, rather than a global solution. As a consequence, only features representing the visible part of y are possible.

A different aspect, relating the model and the inference components, concerns the solution space $\mathcal{Y}$ and the use of a resource which explains the structure of it. As an example, in full syntactic analysis it is common to make use of a grammar of the language to explore the solution space. In our framework, the output space is that of all possible well-formed structures of phrases –i.e., those in which phrases do not overlap and, occasionally, form hierarchies. In this scenario, a grammar is seen as a knowledge resource for limiting the output space: out of all solutions, only those which can be generated with grammar rules need to be considered. The phrase recognition architectures of this thesis do not make use of grammar-like resources, but consider all well-formed structures as potential solutions. However, the employed inference strategies are essentially

the same used for context-free grammar models. Thus, we want to point out that the use of a grammar is compatible with the discussed models.

3.3 Inference Algorithms

Inference algorithms provide strategies to recover the best phrase structure for a sentence, given a particular model with its scoring functions. Two properties are of particular interest in these algorithms:

- **Efficiency**. A naive approach to find the best solution is to enumerate all possible solutions, score them, and select the optimal one. However, the output space (i.e., sets of phrases) is exponentially large with the length of the phrase. We desire inference algorithms of polynomial time.

- **Robustness**. The learning functions of the model will never be perfect. In the architecture, decisions are often chained, and errors on early predictions may severally degrade the accuracy of later decisions. Thus, we desire inference strategies which are able to recover from local prediction errors. This will be possible by checking the consistency of the local predictions in a global context, with the aim that a local error will further produce an inconsistency, or will lead to a non-plausible (low-scored) global solution.

An inference strategy can be casted into a search scheme: given the space of all possible global solutions, and the optimality criterion fixed in the model, the inference strategy searches for the best solution for a given sentence.

The particular search strategy relies on the decomposition of the score function defined by the model. In the model, a global solution is broken down into a set of parts (in our framework, tagged words or phrases). Accordingly, the strategy consists of two related subtasks. First, visiting parts of the sentence and assigning partial output structures to them. Then, combining partial structures of the different parts to form a global structure, checking the consistency between parts with respect to the structural constraints of the model.

The general scheme for the inference algorithms discussed in this thesis combines both subtasks, that is, solutions are built incrementally while computing predictions. Figure 3.3 presents an algorithmic view of it. The strategy consists of visiting parts of the input sentence in a fixed order. We assume a decomposition function $D(x)$ which enumerates the sequence of parts of a sentence x. In the time-line, at each time-step a new part is visited, with no backtracking. At each part, a number of candidate partial solutions is maintained, noted as Y_t in the algorithm. When a part is visited:

- The list of partial solutions, Y_t, is built by combining solutions of the previously visited parts, $Y_{0:t-1}$.

- Each partial solution is incremented with local structure on the current part, checking that the incremented solution is coherent with respect to the

- Define a decomposition of the sentence x into a sequence of parts: $D(x) = [p_1, \ldots, p_t, \ldots, p_T]$
- Initialize pool of solutions to empty : $Y_0 = \{\lambda\}$.
- Visit parts: For each $p_t \in D(x)$ do :
  - For each well-formed solution y combined from $Y_0 \ldots Y_{t-1}$ For each value v for p_t consistent with y (including *null*):
    - Increment y: $y' = y + (p_t, v)$
    - Evaluate y': $\text{score}(x, y') = \text{score}(x, y) + \text{score}(x, y', p_t, v)$
    - Add y' to Y_t
  - Prune Y_t : filter out non-plausible solutions from the set, according to their score.
- Return $\arg\max_{y \in Y_T} \text{score}(x, y)$

Figure 3.3: A general inference scheme for recovering phrase structures

structural constraints. The score function is used to produce a confidence value for the incremented part, which is accumulated to the global score of the partial solution.

- Before leaving the part, the set of partial solutions is pruned, with two different motivations. On the one hand, some solutions will not lead to the optimal solution, and therefore can be discarded. On the other hand, for efficiency reasons, the search can be made aggressive, and approximated, by selecting only the best solutions under some heuristics.

After all parts have been visited, the candidate solutions of the last part, Y_T, are complete solutions. Among them, the selected solution is the one with highest score.

An important aspect of the inference strategy is whether, given a particular model with trained functions, the strategy is exact or approximated. In other words, whether it selects the top scored solution of $\mathcal{Y}$, or might return a sub-optimal of it. There are two conditions for the inference strategy to be exact, related to dependencies established by the model on the parts of a decomposed solution. Satisfying the dependencies implies exact inference. The dependencies are the following:

- **Structural Dependencies.** These dependencies are imposed by the structural constraints of the model. In incremental inference, a partial solution in a certain part is built by combining partial solutions on previous

parts, together with local structure for the current part. Only combinations which satisfy the structural constraints are possible. Thus, the local structure that can be assigned at a certain part *depends structurally* on the partial solutions of previous parts. It may happen that some local assignment is never considered because that assignment is not coherent with any of the partial solutions on previous parts. If this situation is possible, the inference strategy may lead to a sub-optimal solution. When the strategy maintains enough partial solutions at each part so that all assignments are possible in further parts, the inference is structurally exact.

- **Feature Dependencies.** When scoring a local assignment, the learning functions receive in the input the partial structure y which is being incremented. This partial structure is used to extract features from it. Let $\varphi(y)$ denote the part of y from which features are extracted –that is, $\text{score}(x, y, p_t, v) = \text{score}(x, \varphi(y), p_t, v)$. For instance, $\varphi(y)$ might be the top-most phrases of y. As it will be shown in the next section, the quantity predicted by a learning function depends linearly on the values of the instance features. Thus, different values on an instance feature produce different predictions, and such difference depends on a weight of the feature that is set during learning. Hence, the plausibility score of assigning a value v to a part p_t depends on the value of $\varphi(y)$ from which features are extracted. We call *feature dependencies* to the dependencies established between the score of a local assignment and the part of y exploited with features, $\varphi(y)$. To guarantee exact inference, the strategy has to maintain at a given part one solution for every possible instantiation of $\varphi(y)$, so that in a further part a certain assignment can be scored on the whole range of $\varphi(y)$ values. Note that these dependencies are established when designing the representation of parts into features, in a particular task –a representation which is defined as a function Φ that transforms an instance into a feature-vector representation. In some simple domains, it might seem adequate to make the scoring of a part independent of its global structure ($\varphi(y)$ would be empty), resulting in a model with no feature dependencies. In other domains, it might be necessary to condition on the global structure, resulting in a recurrent architecture where predictions of early steps are used as input values of the predictions of further steps.

The inference strategies we discuss go from exact to approximated explorations. Exact explorations maintain at each part the necessary partial solutions to satisfy structural and feature dependencies between local and further assignments. To approximate the inference, and make it more efficient, conditions are defined on the behavior of the score function, in such a way that some solutions, considered non-plausible, can be discarded.

In the rest of the section we present instantiations of incremental inference strategies for the models of the previous section, discussing technical details and properties of the different variations. First we discuss inference in word-based models, and then for phrase-based models. Finally, we discuss the influence of the type of inference in a phrase recognition architecture.

3.3.1 Inference in Word-Based Models

In word-based models, learning is applied at word level. The scoring function predicts a confidence score for assigning a certain tag to a certain word of the sentence. Thus, the inference strategy consists of visiting words and assigning tags to them, with the goal of building the best sequence of tags. The exploration of a sentence x, of size n, is done sequentially in a certain direction. We assume that words are visited from left to right, $(x_1, \ldots, x_n)$, although the same strategy works from right to left $(x_n, \ldots, x_1)$.

Since the output in word-based models is a sequence of tags, let $\mathcal{T}$ denote the set of possible tags, and let y denote a sequence of tags representing a solution (rather than a solution itself), that is, $y = [y_i]_{i=1}^{n}$, with $y_i \in \mathcal{T}$. For brevity, we will denote as $y_{r:s}$ the subsequence of tags of $[y_i]_{i=r}^{s}$.

Solutions are incrementally built while visiting words. After visiting the i-th word, the partial solutions associated to that part will be of i tags, from word 1 to word i. After exploring the sentence, the solutions at word x_n will be complete.

Greedy Decoding An aggressive inference strategy consists of maintaining only one partial solution in the exploration, named y. At the beginning, y is a null sequence. Then, at the i-th word, in increasing order, y is incremented with the corresponding tag, so before moving to the next word y is equivalent to $y_{1:i}$. We assume a function of the model, $\text{valid} : \mathcal{T}^* \times \mathcal{T} \rightarrow \{true, false\}$, indicating whether augmenting a partial sequence y with a tag t forms a coherent partial sequence. When visiting the i-th word of a sentence x, the selected tag is given by the expression:

$$y_i = \underset{t \in \mathcal{T} | \text{valid}(y,t)}{\arg\max} \text{score}_{\text{w}}(x, y_{1:i-1}, i, t)$$

This greedy strategy favors the efficiency of the search. It does not guarantee to select the optimal sequence of tags in terms of global score, both for feature and structural dependencies. Concerning feature dependencies, the score functions at word level receive in the input the partial tag sequence $y_{1:i-1}$. If features are extracted from it, then selecting the best local tag might conduce to a local maximum. Concerning structural constraints, it may happen that selecting locally the optimal tag conduces to a suboptimal completion of the sequence. For example, in `BIO` tagging, if a `B` tag is not assigned to a certain word, then the `I` tags on subsequent words are not possible.

Viterbi Decoding under Markov Assumptions In Markov models, it is assumed that a tag of the sequence at a certain position only depends on the d previous tags, where d is the order of the model. In this case, the inference mechanism is solved by the Viterbi algorithm, which can be explained as follows. At each word x_i, the pool of solutions $\mathcal{Y}_i$ contains up to $|\mathcal{T}|^d$ partial solutions, one for each possible d-gram of tags except for those which are not coherent. For generating the pool, each solution of $\mathcal{Y}_{i-1}$ is incremented with a valid tag for

the current word. We assume a function, valid : $\mathcal{T}^d \times \mathcal{T} \rightarrow \{true, false\}$, that indicates whether concatenating a tag to a d-gram of tags forms a valid sequence, with respect to structural constraints. Then, the generation of solutions at a word x_i is:

$$Y_i' = \{\ y = [y', t]\ \mid\ y' \in Y_{i-1}\ \wedge\ t \in \mathcal{T}\ \wedge\ \text{valid}(y_{i-d:i-1}, t)\ \}$$

Each partial solution of Y_i is scored by predicting the score of the current tag and accumulating it to the score that the solution had in the previous step:

$$\text{score}(x, y = [y', t]) = \text{score}(x, y') + \text{score}_\text{w}(x, y, i, t)$$

Due to the Markov assumption, the partial solutions ending with the same d-gram of tags will produce the same tags in further steps of the inference. Thus, only the best solution for each d-gram needs to be maintained. The pruned solution set can be expressed as:

$$Y_i = \bigcup_{\mathbf{t} \in \mathcal{T}^d} \underset{y \in Y_i \mid \mathbf{t} = y_{i-d+1:i}}{\arg\max} \text{score}(x, y)$$

When scoring a tag, the score_w function can extract features from the current tag sequence y, which spans from word 1 to word i. Exact inference is guaranteed as long as only features from $y_{i-d+1:i}$ are extracted.

The cost of the algorithm is linear with the length of the sentence n. However, it is important to note that the algorithm maintains $|\mathcal{T}|^d$ partial solutions, each of which is incremented at each step. Although it is independent of n, it rapidly becomes a huge number when dealing with big tag sets or the order of the dependencies d increases.

In practice, thus, it might be appropriate to make the search efficient, and approximated, by maintaining only a *beam* of the best partial solutions at each Y_i, rather than the complete list associated to each d-gram.

A main drawback of Markovian sequence modeling in phrase recognition tasks is that the independence assumption might be a quite strong assumption. Computationally, it is feasible to work with dependency orders of 2 or 3, meaning that a word tag can be determined looking only at the two or three previous tags. However, it is common to find phrases which span a flexible number of words, due to a recursive nature of them. Presumably, therefore, dependencies between phrases occur in contexts of flexible size, rather than in a fixed window of a few words.

3.3.2 Inference in Phrase-Based Models

Phrase-based models decompose the problem at phrase level, and rely on the score_p function to predict a confidence score for a phrase candidate. The inference problem is to incrementally build the best structure of phrases. The strategy consists of visiting phrase spans and assigning phrases to them, with the goal of building the best global phrase structure.

In a sentence of n words, the phrase spans can be represented as start-end pairs (s, e), with $1 \leq s \leq e \leq n$. The inference strategies we discuss explore the sentence from the bottom up, in such a way that when visiting the (s, e) span all the internal spans have been visited. This can be accomplished, for example, by priorizing the length of the phrase, resulting in a CKY-style decoding:

$$D_{\mathrm{p}}(x) = [\ \underbrace{(1,1),\ldots,(n,n)}_{\text{length 1}}, \underbrace{(1,2),\ldots(n-1,n)}_{\text{length 2}}, \underbrace{(1,3),\ldots(n-2,n)}_{\text{length 3}}, \ldots, \underbrace{(1,n)}_{\text{length } n}\ ]$$

Alternatively, the sentence can be processed from left-to-right, visiting phrase spans up to a certain word:

$$D_{\mathrm{p}}(x) = [\ \underbrace{(1,1)}_{\text{end at 1}}\ ,\ \underbrace{(2,2),(1,2)}_{\text{end at 2}}, \underbrace{(3,3),(2,3),(1,3)}_{\text{end at 3}}, \ \ldots,\ \underbrace{(n,n),\ldots,(1,n)}_{\text{end at } n}\ ]$$

At each phrase span (s, e), a set $Y_{s,e}$ of candidate partial solutions is maintained. After visiting the span, the set will contain all the recognized partial structures formed of phrases found within the span. At the end of the process, the selected solution is the top-scored structure of $Y_{1,n}$. We first discuss inference algorithms exploring the sentence in CKY-style. Then, we describe left-to-right strategies.

CKY-style Cubic Inference in Phrase-Based Models

CKY-style inference visits first phrase spans which are shorter in length. This allows an inference strategy with no structural dependencies, that is, at every span all local assignments will be possible. [1] The strategy works as follows. For building structure at a given span (s, e), we differentiate between the local phrase, covering completely the span, and the internal phrases, strictly contained in within.

The internal structure can be determined by choosing a splitting point m, such that $s \leq m < e$, which divides the span into a left part (from s to m) and a right part (from $m+1$ to e). Recall that the partial solutions of all the internal spans are already build, together with their scores, when visiting the span. Any combination of a structure y_L in $Y_{s,m}$ and another y_R in $Y_{m+1,e}$ constitutes a complete and well-formed internal structure for the current span. For recursion, both y_L and y_R are well formed solutions, and since they correspond to disjoint parts of the sentence, phrases in them cannot overlap nor embed.

As for the local phrase, covering completely the span, all types of phrases are in principle considered as candidates for it[2], while it is also possible to assign the null tag, denoting no phrase at the span.

[1] In the original CKY, which applies to context-free grammar-based parsers, not all local assignments are possible in a span, but only those that match with some rule of the grammar, given the constituents found within the span in earlier steps. As it has been pointed out, in this thesis we do not make use of grammars to guide the exploration.

[2] At this point, one could use a grammar to consider only combinations of internal structure and local phrase that match some grammar rule.

Finally, to generate the partial structure of a span, the local phrase and the internal structure must be combined. At this point, the non-embedding constraint must be considered. When dealing with sequential structure, the non-embedding constraint is active. Thus, the partial structure of the span is either the local phrase or the internal structure, and the score of each is used to select the best. With hierarchical structure, in principle any internal structure is compatible with any local phrase at the span. Note that such a combination will result in a phrase structure where the local phrase embeds the internal phrases.

We now enumerate all possible solutions in a span, when embedding is allowed. Let $\mathcal{K}'$ be the set of types $\mathcal{K}$ augmented with a null type indicating no phrase. Assuming recursion, the potential set of partial solutions in (s, e) can be formalized as:

$$Y_{s,e} = \{\ y_L \cup y_R \cup \{(s,e)_k\} \mid s \leq m < e\ \wedge\ y_L \in Y_{s,m}\ \wedge\ y_R \in Y_{m+1,e}\ \wedge\ k \in \mathcal{K}'\ \}$$

Let $y^m_{s,e}$ denote the internal structure $y_{s,m} \cup y_{m+1,e}$, with $y_{s,m} \in Y_{s,m}$ and $y_{m+1,e} \in Y_{m+1,e}$. Let $y^{m,k}_{s,e}$ denote the partial structure formed with $y^m_{s,e}$ and the local phrase $(s,e)_k$, possibly null since $k \in \mathcal{K}'$. Each solution in $Y_{s,e}$ is scored as:

$$\text{score}(x, y^{m,k}_{s,e}) = \text{score}(x, y_{s,m}) + \text{score}(x, y_{m+1,e}) + \ \text{score}_{\text{p}}(x, y^m_{s,e}, (s,e)_k)$$

The above enumeration is exhaustive and grows exponentially with the length of the span. Thus, it is not tractable and some assumptions must be made to limit the exploration. In any case, the inference strategy works at least in cubic time, since there is a quadratic number of phrase spans, and at each one there is a linear number of splitting points to be considered. We now discuss some possibilities for limiting the exploration in a span, by increasing order of computational cost:

- **Independence between internal structure and local scoring.** In this case it is assumed that the best internal structure in a span can be determined independently of the local phrase. With this assumption, only one partial structure needs to be maintained at each span, noted $y_{s,e}$, which is determined in two steps. The first step is to select the optimal splitting point between s and e:

  $$m = \arg\max_{s \leq m < e} \text{score}(x, y^m_{s,e})$$

  With it, the internal structure is formed, $y^m_{s,e} = y_{s,m} \cup y_{m+1,e}$. Then, the possible local phrase in (s,e) is considered. To do so, each possible type in $\mathcal{K}$ is tried, and the top scored is selected:

  $$k = \arg\max_{k \in \mathcal{K}} \text{score}_{\text{p}}(x, y^m_{s,e}, (s,e)_k)$$

  In the case that the score associated to the optimal type k is negative, no local phrase would be inserted to $y_{s,e}$ (which can be seen as assigning

the special null type to the local phrase, with 0 score). This strategy is approximated when the score_p function extracts features from the internal structure $y^m_{s,e}$, since the local score depends on the internal structure, but only the selected $y^m_{s,e}$ is considered for choosing the local phrase. If no features are extracted, inference is exact.

For sequential phrase structures, this assumption is quite reasonable. Due to the non-embedding constraint, it is not possible that a phrase contains other phrases in within. Thus, it makes sense to score a phrase independently of its internal structure, since we know that it must be empty. [3]

- **One solution per span.** A closely related approach is to assume that the optimal global solution is composed of optimal partial solutions, that is, each of the substructures is optimal at its corresponding span. With this assumption, only one partial structure needs to be maintained at each span, the one with the best score. To select it, the strategy will explore all possibilities, by optimizing the best splitting point and the best local phrase dependently:

$$(m, k) = \underset{s \leq m < e,\ k \in \mathcal{K}'}{\arg\max}\ \text{score}(x, y^m_{s,e}) + \text{score}_\text{p}(x, y^m_{s,e}, (s, e)_k)$$

  This strategy considers at each span $(e - s + 1) * |\mathcal{K}'|$ possibilities. Among them, the top-scored partial structure is the one selected for the span. Note that this strategy treats the local scoring dependently of the internal structure. However, extracting features from the internal structure will make the inference approximated.

- **Several solutions per span.** The last strategy is based on maintaining several solutions at each span, under some criterion which permits the overall exploration in polynomial time. The generation of solutions is exhaustive: for each splitting point between s and e, all combinations of solutions from the left and right part are considered, together with a value in $\mathcal{K}'$ for the local phrase. The crucial point in this approach is the criterion for maintaining a limited number of solutions at each span. Several options exist, such as:

  - Beam. Only the top scoring N solutions are maintained. Alternatively, we can select the solutions with score at most δ from the best scored solution.
  - Only positive phrases. Phrases with a non-positive confidence score are discarded. The feasibility of this approach obviously depends on a score_p function which predicts negative scores for non-plausible phrases. If so, we can assume that only a limited number of phrases will score positively, and the space of generated structures is tractable.

[3] However, in this case we may wish to predict the score of a phrase given the neighboring phrases –as discussed below in left-to-right inference.

– φ-equivalence classes. We define that two partial structures are φ-equivalent if their φ structure is the same. We then maintain only the best solution for each φ-equivalence class. For instance, we may define that $\varphi(y)$ is the top-level structure of the partial solution y. With it, consider two different partial structures in a span which share the same local phrase: these structures are φ-equivalent, hence only the top scored one is selected. Note that the top-level structure of a partial solution is not necessarily the local phrase: in partial structures with no local phrase (i.e., null assignment in local phrase), the top-level structure is a mixed sequence of phrases (at a lower level of the span) and input words. This inference strategy, using a notion of φ-equivalence to maintain only the best solution of each class, is guaranteed to be exact as long as the scoring function extracts features *only* from the φ part of a partial solution. This is the type of inference used in parsing with context-free grammar formalisms. There, the dependencies of the data are expressed via rule productions, which relate a constituent –the left-hand side of the rule– with its the top-level internal structure –the right-hand side of the rule.

Left-To-Right Quadratic Inference for Phrase-Based Models

In this inference strategy, the sentence is explored from left to right along words. At each word i, all phrase spans ending at that word are considered, with increasing order of length. The enumeration of phrase spans is as follows:

$$D_{\mathrm{p}}(x) = [\ \underbrace{(1,1)}_{\text{end at } 1}\ ,\underbrace{(2,2),(1,2)}_{\text{end at } 2},\underbrace{(3,3),(2,3),(1,3)}_{\text{end at } 3},\ldots,\underbrace{(n,n),\ldots,(1,n)}_{\text{end at } n}\ ]$$

The scheme is of quadratic cost, since there is a quadratic number of visited spans and the operations at each span will be kept simple enough to be of constant cost. Depending on whether the output phrase structure is sequential or hierarchical, we differentiate two inference algorithms:

Phrase-Based Sequential Inference In this case it is assumed that any subsequence of the optimal phrase sequence is optimal in its context. The strategy maintains a solution at each ending word i, y_i, which is the optimal phrase structure in the $(1, i)$ span. For convenience, we assume an empty solution in y_0. When visiting the ending word i, the strategy searches for the best point j, with $0 \leq j < i$, and the best type $k \in \mathcal{K}'$, such that $\mathrm{score}(x, y_j) + \mathrm{score}_{\mathrm{p}}(x, y_j, (j+1, i)_k)$ is maximum. Note that, here, the scoring of the phrase $(j+1, i)$ receives a partial structure y_j containing phrases to the left of the target phrase, while in CKY-style inference the partial structure was in within. Importantly, conditioning the scoring of a phrase on features extracted from y_j violates the assumption that a partial solution is optimal in its context. So, if there are feature dependencies between a phrase score and its preceding phrases the strategy is an approximation, otherwise it is exact. This

inference procedure is equivalent to the one introduced in Punyakanok and Roth [2001, 2004], where, interestingly, the strategy is described as a reduction to the shortest path algorithm for graphs, where the graph compactly represents all valid phrase structures.

Greedy Inference for Hierarchical Phrase Structures In this case it is assumed that the scoring functions produce a positive score for correct phrases and a negative score for incorrect ones —that is, the score function is assumed to behave as a typical binary classifier. Only one solution is maintained, y, which initially is set to empty. When visiting a phrase span (s, e), first it is checked whether a phrase $(s, e)_k$, for some $k \in \mathcal{K}$, is coherent with y (in terms of non-overlapping). If it is coherent, the type $k \in \mathcal{K}$ with better confidence in $\text{score}_{\text{p}}(x, y, (s, e)_k)$ is selected, as long as the score is positive. If it is not coherent, then it is not considered. Note, therefore, that the exploration does not satisfy the structural dependencies for being exact: deciding for a certain phrase (s, e) invalids all phrases (s', e') such that $s < s' \leq e$ and $e < e'$, which still have not been explored.

Here, when scoring a particular phrase, the solution y will contain the internal structure of the phrase, as well as the phrases to the left of the current phrase.

This strategy is completely greedy, and requires a scoring function which produces little error. In particular, the only recoverable type of error corresponds to predicting a positive score for an incorrect phrase $(s, e)_{\text{k1}}$; the strategy will recover from it as long as there exists a correct phrase in the same span, $(s, e)_{\text{k2}}$, and the correct one receives higher score than the incorrect one.

3.3.3 Inference in a Phrase Recognition Architecture

So far, we have discussed models for phrase recognition based on scoring predictors at word or phrase level, and a variety of inference strategies for them. The difference between inference strategies concerns the robustness of the exploration. On the one hand, exact strategies exploit all possible dependencies established by the model, and guarantee that the predicted global solution is the one receiving highest score, given the model functions. On the other hand, approximated strategies rely on assumptions on the scoring functions to discard partial solutions, before reaching a context at which such solutions can be safely considered non-optimal.

Eventually, the scoring functions are trained from data. As we show in the next section, supervised learning provides techniques to induce a pre-specified behavior to the function being learned. Therefore, the efficacy of a phrase recognition architecture depends on the feasibility to accurately learn scoring predictors that behave as assumed when designing the inference strategy.

For example, assume that the gold behavior of the scoring functions is to predict a positive score for correct local assignments, and a negative score otherwise. If we were able to learn such function, with perfect accuracy, then the

most greedy inference strategy would recover always the correct structure in the most efficient way.

At this point, two questions are of particular interest. First, what level of exploration and robustness is required so that the assumed behavior for the score functions is learnable. Second, once the functions have been learned, which is the improvement, in terms of global accuracy, achieved by robustifying the inference (i.e., to which extent we can recover from local prediction errors).

The following section presents techniques to learn scoring functions for a phrase recognition architecture. The presented questions will be studied empirically in later chapters, experimenting with particular phrase recognition tasks.

3.4 Learning Algorithms for Phrase Recognition

In this section we describe techniques to obtain the learning functions of a phrase recognition architecture, using machine learning.

Each of the learning functions for phrase recognition –such as score_w, score_p, start, end– receive in the input a number of structures and values which, together, represent a learning *instance*. Such input parameters usually include the sentence, some partial structure, and a specific part of the structure to which we are willing to assign a value. The goal of a learning function is to predict a value for the part. We can view this prediction as a classification task, in which the learning function selects the best value for the input instance. Eventually, the learning function receives the instance together with a proposed value, and predicts a confidence score for that assignment, so choosing the best value for the instance corresponds to choosing the value with highest confidence score.

We assume a representation function Φ which takes an instance and extracts features from it. The output of the Φ function is a vector indexed by the feature space: each component of the vector corresponds to one feature, and the value of the component is the value of the feature in a particular instance. In general, the range of each feature is a number in $\mathbb{R}$, although in practice most of the features will be binary-valued. Thus, we assume that learning instances are represented as a feature vector in $\mathbb{R}^n$, where n is the number of features that the Φ function considers.

We restrict our attention to learning functions of the form of linear functions: for each feature, the learning function maintains a weight parameter indicating the contribution of that feature to the prediction. The learning problem is then to estimate the weights of the linear function, in a supervised manner. In the following subsection, we review how classification and ranking learning problems are modeled with linear functions, and we describe the notion of *margin* of a prediction. Then, we present an extended version of the classical Perceptron algorithm, as a simple margin-based learning algorithm to estimate the weights. Finally, we discuss how these techniques can be applied to train the learning components of a phrase recognition architecture.

3.4.1 Linear Functions for Supervised Classification and Ranking

Linear functions are hyperplanes in a $\mathbb{R}^n$ space, parametrized by a weight vector $\mathbf{w}$, specifying the normal vector of the hyperplane, and a bias b, which is the distance from the hyperplane to the origin when $\mathbf{w}$ is normalized. Accordingly, the hypothesis space of linear separators is the set of all hyperplanes in $\mathbb{R}^n$,

$$\mathcal{H} = \{\ h_{\mathbf{w},b} \mid \mathbf{w} \in \mathbb{R}^n,\ b \in \mathbb{R}\ \}$$

A linear function computes the following prediction for an instance $\mathbf{x}$ belonging to $\mathbb{R}^n$:

$$h_{\mathbf{w},b}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b$$

Here, $\langle \mathbf{w}, \mathbf{x} \rangle$ is the inner product, computed as $\sum_{i=1}^{n} \mathbf{w}_i \mathbf{x}_i$, which expresses a similarity score between the input and weight vectors. For simplicity in the notation, sometimes we will be omitting the bias b of a hyperplane, assuming that it is encoded in the weight vector $\mathbf{w}$ as a special component $\mathbf{w}_0 = b$, and that instances have such component set constant to 1, $\mathbf{x}_0 = 1$. Note that the prediction expression is equivalent.

Binary Classification In binary classification the output space is $\mathcal{Y} = \{-1, +1\}$. In this setting, a linear function induces the following classification rule:

$$h_{\mathbf{w}}(\mathbf{x}) = \text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle) = \begin{cases} +1 & \text{if } \langle \mathbf{w}, \mathbf{x} \rangle > 0 \\ -1 & \text{otherwise} \end{cases}$$

Decision Confidence and Margin The decision rule above for binary classification has a clear geometric interpretation. A hyperplane $\mathbf{w}$ is formed by the set of points $\mathbf{z}$ in the Euclidean space $\mathbb{R}^n$ that satisfy $\langle \mathbf{w}, \mathbf{z} \rangle = 0$. Thus, a hyperplane is dividing the input space into two regions or half-spaces: the +1 region, for points falling on the positive side of the hyperplane; and the -1 region, for points falling on the negative side. The set of points lying on the hyperplane is called the *decision boundary*, and arbitrarily takes one of the labels, in our setting -1.

Given an instance $\mathbf{x}$, the quantity $d = \langle \mathbf{w}, \mathbf{x} \rangle$ is geometrically the distance from the hyperplane $\mathbf{w}$ to the instance $\mathbf{x}$. As explained, the sign of d determines the predicted class. It is intuitive to interpret the magnitude of d as a natural measure of *confidence* on that prediction. A closely related concept is that of *margin* of a training example $(\mathbf{x}, y)$, expressed as:

$$\gamma_{\mathbf{w}}(\mathbf{x}, y) = \frac{y\ \langle \mathbf{w}, \mathbf{x} \rangle}{\|\mathbf{w}\|}$$

Note that $\gamma_{\mathbf{w}}(\mathbf{x}, y) > 0$ only when correct classification is produced, and if it is much greater than 0 then the example has been classified correctly with

high confidence. Theoretical work of Vapnik [1998] states that hyperplanes which induce large margins on the training examples have justifiably good generalization properties. This principle, therefore, constitutes a strategy for the design of learning algorithms. The most prominent learning algorithm is Support Vector Machines (SVM), which explicitly seeks for the linear separator that maximizes the margin in a training sample. Cristianini and Shawe-Taylor [2000] or Burges [1998] introduce SVMs and the main results of the learning theory related to margin. Other binary classification learning algorithms, such as the classical Perceptron [Rosenblatt, 1958; Freund and Schapire, 1999] or AdaBoost [Schapire, 2002], have also been explained through margin analysis, though not being originally designed to make margins large.

Ranking functions for Multiclass Output Spaces In multiclass prediction problems, an instance $\mathbf{x}$ in $\mathbb{R}^n$ belongs to a label of a finite set $\mathcal{Y}$ of cardinality k. For convenience, we assume that the labels are $\mathcal{Y} = \{1, \ldots, k\}$. It is usual to implement a multiclass prediction function with the so-called *winner-take-all* scheme, consisting of k linear functions in $\mathbb{R}^n$, one for each label, $\mathbf{w}^1$ to $\mathbf{w}^k$. Then, the multiclass function follows the form:

$$h_{\mathbf{w}^{1:k}}(\mathbf{x}) = \arg\max_{i \in \mathcal{Y}} \langle \mathbf{w}^i, \mathbf{x} \rangle$$

That is, the predicted label is the one whose corresponding weight vector achieves the highest similarity score. The confidence of a prediction increases with the difference between the score of the correct label and the score of the rest of the labels. Consequently, the margin of the prediction is defined as the difference between the score of the correct label and the maximum among the scores of the rest of he labels. Formally, if $(\mathbf{x}, y)$ is an example, the margin is:

$$\gamma_{\mathbf{w}^{1:k}}(\mathbf{x}, y) = \langle \mathbf{w}^y, \mathbf{x} \rangle - \max_{i \neq y} \langle \mathbf{w}^i, \mathbf{x} \rangle$$

As a note, the k weight vectors $\mathbf{w}$ in $\mathbb{R}^n$ can be seen as a single global vector in $\mathbb{R}^{kn}$, corresponding to the concatenation of the k vectors. With these definitions of a multiclass function and the margin of a prediction, there exist learning algorithms to train the weights of the multiclass function so that margins in a sample are positive [Crammer and Singer, 2003b; Har-Peled et al., 2002]), or maximum [Crammer and Singer, 2001].

Ranking functions for Complex Output Spaces A more complex level of prediction problem is found when the output space is structured. This is the case of the problems we are dealing with in this thesis, in which the input space corresponds to sequences of words which form sentences, and the output space are phrase structures in a sentence of some nature. In this scenario, the output space is of exponential size with respect to the length of the input sentence. First, it is infeasible to enumerate the possible values in the output space $\mathcal{Y}$. But also, the output structures have a clear recursive nature, and therefore it

is inadequate to treat two structures with common substructure as completely different output values.

In the general setting of complex spaces, we rely on a representation function $\Phi(x, y)$ which takes an input instance x together with an structured output value y and produces a feature vector in $\mathbb{R}^n$. Then, we define a linear function parametrized by $\mathbf{w}$ which ranks input-output pairs. The resulting function computes a prediction as:

$$h_{\mathbf{w},\Phi}(x) = \arg\max_{y \in \mathcal{Y}} \langle \mathbf{w}, \Phi(x, y) \rangle$$

Similarly to the multiclass setting, the predicted value is the one which maximizes the score. However, due to the exponential size, the learning models here rely on properties of the structures of $\mathcal{Y}$ and the representations induced by Φ to design efficient inference algorithms which select the top-ranked solution in polynomial time (e.g., dynamic programming algorithms). The margin of a prediction for an example (x, y) is defined as:

$$\gamma_{\mathbf{w},\Phi}(x, y) = \langle \mathbf{w}, \Phi(x, y) \rangle - \max_{y' \in \mathcal{Y}, y' \neq y} \langle \mathbf{w}, \Phi(x, y') \rangle$$

To our knowledge, margin-based learning for structured domains was first introduced by Collins [2004, 2002], which proposed a Perceptron algorithm to train the parameters. Recently, maximum-margin algorithms for this scenario have also been proposed [Taskar et al., 2003, 2004; Tsochantaridis et al., 2004].

3.4.2 Perceptron Algorithms

The Perceptron learning algorithm is one of the simplest algorithms to train a linear function. It was introduced by Rosenblatt [1958] for binary classification. We show how the original Perceptron is extended to multiclass and structured learning scenarios, following respectively Crammer and Singer [2003b] and Collins [2002].

Perceptron is an online learning algorithm that, supervisedly, trains the weights of a linear function so that it makes no error on the training sample. In other words, the learning bias of the Perceptron is to make prediction margins of the training examples positive. It can be proved that if the training sample is linearly separable (i.e., there exists a vector of parameters that makes all prediction margins positive), then the Perceptron algorithm will converge.

As an online algorithm, the type of update it performs is of additive nature. Perceptron is conceptually simple and easy to implement. Also, it is a kernel method, so it can benefit from a kernel function to implicitly induce non-linear separators, or exploit the recursiveness of the structures.

Basic Perceptron

In binary classification, where $\mathcal{Y} = \{+1, -1\}$, the Perceptron algorithm works as follows. It starts with the weight vector $\mathbf{w}$ arbitrarily initialized to 0. Then, it

visits examples in the training set sequentially, one at a time. Given an example $(\mathbf{x}, y)$, its prediction is computed using the current vector, as $\hat{y} = \text{sign}\langle \mathbf{w}, \mathbf{x} \rangle$. If the predicted value is not correct, the vector is updated additively: $\mathbf{w} = \mathbf{w} + y\mathbf{x}$. This update rule encodes the two types of errors which are possible in this scenario. If the example was positive but the prediction was negative, the weight vector is moved toward that example, *promoting* its weight. On the other hand, if the example was negative but the prediction positive, the weight vector is moved away from the example, *demoting* its weight.

Generalized Perceptron

We now describe a generalized version of the Perceptron which applies to binary, multiclass and structured learning scenarios. Figure 3.4 shows the pseudo-code of the algorithm. In the extension setting, instances x belong to some space $\mathcal{X}$ and are to be classified into an output space $\mathcal{Y}$. We then assume a representation function Φ which takes a pair (x, y) and outputs a feature vector in $\mathbb{R}^n$. The prediction function, parametrized by a weight vector $\mathbf{w} \in \mathbb{R}^n$, is of the form: $h_{\mathbf{w}}(x) = \arg\max_{y \in \mathcal{Y}} \langle \mathbf{w}, \Phi(x, y) \rangle$. The learning problem is to train the $\mathbf{w}$ parameters with training data. As in the original Perceptron, the algorithm visits examples online. At a given example (x, y) it first computes the prediction $\hat{y} = h_{\mathbf{w}}(x)$, and if it is not correct the parameter vector is updated. The update rule consists of promoting the correct solution y and demoting the predicted solution $\hat{y}$.

Perceptron for Binary Classification We fix now the setting to be binary classification, with $\mathcal{Y} = \{+1, -1\}$. Following Crammer and Singer [2003b], we define the learning components so as to show that the generalized Perceptron can be particularized to the above-described original Perceptron. For convenience, we assume instances $\mathbf{x}$ to be in $\mathbb{R}^n$. The $\mathbf{w}$ vector will be composed of $2n$ dimensions, as $\mathbf{w} = (\mathbf{w}_{+1}, \mathbf{w}_{-1})$, where both $\mathbf{w}_{+1}$ and $\mathbf{w}_{-1}$ are in $\mathbb{R}^n$. Moreover, it will be accomplished that $\mathbf{w}_{-1} = -\mathbf{w}_{+1}$. Finally, the function $\Phi(x, y)$ embeds the vector $\mathbf{x} \in \mathbb{R}^n$ in a $2n$-dimensional space: if $y = +1$ it outputs $(\mathbf{x}, 0^n)$, otherwise it outputs $(0^n, \mathbf{x})$. With this definitions, it is easy to see that the extended Perceptron behaves as the original version for binary classification, the only difference being that in the extended version we redundantly maintain two parameter vectors of n dimensions.

Perceptron for Multiclass Classification We now move to a multiclass scenario, with $\mathcal{Y} = \{1, \ldots, k\}$. Again, we assume instances $\mathbf{x} \in \mathbb{R}^n$. The $\mathbf{w}$ vector is composed of kn dimensions, maintaining a chunk of n dimensions for each label. Similarly to the binary case, the Φ function embeds a pair $(\mathbf{x}, y)$ into a kn-dimensional vector, mapping the x vector into the corresponding y-th chunk of dimensions, $\Phi(\mathbf{x}, y) = (0^{(y-1)n}, x, 0^{(k-y)n}) \in \mathbb{R}^{kn}$. In this case, the behavior of the update rule is as follows. Suppose that for an instance $\mathbf{x}$ the $\hat{y}$ label is predicted, while y is the correct one. This implies that $\hat{y}$ received more weight than y. Thus, the update rule demotes the parameters corresponding to

$\hat{y}$ and promotes the ones corresponding to y. In the multiclass scenario, there exist other ways of updating the parameter vector, in which not only the top-ranked incorrect label is demoted, but all labels which receive a similarity score higher than the score of the correct label [Crammer and Singer, 2003b].

Perceptron for Structured Classification When the output space $\mathcal{Y}$ is complex and structured, the learning mechanism of the Perceptron is the same: it visits examples so as to find incorrect predictions, and corrects the parameter vector on each error with a pair of promotion-demotion additive updates. Collins [2002, 2004] shows that the key point which makes learning possible is that the representation defined by the model –in terms of features of Φ and the corresponding parameters $\mathbf{w}$– allows efficient inference to compute the arg max . This property can be accomplished by defining a representation based on the parts of a decomposed solution, as discussed in Section 3.2, and use the inference algorithms of Section 3.3. Let $D(x)$ be a decomposition of a sentence x into parts. Let ϕ be a feature extraction function representing a part p_t of a solution y, and let y_t be the value of the part in the solution. The global feature function is defined as:

$$\Phi(x, y) = \sum_{p_t \in D(x)} \phi(x, y, p_t, y_t)$$

The global vector $\mathbf{w}$ has a weight parameter for each feature of Φ (or ϕ). Thus, shared parts of two different solutions share also the same parameters in the model (as opposed to the multiclass version, where each label has its own parameters). The inference process is then:

$$h_w(x) = \arg\max_{y \in \mathcal{Y}} \langle \mathbf{w}, \Phi(x, y) \rangle = \arg\max_{y \in \mathcal{Y}} \sum_{p_t \in D(x)} \langle \mathbf{w}, \phi(x, y, p_t, y_t) \rangle$$

Extensions for Perceptron

In this section we present two extensions that can be incorporated to Perceptron. The first is about averaging the predictions of Perceptron, that has been shown to increase robustness. The second concerns the use of kernel functions.

Voted Perceptron Freund and Schapire [1999] introduced the *Voted Perceptron* algorithm, a modified version of the original algorithm that uses votes to make predictions more robust. The key point of the voted version is that, while training, it stores information in order to make more robust predictions on test data. Specifically, each prediction vector $\mathbf{w}^j$ generated after every mistake is stored, together with a weight c^j that is set during training. In particular, when a vector is generated its weight is set to one, and for each correct prediction it makes its weight is incremented by one. In other words, the weight of a vector counts how many examples the vector *survives* until a mistake is committed. Let J be the total number of vectors that a Perceptron accumulates. The final

Input: A labeled training set $\{(x_1, y_1), \ldots, (x_m, y_m)\}$,
with $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$
A feature extraction function $\Phi : \mathcal{X} \times \mathcal{Y} \rightarrow I\!R^n$
Output: The weight vector $\mathbf{w} \in I\!R^n$

initialize: $\mathbf{w} = 0$;
repeat
for $i = 1, \ldots, m$ **do**:
$\hat{y} = h_{\mathbf{w}}(x_i) = \arg\max_{y \in \mathcal{Y}} \langle \mathbf{w}^i, \Phi(x, y) \rangle$
if $\hat{y} \neq y$ **then**
$\mathbf{w} = \mathbf{w} + \Phi(x_i, y_i) - \Phi(x_i, \hat{y})$
until no changes in $\mathbf{w}$ during the epoch

Figure 3.4: The Generalized Perceptron Learning Algorithm

hypothesis is an averaged vote over the predictions of each vector, computed with the expression:

$$h_{\mathbf{w}}(\mathbf{x}) = \sum_{j=1}^{J} c^j (\mathbf{w}^j \cdot \mathbf{x})$$

This expression corresponds to the *averaged* prediction method. The authors also propose a *voted* prediction method which takes the sign of each $\mathbf{w}^j \cdot \mathbf{x}$, instead of the real value. In the experimental section we show that both methods substantially outperform the unweighted prediction expression of the Perceptron (denoted *last* by the authors, since it only makes use of the last vector, $\mathbf{w}^J$).

Dual Formulation and Kernels Freund and Schapire [1999] show that in Perceptron the weight vector $\mathbf{w}$ can be expressed as a linear combination of the training instances that were added or subtracted during training. Consider a training sample of m examples, each of the form (x_i, y_i). Let α_i be a variable counting how many times Perceptron updated the weight vector using the example x_i, and note that such updates are either additions or subtractions depending on the sign y_i. It follows that:

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i$$

The weight vector is called the *primal form* of the linear hyperplane. On the other hand, the vector of α_i's, equivalently expressing the same linear hyperplane, is called the *dual form* of the hyperplane. Due to this equivalence, learning can be thought either as estimating the weights in $\mathbf{w}$ or the dual variables α_i.

An important result of the research related to SVMs is the use of *kernels* when working with the dual form. A kernel function is a function efficiently computing the inner product between two instances in an extended feature space. The motivation for using kernels is that in the extended space the data might be better separated. For example, there are kernel functions that induce a non-linear transformation.

When a learning algorithm, working in dual form, uses a kernel function, it is operating in the extended feature space. Thus, though estimating a linear number of parameters (i.e., the α_i's), it might learn a non-linear separator in the original space.

The class of learning methods that admits kernels is known as *kernel methods* [Schölkopf and Smola, 2002], and Perceptron and SVMs belong to it.

Kernel methods are particularly interesting for structured domains, such as phrase recognition tasks, since *convolution kernels* allow to work with exponential feature spaces consisting of all subsequences or trees [Haussler, 1999; Collins and Duffy, 2002] .

3.4.3 Learning in a Phrase Recognition Architecture

In the current section we have presented a range of learning techniques to learn a linear function from data, all of them designed through the concept of margin of training examples. Here, we briefly discuss different learning strategies for training the functions of a phrase recognition architecture. In later chapters, these strategies will be discussed more technically, and will be empirically tested on particular architectures and natural language problems.

In this discussion, we will assume that the model of the architecture is fixed, and that the feature extraction function needed by the predicting functions is also defined. Thus, from a learning point of view, the representation of solutions and the parameters of the architecture will be the same in all cases. For instance, consider that the problem is to train a phrase-based architecture, with a single predicting function to score phrases. We assume the existence of a training set consisting of a collection of sentences, each paired with its correct phrase structure, referred to as *gold structure*. The learning strategy defines how to train the functions of the architecture so that for each training sentence the gold structure is optimal according to the model (with generalization guarantees on it). Still, if the inference strategy is not exact (and, thus, the selected solution may not be optimal) we may wish to train the functions according to the inference strategy, so that the output of the inference is the correct solution. Thus, an important aspect of the learning strategy is to enforce a behavior to the learning functions that respects the assumptions of the inference strategy. Related to learning scenarios for linear functions, we differentiate three alternative ways of modeling a score function.

First, one can think of having k binary classifiers, each associated to a class in $\mathcal{K}$, deciding whether a certain span is a phrase of the corresponding type or not. In this case, each classifier is independent from the others. For training them, a training set is derived for each one from the collection of sentences. In

particular, for a class k, phrase spans in the collection which are of class k are positive examples for the classifier, while all other phrase spans are negative examples. At prediction time, the magnitude of a prediction –whose sign determines positive or negative classification– provides the confidence measure of the score function. This approach is known as the *one-versus-all* strategy, since for each class we are willing to learn a linear separator between the class' examples and the rest of them. Note that, for the ultimate goal of the architecture, this assumption is quite strong. Looking at any of the inference mechanisms, to achieve correct recognition it suffices that the correct class receives higher score than any of the competing classes, rather than a positive score for the former and negative scores for the laters.

As a second choice, one can think of the score function as a multiclass classifier, predicting a confidence score for each class in $\mathcal{K}'$, but also selecting which is the most plausible class –namely, the top-scored class. In this case, the training set consists of the phrase spans of the sentence collection, each paired with the class of span. Note that one of the classes here will be the null class, indicating that the span is not a phrase. Although there exist many valid techniques for learning a multiclass classifier, a reasonable choice is the Perceptron approach for multiclass classification discussed above. There, $|\mathcal{K}'|$ linear separators –one for each class– are trained dependently to make the prediction of the correct class higher than any other, but without enforcing a negative score for incorrect classes. Thus, the one-vs-all limitation is naturally solved. Still, a score function modeled as a multiclass classifier is independent from the particular architecture. Specifically, the inference strategy selects a local assignment not only on the basis of its confidence score, but also for its coherence within a globally scored structure. These interactions, at the global context of the sentence, are not considered in multiclass training.

Finally, the score function can be globally modeled at sentence level. As shown with Perceptron, the score function at sentence level corresponds to a composition of score functions at phrase level. In particular, the score of a structure is the addition of scores of the phrases it contains. The parameters of the score function at sentence level are the same than the parameters of the score function at phrase level. Because of this property, such parameters can be trained at global level. In this setting, thus, an example is directly a sentence with its phrase structure. Recall that this learning strategy requires an inference algorithm to recover the top-scored solution of a sentence, for some value of the parameters being trained. Obviously, the inference algorithm is that of the phrase recognition architecture. Thus, in this setting the scoring functions are trained dependently of the inference.

3.5 Summary

In this chapter, we have discussed techniques to implement the main components of a phrase recognition architecture. We have presented models that put learning at word or phrase levels. Then, we have discussed incremental in-

ference strategies for such models, that go from greedy to robust explorations of the output space. Here, the success of a greedy search will depend on the success of predictors at guessing the correct values for each decision. Finally, we have presented classification and ranking learning scenarios, and we have shown how Perceptron can be used to train linear functions in such scenarios. We have also discussed the application of these learning techniques to train a phrase recognition architecture, yielding training strategies that are either local or global. Overall, we have discussed a number of choices in the design of a phrase recognition system.

The next chapter introduces a learning architecture that can be though as a particular choice within this framework for phrase recognition. It also presents a global training algorithm for the architecture, and illustrates the empirical behavior of it, contrasted with that of a local training strategy.

Chapter 4

A Filtering-Ranking Learning Architecture

This chapter proposes a novel learning architecture to recognize a set of phrases in a sentence. We name it Filtering-Ranking Architecture.

The chapter is organized in three sections. The following section describes the filtering-ranking architecture (model and inference). Next, Section 4.2 presents FR-Perceptron, a global learning algorithm to train the learning functions of the architecture. Finally, Section 4.3 presents extensive experiments on a Partial Parsing task, namely Clause Identification, with the aim of showing the behavior of FR-Perceptron on real data. We give evidence that a global learning strategy is much better than a local approach, at least with our filtering-ranking architecture. Then, in Chapter 5 we describe in detail three applications of the filtering-ranking architecture to phrase recognition tasks, and contrast the results of FR-Perceptron with other systems in the literature.

4.1 Filtering-Ranking Architecture

This section describes a phrase recognition architecture based on filters and rankers. The decomposition of the problem has two layers of processing: filtering, operating at word level, and ranking, operating at phrase level. The filtering layer identifies plausible phrase candidates with *start-end* decisions. The ranking layer scores phrases and builds the best phrase set for the sentence. We first present the model that decomposes the global problem into many decisions. Then, we discuss the inference strategy that computes the best structure for a sentence. Finally, we define how the learning functions of the architecture are implemented, and which are the parameters that a learning algorithm has to estimate to train the architecture.

Notation. Let $\mathcal{X}$ be the input space of sentences, and $\mathcal{Y}$ be the output space of possible phrase structures. Let $\mathcal{P}$ be the space of possible phrases. We define

$\mathcal{Y}$ as the *sets* of phrases in $\mathcal{P}$ that form a coherent structure. This definition allows either sequential or hierarchical structures, and affects only the type of inference performed (see a more formal definition in Section 3.1). For simplicity in notation, we will assume that $\mathcal{P}$ and $\mathcal{Y}$ are restricted to phrases and structures that do not fall outside a given input sentence $x \in \mathcal{X}$.

4.1.1 Model

The model for recognizing phrases is described as a function $\mathrm{R} : \mathcal{X} \rightarrow \mathcal{Y}$ which, given a sentence $x \in \mathcal{X}$, identifies a phrase structure $y \in \mathcal{Y}$ for x. We define two components within the R function, both being learning components of the recognizer. First, we define a filtering function F which, given a sentence x, identifies a set of candidate phrases, not necessarily coherent, for the sentence, $\mathrm{F}(x) \subseteq \mathcal{P}$. Second, we define a *score* function which, given a phrase, produces a real-valued prediction indicating the plausibility of the phrase in its context. Thus, the Phrase Recognizer (R) is a function that searches a coherent phrase set for a sentence x according to the following optimality criterion:

$$\mathrm{R}(x) = \arg \max_{y \subseteq \mathrm{F}(x) \ \mid \ y \in \mathcal{Y}} \sum_{(s,e)_k \in y} \mathrm{score}(x, y, (s,e)_k) \tag{4.1}$$

That is, the global score of a solution y is the summation of scores of the phrases it contains. The *score* function predicts the score of a phrase in the context of a sentence x and a solution y.

The function F is only used to reduce the search space of the R function. Note that the R function constructs the optimal phrase set by evaluating scores of phrase candidates, and, that there is a quadratic number of possible phrases with respect to the length of the sentence (that is, the size of $\mathcal{P}$ set). Thus, considering straightforwardly all phrases in $\mathcal{P}$ would result in a very expensive exploration. We propose a particular F function aiming at substantially reducing phrase candidates by applying *start-end* decisions at word level. That is, we assume two classifiers, namely *start* and *end*, that predict whether a word starts or ends, respectively, a phrase of type $k \in \mathcal{K}$ in the sentence. With these classifiers, the filtering function can be expressed as:

$$\mathrm{F}(x) = \{\, (s,e)_k \in \mathcal{P} \mid \mathrm{start}(x,s,k) = +1 \ \wedge \ \mathrm{end}(x,e,k) = +1 \,\}$$

In total, the architecture is composed of three learning functions, namely the pair of *start-end* filters and the *score* function.

4.1.2 Inference

Inference is the process that efficiently searches the optimal phrase structure for a sentence, according to the optimality criterion 4.1. Here, we briefly comment an incremental inference process for a filtering-ranking decomposition. A more detailed look at inference strategies for phrase-based models is found in Section 3.3.2.

The filtering functions depend only on the sentence x. Thus, the set of candidate phrases for a sentence can be obtained independently of the ranking process. For *start-end* filtering, the process corresponds to a sequential prediction along the words of the sentence. In the most simpler version, the words of the sentence are processed from left-to-right. At the i-th word, the *start-end* functions are applied for each phrase type. It is possible to exploit dependencies between *start-end* predictions at different words, and then use more robust sequential inference techniques (see Section 3.3.1). After this processing, the classification outcomes (for each word and type) determine the set of phrase candidates: each k-*start* word x_s paired with each k-*end* word x_e, having $s \leq e$, form the phrase candidate $(s, e)_k$.

Ranking, then, consists of building the top-scored structure made of phrase candidates. To do so, each phrase candidate is visited and scored, in some particular order that permits inference. Depending on the complexity of phrase structures, we differentiate two strategies:

- Phrase Sequences. Phrases are visited by increasing order of ending word. At each ending word, the optimal phrase structure from the beginning of the sentence to that word is computed, considering coherent combinations of optimal phrase structures at previous words (recursive step) with phrases ending at the current word (local step). The process is of quadratic cost.
- Phrase Hierarchies: Phrases are visited from the bottom-up, that is, by increasing length of the phrase span. In a span, the optimal phrase hierarchy for the span is computed, considering coherent combinations of hierarchies found within the span (recursive step) with the phrase that covers the whole span (local step). The process is of cubic cost.

The main assumption of these inference strategies is that a certain prediction is independent of other predictions. In practice, this is not true in the experimentation with real data. On the one hand, the *start-end* functions make use of the classifications that have been predicted on previous words. On the other hand, the *score* function makes use of the the solution y that contains the phrase to be scored in order to extract contextual features, and y is in turn determined by earlier predictions of the *score* function. So, in practice, the inference strategies we use are approximated.

An Example of Filtering-Ranking Recognition

Figure 4.1 depicts an schematic example of how the process of recognizing a hierarchical structure of phrases works, considering generic phrases without type for simplicity. In this example, we assume a correct solution and values for the predictions of the learning functions made during the process.

The input sentence is represented at the bottom as a sequence of words $x_1 \dots x_{15}$, each being a small black circle. The correct phrase structure of the sentence is represented as a bracketing along the words, and contains the following phrases:

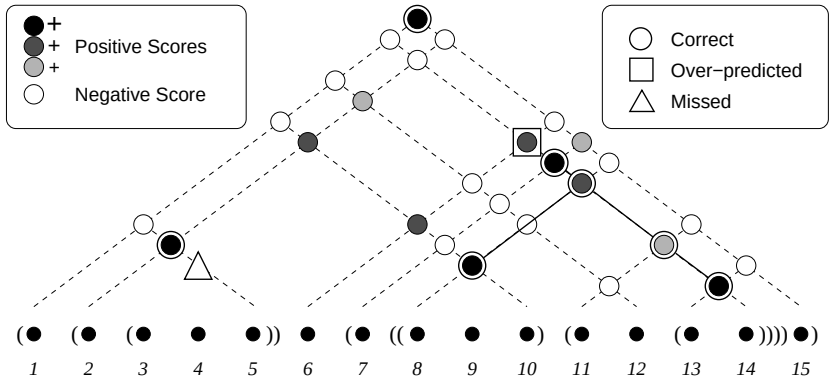

Figure 4.1: Schematic example of the filtering-ranking recognition strategy.

$$(1,15)\ (2,5)\ (3,5)\ (7,14)\ (8,10)\ (8,14)\ (11,14)\ (13,14)$$

To recognize phrases, first the *start-end* functions are applied to each word. In the figure, an oblique dashed line indicates each positive classification: for words predicted as *start* (x_1, x_2, x_6, x_7, x_8, x_{11}, x_{13}) the line goes to the right, whereas for words considered *end* (x_5, x_{10}, x_{12}, x_{14}, x_{15}) the line goes to the left. Note that these local predictions produce errors, such as the missing start at x_3. The intersections of dashed lines correspond to phrase candidates, printed as circles. For instance, the start word x_7 together with the end word x_{14} forms the phrase candidate $(7, 14)$. Note that in the filtered space there are only 27 phrase candidates out of the 120 possible phrases.

After filtering, the *score* function is applied to phrase candidates. The grey scale of circles indicates the prediction to each candidate: white indicates negative predictions (candidates to be rejected), while greys indicate three positive degrees of confidence for phrases. The sentence is processed by visiting phrase candidates from the bottom-up and, while predicting confidence values for candidates, the optimal structure at the explored region is maintained. Thus, the *score* applied to a phrase can take advantage of the predicted solution found within the phrase. For instance, when scoring the phrase $(7, 14)$ a hierarchy of four phrases is found inside (marked in the figure with a solid line that roots the hierarchy to the phrase). This fact allows to work with representations that exploit patterns of the internal structure of a phrase, possibly by means of kernel functions.

After scoring all candidates and completing the sentence exploration, the global optimal structure is also built. The phrase structure that maximizes confidence scores consists of the following phrases:

$$(1,15)\ (2,5)\ (6,14)\ (7,14)\ (8,10)\ (8,14)\ (11,14)\ (13,14)$$

In the figure, the predicted phrases that are correct are marked with a circle, while there is only one predicted phrases that is not correct, namely $(6, 14)$, and

is marked with a square. The precision of the prediction for the sentence is 7/8. There is also one correct phrase that is missed in the solution, namely $(3, 5)$, appearing as a triangle. Recall is thus 7/8. Note that among all local errors produced in the process by the three functions (e.g. predicting x_{12} as *end*, or giving $(2, 10)$ a positive confidence), only three are critical at propagating to the global solution: predicting x_3 as *not start*, predicting x_6 as *start*, and giving $(6, 14)$ a positive score.

4.1.3 Learning Components of the Architecture

The learning functions of the architecture are implemented with linear separators on a feature space defined by a representation function (see Section 3.4.1 for a detailed description of linear separators).

The *filtering* function (F) is composed of two classifiers, *start* and *end*, each of which predicts whether a word x_i starts (S) or ends (E) a phrase of type k, respectively. We implement the F function with a *start-end* pair of prediction vectors for each phrase type $k \in \mathcal{K}$, noted as $\mathbf{w}^k_{\mathrm{S}}$ or $\mathbf{w}^k_{\mathrm{E}}$, and a unique shared representation function ϕ_{w} which maps a word in context into a feature vector. A classifier prediction on a word x_i for a type k is computed as $\mathrm{start}(x, i, k) = \mathbf{w}^k_{\mathrm{S}} \cdot \phi_{\mathrm{w}}(x, i)$, and similarly for $\mathrm{end}(x, i, k)$, and the sign is taken as the binary decision.

The *score* function computes a real-valued score for a phrase candidate $(s, e)_k$. We implement this function with a prediction vector $\mathbf{w}^k$ for each type $k \in \mathcal{K}$, and also a shared representation function ϕ_{p} which maps a phrase into a feature vector. The score prediction is then given by the expression: $\mathrm{score}(x, y, (s, e)_k) = \mathbf{w}^k \cdot \phi_{\mathrm{p}}(x, y, (s, e)_k)$.

4.2 Filtering-Ranking Perceptron

Training the learning functions of the phrase recognizer, namely the *start-end* and the *score* functions, presents the following challenges:

- The learning algorithm should optimize the sentence-level F_1 measure, rather than the accuracies of local decisions.
- During the recognition process, the functions interact. First, the *start-end* functions define the actual input space of the *score* function. Second, in the general case (hierarchical structures) the *score* function is applied recursively on its own recognized structure (i.e., a solution y is built from the bottom-up by adding new phrases). Thus, the learning strategy should take into account such interactions.
- The *score* function operates at phase level, predicting confidence scores for phrase candidates. There is a quadratic number of phrase candidates over the sentence length. Thus, given a dataset of real size, generating all possible phrase candidates produces a vast amount of examples, most

algorithm FR-Perceptron
  input: a training set $S = \{(x^i, y^i)\}_{i=1}^m$
  define: $W = \{\mathbf{w}_{\mathrm{S}}^k, \mathbf{w}_{\mathrm{E}}^k, \mathbf{w}^k \mid k \in \mathcal{K}\}$
  initialize: $\forall \mathbf{w} \in W \ \mathbf{w} := \mathbf{0}$
  for $t = 1 \ldots T$
    for $i = 1 \ldots m$
      $\hat{y} := \mathrm{R}_W(x^i)$
      recognition_learning_feedback$(W, x^i, y^i, \hat{y})$
  output: the vectors in W
end-algorithm

Figure 4.2: Pseudocode for the FR-Perceptron algorithm

of them being negative candidates. So, the algorithm should be efficient at training the *score* function in terms of the number of phrase examples considered.

In this section we introduce a mistake-driven online learning algorithm for training the parameter vectors of the Phrase Recognizer. It is a global algorithm that works at sentence level, and concentrates on the global accuracy of the predicted structures. The parameter vectors are trained all together, aiming to capture their interactions to cope with the above challenges. We name the algorithm FR-Perceptron, since it is a Perceptron-based learning algorithm that approximates the prediction vectors in F as *Filters* of words, and the score vectors as *Rankers* of phrases.

4.2.1 The Algorithm

The algorithm, presented in Figure 4.2, is a generalization of the traditional Perceptron (see Section 3.4.2 for a review of Perceptron and generalizations). It works as follows. It starts with all vectors initialized to $\mathbf{0}$, and then runs repeatedly in a number of epochs T through all the sentences in the training set. Given a sentence, it predicts its optimal phrase solution as specified in (4.1) using the current vectors. As in the traditional Perceptron, if the predicted phrase structure is not perfect the vectors responsible of the incorrect prediction are updated additively within a feedback rule. We propose a feedback rule specially tailored for recognizing phrases with filtering-ranking functions. The next section describes the feedback rule.

4.2.2 Filtering-Ranking Recognition Feedback

The recognition-based feedback is described as a function that receives four parameters in the input, namely the set of weight vectors W, a sentence x, and the correct y and predicted $\hat{y}$ phrase structures on the sentence. The rule updates the vectors in W according to the errors of $\hat{y}$ contrasted with y.

Notation. Let $\hat{y}_{\mathrm{S}}^{k,i}$ and $\hat{y}_{\mathrm{E}}^{k,i}$ stand respectively for the local predictions of *start-end* functions on word i and type k, at prediction time. Let $\hat{y}_{s,e}^{k}$ be the prediction of the *score* function on phrase candidate $(s,e)_k$ at prediction time. Assume similar variables for the gold structure y, the values for them being perfect $\{+1,-1\}$-predictions indicating whether a word starts/ends a phrase, or whether a phrase belongs to the solution.

The function is defined by cases as follows:

1. Phrases correctly identified: $\forall (s,e)_k \in \ y \cap \hat{y}$:
   - Do nothing, since they are correct.
2. Missed phrases: $\forall (s,e)_k \in \ y \setminus \hat{y}$:
   - Update misclassified boundary words (type-A *promotion* updates):
     if $(\hat{y}_{\mathrm{S}}^{s,k} \leq 0)$ then
     $\mathbf{w}_{\mathrm{S}}^{k} := \mathbf{w}_{\mathrm{S}}^{k} + \phi_{\mathrm{w}}(x,s)$
     if $(\hat{y}_{\mathrm{E}}^{e,k} \leq 0)$ then
     $\mathbf{w}_{\mathrm{E}}^{k} := \mathbf{w}_{\mathrm{E}}^{k} + \phi_{\mathrm{w}}(x,e)$
   - Update score function, if applied:
     if $(\hat{y}_{\mathrm{S}}^{s,k} > 0 \ \wedge \hat{y}_{\mathrm{E}}^{e,k} > 0)$ then
     $\mathbf{w}^{k} := \mathbf{w}^{k} + \phi_{\mathrm{p}}(x,y,(s,e)_k)$
3. Over-predicted phrases: $\forall (s,e)_k \in \ \hat{y} \setminus y$:
   - Update score function:
     $\mathbf{w}^{k} := \mathbf{w}^{k} - \phi_{\mathrm{p}}(x,\hat{y},(s,e)_k)$
   - Update words misclassified as S or E (type-B *demotion* updates):
     if $(y_{\mathrm{S}}^{s,k} = -1)$ then
     $\mathbf{w}_{\mathrm{S}}^{k} := \mathbf{w}_{\mathrm{S}}^{k} - \phi_{\mathrm{w}}(x,s)$
     if $(y_{\mathrm{E}}^{e,k} = -1)$ then
     $\mathbf{w}_{\mathrm{E}}^{k} := \mathbf{w}_{\mathrm{E}}^{k} - \phi_{\mathrm{w}}(x,e)$

The updates of the *start-end* vectors are performed only once per word, although an incorrect prediction on a word may generate several incorrect phrases.

This feedback rule has been derived by analyzing the dependencies between each function and a global solution, and naturally fits the phrase recognition setting. In particular, the rule tracks the interaction between the two layers of the recognition process as follows. The *start-end* layer filters out phrase

candidates for the scoring layer. Thus, misclassifying the boundary words of a correct phrase blocks the generation of the candidate and produces a missed phrase. In this case, we move the *start* or *end* prediction vectors toward the misclassified boundary words of a missed phrase. When an incorrect phrase is predicted, we move away the prediction vectors from the *start* or *end* words, provided that they are not boundary words of a phrase in the correct solution. Note that we deliberately do not care about false positives on the filtering layer which do not finally over-produce a phrase, since these local errors do not hurt the global performance of the recognizer. This fact is crucial at explaining the filtering behavior of the *Start-End* layer.

Regarding the *Score* layer, each category prediction vector is moved toward missed phrases and moved away from over-predicted phrases. It is important to note that the feedback rule operates only on the basis of the global predicted solution $\hat{y}$, avoiding to update the *score* function on each of its local predictions on phrase candidates. This fact has the effect of approximating the behavior of the *score* function as a *ranker* over the set of candidate phrases, assigning higher predictions to the phrases in the solution than to the other incorrect competing phrases. As a consequence, this simple feedback rule tends to approximate the desired behavior of the global R function, that is, to make the summation of the scores of the correct phrase set maximal with respect to other phrase set candidates. In the experimental section of this chapter, we show this effect in the context of clause identification.

4.2.3 Binary Classification Feedback

Before discussing the FR-Perceptron algorithm with the proposed feedback rule, we want to describe an alternative feedback rule that updates the prediction vectors to behave as binary classifiers. That is, the *start-end* functions are expected to predict positive magnitudes only for correct boundaries, and the *score* function is expected to predict positive magnitudes only for correct phrases. The feedback rule supervises each local prediction that has been computed when processing a training sentence, and updates additively the prediction errors. Assuming the same notation that in the previous section, the feedback rule is expressed as follows:

1. Update incorrect *start* predictions:
   - foreach i, k such that $(y_{\mathrm{S}}^{k,i}\ \hat{y}_{\mathrm{S}}^{k,i} \leq 0)$ do
     $\mathbf{w}_{\mathrm{S}}^{k} := \mathbf{w}_{\mathrm{S}}^{k} + y_{\mathrm{S}}^{k,i}\phi_{\mathrm{w}}(x,i)$
2. Update incorrect *end* predictions:
   - foreach i, k such that $(y_{\mathrm{E}}^{k,i}\ \hat{y}_{\mathrm{E}}^{k,i} \leq 0)$ do
     $\mathbf{w}_{\mathrm{E}}^{k} := \mathbf{w}_{\mathrm{E}}^{k} + y_{\mathrm{E}}^{k,i}\phi_{\mathrm{w}}(x,i)$
3. Update wrong predictions on phrases that have passed the filter:
   - foreach $(s,e)_k \in \mathrm{F}(x)$ do

– if ($\hat{y}^k_{s,e} \leq 0 \ \wedge \ (s,e)_k \in y$))
$$\mathbf{w}^k := \mathbf{w}^k + \phi_{\mathrm{p}}(x, y, (s,e)_k)$$
– else if ($\hat{y}^k_{s,e} > 0 \ \wedge \ (s,e)_k \notin y$))
$$\mathbf{w}^k := \mathbf{w}^k - \phi_{\mathrm{p}}(x, \hat{y}, (s,e)_k)$$

With this feedback rule, the learning algorithm is also global, in the sense that the algorithm works at sentence level. However, since each prediction vector is updated for each local prediction error it commits, few global interactions between predictors are captured. Rather, the functions are trained independently at the same time. In particular, the only interaction that is taken into account is that the score function concentrates on the actual phrase candidates determined in the filtering layer, rather than on all possible phrase candidates for a sentence.

4.2.4 Discussion on the FR-Perceptron Algorithm

The recognition-based update strategy of the FR-Perceptron is *ultraconservative* in the way defined by Crammer and Singer for multiclass classification [2003b] or category ranking problems [2003a]. Many online algorithms are *mistake-driven* or *conservative*, in the sense that the prediction vectors are only updated on examples on which prediction errors are made. The notion of ultraconservative online algorithms stands for update rules which modify only the prototypes corresponding to mistakes in the global solution of an example. As Crammer and Singer [2003b] point out, an ultraconservative algorithm is also conservative. Furthermore, in binary classification a local prediction is directly the global solution, therefore the two definitions coincide. The difference is relevant in scenarios where there is some kind of inference that produces a global solution given the local predictions, which is the case of multiclass classification, ranking, and structured-output problems. Hence, in our case, the binary classification update rule of Section 4.2.3 is just conservative, because it explicitly corrects all local prediction errors. On the other hand, the recognition-based feedback of Section 4.2.2 is ultraconservative, because it looks only at the predictions which are responsible for global errors. Other ultraconservative versions of Perceptron, for multiclass and structured recognition problems, are revised in Section 3.4.2.

Related to the notion of conservativeness, in [Har-Peled et al., 2002] a generalized *constraint classification* framework is proposed. This general setting allows to model, in such ultraconservative way, multiclass and multilabel classification problems, and ranking problems which can be expressed with order relation pairs.

Directly related to FR-Perceptron are the global linear models for parsing and tagging by Collins [2002, 2004], which apply to solution spaces of exponential size. In this family of models, a sentence-solution pair is represented in a feature vector, and a weight vector in the same dimensionality produces a prediction score for that pair. Then, the learning problem consists of estimating the weight vector so that the correct solution is ranked the highest. In

[Collins, 2002], a Perceptron algorithm is presented for tagging problems, working with representations for which there exists a decoding algorithm that, given the weight vector, picks the top-ranked solution for a sentence in polynomial time (using dynamic programming). Basically, the online learning algorithm runs the decoder on a given example and then updates the weight vector ultraconservatively, looking only at the mistakes found in the top-ranked global solution. FR-Perceptron can be seen as an instance of this basic algorithm, and, in fact, the score functions are trained as in [Collins, 2002] (see proof below in subsection 4.2.5). However, FR-Perceptron extends Collins' algorithm by training also a filtering component which, working at word level (linear with the sentence length), discards solutions and makes the decoding process more efficient in terms of the number of candidates considered at the phrase level (quadratic with the sentence length).

As in all ultraconservative algorithms mentioned, the type of update of FR-Perceptron on each function has a global effect in the sense that interactions of functions are captured and they become dependent. The learning approach is driven so as to optimize the global accuracy of the recognizer, rather than local accuracies of each individual function. Also, by learning online from the output of the decoder, the algorithm naturally selects the most informative instances out of the quadratic number of possible phrase candidates. As we show experimentally in the next section, the algorithm effectively models the functions as word filters and phrase rankers, which contributes positively on optimizing the F_1 measure on precision-recall. We also provide empirical evidence in favor of the FR-Perceptron with respect to learning strategies which train the components independently.

4.2.5 Convergence Analysis of FR-Perceptron

Acknowledgement: The following result was mainly derived by Jorge Castro. We thank him for his contribution.

In this section we discuss a convergence result for the FR-Perceptron algorithm. The result we present depends on some restrictive hypotheses and has to be seen only as a first step in the task of achieving a deep and more useful analysis. The complete proof can be found in Appendix A.

The convergence proof we give is based on the proof by Novikoff [1962] for the basic Perceptron algorithm as much as on the proof presented by Collins [2002] for the Perceptron-based sequence tagging algorithm. Assuming *separability* for the Phrase Recognition function and linear separability for each of the *start* and *end* classifiers, the number of errors committed by the learning algorithm can be upper bounded.

To simplify the analysis, we do not consider phrase types in the solution, so we only deal with three prediction vectors, namely *start* ($\mathbf{w}_S$), *end* ($\mathbf{w}_E$) and *score* ($\mathbf{w}$).

Without the filtering component, our algorithm is the same than the algorithm analyzed by Collins [2002], which is proved to converge if the training

sample is separable (i.e., there exists a vector $\mathbf{w}^*$ that, for each training sentence, perfectly ranks the correct output structure higher than any competing structure).

The *start-end* filtering functions can be seen as general binary classifiers. Thus, if we assume that the training sample is Start-End separable (i.e., there exist vectors $\mathbf{w}_{\mathrm{S}}^*$ and $\mathbf{w}_{\mathrm{E}}^*$ that perfectly indicate whether each word of the training sample is a start/end or not), the classic result by Novikoff's bounds the number of start/end errors that Perceptron commits during learning.

Our proof departs from these two results. It shows that FR-Perceptron trains together the score function (trained as in Collins [2002]) on the top of the filtering functions (trained as binary classifiers), and that the type of updates FR-Perceptron performs permits the convergence of the two layers, as if they were trained separately. The following theorem bounds the number of errors of FR-Perceptronon the training set:

THEOREM **1** *For any training set $S = \{(x^i, y^i)\}_{i=1}^m$ separable with margin $\delta > 0$ and Start-End separable with margin $\gamma > 0$, it holds:*

1. *The number of learning feedbacks that affect to the start-end vectors* (SE-LF *stages*) *is bounded by $2R_{SE}^2/\gamma^2$, where $R_{SE} = \max_{1\leq i\leq m,\ 1\leq j\leq n_i} \|\phi_{\mathrm{w}}(x^i, j)\|$.*

2. *After a learning feedback stage l that has affected the start-end vectors* (SE-LF *stage*) *there are at most*

$$\max\left(m-1, \frac{R^2}{\delta^2} + \frac{2\|\mathbf{w}_0^l\|}{\delta}\right)$$

consecutive learning feedbacks that only affect to the score *vectors* (SCORE-LF *stages*). *Here, $\mathbf{w}_0^l$ is the vector $\mathbf{w}$ when the updates corresponding to stage l have been just made, and R is a constant such that $(\forall i : 1 \leq i \leq m)\,(\forall z : z \in \hat{\mathcal{Y}}(x^i))$ $(||\sum_{p\in y^i} \phi_{\mathrm{p}}(x^i, y^i, p) - \sum_{p\in z} \phi_{\mathrm{p}}(x^i, z, p)|| \leq R)$.*

3. *As a consequence of the previous two points, the* FR-Perceptron *algorithm makes a finite number of errors on the training set. When no more errors occur the total number of errors committed by the algorithm is bounded by*

$$\frac{2R_{SE}^2}{\gamma^2}\left(1 + \max\left(m-1, \frac{R^2}{\delta^2} + \frac{2MAX}{\delta}\right)\right),$$

where MAX is the maximum of the values $\|\mathbf{w}_0^l\|$ at any SE-LF *stage.*

This convergence result (developed in detail in Appendix A) gives insight into the FR-Perceptron learning algorithm, but it has two drawbacks. First, it relies on very restrictive assumptions (two separability hypotheses). Second, the upper bound claimed in the theorem does not illustrate the goodness of FR-Perceptron over the classical approach that considers two independent training steps. Note that assuming both separability hypothesis, this bound is worse than the bound one gets for independent training (i.e., the summation of the

training errors at each step). One of the goals of future work is to improve the convergence proof. We would like to find a convergence result that does not assume full start–end separability. Ideally, it had to show that the algorithm works well even if the start–end classifiers make one–side errors. In addition, the new proof had to provide a tighter bound on the number of errors. Here, one hopes to show a bound at least as good as the bound for the classical approach when training on samples having both separability hypotheses.

4.3 Experiments on Partial Parsing

In this section, we describe a series of experiments with FR-Perceptron on a partial parsing task, in the context of the CoNLL-2001 Shared Task : clause identification [Tjong Kim Sang and Déjean, 2001]. The goal is to recognize hierarchies of syntactic clauses, with no differentiation on the type of clauses. See Section 1.1.2 for a more detailed description of this problem.

In the following subsection we summarize the final settings and practical details of FR-Perceptron, and present the evaluation results of the architecture on the task. Then, the next subsections present several experiments, comparing the proposed learning algorithm to alternative learning strategies for the architecture, and showing the empirical behavior of each strategy in different situations.

In Chapter 5 we show more experimental results of the filtering-ranking architecture on three phrase recognition tasks. In that chapter, we provide much more details about the application of the architecture to Natural Language problems. We also contrast the results we obtain with those of other systems developed for the same tasks.

4.3.1 Experimental Setting and Results

In this section we briefly sketch some final settings of the filtering-ranking architecture. We also show the results that we obtain on the problem data. A more detailed description of the system appears in the next chapter.

The main challenge of Clause Identification problem is the hierarchical nature of the clause structures in a sentence. Since there is no differentiation on clause types, the model is composed by three functions: the *start* filter, the *end* filter and the *score* ranker.

ϕ_{w} and ϕ_{p} Representation Functions. For the *start-end* filters, the representation function (ϕ_{w}) consists of a window of features centered at the target word, that extracts forms, part-of-speech tags and chunk information of the neighboring words. For the *score* function, the representation function (ϕ_{p}) describes a clause candidate by means of many features that capture the structure of it, taking into account the relative pronouns, verbs, chunks, clauses found within the candidate, and punctuation marks. See next chapter for a detailed list of the features used in this task.

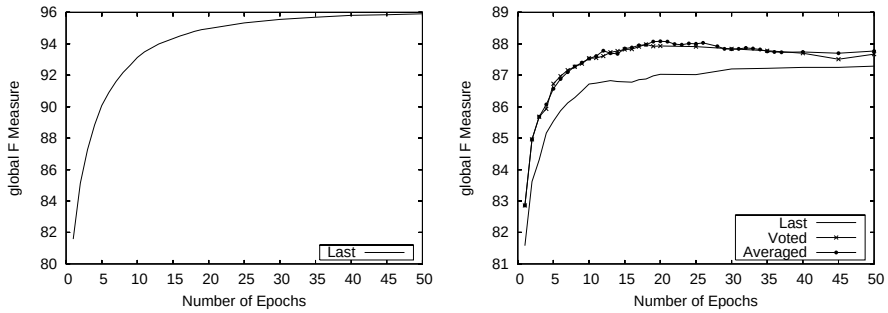

Figure 4.3: Learning curve on Clause Identification. Left plot: training set in *last* prediction mode. Right plot: development set in *last*, *averaged* and *voted* modes.

Polynomial Kernels. All results presented in the experimentation were obtained using polynomial kernels of degree two. Initial tests with a linear kernel revealed a very poor performance.

Voted Perceptron. The Perceptron-based learners of the architecture incorporate the results of Freund and Schapire [1999] on *Voted Perceptron* (see Section 3.4.2 for a detailed description). As we show below, the results we obtain in test data with *averaged* predictions are better that those obtained with standard Perceptron predictions. Also, the convergence is also much better.

Results

We trained the model for up to 50 epochs on the training data. Figure 4.3 shows the performance of the model (F_1 measure) on the training data (left plot) and on the development data (right plot). Clearly, the learning curve on the training set shows that the learning strategy effectively benefits the global F_1 measure on the task through the learning epochs. However, the performance gets stable around 96, indicating that the training set is not separable under the current choice of features and kernel. Looking at the performance on the development set, the model shows a good generalization curve, with the best results over 88% and with no significant overfitting. Although at epoch 50 the FR-Perceptron has not converged, the generalization performance seems to be stable from epoch 20, showing only minor decreases in further epochs.

Looking at the three prediction methods, both the *averaged* and the *voted* methods perform substantially better than the default *last* method. Throughout the experimentation in this section, we compute *averaged* predictions for all Perceptron versions used

4.3.2 Local vs. Global Learning

In the FR-Perceptron setting, all the functions are trained together online via recognition feedback. In this experiment we compare the FR-Perceptron to alternative learning settings in which training is based on a binary classification penalty. That is, each function is understood as a binary classifier: the *start-end* functions decide whether a word is or not a boundary word, and the *score* function decides if a phrase belongs to a solution or not. In this context, each prediction a function has made is subject to correction, and updated when the sign is not correct. We consider two versions of the classification-based strategy, and compare them to FR-Perceptron. Briefly, the three strategies are:

- *Independent Local Classification.* Trains each function independently, with a batch binary classification algorithm optimizing the local accuracy.
- *Global Online Classification.* Trains globally the three functions at the same time, with an online binary classification algorithm optimizing each local accuracy. It corresponds to the algorithm in Figure 4.2 using the feedback rule of Section 4.2.3.
- *Global FR-Perceptron.* Trains globally the three functions at the same time with the online FR-Perceptron, that concentrates only on the global accuracy. It corresponds to the algorithm in Figure 4.2 using the feedback rule of Section 4.2.2.

The following two subsections explain details on how we trained the classification models. Then, we provide comparative evaluation results and discuss the benefits of FR-Perceptron over the classification approaches.

Learning Independent Local Classifiers (local-VP,local-SVM)

Each local function is trained separately from the others with a batch learning algorithm for binary classification. We selected two algorithms which work in the same hypothesis space than FR-Perceptron (kernel-based linear functions): the batch version of the Voted Perceptron (local-VP), and soft-margin Support Vector Machines (local-SVM)[1]. For training, we generated three data sets from training sentences, one for each function. For the *start-end* sets, we considered an example for each word in the data, except those breaking chunks. For the *score* classifier, it is not feasible to generate one example for each possible phrase candidate in the data, since this generation would produce 1,377,843 examples with a 98.2% proportion of negatives. A first direct approach is to generate only phrase candidates formed with all pairs of correct phrase boundaries, which greatly reduces the number of negative examples. However, the resulting *score* classifiers are trained in a context in which perfect identification of phrase boundaries is assumed, which is not the real situation when running on the top

[1]We used the SVM^{light} package by Joachims [1999], available at `http://svmlight.joachims.org`.

Algorithm	Generation	#Neg.	%Neg.	Precision	Recall	F_1
local-VP	goldSE	26,374	51.50	83.84	80.55	82.16
local-SVM	goldSE	26,374	51.50	84.31	82.83	83.57
local-SVM	$\theta = 0$	28,165	53.14	88.14	**82.85**	85.41
local-SVM	$\theta = -0.3$	28,747	53.64	88.34	82.76	85.46
local-SVM	$\theta = -0.5$	29,145	53.99	88.46	82.61	85.43
local-SVM	$\theta = -0.7$	29,610	54.38	88.54	82.48	85.41
local-SVM	$\theta = -0.9$	30,432	55.06	88.91	82.58	85.63
local-SVM	$\theta = -1.0$	59,498	70.55	91.12	81.27	**85.91**
local-SVM	$\theta = -1.1$	97,101	79.63	91.49	80.80	85.82
local-SVM	$\theta = -1.2$	120,856	82.95	91.33	80.51	85.58
local-SVM	$\theta = -1.5$	240,463	90.64	**92.31**	78.01	84.56

Table 4.1: Performance on the Clause Identification development set when training the functions of the model separately as classifiers. In the first two models, the examples for the *score* function are generated using the gold *start-end* words (goldSE). For the models below, phrase examples are generated with the learned *start-end* functions, considering words with with a *start-end* prediction higher than a threshold θ. In any case, the number of positive training examples is 24,841, whereas the number of negative examples is shown in the third column. The percentage of negative examples is shown at the fourth column.

of learned *start-end* functions. Table 4.1 shows the overall performance of the system running with local-VP (averaging predictions) and local-SVM classifiers. As it could be expected, local-SVM perform better than local-VP. But, clearly, the performance is much lower than with FR-Perceptron, suggesting that the *score* functions in this setting are not robust enough.

To overcome this limitation, we generated new training samples for the *score* function, now considering the actual behavior of the learned *start-end* functions. The approach is similar to the one we used in [Carreras et al., 2002b]: first the learned *start-end* functions are applied to the words in the training data; then, phrase candidates are generated considering pairs of boundary words whose prediction is above a certain threshold θ. Besides, the positive phrases are always generated. Thus, this procedure only adds the negative phrase candidates which pass the threshold θ. We performed several trials only with local-SVM, first selecting θ and then the C regularization parameter of the SVM. This threshold was only used for generation of training examples. When testing, the filters were used as usual, with threshold at 0. Table 4.1 summarizes the results. Note that, by considering predicted boundaries in training, the precision of the system easily improves 4 points. However, as the amount of negative instances substantially increases, the recall goes down. The best performance for batch models was found using a threshold of -1.0, with local-SVM. We will use this model to compare against other learning settings.

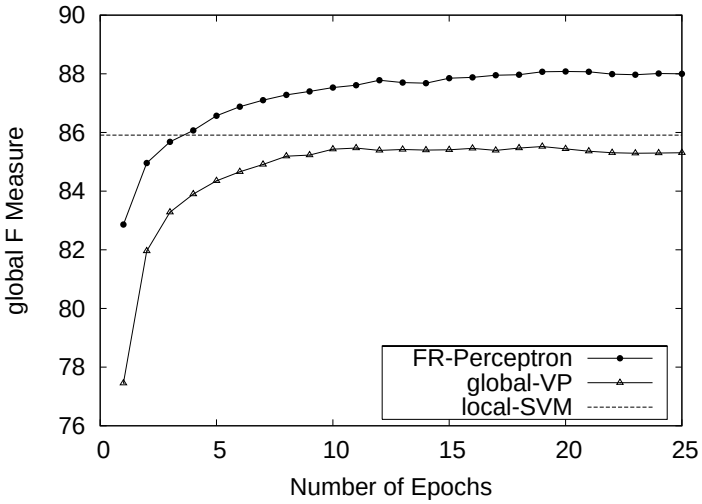

Figure 4.4: Performance on the Clause Identification development set for different learning settings: the FR-Perceptron model, and classification-based models trained with VP in the online setting (global-VP) and with local-SVM in the batch setting.

Learning Online Global Classifiers (global-VP)

In this setting all functions are trained together online with the Voted Perceptron algorithm, visiting one sentence at a time, and providing binary classification feedback to each function for each prediction. Given a sentence, the *start-end* functions are first applied to each word and, according to their positive decisions, phrase examples are generated for the *score* function. In this way, the input of the *score* function is dynamically adapted to the *start-end* behavior. The particular algorithm is similar to FR-Perceptron in that it works globally at sentence level. However, instead of giving the recognition-based feedback rule of Section 4.2.2 to update the functions, it uses the binary classification feedback presented in Section 4.2.3. We trained the online classification model for up to 25 epochs, and tested it with averaged predictions.

Comparative Results

Figure 4.4 shows the learning curve in terms of the F_1 measure of the two online models, together with a straight line corresponding to the performance of the best local-SVM batch model. Clearly, the performance of the FR-Perceptron is better than those of the classification–based models. At any epoch, the curve is more than 2 points higher than the one of the online classification-based model (global-VP). This fact gives empirical evidence in favor of the recognition–based feedback for learning in phrase recognition problems. Below, we provide more

		Development			Test		
Model	T	Prec.	Recall	F_1	Prec.	Recall	F_1
local-SVM	-	**91.12**	81.27	85.91	89.18	77.92	83.17
global-VP	19	91.06	80.62	85.52	**89.25**	77.62	83.03
FR-Perceptron	20	90.56	**85.73**	**88.08**	88.17	**82.10**	**85.03**

Table 4.2: Results on the Clause Identification development and test sets, for different learning settings: the FR-Perceptron model and the classification-based models trained batch with SVM and online with VP (global-VP). The number of epochs used for testing (T) has been optimized on the development set, as well as parameters of the SVM.

results showing why the recognition feedback guides the learning strategy much better than the classification feedback.

Looking at classification-based models, SVM performs slightly better than global-VP. However, the SVM model has been obtained after tuning the generation of negative phrases, taking into account the learned *start-end* functions. It is worth to recall that when training straightforwardly the *score* function with correct boundaries (see Table 4.1), the SVM model performs more than two points worse than the global-VP model. In contrast, the online model automatically rules the interaction between both layers, so no tuning needs to be made.

As a summary, table 4.2 shows the performance of each learning strategy on the development and the test sets. Parameters have been optimized in the development set. The performance is significantly lower on the test set, which, rather than overfitting of parameter tuning, should be attributed to more difficulty on that portion of the data —a similar drop is exhibited by most systems running on this data [Tjong Kim Sang and Déjean, 2001]. It is interesting to see that the better performance of FR-Perceptron comes from the substantially higher recall figures. This fact relates to the behavior of the filtering layer, which will be explored in detail in the next subsections.

As a complementary information, Figure 4.5 shows the size of each learned function in terms of the number of different vectors which compose its dual form.

4.3.3 A Closer Look at the Filtering Behavior

To get an idea of how the learning strategy of FR-Perceptron works, it is interesting to look at the evolution of the performance of the *start-end* filtering layer. Figure 4.6 plots the precision/recall curves of the *start-end* decisions, for each learning strategy considered. On both decisions, the FR-Perceptron starts with high levels of recall and low levels of precision and, along the learning epochs, substantially improves the precision with minor decreases in recall. In contrast, the classification models depart from a high precision and low recall values, and

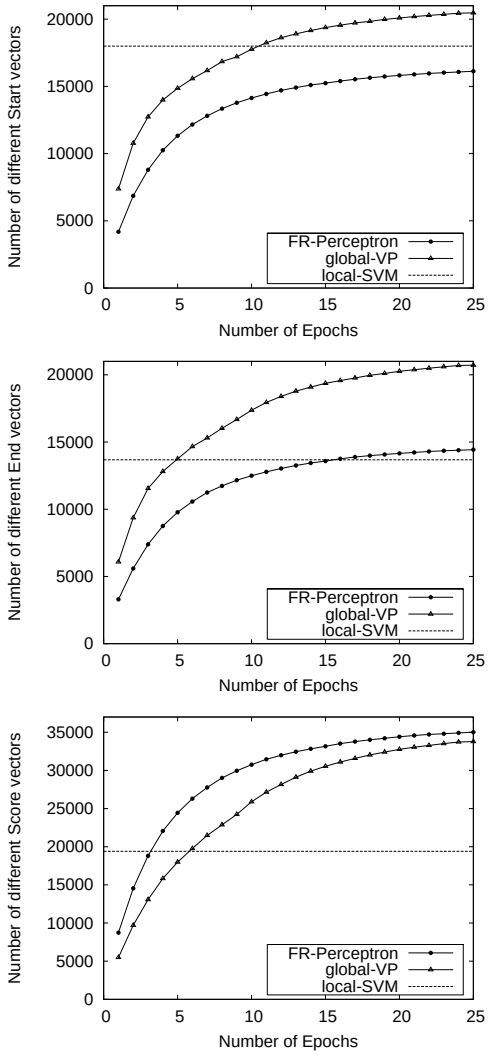

Figure 4.5: Size of the dual form, in terms of the number of different vectors, of the Clause Identification learning functions, with respect to the number of learning epochs. At a certain point of the training process, a kernel-based prediction involves a kernel operation with *each* vector that composes the dual form of the linear separator.

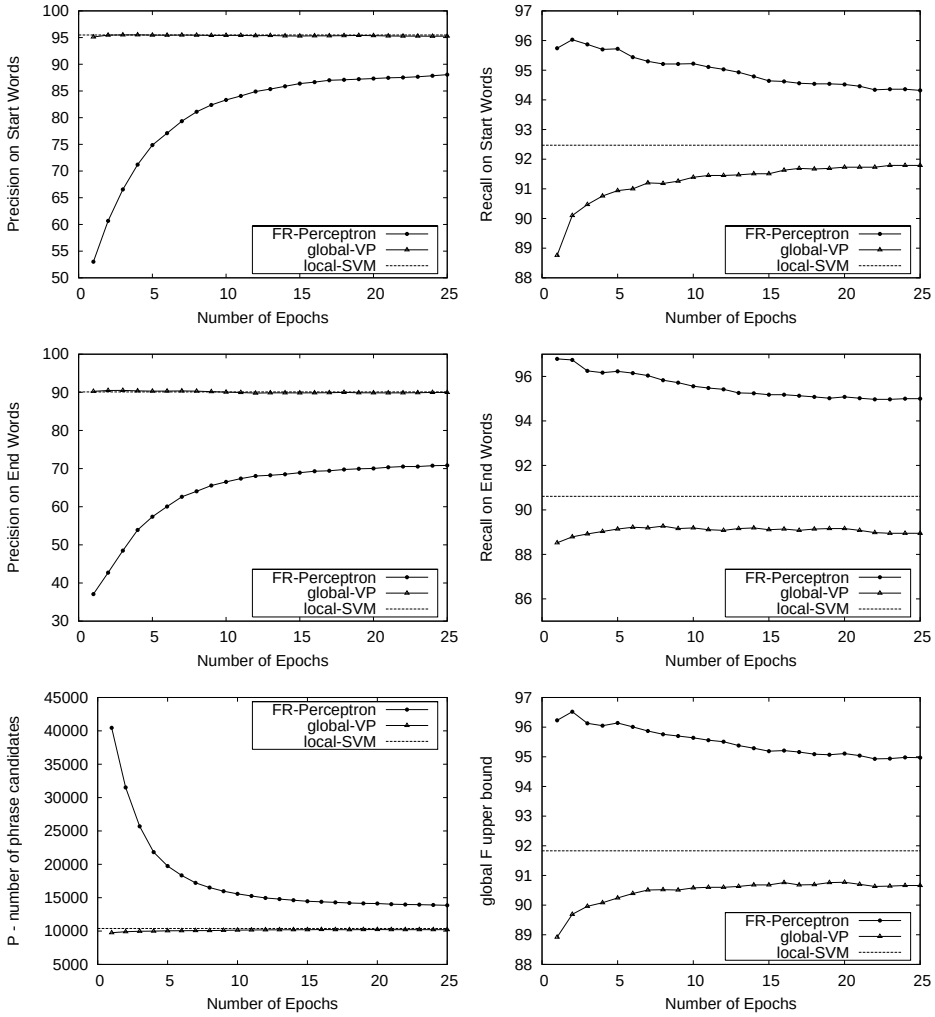

Figure 4.6: *Start-End* behavior, for the three learning strategies, on the Clause Identification development set. The first two rows show precision/recall curves on the identification of Start and End words, respectively. The third row shows the behavior of the *start-end* filtering layer from a global point of view: The left-hand side plots the size of the set of candidate phrases (inversely related to *start/end* precision); The right-hand side plots the maximum achievable global F_1 measure, assuming perfect *score* functions (related to *start/end* recall).

throughout the learning process increase the recall while maintaining the precision. As it has been argued, the optimal *start-end* functions should behave as filters which do not block any phrase of the solution. Thus, they should maintain very high recall values on phrase boundary words, and try to maximize the precision. The plots evince that this behavior is automatically obtained via the global recognition feedback.

The penalty on the global F_1 measure caused by errors in the *start-end* layer is also shown in Figure 4.6 (bottom row, right plot). The curve shows, for each epoch, the maximum achievable global F_1 measure given the phrases proposed by the *start-end* layer, that is, it is assumed a perfect *score* function given the learned *start-end* functions. Additionally, the filtering precision of the *Start-End* layer is also shown in Figure 4.6 (bottom row, left plot), in terms of the number of phrase candidates it produces. For the latter measure, the total number of possible phrase candidates in the development set is 300,511. The FR-Perceptron exhibits the expected behavior for a filter: while it maintains a high recall on identifying correct phrases (above 95%), it substantially reduces the number of phrase candidates throughout the learning process, and, thus, the search space for the scoring layer. As a consequence, the input space of the *score* functions is progressively simplified. Also, since the number of explored phrase instances is reduced at each epoch, the recognition process becomes more efficient. Unfortunately, the size of each function in terms of the number of vectors combined also grows (recall Figure 4.5) and, in practice, the overall model is each time slower.

Far from the behavior of the FR-Perceptron, the models trained via classification feedback do not adapt the filter behavior to maximize the global performance and, although they aggressively reduce the search space, provide only a moderate upper bound on the global F_1.

4.3.4 A Closer Look at the Behavior of the Score Function

A complementary question of the filtering-ranking architecture concerns the learnability of the *score* function. In this experiment we trained *score* functions in special settings of the filtering layer.

In the first setting, we assumed a perfect classification of starting and ending words. In this situation, we trained a *score* function via recognition feedback, and another via classification feedback, visiting in both cases the training sentences online. The top plot of Figure 4.7 shows the learning curves on the development set. The plot also presents the performance of the local-SVM scorer on the top of perfect filters, which has been trained on the same space. The three approaches obtain a similar performance when they are stable, being the recognition-based scorer slightly better. It is also noticeable that the recognition feedback achieves high performance levels much faster than the classification feedback. In all cases, the performance is very high (F_1 over 97.5), indicating that the score function can be effectively learned and that the filtering component is the real bottleneck of the system.

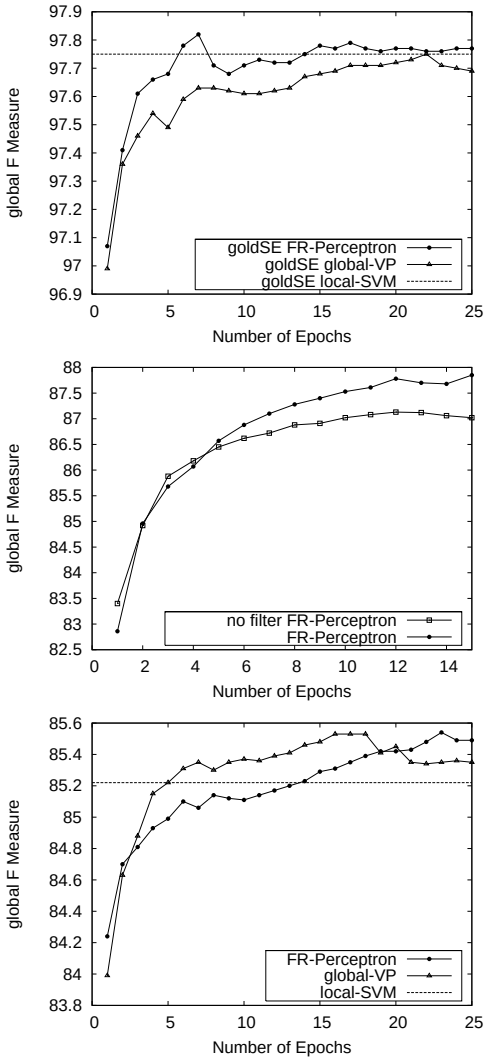

Figure 4.7: Performance on the Clause Identification development set for special settings of the filters. Top plot: using perfect *start-end* filters, showing the performance obtained by training the *score* function as a classifier or as a ranker (classification vs. recognition feedback). Middle plot: effect of using a *start-end* filter or not. Bottom plot: using the learned *start-end* filters of the FR-Perceptron, and learning new *score* functions from scratch.

In the second setting, we considered a model with no filtering layer. Therefore, it explored all possible phrases in a sentence (except those breaking chunks). This fact makes the model computationally very expensive, since at each training epoch it has to visit 1,377,843 phrase candidates. In contrast, the filtering component in the regular model produces a filtered set ranging from 140,000 instances at the first epoch to 65,000 instances when stable (see also bottom left plot of Figure 4.6 for the reduction in the development set). The middle plot of Figure 4.7 shows the performance of this model, together with that of the regular FR-Perceptron. The results in terms of global F_1 measure are very competitive, outperforming the results of the classification-based models of the previous experiment. This fact indicates that the *score* function can be straightforwardly learned, at the cost of working in a computationally very expensive search space. However, the combination of filtering functions with the ranker still provides the best results, and makes the overall model computationally feasible.

Finally, we considered the *start-end* filters learned with the FR-Perceptron in the experiment described in the previous section. These filters define phrase candidates both in the training data and the test data. With the training candidates, we trained *score* functions from scratch, using the three running learning settings:

1. Online via recognition feedback, corresponding to the FR-Perceptron with fixed *start-end* functions.

2. Online with VP, giving binary classification feedback to each prediction. Note that in this setting, where *start-end* functions are fixed, the only advantage of training a classification model in the online setting is that the actual predictions can be used to give values to features, which turns out to provide better results.

3. Batch, with the SVM binary classification algorithm.

The bottom plot of Figure 4.7 depicts the learning curves for these models. The online models reach similar performances in this case: the classification-based model is faster at learning in terms of epochs, but the recognition-based model end with slightly better figures. The SVM batch model behaves slightly worse. Note also that the FR-Perceptron achieves a performance more than 2 points lower than in the regular model which trains both components together. This result suggests that capturing interactions between the two layers while learning leads to a better generalization performance.

4.4 Conclusion of this Chapter

We have presented a global learning algorithm, FR-Perceptron, for the general problem of recognizing structures of phrases, in which, typically, several different learning functions are required to recognize the structure. The FR-Perceptron algorithm works online getting feedback from a global point of view, and trains

all the functions together, so that interactions between functions when performing the task can be captured in the learning process. In particular, our algorithm learns two layers of decision functions: a filtering layer, which reduces the solution space to a set of plausible candidates, and a ranking layer, which explores the candidates to select the optimal ones.

By conducting extensive experimental evaluation on clause identification, we conclude that the global online learning via recognition feedback outperforms alternative training strategies based on traditional binary classification feedback. To explain this fact, we have shown in detail how the behavior of the filtering and the ranking layers during training is effectively adapted by the algorithm so as to benefit the global performance F_1 measure.

Chapter 5

A Pipeline of Systems for Syntactic-Semantic Parsing

In this chapter we develop three Natural Language analyzers that resolve phrase recognition tasks, and that can be run in a pipeline. Namely, we describe systems for the tasks known as Syntactic Chunking, Clause Identification, and Semantic Role Labeling.

In all cases, we make use of the filtering-ranking learning architecture presented in Chapter 4. While all tasks can be casted as phrase recognition problems, they present different characteristics. We show that the filtering-ranking architecture can be particularized for the following tasks of increasing levels of difficulty:

- Noun Phrase Chunking, with a single type of phrases to recognize (i.e, noun phrases) that form a sequential phrase structure in a sentence.
- Syntactic Chunking, a sequential task generalizing Noun Phrase Chunking, with eleven different types of chunks.
- Clause Identification, with a single type of phrases (i.e., clauses) that form a hierarchical phrase structure in a sentence.
- Semantic Role Labeling, with 20 different types of semantic roles. Our approach looks for a hierarchy of arguments in a sentence. In this structure, each argument is linked to one or many verbs, and this argument-verb relation is labeled with the appropriate semantic role.

We develop such applications under the settings proposed in the CoNLL Shared Task series of the 2000, 2001 and 2004 editions. In doing so, we are able to contrast the results obtained with our learning architecture with those of other systems that use different learning algorithms and strategies. As we will see, our analyzers obtain accuracies that are among the top results of the state-of-the-art systems evaluated on CoNLL data.

Apart from the analyzers presented here, systems for the task of Named Entity Extraction have also been developed, as part of the work related to this thesis. This particular task was addressed in two editions of the shared task, namely in 2002 for Spanish and Dutch [Tjong Kim Sang, 2002a], and in 2003 for English and German [Tjong Kim Sang and De Meulder, 2003]. For conciseness we do not present the results here, but they can be consulted in the CoNLL Shared Task papers. In Carreras et al. [2003b], a Filtering-Ranking architecture was presented for the CoNLL-2003 task. In Carreras et al. [2002a, 2003a], named entity extractors based on BIO tagging were developed for CoNLL-2002 and CoNLL-2003 tasks, with AdaBoost as the learning algorithm.

The rest of the chapter is organized as follows. Next section reviews the goals of the tasks we work with, and provides details of the data sets. Section 5.2 presents some generalities of the analyzers in this chapter. Then, section 5.3, 5.4 and 5.5 present, respectively, the systems for Syntactic Chunking, Clause Identification and Semantic Role Labeling, together with their results contrasted with other CoNLL Shared Task systems.

5.1 A Pipeline of Analyzers

The Conference on Natural Language Learning (CoNLL), through the organization of shared tasks, provides benchmarks on Natural Language problems in which many learning-based systems can be compared. [1] The 2000, 2001 and 2004 editions dealt respectively with the tasks of Syntactic Chunking [Tjong Kim Sang and Buchholz, 2000], Clause Identification [Tjong Kim Sang and Déjean, 2001] and Semantic Role Labeling[2] [Carreras and Màrquez, 2004], and in this chapter we follow the corresponding settings.

Basically, to define a benchmark setting that allows to compare systems, a task defines a common evaluation method and common data. Systems are evaluated using precision, recall, and F_1 on the test data (such measures are described in Section 1.1.3). As for data, the three tasks build on sentences from the WSJ part of the Penn TreeBank [Marcus et al., 1993]. Chunks and clauses were extracted from that corpus, while the annotations about semantic roles were extracted from PropBank [Palmer et al., 2005]. Table 5.1 summarizes the WSJ sections used as training, development and test data, as well as the number of sentences and tokens in each set, and other counts that are specific to each task. It has to be noted that for the chunking task of CoNLL-2000 the test set corresponded to section 20, identified in the table as the development set. In that edition, there was no official development set. Whenever we refer in this chapter to test results for chunking, we will be referring to the official test set, that is, WSJ Section 20.

[1] The websites of the CoNLL Shared Tasks can be accessed through the CoNLL general website, at `http://www.cnts.ua.ac.be/conll` .

[2] In 2005, the Shared Task dealt also with Semantic Role Labeling [Carreras and Màrquez, 2005]. The system presented here, however, follows the 2004 setting, which is not directly comparable.

A particular aspect of the tasks we work with is that they can be pipelined: each one receives as input the output of the previous tasks, and adds a new layer of analysis. In a language processor, the pipeline would start with a tokenizer that segments running text into sentences, and sentences into tokens. Then, a PoS tagger would assign to each token the most appropriate PoS tag. After that, the three presented analyzers would proceed: first the chunker, then the clause recognizer, and finally the semantic role labeler. Figure 5.1 shows an example of a sentence analysis divided into columns, each related to an analyzer of the pipeline. Following, we provide more details about the three tasks:

Syntactic Chunking The Syntactic Chunking problem consists of recognizing the set of chunks of a sentence, that is, non-overlapping base syntactic constituents that form a sequential phrase structure. To do so, the available input information consists of the words of the sentence together with their part-of-speech tags (PoS). In CoNLL-2000, eleven different types of chunks were considered. The three most common chunks in data are noun phrases (`NP`), verb phrases (`VP`) and prepositional phrases (`PP`). In the literature related to shallow parsing, it is also common to address a simple version of the problem, that considers only noun phases. This particular problem was the focus of the first Shared Task in CoNLL-1999, and there are many systems in the literature that experiment with it. In turn, the 1999 edition was based on the work by Ramshaw and Marcus [1995], that experimented with noun phrase chunking and verb phrase chunking separately.

Clause Identification The Clause Identification problem consists of recognizing the set of clauses on the basis of words, part-of-speech tags, and chunks –the latter being recognized in the previous task. In the CoNLL-2001 setting the problem was simplified by removing the clause–type labels. As a consequence, there is only one category of phrases to be considered. However, the clause structure of a sentence is hierarchical. This fact makes the problem more complex than the previous task, because recursiveness of clauses has to be taken into account.

Semantic Role Labeling Semantic Role Labeling is the problem of recognizing the arguments of the propositions of a sentence, and label them with semantic roles. In general, a sentence may contain a number of propositions, each formed by a predicate and a set of arguments. Arguments in a proposition do not overlap. Given a predicate, thus, the corresponding set of arguments is a sequential phrase structure, and the task can be thought as a chunking task. The CoNLL-2004 Shared Task addressed this problem considering the PropBank corpus, that defines 20 different types of semantic roles. In PropBank, such roles can be differentiated into numbered arguments (7 types) –whose semantics depend on the verb and the verb usage in a sentence– and adjuncts (13 types) –that express temporal, locative, cause, and other semantic aspects. The goal was to recognize labeled roles of verbal predicates on the basis of partial

	$\|\mathcal{K}\|$	Training *sect. 15–18*	Development *sect. 20*	Test *sect. 21*
Sentences		8,936	2,012	1,671
Tokens		211,727	47,377	40,039
Syntactic Chunking	11			
Chunks (all types)		106,978	23,852	20,159
NP		55,081	12,422	10,341
VP		21,467	4,658	4,186
PP		21,281	4,811	3,814
Clause Identification	1			
Clauses		24,841	5,418	4,856
Semantic Role Labeling	20			
Propositions		19,098	4,305	3,627
Distinct Verbs		1,838	978	855
Arguments (all types)		50,182	11,121	9,598
A0		12,709	2,875	2,579
A1		18,046	4,064	3,429
A2		4,223	954	714
ADV		1,727	352	307
DIS		1,077	204	213
LOC		1,279	230	228
MNR		1,337	334	255
MOD		1,753	389	337
TMP		3,567	759	747

Table 5.1: Information on the training, development and test data sets, common to the three tasks. Below the label of each set, in the second line of the table, the WSJ sections that form the set are indicated (in CoNLL-2000, the chunking systems were actually tested in the set noted here as development, since there was no official set for validating systems). For each data set, the table shows counts of the number of sentences and tokens, as well as counts specific to each task. The second column ($|\mathcal{K}|$) specifies the number of phrase types considered in each task. Then, the total number of phrases (chunks, clauses or arguments) is shown, together with counts of the most frequent types in the data. For Semantic Role Labeling, the table also gives the number of propositions in the data (i.e., chunkings, each related to one predicate), and how many distinct verbs appear in such propositions.

```
Ship         NN   B-NP  (S*      -          (A1*   (A0*   (A0*      *
lines        NNS  I-NP    *      -             *)     *      *      *
operating    VBG  B-VP    *      operate     (V*)     *      *      *
in           IN   B-PP    *      -         (LOC*      *      *      *
the          DT   B-NP    *      -             *      *      *      *
Pacific      NNP  I-NP    *      -             *)     *)     *)     *
plan         VBP  B-VP    *      plan          *    (V*)     *      *
to           TO   B-VP   (S*     -             *    (A1*     *      *
raise        VB   I-VP    *      raise         *      *    (V*)     *
rates        NNS  B-NP    *      -             *      *    (A1*     *
on           IN   B-PP    *      -             *      *      *      *
containers   NNS  B-NP    *      -             *      *      *   (A0*)
carrying     VBG  B-VP   (S*     carry         *      *      *    (V*)
U.S.         NNP  B-NP    *      -             *      *      *   (A1*
exports      NNS  I-NP    *      -             *      *      *      *)
to           TO   B-PP    *      -             *      *      *  (DIR*
Asia         NNP  B-NP    *)     -             *      *      *)     *)
about        RB   B-NP    *      -             *      *    (A2*     *
10           CD   I-NP    *      -             *      *      *      *
%            NN   I-NP    *)     -             *      *)     *)     *
.            .    O       *)     -             *      *      *      *
```

Figure 5.1: An example sentence, annotated in columns. The first and second columns annotate words and PoS tags, respectively. Syntactic chunks are found in the third column, in BIO notation. The fourth column annotates the syntactic clauses, which appear in Start-End notation. The remaining columns (from 5th to 9th) annotate propositional arguments labeled with semantic roles: the 5th column marks the position of verbal predicates of the sentence, which appear in infinitive form; column 6th corresponds to the arguments of "operate" (according to the definition of roles in PropBank, `A1` stands for the operator, while `LOC` denotes a locative adjunct); column 7th annotates the arguments of "plan" (`A0` is the planner, while `A1` is the thing planned); column 8th is for "raise" (`A0` is the agent, `A1` is the thing rising, `A2` is the amount risen); finally, column 9th marks the arguments of "carry" (with `A0` as the carrier, `A1` as the thing carried, and `DIR` as an adjunct denoting direction).

syntactic information, namely chunks and clauses —that in turn are recognized by the two previous analyzers. In practice, the verbal predicates for which to recognize the arguments were marked in the input sentence, although, broadly, these correspond to the words with a verbal PoS tag. The output of a system for a sentence is set of argument chunkings, one for each target verb.

5.2 General Details about the Systems

Before describing the particular systems of this chapter, we comment on two aspects that are general to all systems. The first concerns the representation of learning instances with features. The second is about the use of the Voted Perceptron algorithm, an extension of the original Perceptron to achieve more robust predictions.

5.2.1 On Representation and Feature Extraction

In this section we describe the type of representation of learning instances that is adopted in the phrase recognition models presented in this thesis. Such representations are not novel, but fairly standard in literature.

To remind the reader, the general form of the learning functions we use is $h : \mathcal{X} \rightarrow \mathbb{R}$, where $\mathcal{X}$ is some instance space, and the output of h is a real-valued prediction. The h function is implemented with a weight vector $w \in \mathbb{R}^n$, and a representation function $\phi : \mathcal{X} \rightarrow \mathbb{R}^n$. Given an instance $x \in \mathcal{X}$, $\phi(x)$ outputs an n-dimensional vector of features that represents the instance. The w vector, also with n dimensions, has one weight for each feature. Then, a prediction is computed as the scalar product between the weight and feature vectors, i.e. $h(x) = \langle w, \phi(x) \rangle$. In this section, we concentrate on how the ϕ function works.

Our particular architectures involve two types of representation functions: one for words (ϕ_{w}) and the other for phrase candidates (ϕ_{p}). In both cases, the representation function not only takes the target instance (a word or a phrase), but also the context in which such instance appears. Hence, a representation function takes a sentence, a target word or phrase and, possibly, a partial solution (recall that our recognition strategy is incremental), and outputs a feature vector that describes the information that, supposedly, is relevant for the phenomenon being predicted. In our case, the features we consider are binary-valued, and indicate whether a certain property holds in the instance or not. In the literature, such features are often called *indicator* features.

Window Representation. A very common technique to represent a word in the context of a sentence is that of a *window of words*. Assume a sentence of n words, in which we are willing to make a prediction on the i-th word (e.g., whether word x_i starts an `NP` chunk). To support the prediction, the sentence is annotated, fully or partially, with linguistic tags and elements that have been predicted in previous processes (e.g., PoS tags), or in previous steps of the current one (e.g., "starts" of chunks in previous words). A window representation

		-3	-2	-1	0 ↓	+1	+2	+3	
`form >`	...	operating	in	the	Pacific	plan	to	raise	...
`PoS >`		`VBG`	`IN`	`DT`	`NNP`	`VBP`	`TO`	`VB`	
`Starts >`		`S-VP`	`S-PP`	`S-NP`	?	?	?	?	
`Ends >`		`E-VP`	`E-PP`	`--`	?	?	?	?	

`Word features:`	`word:-2:in`, `word:-1:the`, `word:0:Pacific`, `word:1:plan`, `word:2:to`
`PoS features:`	`pos:-2:IN`, `pos:-1:DT`, `pos:0:NNP`, `pos:1:VBP`, `pos:2:TO`
`PoS bigrams:`	`posb:-2:-1:IN_DT`, `posb:-1:0:DT_NNP`, `posb:0:1:NNP_VBP`, `posb:1:2:VBP_TO`
`Starts+PoS:`	`s+p:-2:-1:PP_DT`, `s+p:-1:0:NP_DT`, *etc.*

Figure 5.2: An example of feature extraction with a window of half-size 2, in the context of predicting *start-end* boundaries of chunks. The top of the figure shows a piece of running text, with the target word under consideration marked with an arrow. The available annotations are the word forms of the sentence and their PoS tags, as well as the values being predicted in the current process (assuming left-to-right exploration), that indicate whether the words to the left of the target word start/end syntactic chunks (actually, these dimensions would be multi-valued, i.e. a particular word can be predicted as start/end of many chunk types at the same time). The line above the words indicates the position of each word, relative to the central word. The bottom part of the figure gives examples of extracted features, for four extraction patterns. The first two extract the forms and PoS tags of the words in the window. The third pattern extracts every pair of contiguous PoS tags in the window (bigram). The Starts+PoS pattern combines "start" indicator values of a word with the PoS tag of the next word. Note that all features appear with relative positions.

extracts the annotations from a target word and their neighboring words. Formally, a window of half-size s anchored in a word x_i extracts annotations of the words $[x_{i-s}, \ldots, x_{i+s}]$. The particular extraction of annotations is made following a number of predefined extraction patterns (e.g., extract the PoS tag of a word). Each pattern is evaluated at different positions in the window. An extraction pattern (denoted by some label), together with a position relative to the central word and the value of the pattern in that position forms a final indicator feature of the system. For example, a typical feature of the system might be *"the PoS tag of the word at position -1 is* `DT`*"*. Figure 5.2 shows an example of a window-based feature extraction, in the context of syntactic chunking.

Representations for Phrase Candidates. The main difficulty at representing phrase candidates is that the length of a phrase is of arbitrary size. If phrases are long, a pattern that represents the sequence of elements forming the phrase will not generalize well, because it will appear very sparsely in data. A solution to this problem is to break down the phrase candidate into many pieces of fixed size. This corresponds to extracting n-grams of the candidate. Another possibility is to look for specific, task-dependent elements within the candidate, and discard the rest. We use both approaches in the systems that follow. Finally, it is also common to represent the context of the phrase candidate by extracting windows of features at the start and end positions.

Throughout this chapter, we describe particular feature extraction functions for the problems that are addressed. As it will be shown, some extraction patterns are task dependent, and have been designed with the advice of experts in linguistics. However, most of the extraction patterns basically codify the annotations that are present in the input part of the task, in a quite exhaustive way.

Two Notes on Feature Extraction. Our representation functions do extract features from values that are predicted in visited parts of the exploration. When training the functions, the online learning paradigm offers two possibilities to give value to these features: (1) consider the "gold" values that come in the training data (this is the variant used with *batch* learning algorithms); or (2) use the actual predictions made during training. In preliminary experimentation, we found slightly better results when working with features extracted from the predicted structure. The second note is about feature reduction. The type of extraction patterns we consider generate hundreds of thousands of binary indicator features (to explain these numbers, consider extraction patterns that generate one feature for *each* word of the English language, or even for combinations of a few words). In this scenario, it is very common to apply a simple feature filtering, related to the frequency of a particular feature in training data. In our online learning process, we did not consider features that appeared less than three times during the first training epoch. This simple frequency threshold reduces a high percentage of possible features, while it has no significant influence on the performance.

Polynomial Kernels. All results presented in the experimentation were obtained using polynomial kernels of degree two (see Section 3.4.2 for the kernel-based version of Perceptron). Initial tests in the context of Clause Identification revealed a very poor performance for the linear case and no significant improvements for degrees greater than two. The development of specific kernels for exploiting the type of phrase structures we address, such as those in [Collins and Duffy, 2002], is a question that deserves further attention.

5.2.2 Voted Perceptron (VP)

We incorporate in the architecture the results of Freund and Schapire [1999] on *Voted Perceptron* (see Section 3.4.2 for a detailed description). It consists of storing all versions of a prediction vector that are generated during training, each accompanied with a weight related to the number of correct positive predictions it makes while it survives until a mistake is produced. [3] This collection of weighted prediction vectors –that ends with the regular Perceptron vector– serves to make robust, weighted predictions when testing. The authors propose two voting methods, namely *voted* –weighting the signs of the predictions (i.e., +1 or –1)– and *averaged* –weighting directly the prediction scores.

Throughout the experimentation in this chapter, we compare the test performance of the two voting prediction methods and that of the standard Perceptron. We show that, in general, *averaged* predictions provide slightly better results than voted predictions. Also, we show that the voting methods converge much faster to stable results than the standard prediction method.

5.3 Syntactic Chunking

In this section we describe a syntactic chunking system, developed in the setting of the CoNLL-2000 Shared Task [Tjong Kim Sang and Buchholz, 2000]. We make use of a filtering-ranking architecture. To our knowledge, all learning-based systems found in the literature rely on learners at word-level that assign tags to words (see, e.g., Muñoz et al. [1999] or Kudo and Matsumoto [2001]). In our case, we make use of learners that work at chunk level.

We first summarize the chunking strategy. Then, we describe the features of the system. After that, we show the results of our system on test data, and we contrast them with those of the top-performing systems that have been developed within the CoNLL-2000 Shared Task setting.

5.3.1 Strategy

Syntactic chunking can be straightforwardly approached with the filtering-ranking architecture described in Chapter 4. Here are the main details of the architecture:

- For each type of chunk, there are three linear separators, namely the *start-end* and the *score*: the first two identify chunk candidates, and the latter predicts a plausibility score for each chunk candidate. In the particular setting of CoNLL-2000, with 11 types, the architecture is formed by 33 separators. We will also show results for the specific task of `NP` chunking, in which the chunker concentrates exclusively on Noun Phrases, and, thus, is formed by three linear separators.

[3] Differing from the original work, we do not consider counts of correct negative predictions for setting the weights, since in recognition-based problems negative predictions are the default, most common values.

- The phrase structures to be recognized are of sequential nature. In other words, chunks do not overlap, and do not admit embedding. This requirement translates into constraints that the inference process will take into account when building the chunk structure for a sentence. The particular type of inference is similar to the Viterbi algorithm, and has been described in Section 3.3.2: it explores the sentence from left to right, and at each word it computes the optimal structure, in terms of chunk scores, from the beginning of the sentence to that word.

5.3.2 Features

Essentially, our system reproduces similar feature spaces than other relevant works for these tasks, which reported state-of-the-art results [Muñoz et al., 1999; Kudo and Matsumoto, 2001; Collins, 2002; Sha and Pereira, 2003]. The main difference is that such systems work only at word level, while our system operates also at phrase level. In the latter level, we develop features for representing a chunk candidate.

Representation of Words

The *start-end* functions of the chunker work with a window-based representation of words. The window is of half-size 2, that is, it considers 5 words including the central word. For a window centered at the i-th word, the following feature extraction patterns are applied to $[x_j]_{j=i-2}^{i+2}$:

- **Word forms** and **PoS tags**.
- **PoS-Grams**, on all possible sequences within the window that include the central word x_i.
- **Left Start/End Flags**. For each type of chunk k and each word x_j to the left of x_i, two features indicating whether x_j has been predicted as a start/end of a chunk of type k.

Representation of Chunk Candidates

The *score* functions of the chunker represent a chunk candidate (s, e) with the following features:

- A **window at** x_s, that extracts the patterns defined above for the words $[x_{s-2}, x_{s-1}, x_s]$.
- A **window at** x_e, that extracts the patterns defined above for the words $[x_e, x_{e+1}, x_{e+2}]$.
- **Bag of words** and **bag of PoS** of the words from s to e.
- **PoS-Grams** on all subsequences of $[x_s, \ldots, x_e]$ of up to size 3, and also the **complete PoS-Gram** on the whole sequence.

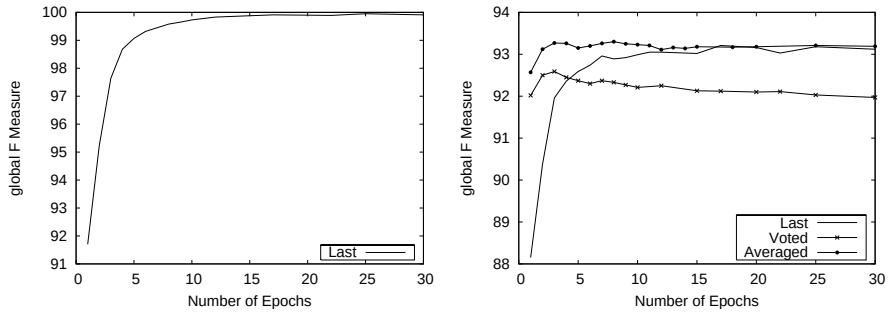

Figure 5.3: Learning curve on Syntactic Chunking for the eleven type FRP-Chunker. Left plot: training set in *last* prediction mode. Right plot: validation data (WSJ Section 21) in *last*, *averaged* and *voted* modes.

5.3.3 Results

We trained a model, named FRP-Chunker, for the 11 types of chunks in the data, with tree functions per chunk type. In CoNLL-2000, there was no official set for validating systems, and systems were tested in what we call the development set (corresponding to WSJ Section 20 —see Table 5.1). In our work, as well and in many others, we used the test set (WSJ Section 21) to tune the parameters of the system. Figure 5.3 plots the evolution of performance of FRP-Chunker in terms of global F_1 measure, on the training (left plot) and test (right plot) sets. As it can be seen, at epoch 15 the traning performance is almost at 100%, indicating that the data is separable under the chosen representation. Looking at the results on the validation data, the best result is obtained by the *averaged* prediction method after 8 learning epochs, with an F_1 measure at 93.3. Both *averaged* and *last* prediction methods perform fairly similar, although the former achieves much faster a better performance. In this problem, the *voted* method degrades the overall performance.

Training Single-Type FRP-Chunkers

Apart from training a multi-type chunker for 11 chunk categories, we were also interested in training 11 single-type chunkers, one for each chunk type. The motivation is two-fold. First, we were intested in experimenting whether training globally a multi-category chunker is advantageous over training several single-category chunkers and then combining them. Second, training a specific chunker for noun phrases allows us to compare with systems that have been recently developed for this particular task.

We trained each chunker for 10 epochs looking only at training annotations of the corresponding chunk type, and selected its best performing epoch on the validation data. Then, for recognizing chunks on test data, we put together the eleven single-category chunkers to form a single multi-category chunker.

	Multi-Type			Single-Type			Single-Type Comb.		
	Prec.	Recall	F_1	Prec.	Recall	F_1	Prec.	Recall	F_1
overall	94.20	93.38	93.79	-	-	-	94.58	92.83	93.69
ADJP	83.80	68.49	75.38	83.38	68.72	75.74	86.76	67.35	75.84
ADVP	85.29	79.68	82.39	83.94	77.25	80.46	85.70	77.48	81.38
CONJP	50.00	44.44	47.06	50.00	44.44	47.06	50.00	44.44	47.06
INTJ	100.00	100.00	100.00	00.00	00.00	00.00	00.00	00.00	00.00
LST	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00	00.00
NP	94.55	94.37	94.46	94.69	93.98	94.33	94.70	93.80	94.25
PP	96.50	98.13	97.31	96.92	97.36	97.14	97.33	97.17	97.25
PRT	78.82	63.21	70.16	75.24	74.53	74.88	82.76	67.92	74.61
SBAR	90.81	79.44	84.75	86.77	83.36	85.03	90.06	81.31	85.46
VP	93.90	93.22	93.56	93.50	93.19	93.34	94.22	93.11	93.66

Table 5.2: Performance in Chunking per phrase types, on the CoNLL-2000 test data (WSJ Section 20). The first triple of columns (Multi-Type) corresponds to the eleven type FRP-Chunker, working with 8 epochs. The second triple of columns presents the results obtained by individual, single-type FRP-Chunkers, each on their specific type. The third triple of columns corresponds to the combination of the 11 single-type FRP-Chunkers.

Table 5.2 presents the results of different filtering-ranking chunkers, detailed per chunk type. The first block (first three columns) corresponds to evaluation results of the multi-category chunker. The second block provides the results of each individual chunker applied separately to the test. The last block shows the results obtained by the combined chunker. As it can be seen, with the exception of two categories (NP and PRT) the results of the combined chunker are slightly better than in the individual performance. Note that in both architectures the local functions are the same, and consequently the local predictions are identical. Thus, the improvement of the combined chunker is caused by the inference process, that selects the most confident chunks to form a coherent, non-overlapping chunk sequence. Compared to the trained 11-category FRP-Chunker, the model combining independent chunkers achieves a similar performance, slightly lower in F_1 but generally better in precision.

This result seems to indicate that in this problem it is not particularly beneficial to globally learn dependent recognizers for many types of phrases at the same time. Therefore, it seems more practical to learn a separate recognizer for each phrase category, as far as, once combined, similar results are obtained for the multi-category setting. First, concentrating on a single category makes the training procedure much easier, since the learning effort (in terms of the number of epochs, complexity of the kernel, etc.) can be adapted to that category according to the complexity of its phrases. Second, from an engineering point of view, it is more flexible to have separate chunkers, because they can be combined at will depending on the requirements of a particular application.

5.3.4 Comparison to Other Works

We divide this discussion in two blocks: systems developed for the general syntactic chunking task, and systems addressing noun phrase chunking.

Syntactic Chunking. Table 5.3 provides the results of the top-performing methods published so-far on the test set of the Chunking task. As it can be seen, our system, named FRP-Chunker, is among the top systems for the problem, which achieve all similar performances. The methods with better results than our system perform the task as a tagging, solved with multiclass learning techniques. Kudo and Matsumoto [2001] performed several taggings which were later combined. Each tagging is produced with SVM classifiers performing a pairwise multiclass prediction. They report a performance of 93.85 with an individual tagging and 93.91 by combining many taggings in a majority voting scheme. Their system makes use of several hundreds of SVM classifiers applied to each word, whereas we only need 22 perceptrons for filtering words and 11 perceptrons for scoring phrases. In contrast, their feature space is simpler than ours, since in the score functions we exploit rich features on phrases. The second best performing work on the data set is by Zhang et al. [2002], which apply regularized Winnow. They report a base performance of 93.57, and an improved performance of 94.17 by making use of enhanced grammatical information (noted with +), which was unavailable to us. Up to date, the best work is by Ando and Zhang [2005]. This system was trained with semi-supervised learning techniques: apart from the official training data, they took advantage of 15 million words of unlabeled data to improve the performance of a linear model for sequential tasks. The system without using unlabeled data scores 93.60 (noted SVD, for singular value decomposition), and it improves to 94.39 with the semi-supervised technique (noted SVD-ASO, for SVD – alternating structure optimization). The last three rows of the table correspond to the best systems at competition time (CoNLL-2000).

Noun Phrase Chunking. Table 5.4 shows comparative results on recognizing individual noun phrases (`NP`). Apart from the referenced systems which apply to the complete chunking task, there have been recently many systems specifically trained for this category. It has to be noted that in the table there are systems working with the full set of chunk types, and others working specifically with `NP` phrases. Thus, training conditions are not exactly the same, since the latter only make use of `NP` phrase annotations. This remark is noted in the second column of the table. Our multi-chunker, achieving 94.46, is situated at second position in the current ranking, whereas our specific `NP`-chunker, at 94.33, obtains competitive performance. As in general chunking, the best result, at 94.70, is by Ando and Zhang [2005], that takes advantage of unlabeled data. Sha and Pereira [2003] obtained 94.38 with Conditional Random Fields. They also report 94.09 for the Markov tagging architecture globally trained with Perceptron of Collins [2002]. The latter work reports an F_1 at 93.53 on the `NP` chunking data originally defined by Ramshaw and Marcus [1995], which

is not identical to the CoNLL-2000 data but fairly comparable. On the same data, Punyakanok and Roth [2004] report F_1 at 94.15 with a conditional Markov model which makes use of probabilistic SNoW classifiers.

Reference	Technique	Prec.	Rec.	F_1
Ando and Zhang [2005]	SVD-ASO	94.57	94.20	94.39
Zhang et al. [2002]	Winnow (+)	94.28	94.07	94.17
Kudo and Matsumoto [2001]	SVM voting	93.89	93.92	93.91
Kudo and Matsumoto [2001]	SVM single	93.95	93.75	93.85
FRP-Chunker	**F&R VP**	**94.20**	**93.38**	**93.79**
Ando and Zhang [2005]	SVD	93.83	93.37	93.60
Zhang et al. [2002]	Winnow	93.54	93.60	93.57
Kudo and Matsumoto [2000]	SVM	93.45	93.51	93.48
van Halteren [2000]	MBL&WPD voting	93.13	93.51	93.32
Tjong Kim Sang [2000]	MBL voting	94.04	91.00	92.50

Table 5.3: Comparison on Syntactic Chunking of CoNLL-2000, with the top-performing systems published so-far on the test data. The last three rows correspond to the best systems at competition time (out of 11 systems presented in that edition, with F_1 measures ranging from 85.76 to 93.48).

Reference	S	Technique	Prec.	Rec.	F_1
Ando and Zhang [2005]	all	SVD-ASO	*unav.*	*unav.*	94.70
FRP-Chunker	**all**	**F&R VP**	**94.55**	**94.37**	**94.46**
Kudo and Matsumoto [2001]	all	SVM voting	94.47	94.32	94.39
Zhang et al. [2002]	all	Winnow (+)	94.39	94.37	94.38
Sha and Pereira [2003]	NP	CRF	*unav.*	*unav.*	94.38
FRP-Chunker	NP	**F&R VP**	**94.69**	**93.98**	**94.33**
Kudo and Matsumoto [2001]	all	SVM single	94.54	94.09	94.32
Sha and Pereira [2003]	NP	MM-VP	*unav.*	*unav.*	94.09
Zhang et al. [2002]	all	Winnow	93.80	93.99	93.89
Collins [2002]	NP	MM-VP	*unav.*	*unav.*	93.53

Table 5.4: Comparison on Noun Phrase Chunking, with the top-performing systems published so-far on the test data of CoNLL-2000. Second column indicates the scope of the chunker, namely whether the system works specifically for that chunk (NP), or for the full set of chunks (all).

5.4 Clause Identification

In this section we describe a clause identification system, developed in the setting of the CoNLL-2001 Shared Task [Tjong Kim Sang and Déjean, 2001]. The main particularity of this partial parsing task is that clauses in a sentence exhibit a recursive nature. Our participating system in that edition [Carreras and Màrquez, 2001] performed much better than the other five systems presented at competition time. It was the only system that applied learned classifiers at all the stages involved in the recognition process —including classifiers that deal with clause candidates. The other systems only applied learning in an initial tagging layer, and then used heuristics to build the recursive structure. The system discussed here makes use of FR-Perceptron to train all the learning components of the parser at the same time, globally. It substantially improves our previous results on the task.

Following, we summarize the main aspects of the strategy. Then, we describe the features used by the system. After that, we show the results of the clause parser, and compare them with other results published on the same data.

5.4.1 Strategy

The filtering-ranking strategy and the FR-Perceptron algorithm presented in Chapter 4 have been designed to recognize phrase hierarchies in a sentence. Hence, their application to this task is quite direct. To remind the reader, we sketch the strategy in the following two points:

- Since there is a single type of phrases, namely clauses, the architecture consists of three learning functions: the *start-end* filters, identifying possible clause candidates, and the *score* function, that predicts plausability scores for clause candidates.

- The inference process looks for a hierarchy of clauses. That is, clauses in a solution do not cross boundaries but admit embedding. It consists of a bottom-up exploration based on dynamic programming: for every span of words, in increasing length, it looks for the partial clause hierarchy that is optimal in terms of clause scores. For more details about this process, see Section 3.3.2.

As a particularity, the parser searches a clause structure that is coherent with the sequence of chunks recognized in the previous layer. That is, clauses that cross boundaries with some chunk in the input are not considered as possible clause candidates. On the positive side, this ensures that the complete partial syntax forms a coherent structure, and also avoids computation in the clausal layer (because much less words are possible start/end boundaries). On the negative side, chunk boundaries that are wrong can degrade more severely the performance on the clause layer.

5.4.2 Features

This section describes the features used by the clause identification system. Most features are the same that in Syntactic Chunking, while others are incorporated to describe linguistic phenomena that might be relevant to identify clause structure. The following list of features is the same than that of our previous work on this task [Carreras et al., 2002b], and, in turn, is based on the features we used for the system at competition time [Carreras and Màrquez, 2001].

Representation of Words

As in Syntactic Chunking, the *start-end* filters work with a window representation of half-size 2. For a window centered at the i-th word, the following feature extraction patterns are applied to $[x_j]_{j=i-2}^{i+2}$:

- **Word form** and **PoS tag**.
- **Chunk tags**, in BIO notation. (e.g.: "word at -1 is `B-NP`")
- **Left Start-End Flags**: For each word x_j to the left of x_i, two binary flags indicating whether x_j has been predicted as *start* and/or *end* words of a clause.
- **Counts**: Flags indicating whether a particular linguistic element appears 0, 1, 2 or more times within a sentence fragment. We consider two fragments, with separate features for each: from the beginning of the sentence to x_i, and from x_i to the end. The following elements are considered (separate counting flags for each):
  - Relative pronouns (e.g., "that")
  - Punctuation marks (`. , ; :`)
  - Quotes
  - Verb phrase chunks
  - Relative phrase chunks

Representation of Clause Candidates

The *score* function represents a clause candidate (s, e) as follows:

- A **window at** x_s and a **window at** x_e.
- **Top-most structure**: A pattern representing the relevant elements of the top-most structure forming the candidate, from s to e. In particular, the following elements are extracted to form the pattern:
  - Punctuation marks
  - Coordinate conjunctions (e.g., "or", "and")
  - The word "that"
  - Relative pronouns (e.g., "that", "which", "who", etc.)
  - Verb phrase chunks

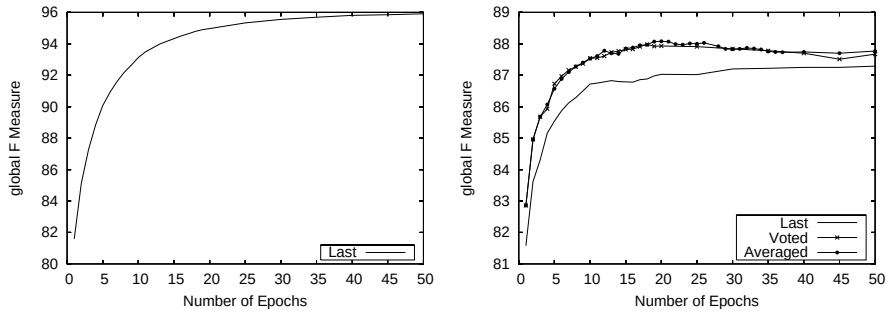

Figure 5.4: Learning curve on Clause Identification. Left plot: training set in *last* prediction mode. Right plot: development set in *last*, *averaged* and *voted* modes.

– The top clauses within the $[x_s, \ldots, x_e]$ fragment. These elements are found in the solution y that being incremented in the current span.

Note that the pattern only considers top-most structure. Thus, if a clause (or chunk) appears in the pattern, the elements within it are not considered. For example, in Figure 5.1, the pattern for the clause "((to raise)$_{\texttt{VP}}$ rates on containers (carrying U.S. exports to Asia)$_{\texttt{S}}$ about 10%)" is `VP_%_S_%`.

- **Counts** in the $[x_s, \ldots, x_e]$ fragment, considering the elements counted in the representation of words, as well as the number of clauses found inside the candidate.

5.4.3 Results

We trained the clause identification model for up to 50 epochs on the training data. Figure 5.4 shows the performance of the model (F_1 measure) on the training data (left plot) and on the development data (right plot). Clearly, the learning curve on the training set shows that the learning strategy effectively optimizes the global F_1 measure on the task through the learning epochs. However, the performance gets stable around 96, indicating that the training set is not separable under the current choice of features and kernel. Looking at the performance on the development set, the model shows a good generalization curve, with the best results over 88% and with no significant overfitting. Although at epoch 50 the FR-Perceptron has not converged, the generalization performance seems to be stable from epoch 20, showing only minor decreases in further epochs. Looking at the three prediction methods, both the *averaged* and the *voted* methods perform substantially better than the default *last* method.

Reference	Technique	Precision	Recall	F_1
FR-Perceptron	**F&R VP**	**88.17**	**82.10**	**85.03**
Carreras et al. [2002b]	AdaBoost class.	90.18	78.11	83.71
Carreras and Màrquez [2001]	AdaBoost class.	84.82	78.85	81.73
Molina and Pla [2001]	HMM	70.85	70.51	70.68
Tjong Kim Sang [2001]	Memory-based	76.91	65.22	70.58

Table 5.5: Comparison on Clause Identification of CoNLL-2001, with the top-performing systems published so-far on the test set. The last three rows correspond to the best systems at competition time (out of 6 systems presented in that edition, with F_1 performances ranging from 52.46 to 81.73).

5.4.4 Comparison To Other Works

Table 5.5 gives results of the top-performing published methods in this problem. FR-Perceptron clearly outperforms our previous system on the task [Carreras et al., 2002b], which consisted of a robust combination of AdaBoost classifiers working with the same filtering-ranking architecture. The scoring classifiers in that system were trained taking into account the type of errors produced in the filtering layer, as we did with the SVM models in Section 4.3.2. The online learning strategy we propose achieves better results automatically, mainly due to a rise in the recall rate. The other three works on the table correspond to the best systems at competition time.

5.5 Semantic Role Labeling (SRL)

We describe a system for the CoNLL-2004 Shared Task on Semantic Role Labeling (SRL) [Carreras and Màrquez, 2004]. The goal of the task is as follows: for a number of target verbs, that are given with each sentence, the system has to recognize the arguments of the propositions governed by such verbs, and label them with the appropriate semantic role. In a proposition arguments do not overlap, and thus form a chunking. However, we observe that arguments accross propositions never cross boundaries, that some of them coincide in boundaries, and that all together form a hierarchy of arguments. Building on these observations, we derive a filtering-ranking architecture that jointly recognizes the arguments of all propositions of a sentence. To our knowledge, it is the only system in literature that proceeds in this way, that is, other systems analyze each proposition independently.

Next, we explain the strategy for recognizing propositional arguments. Then, we describe the features of the system. Finally, we present the results obtained by our CoNLL-2004 system [Carreras et al., 2004], and contrast them with those of other systems presented in the task.

5.5.1 Strategy

The strategy for recognizing propositional arguments in sentences is based on two main observations about the argument structure in the data. Figure 5.5 illustrates the structure of chunks, clauses and arguments for a sentence.

- The first observation concerns the relation of the arguments of a proposition with the partial parse tree formed with chunks and clauses:
  - A proposition places its arguments in the clause directly containing the verb (local clause), or in one of the ancestor clauses.
  - Given a clause, we refer as the *top-most* syntactic elements to the words, chunks or clauses that are directly rooted at the clause. Propositional arguments are formed as subsequences of top-most elements of a clause.
  - Finally, for local clauses arguments are found strictly to the left or to the right of the target verb, whereas for ancestor clauses arguments are usually to the left of the verb. This observation holds for most of the arguments in the data.
- The second observation is that the arguments of all propositions of a sentence do not cross their boundaries, and that arguments of a particular proposition are usually found strictly within an argument of a higher level proposition. Thus, the problem can be thought of as finding a hierarchy of arguments in which arguments are embedded inside others, and each argument is related to a number of propositions of a sentence with a particular role. If an argument is related to a certain verb, no other argument linking to the same verb can be found within it.

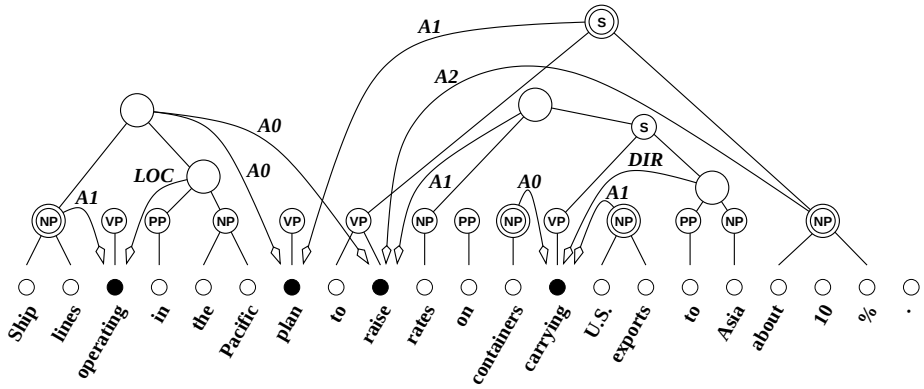

Figure 5.5: An example sentence, with phrase structures corresponding to chunks, clauses, and propositional arguments labeled with semantic roles. The small circles in the bottom line represent the words of the sentence. From these, those in black mark the verbs of the sentence: the goal of the SRL task is to recognize the arguments of these verbal predicates. The medium-sized circles, with a label in within, represent the syntactic phrases of the sentence. The `NP`, `VP` and `PP` are recognized in the chunking task, while the `S` are recognized in the clause identification task. To simplify the graph, the main clause (that governs the complete sentence) is not depicted. Finally, the big circles represent the arguments of the sentence. The labeled edges connecting an argument with a verb represent the semantic roles. Note that an argument can be related to one or many verbs. Note also that sometimes an argument matches exactly a syntactic phrase (e.g., "Ship lines" or "to raise rates ... about 10%"), and sometimes an argument is formed by many contiguous phrases (e.g., "in the Pacific"). Finally, note that the complete phrase structure, formed by chunks, clauses and arguments, is of hierarchical nature.

Our system for SRL translates these observations into constraints which are enforced to hold in a solution, and guide the recognition strategy. Next, we describe the model for SRL. Then, we sketch the strategy to find argument hierarchies in a sentence.

SRL Model

A semantic role is the relation of an argument (s, e) with a verb v. Hence, in this task the labels (i.e. semantic roles) are not assigned to arguments, but rather to argument-verb pairs. Formally, a solution $y \in \mathcal{Y}$ for a sentence x is a set of arguments of the form $(s, e)_v^k$, where (s, e) represents an argument spanning from word x_s to word x_e, playing a semantic role $k \in \mathcal{K}$ with a verb $v \in V$.

This particularity introduces a slight change in the filtering-ranking model presented in Chapter 4: here, the *score* function predicts plausability scores for a span (s, e) being argument of v with role k. The model is expressed as:

$$\mathrm{R}(x) = \arg \max_{y \subseteq \mathrm{F}(x) \mid y \in \mathcal{Y}} \sum_{(s,e)_k^v \in y} \mathrm{score}(x, y, v, (s,e)_k) \tag{5.1}$$

The *start-end* filters remain the same: they classify words for being boundaries of arguments playing a role k, and this prediction is done independently of verbs.

Building Argument Hierarchies

The exploration to build an argument hierarchy for a sentence is as follows:

- The sentence is explored by navigating through the clause hierarchy, in bottom-up order: from the inner-most clauses to the root clause.
- In each clause, a number of arguments is recognized. The solution for the sentence is the union of the arguments found in all clauses.
- In a clause spanning from word S to E, the process is as follows:
  1. The sequence of top-most elements of the clause is extracted. It consists of words, chunks or clauses that are found within (S, E) and that are directly rooted at the current clause.
  2. Argument candidates are formed with top-most elements. In particular, all subsequences of contiguous top-most elements are possible candidate arguments of the clause. At this point, the filter applies: a phrase (s, e) is a candidate argument for the semantic role k only if x_s and x_e receive a positive *start-end* prediction, respectively.
  3. Then, the target verbs found within the clause are considered: either a verb is *local* to the clause (the verb chunk is directly rooted in it), or it is at a *deeper level* (it is contained within a clause rooted at the current clause).
  4. Each candidate argument (s, e) is scored with each verb v of the clause, for semantic roles k that pass the filter. Before scoring, some further filtering conditions are applied: (a) the verb is not in within the argument ($v < s$ or $e < v$), and (b) if the verb is not local to the clause, the argument must be to the left of the verb ($e < v$).
  5. With dynamic programming, the best-scored argument hierarchy is built, checking that:
     - Arguments do not cross boundaries.
     - Arguments related to the same verb are not embedded.

This recognition procedure relies on the observations mentioned above, and ensures that the resulting structure is an argument hierarchy, where the arguments related to a verb form a chunking.

5.5.2 Features

The features of the system are extracted from three types of elements: words, target verbs, and arguments. They are formed making use of PoS tags, chunks and clauses of the sentence. Basically, the features we use are based on previous work on SRL, namely the pioneer system of Gildea and Jurafsky [2002], and those of Surdeanu et al. [2003] and Pradhan et al. [2005]. The main difference, though, is that the mentioned systems work on the top of a full syntactic tree, whereas in our approach we build on partial syntax only. Hence, most of the presented features can be thought as adaptations of the original features to our setting. Most of systems presented in CoNLL-2004 proceeded this way.

Representation of Words

As in previous tasks, features for the *start-end* decisions are extracted via word-windows placed at the candidate boundary words. Considering a window centered at word x_i, the following extraction patterns are applied to $[x_j]_{j=i-2}^{i+2}$:

- **PoS tag**.
- **Form**, if the PoS tag belongs to a closed category[4].
- **Chunk type**, of the chunk containing the word.
- **Binary-valued flags**, indicating structural properties of the word with respect to chunks and clauses enclosing the word:
  - Its chunk is one-word or multi-word
  - Starts and/or ends, or is strictly within a chunk (3 flags)
  - Starts and/or ends clauses (2 flags)
  - Aligned with a target verb;
  - First and/or last word of the sentence (2 flags)

Representation of Argument Candidates

The *score* function receives in its parameters a sentence x, a target verb v and an argument candidate (s, e) for playing the role k with the target verb. We divide the list of feature extraction patterns in three categories, depending on which part of the decision context they are applied:

- **Extraction on the Argument** (s, e)**:**
  - **Window-based features**, of words s–1, s, e, and e+1, each anchored with its relative position.
  - **PoS Sequence**, of PoS tags from x_s to x_e. In particular, (a) n-grams of size 2, 3 and 4; and (b) The complete PoS pattern, if it is less than 5 tags long.

[4]The PoS tag does not match with the Perl regexp: `/^(CD|FW|J|LS|N|POS|SYM|V)/`

  - **TOP sequence**: tags of the top-most elements found strictly from s to e. The tag of a word is its PoS. The tag of a chunk is its type. The tag of a clause is its type (`S`) enriched as follows: if the PoS tag of the first word matches `/^(IN|W|TO)/` the tag is enriched with the form of that word (e.g., `S-to`); if that word is a verb, the tag is enriched with its PoS (e.g., `S-VBG`); otherwise, it is just `S`. For example, in Figure 5.5, the `A1` argument for "raise", "`(` (rates)$_{\texttt{NP}}$ (on)$_{\texttt{PP}}$ (containers)$_{\texttt{NP}}$ (carrying U.S. exports to Asia)$_{\texttt{S}}$ `)`", is represented as `NP_PP_NP_S-VBG`. The following features are extracted from the sequence: (a) n-grams of sizes 2, 3 and 4; (b) The complete pattern, if it is less than 5 tags long; and (c) Anchored tags of the first, second, penultimate and last elements.
  - **Bag of Words**: we consider the top-most elements of the argument which are not clauses, and extract all nouns, adjectives and adverbs. We then form a separate bag for each category.
  - **Lexicalization**: the form of the head of the first top-most element of the argument, via the common head word rules by Collins [1999] adapted to partial syntax. If the first element is a `PP` chunk, we also extract the head of the first `NP` found.

- **Extraction on the Target Verb v:**

  - **Form**, **PoS tag**, and **infinitive form** of x_v.
  - **Voice** : *active* or *passive*. The heuristic rule associated to the indicator is as follows: x_v is in *passive* voice only if x_v has PoS tag `VBN`, and either its chunk is not `VP` or x_v is preceded by a form of "to be" or "to get" within its chunk.
  - **Chunk type** of the chunk that contains x_v.
  - **Binary-valued flags**, indicating structural properties of x_v with respect to chunks and clauses:
    - The chunk of x_v is multi-word or not
    - x_v starts and/or ends clauses (2 flags)

- **Extraction on the Verb-Arg Relation, $(s, e) \rightarrow v$:**

  - **Distance** of v to s and to e: for both pairs, a flag indicating if distance is $\{0, 1, -1, >1, <-1\}$.
  - **PATH sequence**: tags of elements found between the argument and the verb. It is formed by a concatenation of horizontal tags and vertical tags. The horizontal tags correspond to the TOP sequence of elements at the same level of the argument, from it to the phrase containing the verb, both excluded. The vertical part is the list of tags of the phrases which contain the verb, from the phrase at the level of the argument to the verb. The tags of the PATH sequence are extracted as in the TOP sequence, with an additional mark indicating whether an element is horizontal to the left or to the right of the

argument, or vertical. The following features are extracted: (a) n-grams of sizes 4 and 5; and (b) The complete pattern, if it is less than 5 tags long.

5.5.3 Results

We trained a system in CoNLL-2004 data that implements the presented architecture for recognizing arguments labeled with PropBank semantic roles. As a final consideration, the filtering component made no differentiation on the numbered arguments. In other words, the *start-end* functions considered a single type `AN` for the numbered types (`A0`–`A4`).

We ran the learning algorithm on the training set with a polynomial kernel of degree 2, for up to 8 epochs. Table 5.6 presents the obtained results on the development set for several configuration of the learning functions. In particular, we show results using perfect and/or learned filters and rankers. The first row provides results for perfect functions: the performance is not perfect due to limitations of our exploration (e.g., some of the observations we have made do not hold for some unfrequent cases). The second and third rows provide, respectively, the loss suffered because of errors in the filtering and scoring layer. The filtering layer performs reasonably well, since 89.44% recall can be achieved on the top of it. However, the scoring functions clearly moderate the performance, since working with perfect start-end functions only achieve an F_1 at 75.60. Finally, table 5.7 presents final detailed results on the test set.

5.5.4 Comparison to Other Works

Table 5.8 shows the overall results of our system together with those of the systems that contributed to the CoNLL-2004 shared task [Carreras and Màrquez, 2004]. Our system achieved the third position. The second system, of Punyakanok et al. [2004], obtained a performance very close to ours. They used two layers of winnow-based learners, that emitted predictions for argument boundaries and candidates. Interestingly, such predictions were feed into an ILP-based inference engine, that took into account several task-dependent constraints that were forced in the final solution. The top-performing system, of Hacioglu et al. [2004], transformed the SRL task into a BIO-based chunking task. They used Support Vector Machines as learners, that were greedily applied to syntactic chunks in the sentence to determine begin and inside parts of arguments.

A main characteristic of the 2004 shared task edition is that SRL systems work with partial syntax. Most of systems found in the literature, pioneered by that of Gildea and Jurafsky [2002], work with a full syntactic tree. In this setting, the most recent studies situate the SRL performance at around 80 in F_1 measure [Surdeanu et al., 2003; Pradhan et al., 2005]. The CoNLL-2005 Shared Task [Carreras and Màrquez, 2005] addressed again the SRL task, introducing full parsing information in the input, apart from partial parsing. One of the conclusions of that evaluation was that systems based on full syntax are superior in performance than systems based on partial syntax. However, all the

Configuration	Precision	Recall	$F_{\beta=1}$
`gold`–*start-end*, `gold`–*score*	99.92	94.73	97.26
start-end, `gold`–*score*	99.90	89.44	94.38
`gold`–*start-end*, *score*	85.12	67.99	75.60
start-end, *score*	73.40	63.70	68.21

Table 5.6: Performance in Semantic Role Labeling, on the development set. Functions with prefix `gold` are gold functions, providing bounds of our performance. The top row is the upper bound performance of our architecture. The bottom row is the real performance.

	Precision	Recall	$F_{\beta=1}$
Overall	**71.81**	**61.11**	**66.03**
A0	81.83	76.46	79.05
A1	68.73	65.27	66.96
A2	59.41	34.03	43.28
A3	58.18	21.33	31.22
A4	72.97	54.00	62.07
A5	00.00	00.00	00.00
AM-ADV	54.50	35.50	43.00
AM-CAU	58.33	28.57	38.36
AM-DIR	64.71	22.00	32.84
AM-DIS	64.06	57.75	60.74
AM-EXT	100.0	50.00	66.67
AM-LOC	35.62	22.81	27.81
AM-MNR	50.89	22.35	31.06
AM-MOD	97.57	95.25	96.40
AM-NEG	90.23	94.49	92.31
AM-PNC	36.11	15.29	21.49
AM-PRD	00.00	00.00	00.00
AM-TMP	61.86	48.86	54.60
R-A0	78.85	77.36	78.10
R-A1	64.29	51.43	57.14
R-A2	100.0	22.22	36.36
R-A3	00.00	00.00	00.00
R-AM-LOC	00.00	00.00	00.00
R-AM-MNR	00.00	00.00	00.00
R-AM-PNC	00.00	00.00	00.00
R-AM-TMP	00.00	00.00	00.00
V	98.32	98.24	98.28

Table 5.7: Performance in Semantic Role Labeling on the test set, detailed for each semantic role. The role denoting the verbal predicate (`V`) is excluded in the overall measures.

Reference	Algorithm	Precision	Recall	F_1
Hacioglu et al. [2004]	SVM	72.43	66.77	69.49
Punyakanok et al. [2004]	Winnow	70.07	63.07	66.39
Carreras et al. [2004]	**FR-Perceptron**	**71.81**	**61.11**	**66.03**
Lim et al. [2004]	Max-Entropy	68.42	61.47	64.76
Park et al. [2004]	SVM	65.63	62.43	63.99
Higgins [2004]	TBL	64.17	57.52	60.66
van den Bosch et al. [2004]	Memory-Based	67.12	54.46	60.13
Kouchnir [2004]	Memory-Based	56.86	49.95	53.18
Baldewein et al. [2004]	Max-Entropy	65.73	42.60	51.70
Williams et al. [2004]	TBL	58.08	34.75	43.48

Table 5.8: Comparison on Semantic Role Labeling of CoNLL-2004, for the ten systems presented in that edition. Results correspond to test data. The second column corresponds to the learning algorithm used by each system. To explain the difference in performance, though, other components of the systems need to be considered, such as the SRL recognition strategy and the features (see the introductory paper of the task for a detailed comparison [Carreras and Màrquez, 2004])

.

top-performing systems of the 2005 edition made use of several syntactic views (chunks, clauses, and several different full parse trees) by means of system combination. In our opinion, the proper level of syntax needed for SRL is still an open question for future research.

5.6 Conclusion of this Chapter

This chapter has presented the application of the filtering-ranking strategy to three phrase recognition problems, under the settings proposed in CoNLL Shared Task series. We have shown that the presented architecture can be adapted to recognize phrase structures of different characteristics: a single or several phrase types, and sequential or recursive phrase structures. In all cases, our systems obtain a good performance in the CoNLL Shared Task evaluations. It can be concluded, thus, that our systems are situated among the top-performing systems in the state-of-the-art.

Chapter 6

Conclusion

To conclude the thesis, we summarize the contributions and results of this work, and we discuss some directions for future research.

6.1 Summary and Results

We have developed a framework for the problem of recognizing sequential and hierarchical phrase structures. The framework implements the idea that phrase structures can be recognized with learning techniques and inference processes. The strategies we study decompose a phrase recognition problem into many decisions that are learnable (or, at least, assumed to be so). Inference is the process that combines such decisions to recognize the phrase structure of a sentence, efficiently and robustly. The learning decisions consist of predicting which pieces of a sentence structure correspond to words or groups of words of the sentence. We have derived this idea to word-based and phrase-based learners, with appropriate decompositions for them. Then, we have described incremental inference strategies for these decompositions. Due to incrementality, the type of inference allows to use a partial structure that has been recognized to support the prediction of a piece that increments such partial structure. This property allows learners to benefit from high-order expressive features that exploit dependencies between input-output parts of the data. We have shown that, on the one hand, exploiting such dependencies benefits the accuracy of learners, while requires more complex exploration strategies to guarantee exact inference. On the other hand, if very accurate learners can be obtained, one can introduce pruning conditions in the exploration that make inference more efficient, such as discarding low-scored exploration paths. So, a main question is what level of dependencies are needed to obtain accurate learned decisions, and to which extent the inference process can rely on the output of learners to discard solutions.

The main contribution of the thesis is a particular phrase recognition architecture based on filters and rankers. This architecture goes a step further

than the common word-based tagging approaches used for chunking problems. Our filtering-ranking approach applies a tagging on words to identify possible phrases in a sentence –or, in other words, to filter out most of the possibilities. Then, given the reduced set of phrase candidates, it predicts confidence scores for them and builds the best phrase structure for the sentence, considering dependencies between phrases of a same structure.

Together with the phrase recognition strategy, we have presented a global learning algorithm for it, that we name FR-Perceptron. The FR-Perceptron algorithm is based on the global Perceptron scheme by Collins [2002], and is particularly tailored for the task of recognizing phrases with a filtering-ranking approach. Specifically, the algorithm learns the filters and rankers of the architecture at the same time, aiming at benefiting from the interactions that these functions exhibit within the architecture.

Such techniques have been put into practice in real phrase recognition tasks, such as syntactic chunking, clause identification and semantic role labeling. First, we have designed a series of experiments to test the behavior of FR-Perceptron, and to contrast it with other learning algorithms. From these experiments, we conclude that FR-Perceptron effectively approximates the learning functions of the architecture as filters and rankers, and that this fact positively contributes to better evaluation measures of the whole architecture. Second, the resulting phrase recognition systems trained with FR-Perceptron are very competitive in performance, contrasted with other systems in the literature. For the sequential task, syntactic chunking, our system achieves evaluation scores very close to the top-performing system. On the identification of clause hierarchies, a task more complex than chunking due to its recursive nature, our system obtains the best result on the task, and outperforms a previous result from us. Finally, on the recognition of semantic roles on the basis of partial syntactic information, our architecture is among the best ones.

6.2 Future Directions

We differentiate five main directions of future research.

6.2.1 From Greedy to Robust Inference

Directly related to the research presented in this thesis, we would like to conduct a series of experiments to test the influence of robustifying the inference strategy for a fixed model, in the context of syntactic chunking and clause identification.

In particular, for the filtering-ranking phrase recognition model, we aim to explore different inference strategies that go from greedy to exact global optimizations. These strategies, sketched in Section 3.3.2, build the phrase structure incrementally by visiting phrase spans of the sentence. The most greedy strategy assigns the final local structure in a span when visiting it, without the possibility of trading off such assignment with competing structure that would be recognized later. In contrast, an exact strategy maintains all plausible so-

lutions, and trades off competing local assignments to build the optimal global structure. In the middle of these, there exist a range of approximate inference strategies that rely on the ability of the learning functions at predicting negative scores for non-plausible local assignments, and, with them, explore only the phrase structures formed of positive local assignments.

In the series of experiments, such architectures would be trained with FR-Perceptron, to guarantee that the interactions between the learning functions that take place in the inference process can be properly captured. Moreover, each inference strategy can be tested with a set of functions trained independently of inference. Thus, the series of experiments would explore different degrees of robustness in inference, and the influence of training locally or globally such architectures.

Our intuition is that, in general, evaluation results will increase with more robust inference algorithms. However, we expect also to observe that approximate inference can lead to very competitive results, being much more efficient than exact inference. If this idea is confirmed, we would like to devise what is the most desired configuration in the trade-off between efficiency and robustness in learning and inference phrase recognition systems.

6.2.2 Learning issues for FR-Perceptron

We outline three research lines related to the learning strategy of FR-Perceptron.

- The convergence analysis of FR-Perceptron we have presented relies on strong assumptions that, in practice, are not needed. In particular, the filtering functions are analyzed as general binary classifiers, and it is assumed that perfect separability in the filtering stage is possible. However, these functions act only as a filter, that is, it is not critical that filters produce false positive errors. Ideally, the analysis of FR-Perceptron has to show that the algorithm converges as long as: (1) the filters do not block any correct phrase, and (2) phrases that pass the filter can be properly scored so that the correct structure is the top-ranked one. Moreover, we would like to analyze whether by approximating the filters close to a perfect separation, the number of training errors needed to train the ranking layer decreases, as a consequence of the simplification that filters produce in the input space of the ranker. Hopefully, gaining more understanding on the theoretical side will give clues on how to improve the FR-Perceptron learning strategy.

- FR-Perceptron learns by visiting one example at a time, updating the learning functions depending on the predictions on that example. In big training sets –such as those we work with, with tens of thousands of sentences, and many local predictions at each sentence– this simple strategy can be a bit inefficient. Generally, the practical situation is that after visiting a relatively small number of examples, the learning functions produce few errors. However, to encounter such errors the learner has to visit

many examples that are already well-predicted. Since visiting an example is time-consuming, we would like to incorporate a strategy for concentrating on difficult examples. In the literature, Collins and Roark [2004] maintain a cache of examples in which Perceptron has failed, which are visited after every update. Also related, the maximum margin approaches for structured-output recognition [Taskar et al., 2003; Tsochantaridis et al., 2004] maintain an active set of learning constraints (i.e., related to *support* examples) that are used to estimate the optimal separator.

- FR-Perceptron takes advantage of being a mistake-driven online algorithm to train many functions dependently. It is based on Perceptron, which learns by additive function updates. We would like to experiment with Winnow-based updates [Littlestone, 1988], that is, multiplicative updates that are known to be more efficient than additive updates in the present of many irrelevant features.

6.2.3 Natural Language Tasks

We would like to move to other complex Natural Language Processing tasks which involve the recognition of structures, and can be decomposed into many simple steps. In these tasks, we would like to apply global learning, and test the benefits of training dependently several functions that work at different levels. Examples of these tasks include:

- Two annotation levels in one task. For example, part-of-speech tagging and chunking, or recognizing partial syntax consisting of chunks and clauses.

- Full parsing task, as complex task that deals with fine-grained syntactic trees.

- Basic Syntax and Semantic Roles. This task consists of recognizing a basic level of syntax (not necessarily full), together with the argument structures of the predicates in the sentence. An argument of a predicate always corresponds to some syntactic constituent of the sentence, and plays a semantic relation with the corresponding predicate. Thus, this task is attractive because it lies at the syntactico-semantic level, and allows to exploit coherence of a solution.

For English, the Penn TreeBank [Marcus et al., 1993], together with PropBank [Palmer et al., 2005], contains enough annotations to derive datasets for the mentioned tasks. In addition, we are concerned with building phrase recognition systems for Catalan and Spanish, making use of the recently released 3LB TreeBank for such languages [Palomar et al., 2003]. The analyzers obtained in this study would be useful to build language applications, such as information extraction or question answering systems.

6.2.4 Introducing Knowledge

The phrase recognition architectures we have discussed allow to introduce linguistic and task-specific knowledge to support the recognition process.

Up to now, the only type of knowledge is basically in terms of the features that represent a learning instance. While most of the features capture exhaustively patterns that are based on positions (e.g., *"extract the PoS of the current word"*), others look for specific linguistic patterns or other knowledge-based phenomena (e.g, *"check whether there is only one transitive verb in the span"*).

A second possibility is to introduce knowledge that guides the exploration for building the phrase structure. Actually, the clause recognizer we have presented follows the heuristic of considering only clauses that do not overlap with the chunks provided in the input. This heuristic reduces substantially the search space of the clause recognizer, while ensures that the clause structure will be coherent with the chunk structure (note that, when chunks are not perfectly correct, following this heuristic may degrade the accuracy on clauses). As the complexity of the structures increases (specially if we aim to resolve several layers dependently), there is a need to use a principled mechanism, such as a grammar, that dynamically controls the phrases that are visited during the exploration, given the phrases that are already recognized and those provided in the input. In this line, the general architecture is that of a parser that is trained discriminatively to select the best structure generated by the grammar, as in the systems proposed by Collins and Roark [2004], Taskar et al. [2004] or Tsochantaridis et al. [2004]. In our opinion, the underlying grammar of a discriminative learning-based parser must be kept simple, and be used to avoid visiting invalid solutions. Consequently, the exploration becomes faster and the disambiguation problem becomes simpler. For instance, we would like to incorporate the grammar formalisms for chunk-based structure proposed by Abney [1991]. Then, on the top of this informed exploration, the learning components carry out the disambiguation of valid candidate structures, possibly making use of rich lexical information and other type of knowledge.

Following the ideas of Roth and Yih [2004], the third possibility for introducing knowledge is thought as constraints on the global solution, that express dependencies between different elements of the structure. Such dependencies have to be satisfied to ensure coherence of the global structure, but, due to their global nature, they are not considered during the incremental recognition. For example, in the context of syntactic chunking, one may require that there must be at least one verb phrase in the chunk sequence of a sentence. Or, one may run simultaneously a named entity recognizer and a chunker, each independent of the other, and then require coherence between the resulting structures. So, in the resulting architecture, the generation mechanism defines which solutions are well-formed, and the global constraints define which well-formed solutions are coherent. The former generates a number of scored global solutions, and the later are used to select the top-scored one that satisfies the constraints.

6.2.5 On Representations and Kernels

The family of learning algorithms that accept kernel functions —known as kernel methods [Schölkopf and Smola, 2002]— offer the possibility to make use of kernel functions that are specially designed to deal with structured data. Such kernel functions are known as convolution kernels [Haussler, 1999], and compute the similarity between two given structures by looking at common substructures. In literature related to learning in Natural Language, this general idea has been developed for string kernels [Lodhi et al., 2002], kernels for sequences and trees [Collins and Duffy, 2001, 2002], and kernels for data represented in directed acyclic graphs [Suzuki et al., 2003].

While the representation spaces induced by convolution kernels are very powerful, a main drawback is that computing a kernel operation between two structures with a convolution kernel is time-consuming, although being of polynomial cost. Moreover, with kernel methods, computing a prediction translates into computing a kernel operation for every dual vector that composes the predictor. In complex domains, such as natural language, a predictor can be composed of tens thousands of dual vectors, which makes the computation of a prediction very time consuming.

Due to this fact, up to now structured kernel functions have been applied only to classification and reranking tasks, where there is a reasonable number of predictions to be computed. To our knowledge, for tasks that require intensive use of predictors, such as segmenting a sentence into a structure, there are no works on applying structured kernel functions.

It turns out that, with structured data, both a recognition strategy and a kernel calculation are based on a divide and conquer scheme. We would like to gain understanding on both processes, and to come up with a unified kernel-based recognition strategy, that builds up structure from the bottom up, calculating kernel operations and incrementing structure at the same time. We believe that, by unifying these processes, the overall computational cost could be high but feasible.

A complementary direction is to explore the trade off in computational cost between working with the dual form (facing the problem of dealing with many dual vectors), or in primal form (facing the problem of dealing with a feature space of very large dimensionality). For polynomial kernels, the primal form has been shown to be more efficient than the dual form [Kudo and Matsumoto, 2003]. For more sophisticated representations, Cumby and Roth [2003] have proposed a principled language for specifying a desired representation. This language contemplates propositionalized relational patterns, as well as kernel-like feature expansions. Then, a specification of a representation can be translated into a kernel function for working in dual form, or an equivalent feature extraction function for working in primal form.

Bibliography

S. Abney. Rapid incremental parsing with repair. In *Proceedings of the 6th New OED Conference: Electronic Text Research*, pages 1–9, Waterloo, Ontario, 1990.

S. Abney. Partial parsing via finite-state cascades. In *Proceedings of the ESSLLI'96 Robust Parsing Workshop*, Prache, Czech Republic, 1996a.

S. P. Abney. Parsing by chunks. In R. C. Berwick, S. P. Abney, and C. Tenny, editors, *Principle-based parsing: Computation and Psycholinguistics*, pages 257–278. Kluwer, Dordrecht, 1991.

S. P. Abney. Part-of-speech tagging and partial parsing. In Ken Church, Steve young, and Gerrit Bloothooft, editors, *Corpus-Based Methods in Language and Speech*. Kluwer, Dordrecht, 1996b.

Y. Altun, T. Hofmann, and M. Johnson. Discriminative learning for label sequences via boosting. In *Advances in Neural Information Processing Systems (NIPS-15)*, 2002.

Y. Altun, I. Tsochantaridis, and T. Hofmann. Hidden Markov Support Vector Machines. In *Proceedings of the 20th International Conference on Machine Learning, ICML-03*, Washington, DC USA, 2003.

Rie Ando and Tong Zhang. A high-performance semi-supervised learning method for text chunking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 1–9, Ann Arbor, Michigan, 2005.

S. Argamon, I. Dagan, and Y. Krymolowski. A memory-based approach to learning shallow natural language patterns. *Journal of Experimental and Theoretical AI*, 11:369–390, 1999.

U. Baldewein, K. Erk, S. Padó, and D. Prescher. Semantic role labeling with chunk sequences. In *Proceedings of CoNLL-2004*, 2004.

C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Knowledge Discovery and Data Mining*, 2(2), 1998.

X. Carreras and L. Màrquez. Boosting trees for clause splitting. In *Proceedings of CoNLL-2001*, pages 73–75. Toulouse, France, 2001.

X. Carreras and L. Màrquez. Online Learning via Global Feedback for Phrase Recognition. In *Proceedings of the 17th Annual Conference on Neural Information Processing Systems, NIPS-03*, Vancouver, Canada, 2003a.

X. Carreras and L. Màrquez. Phrase Recognition by Filtering and Ranking with Perceptrons. In *Proceedings of the 4th Conference on Recent Advances on Natural Langugae Processing, RANLP-03*, Borovets, Bulgaria, 2003b.

X. Carreras and L. Màrquez. Introduction to the CoNLL-2004 Shared Task: Semantic Role Labeling. In *Proceedings of the 8th Conference on Natural Language Learning, CoNLL-2004*, Boston, MA USA, 2004.

X. Carreras and L. Màrquez. Introduction to the CoNLL-2005 Shared Task: Semantic Role Labeling. In *Proceedings of the 9th Conference on Natural Language Learning, CoNLL-2005*, Ann Arbor, MI USA, 2005.

X. Carreras, L. Màrquez, and J. Castro. Filtering-ranking perceptron learning for partial parsing. *Machine Learning*, 59:1–31, 2005.

X. Carreras, L. Màrquez, and G. Chrupała. Hierarchical Recognition of Propositional Arguments with Perceptron. In *Proceedings of the 8th Conference on Natural Language Learning, CoNLL-2004*, Boston, MA, USA, 2004.

X. Carreras, L. Màrquez, and L. Padró. Named Entity Extraction using AdaBoost. In *Proceedings of the 6th Conference on Natural Language Learning, CoNLL-2002*, Taipei, Taiwan, 2002a.

X. Carreras, L. Màrquez, and L. Padró. A Simple Named Entity Extractor Using AdaBoost. In *Proceedings of the 7th Conference on Natural Language Learning, CoNLL-2003*, Taipei, Taiwan, 2003a.

X. Carreras, L. Màrquez, and L. Padró. Learning a Perceptron-Based Named Entity Chunker via Online Recognition Feedback. In *Proceedings of the 7th Conference on Natural Language Learning, CoNLL-2003*, Taipei, Taiwan, 2003b.

X. Carreras, L. Màrquez, V. Punyakanok, and D. Roth. Learning and Inference for Clause Identification. In *Proceedings of the 14th European Conference on Machine Learning, ECML-02*, Helsinki, Finland, 2002b.

E. Charniak. *Statistical Language Learning*. MIT Press, 1993.

E. Charniak. A maximum-entropy-inspired parser. In *Proceedings of NAACL-2000*, 2000.

S. Chen and J. Goodman. An empirical study of smoothing techniques for language modelling. In *Proceedings of the 34th Meeting of the Association for Computational Linguistics*, pages 310–318, 1996.

K. W. Church. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of 2nd ACL Workshop on Applied Natural Language Processing (ANLP)*, Austin, Texas, 1988.

M. Collins. *Head-driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, 1999.

M. Collins. Discriminative reranking for natural language parsing. In *Proceedings of the 17th International Conference on Machine Learning, ICML-00*, Stanford, CA USA, 2000.

M. Collins. Discriminative training methods for hidden markov models: Theory and experiments perceptron algorithms. In *Proceedings of the SIGDAT Conference on Empirical Methods in Natural Language Processing, EMNLP-02*, 2002.

M. Collins. Parameter estimation for statistical parsing models: Theory and practice of distribution-free methods. In Harry Bunt, John Carroll, and Giorgio Satta, editors, *New Developments in Parsing Technology*, chapter 2. Kluwer, 2004.

M. Collins and N. Duffy. Convolution kernels for natural language. In *Proceedings of NIPS'01*, 2001.

M. Collins and N. Duffy. New Ranking Algorithms for Parsing and Tagging: Kernels over Discrete Structures, and the Voted Perceptron. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, ACL'02*, 2002.

M. Collins and T. Koo. Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–69, 2005.

M. Collins and B. Roark. Incremental parsing with the perceptron algorithm. In *Proceedings of the ACL*, 2004.

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

K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2(5): 265–292, 2001.

K. Crammer and Y. Singer. A family of additive online algorithms for category ranking. *Journal of Machine Learning Research*, 3:1025–1058, 2003a.

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

N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, Cambridge, UK, 2000.

C. Cumby and D. Roth. On Kernel Methods for Relational Learning. In *Proceedings of the 20th International Conference on Machine Learning, ICML-2003*, Washington, DC USA, 2003.

W. Daelemans, A. van den Bosch, and T. Weijters. Empirical learning of natural language processing tasks. In *Proceedings of the ECML-97*, pages 337–344, Berlin, Germany, 1997.

I. Dagan and Y. Krymolowski. Compositional partial parsing by memory-based sequence learning. In R. Bod, R. Scha, and K. Sima'an, editors, *Data-Oriented Parsing*, chapter ? CSLI Publications, 2001.

T. G. Dietterich. Machine learning for sequential data: A review. In *Structural, Syntactic, and Statistical Pattern Recognition*, 2002.

E. Ejerhed. Finding clauses in unrestricted text by finitary and stochastic methods. In *Proceedings of the 2nd ACL Workshop on Applied Natural Language Processing (ANLP)*, Austin, Texas, 1988.

Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, 1999.

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

G. Grefenstette. Light parsing as finite-state filtering. In *Workshop on Extended Finite State Models of Language, ECAI'96*, 1996.

K. Hacioglu, S. Pradhan, W. Ward, J. Martin, and D. Jurafsky. Semantic role labeling by tagging syntactic chunks. In *Proceedings of CoNLL-2004*, 2004.

J. Hammerton, M. Osborne, S. Armstrong, and W. Daelemans. Introduction to the special issue on machine learning approaches to shallow parsing. *Journal of Machine Learning Research*, 2:551–558, 2002.

S. Har-Peled, D. Roth, and D. Zimak. Constraint Classification for Multiclass Classification and Ranking. In *Proceedings of the 16th Annual Conference on Neural Information Processing Systems, NIPS-02*, 2002.

D. Haussler. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, University of California at Santa Cruz, 1999.

D. Higgins. A transformation-based approach to argument labeling. In *Proceedings of CoNLL-2004*, 2004.

T. Joachims. Making large-scale svm learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*. MIT Press, 1999.

M. Johnson. Joint and conditional estimation of tagging and parsing models. In *Proceedings of the 39th meeting of the ACL*, 2001.

M. Johnson, S. Geman, S. Canon, Z. Chi, and S. Riezler. Estimators for stochastic "unification-based" grammars. In *Proceedings of the 37th meeting of the ACL*, 1999.

M. I. Jordan. Graphical models. *Statistical Science, Special Issue on Bayesian Statistics*, 19:140–155, 2004.

D. Jurafsky and J. H. Martin. *An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice-Hall, 2000.

M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory*. The MIT Press, Cambridge, MA, 1994.

D. Klein and C. Manning. Conditional Structure versus Conditional Estimation in NLP Models. In *Proceedings of EMNLP*, 2002.

B. Kouchnir. A memory-based approach for semantic role labeling. In *Proceedings of CoNLL-2004*, 2004.

T. Kudo and Y. Matsumoto. Use of Support Vector Learning for Chunk Identification. In *Proceedings of CoNLL-2000 and LLL-2000*, Lisbon, Portugal, 2000.

T. Kudo and Y. Matsumoto. Chunking with Support Vector Machines. In *Proceedings of the North American Chapter of the Association for Computational Linguistics Conference, NAACL-01*, 2001.

T. Kudo and Y. Matsumoto. Fast Methods for Kernel-based Text Analysis. In *Proceedings of the 41st Meeting of the ACL (ACL-2003)*, Sapporo, Japan, 2003.

J. Lafferty, A. McCallum, and F. Pereira. Conditonal random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning, ICML-01*, 2001.

X. Li and D. Roth. Exploring evidence for shallow parsing. In *Proceedings of CoNLL-2001*, pages 107–110, Toulouse, France, 2001.

J. Lim, Y. Hwang, S. Park, and H. Rim. Semantic role labeling using maximum entropy model. In *Proceedings of CoNLL-2004*, 2004.

N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text Classification using String Kernels. *Journal of Machine Learning Research*, 2:419–444, 2002.

D. M. Magerman. Learning Grammatical Structure Using Statistical Decision–Trees. In *Proceedings of the 3rd International Colloquium on Grammatical Inference, ICGI*, pages 1–21, 1996. Springer-Verlag Lecture Notes Series in Artificial Intelligence 1147.

M. P. Marcus, B. Santorini, and M. Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2): 313–330, June 1993.

Lluís Màrquez. *Part-of-speech Tagging: A Machine Learning Approach based on Decision Trees*. PhD thesis, Universitat Politècnica de Catalunya, 1999.

A. McCallum, D. Freitag, and F. Pereira. Maximum entropy markov models for information extraction. In *Proceedings of the 17th International Conference on Machine Learning*, Stanford, CA USA, 2000.

T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

A. Molina and F. Pla. Clause Detection using HMM. In *Proceedings of the 5th Conference on Natural Language Learning, CoNLL-2001*, Tolouse, France, 2001.

M. Muñoz, V. Punyakanok, D. Roth, and D. Zimak. A Learning Approach to Shallow Parsing. In *Proceedings of the joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora, EMNLP/VLC*, 1999.

A. B. J. Novikoff. On convergence proofs on perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume XII, pages 615–622, 1962.

Ll. Padró. *A Hybrid Environment for Syntax-Semantic Tagging*. PhD thesis, Universitat Politècnica de Catalunya, 1997.

M. Palmer, D. Gildea, and P. Kingsbury. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1), 2005.

M. Palomar, M. Civit, A. Díaz, L. Moreno, E. Bisbal, M. Aranzabe, A. Ageno, M.A. Martí, and B. Navarro. 3LB: Construcción de una base de datos de árboles sintáctico-semánticos para el catalán, euskera y castellano. In *Proceedings of the "XX Congreso Anual de la Sociedad Española para el Procesamiento del Lenguaje Natural (SEPLN)"*, Barcelona, Spain, 2003.

K. Park, Y. Hwang, and H. Rim. Two-phase semantic role labeling based on support vector machines. In *Proceedings of CoNLL-2004*, 2004.

S. Pradhan, K. Hacioglu, V. Krugler, W. Ward, J. Martin, and D. Jurafsky. Support vector learning for semantic argument classification. *Machine Learning. Special issue on Speech and Natural Language Processing*, 2005. To appear.

V. Punyakanok and D. Roth. The use of classifiers in sequential inference. In *Proceedings of the 15th Annual Conference on Neural Information Processing Systems, NIPS-01*, 2001.

V. Punyakanok and D. Roth. Inference with classifiers: The phrase identification problem. *Journal of Machine Learning Research*, 2004.

V. Punyakanok, D. Roth, W. Yih, D. Zimak, and Y. Tu. Semantic role labeling via generalized inference over classifiers. In *Proceedings of CoNLL-2004*, 2004.

L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285, 1989.

L. A. Ramshaw and M. P. Marcus. Text chunking using transformation-based learning. In *Proceedings of the Third Annual Workshop on Very Large Corpora*, 1995.

A. Ratnaparkhi. *Maximum Entropy Models for Natural Language Ambiguity Resolution*. PhD thesis, University of Pennsylvania, 1998.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–407, 1958.

D. Roth. Learning to resolve natural language ambiguities: A unified approach. In *AAAI*, pages 806–813, 1998.

D. Roth. Learning in natural language. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 898–904, 1999.

D. Roth and W. Yih. A linear programming formulation for global inference in natural language tasks. In *Proceedings of CoNLL-2004*, pages 1–8, Boston, MA USA, 2004.

Y. D. Rubinstein and T. Hastie. Discriminative vs informative learning. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, pages 49,53. AAAI Press, 1997.

R. E. Schapire. The boosting approach to machine learning. an overview. In *Proceedings of the MSRI Workshop on Nonlinear Estimation and Classification*, Berkeley, CA, 2002.

R. E. Schapire, Y. Freund, P. Barlett, and W. S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, 1998.

R. E. Schapire and Y. Singer. Improved Boosting Algorithms Using Confidence-rated Predictions. *Machine Learning*, 37(3), 1999.

B. Schölkopf and A. Smola. *Learning with Kernels. Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, 2002.

F. Sha and F. Pereira. Shallow parsing with conditional random fields. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, HLT-NAACL*, 2003.

W. Skut and T. Brants. Chunk tagger. In *Proceedings of the ESSLLI-98 Workshop on Automated Acquisition of Syntax and Parsing*, Saarbrücken, Germany, 1998.

A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans. *Advances in Large Margin Classifiers*. The MIT Press, Cambridge, MA, 2000.

M. Surdeanu, S. Harabagiu, J. Williams, and P. Aarseth. Using predicate-argument structures for information extraction. In *Proceedings of ACL 2003*, Sapporo, Japan, 2003.

C. Sutton, K. Rohanimanesh, and A. McCallum. Dynamic Conditional Random Fields: Factorized Probabilistic Models for Labeling and Segmenting Sequence Data. In *Proceedings of the 21st International Conference on Machine Learning, ICML-04*, Alberta, Canada, 2004.

J. Suzuki, T. Hirao, Y. Sasaki, and E. Maeda. Hierarchical directed acyclic graph kernel: Methods for structured natural language data. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL-2003)*, pages 32–39, 2003.

B. Taskar, C. Guestrin, and D. Koller. Max-Margin Markov Networks. In *Proceedings of the 17th Annual Conference on Neural Information Processing Systems, NIPS-03*, Vancouver, Canada, 2003.

B. Taskar, D. Klein, M. Collins, D. Koller, and C. Manning. Max-margin parsing. In *Proceedings of the EMNLP-2004*, 2004.

E. Tjong Kim Sang. Text Chunking by System Combination. In *Proceedings of CoNLL-2000 and LLL-2000*, Lisbon, Portugal, 2000.

E. Tjong Kim Sang. Memory-based clause identification. In *Proceedings of CoNLL-2001*, 2001.

E. Tjong Kim Sang. Introduction to the CoNLL-2002 Shared Task: Language-Independent Named Entity Recognition. In *Proceedings of the 6th Conference on Natural Language Learning, CoNLL-2002*, 2002a.

E. Tjong Kim Sang. Memory-based shallow parsing. *Journal of Machine Learning Research*, 2:559–594, 2002b.

E. Tjong Kim Sang and S. Buchholz. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of the 4th Conference on Natural Language Learning, CoNLL-2000*, 2000.

E. Tjong Kim Sang and F. De Meulder. Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. In *Proceedings of the 7th Conference on Natural Language Learning, CoNLL-2003*, 2003.

E. Tjong Kim Sang and H. Déjean. Introduction to the CoNLL-2001 shared task: Clause identification. In *Proceedings of the 5th Conference on Natural Language Learning, CoNLL-2001*, 2001.

E. Tjong Kim Sang and J. Veenstra. Representing text chunks. In *Proceedings of EACL'99*, 1999.

I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the 21st International Conference on Machine Learning, ICML-04*, 2004.

L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11): 1134–1142, 1984.

A. van den Bosch, S. Canisius, W. Daelemans, I. Hendrickx, and E. Tjong Kim Sang. Memory-based semantic role labeling: Optimizing features, algorithm, and output. In *Proceedings of CoNLL-2004*, 2004.

H. van Halteren. Chunking with WPDV Models. In *Proceedings of CoNLL-2000 and LLL-2000*, Lisbon, Portugal, 2000.

V. Vapnik. *Statistical Learning Theory*. John Wiley and Sons, inc., New York, 1998.

V. Vapnik. An overview of statistical learning theory. *IEE Transactions on Neural Networks*, 10(5):988–999, 1999.

J. Weston and C. Watkins. Support vector machines for multi-class pattern recognition. In *Proceedings of the 7th European Symposium on Artifical Neural Networks*, 1999.

K. Williams, C. Dozier, and A. McCulloh. Learning transformation rules for semantic role labeling. In *Proceedings of CoNLL-2004*, 2004.

T. Zhang, F. Damereau, and D. Johnson. Text chunking based on a generalization of winnow. *Journal of Machine Learning Research*, 2:615–637, 2002.

Appendix A

Proof for FR-Perceptron

Acknowledgement: The following proof was mainly conducted by Jorge Castro. We thank him for his contribution in this thesis.

We show in this appendix the complete convergence result for the FR-Perceptron algorithm of Chapter 4. The result we present depends on some restrictive hypotheses and has to be seen only as a first step in the task of achieving a deep and more useful analysis.

The convergence proof we give is based on the proof by Novikoff 1962 for the basic Perceptron algorithm as much as on the proof presented by Collins 2002 for the Perceptron-based sequence tagging algorithm. Assuming *separability* for the Phrase Recognition function and linear separability for each of the *start* and *end* classifiers, the number of errors committed by the learning algorithm can be upper bounded. The proof is presented in two steps, which are described in the two following subsections.

To simplify the notation in the following analysis, we deliberately eliminate the k indexes from the formulae, that is, we assume one single type of phrases. The proof below can be easily extended to the general case with several types of phrases making two straightforward changes. First, the *score* vectors $\mathbf{w}^k$ would be concatenated in a single scoring vector $\mathbf{w} \in I\!R^{d \times |\mathcal{K}|}$. Second, we should assume linear separability for each of the *start-end* classification functions associated to k-phrases.

Scoring Layer without Filtering

In this setting, we assume no filtering components in the learning architecture and we require, for each example (x, y), a search space $\hat{\mathcal{Y}}(x) \subseteq \mathcal{Y}$ containing the solution y. Thus, the Phrase Recognition function can be written as:

$$\mathrm{R}(x) = \arg \max_{y \in \hat{\mathcal{Y}}(x)} \sum_{p \in y} \mathrm{score}(p, x, y)$$

We can rewrite the R function in terms of a single dot product, by considering

a *global* representation function Φ_{p}, which corresponds to the summation of the ϕ_{p} representations of all the phrases in y:

$$\begin{aligned}
\mathrm{R}(x) &= \arg\max_{y\in\hat{\mathcal{Y}}(x)} \sum_{p\in y} \mathrm{score}(p,x,y) \\
&= \arg\max_{y\in\hat{\mathcal{Y}}(x)} \sum_{p\in y} \mathbf{w}\cdot\phi_{\mathrm{p}}(p,x,y) \\
&= \arg\max_{y\in\hat{\mathcal{Y}}(x)} \mathbf{w}\cdot\sum_{p\in y} \phi_{\mathrm{p}}(p,x,y) \\
&= \arg\max_{y\in\hat{\mathcal{Y}}(x)} \mathbf{w}\cdot\Phi_{\mathrm{p}}(x,y)
\end{aligned}$$

Given this notation and by using the assumption that the solutions are in the search spaces, the updating rule for the scoring layer of the FR-Perceptron algorithm can be rewritten globally as follows:

$$\mathbf{w} = \mathbf{w} + \Phi_{\mathrm{p}}(x,y) - \Phi_{\mathrm{p}}(x,\hat{y})$$

In this form, the FR-Perceptron algorithm (without filtering) is the same than the algorithm analyzed in Collins [2002], which proves its convergence. We reproduce here his theorem, in a more general version which allows an initial vector $\mathbf{w}_0$ different from zero. Later we will use this fact.

DEFINITION **1** *A training set $S = \{(x^i, y^i)\}_{i=1}^m$ is separable with margin $\delta > 0$ if there exists some vector $\mathbf{w}^*$ with $||\mathbf{w}^*|| = 1$ such that:*

$$\begin{aligned}
&(\forall i : 1 \le i \le m)(\forall z : z \in \hat{\mathcal{Y}}(x^i) \wedge\ z \neq y^i) \\
&\quad (\mathbf{w}^* \cdot \Phi_{\mathrm{p}}(x^i, y^i) \ge \mathbf{w}^* \cdot \Phi_{\mathrm{p}}(x^i, z) + \delta).
\end{aligned}$$

THEOREM **2** *For any training set $S = \{(x^i, y^i)\}_{i=1}^m$ which is separable with margin $\delta > 0$, the* FR-Perceptron *algorithm with no filtering components makes a number of mistakes bounded by*

$$\frac{R^2}{\delta^2} + \frac{2\|\mathbf{w}_0\|}{\delta},$$

where $\mathbf{w}_0$ refers to the initial value of $\mathbf{w}$ and R is a constant such that $(\forall i : 1 \le i \le m)\,(\forall z : z \in \hat{\mathcal{Y}}(x^i))\ \ (||\Phi_{\mathrm{p}}(x^i, y^i) - \Phi_{\mathrm{p}}(x^i, z)|| \le R)$.

Proof. We essentially follow the main steps in the Collins' proof for a slightly weaker version of Theorem 2 that assumes that the initial vector $\mathbf{w}_0$ is the zero vector.

We enumerate the updates made by the FR-Perceptron algorithm by using an index j and starting from $j = 1$. We have to show that j is upper bounded by $R^2/\delta^2 + 2\|\mathbf{w}_0\|/\delta$.

First, we show a lower bound on the norm of the weight vector at update number j. Let $\mathbf{w}_{j-1}$ be the weight vector just before the j'th mistake is made.

Suppose that this mistake is made at example (x^i, y^i) and let $\hat{y}$ be the output proposed at this example. So, $\hat{y} = \arg\max_{y \in \hat{\mathcal{Y}}(x^i)} \mathbf{w}_{j-1} \cdot \Phi_{\mathrm{p}}(x^i, y)$. It follows from the algorithm update rule that $\mathbf{w}_j = \mathbf{w}_{j-1} + \Phi_{\mathrm{p}}(x^i, y^i) - \Phi_{\mathrm{p}}(x^i, \hat{y})$. We take inner products of both sides with the vector $\mathbf{w}^*$ given by Definition 1:

$$\mathbf{w}^* \cdot \mathbf{w}_j = \mathbf{w}^* \cdot \mathbf{w}_{j-1} + \mathbf{w}^* \cdot \Phi_{\mathrm{p}}(x^i, y^i) - \mathbf{w}^* \cdot \Phi_{\mathrm{p}}(x^i, \hat{y}) \geq \mathbf{w}^* \cdot \mathbf{w}_{j-1} + \delta,$$

where the inequality follows because the training set is separable with margin δ. Hence, reasoning by induction, $\mathbf{w}^* \cdot \mathbf{w}_j \geq \mathbf{w}^* \cdot \mathbf{w}_0 + j\delta$. Now, by applying twice the Schwarz inequality ($|\mathbf{u} \cdot \mathbf{v}| \leq \|\mathbf{u}\|\|\mathbf{v}\|$) and using the fact that $\mathbf{w}^*$ has norm 1 it follows that $\|\mathbf{w}_j\| \geq \mathbf{w}^* \cdot \mathbf{w}_0 + j\delta \geq j\delta - \|\mathbf{w}_0\|$.

An upper bound for $\|\mathbf{w}_j\|^2$ can be also derived:

$$\begin{aligned} \|\mathbf{w}_j\|^2 &= \|\mathbf{w}_{j-1}\|^2 + \|\Phi_{\mathrm{p}}(x^i, y^i) - \Phi_{\mathrm{p}}(x^i, \hat{y})\|^2 \\ &\quad +2\mathbf{w}_{j-1} \cdot (\Phi_{\mathrm{p}}(x^i, y^i) - \Phi_{\mathrm{p}}(x^i, \hat{y})) \\ &\leq \|\mathbf{w}_{j-1}\|^2 + R^2, \end{aligned}$$

where the last inequality follows because, by assumption, $\|\Phi_{\mathrm{p}}(x^i, y^i) - \Phi_{\mathrm{p}}(x^i, \hat{y})\|^2 \leq R^2$, and $\mathbf{w}_{j-1} \cdot (\Phi_{\mathrm{p}}(x^i, y^i) - \Phi_{\mathrm{p}}(x^i, \hat{y})) \leq 0$ because $y^i \in \hat{\mathcal{Y}}(x^i)$ (by the hypothesis in this section) and $\hat{y}$ is the highest scoring candidate for x^i under $\mathbf{w}_{j-1}$ in $\hat{\mathcal{Y}}(x^i)$. Reasoning again by induction we have $\|\mathbf{w}_j\|^2 \leq \|\mathbf{w}_0\|^2 + jR^2$.

Finally, note that if $j \geq \|\mathbf{w}_0\|/\delta$ (otherwise, the bound we want to show is straightforward) then $j\delta - \|\mathbf{w}_0\|$ is non-negative and we can combine the bounds on the norm of $\mathbf{w}_j$ we have just obtained:

$$(j\delta - \|\mathbf{w}_0\|)^2 \leq \|\mathbf{w}_j\|^2 \leq \|\mathbf{w}_0\|^2 + jR^2.$$

From this expression it is immediate to see that $j \leq R^2/\delta^2 + 2\|\mathbf{w}_0\|/\delta$. □

Filtering and Scoring

When including the filtering component, the recognition model is written as:

$$\mathrm{R}(x) = \arg \max_{y \in \mathcal{Y}_{SE}(x)} \mathbf{w} \cdot \Phi_{\mathrm{p}}(x, y)\,,$$

where $\mathcal{Y}_{SE}(x) = \{y \in \mathcal{Y}\,|(\forall (s,e) \in y)(\mathrm{start}(x_s, x) > 0 \wedge \mathrm{end}(x_e, x) > 0)\}$.

Definition **2** *(Start-End separability) A training set* $\{(x^i, y^i)\}_{i=1}^m$ *is Start-End Separable with margin* $\gamma > 0$ *if there exist vectors* $\mathbf{w}_{\mathrm{S}}^*$ *and* $\mathbf{w}_{\mathrm{E}}^*$ *with* $||\mathbf{w}_{\mathrm{S}}^*|| = 1$, $||\mathbf{w}_{\mathrm{E}}^*|| = 1$ *such that:*

$$\begin{aligned} &(\forall i : 1 \leq i \leq m)(\forall j : 1 \leq j \leq n_i) \\ &\quad (\mathrm{goldS}(x_j^i)(\mathbf{w}_{\mathrm{S}}^* \cdot \phi_{\mathrm{w}}(x_j^i, x)) \geq \gamma \wedge \mathrm{goldE}(x_j^i)(\mathbf{w}_{\mathrm{E}}^* \cdot \phi_{\mathrm{w}}(x_j^i, x)) \geq \gamma). \end{aligned}$$

In order to show the convergence of the FR-Perceptron algorithm we assume the separability of the sample in the sense of Definition 1, considering the whole solution space, $\hat{\mathcal{Y}} = \mathcal{Y}$. In addition, we assume also the Start-End separability of Definition 2. We will denote by SE-LF a learning feedback stage of FR-Perceptron where updates affect to the *start-end* vectors $\{\mathbf{w}_S, \mathbf{w}_E\}$ and maybe the *score* vector $\mathbf{w}$, and by SCORE-LF those stages where updates only changes the *score* vector. Our goal is to bound the number of SE-LF and SCORE-LF stages, and, therefore, the number of training errors. We start by bounding the number of SE-LF stages that the algorithm FR-Perceptron makes when running on a Start-End separable sample.

LEMMA **1** *Let* $R_{SE} = \max_{1\leq i\leq m,\ 1\leq j\leq n_i} \|\phi_w(x_j^i, x^i)\|$ *of a training set* $\{(x^i, y^i)\}_{i=1}^m$ *satisfying Definition 2 with margin* γ*. Then the number of* SE-LF *stages of* FR-Perceptron *is at most*

$$\frac{2R_{SE}^2}{\gamma^2}.$$

Proof. It follows immediately from Novikoff's proof Novikoff [1962] for the standard Perceptron algorithm. □

Now, to conclude the convergence of the algorithm FR-Perceptron we must demonstrate that after a SE-LF stage there is room only for a finite number of consecutive SCORE-LF stages before a point where, no more changes of any kind are necessary or a new SE-LF stage is required. We consider two cases. First, we assume that in the last SE-LF stage the updated values of $\mathbf{w}_S$ and $\mathbf{w}_E$ are so that for some $1 \leq i \leq m$ the solution y^i does not belong to $\mathcal{Y}_{SE}(x^i)$. In this case, it is clear that if $\{\mathbf{w}_S, \mathbf{w}_E\}$ have not been yet modified when FR-Perceptron visits the example (x^i, y^i), a change on $\{\mathbf{w}_S, \mathbf{w}_E\}$ of type A (see Figure 4.2) will be made when the algorithm visits such example. So, we have the following lemma.

LEMMA **2** *Assume that for some* i, $1 \leq i \leq m$*, the search space* $\mathcal{Y}_{SE}(x^i)$ *does not include the solution* y^i*. At this point at most* $m-1$ *consecutive* SCORE-LF *stages are possible.*

Second, we consider the case where the last SE-LF stage has updated values for $\{\mathbf{w}_S, \mathbf{w}_E\}$ such that for all i, $1 \leq i \leq m$ it is $y^i \in \mathcal{Y}_{SE}(x^i)$. Let $\mathbf{w}_0$ be the value of $\mathbf{w}$ when this SE-LF stage is completed. Here we are in the hypothesis of Theorem 2 and therefore, after at most $R^2/\delta^2 + 2\|\mathbf{w}_0\|/\delta$ consecutive SCORE-LF stages the algorithm reaches to a point where no more learning feedbacks are necessary or a new SE-LF stage appears (generated by a type-B change, see Figure 4.2).

LEMMA **3** *Assume that for all* i, $1 \leq i \leq m$*, the search space* $\mathcal{Y}_{SE}(x^i)$ *contains the solution* y^i*. At this point at most* $R^2/\delta^2 + 2\|\mathbf{w}_0\|/\delta$ *consecutive* SCORE-LF *stages are possible.*

Summarizing, from Lemmas 1, 2 and 3 we conclude the next theorem that shows the convergence of FR-Perceptron.

THEOREM **3** *For any training set $S = \{(x^i, y^i)\}_{i=1}^m$ separable with margin $\delta > 0$ and Start-End separable with margin $\gamma > 0$, it holds:*

1. *The number of learning feedbacks that affect to the start-end vectors* (SE-LF *stages*) *is bounded by* $2R_{SE}^2/\gamma^2$.

2. *After a learning feedback stage l that has affected the start-end vectors* (SE-LF *stage*) *there are at most*

$$\max\left(m-1, \frac{R^2}{\delta^2} + \frac{2\|\mathbf{w}_0^l\|}{\delta}\right)$$

consecutive learning feedbacks that only affects to the score *vectors* (SCORE-LF *stages*). *Here,* $\mathbf{w}_0^l$ *is the vector* $\mathbf{w}$ *when the updates corresponding to stage l have been just made.*

3. *As a consequence of the previous two points, the* FR-Perceptron *algorithm makes a finite number of errors on the training set. When no more errors occur the total number of errors committed by the algorithm is bounded by*

$$\frac{2R_{SE}^2}{\gamma^2}\left(1 + \max\left(m-1, \frac{R^2}{\delta^2} + \frac{2MAX}{\delta}\right)\right),$$

where MAX is the maximum of the values $\|\mathbf{w}_0^l\|$ at any SE-LF *stage.*

Appendix B

Author's Publications

Following, we list the author's publications that are related to the topics of this thesis. Most of the papers are directly related to the work presented in this thesis. Others are related to machine learning and natural language processing. All papers can be downloaded from the following URL:

http://www.lsi.upc.edu/∼carreras/publications.html

Journals

- Xavier Carreras, Lluís Màrquez and Jorge Castro. "Filtering-Ranking Perceptron Learning for Partial Parsing". *Machine Learning, Special Issue on Learning in Speech and Language Technologies*, 59:1-31 , 2005.

- Enrique Romero, Lluís Màrquez and Xavier Carreras. "Margin maximization with feed-forward neural networks: a comparative study with SVM and AdaBoost". *Neurocomputing*, 57(C):313-344, 2004.

- Montse Arévalo, Xavier Carreras, Lluís Màrquez, Toni Martí, Lluís Padró and Maria José Simon. "A Proposal for Wide-Coverage Spanish Named Entity Recognition". *Procesamiento del Lenguaje Natural, Revista de la SEPLN*. Vol. 28. May 2002.

Conference Papers

- Xavier Carreras and Lluís Màrquez. "Introduction to the CoNLL-2005 Shared Task: Semantic Role Labeling". In *Proceedings of the 9th Conference on Computational Natural Language Learning (CoNLL-2005)*. Ann Arbor, MI USA. June 2005.

- Enrique Romero, Xavier Carreras and Lluís Màrquez. "Exploiting Diversity of Margin-based Classifiers". In *Proceedings of the 2004 International Joint Conference on Neural Networks (IJCNN)*. Budapest, Hungary, July 2004.

- Xavier Carreras and Lluís Màrquez. "Introduction to the CoNLL-2004 Shared Task: Semantic Role Labeling". In *Proceedings of the 8th Conference on Computational Natural Language Learning (CoNLL-2004)*. Boston, MA USA. May 2004.

- Xavier Carreras, Lluís Màrquez and Grzegorz Chrupała. "Hierarchical Recognition of Propositional Arguments with Perceptrons". In *Proceedings of the 8th Conference on Computational Natural Language Learning (CoNLL-2004)*. Shared Task Contribution. Boston, MA USA. May 2004.

- Xavier Carreras and Lluís Màrquez. "Online Learning via Global Feedback for Phrase Recognition". In *Advances in Neural Information Processing Systems 16 (NIPS-2003)*. Vancouver, Canada. December 2003.

- Xavier Carreras and Lluís Màrquez. "Phrase Recognition by Filtering and Ranking with Perceptrons". In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP-2003)*. Borovets, Bulgaria. September 2003.

- Xavier Carreras, Lluís Màrquez and Lluís Padró. "A Simple Named Entity Extractor Using AdaBoost". In *Proceedings of the Workshop on Computational Natural Language Learning (CoNLL-2003)*. Shared Task Contribution. Edmonton, Canada. June 2003.

- Xavier Carreras, Lluís Màrquez and Lluís Padró. "Learning a Perceptron-Based Named Entity Chunker via Online Recognition Feedback". In *Proceedings of the 7th Conference on Computational Natural Language Learning (CoNLL-2003)*. Shared Task Contribution. Edmonton, Canada. June 2003.

- Lluís Màrquez, Adrià de Gispert, Xavier Carreras and Lluís Padró. "Low-cost Named Entity Classification for Catalan: Exploiting Multilingual Resources and Unlabeled Data". In *Proceedings of the 1st ACL Workshop on Multilingual and Mixed-language Named Entity Recognition: Combining Statistical and Symbolic Models (MulNER'03)*. Sapporo, Japan. July 2003.

- Xavier Carreras, Lluís Màrquez and Lluís Padró. "Named Entity Recognition For Catalan Using Spanish Resources". In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL'03)*. Budapest, Hungary. April 2003.

- Xavier Carreras, Lluís Màrquez and Lluís Padró. "Wide-Coverage Spanish Named Entity Extraction" In *Proceedings of the VIII Conferencia Iberoamericana de Inteligencia Artificial (IBERAMIA'02)*. Sevilla, Spain. November 2002.

- Xavier Carreras, Lluís Màrquez and Lluís Padró. "Named Entity Extraction using Adaboost". CoNLL Shared Task Contribution. In *Proceedings of the 6th Workshop on Computational Natural Language Learning (CoNLL-2002)*. Taipei, Taiwan. September 2002.

- Xavier Carreras, Lluís Màrquez, Vasin Punyakanok and Dan Roth. "Learning and Inference for Clause Identification" In *Proceedings of the 13th European Conference on Machine Learning (ECML'02)*. Helsinki, Finland. August 2002.

- Xavier Carreras and Lluís Màrquez. "Boosting Trees for Clause Splitting" CoNLL Shared Task Contribution. In *Proceedings of the 6th Workshop on Computational Natural Language Learning (CoNLL-2001)*. Toulouse, France. July 2001.

- Xavier Carreras and Lluís Màrquez. "Boosting Trees for Anti-Spam Email Filtering" In *Proceedings of the RANLP-2001: Recent Advances in NLP*. September, 2001.