Part–of–speech Tagging: A Machine Learning Approach based on Decision Trees

Memòria presentada al Departament de Llenguatges i Sistemes Informàtics de la Universitat Politècnica de Catalunya per a optar al grau de Doctor en Informàtica

Lluís Màrquez i Villodre

sota la direcció del doctor
Horacio Rodríguez Hontoria

Barcelona, Maig de 1999

Abstract

The study and application of general Machine Learning (ML) algorithms to the classical ambiguity problems in the area of Natural Language Processing (NLP) is a currently very active area of research. This trend is sometimes called Natural Language Learning. Within this framework, the present work explores the application of a concrete machine–learning technique, namely decision–tree induction, to a very basic NLP problem, namely part–of–speech disambiguation (POS tagging). Its main contributions fall in the NLP field, while topics appearing are addressed from the artificial intelligence perspective, rather from a linguistic point of view.

A relevant property of the system we propose is the clear separation between the *acquisition* of the language model and its *application* within a concrete disambiguation algorithm, with the aim of constructing two components which are as independent as possible. Such an approach has many advantages. For instance, the language models obtained can be easily adapted into previously existing tagging formalisms; the two modules can be improved and extended separately; etc.

As a first step, we have experimentally proven that decision trees (DT) provide a flexible (by allowing a rich feature representation), efficient and compact way for acquiring, representing and accessing the information about POS ambiguities. In addition to that, DTs provide proper estimations of conditional probabilities for tags and words in their particular contexts. Additional machine learning techniques, based on the combination of classifiers, have been applied to address some particular weaknesses of our tree–based approach, and to further improve the accuracy in the most difficult cases.

As a second step, the acquired models have been used to construct simple, accurate and effective taggers, based on different paradigms. In particular, we present three different taggers that include the tree–based models: RTT, STT, and RELAX, which have shown different properties regarding speed, flexibility, accuracy, etc. The idea is that the particular user needs and environment will define which is the most appropriate tagger in each situation. Although we have observed slight differences, the accuracy results for the three taggers, tested on the WSJ test bench corpus, are uniformly very high, and, if not better, they are at least as good as those of a number of current taggers based on automatic acquisition (a qualitative comparison with the most relevant current work is also reported.

Additionally, our approach has been adapted to annotate a general Spanish corpus, with the particular limitation of learning from small training sets. A new technique, based on tagger combination and bootstrapping, has been proposed to address this problem and to improve accuracy. Experimental results showed that very high accuracy is possible for Spanish tagging, with a relatively low manual effort. Additionally, the success in this real application has confirmed the validity

of our approach, and the validity of the previously presented portability argument in favour of automatically acquired taggers.

Lluís Màrquez i Villodre

TALP Research Center
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Jordi Girona Salgado, 1–3. E08034 Barcelona. Catalonia.
E–mail: lluism@lsi.upc.es
URL: http://www.lsi.upc.es/~lluism

Agraïments

Voldria fer arribar el meu sincer agraïment a un bon nombre de persones que m'han ajudat directa o indirectament a desenvolupar la meva tasca com a recercaire durant els darrers anys i, finalment, a donar forma a aquest treball.

Ara ve una llista, oi?

En primer lloc i molt especialment al meu director, l'Horaci Rodríguez, que apart de ser el savi més modest que conec, és, sens dubte, el director de tesi que jo recomanaria al meu millor amic.

Voldria donar les gràcies també especialment a en Lluís Padró, amb qui he realitzat la major part de recerca d'aquest treball. Val a dir que ell i en German Rigau han estat uns companys d'una vàlua humana i científica incomparables i de fet han exercit de "segons directors" de tesi (si és que aquesta figura existeix).

També, a tota la resta de companys del Grup de Recerca en Processament del Llenguatge Natural del departament de LSI i de la Universitat de Barcelona, amb qui hem creat (i seguim creant) un bon equip de treball: Alicia Ageno, Jordi Atserias, Núria Castell, Neus Català, Irene Castellón, Salvador Climent, Gerard Escudero, Toni Martí, Mariona Taulé, Jordi Turmo,... i a tot els que em deixo. També a en Josep Carmona i a en Josep Montolio (els Joseps) que han estat ajudant-me en els darrers experiments.

En general, a tots els companys del departament i especialment a en Rafel Cases, Ton Sales i Martí Vergés, de qui he après una pila de coses no necessariament relacionades amb el tema de la tesi.

També voldria destacar a Itziar Aduriz i a tot el grup de processament del lleguatge (IXA taldea) de la Euskal Herriko Unibersitatea pel boníssim acolliment que em van dispensar durant la meva estada a Donostia l'any 1998 (bereziki goikoei!)

També cal mencionar el laboratori de cálcul i la secretaria del nostre departament que funcionen de manera modèlica i estan formats per persones magnífiques.

Finalment, voldria fer constar que durant el periode final de realització de la tesi he gaudit d'una descàrrega docent atorgada pel departament de LSI, que m'ha permès, sens dubte, acabar aquest treball una miqueta abans.

La revisió de l'anglès en les parts més atepeïdes de la tesi l'ha fet també un bon amic, David Owen. Tota la resta és exclusivament culpa (i merit) d'un servidor.

Acknowledgements

I would like to thank two anonymous referees for helpful and encouraging comments on a previous extended abstract of this document.

The research reported in this dissertation has been developed inside the framework of several research projects, funded by the following institutions: The Spanish Research Department (CICYT's ITEM project, ref: TIC96–1243–C03–02); The EU Commission (ACQUILEX II, ESPRIT-BRA 7315 and EuroWordNet, LE4003) and The Catalan Research Department (CIRIT's consolidated research group 1997SGR 00051 and CREL project).

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

In the eighties but especially throughout the nineties, an important resurgence of empirical and statistical methods, applied to the automatic processing of natural language, has been observed. This popularity has its origin in four main factors:

- The growing availability of big machine–readable corpora, from different sources, levels of annotation, languages, etc.
- The improvement in performance of current software and hardware architectures that allow the processing of huge amounts of information, by highly time–consuming algorithms (as statistical models usually demand), at an admissible cost of time and money.
- The initial success obtained with the statistical processing of low–level language problems, such as those related with speech recognition and syntactic tagging.
- The appearance and development of a large amount of text–based natural language applications with specific requirements, for which conventional methods based on linguistic knowledge seemed not to be appropriate.

The choice of automatically processing such massive quantities of free text (commonly referred to *data–intensive approach* or *corpus–based approach*) has contributed to developing a number of methods and techniques with an application to a great variety of natural language acquisition and understanding problems, including: automatic extraction of lexical knowledge, lexical and structural disambiguation (part–of–speech tagging, word sense disambiguation and prepositional phrase attachment disambiguation), grammatical inference and robust parsing, information extraction and retrieval, machine translation, etc. Additionally, corpora have provided linguists with benchmarks for the empirical evaluation of theoretical studies and models of language. The reader may find good and plain introductions to the corpus–based NLP in [**LF92, CM93, YB97**].

Most of the initial corpus–based language acquisition methods applied by the NLP community researchers were borrowed from statistics and information theory. As a consequence of this collaboration, a significant progress was made in the development and adaptation of well–known statistics based techniques to the particular problems of automatic natural language modelling and processing. This progress was especially noticeable in the low–level tasks of speech processing and lexical knowledge extraction and disambiguation, but a considerable effort was also devoted to grammatical inference, robust parsing and other semantic and discourse level NLP tasks.

Several articles and introductory books, surveying statistical approaches in Computational Linguistics, have appeared in recent years. We especially recommend [**Cha93, Cha97, KS97, MS99, JM99**].

Starting in the early 90s, but particularly in recent years, the application of machine learning (ML) based techniques to language learning and acquisition problems has been the focus of increasing attention in the NLP community. The core of problems addressed by machine learning techniques are those of natural language disambiguation (appearing at all levels of the language understanding process). They are particularly appropriate because they can be easily recast as *classification problems*, a generic type of problem with a long tradition in the artificial intelligence (AI) area, and especially addressed by the ML community.

The connection and collaboration between NLP and machine learning communities has become very productive. Common interests are clearly noticeable in the international conferences and journals of both areas, which have experimented a proliferation of special issues on *natural language learning*, *applications of ML techniques to natural language processing*, etc.

This recent approach permits the use and adaptation of general purpose machine learning algorithms for classification –which are well known and widely studied methods–, with the aim of providing general frameworks in which many disambiguation problems could be addressed simultaneously by homogeneous techniques **[Bri95a, Car96a, DWB97, Rot98]**.

Already applied ML based methods include several *traditional* symbolic inductive learning paradigms: instance–based learning, decision trees, threshold linear separators, inductive logic, unsupervised clustering, etc.; and also a number of subsymbolic and connectionist approaches, such as neural networks and genetic algorithms. Finally, some new specific learning algorithms have also been proposed in recent years. This is the case of *Transformation–based Learning* in **[Bri95a]**.

For an extensive compilation of relevant articles about the current approaches to natural language learning, one may consult the book **[WRS96]**.

Within this presented framework, the aim of this work is to explore the possibility of applying well known ML techniques, namely the automatic induction of decision trees, to improve results in a very basic disambiguation task, namely part–of–speech (POS) tagging.

This is mainly an empirical work, that is, it proposes some new solutions or variants to a practical problem and it tests the appropriateness of such approaches by a thorough set of experiments. All the proposals have been fully implemented and the experimentation has been conducted by defining a clear and rigorous framework, specifying the methodology and the reference domain in which prototypes are tested. Direct comparisons with other works and approaches have been included whenever they were feasible. Conclusions of experiments and comparisons are always provided with a certain degree of confidence, on the basis of performing statistical tests.

It must be said that the main contributions of this work fall in the NLP field and that topics appearing are addressed from the artificial intelligence perspective, rather from a linguistic point of view.

1. Setting

1.1. The Ambiguity Problem. Natural language is ambiguous in nature. Ambiguity appears at many levels of the usual language processing chain, and it represents many of the most difficult problems of language understanding. Some classical examples of ambiguity are: Word selection in speech recognition, part of

speech ambiguity (e.g. past vs. participle in regular verbs), semantic ambiguity in polysemic words, structural ambiguity in parsing (e.g. PP–attachment), accent restoration, reference ambiguity in anaphora resolution, word choice selection in machine translation, context–sensitive spelling correction, etc.

Such *ambiguity resolution* problems can be generically characterized as follows: *At a certain moment, the* NLP *system reaches a segment of information for which it has multiple interpretations, and it must decide which interpretation is appropriate in the current context. In order to resolve this difficulty, it is necessary to disambiguate two or more semantically, syntactically or structurally distinct forms based on the properties of the surrounding context* [**Car96a**].

Consider, for instance, the following simple sentence, which was picked from the Wall Street Journal (WSJ) corpus:

(1) *He was shot in the hand as he chased the robbers in the back street.*

First, the sentence contains a number of POS ambiguities that should be resolved before the sentence can be understood. For instance, "shot" and "hand" can be a noun or a verb; "chased" can be an adjective or a verb; "back" can be a noun, an adjective, an adverb, and a verb; finally, "in" and "as" can be a preposition or an adverb. Even if we know its part of speech, the intended meaning of the word in a particular context often requires disambiguation. In the present example, the word "hand" is highly polysemous. Among other interpretations, it could refer to a part of the body in the *anatomical* sense, but also to a part of a clock in a *mechanical* sense. In addition to these cases of lexical ambiguity we also find an example of structural ambiguity: The prepositional phrase "in the back street" could modify the noun "robbers" or the verb "chased". Both readings are syntactically legal and the NLP system must access some kind of semantic information to make the correct attachment decision.

1.1.1. *General Approaches.* The current approaches to disambiguation problems, such as POS tagging, WSD, PP–attachment disambiguation, etc., can be classified in two broad families: linguistic- and statistical–based.

On the one hand, the classical and most straightforward is the linguistic approach, which typically uses rule–based models manually written by linguists. The linguistic models are developed by introspection (sometimes with the aid of reference corpora). This makes it particularly costly to obtain a good language model. Transporting the model to other languages means starting over again. They usually do not consider frequency information and thus have a limited robustness and coverage. Their advantages are that the model is written from a linguistic point of view and explicitly describes linguistic phenomena, and that the model may contain many and complex kinds of information. Both things allow the construction of extremely accurate systems.

On the other hand, statistical approaches are based on collecting statistics from existing corpora, either tagged (supervised training) or untagged (unsupervised training). This makes the model development much shorter and the transportation to other languages much easier, provided there are corpora in the desired language. They take into account frequency information, which gives them great robustness and coverage. The statistical approaches can be divided in two categories, depending on the complexity of the statistical model they acquire, and on the type of algorithm they use. First, in the *simple–model class*, the considered knowledge consist of a set of co–occurence frequencies for some predetermined features, which

are straightforwardly acquired from a corpus. The combination of this knowledge is usually performed by using the Bayes' theorem with the corresponding independence assumptions. Second, machine–learning algorithms are used to acquire *high–level language knowledge* from a training corpus. Already applied paradigms range from instance based learning and induction of decision trees, to neural networks and genetic algorithms. The knowledge acquired may take the form of rules, decision trees, frames, etc. but it will be more complex than a simple set of frequency countings. In this case the model is explicit, since usual machine–learning produces symbolic knowledge, but it does not necessarily have a direct linguistic interpretation.

1.2. The Part–of–speech Tagging Problem. POS tagging is a very basic and well known NLP problem which consists of assigning to each word of an input text the proper morphosyntactic tag in its context of appearance. In most cases ambiguous words can be completely disambiguated by taking into account an adequate context. For instance, recalling the previous example sentence 1, the word "hand" would be disambiguated as a noun because it is preceded by the determiner "the". Although in this case the word is disambiguated simply by looking at the preceding tag, it must be taken into account that the preceding word could be ambiguous, or that the necessary context could be much more complex than merely the preceding word. Furthermore, there are even cases in which the ambiguity is non–resolvable by using only morphosyntactic features of the context, and require semantic and/or pragmatic knowledge.

1.2.1. *Utility.* The utility of such disambiguation tasks is also well known. Part–of–speech tagging is of interest for a number of applications, including: Speech recognition and generation [**NMKS90, HA97**], access to text data bases [**Kup93**], partial parsing [**Abn91, KVHA95, Wau95, Abn97, VP97, Pad98**] and general parsing [**DeM90, CCA+94**], grammatical inference [**Mag96, STV96**], machine translation, information extraction, information retrieval, and lexicography. As a curiosity, we mention that POS tagging has also been proposed in a particular technique of stylometry as a way of investigating the authenticity of *Rhesus* (an ancient Greek text) by the Attic dramatist EURIPIDES. The method tries to characterize the style of the writer by observing the typical distributions of categories in his texts [**Lud97**].

Additionally, since tagging can be seen as a prototypical problem in lexical ambiguity, advances in part of speech tagging could readily translate to progress in other areas of lexical, and perhaps structural, ambiguity.

Most language understanding systems are formed by set of pipelined modules, each of them accounting for a specific level of analysis. That implies that results of one module may seriously influence the performance of the following modules, and that the errors may exponentially propagate along the pipeline. As POS tagging is one of the very basic tasks to be performed, it is crucial not to introduce many errors. Several authors have pointed out and quantified the influence of tagging in further stages of natural language processing. See, for instance [**Kro97**], [**WS97**], and [**CCA+94**] discussing the influence of POS tagging as a preprocess for information retrieval, word sense disambiguation, and parsing.

1.2.2. *Approaches.* A general representation of the POS tagging process is depicted in figure 1. We distinguish two main components. On the one hand, the system uses some kind of knowledge about the language to be disambiguated. This

knowledge may come from several sources and may have different representations. We refer to it as the *language model*[1]. On the other hand, there is the proper disambiguation algorithm which decides the best tag assignment according to the language model. These two components are certainly related and we usually find them embedded into a single tagger description. However, there are cases in which algorithms and models are relatively independent and, therefore, easy to transport.

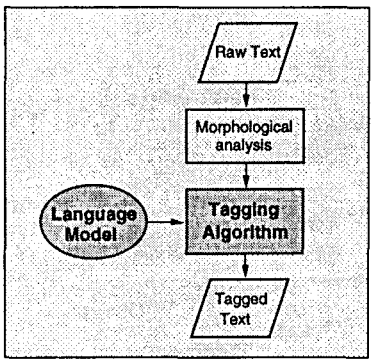

FIGURE 1. POS tagging schemata

The input to the disambiguation algorithm consists of a list of lexical units with an associated list of possible morphosyntactic tags. This information is usually obtained by performing a previous morphological analysis, or by simply collecting a word–form lexicon extracted from pretagged corpora, in the case of largely uninflected languages (such as English). The output consists of the same list of lexical units with the ambiguity reduced, ideally, to a unique tag for each unit.

Existing approaches to POS tagging can be classified in many ways depending on the focus of attention. First, according to the way in which the knowledge is acquired, we could talk about *manual*, *automatic* or *semi–automatic* taggers. Second, there are several ways of representing the language model. We find rule–based systems, Markov models, neural networks, collections of examples, decision trees, etc. Third, there are taggers that output a completely disambiguated sequence of pairs word–tag, while others perform a kind of iterative algorithm to progressively reduce the input ambiguity, thereby allow residual output ambiguity. This difference becomes important when comparing the accuracy of different taggers, because a study between *recall* and *precision* will be required for the latter. Finally, the most wide–spread categorization system takes into account the nature of the knowledge used for the disambiguation algorithm. In this way we talk about: linguistic, statistical and machine–learning families.

The *real* categorization should be, in fact, a combination of all these distinct categorization systems, however, for the sake of simplicity, in this work we will use only the last as the basic classification criterion. The other distinctions will be mentioned only whenever they are relevant.

[1] Although sometimes it has a clear parallel, this term should not be confused with the *language model* term used in the field of information theory. In that domain, the term is used to refer to a probability distribution on the set of all word sequences for a fixed vocabulary.

1.2.3. *Performance.* Accuracy reported by first rule–based linguistic English taggers was slightly below 80% of correct part–of–speech assignments. This is certainly a low figure, especially so when a broad coverage word–form lexicon with associated lexical probabilities would provide an accuracy of around 90% by simply choosing the most frequent tag for each input word. The accuracy reported by most current statistic and machine learning English taggers is consistently over 95%, achieving 96–97% in some cases, while linguistic Constraint Grammars report the best results. For instance, in [**SV97**], a CG for English tagging is presented which achieves a recall of 99.5% with a very high precision around 97%. These accuracy values are usually computed on a test corpus which has not been used in the training phase. Some corpora commonly used as test benches are the Brown Corpus, the Wall Street Journal (WSJ) corpus and the British National Corpus (BNC).

Very few direct comparisons have been performed in the literature. The differences in the reference test corpus, the size and granularity of the tagset, the size of the training set, etc., can significantly influence the obtained results and, so, invalidate the conclusions of some comparisons.

Regarding to the tagging efficiency of such systems, very different experimental figures have been reported in the literature, which range from a few dozen words per second to the impressive result of 10,800 words per second [**RS95**]. Such high speed is achieved by means of recasting rule–based tagging as a composition of finite state transducers. Classic statistical taggers run about ten times slower (~1000 wps).

1.2.4. *Improvements.* Finally, we would like to mention some of the current open lines of research to further improve POS tagging. Providing more flexible environments in which to incorporate many sources of knowledge of different levels, is one of the current active areas. A second line refers to the study of the *real* portability of the automatic tagging paradigm, especially when they are moved to different corpora, languages, and NLP systems with different goals. With respect to the accuracy of current automatic taggers, we think that there is still room for improvement towards the accuracy reported by current linguistic systems.

2. A Particular Approach to Tagging

Our particular approach to POS tagging belongs to the machine–learning family, and it is based on the fact that POS tagging can easily be interpreted as a classification problem, in which the finite set of classes is identified with the set of possible tags, and the training examples are particular occurrences of words with their respective contexts.

A relevant property of our system is the clear separation between the *acquisition* of the language model and its *application* within a concrete disambiguation algorithm, with the aim of constructing two components which are as independent as possible. Such an approach has many advantages. For instance, the language models obtained can be easily adapted into previously existing tagging formalisms; the two modules can be improved and extended separately; etc.

We used induction of decision trees to acquire and represent the language models. This is a wide–spread formalism in the ML field that has recently been applied in basic NLP tasks such as speech recognition, tagging, parsing, sense disambiguation, etc., with notable success. To avoid the harmful effects of data fragmentation and to constrain the problem into manageable dimensions, we consider a different

problem (and so a different tree) for each ambiguity class, i.e. the set of words than can be a noun or an adjective form the *noun–adjective* ambiguity class, which will be addressed independently from other ambiguity classes, as *noun–verb*, *preposition–adverb*, etc. The set of all these trees is what we refer to as the language model.

The algorithm we used for constructing the statistical decision trees is a non–incremental supervised learning–from–examples algorithm of the TDIDT (Top–Down Induction of Decision Trees) family. Such an algorithm is very similar to those of CART [**BFOS84**] and C4.5 [**Qui93**] systems, but it includes some variations to better adapt to our particular domain.

With respect to the application phase, we have constructed two taggers, based on different principles, that overcome the application of the trees as direct classifiers. For this reason, we benefited from the statistical information provided by the trees. Additionally, we have investigated the integration of the tree–based model with other kinds of information, within the framework of a third tagger.

3. Contributions

Although this work is closely related to the machine learning field, from which we borrowed existing techniques and tools, its major contributions have to be placed in the natural language processing area of artificial intelligence.

We summarize below the list of topics that, according to our opinion, constitute the main contributions of this work. The first three points refer to real contributions from a scientific perspective. However, we want to mention two additional relevant aspects. The first refers to the effort that we carried out to provide a broad and accurate survey of recent applications of machine learning techniques to natural language learning problems. The second point aims at highlighting the results of this work in terms of generated tools and resources, which may be of interest for the NLP community, especially for those groups working with the Spanish language.

3.1. On the Application of Decision Trees. We have successfully applied Statistical Decision Trees to a very basic NLP disambiguation problem, namely POS tagging, obtaining very high levels of accuracy. This contribution is threefold:

- We have proposed decision trees as a method to automatically acquire the statistical information that models part–of–speech ambiguities. We have tested the acquired trees and we have proven that they are a quite compact, accurate and efficient way of extracting and representing this information. Decision trees are also a flexible way of handling several features, for which the set of attributes describing each example can easily be enriched.
- Using the statistical information collected with the decision trees, we have developed and implemented two practical taggers, following different approaches. We have shown that the obtained tagging accuracies are as good (and better in some cases) as the results of a number of the non–linguistically motivated state–of–the–art taggers.
- We have explored the possibility of combining different kinds of information to improve tagging accuracy. On the one hand, by introducing n–gram statistical information in our tagger. On the other hand, by adapting the trees into a constraint formalism to feed a flexible–model tagger based on relaxation labelling, once more including n–grams, but also including linguistic

information. We have proved that in both ways the resulting accuracy is significantly higher.

3.2. On the Combination of Classifiers. Combination of classifiers is a very productive research direction within the ML community. We have investigated whether the combination of different taggers can improve the results of a single tagger. In our case the contribution is twofold:

- We have applied several algorithms, from the ML field, to grow ensembles of weak classifiers from the same training data (namely bagging, boosting, etc.), in order to improve tagging accuracy in the most relevant ambiguity cases. Additionally, we propose two new methods in the same direction, which have been carefully evaluated.
- We also have presented a bootstrapping method to develop an annotated corpus, which is especially useful for languages with few available resources. The method consists of taking advantage of the collaboration of several taggers. The cases in which a majority of taggers agree present a higher accuracy, and are used to retrain the taggers in an iterative approach. We have applied this method to improve the tagging accuracy in the annotation of the LEXESP corpus.

3.3. On the Evaluation and Comparison of Taggers. In chapter 7, we point out several difficulties arising when evaluating and comparing tagger performances against a reference test corpus, and we make some criticism about common practices followed by the NLP researchers in this issue. One of the drawbacks of such evaluation practices is that taggers are tested and compared against noisy corpora. We have proven that in this case, no reliable conclusions can be extracted, and that more rigorous testing experimentation designing is needed.

3.4. On the Survey. Although this is not the most important contribution, we want to mention the work carried out in the compilation and organization of an understandable, broad-coverage list of references linking the fields of machine learning and natural language processing. We think that it provides valuable information that can contribute to ease the approach of any researcher to the use of machine learning techniques in processing natural language.

Those interested can consult the following URL to find a comprehensive list of the references appearing in this thesis:

`http://www.lsi.upc.es/~lluism/BibTexDB.html`

The available format is BibTex and several links are provided to help obtain postscript versions of the original papers.

3.5. From a Practical Perspective. The present work has been developed inside the framework of European and Spanish research projects. As a result of this, some practical tools and resources have been developed. In particular, there are available taggers for English and Spanish, and a general framework for morphosyntactic analysis, tagging and parsing of unrestricted Spanish texts.

The reader can access these tools (and others available) through the URL of the Natural Language Research Group at the Software Department of the Catalan Polytechnical University: `http://www.lsi.upc.es/~acquilex/nlrg.html`

The LEXESP annotated Spanish Corpus will be also available for research purposes. The contact information is included in appendix E.

4. Overview of the Thesis

This section describes the contents and the organization of the rest of the thesis.

The body of the work is covered by chapters 3, 4 and 5. They describe the use of decision trees for tagging English and Spanish languages. A complete empirical evaluation is presented and, additionally, the annotation of a Spanish corpus of over 5 million words is described. These chapters are complemented with a survey of the state of the art (chapter 2), a study about the ability of ensembles of classifiers to improve tagging results (chapter 6), a review of evaluation and comparison of taggers, including a comparison with closely related approaches (chapter 7), and a final chapter providing general conclusions (chapter 8).

More particularly, the material presented in each chapter is summarized below.

Chapter 2 State of the Art. This chapter surveys the state–of–the–art main approaches related with the thesis contents. It is divided into four sections, accounting for. 1) Corpus–based NLP; 2) Part–of–speech Tagging; 3) Applying Machine–learning Techniques to NLP; and 4) Decision Trees. The main contribution is the broad survey into the application of ML paradigms in addressing natural language ambiguity problems.

Chapter 3 Tagging–oriented Language Modelling Using Decision Trees. In this chapter our approach of applying Statistical Decision Trees to model part–of–speech ambiguities is explained. We outline the general acquisition algorithm and some particular implementations and extensions. The models obtained are evaluated against a reference corpus with a rigorous methodology. The problem of *unknown words* is also addressed and evaluated in this chapter. Finally, two techniques are outlined to deal with the problem of data sparseness in training sets with few examples.

Chapter 4 Tagging with the Acquired Tree–based Models. This chapter is devoted to the construction and evaluation of taggers using acquired tree–based models. In particular, we present two new taggers (based on different approaches) that use the statistical information provided by the decision trees, in a fairly simple way. A concrete problem of specificity is discussed and an extension to the second tagger is presented in order to incorporate more general n-gram information in a back–off approach. Additionally, the possibility of adding more information is explored by adapting our trees to a flexible–model tagger based on relaxation labelling.

Chapter 5 Spanish Part–of–speech Tagging. In this chapter we develop taggers for Spanish in a framework defined by the LEXESP project, consisting of the annotation of a large, general–use Spanish corpus. We describe the annotation of this corpus, and we also propose an approach of tagger combinations to improve results when training material is scarce.

Chapter 6 Ensembles of Classifiers. In this chapter we explore the possibility of improving tagging results by paying more attention to the most relevant ambiguity cases. The approach studied consists of constructing ensembles of individual classifiers, and combining them in a voting strategy. This is a currently

emerging trend in the ML community, from which we explore some standard algorithms. Other particular approaches are also proposed. Finally, some criticism is given regarding the results obtained.

Chapter 7 Related Work: A Comparative Analysis. This chapter is devoted to the study of the evaluation and comparison of taggers. It consists of two parts. In the former, a list of problems regarding the comparison of taggers is presented, with particular attention to the problem of the noise in testing corpora. The latter provides a comparison between our work and other relevant approaches appearing in the literature, and previously described in chapter 2.

Chapter 8 Conclusions. The last chapter provides general conclusions, summarizes the research and contributions described in this thesis, and outlines several directions for further research.

Appendixes. Some appendixes have been added in order to cover the complementary details that, for the sake of brevity, have not been included in the chapters.

More precisely, the list of included materials is:

- Appendix A which includes the references of some papers published in related conferences and journals, mainly covering the contents of the present thesis.
- Appendix B which is devoted to presenting some definitions and technical details of the parts of the thesis which are not original work. In particular we describe the different measures for attribute selection, the pruning algorithm, and the relaxation–labelling algorithm.
- Appendix C fully describes the tagsets used for tagging the English and Spanish corpora.
- Appendix D includes a complete description of a real acquired decision tree which is partially presented as an example in chapter 3.
- Appendix E provides extra information about the research projects in which the present work has been developed.

CHAPTER 2

State of the Art

The chapter is intended to be a general survey of the state of the art in the main areas related with the thesis contents. It is divided into four sections, addressing the following topics: (1) Corpus–based NLP; (2) Part–of–speech tagging; (3) Applying machine–learning techniques to NLP; and (4) Decision trees from a machine–learning perspective.

Roughly speaking, the first one contains an historical introduction to corpus–based NLP, jointly with a broad set of references to introductory books and tutorial notes about related topics. In addition, it contains the description (with some comments about their degree of annotation) of several of the most popular corpora that serve as test benches for many NLP applications, and some interesting pointers to currently available tools and corpora.

Section 1 surveys the work performed around POS tagging, from the latest seventies to the present, paying special attention to the current lines of research, and including some discussion points about related issues, e.g., *smoothing*, treatment of *unknown words*, existing comparisons of taggers, etc.

Section 2 provides a broad–coverage compilation of references about the application of general machine–learning algorithms to address several NLP problems, involving some kind of language learning. This line of collaboration between both disciplines has been very productive in the latest years and we think that it would be even more in the near future.

Finally, the fourth section contains a description of tree–structured classifiers, from a machine learning perspective, with the aim of properly fixing the notation, and identifying some important issues concerning supervised decision–tree induction that will frequently appear in the following chapters.

1. Corpus–based Linguistics

In recent years many efforts have been devoted to to enrich the conceptual and linguistic components of the systems for natural language processing. While the more traditional NLP applications, such as natural language interfaces for accessing data bases, tutoring systems, expert systems, etc., could be designed within a relatively coarse semantic domain with a limited linguistic coverage, other applications such as spelling or grammatical correction, machine translation, automatic summarization, and especially information retrieval and information extraction are potentially applicable to any domain and they are required to process big amounts of information. Therefore, these systems should be able to surpass the reduced scope limitations to guarantee a minimum coverage and robustness. Additionally, most of these applications based on the massive processing of non restricted natural language texts are the focus of interest not only for the research community but also for industrials, since they may result in commercial products with a wide range

of application. We mainly have in mind all the NLP applications related to the use of Internet and WWW, which are nowadays clearly 'in fashion'.

The above described situation led to a big resurgence of the *empiricist* (by opposition to the *rationalist*) approach to natural language acquisition and processing during the last decade. This approach is based on the belief that most of the linguistic knowledge can be achieved through experience and that it can be acquired from textual corpora, by means of a small set of simple mechanisms, including association and generalization.

This new dimension of NLP has caused two main effects:

- The development of alternative (or complementary) techniques to the strictly linguistic techniques applied so far (the use of statistical techniques for syntactic analysis and for semantic interpretation of sentences is a significant example).
- The development of techniques for automatically acquiring linguistic knowledge (especially of lexical nature, but also structural), with the aim of substituting, or at least complementing, the very expensive manual construction of models.

There are many currently available surveys, tutorial notes, and books describing the application of these statistically-based techniques in the current NLP trends. Among others we recommend the following list:

- Two general (and relatively old) introductory papers by Leech and Fligelstone [**LF92**], and Church and Mercer [**CM93**]. See also the tutorial notes by Liberman [**LS93**].
- A compendium for a course on Statistical Approaches in Computational Linguistics by Krenn and Samuelsson [**KS97**], which emphasizes the statistical aspects. In the same direction, see also the tutorial notes [**Dag98**] by Dagan.
- The first comprehensive book about all these topics, "Statistical Language Learning" by Charniak [**Cha93**], and a useful complementary review of the same book provided by Magerman [**Mag95b**].
- From a more philosophical perspective, we recommend the article by Abney [**Abn96**] which, in words of the same author, is "an apologize for statistical methods, written for a linguistic audience". Also interesting from the linguistic perspective, we find the book by McEnery and Wilson [**MW96**], which also provides an introductory course on corpus linguistics accessible on the Web.
- Two more recent books covering similar topics: "Corpus-based Methods in Language and Speech Processing" [**YB97**], and "Foundations of Statistical Natural Language Processing" by Manning and Schütze [**MS99**]. Both are excellent. The first puts the accent on speech processing. The second is currently in press but a draft is available through Internet.
- Finally two more specific articles by Charniak [**Cha97**] and Abney [**Abn97**] which talk about statistical methods applied to parsing, and a survey on WSD by Ide and Véronis [**IV98**].

The last but not the least is the appearance of machine learning techniques for performing natural language learning. This trend aims to overcome some limitations of the pure statistical approach and is getting more attention every day.

See [**WRS96**] for a collection of relevant articles around connectionist, statistical, and symbolic approaches to learning for NLP. Additionally, in sections 2 and 3 of the present chapter, we provide specific information and references about ML techniques applied to corpus–based NLP with an special emphasis on POS tagging.

The aforementioned success of statistical methods in natural language processing would not have been possible without the existence of large amounts of machine–readable text from which statistical data could be collected. A compilation of naturally occurring linguistic phenomena in newspapers, literature, parliament acts, etc. is known as a *textual corpus* (or simply corpus), from which linguistic information can be derived. Corpora are constructed with the aim of compiling a representative sample of the usage of a language, a particular domain, a linguistic use, etc.

Using corpora as a source of linguistic information has many advantages.

- Huge amount of information: Small corpora contain about one million words, while the biggest contain over a hundred millions. As a consequence, statistical methods for processing and acquiring linguistic information apply very well.
- They are good benches to test and verify theories and intuitions about language.
- They are easy to classify by styles, domain, epoch, etc., and thus they allow a selective study and comparison of language utterances relative to the particular domains.
- They provide specific linguistic information which is very difficult to obtain from other sources or from introspection, and which is very useful for developing lexical resources, estimating parameters of statistical models, inducing grammatical structure, etc. We refer, for instance, to *collocations* (co–occurrrence relations between words, senses, semantic classes, syntactic categories, etc.), frequency counts of lexical units, typical contexts in which these units appear, lexical relations, examples of real use, argument structure (number, type, obligatory nature, etc. of the arguments that a particular verb admits), selectional restrictions (semantic restrictions that a verb can impose to its arguments), nominal compounds, lexicalized units, idioms, etc.

1.1. Corpora Compilation. The compilation of raw text corpora is no longer a severe problem, since nowadays most documents, books and publications are written on a machine readable support. But corpus have a higher linguistic value when they are *annotated*, that is, they contain not merely words, but also linguistic information on them (part–of–speech, semantic labels, syntactic analysis, etc.).

When a corpus compilation project is started, some important issues must be taken into account.

First, whether the corpus should be balanced or not. This is an open question that has not found a definitive answer in years. As stated in [**CM93**], it comes down to a tradeoff between quantity and quality: while American industrial laboratories (e.g. IBM, AT&T) tend to favour quantity, the BNC, NERC, and many dictionary publishers —especially in Europe— tend to favour quality. Biber [**Bib93**] claims for quality, since poor sampling methods or inappropriate assumptions can produce misleading results.

Second, which annotation will be included in the corpus, and how will be the annotation task performed. Automatic annotation introduces a certain amount of errors in the corpus, while manual annotation is very expensive in terms of human resources. Some research aiming to reduce the human effort when annotating training corpus is presented in [**ED96**]. It consists of algorithms which select the most informative samples that should be annotated to be later used in training. The same idea is present in the work by Lehmann et al. [**LORP+96**], who developed a database containing positive and negative examples of different linguistic phenomena, so that a test or training corpus focused on a certain phenomena can be built at a low cost. See [**ACO92**] for further information on corpus design and development.

1.2. Existing Corpora. The most well-known corpora are probably the Brown Corpus (BC) and the London-Oslo-Bergen (LOB) corpus. The BC [**FK82**] contains over a million words of American English and it was tagged in 1979 using the TAGGIT tagger [**GR71**] plus hand post-edition. The LOB corpus contains the same amount of British English was also tagged in 1979. These two corpora (and other linguistic materials) can be obtained through the International Computer Archive of Modern English (ICAME).

Nowadays, corpora tend to be much larger, and are compiled mainly through projects and initiatives such as the Linguistic Data Consortium (LDC), the Consortium for Lexical Research (RLC), the Electronic Dictionary Research (EDR), the European Corpus Initiative (ECI), or the ACL's Data Collection Initiative (ACL/DCI).

Those associations provide corpora as the Wall Street Journal (WSJ, 300 million words of American English), the Hansard Corpus (bilingual corpus containing 6 years of Canadian Parliament sessions), the Lancaster Spoken English Corpus (SEC), the Longman/Lancaster English Language Corpus, the Nijmegen TOSCA corpus, the 200-million-word Bank of English corpus (BoE) —tagged using the ENGCG environment [**Jär94**]—, or the 100-million-word British National Corpus (BNC) tagged with the CLAWS4 tagger [**LGB94**].

Surveys on existing resources can be found in [**Edw93, WSG96**]. Although most corpora limit their annotation level to part-of-speech tags, some offer higher level annotations and constitute an important source of knowledge for those researching in NLP. We find, for instance, syntactically analyzed corpora (also called treebanks) such as the Susanne corpus, the Penn Treebank (3 million words from the WSJ corpus) [**MMS93**], or the IBM/Lancaster treebank. Also, SemCor [**MLTB93**] contains over 200,000 words of the Penn Treebank semantically tagged using WordNet synsets [**MBF+91**]. A review of the state of the art of the art in using parsed corpora can be found in [**SA94**].

Some corpora have been specifically developed to serve as test benches for particular NLP tasks. This is the case of the corpora related with the Message Understanding Conferences (MUC) on information extraction, those related with the Text REtrieval Conferences (TREC) on information retrieval, and those of the SENSEVAL workshop on semantic disambiguation.

Until a few years ago, the existing corpora were all of the English language. Nevertheless, the success and applicability of corpus in linguistics as well as in NLP, has raised a wide interest and caused its quick extension to other languages. The

already mentioned ECI, the PAROLE project, or the ELRA association present significant contributions. The following list (not exhaustive) provides some examples of available corpora of languages other than English[1].

- *Spanish*: The LEXESP corpus [**CCP95, CCM+98**] which contains 5.5 million morphosyntactically tagged words, the corpus of the Real Academia Española, which contains 200 million tagged and lemmatized words, the CRATER corpus of morphosyntactically tagged telecommunication manuals, and the ALBAYZIN spoken corpus. A good information source on Spanish lexical resources is the report edited by the Instituto Cervantes [**Cer96**].
- *German*: The NEGRA corpus from the Saarland University, which contains German newspaper texts with syntactic annotation.
- *French*: The 'Trésor de la Lange Française' (TLF) which contains 150 million words of written French.
- *Swedish*: The 'Bank of Swedish' corpus and other materials collected by the Department of Swedish of the University of Göteborg.
- *Catalan*: The CTILC corpus from the Institut d'Estudis Catalans, which compiles over 50 million words of modern Catalan.
- *Basque*: The EEBS corpus, which contains a balanced sample of about four million words of modern Basque, and which is morphologically annotated and bracketed.
- *Bosnian*: The Oslo Corpus of Bosnian Texts, which consists of approximately 1.5 million words of general texts.

2. Part–of–speech Tagging

2.1. Linguistic Taggers. When automated part of speech tagging was initially explored in middle sixties and seventies [**Har62, KS63, GR71**], people manually engineered rules for tagging, sometimes with the aid of a corpus. The most representative of such pioneer taggers was TAGGIT [**GR71**], which was used for an initial tagging of the Brown Corpus (BC). From that time to nowadays, a lot of effort has been devoted to improving the quality of the tagging process in terms of accuracy and efficiency.

Current linguistic–based taggers still represent the knowledge involved as a set of rules, or constraints, written by linguists and obtained by introspection and available statistical tools for corpus studying. However, current models are much more expressive, comprehensive and accurate and they are used in very efficient disambiguating algorithms. The linguistic models range from a few hundreds to several thousand rules, and they usually require years of labour. The work of the TOSCA group [**Oos91**] and, more recently, the development of EngCG, an English shallow parser based on Constraint Grammars, at the Helsinki University [**VJ95, Vou95, KVHA95**] can be considered the most important in this direction. The Constraint Grammar formalism has also been applied to other languages, as Turkish [**OK94**] and Basque [**AAA+95**].

[1] A very useful annotated list of tools and resources for statistical natural language processing and corpus-based computational linguistics can be found at the URL: http://www.sultry.arts.usyd.edu.au/links/statnlp.html.

2.2. Statistical Taggers. The most extended approach nowadays is the statistical, or probabilistic, family. This approach basically consists of building a statistical model of the language and using this model to disambiguate a word sequence by assigning the most probable sequence of tags given the concrete sequence of words in a maximum–likelihood approach. The language model is coded as a set of co–occurrence frequencies for different kinds of linguistic phenomena, commonly reduced to a simple set of n–gram probabilities, collected from previously annotated corpora.

The seminal work in this direction is the CLAWS system [**LGA83, GLS87**], which used bigram information and was the probabilistic version of TAGGIT. It was later improved in [**DeR88**] by using dynamic programming for efficiently calculating the most probable sequence, and extended in a recent version CLAWS4 [**LGB94**] for tagging the British National Corpus (BNC). Other works that can be placed in the statistical family are those of [**Sam93, Sam95**].

Statistical taggers can be, equivalently, modelled with Hidden Markov Models (HMM). This approach has the advantage that the parameters of the model can be re-estimated with the Baum–Welch (or Forward–Backward) algorithm [**Bau72**] to iteratively increase the likelihood of the observed data (in this case the corpus). This permits to avoid the use of annotated training corpora or at least to reduce the amount of training data needed to estimate a reasonably good model. HMM were initially applied to speech processing in the 1970s by Baker at CMU and by Jelinek and colleagues at IBM. HMM–based taggers were imported from speech recognition and applied to tagging by [**BM76, DM84**]. The tagger by Church [**Chu88**], which used a trigram model, has been a basic reference for all successors [**DeM90, MSW91**]. The Baum–Welch re–estimation algorithm was first used in the XEROX tagger, by Cutting et al. [**CKPS92, Kup92**]. Among the numerous successors we may cite [**WSP+93, CHJP93, Mer94, Elw94, BCPS94, LGB94, SN95**]. In particular, Merialdo [**Mer94**] presents a valuable overview of statistical tagging.

Despite no pretagged text is necessary for training the previously described HMM–based taggers, a lexicon is still needed that supplies the possible parts of speech for every word. There have been some efforts at learning parts of speech with no a priori knowledge about grammatical categories, using unannotated corpora as the sole source of information in a *learning language from scratch* approach (see [**Pow97**]). Some examples in this direction are the works by Schütze [**Sch93, Sch95b**] considering only word distributions. Genetic algorithms have been used for the same purpose in [**Lan94a, Los94**].

Some authors have performed comparisons between linguistic (inside the Constraint Grammar framework) and statistical taggers, with favorable conclusions, in terms of tagging accuracy, to the linguistic family. This is the case of [**CT95, SV97**].

2.3. Machine–Learning–based Taggers. Although the statistical approach involves some kind of, either supervised or unsupervised, learning of the parameters of the model from a training corpus, and although machine learning algorithms for classification tasks can be seen as statistical in nature, we place in the machine–learning family only those systems that acquire more sophisticated information than a n–gram model and those that use classical paradigms of symbolic and subsymbolic machine learning.

First attempts in acquiring disambiguation rules from corpus were done by Hindle [**Hin89**]. More recently, Brill's tagger and variants [**Bri92, Bri94a, Bri95b, RS95, AH96**] automatically learn a set of transformation rules which best repair the errors committed by a most-frequent-tag tagger. The learning algorithm he proposed is called *Transformation-Based Error-Driven Learning* and it has been widely used in tackling several ambiguity problems of NLP. In [**Bri95b**] it is included a version with *semi supervised* training that achieves roughly the same accuracy.

Instance-based (or example-based) learning has been also applied by several authors to resolve a number of different ambiguity problems, and in particular to tagging. This is the case of [**Car93a, DZBG96**].

Decision trees have been used in POS tagging and parsing. This is the case of [**BJL+92b, Mag95a**]. The work that we present here also applies decision trees induced from tagged corpora to part-of-speech disambiguation [**MR97, MR98**]. Additionally, the previously referenced work [**DZBG96**] can be seen as an application of a very special type of decision trees.

Techniques borrowed from the grammatical inference field have been used also for constructing competitive taggers [**PP98**]. The work in [**STV96**] is an effort of automatically acquire Constraint Grammar rules from tagged corpora.

Within the subsymbolic approach, we would like to mention the works in [**NMKS90, Sch93, EG93, Sch94a, MI98**] that apply neural net architectures to POS tagging.

Finally, there also exist some mixed approaches. For instance the forward-backward algorithm (used to reestimate the parameters of a HMM) is used to smooth decision tree probabilities in the works of [**BJL+92b, Mag95a**], and, conversely, decision trees are used to acquire and smooth the parameters of a HMM model in [**Sch94b, Sch95a**]. Chapter 4 of this thesis represents another step in the same direction as it implements a HMM-based tagger that combines n-gram information with the acquired decision trees.

2.4. Current Research. The most recent efforts have been done in the following six directions:

2.4.1. *Increasing the complexity of the language models inside the statistical approach.* The speech recognition field is very productive in this issue. Recent works try to not to limit the model to a fixed order n-gram by combining different order n-grams, morphological information, long-distance n-grams, non-adjacent words, etc. In particular we find Aggregate Markov Models and Mixed Markov Models in [**BPS+92, SP97**], Nonuniform Markov Models [**RT96**], Hierarchical Non-emitting Markov Models [**RT97**], Triggering Pairs [**Ros96b**], addition of linguistic information [**BFHM98**], Mixtures of Prediction Suffix Trees [**PST95**], etc. Some of these approaches have already been applied to POS tagging. It is the case of: Variable memory Markov Models [**SS94**], Mixture of Hierarchical Tag Context Trees [**HM97**] and Triggering pairs applied to tagging and parsing [**BEKS98, BFK98**].

Additionally, Samuelsson [**Sam97**] generalized the standard model for HMM-based part-of-speech tagging of input word strings to handle input word graphs, were each word has been assigned a probability.

2.4.2. *Combination of statistical information.* The combination of statistical information of a different generalization degree has been proposed by several of the statistically-based taggers previously mentioned, as a way of dealing with the

sparseness of the data and to obtain more accurate estimations of the parameters of the model. However, relatively simple techniques have been used: Back–off, linear interpolation, successive abstraction, etc. Recently, there have been several efforts in the direction of providing more flexible environments to integrate and combine different sources of information (sometimes with the aim of providing a general framework for solving several ambiguity problems at a time). The following are some examples:

The Maximum Entropy (ME) approach —briefly described in section 2.5.3— provides a general way for combining statistical information from different sources [**JPCK96, Rat96**] that have proved to surpass the maximum–likelihood approach.

The combination of statistical and linguistic information has been performed usually inside rule/constraint–based environments, as in [**OT96, VP97, Pad96, EAA+98, TO98, TRG97**].

Part of the work described in this thesis (chapter 4) and referenced in [**MP97, Pad98**] presents a hybrid approach consisting of applying relaxation techniques over a set of constraints involving statistical, linguistic and machine–learned information.

2.4.3. *Improving the efficiency of taggers.* Several rule–based taggers have been impressively speedup by compiling the pattern–action rules into a single deterministic finite-state transducer (FST) [**Moh97**][2]. This is the case of Brill's tagger and others [**RS95, TR97**]. The more complex constraints appearing in the Constraint Grammar formalism can also be properly compiled into a single deterministic FST by using the *replace operator* [**Kar95, KK96**]. The resulting tagger [**Tap96**], also applied to Turkish [**TO98**], is one of the fastest and the most accurate of the existing English taggers. In [**Kem98**] it is suggested that the composition of FSTs is an adequate framework for the combination of several different rule–based taggers.

Finally, Hidden Markov Models can also be viewed as stochastic finite-state transducers [**PRS94**] and it is possible to closely approximate them by composing FST in a deterministic way [**Kem97, Kem98**], without a significant loss in accuracy. The achieved speedup is also very important.

2.4.4. *Improving tagging accuracy by combining different taggers in a voting approach.* The main contributions to this area are [**Hal96, BW98, HZD98, MI98, MPR98**]. The experiments and results presented in chapter 6 are also in the same direction. For more information see the introductory survey at the beginning of chapter 6.

2.4.5. *Construction of* POS *taggers for a great variety of languages.* This issue implies taking into account new problems as: Small training corpora, very big tagsets, complex morphology in highly agglutinative languages with productive inflectional and derivational morphological phenomena, etc. See, for instance, [**Kem94, Lud97, OK94, Sch95a, AAA+95, TR96, HH97, HH98, MPR98**].

In particular, taggers have been described for the following languages: Basque [**AAA+95**], Czech [**HH98**], Dutch [**DK95a, DZB96**], French [**CT95, TRG95**], German [**Fel95, Sch95a, LRW96**], Greek [**DK95a**], Italian [**DK95a**], Japanesse [**MMW93, HM97**], Portuguese [**ML96**], Swedish [**Cut93, BS95**], Spanish [**MT94, SN95, AH96, MPR98**], and Turkish [**OK94**].

[2] FST techniques had been successfully applied previously to morphological analysis [**KK94, Kos83**]

2.4.6. *Providing adaptive and transportable taggers.* The main goal here is to provide taggers which are able to move from one domain to another at a very low cost and without serious degradation in tagging accuracy. We think that relatively little attention has been paid to this important topic. However, Roth and Zelenko [**RZ98**] have recently presented the SNOW architecture: An adaptive tagger based on a network of linear separators, which benefits from the ability of learning while testing new examples (on–line learning), and, so, it becomes more adequate for any tuning task.

2.5. Related Issues. The following subsections supply some information about several important issues related to POS tagging, which can greatly influence the performance of the employed taggers, as well as the process of evaluation and comparison of taggers.

2.5.1. *The Tagset.* With respect to the tagset, the main feature that concerns us is its granularity, which is directly related with its size.

If the tagset is too coarse, the tagger accuracy will be much higher, since only important distinctions are considered and thus the disambiguating task to be performed is much easier. Nevertheless, the results would probably supply an excessively poor information. On the contrary, a too fine–grained tagset would enrich the supplied information, while would probably decrease the tagger performance —because the model will have to be much richer and so, more difficult to learn.

Even though a very complete tagset is used it has to be noted that some fine distinctions in POS tagging cannot be solved on the basis of purely syntactic or contextual information, but need some semantic, or even pragmatic, knowledge.

Some samples of commonly used tagsets can be found in [**KS97**], in which the word level tags —such as POS tags— are divided into three classes, according to the number of linguistic dimensions they specify:

- Single–dimension tags, which will usually contain the syntactic category of the word, such as N (*noun*), V (*verb*), A (*adjective*), D (*determiner*), etc.
- Multiple–dimension tags, which incorporate additional word features such as gender, number, person, etc. For instance the tag VIPS3 could indicate that a word form is a *verb* with the features: *indicative*, *present*, *third person*, and *singular*.
- Combination of separate multiple dimensions in sets (or *readings*). As in Constraint Grammar formalism, a word would have a set of labels, each one containing information on a single linguistic feature. For instance, (⟨SVO⟩ V PRES -SG3 VFIN) states that a word is a *verb*, *transitive*, *present*, *non–third singular*, *finite*. This representation has the advantages of allowing a selective use of features (i.e. by selecting some subset of them), and enabling the introduction of new dimensions, as for instance syntactic roles or semantic information.

Some studies on the tagset size influence on a tagger results have been done. For instance, Sánchez and Nieto [**SN95**] proposed a 479–tag tagset for using the Xerox tagger on Spanish, and later reduced it to 174 tags since the first proposal was considered too fine–grained for a probabilistic tagger. (this is the same situation that we will find in chapter 5 when applying the presented POS taggers for disambiguating a Spanish corpus).

On the contrary, rule–based linguistic taggers adapts very well to large tagsets. In the paper by Samuelsson and Voutilainen [**SV97**] a statistical tagger is compared to the Constraint Grammar based tagger. In order to make the comparison feasible, a pre–processing step of tagset reduction is required for the statistical tagger.

Elworthy [**Elw94**] states that the tagset sizes (48 tags for the Penn Treebank and 134 for the LOB corpus) do not affect greatly to the behaviour patterns of the re–estimation algorithms. The work in [**BCPS94**] is also related with this topic, since POS tagging experiments on different languages (English, Dutch, French and Spanish), each with different corpus and tagset were tested and compared. Finally, the work in [**TSHS96**] present an elaborate methodology for comparing taggers which takes into account, among others, the effect of the tagset on the tagger evaluation.

All in all, when a tagset is about to be designed, some aspects should be carefully considered in order to take appropriate decisions. Such aspects include: POS tagger technology to be used (e.g. statistical, rule based, supervised/unsupervised learning?), available resources (e.g. for hand annotating a big enough training corpus), type of NLP application (does category distinctions suffice or more complex information is needed for interacting with subsequent modules?), etc.

2.5.2. *Handling Unknown Words.* Another factor that can affect tagger accuracy is the way in which unknown words are handled. An *unknown word* is a word that it is not recognized by the NLP modules previous to the POS tagger. If a broad coverage morphological analyzer is used (as it is normal for highly inflected languages), unknown words consist basically of uncommon proper nouns, odd numerical expressions, typewriting or printing errors, etc., which represent a very small proportion of the total number of words, and which are generally very easy to identify and properly annotate. A more serious problem appears when a relatively small word–form dictionary, derived from the annotated training corpus, is used as the morphological analyzer (note that this is a common feature of existing English POS taggers). In this case, unknown words represent a bigger proportion of the total and they include many nouns, adjectives, verbs, etc., which do not occur in the training corpus.

The most usual methods for dealing wit unknown words are the following:

- Do not consider the possibility of unknown words. That is, to assume a morphological analyzer which provides any unknown word with the set of possible tags (usually with no information about relative probabilities). This approach, which is often referred to as the *closed vocabulary assumption* tends to produce higher performance results, though it is in fact less robust than the following (running under the open vocabulary assumption).

- Assume that unknown words may potentially take any tag —excluding those corresponding to closed categories (preposition, determiner, etc.), which are considered to be all known. Although this is more realistic than the previous approach, it introduces more noise (and sometimes high levels of ambiguity), and so the performance reported will probably be lower.

- Use available information to guess which are the candidate tags for a given unknown word. This is the most robust solution, and it has been applied in different ways by several researchers. For instance, Meeter et al. [**MSW91**],

and Weischedel et al. [**WSP+93**] take into account inflectional and derivational endings as well as capitalization and hyphenation to guess the possible POS tags for a word, while Adams and Neufeld [**AN93**] use a statistical model of fixed-length suffixes combined with capitalization features to guess the possibilities for unknown words. In a similar way, Schmid [**Sch94b**] and Samuelsson and Voutilainen [**SV97**] construct a statistical-based reverse suffix-tree. Ren and Perrault [**RP93**] perform a frequency study of the cases when a word is *actually* unknown or when it is a typewriting error, and a thorough subclassification of each case is exposed. Machine learning techniques are also used to deal with unknown words: Mikheev [**Mik96b, Mik96a, Mik97**] learns morphological rules from a lexicon and a corpus using unsupervised statistical acquisition, which can be later used to guess the possible tags for an unknown word. Finally, Brill [**Bri95a**] uses transformation-based error-driven learning to acquire a set of transformation rules for disambiguating unknown words, and Daelemans et al. [**DZBG96**] uses memory-based learning to acquire a specific base of examples for the same purpose.

As it will be explained in following chapters we first used the closed vocabulary assumption on the initial tagging experiments (chapter 4), and then we acquired specialized decision trees for dealing with the real unknown words (chapter 6).

2.5.3. *Smoothing Techniques.* Most of the supervised-learning approaches to POS tagging, and particularly the pure statistically-based ones, suffer from the problem of data sparseness.

The low or zero frequency events produce innacurate estimations for the probability of events than happen scarcely in the training set. For instance if event A is observed to happen once and event B to happen twice, the maximum likelihood estimates (MLE) consider B double probable than A, while this is not necessarily true. The zero-frequency events problem is even worse, since zero probability is assigned to events not observed in the training corpus, when they are not necessarily *impossible* to happen.

The term *smoothing* is applied to describe any method used to counteract the effect of statistical variability (which is particularly relevant when using small training sets). Smoothing can be done in many ways. For example by re-arranging the probability mass in order keep a part of it for unobserved events (count re-estimation or discounting methods), backing off to lower-order models, or combining models by linear interpolation (also called deleted interpolation). Maximum Entropy modelling is an alternative to deal with sparseness and with the combination of several statistical knowledge sources.

Count re-estimation methods, such as Add-One (also known as Laplace's law), or Good-Turing estimation [**Goo53**], try to correct the false estimations of rare events by re-distributing the frequency countings before the estimation. Add-One adds one to all frequencies, thus avoiding zeroes and reducing the proportion between rare happening events. Lidstone's law is a variation of Laplace's which adds not one but some smaller positive value λ.

Good-Turing redistributes the amount of observations to favour those events with less observations. Usually this redistribution is either smoothed or performed

only on low–frequency events, because it produces unreliable results for high frequency events. Church and Gale [**CG91**] presents a comparison of Add–One and Good–Turing techniques.

Smoothing through linear interpolation [**BJM83, BBDM89, Cha93**] is performed by computing the probability of an event as the weighted average of the estimated probability of its sub–events (which represent more general, and thus more frequent, information). For instance, the smoothed probability of a trigram could be computed as the weighted average of the estimated probability for the trigram itself, and for the corresponding bigram and unigram, that is,

$$P(x_n|x_{n-1}, x_{n-2}) \approx \lambda_1 \cdot \hat{p}(x_n) + \lambda_2 \cdot \hat{p}(x_n|x_{n-1}) + \lambda_3 \cdot \hat{p}(x_n|x_{n-2}, x_{n-1}),$$

where the optimal values for λ_i are usually computed with the Expectation Maximization (EM) algorithm [**DLR77**].

A particular case of linear interpolation is the Backing–off approach [**Kat87**], in which the combination of knowledge sources is performed by selecting only the best knowledge source among the available, instead of calculating a weighted average of all of them. The criterion for choosing the best information source (or history) is very simple: select the highest order history that has more than N examples (note that this is equivalent to set all λ weights to zero except that of the selected history, which is set to one, in the interpolative approach).

Other approaches to smoothing include: estimation of n–grams by using decision trees [**Sch94b**], Kneser–Ney smoothing [**KN95**], *Successive Abstraction* by Samuelsson [**Sam96**], using similarity metrics for smoothing in a Memory–Based Learning environment [**ZD97**], etc. See [**YB97**] for an excellent survey, and [**CG96**] for a complete empirical evaluation of several smoothing methods applied to language modelling.

Finally, a recent approach which handles the scarce data problem is Maximum Entropy Estimate [**Ros94, Ris97, Rat97b**], which on the contrary than MLE, assume maximum ignorance (i.e. uniform distribution, maximum entropy) and observed events tend to lower the model entropy. Under this approach, unobserved events do not have zero probability, but the maximum they can given the observations. That is, the model does not assume anything that has not been specified. Additionally, the ME approach allows an easy and natural combination of several knowledge sources, from which very weak hypothesis need to be assumed.

2.5.4. *Comparisons of Taggers.* Unfortunately there are few rigorous comparative studies of alternative approaches to POS tagging. By rigorous, we mean that the different POS taggers should be trained and tested using exactly the same corpora, and that simple performance figures do not suffice for extracting reliable conclusions, but it is necessary to observe the behaviour of the taggers when changing the granularity of tagsets, the domain of application, etc.

However, we want to mention some works around the comparison and evaluation of existing taggers. For instance, in two recent papers [**HZD98, BW98**] some English POS taggers are combined in order to improve their individual results. As a first step of this work, a comparison of all taggers is performed using the same training and test sets.

The first of these papers puts together a statistical trigram–based tagger, the MBT tagger [**DZBG96**], Brill's TBL tagger [**Bri95a**], and the ME–based tagger by Ratnaparkhi [**Rat96**]. The four taggers are tested on the LOB corpus with the

following results: The statistical tagger performed worse than the rest; MBT and TBL were indistinguishable; and ME performed better than the rest.

The second work considers the same set of taggers except of MBT. The test is performed on the WSJ corpus and again the ME approach shows a slightly higher performance than the others. Nevertheless, the comparison between the taggers was not the ultimate aim of these works and thus only absolute accuracy figures are reported.

Also regarding English POS tagging, Samuelsson and Voutilainen [**SV97**] performed a comparison between two antagonistic approaches, that is, statistical vs. linguistic. To do so, they tested on the Brown corpus a classical HMM trigram tagger and the EngCG-2 tagger [**Tap96**] based on Constraint Grammars. The results obtained were definitely favourable to the linguistic approach.

A similar comparison between statistical and linguistic taggers is performed in [**CT95**] for the French language. In this case the authors impose a time limit (one month) for developing both taggers from the scratch. After a month the obtained taggers achieved a similar performance to that of state–of–the–art English taggers. Despite the limited time spent on the developing of linguistic rules, the constraint–based tagger seemed to be superior than the statistical one.

Finally, regarding German language, several comparisons have been made. Two initial works [**TSHS96, LRW96**] compare the performance of a set of statistically based taggers. More recently, Volk and Schneider [**VS98**] compare again the TBL tagger with the statistical tree–based tagger by Schmid [**Sch95a**]. The test is performed on a German corpus and the performance achieved by both taggers is similar. However, the authors identify some types of words for which the behaviour of both taggers is different.

2.6. Acknowledgments. Part of the information appearing in the previous survey has been borrowed from previously reported good introductions and survey papers about POS tagging. Among others, we specially mention: [**Cha93, Mer94, KVHA95, Bri95a, Abn97, YB97, Pad98**].

3. Application of Machine–learning Techniques to NLP

Learning approaches are usually categorized as statistical (also probabilistic or stochastic) methods and symbolic methods, belonging to the latter the typical learning paradigms that do not explicitly use probabilities in the hypothesis (decision trees, instance–based learning, rule–induction systems, etc.)[3]. We will follow this criterion in the following exposition merely for clarity reasons. Additionally, we have treated separately the subsymbolic and connectionist approach, and we have included a last category, containing unsupervised approaches (all other referenced methods belong to the supervised family) and the recently emergent approach of combining classifiers. The main focus will be on the symbolic family.

3.1. Stochastic Machine Learning Approaches. Dietterich [**Die97**] define a *stochastic model* as a model that describes the real–world process by which the observed data are generated. The stochastic models are typically represented as a probabilistic network that represents the probabilistic dependencies between

[3]However, as Roth points in [**Rot98**], all learning methods are statistical in the sense that they attempt to make inductive generalization from observed data and use it to make inferences with respect to previously unseen data.

random variables. Each node in the graph has a distribution, and from these individual distributions, the joint distribution of the observed data can be computed. Different approaches vary in how this probabilistic network is acquired and in which is the method applied to combine individual probability distributions.

The most simple approach to stochastic classification is to use the Naive Bayes Classifier (NB), originally described in [**DH73**], which is based on the Bayes' theorem and the assumption of independence between features. Despite its simplicity it has been widely used in the ML and NLP communities with a surprising success. NB provides a simple way to combine information from several sources, however, when the statistical sources to combine are of different degree of generalization NB is usually combined with back–off estimates. We can find the NB algorithm (either the basic version or other variations and hybrids) applied to the following NLP disambiguation tasks: Context–sensitive spelling correction [**Gol95, GR98**], POS tagging [**Sam93, RZ98**], PP–attachment disambiguation [**CB95**], Word sense disambiguation [**GCY93, LCM98**] and Text Categorization [**LR94, SS98a**].

Recently, Lau, Rosenfeld and Roukos [**LRR93, Ros94**] have proposed a new approach for combining statistical evidences from different sources, that is based on the *Maximum Entropy Principle* (ME). This work was originated within the speech recognition field [**Ros94**], but it has also been successfully applied to: word morphology [**PPL95**], POS tagging [**JPCK96, Rat96**], PP–attachment disambiguation [**RRR94**], identification of clause boundaries [**RR97**], partial and general parsing [**SB98b, Rat97a**], and machine translation [**BPP96**]. See [**Rat98**] for a broad introduction to ME methods and a survey of existing applications.

Hidden Markov Models [**Rab90**], already referenced in the previous section, can be also seen as stochastic models of learning. HMMs had their major success in the low-level tasks of language disambiguation, that is, speech recognition and synthesis [**Rab90, JMR92**] and POS tagging [**Chu88, CKPS92, Mer94**] (see the previous section). However, there have been also efforts to extend the use of HMM to WSD [**SSGC97**], and partial parsing by tagging grammatical functions and bracketing simple constituents [**BSK97, SB98c**].

The *Expectation Maximization* (EM) algorithm is an iterative algorithm that starts with an initial value for the parameters of the model and incrementally modifies them to increase the likelihood of the observed data [**DLR77**]. Particular instances of the EM algorithm have been applied to a number of different problems in NLP. For instance, the parameter estimation of HMM models is done by the EM algorithm, namely the Baum–Welch or *Forward–Backward* algorithm [**Cha93**]. The *Inside–Outside* algorithm [**Cha93**] is another version of the EM algorithm related to the estimation of parameters for probabilistic grammars (Stochastic Context–Free Grammars, Stochastic Lexicalized Tree–Adjoining Grammars, etc.) and to the grammar inference from annotated corpus to produce robust parsers [**PS92, Bri94b, Cha93, LS93**]. The EM algorithm is also used in the *Linear Interpolation* approach for smoothing in HMM–based models [**Cha93, Mag95a**], in a version of unsupervised word sense disambiguation by Schütze [**SP95**], and, in combination with the naive Bayes classifier, in a semi–supervised approach to text classification [**NMTM98**].

Finally, *log–linear* models [**Chr97**] are also being applied to natural language processing. In particular, log–linear regression [**SD89**], a popular technique for binary classification, is used in [**Sie97**] to classify verbs for machine translation

purposes. In the same direction, the work by Marques et al. [**MLC98**] use log–linear models to induce verbal transitivity.

3.2. Symbolic Machine Learning Approaches.

3.2.1. *Decision Trees.* Decision tree based methods of supervised learning from examples represent one of the most popular approaches within the AI field for dealing with classification problems [**BFOS84, Qui79, Qui86, Qui93**]. See section 4 for details about the decision–tree paradigm of supervised learning.

Their application to NLP is also noticeable, and we find tree–based solutions to address natural language ambiguity problems at several levels: Speech recognition [**BBDM89, BD99**], POS tagging [**Sch94b, Mag95a, MR95, MR97**], word sense disambiguation [**BPPM91**], parsing [**Mag95a, HSO98**], text categorization [**LR94**], text summarization [**MB98**], dialogue act tagging [**SCVS98a**], co–reference resolution [**AB96, ML95**], cue phrase identification [**Lit94**], and machine translation (verb classification) [**Tan96, Sie97**].

In Magerman's approach [**Mag95a**], decision trees are used for a number of simultaneous different decision–making problems, such as: Assigning part–of–speech tags to words, assigning constituent labels, determining the constituent boundaries in a sentence, deciding the scope of conjunctions, etc. In a previous work [**BJL+92a**] a mixed, statistical and tree–based, approach was used to pick up the correct parse among all possibilities. Other mixed approaches are that of Schmid [**Sch94b**] and Kempe [**Kem94**] who introduced decision trees for estimating the transition probabilities in HMM–based taggers.

In concept–learning, decision trees are sometimes translated into rules (and eventually pruned) for representing the target concept. The most representative system is C4.5-RULES, a variant of C4.5 [**Qui93**], which is used for instance in [**MB98**] for automatic summarization.

There are other popular logic–based rule–induction systems that employ different representations of concepts: Disjunctive normal form (DNF), conjunctive normal form (CNF) and decision lists. The FOIL algorithm [**Qui90**] and some variants [**Moo95, WBM95**] have been widely used to acquire first–order logic representations, and, in relation to the NLP classification problems, we find them tested in many of the already cited works as benchmarks for any other applied inductive learning algorithm.

3.2.2. *Decision Lists.* Decision lists [**Riv87**] are ordered lists of conjunctive rules (where rules are tested in order and the first one that matches an instance is used to classify it) which have been applied in a number of concept–learning systems [**CN89, MC95, Qui96b**].

Decision lists work well in domains with many attributes (or with attributes with many values) because they avoid to some extent the data fragmentation problem. Thus, regarding NLP, they have been applied to lexical ambiguity resolution. In particular: Word sense disambiguation, lexical choice in machine translation, homograph disambiguation in speech synthesis and accent restoration [**Yar93, Yar94b, Yar94a, Moo96**].

3.2.3. *Transformation–Based Error–Driven Learning* (TBL). TBL was introduced by Brill in the early 90s, as a new approach to corpus–based natural language learning. The learning algorithm is a mistake–driven greedy procedure that produces a set of rules. It works iteratively by adding at each step the rule that best repairs the current errors. Concrete rules are acquired by instantiation of a predefined set of template rules.

This algorithm has been applied to a number of natural language problems, including part–of–speech tagging [**Bri92, Bri94a, Bri95b, AH96, RS95**], PP–attachment disambiguation [**BR94**], parsing [**Bri93**], spelling correction [**MB97**], and word sense disambiguation [**DTS98**].

One major drawback of TBL is its computational cost since all instantiations of templates are tested at each iteration to find the best rule. Recently, Samuel [**Sam98**] presented an efficient approximation called Lazy TBL which restrict the search to a small subset of all possible instantiations, by applying Monte Carlo sampling techniques, with a very slight decrease in accuracy. In this way, more complex problems can be faced using LTBL. In particular, Samuel and colleagues have applied this new algorithm to dialogue act tagging [**SCVS98a, SCVS98b**], that is, to label each utterance in a conversational dialogue with the correct *dialogue act*, which is a concise abstraction of a speaker's intention.

Another rule–learning system successfully applied to text categorization is Cohen's RIPPER algorithm [**CS96**]. In this case, the algorithm learns a classifier in the form of a boolean combination of simple terms.

3.2.4. *Linear Separators.* Linear separators with multiplicative weight–update algorithms[4], have been shown to have exceptionally good behaviour when applied to very high dimensional problems, in the presence of noise, and specially when the target concepts depend on only a small subset of the features in the feature space. Clearly, this is a usual scenario in the text processing domain. Roth and colleagues have designed the SNOW architecture [**Rot98**], a sparse network of linear separators in the feature space, using the WINNOW algorithm [**Lit88**], for on–line and adaptive learning. They have applied it successfully to a broad spectrum of natural language disambiguation tasks, including context–sensitive spelling correction [**GR98**], POS tagging [**RZ98**], and PP–attachment disambiguation [**KR98**], achieving state–of–the-art accuracies and surpassing several alternative algorithms.

Other methods based on linear separators have been applied to the text categorization task. Cohen and Singer [**CS96**] presented EXPERTS (based on the weighted majority algorithm [**LW94**]), Lewis et al. [**LSCP96, DBUG98**] used *Widrow-Hoff* [**WS85**] and EG (Exponentially Gradient) [**KW94**] algorithms to text categorization and routing. Again, another variation on the WINNOW algorithm, BALANCEDWINNOW$^+$ [**DKR97**], has reported the best results up to date in the text categorization task. All these algorithms proved to overcome one of the most commonly used techniques, the Rocchio algorithm and variants [**Roc71, Har92, LSCP96, SS98a, DBUG98**], which is a classifier that uses vectors of numeric weights to represent the data (vector space model) and works in a relevance feedback context.

3.2.5. *Instance–based Learning.* Instance–based learning algorithms [**AKA91, Aha97, LP97**] have appeared in several areas of the AI with many different names: Similarity–based, example–based, case–based, memory–based, exemplar–based, analogical, etc. It is a form of supervised, inductive learning from examples, based on keeping full memory of training material and classifying new examples by using a sort of k–nn (k nearest neighbours) algorithm.

[4]Linear threshold algorithms, like WINNOW, are very simple on–line learning algorithms for 2–class problems with binary (i.e., 0/1–valued) input features. To classify new examples, they simply calculate a weighted sum of input features (linear combination) and outputs 0 if the result is below the threshold, and 1 otherwise. Wrongly predicted training examples make the weights of the model change, in a multiplicative way, to better fit the training set.

We find several uses of this kind of algorithms in NLP tasks. Particularly relevant is the work by Cardie [**Car93a, Car94**], which addressed the lexical, semantic and structural disambiguation of full sentences (in limited domains), within an information extraction (IE) environment. Additionally, her instance-based system take advantage of decision-trees for identifying relevant attributes [**Car93b**]. More recent work refers to relative pronoun resolution [**Car96b**], and to the description of the *Kenmore Framework* [**Car96a**], a general framework that embedded machine learning algorithms for a global treatment of many natural language processing tasks.

Equally essential is the recent work of the ILK Group at Tilburg University. Daelemans and colleagues have developed the TiMBL (Tilburg Memory-based Learning Environment) which is a general instance-based algorithm that makes a compression of the base of examples into a tree-based structure, IGTree [**DBW97**], used later for classifying new examples. These trees have proved to reduce significantly the space requirements and to be very efficient and accurate in several domains, including: Phonology (stress, word pronunciation) and morphology [**Dae95, BDW96, DBG96, BWD98**], POS tagging [**DZBG96, DZB96, HZD98**], PP-attachment disambiguation [**ZDV97**], shallow parsing [**Vee98**], and smoothing of probability estimates [**ZD97**].

The work of other authors include applications to: Partial parsing (chunking) and context-sensitive parsing [**SY92, ADK98**], WSD [**NL96, Ng97, FITT98**], text categorization [**RL94, YC94**], semantic interpretation [**Car94**], machine translation [**Jon96**], and lexical acquisition by analogy [**FP96, FMPC98**].

3.2.6. *Inductive Logic Programming* (ILP). This is a discipline devoted to the inductive learning of Prolog programs from examples. The most relevant work in relation to natural language learning has been carried out by Mooney and colleagues at the University of Texas. A general survey of applications of ILP to NLP can be found in [**Moo97**]. Particular works include applications to: Grammatical inference [**ZM93, ZM94**], automatic induction of natural language interfaces for querying data bases [**ZM96, TMT97**], information extraction tasks [**CM97**], and learning the past tense of English verbs [**MC96**].

3.3. Subsymbolic Machine Learning Approaches.

3.3.1. *Neural Networks.* In their relation to NLP, neural networks [**Hay94**] have been used basically to address low-level problems, such as OCR [**SB98a**], speech recognition and synthesis [**SR87, Lip89, NMKS90, Lee96, WW96**], and POS tagging [**NMKS90, Sch93, EG93, Sch94a, MI98**]. The basic models refer to *feed-forward* multilayer neural networks trained with the *backpropagation* algorithm, but also include some examples of recurrent networks and ensembles of several single neural networks.

Other examples addressing more complex problems, sometimes in combination with symbolic approaches, are: Identification of clause boundaries [**HP94**], parsing and sentence analysis [**Leh91, CSL93, Lyo94, LD95**], grammatical inference [**LSG95**], PP-attachment disambiguation [**SLL98, Lóp98**], WSD [**TV98**], text categorization [**WPW95**], and detecting spelling errors [**Lew98**]. In [**Lóp98**] it is included a present day survey of neural networks with application to NLP[5].

[5]This survey is written in Spanish.

3.3.2. *Genetic Algorithms.* Genetic Algorithms [**Gol89, SP94**] have been basically used in language learning problems [**SW95**] from a non informed perspective, that is trying to infer word categories and syntactic structure from the sole source of unannotated corpora and with no a priori knowledge. This is an approach that it has been also addressed with unsupervised learning algorithms for clustering. Relevant contributions here are: [**Lan94a, Lan94b, SW95**] and [**Los94**], in which language learning is approached within an information retrieval and filtering framework. The work by Yang [**Yan93**] also applies genetic algorithms to information retrieval. Finally, in [**Sie97**] we find an application to verbal classification for machine translation purposes.

3.4. Others.

3.4.1. *Clustering Algorithms.* Concept formation and clustering algorithms are instances of the unsupervised machine learning paradigm [**Fis91, FPL91**]. They have been used in the NLP field in tasks such as: Document retrieval, automatic hyphenation, semantic, syntactic and phonological classification, extraction of hierarchical structure, machine translation, etc. See [**Pow97**] for a good survey including pointers to relevant references. Additionally, conceptual clustering algorithms were used by Cardie [**Car92**] to tackle relative pronoun resolution in a information extraction framework.

3.4.2. *Ensembles of Classifiers.* An ensemble of classifiers is a set of classifiers whose individual decisions are combined in some way (usually by weighted or unweighted voting) to classify new examples. These techniques have been mainly studied in the supervised learning area and it has been proven that ensembles are often much more accurate than the individual classifiers that make them up[6]. Within the voting (and variants) approaches, and related to NLP problems, we find ensembles applied to part-of-speech tagging [**HZD98, BW98, MPR98**], context-sensitive spelling correction [**GR98**], word sense disambiguation [**RAA97**], and anaphora resolution [**MBS98, Mit98**][7].

More complex combination strategies, and algorithms for constructing the ensembles, including stacking, bagging, boosting, etc. can be found in text categorization [**SS98a**] and text filtering [**SSS98**], where an adapted version of the popular ADABOOST algorithm is presented for information retrieval tasks. [**BM98**] is another relevant example of the combination of classifiers applied to information retrieval, in which the combination of classifiers allows the use of a big set of unlabelled examples (semi-supervised approach) to iteratively improve the classification accuracy in the task of filtering web pages.

Finally, chapter 6 and section 2 of chapter 5 are also devoted to apply this methodology to POS tagging.

3.5. A Reversed Summary. The already described survey about the interactions between the fields of NLP and ML has been organized by taking as a reference the machine learning paradigms and showing which NLP problems have been addressed by each of them. In the current section, we present a reversed summary, that is, indexing the information by the type of NLP task to be solved. The information is presented in the form of tables.

[6]The reader will find a broader introduction and several pointers to the machine learning methods for combining classifiers in the beginning of chapter 6.

[7]In these cases, the disambiguation is performed by straightforwardly combining a set of pre-existing classifiers, heuristics, or predictors.

The notation used for the machine–learning algorithms appearing in the tables is the following: DTs stands for Decision Trees, HMMs stands for Hidden Markov Models and statistical approaches, ME stands for the Maximum Entropy Approach, IBL stands for Instance–Based (or memory–based) Learning, NNs stands for Neural Networks, TBL stands for the Transformation–Based (error–driven) Learning, NB stands for Naive Bayes classifiers and derived approaches, LSM stands for Linear Separators (on–line classifiers with Multiplicative updating functions), GAs stands for Genetic Algorithms, Clust stands for Clustering algorithms, DLs stands for Decision Lists, ILP stands for Inductive Logic Programming, Rocchio stands for Rocchio's algorithm for text categorization, RI stands for Rule Induction algorithms, EC stands for any algorithm that uses ensembles of classifiers or simple combination of heuristics, and, finally, LogL stands for Log–linear Models.

Table 1 contains information about low–level NLP tasks, such as speech processing, morphology and POS tagging.

	DTs	HMMs	ME	IBL	NNs
Speech recognition and synthesis	[BBDM89, BD99]	[JMR92]	[Ros94]	[Dae95, DBG96, BWD98]	[SR87, Lip89, NMKS90, Lee96, WW96]
Morphology				[BDW96]	
POS tagging	See section 2 of this chapter				

TABLE 1. References corresponding to some low–level NLP tasks

	DTs	HMMs	ME	IBL
Clause Boudaries			[RR97]	
Shallow Parsing		[Chu88, Abn91, BSK97, SB98c]	[SB98b]	[ADK98, Vee98]
Parsing	[BJL+92a, Mag95a, HSO98]		[Rat97a]	[SY92, Car93b, Car93a, Car94]
PP–attachment disambiguation			[RRR94]	[ZDV97]

	TBL	NB	NNs	LSM
Clause Boudaries			[HP94]	
Shallow Parsing	[Bri93]		[Lyo94, LD95]	
Parsing			[Leh91, CSL93]	
PP–attachment disambiguation	[BR94]	[CB95]	[Lóp98, SLL98]	[KR98]

TABLE 2. References corresponding to syntactic analysis and structural ambiguity NLP problems

Table 2 contains the references about parsing (either shallow or general) and structural ambiguity resolution.

Table 3 groups the references about semantic and discourse-level NLP tasks, namely, sense disambiguation, co-reference resolution, anaphora resolution, dialogue act tagging, and text filtering and categorization, which are NLP tasks usually associated to information retrieval and information extraction.

	DLs	DTs	NB	TBL
WSD	[Yar93, Moo96]	[BPPM91, Moo96]	[GCY93, Moo96, LCM98]	[DTS98]
Text categorization and filtering		[LR94]	[LR94, SS98a, NMTM98]	
Dialogue act tagging		[SCVS98a]		[SCVS98b, SCVS98a]
Co-reference and anaphora resol.		[AB96, ML95]		
Cue phrase identification		[Lit94]		

	IBL	NNs	EC	Clust
WSD	[NL96, Ng97, FITT98]	[Moo96, TV98]	[RAA97]	[Sch98]
Text categorization and filtering	[RL94, YC94]	[WPW95]	[SSS98, SS98a, BM98]	
Co-reference and anaphora resol.	[Car96b]		[MBS98, Mit98]	[Car92]

	Rocchio	RI	LSM	GAs
Text categorization and filtering	[Roc71, Har92, LSCP96, SS98a, DBUG98]	[CS96, MB98]	[CS96, LSCP96, DKR97, DBUG98]	[Yan93, Los94]

TABLE 3. References corresponding to the discourse-level semantics NLP problems.

Table 4 summarizes the references corresponding to different levels (lexical, syntactic, semantic, etc.) of language acquisition.

Finally, table 5 contains references about other NLP tasks, such those related with machine translation, spelling correction, etc.

3.6. Acknowledgments. During the compilation of the above described references, we benefited from a number of good related introduction chapters and

	IBL	ILP	NNs	GAs	Clust
Lexical acquisition	[FP96, FMPC98]				[Pow97]
POS acquisition				[Lan94a, Los94]	
Grammatical Inference		[ZM93, ZM94, ZM96, TMT97]	[LSG95]	[Lan94b, Los94, SW95]	
Semantic acquisition (concept extraction)	[Car94]				[Pow97]

TABLE 4. References corresponding to automatic language inference tasks.

	DTs	ME	IBL	TBL	NB
Acquisition of verbal properties	[Tan96, Sie97]				
General machine translation		[BPP96]	[Jon96]		
Spelling correction				[MB97]	[GCY93, Gol95, GR98]

	DLs/ILP	NNs/Clust	GAs	LSM	LogL
Acquisition of verbal properties	[MC95, MC96, CM97]		[Sie97]		[Sie97, MLC98]
General machine translation		[YPM96]			
Spelling correction	[Yar94a]	[Lew98]		[GR98]	

TABLE 5. References corresponding to Machine Translation and other NLP tasks

small surveys. They are roughly included in the following list: **[WRS96, Car96b, DZBG96, Die97, SCVS98b]**.

4. A Machine–learning Oriented Review of Decision Trees

Decision trees are a way to represent rules underlying training data, with hierarchical sequential structures that recursively partition the data. They have been used for years in several disciplines such as statistics, engineering (pattern recognition), decision theory (decision table programming), and signal processing. More recently renewed interest has been generated by the research in artificial intelligence (machine learning, expert systems, etc.).

In all these fields of application, decision trees have been used for data exploration with some of the following purposes[8]: *Description* (i.e. to reduce a volume

[8]This classification is extracted from **[Mur95]**.

of data by transforming it into a more compact form), *Classification* (i.e. discover whether the data contains well-separated and meaningful clusters of objects) and *Generalization* (i.e. uncovering a mapping from independent to dependent variables that is useful for predicting the value of the dependent variable in the future).

Due to this diversity, many redundant efforts have been carried out by the different communities, and often quite different names are given to very similar topics. For that, the related literature may be sometimes very confusing. We recommend the introductory chapter in [**Mur95**] which provides a comprehensive and multi-disciplinary survey of the work performed around decision trees.

In the present dissertation, we will see decision trees from a machine-learning perspective and we will use them with the aim of acquiring an approximation of the underlying mapping between words and morphosyntactic categories that will permit us to predict the correct categories of new ambiguous words (i.e. using decision trees for generalizing —or inducing— classification rules).

The rest of the section is devoted to properly fix the terminology and the notation that will be used in the following chapters, and also to provide general bibliographical references to the most relevant topics around decision-tree induction.

4.1. Supervised Learning for Classification. The goal in supervised automated learning for classification consists of inducing an approximation (or *hypothesis*) of an unknown function f defined from an input space Ω to a discrete unordered output space[9]: $\{1,\ldots,K\}$, given a set of training examples: $T = \{(\mathbf{x}_1, f(\mathbf{x}_1)),\ldots,(\mathbf{x}_n, f(\mathbf{x}_n))\}$.

The components of each example $\mathbf{x}_i$ are typically vectors of the following form: $\langle x_{i,1}, x_{i,2},\ldots, x_{i,m}\rangle$, whose components, called *features* (or *attributes*) of $\mathbf{x}_i$, are discrete or real-valued. Therefore the objects of the domain are completely described by a set of attribute-value pairs, and a class label.

The function $f : \Omega \to \{1,\ldots,K\}$ defines a K-partition of the input space into sets $f^{-1}(k)$ called *classes* and denoted y_k.[10]

Given a training set T, a learning algorithm outputs a classifier, denoted h, which is a hypothesis about the true function f. This process of deriving classification rules from samples of classified objects is sometimes called *discrimination*. Given new $\mathbf{x}$ values, h predicts the corresponding y values, i.e. it *classifies* the new examples.

4.2. Decision Trees. Decision trees are n-ary branching trees that represent *classification rules* for classifying the *objects* of a certain domain into a set of mutually exclusive *classes*.

A particular decision tree contains zero or more *internal* (or *non-terminal*) nodes and one or more *leaf* (or *terminal*) nodes.

All internal nodes contain *splits*, which *test* the value of a mathematical or logical expression of the attributes. For each possible outcome of this test there is an *edge* to follow to the child nodes. Each internal node contains at least two edges to child nodes.

---

[9] When a continuous output space is considered we talk about *regression* (and *regression trees*) instead of *classification* (and *decision trees*).

[10] Sometimes f is viewed as a distribution (and not a deterministic mapping). In this more general situation f^{-1} would be ill-defined. Nevertheless, we state it as deterministic for the sake of simplicity.

Each leaf node has a class label associated with it, which represent the most frequent class of the examples associated to that leaf. If all of them belong to the same class, the node is said to be *pure*.

A *univariate* decision tree is one in which the test at each internal node uses a single attribute, while *multivariate* decision trees can use splits that contain more than one attribute at each internal node, typically a linear combination of them. Unless the contrary is said, from now on we will consider only univariate trees with discrete–valued attributes[11].

Classifying a new example $\mathbf{x}$ consists of computing its class label given its attribute values. Given a decision tree, this is done by simply following the convenient path starting on the root of the tree until a leaf is reached. The test at each internal node along the path is applied to the attributes of $\mathbf{x}$ to determine the next edge along which $\mathbf{x}$ should go down. The label associated to the ending leaf node is outputted as the class for $\mathbf{x}$.

Sometimes decision trees are used to provide probabilities instead of single classes. In this case we talk about *statistical* decision trees, which only differ from common decision trees in that leaf nodes define a conditional probability distribution over the set of classes. These probabilities are usually estimated from relative frequencies plus some kind of smoothing in order to get better estimations for the less represented events.

Sometimes decision trees are translated into classification rules (or weighted constraints) for using them in any rule–based system, or to get a logical representation of the concept learned[12]. The most popular system for acquiring rules through decision tree–induction is C4.5-RULES [**Qui93**].

4.3. Decision–tree Induction. The task of constructing the tree from the training set is called tree *induction*. Most existing systems for learning decision trees proceed recursively in a greedy top-down way. That is, they start by considering the whole set of examples at the root level and construct the tree in a top–down way branching at any non–terminal node according to the best split selected with a *goodness measure* (which is normally called *feature selection function*). The different outcomes of this test induce a partition of the set of examples. Then, the process is recursively applied in the resulting subsets of examples in order to generate the different subtrees. The recursion ends, in a certain node, when a predefined stopping criterion holds, e.g., when all (or almost all) examples of the target node belong to the same class, or when the number of examples is too small to make any statistical inference.

More details on the basic algorithm and its variants can be found in chapter 3.

This family of algorithms is sometimes called TDIDT, standing for *Top–Down Induction of Decision Trees*. Its most wide–spread representatives are: CART by Breiman et al. [**BFOS84**], Quinlan's ID3 [**Qui86**] and the successor C4.5 [**Qui93**], and ASSISTANT [**CKB87**] and ASSISTANT-R [**KŠR95**] by Cestnik and colleagues at the Ljubljana University.

[11] This is the simplest approach, which makes splitting equivalent to select the best attribute and partitioning according its concrete values.

[12] The *concept* underlying a data set is the true mapping between the attribute set and the class label. This is common terminology of concept–learning discipline.

This list is by no means exhaustive and it does not include the more recent evolutions of the systems, including several new features: *windowing*, resampling techniques such as *bagging*, *boosting*, filtering of non-relevant attributes, etc.

In the following subsections we will comment some particular aspects of the tree-induction algorithm, highlighting common problems and usual solutions and extensions.

4.3.1. *Greedy Search.* Decision-tree inductive learning algorithms typically use variants of greedy search to overcome the combinatorial explosion during the search for good hypothesis (i.e. no backtracking is performed). The major component of greedy search is a heuristic function that evaluates the potential successors, which in the case of univariate-tree induction is the function that decides the best attribute for branching typically by ranking them according to a goodness measure. The goodness of an attribute is usually based on quantifying the impurity reduction that results from partitioning the data according its values.

There is a large number of approaches to feature selection in the literature, which can be classified into three categories according to the classification of Ben-Bassat [**BB87**]: (1) Functions derived from information theory (i.e. on the concept of Shannon's entropy); (2) Functions derived from distance measures between class probability distributions; and (3) Functions derived from dependence measures between random variables (also called statistical criteria).

There are also several empirical studies that compares different approaches for feature selection. The following list include some of the more referenced in the ML literature: [**Min89, BN92, Lop91, WL94, LCG96**]. In chapter 3 we also test a number of feature selection functions belonging to the three proposed categories. As in many other studies we find that not much significant differences can be observed between them.

The problem of ranking features in tree-induction algorithms can be seen as a particular instance of the more general problem of ranking a set of attributes according their relevance to the problem at hand in order to obtain a good (and ideally small) subset of features to work with. This general problem of feature subset selection is recently the focus of much attention in the ML community since scaling up inductive-learning algorithms to domains with thousands of features requires to be able to previously select the relevant attributes and to filter the non-relevant or highly correlated ones. Excellent surveys on the selection of relevant features can be found in [**BL97**] and [**Die97**].

Some extensions of the purely greedy inductive-algorithms have been carried out in order to overcome their 'myopia'. They include: Designing of feature selection functions for better capturing the correlation between attributes [**KŠR95**], searching with a limited lookahead [**Mur95**], performing a *beam* search, etc. These approaches have been successfully applied to some (not all) domains, and in all cases they introduce a clear trade-off between the myopia and the search complexity, which has to be particularly considered for each domain of application.

4.3.2. *Overfiting.* Basic learning algorithms tend to derive hypotheses that fit the learning data as much as possible. When this bias is exaggerated the generalization ability of the tree is substantially reduced. In this case, it is said that the classifier *overfits* the training data. In the case of noise, the constructed hypothesis

usually overfits the learning data (and thus also fits the misclassified examples) and yield poor results on testing data at the same time.

Tree pruning is probably the most used mechanism for handling noise and overfiting. The basic idea is that branches in a tree that are not statistically significant should be cut off. As a result, not only the prediction accuracy on an independent data is improved but also the tree size is reduced significantly.

There are two variants of tree pruning depending on when the pruning process actually occurs.

If the pruning is performed by deciding to stop the recursive partition of the data on a certain impure node, i.e. pruning in advance during the construction of the tree, we talk about *pre-pruning* (or *forward pruning*). This kind of pruning is usually done by defining appropriate *stopping criteria.* Usual stopping conditions are: (1) A very high proportion of the examples of the target node belong to the same class; (2) The number of examples is less than a fixed threshold or not enough to make any statistical inference; (3) All the examples share the same feature vector, despite they may belong to different classes (i.e. they are indistinguishable); or (4) The benefit attributed to all possible tests which partition the set of examples is too low.

If the pruning is performed by cutting off some branches after constructing a full–sized tree we talk about *post–pruning* (or *backward-pruning*). Many methods have been proposed for post–pruning decision trees. Among others we find: Reduced Error Pruning [**Qui87**], Pessimistic Error Pruning [**Qui87**], Minimum Error Pruning [**NB86**], Cost–Complexity Pruning [**BFOS84**], Error–Based Pruning [**Qui93**], etc. Several studies indicate that the second choice is preferred due to its theoretical soundness.

An alternative to pruning would be that of smoothing (also backwards) the conditional probability distributions of the leafs of the tree using a fresh part of the training set [**Mag96**].

See [**EMS97**] for a complete survey and an empirical comparison of several decision–tree pruning algorithms.

4.3.3. *Splits.* Univariate trees imposes a partitioning in the ordered attribute space that can be geometrically represented as a collection of hyperplans and regions. It is observed that this type of partition is not adequate in some classification problems, and multivariate trees are proposed as a solution for those cases. Most of the work on multivariate decision trees has been done by allowing some kind of linear combination of attributes in the splits (*oblique* decision trees).

The induction of oblique trees lead to better results in some cases but the acquisition of optimal linear splits is computationally very expensive, and heuristic methods are required for finding good subobtimal combinations. See again [**Mur95**] for a good survey on mutivariate decision trees.

4.3.4. *Instability and High Variance.* Decision–tree, as well as other supervised learning methods such as neural networks and rule learning algorithms, are said to be unstable because small variations on the training set can provoke great variations on the induced classifiers. Some methods, mainly based on resampling, randomization, and recasting the set of classes, have been designed to construct ensembles of unstable classifiers in order to reduce their high variance: *Bagging* [**Bre96a**], AdaBoost [**FS95**], ECOC [**DB95**], etc., are some significant examples. It is observed

that the combination of several classifiers contribute to reduce bias and variance and to improve the global performance.

For a survey on constructing ensembles of classifiers see [**Die97**] and the introduction to chapter 6.

4.3.5. *The Problem of Small Disjuncts.* For learning systems that describe a learned concept as a disjunction of conjunctions of conditions (as decision trees do), small disjuncts are disjuncts which cover a small number of training instances and often entail high error rates. On the other hand, large disjuncts cover a large proportion of the training instances and have low error rates. It is observed that the learning system's bias to 'maximum generality' is the main cause of small disjuncts problem, while changing the bias to 'maximum specificity' for small disjuncts, successfully improves their prediction accuracy [**HAP89**]. However, the same work also concludes that it is difficult to eliminate the error-prone small disjuncts without affecting the performance of large disjuncts.

In decision-tree induction, the 'maximum specificity' bias would correspond to construct full-sized trees with no post-pruning, which tend to produce overfiting. The application of pruning methods biases the learning method to a more general representation, but it emphasizes the problem of small disjuncts. Therefore, there is a difficult trade-off between the generality of the representation and the prediction accuracy on different types of examples.

Some authors [**Tin94, Koh96**] propose a solution of mixing decision trees with instance-based learning, which has higher predictive accuracy on small disjuncts, to create a hybrid classifier. General work on ensembles of classifiers also contributes to alleviate the problem of small disjuncts. Our proposal in chapter 6 of constructing ensembles of decision trees using features at different levels of specificity can be considered in the same direction.

4.4. Acknowledgements. The general overview on decision-tree induction already presented has been developed using the information provided by three journal articles [**BCK96, EMS97, BL97**], a Phd dissertation [**Mur95**] and a book on automated knowledge acquisition [**SD94**].

CHAPTER 3

Tagging–oriented Language Modelling Using Decision Trees

To enable a computer system to process natural language, it is required that language is *modelled* in some way, that is, that the phenomena occurring in language are characterized and captured, in such a way that they can be used to predict or recognize future uses of language. Rosenfeld [**Ros94**] defines language modelling as *"the attempt to characterize, capture and exploit regularities in natural language"*, and states that the need for language modelling arises from the great deal of variability and uncertainty present in natural language.

As described in chapter 1, language models can be hand–written, statistically derived, or machine–learned. The present chapter is devoted to present the application of decision–tree induction to acquire language models suitable for addressing POS tagging[1]. In subsequent chapters we will see how these machine–learned models can be used to disambiguate a input sequence of ambiguous words, and how to combine them with pure statistical information and even with hand–developed linguistic information.

The chapter is organized as follows: Sections 1 and 2 present the general framework for POS tagging and the particular implementation of the tree–induction algorithm; Section 3 is devoted to the evaluation of the acquired models on a corpus of reference; and section 4 supplies information about some extensions of the basic approach that we apply for improving performance and for handling unknown words and low represented cases of ambiguity.

1. Setting

Our approach is based on the fact that POS tagging, as any NLP disambiguation problem, can be easily interpreted as a classification problem (see section 4 of chapter 2). In this case, the finite set of classes is identified with the set of morphosyntactic tags (the tagset), and the underlying concept to be learned is the unknown mapping between a word and the appropriate tag in a particular context of occurrence.

In the following subsections we will define at which level of granularity the disambiguation problem is stated, and which are the common contextual features for describing the learning examples that best help disambiguating.

1.1. Ambiguity Classes. The first problem that arises from this approach is to determine an adequate level of granularity. For instance, POS tagging could be straightforwardly stated as a unique classification problem for all words, with

[1] As we explained in chapter 2, decision trees have been successfully applied to many domains in NLP, and, in particular, in POS tagging they have proven to be a very efficient and compact way of capturing the relevant information for disambiguating.

as many classes as tags in the tagset. However, this approach presents two clear problems. On the one hand, the learning algorithm would probably have some problems to efficiently handle and exploit the huge amount of examples present in the training set (usual training corpora may have around one million words), and, on the other hand, the number of possible outcomes would be too high (tagset sizes usually move from 40–50 to several hundreds) to obtain an accurate classifier.

The opposite pole is to consider a classification problem for each different word. In this way, the number of classes is significantly reduced, since ambiguous words admit a limited number of readings. The problems here are the opposite, namely, too many learning problems (thousands of different ambiguous word-forms can be found in unrestricted English corpora) and not enough examples for many infrequent words.

We have chosen an intermediate level consisting of identifying the different types of ambiguity occurring in the corpus and treating them as separate classification problems. These types of ambiguity, *ambiguity classes* from now on, group the words of the training corpus into equivalence classes according to the set of readings they can take (e.g., adjective–noun, noun–verb, adjective–noun–verb, etc.)[2]. In this way, both the number of problems and the number of training examples keep into reasonable bounds (for instance, in the one million word WSJ English corpus, which is tagged with the Penn Treebank tagset, the number of ambiguity classes is around 200 and the number of training examples range from few hundreds to 20-30 thousand), while the number of classes is fairly small.

Figure 1 depicts some of the ambiguity classes appearing in the WSJ corpus[3]. In this figure, ambiguity classes are presented in different layers according to the degree of ambiguity (i.e. the number of possible tags), and the edges between classes stand for an inclusion relation. For instance, all words that can be adjective, adverb and noun, can be, in particular, adverb and noun), that is, NN-RB is included in JJ-NN-RB. Note that this taxonomic representation has a DAG structure.

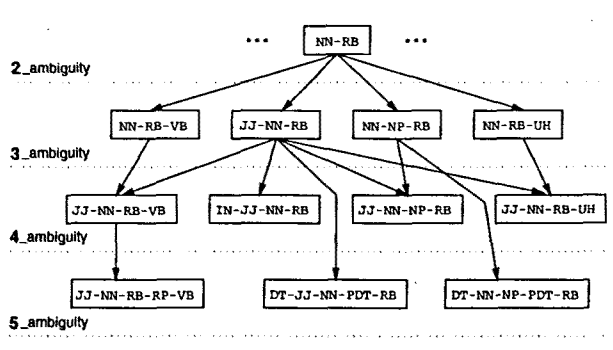

FIGURE 1. A part of the ambiguity-class taxonomy for the WSJ corpus

[2]The use of ambiguity classes is not a new idea. In [**CKPS92**], and also in [**TRG97**] (under the term *genotype*) ambiguity classes are used as way to alleviate the sparseness problem when collecting lexical probabilities from small training corpora.

[3]The meaning of the tags contained in the figure can be found in appendix C, where a full description of the Penn Treebank tagset is supplied.

Even though ambiguity classes are considered, there still exist cases in which there are very few examples for learning. This problem of ambiguity–class sparseness will be especially addressed in the final part of this chapter (section 4.3), in which two procedures for enlarging the set of training examples will be proposed and tested: the first works by successively collecting more general examples through the inclusion relation of the ambiguity–class taxonomy, and the second consists of generating new examples from the existing data by applying a procedure of random combination of features to pairs of examples. The latter will be tested in a real tagger in chapter 6.

1.2. Context Modelling. Usual POS tagging algorithms, based on automatic acquisition, use very limited contextual information to perform the disambiguation. Typically, the morphosyntactic categories of the surrounding words, in a narrow window of two or three words, suffice to perform disambiguation at a very good level of accuracy (statistically, and machine–leaning based English taggers, show accuracies consistently over 95%). However, this fact makes POS tagging a very particular problem since to perform the disambiguation the algorithm needs to know the POS tags of the neighbour words, which implies that the text should be previously disambiguated ¡!

Statistical taggers solve this circularity by calculating the most probable sequence of tags given an estimated set of transition probabilities. Other approaches to tagging have to make some kind of guessing about the category of the neighbour words when it is required (e.g. by choosing the most probable reading). Other possibilities are: to perform disambiguation in a particular direction and use only the information of the already disambiguated part to continue disambiguating the rest (e.g. to proceed from left to right and use the information about left context), or to use the morphosyntactic information in a broader sense, i.e., considering the set of all possible tags for the still ambiguous words, which can be weighted by relative probabilities.

Another relevant aspect of POS tagging is that the baseline accuracy is very high. Using very naive information a global accuracy slightly over 90% can be obtained for English tagging, which drastically reduces the room for improvement[4]. This is due to two main reasons: (1) The amount of unambiguous words is usually over 60%; and (2) Many readings are very infrequent at a word level, therefore a heuristic function that picks the most probable tag for each ambiguous words gets a very high accuracy (typically over 80% on ambiguous words). This fact makes the estimation of lexical probabilities[5] a very important issue.

Due to the above described situation, our basic context modelling will consist of: (1) The POS tags of the neighbour words in a window covering 3 tags to the left and 2 tags to the right[6]; and (2) The target word form —as a way to capture

[4]Nevertheless, we saw in chapter 1 that 8–10% error rate on POS tagging is inadmissible for most NLP applications.

[5]The term 'lexical probabilities' refers to the conditional probability distribution over the possible tags, given a particular word. They are usually estimated from frequency counts performed on an annotated corpus.

[6]The size of the window was empirically determined. The consideration of larger windows reported no benefits but the acquisition of too specific, and sometimes non–relevant, information which decreased the generalization ability of the acquired classifiers over unseen examples.

specific behaviours of some words, and implicitly including information about lexical probabilities[7].

Table 1 presents this basic set of of discrete–valued attributes used for acquiring decision trees. The Type column refers to the way of determining the set of values that hold for the attribute in a particular step of the learning algorithm: S stands for *static*, while D stands for *dynamically–valued*. This particularity that applies to the attributes with many values will be explained in subsection 2.2.4.

	Attribute	Values	Type
1	tag(-3)	Any tag in the Penn Treebank tagset	S
2	tag(-2)	"	S
3	tag(-1)	"	S
4	tag(+1)	"	S
5	tag(+2)	"	S
6	word(0)	Any word of the ambiguity class	D

TABLE 1. Set of basic attributes

In section 4.2, the basic set of attributes is enriched with lexical features about the local context, and the acquisition algorithm is tested with several combination of features. The basic set of attributes is also extended with a number of features about the orthography of the target word for dealing with unknown words. This is addressed in section 4.1

For each ambiguity class, the set of training examples is built by selecting from the training corpus all the occurrences of the words belonging to this ambiguity class. Table 2 shows some real examples belonging to the training set for the words that can be preposition and adverb (IN-RB ambiguity class).

tag(-3)	tag(-2)	tag(-1)	⟨word(0),tag(0)⟩	tag(+1)	tag(+2)
RB	VBD	IN	⟨'after',IN⟩	DT	NNS
VB	DT	NN	⟨'as',IN⟩	DT	JJ
DT	JJ	NNS	⟨'as',RB⟩	RB	IN
JJ	NN	NNS	⟨'below',RB⟩	VBP	DT
			...		

TABLE 2. Training examples for the preposition–adverb ambiguity class

In our work we do not deal with long distance syntactic relations to help part–of–speech disambiguation. This is certainly a limitation because it is obvious that simple context conditions cannot resolve all POS ambiguities[8]

Long distance conditions are rather common in the outstanding linguistic rule–based taggers. In the work by Voutilainen around English POS tagging and shallow parsing using the Constraint Grammar framework [**Vou94**], several especially designed rules ask for some condition to hold in some non–limited position to left or right of the target word (always inside the sentence boundaries).

[7]Lexical probabilities will be introduced also in the tagging algorithms presented in chapter 4, as an informed starting point about word–tag probabilities for performing disambiguation.

[8]The semantic and pragmatic clues necessary to resolve some noun–adjective ambiguities are beyond the scope of this work.

However, these rules should be applied with much caution since the possibility of generalize in a broader context is very uncommon. So they are usually provided with a strong set of provisos (which are a list of exceptions called *barriers*) for preventing undesired applications of the rule. As an example, see the rule defined in section 4.2 of chapter 4 which states that a verb–participle tag should be assigned when the target word is preceded (at any distance) by an auxiliary verb, provided that there is no other participle, preposition, adjective nor any phrase change in between.

This type of sophisticated rules are very difficult to acquire by general machine learning procedures, because if the list of exceptions to be considered is large, the underlying rule will be not much significant at the beginning and its chance to appear will be very low.

Despite the local–context limitation, most statistical and machine–learning tagging algorithms achieve good results because they look for a tag assignation that maximizes a certain global compatibility on the input sequence. Thus, in some sense local conditions are propagated along the sentence to find the best assignment. Additionally, it has to be said that POS ambiguities fully depending on a long distance condition are by no means the most frequent ones.

The above described characteristics make impossible to acquire decision trees to completely classify the training examples[9]. Instead, we aspire to obtain *statistical* decision trees representing fairly accurate probability distributions of the words over their possible tags, conditioned to the more relevant contexts of appearance.

Therefore, instead of using the acquired trees as direct classifiers, we aim to incorporate them as a statistical module into more complex statistically–based tagging algorithms. This inclusion could be performed under different internal representations: decision trees, weighted rules, weighted constraints, etc., depending on the type of POS tagging algorithm at hand.

2. Automatic Acquisition

2.1. Basic Algorithm. Regarding the tree–learning algorithm, we have implemented a particular algorithm belonging to the 'top–down induction of decision trees' family of supervised learning algorithms. This algorithm is quite similar to the well–known CART [**BFOS84**], and C4.5 [**Qui93**], but it is adapted to our particular domain.

Recall that classical a TDIDT–algorithm is a recursive procedure that departs from considering the whole set of examples at the root level and constructs the decision tree in a greedy top–down fashion by recursively partitioning the training data at each internal node according to the possible outcomes of a test on some of the available attributes. In this case we consider simple tests on a single discrete–valued attributes, thus the partition is induced by the different values of the selected attribute (i.e., one edge to follow for each value).

The basic inductive algorithm is depicted in figure 2. In that figure, X stands for the set of training examples, and A stands for the set of attributes. The functions involved are described below.

[9]Another factor that should be mentioned here is the fact that there is a significant level of noise in both the training and test data more commonly used (WSJ corpus is estimated to have about 2–3% of mistagged words).

```
function TDIDT (X: set-of-examples; A: set-of-features)
  var: tree1,tree2: decision-tree;
       X': set-of-examples;
       A': set-of-features
  end-var
  if (stopping-criterion(X)) then
    tree1 := create-leaf-tree(X)
  else
    amax := feature-selection(X,A);
    tree1 := create-tree(X,amax);
    for-all val in values(amax) do
      X' := select-examples(X,amax,val);
      A' := A-{amax};
      tree2 := TDIDT (X',A');
      tree1 := add-branch(tree1,tree2,val)
    end-for
  end-if
  return (tree1)
end-function
```

FIGURE 2. Pseudo-code of the TDIDT algorithm

- *stopping-criterion*(X): returns true if the stopping criterion holds for the set of examples X, and false otherwise.
- *create-leaf-tree*(X): returns a tree consisting of a single leaf node with the most frequent class in X.
- *feature-selection*(X,A): selects, from the set of attributes A, the attribute that best help distinguish between different-class examples of X.
- *create-tree*(X,a): returns a tree consisting of a single internal node with the information about the attribute a which will be used at the next step to create the descendant subtrees.
- *values*(a): returns the set of possible values for attribute a.
- *select-examples*(X,a,v): returns the subset of examples of X that have the value v for the attribute a.
- *add-branch*(t_1,t_2,v): returns the tree that results from adding the child tree t_2 to the root of the parent tree t_1. The new branch is labelled with the value v.

2.2. Particular Implementation. Our implementation roughly follows the basic algorithm presented in the previous section, however, we have introduced some variations in order to better adapt it to the particular domain of application. The following subsections (from 2.2.1 to 2.2.7) are devoted to briefly explain some choices and solutions applied to some particular problems.

2.2.1. *Attribute Selection Functions.* As we have seen in section 4 of chapter 2 the heuristic function for selecting the most useful attribute at each step of induction is very important in order to obtain simple trees with a high degree of generalization.

We have implemented and tested several well–known feature selection criteria belonging to the three existing families. They are listed below and explained in detail in appendix B:

1. Functions derived from information theory, which measure the entropy decrease resulting from partitioning the data according to one or another attribute. From this family we considered Quinlan's *Information Gain* [**Qui86**] and *Gain Ratio* [**Qui86**]. We will call them IG and GR, respectively.
2. Functions derived from distance measures between class probability distributions, which look for the partition that obtains class probability distributions as pure as possible (recall that a set of examples is said to be pure when all the example belong to the same class). From this family we consider the *Gini Diversity Index* of impurity [**BFOS84**], and RLM, which is a distance–based function by López de Mántaras [**Lop91**] that uses Shannon's entropy to define the distance measure.
3. Statistically–based functions derived from dependence measures between random variables. From this family, we used a χ^2 *Test* [**SD94**] and the *Symmetrical Tau* criterion [**ZD91**].

Apart from these functions we also implemented a variant of RELIEFF [**KŠR95**] that we will call RELIEFF-IG. RELIEFF is a non–myopic function that scores the attributes by also considering the conditional dependencies between them (on the contrary, all the above described feature selection criteria assume that attributes are conditionally independent given the class). RELIEFF has been also used to address the problem of finding the a subset of relevant features by preprocessing. In our implementation, RELIEFF-IG use Quinlan's Information Gain to weight the features in order to make a better use of the k-nearest neighbour retrieval algorithm inside RELIEFF.

2.2.2. *Splits.* When dealing with discrete attributes, usual TDIDT algorithms consider a branch for each value of the selected attribute. However there are other possibilities. For instance, some systems perform a previous recasting of the attributes in order to have binary–valued attributes [**Mag96**]. The motivation could be efficiency (dealing only with binary trees has certain advantages), and avoiding excessive data fragmentation (when there is a large number of values). Although this transformation of attributes is always possible, the resulting attributes lose their intuition and direct interpretation, and explode in number.

We have chosen a mixed approach which consists of splitting for all values, and subsequently joining the resulting subsets into groups for which we have insufficient statistical evidence for there being different distributions (this statistical evidence is tested with a χ^2 test, with a previous smoothing of data in order to avoid zero probabilities). In this way the resulting trees contains splits which test a disjunction of some values of a concrete attribute.

The algorithm for merging redundant branches does not test all possible groups of values (which would have an unfeasible exponential cost), but it takes greedy decisions based on the exploration of the values in a fixed order. Despite the problems introduced by the naive nature of the algorithm (for instance, the result depends on the order in which values are considered), we obtained very good results with this technique that helps preventing the data fragmentation (see section 3.3.1 for the results of the evaluation).

2.2.3. *Right-sized Trees.* In order to decrease the effect of overfiting —which in this domain is quite important due to the noise in the training set— we have used a post pruning technique (although some simple stopping criteria are also considered). In a first step the tree is almost completely expanded and afterwards is pruned following a minimal cost-complexity criterion (CART pruning [**BFOS84**]).

Roughly speaking this is a process that starts with the full-sized decision tree, from which it iteratively cuts those subtrees producing only marginal benefits in accuracy, obtaining smaller trees at each step. The trees of this sequence are tested using a comparatively small fresh part of the training set (10%-15%) in order to predict which is the most accurate tree on new examples.

The alternative, of smoothing the conditional probability distributions of the leaves using a fresh corpus proposed by Magerman in his SPATTER tree-based parser [**Mag96**], has been left out because we also wanted to reduce the size of the trees.

2.2.4. *Attributes with Many Values.* Attributes with many values may produce the undesired effect of contributing to the data fragmentation problem, when used for branching in a certain node, even though the strategy of merging branches is applied.

In our case we have to deal with several attributes with many values. For instance, this is the case of the 'word-form' attribute, since there are ambiguity classes which contains many different words (up to 850 in the worst case). Other examples will appear further on when dealing with unknown words, or when introducing lexical attributes referring to words or sequences of tags/words.

These attributes are handled by dynamically adjusting the number of values to the N most frequent, and joining the rest in a new *otherwise* value. 'Dynamically' means here that the set of values is recalculated at each node of the tree, even if the attribute has been used before. In this way, a dynamic attribute may be used many times along the tree-induction process involving different values at each time.

In the case of the basic set of attributes, only the word-form is *dynamic*, and the maximum number of values N is fixed to the tagset size in order to have homogeneous attributes with the same number of values. The other attributes will be referred to as *static*, by opposition. This is the notation used in the descriptive tables of attributes of this chapter.

2.2.5. *Missing Values.* In our domain we do not consider inter-sentential relations so the local context is restricted to be in the same sentence as the tagset word (i.e. it is not allowed to cross any end-of-sentence marker).

Due to this restriction, some examples may contain non-valid values for some contextual attributes. For instance, if the target word to be disambiguated is the first of the sentence, then the left context attributes contain values that should be considered uninformative. Otherwise they would introduce a counterproductive noise in the training set.

A usual way for handling training examples with missing values for some attributes is to suppose a probability distribution over all possible values for those attributes, and to complete the missing values with these probability distributions. When completing an example, the probability distribution for a certain attribute for which the example is undefined may be uniform, or estimated from the values appearing in the other training examples. In the second case there are still two possibilities: to estimate probabilities from unconditional counts of the different

values, or from the counts conditioned to the class label of the target example. This approach implies that during the tree induction the so completed examples will be considered as many weighted examples regarding the incomplete attribute: One for each possible value.

However in our case the situation is a bit different because the value is not actually unknown but it is uninformative, so we do not have to guess the possibilities. The way we proceeded is to introduce a new value NULL standing for non relevant information (interpreted as: 'there is a sentence boundary in between the target word and this attribute') and do not change the learning algorithm at all. These NULL values are assigned by preprocessing.

2.2.6. *Probability Estimates.* As previously said, we will deal with statistical decision trees instead of common decision trees. This is done by storing, for each node of the tree, the information about the number of examples relative to each class. These figures are then used to estimate the conditional probabilities.

We tested alternative ways of estimating probabilities from the relative frequencies present in leaf nodes. From more to less simple they are: maximum likelihood estimates (MLE), *discounting and redistribution* smoothing methods [**YB97**], and m-estimates [**Ces90**].

Given a concrete node of a decision tree with its associated set of examples X, the probability of a certain tag t is straightforwardly estimated by MLE as the proportion of examples that have tag t over the total number of examples, that is:

$$\hat{p}(t|X) = \frac{f(t|X)}{\|X\|}.$$

In order to smooth the MLE probability estimates, one can consider the following general formulation for discounting some probability mass from frequently seen events to redistribute it among the less frequent events:

$$\hat{p}(t|X) = \frac{f(t|X) + \lambda}{\|X\| + \lambda K},$$

where K is the number of possible tags, and λ is a positive real value. When $\lambda = 1$ the above formula is known as Laplaces's law of succession. As some authors observe, the appropriate value for λ in general problems of language modelling is significantly lower than 1. We have set this value to $\lambda = (K-1)/K$, which depends on the number of possible tags. It starts in 0.5 for two-tags ambiguity classes and increases asymptotically to 1 (Laplace's law) as the number of possible tags increase.

Another possibility of performing smoothing is to use m-estimates [**Ces90**], which represents a kind of back–off interpolation using the *a priori* probabilities of the ambiguity class (stored in the root node of the tree). Its formulation is the following:

$$\hat{p}(t|X) = \frac{f(t|X) + m\,\hat{p}(t)}{\|X\| + m},$$

where m is the parameter (that must be experimentally set) which regulates the degree in which prior probabilities affects the estimates.

From the three possibilities we observed better results applying smoothing, but with no significant differences between the discounting method and m-estimates. Therefore, we use the first which is simpler[10].

When classifying a new example using a decision trees it may be the case that a certain internal node tests a concrete attribute for which it has no edge to follow through the value of the example (this means that this type of examples was not present in the training set). In this situation, the class probability of this internal node is outputted as the "default" result[11].

2.2.7. *Subsampling.* We had some problems with the computational effort involved in: (1) The testing experiments, in which computationally expensive cross-validation experiments were performed; and (2) The RELIEFF-IG function, which is not as efficient as the other because it uses a k-nearest neighbour algorithm which is computationally expensive, unless sophisticated indexing techniques are used[12].

In order to speed-up the learning algorithm on the large training sets (the biggest contains around 38,000 examples) we used a simple strategy of selecting a moderately big subsample of the whole training set to perform the selection of the best split at the shallowest nodes of the tree. Subsamples are obtained by randomly selecting a percentage of the set of examples belonging to the current node. This percentage is 100% if the number of examples is lower than a minimum (say 5,000 examples) and linearly decreases up to 50% for the case of the maximum number of examples in the domain (say 38,000 examples).

This is certainly a naive way of collecting subsamples (see, for instance, the work [**MCR92**], which proposes a method for dynamically choosing the adequate sample based on how difficult the decision is at each node of the tree), but due the high redundancy in our domain, the performance observed by applying this simplifying technique was almost the same as those obtained with full set of examples.

2.3. An Example. We present a real example of a decision tree branch learned for the preposition–adverb ambiguity class (`IN-RB`) which has a clear linguistic interpretation[13].

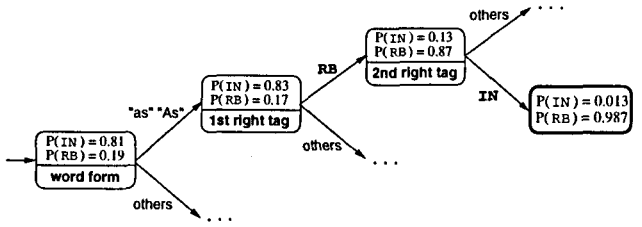

FIGURE 3. Example of a decision tree branch

[10] We used smoothed estimates not only when classifying with the acquired decision trees, but whenever that probability estimation was required from frequency counts. This includes: the pruning algorithm, several functions for feature selection, lexical probabilities estimated from training corpora, etc.

[11] Kononenko et al. [**KŠR95**] suggest to use the probability estimates resulting from the naive Bayes calculation over all the attributes involved in the partial path from the root to the current intermediate node. We also tested this proposal with negative results.

[12] Apart from the computational cost of the involved algorithms, another factor affecting efficiency was the initial LISP-based implementation of the learning algorithms.

[13] See appendix D for a complete listing of the decision tree.

We can observe in figure 3 that each node in the path from the root to the leaf contains a question on a concrete attribute and a probability distribution. In the root it is the prior probability distribution of the class. In the other nodes it represents the probability distribution conditioned to the answers to the questions preceding the node. For example the second node says that the word 'as' is more commonly a preposition than an adverb, but the leaf says that the word 'as' is almost certainly an adverb when it occurs immediately before another adverb and a preposition. This is the case of the collocations: "as much as", "as well as", "as soon as", etc., which are systematically tagged adverb–adverb–preposition in the WSJ corpus.

3. Model Evaluation

This section is devoted to the evaluation of the appropriateness of statistical decision trees for modelling POS ambiguities. It starts by describing the main characteristics of the corpus of application in sections 3.1 and 3.2, and then, the results of the evaluation are presented in section 3.3.

3.1. The Wall Street Journal Annotated Corpus. We have used a portion of 1,170,000 words (1,17Mw) of the WSJ, tagged according to the Penn Treebank tagset[14], from which 1,12Mw were randomly selected to form the training corpus. The remaining 50,000 words were used for testing.

The training corpus has been used to create a word form lexicon —of 49,206 entries— with the associated lexical probabilities for each word. These probabilities are estimated simply by counting the number of times each word appears in the corpus with each different tag and applying smoothing as described in 2.2.6.

Due to errors in corpus annotation, the resulting lexicon has a certain amount of noise. In order to partially reduce this noise, the lexicon has been filtered by manually checking the entries for the most frequent 200 words in the corpus —note that the 200 most frequent words in the corpus represent over half of it. For instance the original lexicon entry (numbers indicate frequencies in the training corpus) for the very common word "the" was:

```
<the> CD:1 DT:47715 JJ:7 NN:1 NNP:6 VBP:1,
```

since it appears in the corpus with the six different tags: `CD` (cardinal), `DT` (determiner), `JJ` (adjective), `NN` (noun), `NNP` (proper noun) and `VBP` (verb–personal form). Nevertheless, it is obvious that the only correct reading for the word "the" is determiner. Although in most cases the checking was done by filtering out wrong tags, there are some entries for which some missing readings were also added.

As noted by Ratnaparkhi [**Rat96**], the WSJ corpus also have some inconsistencies due to not completely coincident criteria applied by different annotators. He observed that tag distributions for concrete words change as a function of the article in which occur, and that this change has a big correlation with the concrete annotator. This fact may negatively affect the use of lexical attributes (i.e. attributes that refer to concrete words) to perform learning (see section 4.2 for the application of such features). Nevertheless, this is an issue that we will not address in this work.

[14]The Penn Treebank tagset contains 45 tags. A full listing can be found in appendix C.

Taking the resulting filtered lexicon as a reference, the ambiguity figures about the training corpus are: About 34.05% of the words are ambiguous, with 2.40 tags per word on average (over all words, the ambiguity ratio is 1.48 tags/word)[15].

Regarding ambiguity sets, the training corpus contains 243 different ambiguity classes. The number of tags per ambiguity class range from 2 to 6, and their number of examples range from a few dozens to several thousands (with a maximum of 38,112 examples for the preposition–adverb–particle ambiguity). It is noticeable that only the 37 most frequent ambiguity classes concentrate up to 90% of the ambiguous occurrences of the training corpus. Tables 3 and 4 contain more information about these issues.

	2-tags	3-tags	4-tags	5-tags	6-tags
# classes	103	90	35	12	3

TABLE 3. Number of ambiguity classes containing n tags

	50%	60%	70%	80%	90%	95%	99%	100%
# classes	8	11	14	19	37	58	113	243

TABLE 4. Number of ambiguity classes that cover the x% of the ambiguous words of the training corpus

The already collected information provides a simple heuristic function for disambiguating, which consists of choosing for each word its most probable tag, taking as a reference the tag probability distribution corresponding either to the word (called lexical probability), or to the ambiguity class which the word belongs to. We will call these naive taggers MFT, standing for Most–Frequent–Tag tagger. The two versions will be denoted by MFT_1 and MFT_2, corresponding to the more general ambiguity–class approach, and to the lexical probability approach, respectively. Note that such taggers do not use any contextual information, but only the tag frequencies of isolated words or ambiguity classes.

Figure 4 shows the performance of the baseline MFT taggers on the WSJ domain for different sizes of the training corpus (first plot). The second plot is the same but it includes a lower bound result of random selection. The reported figures refer to ambiguous words.

Some conclusions can be extracted from those plots:

- Simple non–contextual information establishes a bench mark for POS tagging accuracy which is far better than random selection.
- Both MFT heuristics perform equally on very small training sets, and in fact MFT_1 has a higher starting point than MFT_2. This is not strange since general estimators work better than specific when sample data is scarce[16]. However, MFT_2 compares favourably to MFT_1 for training sets bigger than 25,000 words (25Kw), and it becomes much better for training corpus bigger

[15]The ambiguity figures before the manual processing were slightly higher: 36.5% of ambiguous words, with an ambiguity ratio of 2.45 tags/word over the ambiguous words, 1.50 overall.

[16]See, for instance, the the work by Tzoukermann et al. [**TRG97**] in which ambiguity classes are used instead of lexical counts to alleviate the problem of data sparseness in tagging French.

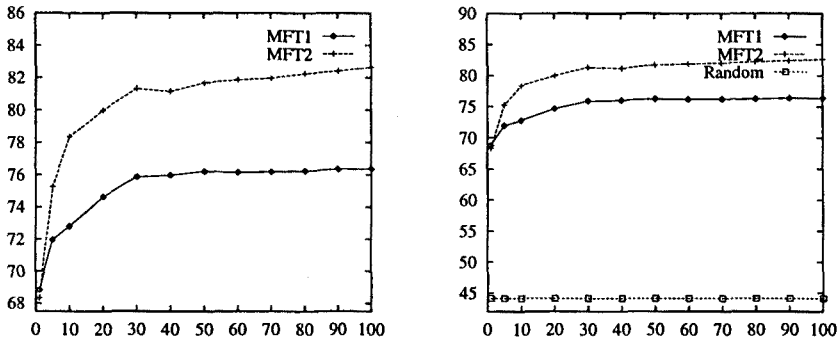

FIGURE 4. Performance of the MFT_1 and MFT_2 heuristics related to the training set size. In both plots X-axis represents the number of training examples (divided by 10,000) and the Y-axis the percentage of accuracy on ambiguous words

than 100Kw. This is due to the fact that English is not a morphologically rich language and so the number of different word forms is quite limited. Consequently, the lexical probabilities can be reasonably estimated even with a small number of examples.

- For training corpora bigger than 400Kw, the accuracy of MFT_1 remains unvarying at about 76% (which is clearly an upper bound for this heuristic), while the accuracy of MFT_2 is between 81–83%. Although the final tendency for MFT_2 is still slightly increasing, it is not reasonable to think that it could be significantly raised simply by adding more training examples to better estimate lexical probabilities.

3.2. Domain of Evaluation and Methodology. We selected the set of the twelve most significant ambiguity classes for testing purposes. The importance of each ambiguity class was measured in terms of the number of available examples, and the difficulty to acquire accurate classifiers, since very representative and/or difficult classes will greatly influence the final tagger performance. In particular, only these 12 data sets cover about 58% of the ambiguous word occurrences in the corpus (the total number of ambiguous words is 374,622), and they directly take part in about 67% of the MFT tagging errors. The information about these twelve classes is depicted in table 5. Base1 and Base2 stand for the baseline estimated error rates of the MFT heuristics, while the Test column presents the real number and percentage of errors commited by MFT_2 on the 50Kw test set.

Table 6 contains example words belonging to each ambiguity class, as well as the count of different word forms of each class appearing in the whole training set.

The comparisons between alternative methods that will be reported in the next section (and also in the rest of the document) are generally done using a statistical test of significance, based on 10-fold cross validation. When the repetition of 10 experiments was not feasible, we applied a test for the difference of two proportions over the results of a single execution (see [**Die98b**] for a description of these and other approximate statistical tests commonly used for comparing supervised classification learning algorithms).

	Amb. Class	Examples	Base1	Base2	Test	%Err
1	IN-RB-RP	38,112 (10.17)	13.26	9.40	210 (11.70)	7.56
2	VBD-VBN	28,417 (7.59)	33.56	19.90	267 (18.98)	9.61
3	NN-VB-VBP	26,970 (7.20)	29.78	20.22	255 (20.10)	9.18
4	VB-VBP	19,718 (5.26)	35.48	23.16	226 (24.42)	8.13
5	JJ-NN	18,824 (5.02)	40.48	16.91	144 (16.54)	5.18
6	NNS-VBZ	16,928 (4.52)	18.69	11.21	81 (11.40)	2.91
7	NN-VB	16,239 (4.33)	10.51	8.95	67 (9.10)	2.41
8	JJ-VBD-VBN	12,607 (3.37)	44.96	33.27	180 (31.64)	6.48
9	NN-VBG	10,560 (2.82)	49.40	19.77	116 (21.68)	4.17
10	JJ-NNP	9,650 (2.58)	25.32	14.04	68 (13.91)	2.45
11	JJ-RB	9,628 (2.57)	35.58	15.60	73 (16.49)	2.63
12	DT-IN-RB-WDT	9,237 (2.48)	39.36	39.36	187 (40.34)	6.73
Total		216,890 (57.90)	—	—	1,874	67.44

TABLE 5. Information about twelve selected significant ambiguity classes. Examples: Number of examples and, in parenthesis, percentage with respect to the total number of training examples; Base1: Error rate of the MFT_1 heuristic ('ambiguity class' level); Base2: Error rate of the MFT_2 heuristic ('word' level); Test: Number of errors and error rate of the MFT_2 heuristic on the test set; and %Err: percentage of the total number of errors on the test set

	Amb. Class	Example words	#Ws
1	IN-RB-RP	*up, in, about, on,...*	14
2	VBD-VBN	*named, said, stopped, heard, studied,...*	845
3	NN-VB-VBP	*show, rate, study, support, process, question,...*	301
4	VB-VBP	*make, filter, result, use, have,...*	422
5	JJ-NN	*common, standard, material, blue, human,...*	676
6	NNS-VBZ	*makes, filters, results, uses,...*	392
7	NN-VB	*group, increase, interest, name,...*	379
8	JJ-VBD-VBN	*reported, used, expected, classified, exposed,...*	268
9	NN-VBG	*publishing, talking, making, finding, holding,...*	365
10	JJ-NNP	*New, National, Western, Financial, Pacific,...*	265
11	JJ-RB	*likely, very, early, far, virtually, only,...*	90
12	DT-IN-RB-WDT	*that*	1

TABLE 6. Some example words from the twelve ambiguity classes of reference. #Ws stands for the number of different word forms belonging to that class

The measures employed to test the goodness of the induced decision trees were the usual: classification accuracy (or, conversely, the error rate) exhibited on new examples, and size (number of nodes), despite more sophisticated measures can be considered. This is the case of using the *average information score* [**KŠR95**] to complement classification accuracy, or considering *optimally pruned trees* (see [**EMS97**]) when evaluating pruning performance of some concrete method. We have not used them because our aim was not to perform a thorough comparison between methods in order to strongly conclude that some approach outperform others, but we only wanted to properly tune our learning algorithm.

3.3. Results. Due to space limitations we are not able to reproduce the whole set of experiments performed to properly tune the learning algorithm. Nevertheless, we will report information about the most important ones, which are: (1) Testing the appropriateness of pruning and stopping criteria; (2) Evaluating the benefits coming from the branch–merging strategy; and (3) Selecting, from a set of 10, the most adequate function for feature selection;

Regarding the model evaluation proper, apart from quantifying the accuracy improvements when testing the classifiers on the test bench data sets, we also evaluate the appropriateness of the tree–modelling for a posterior use as a statistical module in a more complex disambiguation algorithm (e.g. RTT, STT, and RELAX taggers introduced in chapter 4). Since trees will be used to predict tag probabilities conditional to some particular relevant contexts, we focus in class probability estimates.

3.3.1. *Branching Strategy and Pruning.* Table 7 shows the results obtained when introducing "branch merging" and CART post–pruning features to our basic tree–induction algorithm.

The results are presented in three steps for each of the twelve data sets of reference. The 'Base' column stands for the basic learning algorithm, which is run with the basic set of six attributes of table 1, and the RLM criterion for attribute selection. The '+Merging' stands for the results obtained when introducing branch–merging in the basic algorithm, and, finally, the '+Pruning' column shows results when adding post–pruning to the previous algorithm (i.e. including also branch–merging).

The figures in table 7 are the average results of a ten fold cross–validation experiment, thus in each fold the tree is learned from the 90% of the whole training set and tested on the remaining 10%.

		Base		+Merging		+Pruning	
	Amb. Class	%Err	Nodes	%Err	Nodes	%Err	Nodes
1	IN-RB-RP	9.00	2,062	8.81	1,430	8.55	147
2	VBD-VBN	6.24	3,326	6.26	1,148	6.12	820
3	NN-VB-VBP	3.83	1,295	3.71	614	3.84	502
4	VB-VBP	4.22	1,133	4.10	528	4.18	320
5	JJ-NN	15.46	3,598	14.87	1,772	14.75	702
6	NNS-VBZ	6.08	1,465	5.75	498	5.72	450
7	NN-VB	1.30	430	1.36	211	1.42	199
8	JJ-VBD-VBN	18.15	3,301	18.66	1,538	18.12	751
9	NN-VBG	13.66	2,833	14.16	1,151	13.54	539
10	JJ-NNP	5.15	643	4.81	295	4.97	270
11	JJ-RB	10.88	1,144	9.97	887	10.57	712
12	DT-IN-RB-WDT	7.96	865	7.95	602	7.80	303
Average/Total		8.49	22,095	8.36	10,674	8.30	5,715

TABLE 7. Results of testing the branch–merging and post–pruning techniques

We obtain the following conclusions:

- The branch–merging strategy significantly reduces the tree sizes. Globally, we jump from 22,095 to 10,674 nodes which is a reduction of almost 52%. This substantial reduction is achieved with no loss in accuracy, since the average error rate is even slightly diminished (not significant).

- The CART post-pruning technique allows a further decrease of the size of the trees: 46.4% reduction. Again, the error rate is slightly decreased. This change is not significant but it allows to say that the induced trees are at least as accurate as those learned in the absence of pruning.

We have seen that both techniques are very effective to substantially reduce the size of the induced trees, and that these trees retain (if no increase) their generalization ability. Therefore, they will be incorporated to the basic algorithm from now on.

3.3.2. *Attribute Selection Functions.* We added three simple functions for feature selection to those presented in subsection 2.2.1 in order to serve as increasingly better bench mark references. They are the following:

- *Random selection.*
- *Minimum number of errors* (MNE). That is, choosing for branching the attribute which induces a partition with minimum number of errors. The number of errors in a certain node is simply the MLE estimate of the error rate, i.e. $N-M$ where N is the total number of examples and M is the number of examples of the most frequent tag.
- *Global Scoring* (GS-IG). In this case, the criterion consists of determining an initial score for each attribute (based on the Information Gain measure calculated over all examples) and selecting at each step the best scored of the remaining attributes.

The following table 8 shows the error rates obtained by using the ten different functions for feature selection on the twelve ambiguity classes of reference. The notation used to describe each function is that introduced in subsection 2.2.1, except for the last column 'RF-IG' which stands for RELIEFF-IG. Again, figures reported are calculated by averaging the results of the ten folds.

The main conclusions that can be extracted are the following:

- The seven basic attribute selection criteria perform roughly equal on average (their relative differences lead to no statistical significance). The best is RLM with an average error of 8.24% on the 12 data sets, while the worst is Gini diversity index reporting 8.69%. RLM shows a slightly better stability and obtains the best figure in several ambiguity classes (additionally, its performance is within the best in all cases), therefore it will be the preferred function[17].
- Regarding the baseline functions: (1) Random selection performs worse than any other function; (2) GS-IG performs worse than any other function (except Random selection). Observe that this is due to a great variance of GS-IG that makes it to be the best on some data sets (`IN-RB-RP` and `DT-IN-RB-WDT`) but the worst on many others (`JJ-NN`, `JJ-VBD-VBN`, `NN-VBG`, etc.); and (3) MNE performs only slightly worse (not significant) than the seven basic functions. This is surprising since some authors [**Mur95**] observe that MNE is very unstable and has many problems of tuning.

[17] An empirical study by López de Mántaras [**LCG96**] states that RLM function lead to statistically smaller trees than Quinlan's Gain Ratio and that it is not favourably biased to attributes with many values

	Amb. Class	Random	MNE	GS-IG	RLM	IG
1	IN-RB-RP	11.21	9.15	8.59	8.65	8.61
2	VBD-VBN	16.05	6.57	13.80	6.17	6.14
3	NN-VB-VBP	12.12	3.34	3.83	3.73	3.26
4	VB-VBP	13.52	4.67	4.72	4.08	4.27
5	JJ-NN	21.43	15.40	28.07	14.78	14.22
6	NNS-VBZ	7.50	6.50	7.32	5.65	6.14
7	NN-VB	5.44	1.77	1.50	1.36	1.37
8	JJ-VBD-VBN	32.40	19.19	24.45	18.43	18.05
9	NN-VBG	19.48	14.12	21.70	13.54	13.65
10	JJ-NNP	18.33	5.73	7.22	4.81	5.50
11	JJ-RB	16.04	10.40	11.34	9.94	10.31
12	DT-IN-RB-WDT	33.31	8.56	7.01	7.93	8.19
Average		17.24	8.78	11.63	8.24	8.31

	Amb. Class	GR	Gini	χ^2	Tau	RF-IG
1	IN-RB-RP	8.70	9.08	8.64	8.93	8.99
2	VBD-VBN	6.49	6.45	6.37	6.10	6.27
3	NN-VB-VBP	3.38	3.71	3.59	3.95	3.19
4	VB-VBP	4.38	4.72	4.55	4.67	4.09
5	JJ-NN	14.75	15.52	14.40	14.29	15.30
6	NNS-VBZ	6.47	5.56	5.50	5.69	5.39
7	NN-VB	1.44	1.37	1.40	1.35	1.29
8	JJ-VBD-VBN	18.05	19.63	20.07	18.84	18.71
9	NN-VBG	13.75	13.85	12.92	13.54	13.44
10	JJ-NNP	5.73	5.84	5.96	5.27	6.62
11	JJ-RB	11.57	9.97	10.00	9.51	9.56
12	DT-IN-RB-WDT	8.19	8.55	8.79	8.07	7.95
Average		8.58	8.69	8.52	8.35	8.40

TABLE 8. Comparative results using different functions for feature selection

- We have seven functions, based on different principles, which are similarly accurate[18]. This fact will allow us to construct useful ensembles of decision trees to reduce the combined error (this issue will be addressed in chapter 6).

3.3.3. *Probability Estimation.* One way to easily evaluate the quality of the class–probability estimates given by a classifier is to calculate a *rejection curve* [**DB95**]. That is to plot a curve showing the percentage of correctly classified test cases whose *confidence level* exceeds a given value θ, for increasing values of this parameter. In the case of statistical decision trees this confidence level can be straightforwardly computed from the class probabilities given by leaves of the trees. In our case we calculate the confidence level as the difference in probability between the two most probable cases (if this difference is large, then the chosen class is clearly much better than the others; if the difference is small, then the chosen class is nearly tied with another class). A rejection curve that increases smoothly, indicates that the

[18]When used within tagging algorithms (see chapter 4), the trees acquired using all seven basic functions clearly improve the results of the MFT heuristics reported in table 5. This improvement is substantial in all data sets, except in two cases: IN-RB-RP and JJ-NN, in which only a moderate decreasing of error rate is observed.

confidence level produced by the classifier can be transformed into an accurate probability measurement, while flat or decreasing segments of the curve indicate cases where the confidence estimate of the learning algorithm is unrelated or inversely related to the actual performance of the algorithm.

Figure 5 depicts the rejection curves corresponding to the decision trees acquired for the twelve reference ambiguity classes. For calculating the curves the parameter θ was increased from 0 to 1 with a step of 0.01 (corresponding to 1%).

We list below the main conclusions:

- The rejection curves for most cases increase fairly smoothing, with very few abrupt, flat, or decreasing segments, indicating that the acquired decision tress provide good confidence estimates.
- We observe that some curves end much before rejecting 100% of the examples (this is particularly noticeable in the `NN-VB` and `IN-RB-RP` ambiguity classes). This situation occurs when the final increment in θ causes all examples to be rejected (call θ_{max} the last value of θ for which the rejection is lower than 100%), and indicates that there is a big amount of examples for which the algorithm gives "equal" confident estimates (between θ_{max} and $\theta_{max} + 0.1$). In these particular domains this situation is explained because there exist very strong indicators for some concrete readings which concentrates many examples (e.g. after a a determiner or an adjective the "verb" reading is prohibited in the `NN-VB` domain, and this is the contextual situation of almost 50% of the training examples of that ambiguity class) which are all classified with a very high confidence.

4. Extending the Basic Model

We have extended the basic algorithm following three different directions: (1) Considering the construction of specific decision trees for dealing with *unknown words*[19]; (2) Enriching the set of features that describe the training examples by adding lexical patterns; and (3) Specifically addressing the problem of small ambiguity classes.

These new features are described in the following sections.

4.1. Dealing with Unknown Words. *Unknown* words are those words not present in the lexicon (i.e. in our case, the words not present in the training corpus), for which we do not know in advance which are their possible tags (i.e., to which ambiguity class they belong).

One might think that it is not necessary to consider the possibility of unknown words, because a robust morphological analyzer that would supply the set of possible tags for each word form can be assumed. However, this is not the most realistic scenario. First, a morphological analyzer is not always present (due to the morphological simplicity of the language treated, the existence of some efficiency requirements, or simply the lack of resources). Second, if it is available, it very probably has a certain error rate that makes it necessary to consider the noise it introduces. Therefore, it seems clear that we have to deal with unknown words in order to obtain more realistic figures about the real performance of the proposed tagging approach.

[19]The precise meaning of the term 'unknown words' is explained in section 4.1.

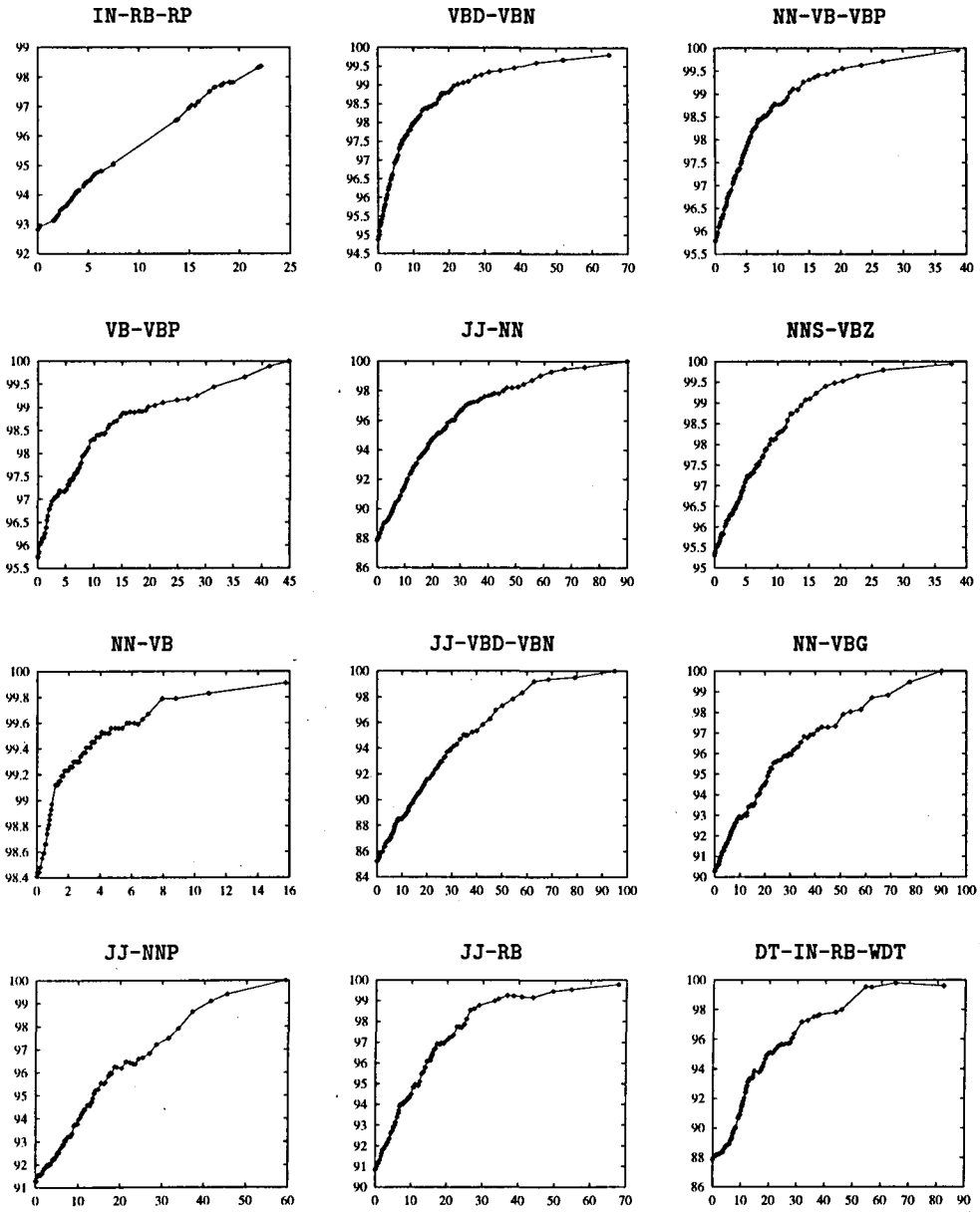

FIGURE 5. Rejection curves for the twelve ambiguity classes. In all of them, X–axis represents the percentage of rejection and the Y–axis the percentage of accuracy

There are several approaches for dealing with unknown words (see section 2.5.2 of chapter 2 for a brief survey). In our case, we consider unknown words as words belonging to a new ambiguity class containing all possible tags corresponding to open categories (i.e. noun, proper noun, verb, adjective, adverb, cardinal, etc.).

The number of candidate tags come to 20 in the Penn Treebank tagset, so we state a classification problem with 20 different output classes. Therefore, unknown words are treated exactly in the same way as the other ambiguity classes, except for the attributes describing the examples.

We have used very simple information about the orthography and the context to acquire decision trees for the **unknown-word** ambiguity class. More particularly, from an initial set of 17 potential attributes, we have empirically decided the most relevant (using the RELIEFF-IG function to score them by relevance, and testing some combinations of attributes), which turned out to be the following ten: (1) In reference to word form: the first letter, the last three letters, and other four binary–valued attributes accounting for capitalization, whether the word is a multi–word or not, and for the existence of some numeric characters in the word. (2) In reference to the context: only the preceding and the following POS tags.

This set of attributes is fully described in table 9.

	Attribute	Values	Type
1	tag(-1)	Any tag in the Penn Treebank tagset	S
2	tag(+1)	"	S
3	char(1)	Any printable ASCII character	D
4	char(n)	"	D
5	char($n-1$)	"	D
6	char($n-2$)	"	D
7	capitalized?	{yes,no}	S
8	other_capital_letters?	"	S
9	multi-word?	"	S
10	has_numeric_characters?	"	S

TABLE 9. Set of basic attributes for dealing with unknown words

4.1.1. *Evaluation.* We have estimated the relative proportions in which the 20 tags of the **unknown--word** class appear naturally in the WSJ as unknown words, and we have collected the examples from the training corpus according to these proportions. The most frequent tag, NNP (proper noun), represents almost 30% of the sample. This fact establishes a lower bound for accuracy of 30% in this domain (i.e. the performance that a MFT tagger would obtain on unknown words).

Table 10 shows the generalization performance of the trees learned from training sets of increasing sizes up to 50Kw words ('Dtrees' columns). In order to compare these figures with a close approach we have simulated IGTREE system [**DZBG96**] and we have tested its performance exactly under the same conditions as ours.

IGTREE system is the core of a memory-based POS tagger which stores in memory the whole set of training examples and then predicts the part of speech tags for new words in particular contexts by extrapolation from the most similar cases held in memory (k–nearest neighbour retrieval algorithm). See section 3.2.5 of chapter 2 for more details. The main connection point to the work presented here is that huge example bases are indexed using a tree–based formalism, and that the retrieval algorithm is performed by using the generated trees as classifiers. Additionally, these trees are constructed on the base of a previous weight assignment for attributes (contextual and orthographic attributes used for disambiguating are very similar to ours) using Quinlan's Gain Ratio [**Qui86**].

Note that the final pruning step applied by IGTREE to increase the compression factor even more has also been implemented in our version. The results of IGTREE are also included in table 10. Figures 6 and 7 contain the plots corresponding to the same results.

	DTrees		IGTREE	
#Exs.	Acc.	#Nodes	Acc.	#Nodes
2,000	77.53%	224	70.36%	627
5,000	80.90%	520	76.33%	1,438
10,000	83.30%	1,112	79.18%	2,664
20,000	85.82%	1,644	82.30%	4,783
30,000	87.32%	2,476	85.11%	6,477
40,000	88.00%	2,735	86.78%	8,086
50,000	88.12%	4,056	87.14%	9,554

TABLE 10. Generalization performance of the trees for unknown words

Observe that our system produces better quality trees than those of IGTREE. On the one hand, we see in figure 6 that the generalization performance is better in all cases. On the other hand, figure 7 seems to indicate that the growing factor in the number of nodes is linear in both cases, but clearly lower in ours.

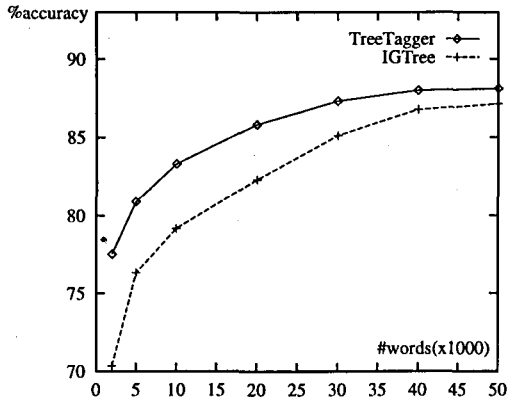

FIGURE 6. Accuracy vs. training set size for unknown words

Important aspects contributing to the lower size are the merging of attribute values and the post pruning process applied in our algorithm, which allow a relevant reduction of the tree size without loss in accuracy.

The better performance is probably due to the fact that IGTREEs are not actually decision trees (in the sense of trees acquired by a supervised algorithm of top–down induction, that use a certain *attribute selection function* to decide at *each step* which is the attribute that best contributes to discriminate between the current set of examples), but only a tree–based compression of a base of examples inside a kind of weighted nearest–neighbor retrieval algorithm. The representation and the weight assignment for attributes allows us to think of IGTREEs as the decision trees that would be obtained by applying the usual top–down induction algorithm

with a simple attribute selection function consisting of making a previous unique ranking of attributes using Quinlan's Gain Ratio over all examples and later selecting the attributes according to this ordering, i.e. exactly the GS-IG function tested in section 3.3.2. The experimental results of that section show that it is slightly better to reconsider the selection of attributes at each step than to decide on an *a priori* fixed order.

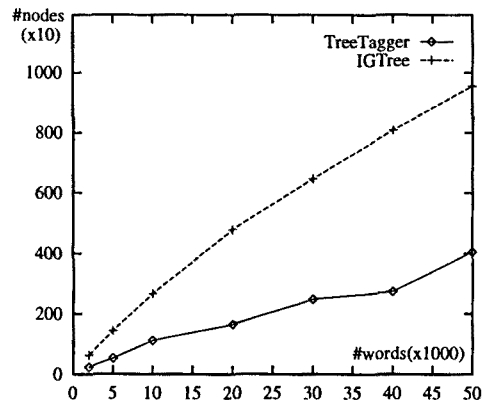

FIGURE 7. Number of nodes of the trees for unknown words

Of course, these conclusions have to be taken in the domain of small training sets, since the same plot in figure 6 suggests that the difference between the two methods decreases as the training set size increases. Using bigger corpora for training might improve performance significantly. For instance, [**DZBG96**] report an accuracy rate of 90.6% on unknown words when training with the whole WSJ (2 million words). So our results can be considered better than theirs in the sense that our system needs less resources for achieving the same performance.

4.2. Enriching Features. This extension could be done in many directions, e.g. by adding morphological, syntactic, or semantic features, by considering long-distance or inter-sentential relations, etc. being, of course, conditioned by the availability of the required extra information —something that is dubious in most cases, because it corresponds to higher levels of language understanding in the usual pipelined approach to NLP.

In this section we will report the results of some experiments performed around the addition of new attributes, which are of different levels of specificity, and have in common that need neither syntactic nor semantic knowledge about the domain. Table 11 presents a full listing of the enlarged set of features.

The new set of attributes incorporates lexical information about words appearing in the local context of the target word (rows 11–15), and the ambiguity classes of the same words (rows 30–34). In this way, we consider information about the surrounding words at three different levels of specificity: word form, POS tag, and ambiguity class.

This model also includes some features, which are very similar to Brill's lexical patterns [**Bri95a**], for capturing collocational information. Such features are obtained by composition of the already described single attributes and they are sequences of contiguous words and/or POS tags (up to three items).

		Attribute	Values	Type
A	1	tag(-3)	Any tag in the Penn Treebank tagset	S
	2	tag(-2)	"	S
	3	tag(-1)	"	S
	4	tag(+1)	"	S
	5	tag(+2)	"	S
	6	word(0)	Any word of the ambiguity class	D
B	7	tag(-1) × tag(+1)	pairs of tags	D
	8	tag(-2) × tag(-1)	"	D
	9	tag(+1) × tag(+2)	"	D
	10	tag(-3) × tag(-2) × tag(-1)	trios of tags	D
C	11	word(-3)	Any plausible neighbour word of word(0)	D
	12	word(-2)	"	D
	13	word(-1)	"	D
	14	word(+1)	"	D
	15	word(+2)	"	D
D	16	tag(-1) × word(0)	pairs (tag,word)	D
	17	word(0) × tag(+1)	"	D
	18	tag(-2) × tag(-1) × word(0)	trios (tag,tag,word)	D
	19	word(0) × tag(+1) × tag(+2)	trios (word,tag,tag)	D
	20	tag(-1) × word(0) × tag(+1)	trios (tag,word,tag)	D
E	21	word(-1) × tag(-1)	pairs (word,tag)	D
	22	word(+1) × tag(+1)	"	D
	23	word(-1) × tag(-1) × word(0)	trios (word,tag,word)	D
	24	word(0) × word(+1) × tag(+1)	trios (word,word,tag)	D
F	25	word(-2) × word(-1)	pairs (word,word)	D
	26	word(+1) × word(+2)	"	D
	27	word(-1) × word(0)	"	D
	28	word(0) × word(+1)	"	D
	29	word(-1) × word(0) × word(+1)	trios (word,word,word)	D
G	30	Aclass(-3)	Any ambiguity class	D
	31	Aclass(-2)	"	D
	32	Aclass(-1)	"	D
	33	Aclass(+1)	"	D
	34	Aclass(+2)	"	D
H	35	Cap(word(-1))?	{yes,no}	S
	36	Cap(word(0))?	"	S
	37	Cap(word(+1))?	"	S

TABLE 11. Extended set of attributes

Finally, and also inspired on the original paper describing Brill's tagger [**Bri92**], we add three binary-valued attributes describing if the surrounding words are capitalized or not (rows 35–37).

Regarding the evaluation, we are not only interested on how does the basic set of attributes perform compared to the enlarged set, but also on the particular behaviour of each type of attribute. For this reason, we grouped them, according to their degree of specificity, in eight types of attributes (A to H groups in the previous table): POS tags, word forms, ambiguity classes, pairs of tags, pairs of words, etc.

Most of these groups are too specific (or contain too few features) for performing well alone, thus they were complemented with two basic features, namely tag(-1) and tag(+1), before testing them. The results obtained are presented in table 12. The first column corresponds to the basic set of attributes, while the last column corresponds to the full set of 37 attributes. Columns in between show the test of the different groups of attributes ("+X" stands for the addition of the two complementary attributes).

	Amb. Class	A	B+X	C+X	D+X	E+X
1	IN-RB-RP	8.64%	8.78%	8.15%	9.13%	8.32%
2	VBD-VBN	6.27%	6.41%	**5.71%**	7.65%	6.91%
3	NN-VB-VBP	3.71%	3.78%	**3.54%**	4.36%	3.95%
4	VB-VBP	4.10%	**3.99%**	5.50%	5.28%	6.01%
5	JJ-NN	14.87%	13.58%	14.70%	14.64%	14.17%
6	NNS-VBZ	5.65%	5.81%	5.75%	5.89%	5.49%
7	NN-VB	1.37%	1.29%	1.33%	1.16%	1.27%
8	JJ-VBD-VBN	18.67%	18.57%	19.98%	18.84%	19.37%
9	NN-VBG	14.07%	13.54%	**12.39%**	13.75%	11.87%
10	JJ-NNP	4.81%	5.27%	5.50%	5.61%	5.84%
11	JJ-RB	9.97%	9.85%	10.08%	9.94%	9.62%
12	DT-IN-RB-WDT	7.96%	7.84%	8.31%	8.79%	8.55%
Average error-rate		8.34%	**8.22%**	8.41%	8.75%	8.45%
num-nodes		480.1	488.0	519.6	504.6	524.6

	Amb. Class	F+X	G+X	H+A	All
1	IN-RB-RP	**8.06%**	8.35%	8.84%	8.64%
2	VBD-VBN	7.72%	6.60%	6.10%	5.84%
3	NN-VB-VBP	4.61%	3.67%	3.70%	**3.54%**
4	VB-VBP	4.15%	4.83%	4.00%	4.38%
5	JJ-NN	**13.52%**	14.46%	14.80%	15.05%
6	NNS-VBZ	5.48%	**5.40%**	5.77%	6.01%
7	NN-VB	**1.09%**	1.44%	1.36%	1.44%
8	JJ-VBD-VBN	18.05%	19.01%	18.76%	**17.70%**
9	NN-VBG	14.37%	19.58%	14.37%	13.65%
10	JJ-NNP	5.91%	5.48%	**4.69%**	5.61%
11	JJ-RB	**9.60%**	9.74%	9.85%	11.11%
12	DT-IN-RB-WDT	9.14%	**7.48%**	7.83%	8.31%
Average error-rate		8.48%	8.84%	8.33%	8.44%
num-nodes		561.9	463.8	479.2	398.8

TABLE 12. Testing the extended set of attributes

The previous results lead to the following conclusions:

- Comparing the 'A' and 'All' columns, we see that, despite some particular differences on some data sets, they perform globally equal (average error rates: 8.34% and 8.44%). Thus, adding the whole set of attributes at a time leads to no accuracy improvement. However, when using the enlarged set of

attributes smaller trees were acquired (average number of nodes: 480.1 and 398.8), since the big number of attributes allows to better fit the training set.

- Regarding the particular groups of attributes, the best is B+X (tag patterns) while the worst is G+X (ambiguity classes). The other combinations are in between with fairly low global differences. Comparing to the A–set of basic attributes, we see that the particular groups have a quite bigger variance that affects both the more specific (e.g. F+X) and the more general sets (e.g. G+X). For instance F+X obtains the best result (among all combinations) in the `JJ-NN` data set but also the worst in the `VBD-VBN` set. Similarly, G+X obtains much worse results than the rest in the `NN-VBG` data set, but the best score in the `DT-IN-RB-WDT` domain. On the contrary, the A set is much more stable and despite achieving none of the best results is always among the better groups. This may suggest that the perfect combination of attributes depends on the concrete data set and that, in general, none is superior to the rest.

  Finally, note that the greater variability of particular groups of attributes (with the exception of G+X) is clearly accompanied by a greater size of the induced trees. This may be explained by the fact that a relative small set of specific attributes is not enough to concisely fit the training data.

- The fact that several groups of attributes result in trees that perform all reasonably well allows the construction of ensembles of trees, which are able to exploit the differences between them and obtain combined classifiers that outperform all the individuals. This extension will be explained in detail in chapter 6. Note that the set of features used to deal with unknown words was also enlarged, up to 23 attributes (see table 3 of chapter 6), with the same purpose.

4.3. Dealing with Sparseness. We mentioned in the introduction of the chapter that even though the examples are grouped into ambiguity classes of equivalence, there are some training sets that suffer from sparseness, i.e., they contain not enough examples to reliably apply statistical inductive methods. We have proposed two approaches to address the problem of small training sets. Both are discussed below.

4.3.1. *Global Smoothing.* Taking advantage of the taxonomic structure of ambiguity classes, we proposed a technique consisting of backing off to more general ambiguity classes to increase the number of examples of low represented classes. We will call this method *global smoothing.*

Figure 8 reproduces part of the WSJ ambiguity–class taxonomy (which was previously presented in figure 1). Boxes with thick lines represent terminal nodes in the DAG, while shadowed boxes represent the successive generalization of the examples of the `JJ-NN-RB` ambiguity class (adjective–noun–adverb, e.g. "enough").

To complete the training set of this class, the 'global smoothing' approach would begin by adding examples belonging to the 4-ambiguity level superclasses: e.g. adjective–adverb–noun–verb (e.g. "right"), adjective–adverb–noun–preposition

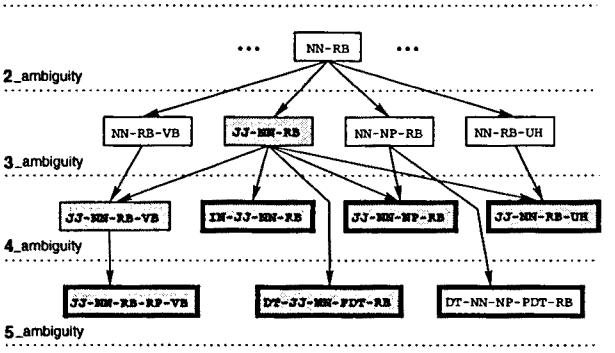

FIGURE 8. A part of the ambiguity-class taxonomy for the WSJ corpus

(e.g. "outside"), etc., taking into account only adjective, noun, and adverb occurrences. In subsequent steps, more general examples could be added, i.e. those of 5–ambiguity level, and so on.

We tested this approach on four relatively small data sets, applying an incremental procedure by considering a new level of generality at each step. The aggregation of new examples was performed in a naive way consisting of adding them all at the beginning of the tree–induction process. The four first rows of table 13 contain the results obtained. The 'Base' column shows the results of the trees learned with the basic training set, while the 'Glevel-n' columns stand for successive steps of generalization. Error rates and the number of training examples (the concrete increase is in parentheses) are reported in each case.

	AClass	Base	Glevel–1	Glevel–2	Glevel–3	Glevel–4
1	NN-RB	6.50% 2,621	6.29% 3,716 (1,095)	6.52% 4,199 (483)	6.78% 4,601 (402)	—
2	JJR-RBR	16.21% 3,178	13.94% 4,001 (823)	14.39% 4,313 (312)	—	—
3	JJ-NN-RB	14.37% 2,908	12.57% 3,863 (955)	12.44% 3,948 (85)	12.01% 4,062 (114)	—
4	JJ-NN-VB	13.18% 2,363	12.07% 3,971 (1,608)	11.90% 4,151 (180)	11.98% 4,343 (192)	—
5	JJ-RB	10.24% 9,628	12.09% 10,922 (1,294)	12.39% 12,201 (1,279)	12.42% 12,430 (229)	—
6	JJ-NN	14.66% 18,824	14.50% 25,113 (6,289)	14.78% 27,300 (2,187)	14.90% 27,687 (387)	14.90% 27,527 (140)
7	NN-VBG	13.89% 10,560	14.39% 12,604 (2,044)	14.30% 12,782 (178)	—	—
8	VBD-VBN	6.24% 28,417	8.33% 40,150 (11,733)	8.39% 40,504 (354)	8.51% 41,258 (754)	8.48% 41,494 (236)

TABLE 13. Results of applying the global smoothing technique

It can be seen that the accuracy was improved in three of the four data sets, while in the fourth (NN-RB), the accuracy remained invariable. It is important to note that the accuracy increased up to a maximum and then tended to stabilize (to slightly decrease in some cases) as more general examples were added.

We also tested if this approach could be a feasible way for improving accuracy in general ambiguity classes. For that, we applied global smoothing to four of the very frequent ambiguity classes. The results were negative in this case (see columns 5–8 of table 13), since in two of the four data sets the accuracy did not change, while in the other two the smoothing was significantly counterproductive.

The presented approach suffers from two basic problems: On the one hand, the scarceness of more general examples is evident in many ambiguity classes[20]. On the other hand, the more back–off steps are carried out, the more degradation appears on the quality of the added information. As previously noted, this last drawback may have a serious influence on the achievable improvement with this technique.

Additionally, there are some open question that should require further research: e.g, which is the best way to perform the aggregation of new examples, when to stop, etc.

4.3.2. *Generating Convex Pseudo Data* (CPD). We have tested an alternative based on a recent paper by Breiman [**Bre98b**], in which he describes a quite simple and effective method for generating new pseudo–examples from existing data and incorporating them into a tree–based learning algorithm to increase prediction accuracy in domains with few training examples.

We describe it below, with a slight variation and assuming that all features are discrete–valued, as it is the case in the our tagging domain.

The method for generating new data from the old depends only on a single parameter d, $0 < d < 1$, and it reminds the process of *gene combination* to obtain new generations in genetic algorithms. More particularly, to create a new data instance, these steps are followed: (1) Select two instances $(\mathbf{x}_1, y)$, $(\mathbf{x}_2, y)$ at random from the training set[21]; (2) Select a random number v from the interval $[0, d]$, and let $u = 1 - v$. (3) The new instance is $(\mathbf{x}_3, y)$ is obtained by combination in the following way. For each attribute i, $1 \leq i \leq m$, let $x_{1,i}$ and $x_{2,i}$ be the i-th attribute values in first and second examples, respectively. Then $x_{3,i}$ is assigned $x_{1,i}$ with probability u and $x_{2,i}$ with probability v.

Additionally, Breiman proposes a particular method for introducing the pseudo–examples in the CART tree–induction algorithm [**BFOS84**]. Let X be the initial small data set, and N its number of examples. The tree–induction algorithm considers the X set unchanged at the root of the tree, but for any subsequent internal node the associated subset of examples is enlarged with dynamically generated pseudo–examples, until it contains N examples. The induction process ends, for each parent node, when either of the two children nodes (CART system learns binary trees for classification) contain no examples of the original training set X.

The implementation of this method presents some computational problems, as the proper author notes in his paper. For the sake of simplicity, we have tested the application of this technique by adding a fixed amount of generated pseudo–examples at a time, at the root level of the tree. In spite of the simplicity of the proposal, very good results were obtained.

In the original paper, Breiman does not propose any optimization of the generation parameter d (the role of d is to regulate the amount of change allowed in

[20] For instance, the very basic JJ-VBI and JJ-VBG ambiguity classes, contain only 4 and 69 more general examples in the training corpus, respectively.

[21] Breiman allows the classes of both examples to be different. However we empirically observed that, in our domain, much better results were obtained by mixing only pairs of examples of the same class.

the combination of two examples), instead, he performs a limited amount of trials with different values of d and simply reports the best achieved result. In our case, we have observed a great variability on the results, depending on the value of d (with a slight advantage of values bigger than 0.3). Instead of trying to optimize it, we propose a new solution consisting of learning several decision trees, each one corresponding to a different value of d, and combining the outcomes of single decision trees by averaging the probabilities over the possible tags. In this way, we generally obtain a combined result which is at least as accurate as any of the individual classifiers, and therefore we make the global classifier more robust.

The results are impressive in some domains. For instance, dealing with the preposition–adverb ambiguity class, an experiment was performed using an increasing number of original examples as the base for generating new pseudo–data. The results were that, starting with a set of only 100 examples, the addition of pseudo–examples reported an accuracy similar to that obtained from 250 real examples; the improvement achieved departing from 500 original examples was comparable to that of training with 2000, etc.

These results are depicted in figure 9 which plots the curves of the error rate with respect to the training set size. In that figure, 'DTrees' stands for the results obtained by the trees learned from unchanged training sets, while 'CPD' stands for the results achieved with the ensembles of trees acquired by enlarging the training sets with a fixed amount of pseudo–examples generated using increasing values of the generation parameter (between 0.3 and 0.8).

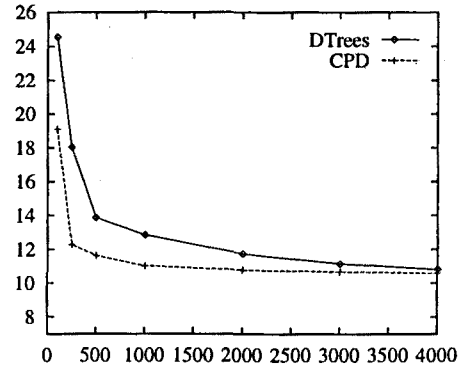

FIGURE 9. Performance of the CPD method vs. normal decision trees on the preposition–adverb data set. The X–axis represents the number of training examples and the Y–axis the error rate in %

Unfortunately, the resulting overhead was also noticeable. For instance, in the third point of the previous example we are talking about jumping from one single tree of about 170 nodes acquired from 500 examples, to the combination of six trees learned from 3000 examples (500 original and 2500 generated) of about 300 nodes each. In addition, we can observe that the improvement becomes negligible when the algorithm starts with a big enough number of training examples, so, as the 'global smoothing' method, CPD is not guaranteed to be useful when applied to very frequent ambiguity classes.

Finally, compared to the global smoothing approach, the generation of convex pseudo–examples overcomes the two practical drawbacks mentioned in the las section, and it was superior in the last reported experiment, where the average error reduction was 25.3% higher.

4.3.3. *Discussion.* The above presented approaches for addressing the problem of sparseness in ambiguity classes, especially the generation of convex pseudo–data, seem very promising, moreover when some improvements are still to be done. For instance, a more careful way of introducing the completing examples in the decision tree particular nodes, instead of including them all at the beginning, would probably report some benefits. In addition, it seems reasonable to think in assigning confidence weights to the examples, so that it would be possible to place a greater importance on the original examples than to the generated pseudo–examples, etc.

However, in chapter 6 we will empirically test the effect of CPD on several real POS taggers and we will see that the global improvement is very small. Some additional discussion will be provided in that chapter.

4.4. Pending Work. In another direction, further work should be carried out to extend the capabilities of the learning algorithm. The limitations of usual tree–induction algorithms are well known: greedy search, instability, difficulties to deal with small disjuncts, univariate splits that lead to simple linear partitions on the feature space, etc. (see section 4 of chapter 2 for more details).

Several techniques have been proposed to partially reduce the effect of such intrinsic problems. Among others, we should mention: Pruning, searching with lookahead, multi–pass construction, multivariate splits, construction of ensembles of decision trees, etc. Some of them have already been applied in the work presented in this chapter, and other regarding ensembles will be presented in chapter 6. Nevertheless, we think that other two extensions should be investigated, namely, searching with a limited lookahead and the inclusion of multivariate splits.

CHAPTER 4

Tagging with the Acquired Decision Trees

Once the tree–based model has been acquired we have to decide how to use it for tagging a sequence of ambiguous words. The most straightforward approach is to use the trees as direct classifiers to decide the proper part–of–speech of each word in a single pass, e.g., from left to right, through the sequence. Even this simple approach has to face a specific problem because determining the part–of–speech of a concrete word may depend on the POS of the some neighbour words that are still ambiguous, e.g., the words to the right of the current word, if we proceed from left to right. Even though this difficulty can be easily overcomed, we have experimentally observed that the accuracy of such an algorithm can be significantly improved by considering the statistical information provided by the trees within more complex disambiguation algorithms. More particularly, we have implemented two taggers following different approaches, namely:

- A reductionistic tagger (RTT), which uses an an iterative algorithm that reduces the initial POS ambiguity at each step by filtering out low probable tags. The update of probabilities at each iteration is performed by applying the decision trees to the ambiguous words.
- A statistical tagger (STT), which disambiguates the input sequence by assigning the most probable sequence of tags, given the observed sequence of words. In this case, decision trees are used for estimating the contextual probabilities of the statistical model.

The former is presented and evaluated in section 1, while the latter is presented in section 2. Some comments about the comparison of both taggers, and about their limitations are presented in section 3. Finally, section 4 is devoted to present some extensions with the aim of overcoming the previously outlined problems. In particular, we show how the decision–tree–based model can be adapted to a flexible POS tagger based on relaxation labelling.

1. RTT: a Reductionistic Tree–based Tagger

We have implemented a *reductionistic* tagger in the sense of Constraint Grammars [**KVHA95**], i.e., in an initial step, a word-form frequency dictionary constructed from the training corpus provides each input word with all possible tags with their associated lexical probability, and, after that, an iterative process reduces the ambiguity (by discarding low probable tags) at each step until a certain stopping criterion is satisfied. We will call this tagger RTT standing for *Reductionistic Tree–based Tagger*. The architecture of RTT is represented in figure 1.

More particularly, at each step and for each ambiguous word the work to be done in parallel is:

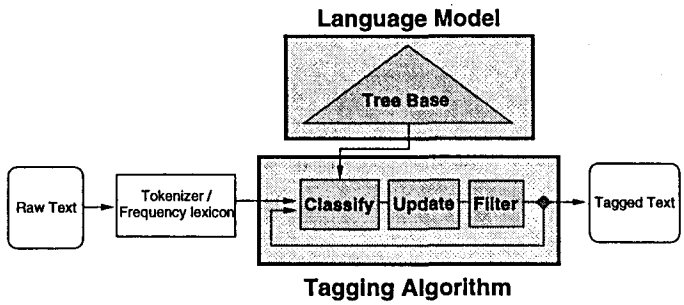

FIGURE 1. Architecture of RTT

1. Classify the word using the corresponding decision tree. The ambiguity of the context (either left or right) during classification may generate multiple answers for the questions of the nodes. In this case, all the paths are followed and the result is taken as a weighted average of the results of all possible paths. The weight for each path is actually its probability, which is calculated according to the current probabilities of the POS tags of the features involved in the nodes of the path.
2. Use the resulting probability distribution to update the probability distribution of the word. The probability updating is done by simply multiplying previous probabilities per new probabilities coming from the trees and renormalizing the results (so they sum to one again).
3. Discard the tags with *almost* zero probability, that is, those with probabilities lower than a certain *discard boundary* parameter.

As an example, we present in table 2 the real process of disambiguation of a text. In this figure we can see at each iteration which are the *surviving* part–of–speech tags for each word and their associated probabilities. In this simple example the text becomes unambiguous (and correctly disambiguated) at the second iteration of the algorithm. Note that the text disambiguated in that example is a part of a sentence collected from the WSJ corpus which is fully presented in table 1, together with the possible parts–of–speech for the ambiguous words.

The_DT first_JJ time_NN he_PRP was_VBD shot_VBN in_IN the_DT hand_NN as_IN he_PRP chased_VBD the_DT robbers_NNS outside_RB ...

first	time	shot	in	hand	as	chased	outside
JJ	NN	NN	IN	NN	IN	JJ	IN
RB	VB	VBD	RB	VB	RB	VBD	JJ
		VBN	RP			VBN	NN
							RB

TABLE 1. A sentence and its POS ambiguity. Appearing tags, from the Penn Treebank corpus, are described in appendix C.

After the stopping criterion is satisfied, some words could still remain ambiguous. Then there are two possibilities: (1) Choose the most probable tag for each still–ambiguous word to completely disambiguate the text, or (2) Accept the residual ambiguity (for subsequent treatment).

...	as	he	chased	the	robbers	outside	.
it.0	IN:0.83	PRP:1	JJ:0.25	DT:1	NNS:1	IN:0.54	.:1
	RB:0.17		VBD:0.28			JJ:0.36	
			VBN:0.57			NN:0.06	
						RB:0.04	
it.1	IN:0.96	PRP:1	VBD:0.97	DT:1	NNS:1	IN:0.01	.:1
	RB:0.04		VBN:0.03			JJ:0.01	
						RB:0.98	
it.2	IN:1	PRP:1	VBD:1	DT:1	NNS:1	RB:1	.:1
stop							

TABLE 2. Example of disambiguation

Note that a unique iteration forcing the complete disambiguation is equivalent to using the trees directly as classifiers, and results in a very efficient tagger, while performing several steps progressively reduces the efficiency, but takes advantage of the statistical nature of the trees to get more accurate results.

Another important point is to determine an appropriate stopping criterion[1]. Some experiments seem to indicate that the performance increases up to a unique maximum and then softly decreases as the number of iterations increases. Figure 2 illustrates this behaviour.

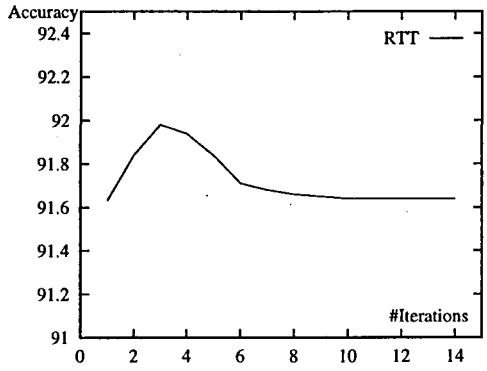

FIGURE 2. Accuracy, on ambiguous words, of RTT depending on the number of iterations

This phenomenon is studied by [**Pad98**] and the noise in the training and test sets is suggested to be the major cause. For the sake of simplicity, in the experiments reported the following section, the number of iterations was experimentally fixed to three. Although it might seem an arbitrary decision, broad-ranging experiments performed seem to indicate that this value results in a good average tagging performance in terms of accuracy and efficiency.

1.1. Evaluation. We divided the WSJ corpus into two parts: 1,120,000 words were used as a training/pruning set, and 50,000 words as a fresh test set. See table 3 for some details about the training and test corpora.

[1] The disambiguation procedure is heuristic, and its convergence is not guaranteed. However, in most of our experiments convergence was reached with a small number of iterations.

	S	W	W/S	AW	T/W	T/AW
Training	45,993	1,121,776	24.39	381,737 (34.05%)	1.48	2.40
Test	2,151	51,990	24.17	17,619 (33.89%)	1.45	2.40
Total	48,144	1,173,766	24.38	399,356 (34.02%)	1.47	2.40

TABLE 3. Information about the WSJ training and test corpora. S: number of sentences; W: number of words; W/S: average number of words per sentence; AW: number and percentage of ambiguous words; T/W: average number of tags per word; and T/AW: average number of tags per ambiguous word

We used the lexicon described in section 3.1 of chapter 3, which is derived from training corpus, and contains all possible tags for each word, as well as their lexical probabilities. For the words in the test corpus not appearing in the training set, we stored all the tags that these words have in the test corpus, but no lexical probability (i.e. assigning uniform distribution). This approach corresponds to the assumption of having a morphological analyzer that provides all possible tags for unknown words, and it is often referred to as the *closed vocabulary assumption*. In following experiments (see chapter 6) we will treat unknown words in a less informed way.

From the 243 ambiguity classes the acquisition algorithm learned a base of 194 trees (covering 99.5% of the ambiguous words) and requiring 565 Kb of storage. The learning algorithm (implemented using Lucid Common Lisp) took about 12 CPU-hours running on a SUN SparcStation-10 with 64Mb of primary memory.

The first four columns of table 4 contain information about the trees learned for the twelve most representative ambiguity classes (those described in table 5 of the previous chapter). They present figures about the number of examples used for learning each tree[2], and the classification error over the set of examples used for pruning. This last figure could be taken as a rough estimation of the error of the trees when used in RTT, though it is not exactly true, since in the pruning examples the neighboring tags are given their correct tags from the supervised annotation, while during tagging both contexts —left and right— can be ambiguous.

The tagging speed of the first RTT prototype, implemented in PERL-5.003 (which is an interpreted language), was slightly over 300 words/sec. running on a SUN UltraSparc2 machine with 128MB of primary memory.

The results obtained can be seen at different levels of granularity.

- The right part of table 4 contains information about the performance of the decision trees corresponding to the twelve reference data sets. The number and percentage of errors is presented in 'RTT' column, while the '%Red' column reflects the percentage of error reduction due to the use of RTT, comparing to a most-frequent-tag tagger (MFT_2 heuristic). On the one hand, we observe a remarkable reduction in the number of errors: 56.6% (recall that the twelve data sets concentrate up to 67.4% of the errors committed by the most-frequent-tag tagger). On the other hand, this table allows the identification of some problematic cases. For instance, JJ-NN seems to be

[2]The Lisp-based implementation of the tree-learner had problems to work with more than 30,000 examples. This is why we did not use the full —38,112 examples— training set for the IN-RB-RP ambiguity class. The proportion of examples considered for pruning varied from 0 (no pruning for the small training set) to 15% depending on the training set size.

	Amb. Class	#Exs	#N	%Err	MFT_2–err	RTT–err	%Red
1	IN-RB-RP	30,000	99	7.13%	210 (11.70%)	164 (9.14%)	21.88%
2	VBD-VBN	28,417	844	7.53%	267 (18.98%)	91 (6.47%)	65.91%
3	NN-VB-VBP	26,970	576	3.32%	255 (20.10%)	51 (4.02%)	80.00%
4	VB-VBP	19,718	313	3.67%	226 (24.42%)	46 (4.97%)	79.65%
5	JJ-NN	18,824	680	16.30%	144 (16.54%)	122 (14.01%)	15.30%
6	NNS-VBZ	16,928	688	4.37%	81 (11.40%)	44 (6.19%)	45.70%
7	NN-VB	16,239	221	1.11%	67 (9.10%)	12 (1.63%)	82.09%
8	JJ-VBD-VBN	12,607	761	18.75%	180 (31.64%)	95 (16.70%)	47.22%
9	NN-VBG	10,560	564	16.54%	116 (21.68%)	58 (10.84%)	50.00%
10	JJ-NNP	9,650	351	5.90%	68 (13.91%)	26 (5.29%)	61.97%
11	JJ-RB	9,628	854	11.20%	73 (16.49%)	48 (10.84%)	34.26%
12	DT-IN-RB-WDT	9,237	271	6.07%	187 (40.34%)	56 (12.08%)	70.05%
Total		208,707	6,222	—	1,874	813	56.62%

TABLE 4. Information about the decision trees acquired for the twelve ambiguity classes of reference. #Exs: number of training examples; #N: size of the tree in number of nodes; %Err: estimated error rate of the tree; MFT_2–err: performance ('number of errors' and 'error rate' in parenthesis) of the most–frequent–tag tagger on the test set. RTT–err: same meaning but considering the RTT tagger; and %Red: percentage of error reduction

the most difficult ambiguity class, since the associated tree obtains only a slight error reduction from the MFT baseline tagger (15.3%). This is not surprising since semantic knowledge is necessary to fully disambiguate between noun and adjective. Results for the DT-IN-RB-WDT ambiguity reflect an over–estimation of the generalization performance of the tree —the predicted error rate (6.07%) is much lower than the real (12.08%)— which may be indicating that the decision tree overfits the pruning data set.

- Global results at the third iteration are the following: when forcing a complete disambiguation, the resulting accuracy is 97.29%, while accepting the residual ambiguity the *recall* is increased up to 98.22%. In this case, the resulting ambiguity ratio is 1.08 tags/word over the ambiguous words and 1.026 tags/word overall, which corresponds to a *precision* of 95.73%[3]. In other words, 2.75% of the words remained ambiguous (over 96% of them retaining only 2 tags). In chapters 6, and 7 it is shown that these results are comparable to the best state–of–the–art taggers based on automatic model acquisition.

In order to complement the previous information, we present in figure 3 the performance achieved by our tagger with increasing sizes of the training corpus. Results in accuracy are computed over all words. The same figure includes MFT_2 results, which can be seen as a reference baseline.

[3] 'Recall' and 'precision' measures, which are usually complemented with figures about coverage, are the base for the standard evaluation of systems for information extraction. Regarding POS tagging, these measures have a simpler meaning, and they are used to measure the quality of taggers that perform an ambiguity reduction but do not necessarily a complete disambiguation. Precisely, 'recall' stands for the percentage of words that retain the correct tag, while 'precision' stands for the percentage of correct tags among all retained.

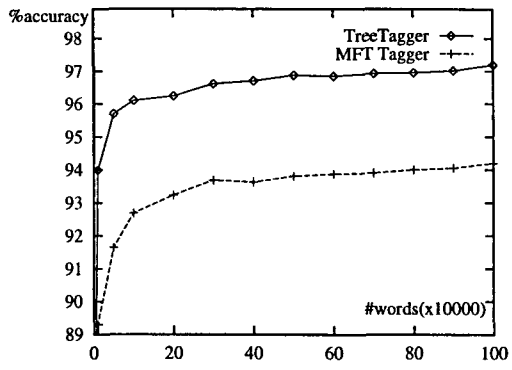

FIGURE 3. Performance of the tagger related to the training set size

Following the intuition, we see that performance grows as the training set size grows. The maximum is at 97.29%, as previously indicated.

The rejection curve for our classifier was also calculated in order to test the quality of the probability estimates given by the decision trees. Following the same explanation than in section 3 of chapter 3, the confidence levels were calculated as the differences between the two most probable predictions. As a difference, in this case the rejection curve is not plotted for each individual tree, but it is calculated from the predictions of all trees on the test set after one iteration of the disambiguation algorithm (without filtering any part–of–speech tag).

The rejection curve, which is depicted in figure 4, increases fairly smoothly, giving the idea that the tree–based classifier provide good confidence estimates. This is in close connection with the aforementioned high recall figures yielded by the tagger when disambiguation in the low–confidence cases is not forced.

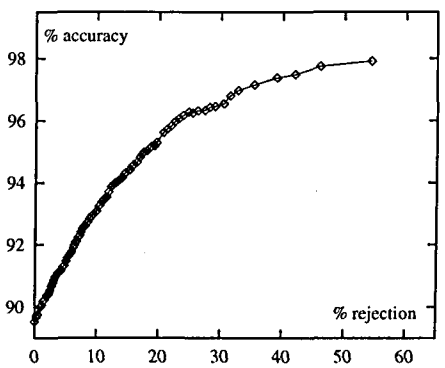

FIGURE 4. Rejection curve for the RTT classifier, trained using the whole training corpus. Accuracy figures are given over ambiguous words

2. STT: A Statistical Tree–based Tagger

First, we will present the general framework in which statistical taggers take place, and second, we will explain how to introduce the acquired decision trees into the statistical approach.

2.1. Statistical and HMM–based Tagging[4]. The aim of statistical or probabilistic tagging is to assign the most likely sequence of tags given the observed sequence of words, that is,

$$\arg\max_{TagSeq} P(TagSeq \mid WordSeq)$$

Statistical taggers make use of two sources of information: (1) the *lexical probabilities*, i.e., the probability of a particular tag conditional on the particular word, $P(t|w)$, and (2) the *contextual probabilities*, which describe the probability of a particular tag conditional on the surrounding tags. This second source is often reduced to the conditioning of the n–1 preceding tags, that is, $P(t_k|t_{k-n+1}, \cdots, t_{k-1})$. Such probabilities are often referred to as *transition probabilities* or n–grams (e.g., bigrams, trigrams, etc.).

At runtime, the tagger typically works as follows: First, each word is assigned a set of all possible tags according to the lexicon. This will create a lattice. A dynamic programming technique, the *Viterbi* algorithm (described for instance in [**DeR88**]), is then used to find the sequence of tags $t_1 \cdots t_n$ that maximize:

$$P(t_1, \ldots, t_n|w_1, \ldots, w_n) = \prod_{k=1}^{n} P(t_k|t_1, \ldots, t_{k-1}; w_1, \ldots, w_n) \approx$$

$$\prod_{k=1}^{n} P(t_k|t_{k-2}, t_{k-1}, w_k) \approx \prod_{k=1}^{n} \frac{P(t_k|t_{k-2}, t_{k-1}) \cdot P(t_k|w_k)}{P(t_k)} =$$

$$\prod_{k=1}^{n} \frac{P(t_k|t_{k-2}, t_{k-1}) \cdot P(w_k|t_k)}{P(w_k)}$$

The previous chain requires the application of Bayesian inversion and two simplifying assumptions. The first is to assume that the probability of a certain tag in position k only depends on the k–th word and the two previous tags (trigram modelling). The second, assumes independence between information sources.

Finally, since $P(w_k)$ does not depend on the tag sequence, statistical POS tagging can be stated as:

$$\arg\max_{t_1 \ldots t_n} \prod_{k=1}^{n} P(t_k|t_{k-2}, t_{k-1}) \cdot P(w_k|t_k)$$

Taggers based on Hidden Markov Models [**Rab90**] work exactly in the same way, however its abstract formulation is a bit different. In its application to POS tagging, each word is viewed as a signal emitted from some (hidden) internal state consisting of a tag, $[t^i]$, in the bigram case, or a pair of tags, $[t^i t^j]$, in the trigram case. The *signal model* refers to the set of parameters describing the probabilities of each word being emitted by some state, e.g., $P(w|t^j)$ approximates the probability of state $[t_i t_j]$ emitting w. Transitions between states contain n–gram probabilities, e.g., for the trigram case, the edge between state $[t^i t^j]$ and state $[t^j t^k]$ would contain

[4]This introduction to statistical tagging follows that of Samuelsson and Krenn, in [**KS97**].

the probability $P(t_n^k|t_{n-2}^i, t_{n-1}^j)$, that is the probability of tag t^k being preceded by the pair of tags t^i, t^j. This set of parameters define what is often referred to as the *language model.* The tagging task then consists of finding the most likely sequence of states (i.e. the most likely sequence of tags) given the observed sequence of words. The same dynamic programming based algorithm is applied in this case.

The already given presentation follows the probabilistic formulation because it allows a simpler and more brief explanation, and it better fits the extension to decision trees presented below.

2.2. The STT Tagger. Statistical decision trees can be seen as compact representations of the contextual model for a statistical tagger. In particular, each tree can be seen as a function $\mathcal{T}(t; C)$, that return the probability of a tag t for a concrete word, given its context of appearance, C. Following exactly the same formulation presented in the previous section, and making the same assumptions, we can recast the problem of tagging as:

$$\arg\max_{t_1 \ldots t_n} P(t_1, \ldots, t_n | w_1, \ldots, w_n) \approx \arg\max_{t_1 \ldots t_n} \prod_{k=1}^{n} \mathcal{T}_{A_k}(t_k; C_k) \cdot P(w_k|t_k),$$

where A_k stands for the ambiguity class of word w_k, and $\mathcal{T}_{A_k}(t_k; C_k)$ stands for the probability of tag t_k for the word w_k in the context C_k. This probability is provided by the tree corresponding to the concrete ambiguity class, that is, $\mathcal{T}_{A_k}$. The point here is that the context conditioning the tag t_k is not restricted to the two preceding tags as in the trigram formulation. Instead, it is extended to all the contextual information used for acquiring the decision trees, e.g. the three preceding tags, the two following tags and the word form of the word to be disambiguated.

By using dynamic programming, this most-likely sequence of tags can be calculated with a linear cost on the sequence length[5]. The algorithm is the same *Viterbi* algorithm, in which we substitute the n-gram probabilities by the application of the corresponding decision trees. However, one problem appears when applying conditionings on the right context of the word to be disambiguated, since the disambiguation proceeds from left to right and, so, the right hand side words may be ambiguous. Although dynamic programming can be used again to calculate the most likely sequence of tags to the right (in a forward–backward approach), we propose a simple and cheaper approach consisting of calculating the contextual probabilities by a weighted average of all possible tags for the right context.

Suppose that we denote the left context by C^- and the right context by C^+, and suppose that the right context consists of the two following tags. Then, the contextual probability of tag t_k for word w_k is calculated by:

$$\mathcal{T}_{A_k}(t_k;\ C_k^-, t_{k+1}, t_{k+2}) = \sum_{(t_i,t_j) \in C_k^+} \mathcal{T}_{A_k}(t_k;\ C_k^-, t_i, t_j) \cdot P((t_i, t_j)|C_k^+),$$

where $P((t_i, t_j)|C_k^+)$ stands for the probability of words w_{k+1} and w_{k+2} being tagged as t_i and t_j. These probabilities are extracted from lexical probabilities.

[5]In our implementation the disambiguation is performed sentence by sentence (i.e., the sequence length n in the formulation refers to the length of the currently addressed sentence), since we assume no inter–sentential dependencies.

2.2.1. *Some Comments on this Approach.* The main theoretical advantage of the decision-tree approach is that it represents an easy, compact and efficient way of extending the n-gram modelling in statistical tagging.

The usual problem in extending n-gram models is the exponential growth of the number of parameters to be estimated, that makes available corpora not enough to reliably estimate them. The sparseness problem comes together with a high space requirement to store the huge matrix of statistical parameters.

Most systems use a kind of smoothing procedure to deal with sparseness, by using statistical information of different levels of generality and applying it in a back-off approach, that is, to apply concrete models when there is enough evidence and back off to more general information when there is not. A brief presentation of general methods for smoothing is included in chapter 2.

The induction method for acquiring decision trees is an easy way of automatically decide a reduction of statistical parameters and perform a kind of smoothing, alleviating both problems of sparseness and space requirements. For instance, taking a penta-gram model (which is the usual context taken into account in our decision tree modelling) and assuming the Penn Treebank tagset, containing 45 tags, the potential number of parameters to be estimated[6] is $45^5 = 184,528,125$. It is obvious that they can not be reliably estimated with a 1 million words corpus.

In our approach, the induction algorithm decides which of this information is useful in each case. The final representation with trees contains only a subset of the whole set of statistical parameters, mixing different levels of generality. For instance, sometimes the bigram evidence is enough —this would be the case of a depth 2 branch of a tree— but sometimes more precise information is required in the deeper nodes. Additionally, n-grams are not necessarily continuous and can refer to the left or right of the word to be disambiguated. In the aforementioned domain, the final model, consisting of less than 200 trees, requires a manageable 0.5Mb of storage, and, what is more important, captures relevant information that allows to surpass the bigram and trigram models.

2.2.2. *Evaluation.* By the time in which the previously explained experiments using RTT on the WSJ were performed (January '97), STT was still not available. Therefore, we cannot provide the results obtained with the STT tagger under exactly those experimental conditions.

Nevertheless, we performed more recent experiments, using the WSJ corpus, in which both taggers were tested. Additionally, we could also reproduce the same training and test sets for an experiment on a Spanish corpus. The former are explained in chapter 6, while the latter appear in chapter 5. We reproduce below, the global results obtained in both cases.

- Tested on the WSJ, under the conditions expressed in chapter 6, STT obtained a global accuracy of 96.63%, which is almost equal than that obtained by RTT, 96.61%. However STT was slower than RTT: 321 word/sec. vs. 426 words/sec.

[6] In fact the real number of syntactically valid sequences of 5 tags is much lower, but here is precisely where the main problem appears, which is to distinguish the impossible sequences of tags from those that are simply very rare and do not appear in the corpus.

- Tested on the Spanish LEXESP corpus, STT achieved an accuracy of 97.51%, while RTT was slightly less accurate, 97.44%[7].

3. Comparison and Discussion

We think that none of the taggers is better than the other in absolute terms. The choice of which tagger to use for a particular NLP application would strongly depend on the user needs and on the kind of application. The following points summarize some pros and cons of both approaches:

- Although both taggers have the same linear time complexity on the input sequence, RTT has empirically proved to be faster than STT (especially when STT incorporates n–gram information) . See chapter 6 for some comparative figures.
- Due to its probabilistic formulation, STT obtains the most likely sequence of tags, that is, exactly one tag for each input word. Instead, RTT, can be adapted, by tuning the filtering threshold and the number of iterations, to obtain the desired level of residual ambiguity, avoiding to decide in the most difficult cases. Depending on the user needs, it might be worthwhile accepting a higher remaining ambiguity (lower precision) in favour of a higher recall. In this way a tradeoff between recall and precision can be easily stated for RTT.
- STT achieved roughly the same accuracy than STT on the English corpus, but it was slightly more accurate on the Spanish corpus. Nevertheless, this difference is not significant, and broader comparisons should be performed to verify a possible superiority of STT in terms of accuracy.
- The disambiguating procedure used in RTT is heuristic and so, the convergence is not guaranteed. Despite this lack of theoretical support, our empirical experience shows that it is easy to properly tune the algorithm to acquire fairly good levels of efficiency and tagging accuracy.
- STT allows a straightforward incorporation of n–gram information into the disambiguation algorithm. In this way, the data sparseness problem coming from the ambiguity–class approach can be better alleviated. This is an issue that will be addressed in the following sections.

3.1. Identifying Problems. The already presented taggers suffer from one problem, namely, the restriction to address the ambiguity problem through the specific ambiguity classes, which in some cases presents the data sparseness problem. This problem becomes evident in a sentence such as[8]:

El hombre bajo toca el bajo bajo las estrellas

(The short man plays the bass under the stars)

Here, the challenge is to disambiguate the word "bajo" which has several readings: noun ("bass"), adjective ("short"), preposition ("under"), and, not present in the above sentence, two additional readings: adverb ("softly") and verb (1st person, present form of the verb "to descend"). The problem is that this word is the

[7]These figures were obtained by using a training set of 71Kw and a retraining step consisting of an extra set of 800Kw words automatically tagged with the first tagger, i.e., using the $\mathcal{C}^1_{800}$ training corpus, according to the notation presented in chapter 5.

[8]The sentence is given in Spanish because it corresponds to the real example that we were studying when we first noticed the reported problem.

unique word, in the training corpus, belonging to the *noun–adjective–preposition–adverb–verb* ambiguity class (i.e., in this case the acquired tree accounts for a single word), and that almost always appears as a preposition (99 occurrences vs. 2 occurrences as an adjective, 1 as a noun and none for the rest). With this training set, is almost impossible to learn the cases in which "bajo" acts as an adjective or a verb, and definitely impossible for the verb and adverb cases. The result is that in the previous sentence all occurrences of "bajo" are tagged as a preposition, when the correct tagging is: adjective, noun, and preposition, respectively.

However, in this case a simple bigram probabilities give strong evidence to the correct tag assignment for the word "bajo". For instance, the noun "hombre" ("man") restricts the first "bajo" to be basically adjective or preposition, and the following verb "toca" ("plays") gives ten times more evidence to the adjective reading than to preposition. The determiner "el" preceding the second occurrence of "bajo", coerces it to be noun or, at a great distance, adjective. The noun reading is highly compatible with the preposition and adjective readings for the third "bajo", and the adjective reading shows preferences to be followed by noun and preposition. All these combinations are resolved by observing that the preposition reading for the third "bajo" is the only really admissible in combination with the following unambiguous determiner "las". Apart from the preceding argument, the tag sequence *preposition–preposition*, is very uncommon in Spanish and it should *never* be assigned to the two adjacent "bajo".

One should expect that presented evidence is strong enough to overcome the lexical probability bias to the preposition reading. This is the case of a simple bigram based statistical tagger, which succeeds in the tagging of the sentence, while the much more complicated tree–based approach clearly fails.

This observation made us to consider the systematic addition of simple n–gram information to ensure a minimum coherence in the final tag sequence. In a first step (section 3.2), this was achieved by using n–grams within STT. In a second step (section 4) we propose a more general solution which consists of combining information from different sources (including n–grams and decision trees) by using a constraint–based flexible POS tagger.

3.2. Adding n–grams to the STT. The statistical approach of STT allows a straightforward incorporation of n–gram probabilities, by linear interpolation, in a back–off approach including, from most general to most specific, unigrams, bigrams, trigrams and trees. The general expression for such linear interpolation is:

$$P(t_k|C_k) \approx \lambda_4 \cdot \mathcal{T}_{A_k}(t_k; C_k) + \lambda_3 \cdot \hat{p}(t_k|t_{k-2}, t_{k-1}) + \lambda_2 \cdot \hat{p}(t_k|t_{k-1}) + \lambda_1 \cdot \hat{p}(t_k)$$

where n–gram probabilities are simply estimated by relative frequencies in the training set. The λ coefficients could be estimated by iteratively maximizing the probability of held–out data (deleted interpolation), using the forward–backward, algorithm (see, for instance [**Cha93**]). Nevertheless, for the sake of simplicity, the weights have been experimentally–tuned and set to $\lambda_1 = 0.5$, $\lambda_2 = 0.29$, $\lambda_3 = 0.19$, $\lambda_4 = 0.02$ for the following experiments and for those reported in chapter 6.

From now on, we will refer to STT as STT$^+$ when using n–gram information (this name will frequently appear in chapter 6). The results obtained using STT$^+$ are better than those of STT in both the English and Spanish corpora, providing

evidence for the utility of the presented approach. These results are summarized in the two following points.

- When applied to the Spanish corpus, apart from correctly disambiguating the example sentence, STT$^+$ increased the accuracy of STT up to 97.69%. This figure was achieved by applying a simple strategy of backing off to more general information sources whenever there is no samples in the current level (i.e., to assign one to the 'λ_i' corresponding to the most specific information source for which there exist evidence, and zero to the rest of 'λ's). Slightly better results where obtained, 97.81% accuracy, when using the hand tuned set of weights for the λ_i parameters. Recall that the accuracy of simple STT was 97.51% on the same corpus.
- As we can see in table 4 of chapter 6, STT$^+$, which obtained an accuracy of 96.84%, also improved the 96.61% result of STT.

4. Using the Tree–model in a Flexible Tagger

From a different perspective, decision trees may be seen as a compact way of representing a set of classification rules. The transformation from trees to rules is easily performed by converting each path from the root to the leaves into a single rule. In our case, we take advantage from the probabilistic information of leaf nodes to construct a set of weighted constraints (the weight is represented by the mutual information between tags and contexts), to feed RELAX [**Pad96**], a flexible–model tagger based on relaxation labelling (RL). This approach was first published in [**MP97**], and lately applied to the annotation of a Spanish corpus [**CCM$^+$98**].

We start by describing the main features of the RELAX POS tagger (section 4.1), and then we address the inclusion of the decision trees in it (section 4.2). Regarding this last point, we will see that results obtained by combining trees, n–grams and linguistic rules surpass any other combination.

4.1. RELAX: A Relaxation–labelling based Tagger. Relaxation is a generic name for a family of iterative algorithms which perform function optimization, based on local information. They are closely related to neural nets [**Tor89**] and gradient descent [**LM95a**].

Although relaxation operations had long been used in engineering fields to solve systems of equations [**Sou40**], they did not achieve their breakthrough success until relaxation labelling —their extension to the symbolic domain— was applied to the field of constraint propagation, especially in low–level vision problems [**Wal75, RHZ76**].

Relaxation labelling is a technique that can be used to solve consistent labelling problems (CLPs), as described by [**LM95b**]. A consistent labelling problem consists of, given a set of variables, assigning to each variable a value compatible with the values of the other ones, satisfying —to the maximum possible extent— a set of compatibility constraints.

In the Artificial Intelligence field, relaxation has been mainly used in computer vision —since it is where it was first used— to address problems such as corner and edge recognition or line and image smoothing [**RLS81, Llo83**]. Nevertheless, many traditional AI problems can be stated as a labelling problem: the travelling salesman problem, n–queens, or any other combinatorial problem [**AK87**].

The application of constraint satisfaction to perform NLP tasks is not a novel idea. The relaxation labelling algorithm in particular was first proven to be useful

for such tasks by [**PR94, PM94**], who used POS tagging as a toy problem to test some methods to improve the computation of constraint compatibility coefficients for relaxation processes. Applications to real NLP problems (including POS tagging, shallow parsing and word sense disambiguation), dealing with large unrestricted texts, are presented in the following works: [**Pad96, VP97, MP97, Pad98**].

The tagger we present here has the architecture described in figure 5: A unique algorithm uses a language model consisting of constraints obtained from different knowledge sources[9]. The disambiguation algorithm, which works by relaxation labelling, applies the constraints contained in the language model in order to iteratively update the weights for each possible label for each word. If constraints are consistent, the algorithm converges to a local optimum which satisfies as much as possible the constraint set. For a deeper discussion on the convergence of the algorithm, see [**Pad98**].

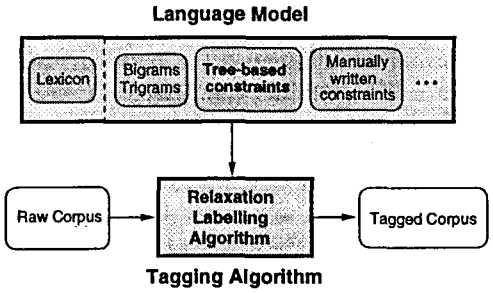

FIGURE 5. Architecture of RELAX tagger

From our point of view, the most remarkable feature of the algorithm is that, since it deals with context constraints, the model it uses can be improved by writing into the constraint formalism any available knowledge. The constraints used may come from different sources: statistical acquisition, machine–learned models or hand coding. An additional advantage is that the tagging algorithm is independent of the complexity of the model.

We summarize the main features of the RL based tagger in the following points:

- It uses a constraint oriented language model. This enables it to deal with many kinds of information (n–grams, decision tree branches, linguistic information, etc.) provided they are expressed in the form of constraints.
- It performs parallel constraint satisfaction, that is, the constraints are not applied in a predefined order.
- It enables the use of heterogeneous information. For instance, to perform POS tagging, one can use constraints on the POS tags for words in context, but also their morphological, semantic o syntactic features, if available.
- It enables the simultaneous resolution of several disambiguation tasks. For instance, by choosing among a set of pairs (*tag,sense*) instead of among a set of tags, one can perform simultaneously POS tagging and WSD, and use constraints which take advantage of the cross information between both tasks.

[9]The type of constraint formalism will be roughly described in the next section.

For more details on the algorithm and its application to different NLP tasks, such as WSD or shallow parsing, see [**Pad98**]. Additionally, appendix B contain a detailed description of the relaxation labelling algorithm.

4.2. Incorporating Decision Trees into RELAX. In order to feed the RELAX tagger with the language model acquired by the decision–tree learning algorithm, the group of trees for the 44 most representative ambiguity classes (covering 83.95% of the examples) were translated into a set of weighted context constraints. RELAX was fed not only these constraints, but also with bi/tri–gram information.

The Constraint Grammar formalism [**KVHA95**] was used to code the tree branches. CG is a widespread formalism used to write context constraints for developing linguistic–based grammars for morphosyntactic disambiguation and shallow parsing. Since it is able to represent any regular context pattern, we will use it to represent all our constraints: n–gram patterns, hand–written constraints, or decision–tree branches.

Since the CG formalism is intended for linguistic uses, the statistical contribution has no place in it: Constraints can state only full compatibility (constraints that SELECT a particular reading) or full incompatibility (constraints that REMOVE a particular reading). Thus, we slightly extended the formalism to enable the use of real–valued compatibilities, in such a way that constraints are not assigned a REMOVE/SELECT command, but a real number indicating the compatibility between the reading and the context represented in the constraint. This measure, which should reflect the association between the two components of the constraint, was computed as the *mutual information* between the focus tag and the context.

Although the compatibility values for each constraint could be computed in different ways, recent experiments [**Pad96, Pad98**] indicated that the best results in our domain are obtained when computing compatibilities as the mutual information between the tag and the context. Mutual information measures how informative is a discrete random variable with respect to another, and is computed as the expectation of the expression in (2) for every possible pair of values [**CT91**]. Since we are interested on events rather than on distributions, we will use the corresponding expression for the outcomes A and B rather than its expectation [**KS97**].

$$MI(A,B) = log\frac{P(A,B)}{P(A)\cdot P(B)} \tag{2}$$

If A and B are independent events, the conditional probability of A given B will be equal to the marginal probability of A and the measurement will be zero. If the conditional probability is larger, it means than the two events tend to appear together more often than they would by chance, and the measurement yields a positive number. Inversely, if the conditional occurrence is scarcer than chance, the measurement is negative. Although Mutual information is a simple and useful way to assign compatibility values to our constraints, a promising possibility still to be explored is assigning them by Maximum Entropy Estimation [**Ros94, Rat97b, Ris97**].

4.2.1. *Including n–gram Constraints.* The translation of bi/tri–grams to context constraints is straightforward. A left prediction bigram and its right prediction counterpart would be, for instance:

```
4.21  (NN)          3.82  (DT)
      (-1 DT);            (1 NN);
```

The training corpus contains 1,404 different bigrams. Since they are used both for left and right prediction, they are converted in 2,808 binary constraints.

A trigram may be used in three possible ways (i.e. the ABC trigram pattern generates the constraints: C, given it is preceded by AB; A, given it is followed by BC; and B, given it is preceded by A and followed by C). A real example corresponding to the 'DT NN VB' trigram, would be:

```
2.16  (VB)          1.54  (NN)          1.82  (DT)
      (-2 DT)             (-1 DT)             (1 NN)
      (-1 NN);            (1 VB);             (2 VB);
```

In this case, the 17,387 tri-gram patterns in the training corpus produced 52,161 ternary constraints.

4.2.2. *Translating Decision Trees.* The usual way of expressing trees as a set of rules was used to construct the context constraints. For instance, the tree branch represented in figure 3 of chapter 3 was translated into the two following constraints:

```
-5.81 (IN)              2.366 (RB)
      (0 "as" "As")           (0 "as" "As")
      (1 RB)                  (1 RB)
      (2 IN);                 (2 IN);
```

which express the compatibility (either positive or negative) of the tag in the first line with the given context (i.e. the focus word is "as", the first word to the right has tag RB and the second has tag IN).

The decision trees acquired for the 44 most frequent ambiguity classes resulted in a set of 8,473 constraints.

4.2.3. *Including Linguistic Information.* The main advantage of RELAX is its ability to deal with constraints of any kind. This enables us to combine statistical n-grams (written in the form of constraints) with the learned decision tree models, and even with linguistically motivated hand-written constraints, such as the following,

```
10.0  (VBN)
      (*-1 VAUX BARRIER (VBN) OR (IN) OR (<,>) OR
                        (<:>) OR (JJ) OR (JJS) OR (JJR));
```

which states a high compatibility value for a VBN (participle) tag when preceded by an auxiliary verb, provided that there is no other participle, adjective nor any phrase change in between.

4.2.4. *Evaluation.* The results obtained using the different knowledge combination are shown in table 5. The results produced by two baseline taggers —MFT_2: most-frequent-tag tagger, HMM: bi-gram Hidden Markov Model tagger by Elworthy [**Elw93**]— are also reported. B stands for bigrams, T for trigrams, and C for the constraints acquired by the decision tree learning algorithm. Results using a sample of 20 linguistically-motivated constraints (H) can be found in table 6. These constraints deal with the participle-past tense ambiguity (1 constraint), with the noun-adjective ambiguity (2 constraints) and with special constructions using particles such as "about" (4 constraints), "as" (7), "up" (3), "out" (2) and "more" (1).

	MFT_2	HMM	B	T	BT	C	BC	TC	BTC
Ambig.	85.31	91.75	91.35	91.82	91.92	91.96	92.72	92.82	92.55
Overall	94.66	97.00	96.86	97.03	97.06	97.08	97.36	97.39	97.29

TABLE 5. Accuracy results (%) of the baseline taggers and of the RELAX tagger using every combination of constraint kinds

Since the cost of the algorithm depends linearly on the number of constraints, the use of the trigram constraints (either alone or combined with the others) makes the disambiguation about six times slower than when using BC, and about 20 times slower than when using only B.

	H	BH	TH	BTH	CH	BCH	TCH	BTCH
Ambig.	86.41	91.88	92.04	92.32	91.97	92.76	92.98	92.71
Overall	95.06	97.05	97.11	97.21	97.08	97.37	97.45	97.35

TABLE 6. Accuracy results (%) of our tagger using every combination of constraint kinds plus the hand written constraints

The above results show that the addition of the automatically acquired context constraints lead to an improvement in the accuracy of the tagger, overcoming the bi/tri-gram models and properly cooperating with them. See also figure 6, which complements table 5 by depicting the 95% confidence intervals for the accuracy figures obtained by the different combinations.

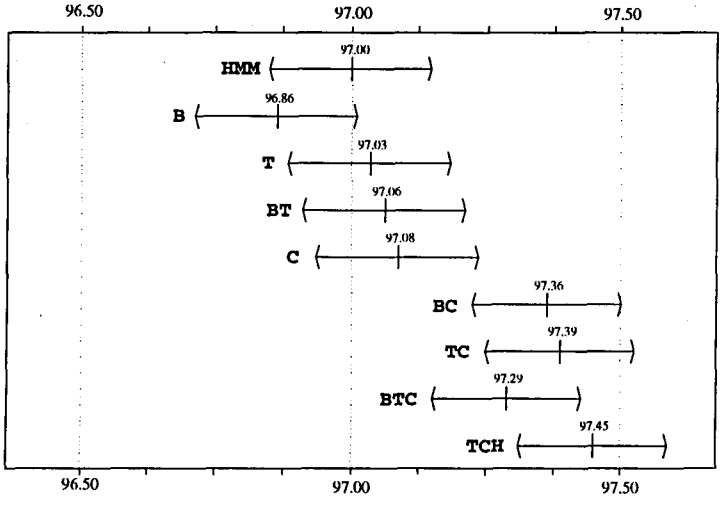

FIGURE 6. 95% confidence intervals for the RELAX tagger results

The main conclusions that can be drawn from those data are described below.

- RELAX is slightly worse than the HMM tagger when using the same information (bi-grams). This may be due to a higher sensitivity to noise in the training corpus.

- There are two significantly distinct groups: Those using only statistical information, and those using statistical information plus the decision trees model. The n-gram models and the learned model belong to the first group, and any combination of a statistical model with the acquired constraint belongs to the second group.
- Although the hand-written constraints improve the accuracy of any model, the size of the linguistic constraint set is too small to make this improvement statistically significant. More work should be performed around the design of a more complete linguistic model to properly test the collaboration of linguistic information within the RELAX environment.
- The combination of the two kinds of model produces significantly better results than any separate use. This indicates that each model contains information which was not included in the other, and that relaxation labelling combines them properly.

4.3. Using Small Training Sets. In this last section we will discuss the results obtained when using the previously described taggers to apply the language models learned from small training corpus.

The motivation for this analysis is the need for determining the behaviour of our taggers when used with language models coming from scarce training data, in order to best exploit them for the development of Spanish and Catalan tagged corpora starting from scratch. This issue will be addressed in the next chapter.

In particular, we used 50,000 words of the WSJ corpus to learn a set of decision trees and to collect bigram statistics. Trigram statistics were not considered since the size of the training corpus was not large enough to reasonably estimate the big number of parameters of the model. Note that a 45-tag tagset produces a trigram model of over 90,000 parameters, which obviously cannot be estimated from a set of 50,000 occurrences.

Using this training set the learning algorithm was able to reliably acquire over 80 trees representing the most frequent ambiguity classes (note that the training data was insufficient for learning sensible trees for about 150 ambiguity classes). Following the formalism described in the previous section, we translated these trees into a set of about 4,000 constraints to feed the relaxation labelling algorithm.

The results obtained are presented in table 7. Those figures are computed as the average of ten experiments using randomly chosen training sets of 50,000 words each. B stands for the bigram model and C for the learned decision trees when translated to context constraints.

	MFT_2	RTT	RELAX(C)	RELAX(B)	RELAX(BC)
Ambig.	75.35%	87.29%	86.29%	87.50%	88.56%
Overall	91.64%	95.69%	95.35%	95.76%	96.12%

TABLE 7. Comparative results using different models acquired from small training corpus

The presented figures may lead to the following conclusions:

- We think this result is quite accurate. In order to corroborate this statement we can compare our accuracy of 96.12% with the 96.0% reported by [**DZBG96**] for the IGTREE Tagger trained with a double size corpus (100

Kw). In the next chapter we will see how this promising results also hold for Spanish.

- RTT yields a higher performance than the RELAX tagger when both use only the C model. This is caused by the fact that, due to the scarceness of the data, a significant amount of test cases do not match any complete tree branch, and thus RTT uses some intermediate node probabilities. Since only complete branches are translated to constrains —partial branches were not used to avoid excessive growth in the number of constraints—, the RELAX tagger does not use intermediate node information and produces lower results. A more exhaustive translation of tree information into constraints is an issue that should be studied.
- The RELAX tagger using the B model produces better results than any of the taggers when using the C model alone. The cause of this is related with the aforementioned problem of estimating a big number of parameters with a small sample. Since the model consists of six features, the number of parameters to be learned is still larger than in the case of trigrams, thus the estimation is not as complete as it should be.
- The RELAX tagger using the BC model produces better results (statistically significant at a 95% confidence level) than any other combination. This suggests that, although the tree model is not complete enough on its own, it contains different information than the bigram model. Moreover this information is proved to be very useful when combined with the B model by RELAX.

CHAPTER 5

Spanish Part–of–speech Tagging

This chapter describes the work performed around the morphosyntactic annotation of the LEXESP Spanish corpus. It is divided into two parts. The first is introductory and it is devoted to describe the corpus, the morphological analyzer, and the adaptation to Spanish of the taggers presented in chapter 4. The main contributions are in the second part, in which we show how to improve the tagging performance obtained in the previous section.

More particularly, we present a bootstrapping method to develop an annotated corpus, which consists of taking advantage of the collaboration of two different POS taggers. The cases in which both taggers agree present a higher accuracy and are used to retrain the taggers. This method, which is especially useful for languages with few available resources, has been applied to obtain the final annotation of the corpus.

1. Tagging the LEXESP Copus

The LEXESP Project is a multi–disciplinary effort headed by the Psychology Department of the University of Barcelona in collaboration with the Psychology Department of the University of Oviedo. It aims to create a large database of language usage in order to enable and encourage research activities in a wide range of fields, from linguistics to medicine, through psychology and artificial intelligence, among others. One of the main issues of this database of linguistic resources is the LEXESP corpus, which contains 5.5 Mw of written material, including general news, sports news, literature, scientific articles, etc., and which aims to be a balanced and general sample of modern Spanish language usage.

The corpus will be morphologically analyzed and disambiguated as well as syntactically parsed. The tagset used for morphosyntactic annotation is PAROLE compliant, and consists of some 230 tags[1] in their fully expanded form (i.e. using all information about gender, number, person, tense, etc.), which can be reduced to 62 tags when only category and subcategory are considered.

The annotation of the corpus is being performed within a more general framework, supported by two research projects, consisting of the creation and integration of several tools and resources for automatic morphosyntactic analysis, tagging and parsing of unrestricted Spanish text, as well as their application to related NLP tasks, such as information extraction. The research projects involved are: ITEM (ref: TIC96–1243–C03–02, Spanish Research Department) and LEXESP (ref: APC96–0125). See appendix E for details about these projects, about the research partners, and about the links to the available tools and resources.

[1] There are potentially many more possible tags, but they do not actually occur.

The syntactic analysis of the LEXESP corpus is currently being performed, and therefore it will not be explained here[2]. The morphological analysis and the subsequent disambiguation is explained in the two following sections.

1.1. The MACO+ Morphological Analyzer. MACO+ is a general purpose morphological analyzer for unrestricted Spanish text developed at the Natural Language Research Groups belonging the Software Department of the Polytechnical University of Catalonia, as an evolution of a previous MACO analyzer [**AAC+94**].

The architecture of the morphological analyzer is a modular pipeline of specialized recognizers, as showed in figure 1.

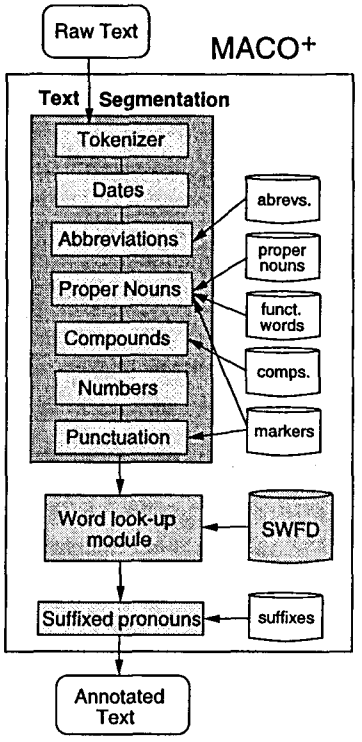

FIGURE 1. General architecture of MACO+

The first block of modules, labelled *Text Segmentation*, performs a proper segmentation of the text, labelling the punctuation marks and joining groups of words identified as one lexical unit (e.g. proper nouns or compounds such as *'aparte de'*, *'sin embargo'*), date or numerical expressions, etc. They are a collection of specific heuristics to identify the following special items:

- Simple date patterns: '23/3/79', *'año* 1983', '13 *de diciembre'*, '30 *de julio de* 1993', ...
- Abbreviations: *cm.*, *MHz.*, *Sr.*, ...
- Proper nouns: *'María Elena'*, *'San Cristóbal de las Casas'*, *'Ministerio de Cultura'*, *'Universidad de Lodz'*, ...

[2]A robust partial parser for shallow bracketing is already developed and available.

- Multi–word compounds: *'sin embargo'*, *'no obstante'*, ...
- Numbers and numerical expressions: '12,12', 11.000, 1-3-1, 33942206-S, ...
- Punctuation marks.

These modules use a set of files containing compilations of typical abbreviations, proper nouns (personal, geographical, marks, enterprises, etc.), multi–word compounds, functional words allowed to be inside compounds, punctuation marks, and so on.

Modules can be activated or deactivated for each particular analysis and, obviously, heuristics in each module can be improved independently.

All the tokens not recognized by any of the preceding modules are pipelined to the word look–up module, which is the *real analyzer*, containing fast algorithms for retrieving information from a Spanish comprehensive word form dictionary, which is labelled SWFD in figure 1. This dictionary has about one million entries (verbs, nouns, adjectives and adverbs), containing for each form the lemma and a PAROLE compliant morphological tag describing information such as category, subcategory, gender, number, person, mode, etc. The dictionary was constructed by applying a set of about 400 semi–automatically derived inflectional rules to a big root dictionary —consisting of about 12,000 verbal roots, 85,000 nominal and adjectival roots and 3,000 closed–category roots— which was automatically extracted from existing MRD's and available corpora. For more details, we refer the reader to [CCM+98].

Finally, a post process is performed on the non-recognized words in order to identify verbal forms with suffixed pronouns (named *clíticos*). This is a type of pronouns that are added as suffixes to the verb forms, acting usually as syntactic objects. For instance, the form *dándonosla* ('giving it (*fem.*) to us') has two suffixed pronouns: '-nos' and '-la' indicating first person plural indirect object and 3rd person singular feminine direct object, respectively. These particular forms were not generated and included in the dictionary because there exist potentially infinite combinations due to the possible recursive application of suffixes. Even restricting to the combinations of two pronouns, which is a realistic simplification, would result in an unfeasible increase of the dictionary[3].

An implementation of MACO+, using PERL-5.003, was run (with all modules activated) on a SUN UltraSparc2 machine with 194Mb of RAM to analyze the whole LEXESP corpus. It took 2.54 hours (including input/output processing time) to analyze the 5.5 million words at an average speed of 600 words/sec[4].

The main results of the morphological analysis are the following:

- The resulting coverage was about 99.5%, which is a remarkable figure working on an unrestricted text as LEXESP. Those words that remained unrecognized after the pipeline were labelled as *unknown*.
- The recall —i.e., the words that get the correct tag among the proposed— is estimated to be 99.3%.
- Using the reduced tagset of 62 tags, the percentage of ambiguous words is 39.26% and the average ambiguity ratio is 2.63 tags per word for the

[3]Further extensions to deal with the most productive classes of derivation and other morphological variations are in progress.

[4]Comparatively, the time that would take running on a Pentium-120/24Mb architecture was estimated to be 7.64 hours, at an average speed of 200 words/sec.

ambiguous words, 1.64 overall. These figures on ambiguity are slightly higher than those observed on the WSJ English corpus (see chapter 4).

1.2. Adapting the English POS Taggers. The ambiguous output produced by MACO+, was used as the input for the POS taggers described in chapter 4, namely, RELAX, RTT, and STT.

Since both taggers require training data, 96 Kw from the morphological analyzed corpus, were hand disambiguated to get an initial training set ($\mathcal{C}^0$) of 71 Kw and a test set ($\mathcal{T}$) of 25 Kw.

It has to be noted that the size of the training set is much lower than the usual one million word training corpus derived from the WSJ. This is a common problem when dealing with languages with a reduced amount of available linguistic resources, since the manual tagging of a big enough training corpus is very expensive, both in time and human labour[5].

The reduced set of tags was used for the tagging purpose —the use of the whole set of over 200 tags would have unrealistically increased the number of parameters of the model, and therefore it would have aggravated the sparseness problem— which means taking into account the 'category' and 'subcategory' information of the PAROLE labels, and ignoring the other features. Due to this simplification, some ambiguities cannot be resolved since they do not appear at this level of granularity. For instance, 1st and 3rd person (singular) of the past continuous verbs share the same form in Spanish: *corría* both means 'I' or 'he/she' was running. A similar situation occurs with noun word forms that have a different sense depending on the genre: *capital*, which means an amount of money (masc.) and the main city of a country (fem.).

Therefore, these readings, which proportion on the LEXESP corpus is slightly over 2%, are not differentiated and they are left ambiguous after the tagging process. This type of ambiguity provokes an almost negligible effect on some NLP applications, e.g., Information Retrieval, however it should be properly addressed, by a kind of post-tagging process, or by the parser itself, if the acquisition of deeper linguistic knowledge was involved in the task at hand.

The training set was used to extract bigram and trigram statistics and to learn decision trees to apply to RTT and STT. The taggers also require lexical probabilities, which were computed from the occurrences in the training corpus — applying smoothing (Laplace correction) to avoid zero probabilities— For the words not occurring in the training set, the probability distribution for their ambiguity class[6] was used. For unseen ambiguity classes, unigram probabilities were used. This is a very simple smoothing procedure with a back-off strategy.

Initial experiments consisted of evaluating the precision of both taggers when trained on the above conditions. Table 1 shows the results produced by each tagger. The different kinds of information used by the relaxation labelling tagger are coded as follows: B stands for bigrams, T for trigrams and BT for the joint set. A baseline result produced by a most-frequent-tag tagger (MFT) is also reported.

---

[5] A trained human annotator reached a rate of 2000 words per hour (which corresponds to an average of five seconds per ambiguous word), using a especially designed Tcl/Tk graphical interface. So, the 100Kw were annotated in about 50 man hours.

[6] The number of ambiguity classes appearing in the training set was 137, while those given by the morphological analyzer are 164.

Tagger Model	Ambig.	Overall
MFT	88.9%	95.8%
RTT	92.1%	97.0%
STT	92.7%	97.2%
STT$^+$	93.2%	97.4%
RELAX(B)	92.9%	97.3%
RELAX(T)	92.7%	97.2%
RELAX(BT)	93.1%	97.4%

TABLE 1. Results of different taggers using the $\mathcal{C}^0$ training set

From these results, we may conclude that a 71 Kw training set manually disambiguated provides enough evidence to allow the tagger to get quite good results. Nevertheless, it is interesting to notice that the trigram model has lower accuracy than the bigram model. This is caused by the size of the training corpus, which is too small to estimate a good trigram model. In the same direction, the sparseness of data, which is more important at the ambiguity–class level, negatively affects the tree–based model making RTT and STT to perform worse than the bigram–based tagger. More evidence on this direction is given by the fact that STT$^+$ (which uses n–gram information by interpolation) corrects this deviation and gets better results.

The absolute accuracy figures are comparable to that obtained when tagging the WSJ corpus (note that both corpora have similar ambiguity rates, and that the granularity of the tagsets is also similar). Nevertheless, considering that the training corpus was significantly smaller the results are fairly better[7].

2. Improving Accuracy by Combining different Taggers

2.1. Motivation. Usual automatic tagging algorithms involve a process of acquisition of a statistically–based language model from a previously tagged training corpus. The statistical models contain lots of parameters that have to be reliably estimated from the corpus, so the sparseness of the training data is a severe problem.

When a new annotated corpus for a language with a reduced amount of available linguistic resources is developed, this issue becomes even more important, since no training corpora are available and the manual tagging of a big enough training corpus is very expensive, both in time and human labour. If costly human labour is to be avoided, the accuracy of automatic systems has to be as high as possible, even starting with relatively small manually tagged training sets.

As we saw in chapter 2, some methods have been developed to avoid the need of fully supervised training. In POS tagging we find the Baum–Welch re–estimation algorithm which has been successfully used to improve tagger accuracies when limited disambiguated material is available **[CKPS92, Elw94, Mer94]**. Also Brill **[Bri95b]** presented a weak–supervised version of the transformation–based learning algorithm for tagging. Recently, similar techniques that use unsupervised data

[7] In another experiment, reported in section 4.3 of chapter 4, we trained the English taggers using a small set of about 50,000 words. The accuracy obtained on tagging the WSJ corpus was 96.12%.

to aid supervised training have been applied in the domain of text categorization [**BM98, NMTM98**].

Bootstrapping is one of the methods that can be used to improve the performance of statistical taggers when only small training sets are available[8]. The bootstrapping procedure starts by using a small hand-tagged portion of the corpus as an initial training set for a certain POS tagger. Then, the tagger is used to disambiguate further material, which is incorporated to the training set and used to retrain the tagger. Since the retraining corpus can be much larger than the initial training corpus we expect to better estimate the statistical parameters of the tagging model and to obtain a more accurate tagger. Of course, this procedure can be iterated leading, hopefully, to progressively better language models and more precise taggers. The procedure ends when no more improvement is achieved.

According to the above formulation, the bootstrapping refining process is completely automatic. However each step of training corpus enlargement and enrichment could involve a certain amount of manual revision and correction. In this way the process would be semi–automatic.

The main problem of this approach is that the retraining material contains errors (because it has been tagged with a still poor tagger) and that this introduced noise could be very harmful for the learning procedure of the tagger. Depending on the amount of noise and on the robustness of the tagging algorithm, the refining iteration could lead to no improvement or even to a degradation of the performance of the initial tagger. Therefore, keeping a low error rate in retraining material becomes an essential point if we want to guarantee the validity of the bootstrapping approach.

We will propose the joint use of two taggers as a way to reduce the error rate introduced by a single tagger by selecting as retraining material only those cases in which both taggers coincide. This approach relies on the observation that the cases in which both taggers propose the same tag present a much higher precision than any of them separately, and that these coincidence cases represent a very high coverage. Then, the corpus to retrain the taggers is built, at each step, on the basis of this intersection corpus, keeping fairly low error rates and leading to better language models and more precise taggers.

In addition, it is clear that the combination of taggers can be used to get a high recall tagger, which proposes an unique tag for most words and two tags when both taggers disagree. Depending on the user needs, it might be worthwhile accepting a higher remaining ambiguity in favour of a higher recall.

2.2. Bootstrapping algorithm. The proposed bootstrapping algorithm is described in detail in figure 2. The meaning of the involved notation is explained below:

- $\mathcal{C}^i$ stands for the retraining corpus of i-th iteration. In particular, $\mathcal{C}^0$ stands for the initial hand–tagged training corpus.
- $\mathcal{T}$ stands for a hand–tagged test corpus used to estimate the performance of the subsequent taggers.

[8] In the case of English, existing resources are usually enough, thus current work on developing corpora does not rely much in bootstrapping, although re–estimation procedures are widely used to improve tagger accuracies [**Chu88, BCPS94, Elw94**].

```
### Train taggers using manual corpus
M_1^0 := train(A_1, C^0);
M_2^0 := train(A_2, C^0);
### Compute achieved accuracy
Acc-current := test(T, A_1, M_1^0, A_2, M_2^0);
Acc-previous := 0;
### Initialize iteration counter
i := 0;
while (Acc-current significantly-better Acc-previous) do
    N := fresh-part-of-the-raw-corpus;
    ### Tag the new data
    N_1^i := tag(N, A_1, M_1^i);
    N_2^i := tag(N, A_2, M_2^i);
    ### Add the coincidence cases to
    ### the manual training corpus
    C^{i+1} := combine(C^0, N_1^i ∩ N_2^i);
    ### Retrain the taggers
    M_1^{i+1} := train(A_1, C^{i+1});
    M_2^{i+1} := train(A_2, C^{i+1});
    ### Prepare next iteration
    Acc-previous := Acc-current;
    Acc-current := test(T, A_1, M_1^{i+1}, A_2, M_2^{i+1});
    i := i+1
end-while
```

FIGURE 2. Bootstrapping algorithm using two taggers

- $\mathcal{N}$ stands for the fresh part of the raw corpus used at each step to enlarge the training set. For simplicity we consider it independent of the specific iteration.
- A_1 and A_2 stand for both taggers (including, indistinctly, the model acquisition and disambiguation algorithms).
- M_i^j stands for the j-th language model obtained by i-th tagger.
- $train(A_i, \mathcal{C}^j)$ stands for the procedure of training the i-th tagger with the j-th training corpus. The result is the language model M_i^j.
- $test(\mathcal{T}, A_1, M_1^i, A_2, M_2^i)$ stands for a general procedure that returns the best accuracy obtained by any of the two taggers on the test set.
- $tag(\mathcal{N}, A_i, M_i^j)$ stands for the procedure of tagging the raw corpus $\mathcal{N}$ with the i-th tagger using the j-th language model, producing $\mathcal{N}_j^i$.
- $combine(\mathcal{C}^0, \mathcal{N}_1^i \cap \mathcal{N}_2^i)$ is the general procedure of creation of (i+1)-th training corpus. This is done by adding to the hand disambiguated corpus $\mathcal{C}^0$ the cases from $\mathcal{N}$ in which both taggers coincide in their predictions (noted by $\mathcal{N}_1^i \cap \mathcal{N}_2^i$).

Two properties must hold for this method to work: (1) the accuracy in the cases of coincidence should be higher than those of both taggers individually considered, and (2) the taggers should coincide in the majority of the cases (high coverage).

We empirically checked that these properties hold in our case, and that larger training corpora with a fairly low error can be constructed using the combination

of RELAX and RTT taggers[9]. For instance, using a first set of 200Kw ($\mathcal{N}$) and given that both taggers agree in 97.5% of the cases and that 98.4% of these cases are correctly tagged (as it is shown in table 5), we get a new corpus of 195Kw with an error rate of 1.6%. Additionally, If we add the manually tagged 70Kw (assumed error free) from the initial training corpus we get a 265Kw corpus with a 1.2% error rate.

2.3. Applying and Evaluating the Bootstrapping Algorithm. In this section we study the proper tuning of the algorithm in our particular domain by exploring the right size of the retrain corpus (i.e: the size of $\mathcal{N}$), the combination procedure (in particular we explore if a weighted combination is preferable to the simple addition) and the number of iterations that are useful. Additionally, we test if the relatively cheap process of hand-correcting the disagreement cases between the two taggers at each step can give additional performance improvements.

2.3.1. *Size of the Retraining Corpus.* First of all, we need to establish which is the right size for the fresh part of the corpus to be used as retraining data. We have 5.4Mw of raw data available to do so, but note that the bigger the corpus is, the higher the error rate in the retraining corpus will be —because of the increasing proportion of new noisy corpus with respect to the initial error free training corpus— Therefore, we will try to establish which is the corpus size at which further enlargements of the retraining corpus do not provide significant improvements. For doing so, we proceed in several steps increasing the retraining set by approximately 200,000 examples at each step. These corpora are denoted by $\mathcal{C}^1_{200}$, $\mathcal{C}^1_{400}$, $\mathcal{C}^1_{600}$, etc., respectively. The main characteristics of these and other corpora appearing in the chapter are summarized in table 2.

Cp	Words	%Err	%Cov	Big	Trig	Exs	AC
$\mathcal{T}$	25,128	0	—	—	—	—	—
$\mathcal{C}^0$	71,833	0	100 / 100	1,297	7,643	27,398	137
$\mathcal{C}^1_{200}$	292,118	1.22	93.91 / 97.70	1,796	14,352	94,394	152
$\mathcal{C}^1_{400}$	507,017	1.40	"	2,003	18,006	161,573	155
$\mathcal{C}^1_{600}$	726,311	1.47	"	2,120	20,607	229,325	155
$\mathcal{C}^1_{800}$	946,057	1.50	"	2,209	22,507	295,637	155
$\mathcal{C}^1_{1000}$	1,165,432	1.53	"	2,267	24,221	363,907	156
$\mathcal{C}^1_{M}$	292,118	1.17	100 / 100	1,890	15,402	108,016	149
$\mathcal{C}^2$	507,017	1.40	95.59 / 98.33	2,013	18,327	166,122	155
$\mathcal{C}^1_{Best}$	946,057	1.46	95.13 / 98.16	2,274	23,242	309,794	155

TABLE 2. Information about the test, training and all retraining sets. Cp: corpus; Words: number of words; %Err: estimated error rate; %Cov: estimated coverage of the intersection on ambiguous-words/overall; Big: number of different bigrams; Trig: number of different trigrams; Exs: number of examples to learn the tree base; and AC: number of ambiguity classes. All retraining corpora contain $\mathcal{C}^0$. Coverage figures are calculated excluding $\mathcal{C}^0$

[9] We used RTT instead of STT for the combination since when the experiments were performed the implementation of STT was still in progress.

The results for each tagger when retrained with different corpus sizes are shown in figure 3. Accuracy figures are given over ambiguous words only[10], and they are computed retraining the taggers with the coincidence cases in the retrain corpus, as previously described in section 2.2.

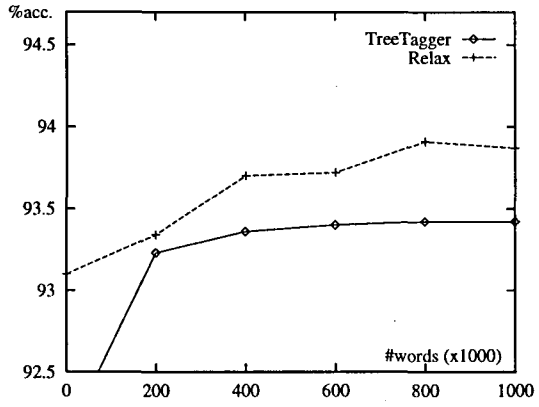

FIGURE 3. Results using retraining sets of increasing sizes

It can be observed that the size at which both taggers produce their best result is that of 800 Kw (namely $\mathcal{C}^1_{800}$), achieving 93.4% and 93.9% accuracy.

2.3.2. *Two Taggers Better than One.* Once we have chosen a size for the retraining corpus, we will check whether the joint use of two taggers to reduce the error in the training corpus is actually better than retraining only with a single tagger.

Comparative results obtained for each of our taggers when using retraining material generated by a single tagger (the size of the fresh part of the corpus to be used as retrain data was also 800 Kw) and when using $\mathcal{C}^1_{800}$ are reported in table 3. Those results point that the use of two taggers to generate the retraining corpus, slightly increases the accuracy of any tagger since it provides a less noisy model.

Tagger Model	single	$\mathcal{C}^1_{800}$
RTT	93.0%	93.4%
RELAX(BT)	93.7%	93.9%

TABLE 3. Comparative accuracy figures when retraining with a new 800Kw corpus

The error rate in the retrain corpus when using the RELAX(BT) tagger alone is 2.4%, while when using the coincidences of both taggers is reduced to 1.4%. This improvement in the training corpus quality enables the taggers to learn better models and slightly improve their performance. Probably, the cause that the performance improvement is not larger must not be sought in the training corpus error rate, but in the learning abilities of the taggers.

[10]Unless the contrary is explicitly said, accuracy results given all along the chapter will refer to ambiguous words.

2.3.3. *Number of Iterations.* The bootstrapping algorithm must be stopped when no further improvements are obtained. This seems to happen after the first iteration step. Using the 800Kw from the beginning yields similar results than progressively enlarging the corpus size at each step. Results are shown in table 4.

Tagger Model	C^1_{800}	C^2_{800}
RTT	93.4%	93.5%
RELAX(BT)	93.9%	93.8%

TABLE 4. Results when retraining with a 800Kw corpus in one and two steps

Facts that support this conclusion are:

- The variations with respect to the results for one re–estimation iteration are not significant.
- RTT gets a slight improvement while RELAX decreases —indicating that the re–estimated model does not provide a clear improvement in all cases—
- One reason that explains this situation is that the intersection corpora used to retrain taggers have roughly the same accuracy in both iterations (98.37% in iteration one vs. 98.44% in iteration two), while the positive difference in the number or coincidences (97.70% in iteration one vs. 98.33% in iteration two) is not large enough to provide extra information. A complete description of the retraining corpora, including the performance of the combination of taggers, is presented in table 5.

		Ambig.	Overall
1st retrain:	t_1=t_2	93.91%	97.70%
	t_1=OK *and* t_2=OK *given* t_1=t_2	95.51%	98.37%
	t_1=OK *or* t_2=OK *given* $t_1 \neq t_2$	96.36%	96.36%
	Combined recall	95.57%	98.32%
	Combined precision	90.08%	96.11%
2nd retrain:	t_1=t_2	95.59%	98.33%
	t_1=OK *and* t_2=OK *given* t_1=t_2	95.77%	98.44%
	t_1=OK *or* t_2=OK *given* $t_1 \neq t_2$	96.17%	96.17%
	Combined recall	95.79%	98.41%
	Combined precision	91.74%	96.79%

TABLE 5. Relevant figures about the intersection corpus of the two first iterations. 't_1' and 't_2' stand for the two POS taggers; 't_1=OK' stands for the cases that the first tagger correctly disambiguates; 't_1=t_2' stands for the cases in which both taggers agree

2.3.4. *Use of Weighted Examples.* We have described so far how to combine the results of two POS taggers to obtain larger training corpora with low error rates. We have also combined the agreement cases of both taggers with the initial hand–disambiguated corpus, in order to obtain a less noisy training set. Since the hand–disambiguated corpus offers a higher reliability than the tagger coincidence set, we might want to establish a *reliability* degree for our corpus, by means of

controlling the contribution of each part. This can be done through the estimation of the error rate of each corpus, and establishing a weighted combination which produces a new retraining corpus with the desired error rate.

As mentioned above, if we put together a hand-disambiguated (assumed error-free) 70Kw corpus and a 195Kw automatically tagged corpus with an estimated error rate of 1.6%, we get a 265Kw corpus with a 1.2% error rate. But if we combine them with different weights we can control the error rate of the corpus: e.g. taking the weight for the correct 70Kw twice the weight for the 195Kw part, we get a corpus of 335Kw[11] with an error rate of 0.9%. In that way we can adjust the weights to get a training corpus with the desired error rate.

Figure 4 shows the relationship between the error rate and the relative weights between $\mathcal{C}^0$ and the retraining corpus.

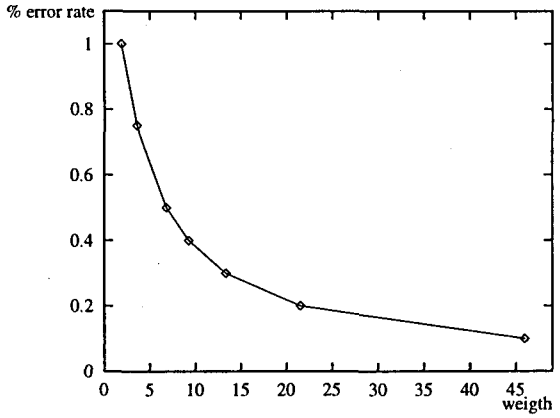

FIGURE 4. Relationship between the error rate and the relative weights for training corpora.

This weighted combination enables us to dim the undesired effect of noise introduced in the automatically tagged part of the corpus.

This combination works as a kind of back-off interpolation between the correct examples of $\mathcal{C}^0$ and the slightly noisy corpus of coincidences added at each step. By giving higher weights to the former, cases well represented in the $\mathcal{C}^0$ corpus are not seriously influenced by new erroneous instances, but cases not present in the $\mathcal{C}^0$ corpus are still incorporated to the model. So, the estimations of the statistical parameters for 'new' cases will improve the tagging performance while statistical estimations of already well represented cases will be, at most, slightly poorer.

We have performed an experiment to determine the performance obtained when the taggers are trained with corpus obtained combining $\mathcal{C}^0$ and the first extension of 200,000 words ($\mathcal{N}_1^1 \cap \mathcal{N}_2^1$) with different relative weights[12]. The steps selected are the weights corresponding to error rates of 0.1%, 0.2%, 0.3%, 0.4%, 0.5%, 0.75% and 1%.

It is obvious that too high weighting in favour of initial examples will produce a lower error rate (tending to zero, the same than the manual corpus), but it will also

[11] Obviously this occurrences are *virtual* since part of them are duplicated.

[12] Weights were straightforwardly incorporated to the bigrams and trigrams statistics. The decision tree learning algorithm had to be slightly modified to deal with weighted examples.

bias the taggers to behave like the initial tagger, and thus will not take advantage of the new cases.

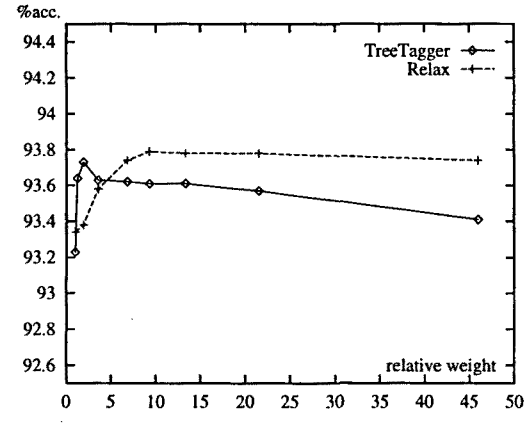

FIGURE 5. Results using the C^1_{200} training set with different weightings

The results summarized in figure 5 show that there is a clear improvement in the tagger performance when combining two training corpora with a proper weighting adjustment. Obviously, there is a tradeoff point where the performance starts to decrease due to an excessive weight for the initial data.

Although the behaviour of both curves is similar, it is also clear that the different tagging algorithms are not equally sensitive to the weighting values: In particular, RTT achieves its highest performance for weights between 1 and 3, while RELAX(BT) needs a weight around 10.

2.3.5. *Hand-correcting Disagreement Cases.* Another possible way to reduce the error rate in the training corpus is hand correcting the disagreement cases between taggers. This reduces the error rate of the new training corpus at a low human labour cost, since the disagreement cases are only a small part of the total amount.

For instance, in C^1_{200} corpus, there were 5,000 disagreement cases. Hand-correcting and adding them to the previous set we obtain a slightly larger corpus of 270Kw with a slightly lower error rate (1.17%), which can be used to retrain the taggers. We call this corpus C^1_M, where M stands for manual revision.

It is interesting to note that the increasing in number of training examples is especially noticeable in the case of decision trees, in which the 94K examples from C^1_{200} are increased to 108K examples in C^1_M (table 2 contains this information). This is due to the fact that each example covers a context window of six items. After hand-correction all sequences of six words are valid while before correction it was quite probable to find gaps (cases of disagreement) in the sequences of six words of the intersection corpus.

Results obtained with the corrected retraining corpus are shown in table 6, together with the results obtained with fully automatic retraining corpus of 200 Kw (C^1_{200}) and 800 Kw (C^1_{800}).

The first conclusion in this case is that the hand-correction of disagreement cases gives a significant accuracy improvement in both cases. However, the gain

Tagger Model	$\mathcal{C}^1_{200}$	$\mathcal{C}^1_{M}$	$\mathcal{C}^1_{800}$
RTT	93.2%	**93.8%**	93.4%
RELAX(BT)	93.3%	**93.8%**	93.9%

TABLE 6. Comparative results using $\mathcal{C}^1_{200}$, $\mathcal{C}^1_{M}$ and $\mathcal{C}^1_{800}$ training sets

obtained is the same order than that obtained with a larger retraining corpus automatically disambiguated. Unfortunately we had neither more available human resources nor time to hand-correct the remaining 15,000 disagreement words of $\mathcal{C}^1_{800}$ in order to test if some additional improvement can be achieved from the best automatic case. Without performing this experiment it is impossible to extract any reliable conclusion. However, we know that the price to pay for an uncertain accuracy gain is the effort of manually tagging about 20,000 words. Even when that would mean an improvement, we suspect that it would be more productive to spend this effort in constructing a larger initial training corpus.

Thus, unless there is a very severe restriction on the size of the available retraining corpus, it seems to be cheaper and faster not to hand correct the disagreement cases.

2.4. Best Tagger. All the the above described combinations produce a wide range of possibilities to build a retraining corpus. We can use retraining corpus of different sizes, perform several retraining steps, and weight the combination of the retraining parts. Although all possible combinations have not been explored, we have set the basis for a deeper analysis of the possibilities.

A promising combination is using the more reliable information obtained so far to build a $\mathcal{C}^1_{Best}$ retraining corpus, consisting of $\mathcal{C}^1_{M}$ (which includes $\mathcal{C}^0$) plus the coincidence cases from the $\mathcal{C}^0_{800}$ which were not included in $\mathcal{C}^1_{M}$. This combination has only been tested in its straightforward form, but we feel that the weighted combination of the constituents of $\mathcal{C}^1_{Best}$ should produce better results than the reported so far.

On the other hand, the above reported results were obtained using only either the RTT with decision trees information or the RELAX tagger using bigrams and/or trigrams. Since the RELAX tagger is able to combine different kinds of constraints, we can write the decision trees learned by RTT in the form of constraints (C), and make RELAX use them as in chapter 4.

Table 7 shows the best results obtained with every combination of constraint kinds. The lists of retraining corpora which yield each maximum result are also reported.

3. Conclusions and Further Work

In this chapter, we have presented the work that we have carried out towards the morphosyntactic annotation of the general–purpose LEXESP Spanish corpus.

The first step consisted of applying MACO+ a broad–coverage morphological analyzer. The second step consisted of adapting the taggers presented in chapter 4 to disambiguate the output of the morphological analyzer. Finally, in a third step, we showed how the collaboration between different POS taggers can be used to increase tagging accuracy without additional manual effort. Regarding this last point we defined a bootstrapping procedure which we applied to carry out the final annotation of the LEXESP corpus.

Tagger Model	Ambig.	Overall	Corpora
RTT	93.8%	97.7%	$\mathcal{C}^1_{\mathrm{M}}$, $\mathcal{C}^1_{\mathrm{Best}}$
RELAX(B)	93.3%	97.5%	$\mathcal{C}^1_{\mathrm{Best}}$
RELAX(T)	93.7%	97.6%	$\mathcal{C}^1_{\mathrm{Best}}$
RELAX(BT)	93.9%	97.7%	$\mathcal{C}^1_{800}$, $\mathcal{C}^1_{1000}$
RELAX(C)	93.8%	97.7%	$\mathcal{C}^1_{\mathrm{Best}}$
RELAX(BC)	94.1%	97.8%	$\mathcal{C}^1_{200}$, $\mathcal{C}^1_{\mathrm{M}}$, $\mathcal{C}^1_{\mathrm{Best}}$
RELAX(TC)	94.2%	97.8%	$\mathcal{C}^1_{200}$
RELAX(BTC)	94.2%	97.8%	$\mathcal{C}^1_{400}$, $\mathcal{C}^1_{\mathrm{Best}}$

TABLE 7. Best results for each tagger with all possible constraint combinations

From the results obtained on our work on the LEXESP corpus we would like to emphasize the following conclusions:

- A 70 Kw manually-disambiguated training set provides enough evidence to allow our taggers to get fairly good results. In absolute terms, results obtained on the LEXESP corpus are better than those obtained on the WSJ English corpus. One of the reasons contributing to this fact may be the less noisy training corpus. However it should be further investigated if the part of speech ambiguity cases for Spanish are simpler on average.
- The combination of two (or more) taggers seems to be useful to obtain larger training corpora with a reduced error rate, which enable the learning procedures to build more accurate taggers. The main results obtained with this approach are the following: Starting with the manually tagged training corpus, the best tagger combination achieved an accuracy of 93.1% on ambiguous words and 97.4% overall. After one bootstrapping iteration, using the coincidence cases in a fresh set of 800 Kw, the accuracy was increased up to 94.2% for ambiguous words and 97.8% overall. It is important to note that this improvement is statistically significant and that it has been achieved in a completely automatic re-estimation process. In our domain, further iterations did not result in new significant improvements.
- We have performed many experiments in order to properly tune the parameters of the bootstrapping algorithm in our domain. Although all possible combinations have not been explored, we have set the basis for a deeper analysis of the possibilities. Further experiments should establish which is the most appropriate bootstrapping policy, and whether it depends on the used taggers or not.
- The combination of two taggers is also useful to build a tagger which proposes a single tag when both taggers coincide and two tags when they disagree. Depending on user needs, it might be worthwhile to accept a higher remaining ambiguity in favor of a higher recall. With the models acquired from the best training corpus, we get a tagger with a recall of 98.4% and a precision of 96.8%. This means a remaining ambiguity of 1.009 tags/word, that is, 99.1% of the words are fully disambiguated and the remaining 0.9% keep only two tags.

The described bootstrapping procedure can easily be extended to a larger number of taggers. We are currently studying the collaboration of three taggers, using a tagger based on techniques for grammatical inference [**PP98**] in addition to the other two. Preliminary results show that the cases in which the three taggers coincide, present a higher accuracy than when only two taggers are used (96.7% compared to 95.5% on ambiguous words) and that the coverage is still very high (96.2% compared to 97.7%). Nevertheless, the difference is relatively small, and it must be further checked to establish whether it is worth using a larger number of taggers for building low error rate training corpora.

CHAPTER 6

Ensembles of Classifiers

The study of general methods to improve the performance in classification tasks, by the combination of different individual classifiers, is a currently very active area of research in ML supervised learning. In the ML literature this approach is known as *ensemble*, *stacked*, or *combined classifiers*[1]. The aim of the current chapter is to show how these machine learning techniques for constructing and combining several classifiers can be applied to improve the accuracy on the already presented English POS taggers.

We start by presenting a condensed survey of the main state–of–the–art approaches to classifier combination. Then we move to our particular application to POS tagging.

1. Introduction

The goal of combining the predictions of multiple learned models is to form an improved estimator. The general approach is to create an ensemble of learned models (or hypothesis) by considering different learning paradigms, by repeatedly applying some learning algorithm to different versions of the training data, or by repeatedly introducing some modifications in the *modus operandi* of the learning algorithm. Afterwards, the predictions of the individual learned models are combined according to a prescribed voting scheme.

Given a classification problem, the main goal is to construct several independent and complementary classifiers[2], since it has been proven that when the errors committed by the individual classifiers are uncorrelated to a sufficient degree, and their error rates are low enough, the resulting combined classifier performs better than all the individual systems [**AP96, TG96b, Die97**].

Several methods have been proposed in order to construct ensembles of classifiers that make uncorrelated errors. Some of them are general, and they can be applied to any learning algorithm, while other are specific to particular algorithms. From a different perspective, there exist methods for constructing homogeneous ensembles, in the sense that a unique learning algorithm has been used to acquire each individual classifier, and heterogeneous (or hybrid) ensembles that combine different types of learning methods with the hope that the particular representation and/or search strategies of each learning algorithm would contribute to produce different errors.

---

[1] A very good survey covering all these topics can be found in [**Die97**]. The following introduction and the notation used roughly follows his proposals.

[2] Concepts like bias, variance, diversity and coverage of the base classifiers in the ensembles are highly relevant to the potential improvement.

1.1. Ensembles of Homogeneous Classifiers. Several effective methods for improving the performance of a single learning algorithm through the combination of several learned models have been recently developed. The general methods for constructing ensembles can be categorized in the following five groups (specific methods will be mentioned later):

1. The methods in the first approach, manipulate the training examples (by means of resampling) to generate multiple hypothesis. The learning algorithm is run several times, each time with a different subset of the training examples. The specific way of selecting these subsamples defines the concrete method. The main representatives are *bagging* [**Bre96a**], *cross-validated committees* [**PMD96, Jel96**], and ADABOOST [**FS95, FS96**]. This technique works especially well with the so-called unstable learning algorithms[3], such as decision-tree, neural network and rule learning algorithms.
2. The second technique consists of giving the learning algorithm different information about the training examples at each time. This is done by selecting adequate subsets of the input features. Of course, this approach is only feasible when the training examples are described with many redundant features. We find examples with application to neural networks [**Che96, TG96b**], and to the Naive Bayes classifier [**Zhe98**].
3. The third approach, only applicable in multiclass problems, is based on the manipulation of the output targets. The use of a *decomposition scheme* allows the transformation of the K-class problem into a series of L binary class problems. A classifier is acquired for each bipartition and after, a *reconstruction method* is provided to make the fusion of the answers of all the L classifiers for a particular input in order to select one of the K classes. Several methods have been proposed varying the decomposition and reconstruction schemes. Among others, we find *one-per-class* (OPC), *pairwise coupling* (PWC), *pairwise coupling with correcting classifiers* (PWC-CC) [**PKPD95, HT98, MM98**], and *error-correcting output coding* (ECOC) [**DB91, DB95, MM97**].
4. The fourth technique consists of injecting randomness into the learning algorithm to produce slightly different classifiers at each time. This is done, for instance, by randomly selecting the initial weights of a neural network [**PMD96**], by randomly selecting the best split among the set of best ranked splits in a decision tree approach [**DK95b, Die98a**], or by randomly selecting the best condition among the set of best ranked conditions in a rule induction system (FOIL) [**AP96**]. More sophisticated techniques, related to the Markov Chain Monte Carlo (MCMC) method, have been applied to neural networks [**Mac92, Nea93**] and decision trees [**CGM96**].
5. Finally, we find hybrid methods in which some of the preceding paradigms are considered jointly. For instance, Raviv and Intrator [**RI96**] mixed bootstrap resampling with injecting noise in the input features, and Schapire [**Sch97**] developed a new boosting algorithm, ADABOOST.OC, in which ADABOOST is combined with error-correcting output coding. Also, Ricci

[3] A learning algorithm is said to be unstable when small variations on the training set can provoke great variations on the induced classifiers.

and Aha [**RA98**] applied a method that combines error–correcting output coding with feature selection.

Regarding particular methods for constructing ensembles, i.e. those methods highly dependent on the learning algorithm at hand, we basically find works applied to neural networks: [**Ros96a, OS96, Car96c, MP98**], and decision trees: [**Bun90, KK97**].

Once the ensembles have been constructed a method is needed to properly combine their individual outcomes. The simplest aggregation consists of using the majority rule in a simple (unweighted) voting system [**Cle89, Mat96**], which is the case of bagging, ECOC, and many other methods. If continuous outputs like posteriori probabilities are supplied, an average or some other linear combination of class probabilities can be made.

A common variation on the simple voting rule is the weighted voting approach [**LW94**], in which a weight is assigned to each base classifier reflecting how accurate the classifier is. These weights are usually obtained by measuring the accuracy of each individual classifier on the training data (or a holdout data set) and constructing weights that are proportional to those accuracies. This is the case of AdaBoost and other approaches [**Bun90, AP96**].

More sophisticated methods of combination are usually related to those techniques for dealing with heterogeneous (hybrid) ensembles of classifiers. Some of these methods will be mentioned in the next section.

Several studies, from both theoretical and empirical perspectives, have been performed in order to explain why the ensemble approach works, and which are the main differences between methods. All the following papers have some theoretical remarks on this issue [**KD95, TG96b, TG96a, DKM96, Die97, SFBL97, Bre97, Bre98a, MBB98, Sch99**]. See also [**KHDM98**] for a complementary set of references from the Pattern Recognition community of AI.

Empirical comparisons have focused on bagging, boosting, ECOC, and randomization. They provide insight about the behaviour of the methods under different experimental conditions, including variations on the training material (presence of noise, redundancy, exceptions, etc.), and on the base learning algorithm. See, for instance, [**Qui96a, AP96, Ali96, MO97, Sch97, Die98a, BK99**].

1.2. Constructing Ensembles of Heterogeneous Classifiers. In this approach, the ensembles o classifiers are constructed by applying different learning algorithms to the same data. This is certainly an appealing approach since learning algorithms based on very different principles will probably produce very diverse classifiers. In the related literature we find combinations of different learning paradigms that complement each other in several aspects. For instance Brodley [**Bro93**] combined instance–based learning with decision trees and linear discriminant functions, Ting [**Tin94**] and Kohavi [**Koh96**] also merged decision trees with instance–based learning, Vanhoof [**Van96**] combined the rule–based and case–based approaches, Ting [**Tin97**] used instance–based classifiers in combination with the Naive Bayes algorithm, etc.

The hybrid approach presents some difficulties (and perhaps this is why less work has been performed on using hybrid ensembles than that performed around the homogeneous approach). For instance, the straightforward aggregation of several learned models does not guarantee the success of the combination, since often some of them globally perform much worse than others. If they are too much unbalanced

the simple aggregation would perform bad and it will not be able to exploit the potentially useful diversity of the base-level classifiers.

Therefore, the individual classifiers should be previously checked for accuracy, diversity and coverage on the domain at hand [**Bro96, Die97**] and an adequate combination function should be induced.

The combining strategy becomes a crucial point here, since it should be able to robustly handle the inherent correlation, or multicollinearity, of the learned models while identifying the unique contribution of each. The usual approach consists of trying to determine the particular area of expertise of each model to properly select the best model (or combination of models) [**KE96**] for classifying each new example[4]. This issue is referred to as the model selection problem by Brodley [**Bro95**]. Similarly, Ortega [**Ort96**] talks about learning a set of referees, one for each model, which characterize the situations in which each of the models is able to make correct predictions.

Some auxiliary learners are commonly used to address the model selection problem by means of a meta-learning step. Therefore, the resulting classifiers, which may contain several generalization stages, are usually called meta-level classifiers. The applied meta-learning strategy and the involved learning algorithms differentiate the existing approaches. Among others we find: Model Class Selection (MCS) [**Bro95**], stacking generalization and variants [**Wol92, Bre96b, TW97, Ska96, FCS96, SW99, Mer99**], hierarchical mixture of experts [**JJ94**], and Principal Components Regression (PCR*) [**MP99**].

1.3. Applying Ensembles of Classifiers. Fairly impressive results have been obtained by using the already described techniques on the common unstable learning algorithms. Table 1 below contains a set of relevant publications in which some empirical evaluation is reported (most of them have been already mentioned, however now they are listed by the type of weak learning algorithm they boost):

Decision trees	[**DB95, DC96, Qui96a, MO97, KK97, CGM96, Bre98a, Die98a, Qui98, MM98, BK99**]
Neural networks	[**Mac92, Nea93, DB95, PMD96, Che96, Lee96, RI96, Ros96a, OS96, Car96c, MO97, MP98, SB98a**]
Rule-induction systems	[**AP96, Qui96b**]
Naive Bayes	[**Zhe98, BK99**]
Instance-based learning	[**DK95b, Jel96, Ska96**]

TABLE 1. References of works evaluating ensembles of classifiers of several base-level learning algorithms

Most of the work described in the preceding papers was validated on the data sets of the UCI Machine Learning repository [**MM96**], however several applications to real tasks have been also carried out.

[4]The results of empirical comparisons of existing learning algorithms illustrate that each algorithm has a selective superiority; each is best for some but not all tasks. This selective superiority applies not only to complete tasks but also to particular areas of the instance space [**Bro93**].

Regarding NLP, we find ensembles of classifiers in context–sensitive spelling correction [**GR98**], word sense disambiguation [**RAA97**], text categorization [**SS98a, BM98**], and text filtering [**SSS98**]. The combination of classifiers have also been applied to POS tagging. For instance, [**Hal96**] combined a number of similar taggers by way of a straightforward majority vote. More recently, two parallel works [**HZD98, BW98**] combined, with a remarkable success, the output of a set of four taggers based on different principles and feature modelling. In these works several variants of weighted combination were used as well as stacked generalization using decision trees and instance–based classifiers. Finally, in the work presented in the previous chapter 5 the combination of taggers is used in a bootstrapping algorithm to train a part of speech tagger from a limited amount of training material.

2. Improving POS Tagging by using Ensembles of Classifiers

This section is devoted to explain our most recent work on improving the POS taggers presented in previous chapters (RTT and STT) by using homogeneous ensembles of decision trees. The fact that these taggers treat separately the different ambiguity classes —by considering a different decision tree for each class— allows a selective construction of ensembles of decision trees focusing on the most relevant ambiguity classes, —which, as we have already seen, greatly vary in size and difficulty—

Therefore, our work presents a slightly different approach to that of constructing an ensemble of different preexisting taggers [**BW98, HZD98**], which consists of internally applying the combination of decision trees within a single tagger (i.e. to construct ensembles of some internal components of a classifier). This approach could also alleviate a drawback of the external hybrid combination, which is that the efficiency of the resulting tagger is proportionally divided to the number of taggers included in the ensemble (since all of them have to vote for each input word)[5]. In our case, we benefit from the decision–tree approach and the ambiguity–class partition to propose the use of ensembles of classifiers only in the cases in which some improvement is feasible, with the aim to improve accuracy with a small time/space penalty. Additionally, it has to be said that our approach does not exclude some further external combination with other taggers.

Another goal of the present work is to practically test two techniques that were proposed in chapter 3 but they were not still incorporated into the taggers, i.e, the tagging of unknown words, under the *open vocabulary assumption*[6] and the generation of convex pseudo–data to alleviate the data sparseness problem. Recall that the latter was combined with the construction of an ensemble of classifiers in order to avoid parameter tuning.

2.1. Setting. In order to better test the tagger under the open vocabulary assumption and to facilitate the comparison to other taggers we used here a larger test set. The same 1,17 Mw part of the WSJ corpus was randomly partitioned into two subsets to train and test the system. The relative proportions were 85% and 15%, which means 998,354 words for training and 175,412 words for testing.

[5]Against the presented argument there is the fact that the combined process of tagging can be done in parallel for each tagger.

[6]Several authors talk about the *closed vocabulary assumption* when no unknown words are considered

The word form lexicon derived from the training corpus contains now 45,469 entries (compared to the 49,206 words of the previous experiments) and the manual corrections for the 200 most frequent words were preserved.

Finally, the test set has been used as completely fresh material to test the taggers. All results on tagging accuracy reported in this section have been obtained against this test set. Figures about ambiguity rates of the test set are similar to those of previous experiments: 2.40 tags per ambiguous word (1.45 overall). The number of unknown words (with respect to the training set) was 3,941 (2.25%). Considering that each unknown word was provided with the 20 possible tags for open class words, the ambiguity rate per ambiguous word increased up to 3.49 tags/word.

The training corpus contains 239 different ambiguity classes and the training examples were extracted exactly in the same way as explained in chapter 3.

2.1.1. *Some Words About Unknown Words.* The training examples for the `unknown-word` ambiguity class were also collected from the training corpus. In principle, all occurrences of nouns, verbs, adjectives, etc. in the training corpus could be valid examples for this class. This is, for instance, the approach followed in chapter 3 and also that of [**Bri95a, DZBG96**] and others. In this way we would collect a big training set of over 500,000 examples. However some problems arise from this approach.

On the one hand, the current implementation of the learning algorithm cannot deal with a so huge set of examples. This flaw is not much important since it could probably be overcomed by partitioning the training set into several small enough sets (say, for instance, 10 sets of 50,000 examples), and applying classifier combination. Additionally, some strategies of sub-sampling designed to work with large data–bases [**MCR92**] could be applied in order to speed–up the acquisition algorithm.

On the other hand, and more important, we observed that the syntactic contexts and the morphological features of these examples are not fully representative of the "real unknown words" (those words not appearing in a medium–big size lexicon), since the relative frequencies are not conserved. For instance, the percentage of adjectives that are multi–word compounds is around 2% in the general training corpus (which does not give a very strong association), while this percentage is five times bigger for the unknown words in the test set (and so, it is a much more indicative feature of the adjective reading than it seemed). Another significant example is that of distinguishing between common and proper nouns. In the general training corpus the proportion of capitalized common nouns is less than 1%, and so the "capitalized?" feature is almost sure for differentiating common nouns from proper nouns. However, in the real cases of unknown words the percentage of nouns that are capitalized is around 15%, and therefore the learned feature fails to characterize common nouns in a relevant percentage of cases.

Although the trees for unknown words were able to properly generalize over an unseen part of the training set (see chapter 3 in which an accuracy close to 90% is obtained using a 50,000 example training set), the previously explained differences make the real accuracy on unknown words to be much lower (below 80%) than we expected.

We obtained far better results collecting the examples with a similar procedure to that employed for constructing training sets for N–fold cross validation: First,

the training corpus is randomly divided into twenty disjoint parts of equal size. Then, the first part is used to extract the examples (tagged with open class categories) which do not occur in the remaining nineteen parts, that is, taking the 95% of the corpus as known and the remaining 5% to extract the examples. This procedure is repeated with each of the twenty parts, obtaining approximately 22,500 examples which behave much more likely the real unknown words (in particular, the resulting proportions of each of the 20 tags are very similar than those appearing in the test set). The choice of dividing by twenty is not arbitrary. 95%–5% is the proportion that results in a percentage of unknown words very similar to the test set (i.e., 2.25%). Table 2 shows the percentage of unknown words resulting from other proportions.

	50%	45%	40%	35%	30%	25%	20%	15%	10%	5%
Unk.Words	3.21	3.06	2.88	2.77	2.62	2.49	2.40	2.36	2.27	2.23

TABLE 2. Percentage of unknown word occurrences in the x% of the training corpus with respect to the remaining (100-x)%

We used a k–nearest neighbour algorithm in order to test the appropriateness of this collection of examples. The k was experimentally set to 10 and the features were weighted by using the Information Gain measure. The accuracy obtained on tagging the unknown words of the test set was 75.46% when using the straightforward training set of 536,415 words, while using only the 22,793 examples extracted according to the described procedure the accuracy raised up to 77.21%.

This procedure allows a very significant space saving and a moderate accuracy improvement. In following sections we will see how this performance can be boosted again by using ensembles of decision trees.

2.2. Baseline Results. Before going into the combination of classifiers, we will test the conventional RTT and STT taggers on the above described domain in order to establish a reference baseline.

In this first experiment we used the basic set of six discrete–valued attributes as the unique information to disambiguate known ambiguous words (i.e, the part–of–speech tags of the three preceding and two following words, and the orthography of the word to be disambiguated). For tagging unknown words, we used an extended set of 23 attributes —fully descibed in table 3— which can be classified into three groups[7]:

A. *Contextual information*: part–of–speech tags of the two preceding and following words.
B. *Orthographic and Morphological information* (about the target word): prefixes (first two symbols) and suffixes (last three symbols); Length; Multi–word?; Capitalized?; Other capital letters?; Numerical characters?; Contain dots?
C. *Dictionary–related information*: Does the target word contains any known word as a prefix (or a suffix)?; Is the target word the prefix (or the suffix) of any word in the lexicon?

[7]Note that the information used to disambiguate unknown words presupposes neither language specific knowledge nor previous morphological analysis.

The features of the last group are inspired in those applied by [**Bri95a**] when addressing unknown words. It has to be noted that these features are not binary–valued attributes but all of them take as values the ambiguity class of the word that match the query, or 0 when it does not exist. In the case of existence, the uniqueness is guaranteed because prefixes and suffixes are searched from longest to shortest and the first match is assigned.

		Attribute	Values	Type
A	1	tag(-2)	Any tag in the Penn Treebank tagset	S
	2	tag(-1)	"	S
	3	tag(+1)	"	S
	4	tag(+2)	"	S
B	5	length	Integer	S
	6	multi–word?	{yes,no}	S
	7	capitalized?	"	S
	8	other_capital_letters?	"	S
	9	all_capital_letters?	"	S
	10	contain_numeric_character?	"	S
	11	contain_periods?	"	S
	12	char(1)	Any printable ASCII character	D
	13	char(2)	"	D
	14	char(n)	"	D
	15	char($n-1$)	"	D
	16	char($n-2$)	"	D
	17	chars(1...2)	Possible prefixes of two symbols	D
	18	chars($n-1$ ... n)	Possible sufixes of two symbols	D
	19	chars($n-2$ ... n)	Possible sufixes of three symbols	D
C	20	contains_word_pref?	Any ambiguity class	D
	21	contains_word_suf?	"	D
	22	contained_word_pref?	"	D
	23	contained_word_suf?	"	D

TABLE 3. Complete set of attributes for dealing with unknown words

In these conditions, the learning algorithm acquired, in about thirty minutes, a base of 191 trees which required about 0,68 Mb of storage. The programs are new implementations using PERL-5.003 and they were run on a SUN UltraSparc2 machine with 194Mb of RAM.

The results of the taggers working with this tree–base are presented in table 4. MFT stands for the baseline most–frequent–tag tagger. RTT, STT, and STT$^+$ stand for the basic versions of the taggers presented in chapter 4. Two technical comments should be done regarding the concrete implementation of taggers. First, RTT was forced to completely disambiguate the input text. The second refers to the fact that the straightforward inclusion of unknown words in the statistical tagger resulted in a severe decreasing of performance (due to the high ambiguity of unknown words the number of partial paths to calculate at each step of the Viterbi algorithm increases drastically). To avoid this situation, we applied the tree for unknown words in a pre–process for filtering low probable tags. In this way, when entering to the tagger the average number of tags per unknown word was reduced from 20 to 3.1.

Going back again to table 4, the overall accuracy is reported in the first column. Columns 2,3, and 4 contain the tagging accuracy on some specific groups of words:

unknown words, ambiguous words (excluding unknown words) and *known* words which is the complementary of the set of unknown words. Column 5 shows the speed of each tagger. The absolute figures are merely informative (the taggers could be drastically speed up by using an optimized *C* implementation). What is relevant here is the performance of each tagger relative to the others. Finally, the 'Memory' column reflects the size of the used language model (the lexicon is not considered here).

Tagger	Overall	Known	Ambiguous	Unknown	Speed	Memory
MFT	92.75%	94.25%	83.40%	27.43%	2818 w/s	0 Mb
RTT	96.61%	97.01%	91.36%	79.22%	426 w/s	0.68 Mb
STT	96.63%	97.02%	91.40%	79.60%	321 w/s	0.68 Mb
STT+	96.84%	97.21%	91.95%	80.70%	302 w/s	0.90 Mb

TABLE 4. Tagging accuracy, speed, and storage requirement of RTT and STT taggers

Three main conclusions can be extracted:

- RTT and STT approaches obtain almost the same results in accuracy, however RTT is faster.
- STT obtains better results when it incorporates bigrams and trigrams, with a slight time–space penalty.
- The accuracy of all taggers is comparable to the best state–of–the art taggers under the open vocabulary assumption (see section 2.5).

2.3. Ensembles of Decision Trees. Our purpose is to improve the performance on two types of ambiguity classes, namely:

- *Most frequent ambiguity classes.* We focused on the 26 most representative classes, which concentrate the 86% of the ambiguous occurrences. From these, eight (24.1%) were already resolved at almost 100% accuracy, while the remaining eighteen (61.9%) left some room for improvement.
- *Ambiguity classes with few examples.* We considered the set of 82 ambiguity classes with a number of examples between 50 and 3,000 and an accuracy rate lower than 95%. They agglutinate 48,322 examples (14.24% of the total ambiguous occurrences).

We have applied several of the methods, described in the first part of the chapter, for constructing homogeneous ensembles of decision trees for the eighteen frequent ambiguity classes of our interest plus the `unknown-word` ambiguity class. We briefly describe the methods that reported major benefits in the following subsections. Additionally, some comments on methods that failed to apply in our domain are provided in section 4 of the present chapter.

Regarding the sparse ambiguity classes, we applied the CPD method (see chapter 3) to increase the number of examples. This is explained in section 2.3.4.

2.3.1. *Bagging.* From a training set of n examples, several samples of the same size are extracted by randomly drawing, with replacement, n times. Such new training sets are called *bootstrap replicates.* In each replicate, some examples appear multiple times, while others do not appear (on the average, they contain 63.2% of the original training set). A classifier is induced from each bootstrap replicate and then they are combined in a voting approach. The technique is called *bootstrap*

aggregation, from which the acronym *bagging* is derived. In our case, the bagging approach was performed following the description of [**Bre96a**], and constructing 10 replicates for each data set (several authors indicate that most of the potential improvement provided by bagging is obtained within the first ten replicates).

2.3.2. *Combining Feature Selection Criteria.* In this case, the idea is to obtain different classifiers by applying several different functions for feature selection inside the tree induction algorithm. In particular, we have selected, from those tested in chapter 3, a set of seven functions that achieve a similar accuracy, namely: *Gini Impurity Index*, *Information Gain* and *Gain Ratio*, *Chi-square statistic* (χ^2), *Symmetrical Tau criterion*, RLM (a distance-based method), and a version of RELIEFF which uses the Information Gain function to assign weights to the features. See chapter 3 for the references to these methods, and appendix B for detailed definitions.

2.3.3. *Combining Features.* For this purpose we used the extended set of features presented in section 4 of chapter 3 (and summarized in table 11 of that chapter) which incorporates lexical information about words appearing in the local context of the target word, and the ambiguity classes of the same words. In this way, we consider information about the surrounding words at three different levels of specificity: word form, POS tag, and ambiguity class.

This model also includes some features, which are very similar to Brill's lexical patterns [**Bri95a**], to capture collocational information. Such features are obtained by composition of the already described single attributes and they are sequences of contiguous words and/or POS tags (up to three items).

The resulting features were grouped, according to their specificity, to generate ensembles of eight trees. The idea here is that specific information (lexical attributes and collocational patterns) would produce classifiers that cover concrete cases (hopefully, with a high precision), while more general information (POS tags) would produce more general (but probably less precise) trees[8]. The combination of both type of trees should perform better because of the complementarity of the information.

Being A, B, C, ... the letters appearing in table 11 of chapter 3, and being X the set of three features including the target word, and the part-of-speech of the preceding and following words, the eight groups of features used for learning each individual tree are, precisely: A, B+X, C+X, D+X, E+X, F+X, G+X, A-G. Note that X has been included to prevent an excessive degradation of the generalization ability of obtained trees in the ambiguity classes where specific attributes does not properly fit.

The features for dealing with unknown words were combined in a similar way to create ensembles of 10 trees. Following the notation of table 3, we used the following groups of attributes: A+B, A+C, A+D, B+C, B+D, C+D, A+B+C, A+B+D, A+C+D, B+C+D, and A+B+C+D. The groups A, B, C, D were not considered alone because they did not guarantee enough accurate trees to work well in the combination (accuracies using those features ranged from 46% to 67%, while combinations of two or more groups were all over 70%).

2.3.4. *Generating Pseudo-Examples.* We used a method by Breiman [**Bre98b**], which was previously described in chapter 3, to increase the number of examples

[8] In other words, we expect to perform the combination of two types of sources: the first with a high precision, but a lower recall, and the second with a high recall and an acceptable precision.

of the sparse ambiguity classes. We call this method CPD, standing for generation of Convex Pseudo–Data.

Recall that the method for obtaining new data from the old is similar to the process of combination of genes for creating new generations in genetic algorithms. First, two examples of the same class are selected at random from the training set. Then, a new example is generated from them by selecting attributes from one or another parent according to a certain probability. This probability depends on a single *generation parameter* (a real number between 0 and 1), which regulates the amount of change allowed in the combination step.

In chapter 3 we showed that a practical solution to avoid the tuning of the generation parameter is to construct several training sets using different values of the generation parameter, to learn a different decision tree for each set, and to combine their results. In this way, we make the global classifier independent of the particular choice, and we generally obtain a combined classifier that is more accurate than any of the individuals.

In this experiment we used 6 fixed values for d ranging from 0.3 to 0.8, while the number of examples generated in order to complement a training set of N examples was the maximum between 4,000 and 2.5 times N. All the examples were introduced at the the beginning of the induction process.

It has to be noted that in the original paper, Breiman proposes a more complex method of introducing the pseudo–examples within the CART decision tree induction algorithm, which consider the introduction of examples at each node of the tree, generating them under demand, and until no more original examples are misclassified by the current learned tree. However, the implementation of this method presents some computational problems, as the proper author notes in his paper. For the sake of simplicity, we decided to add a fixed amount of generated pseudo–examples at a time, at the root level of the tree. Despite the simplicity of the proposal, very good results were obtained, as explained in chapter 3. We know that this is an arbitrary decision that relies on a weak empirical basis, and that further work should be done in this direction.

2.4. Constructing and Evaluating Ensembles. The three types of ensembles were applied to the 19 selected ambiguity classes in order to decide which is the best in each case. The evaluation was done by means of a 10–fold cross–validation using the training corpus. The combination was performed by averaging the results of each classifier in the ensemble (simple voting when probabilities are available). The results obtained confirm that all methods contribute to improve accuracy in almost all domains. The absolute improvement is not very impressive but the variance between trials was generally very low and, so, the gain was statistically significant in the majority of cases.

These results are reported in table 5, in which the error rate of a single basic tree is compared to the results of the ensembles for each ambiguity class. In this table, BAG stands for bagging, FSC for the combination of different functions for feature selection, and FCOMB stands for the combination of different input features. These figures are calculated by averaging the results of the ten folds. The last column, 'BestER', represents the percentage of error reduction for the best method in each row.

Summarizing, BAG wins in 8 cases, FCOMB in 9, and FSC in 2 (including the `unknown-word` class).

	Amb. Class	#exs	%exs	Basic	BAG	FSC	FCOMB	BestER
1	IN-RB-RP	34,489	10.16	8.30	7.31	7.79	**7.23**	12.89
2	VBD-VBN	25,882	7.63	7.44	**5.93**	6.64	6.28	20.30
3	NN-VB-VBP	24,522	7.23	4.10	3.70	3.84	**3.58**	12.68
4	VB-VBP	17,788	5.24	4.13	**3.62**	3.94	3.76	12.35
5	JJ-NN	17,077	5.03	14.71	**13.30**	13.50	13.55	9.59
6	NNS-VBZ	15,295	4.51	5.14	4.37	4.59	**4.34**	15.56
7	NN-NNP	13,824	4.07	9.67	9.10	8.37	**6.83**	29.37
8	JJ-VBD-VBN	11,403	3.36	19.18	17.91	18.05	**17.27**	9.96
9	NN-VBG	9,597	2.83	14.11	**12.53**	12.93	12.99	11.20
10	JJ-NNP	8,724	2.57	5.10	4.50	4.56	**4.35**	14.71
11	JJ-RB	8,722	2.57	10.45	**8.86**	9.75	9.68	15.22
12	DT-IN-RB-WDT	8,419	2.48	7.01	**6.49**	6.84	6.53	7.42
13	JJR-RBR	2,868	0.85	16.40	15.84	15.28	**14.72**	10.24
14	NNP-NNPS-NNS	2,808	0.83	36.50	36.50	35.14	**35.00**	4.11
15	JJ-NN-RB	2,625	0.77	15.31	13.32	**11.83**	12.44	22.73
16	JJ-NN-VB	2,145	0.63	13.32	13.87	12.99	**12.75**	4.28
17	JJ-NN-VBG	1,986	0.59	20.30	**17.98**	18.79	18.23	11.43
18	JJ-VBG	1,980	0.58	21.11	**18.89**	19.39	19.60	10.52
	Total	210,154	61.93	9.35	8.38	8.61	8.25	**13.40**
19	unknown-word	22,594	—	20.87	17.47	**16.86**	17.21	**19.26**

TABLE 5. Comparative results —error rates in %— of different ensembles on the most significant ambiguity classes

In the case of CPD, it was applied to the 82 selected ambiguity classes, with positive results in 59 cases, from which 25 were statistically significant (again in a 10–fold cross–validation experiment). These 25 classes agglutinate 20,937 examples and the error rate was diminished, on average, from 20.16% to 18.17% (a reduction of 9.87%).

2.5. Tagging with the Enriched Model. Ensembles of classifiers were constructed for the ambiguity classes explained in the previous sections using the best technique in each case. These ensembles were included in the tree–base used by the basic taggers of section 2.2 by substituting the corresponding individual trees, and both taggers were tested again using the enriched model.

At runtime, the combination of classifiers was done by averaging the results of each individual decision tree. Some better strategies for combination could be performed, for instance by weighting each tree in the ensemble with a measure of goodness. However, the acquisition of these weights would require a previous testing of the trees in a kind of tuning supervised corpus.

In order to test the relative improvement of each component, the inclusion of the ensembles is performed in three steps: 'CPD' stands for the inclusion of the ensembles generated using the CPD method (basically to address the smallest ambiguity classes), 'ENS' stands for the inclusion of ensembles dealing with frequent ambiguity classes and unknown words, and 'CPD+ENS' stands for the inclusion of both. Results are described in table 6 (the results of the basic versions previously reported in table 4 have been also included in order to ease the comparison).

Some important conclusions are:

Tagger	Overall	Known	Ambig.	Unknown	Speed	Memory
RTT	96.61%	97.00%	91.36%	79.21%	426 w/s	0.68Mb
RTT(CPD)	96.66%	97.06%	91.51%	79.25%	366 w/s	0.93Mb
RTT(ENS)	96.99%	97.30%	92.23%	83.25%	97 w/s	3.53Mb
RTT(CPD+ENS)	97.05%	97.37%	92.48%	83.30%	89 w/s	3.78Mb
STT	96.63%	97.02%	91.40%	79.60%	321 w/s	0.68Mb
STT(CPD)	96.69%	97.07%	91.56%	79.69%	261 w/s	0.93Mb
STT(ENS)	97.05%	97.36%	92.38%	83.78%	70 w/s	3.53Mb
STT(CPD+ENS)	97.10%	97.40%	92.51%	83.68%	64 w/s	3.78Mb
STT$^+$	96.84%	97.21%	91.95%	80.70%	302 w/s	0.90Mb
STT$^+$(CPD)	96.88%	97.25%	92.09%	80.77%	235 w/s	1.15Mb
STT$^+$(ENS)	97.19%	97.48%	92.73%	84.47%	65 w/s	3.75Mb
STT$^+$(CPD+ENS)	97.22%	97.51%	92.81%	84.54%	60 w/s	3.97Mb

TABLE 6. Tagging accuracy, speed, and storage requirements of enriched RTT and STT taggers

- The best result of each tagger is significantly better than each corresponding basic version, and the accuracy consistently grows as more components are added.
- The relative improvement of STT$^+$ is lower than those of RTT and STT, suggesting than the better the tree-based model is, the less relevant is the inclusion of n-gram information.
- The special treatment of low frequent ambiguity classes results in a very small contribution, indicating that there is no much to win from these classes, unless we were able to fix their errors in a much greater proportion than we really did.
- The price to pay for the enriched models is a substantial overhead in storage requirement and speed decreasing, which in the worst case is divided by 5.
- Unknown words are better handled by STT taggers, indicating that the pre-process of filtering low probable tags is useful both for accelerating the tagger and for eliminating the excessive noise due to the big amount of possibilities.

In order to compare our results to others, we list in table 7 the results reported by several state-of-the-art POS taggers, tested on the WSJ corpus with the open vocabulary assumption[9]. In that table, TBL stands for Brill's transformation-based error-driven tagger [**Bri95a**], ME stands for a tagger based on the maximum entropy modelling [**Rat96**], SPATTER stands for a statistical parser based on decision trees [**Mag96**], IGTREE stands for the memory-based tagger by Daelemans et al. [**DZBG96**], and, finally, TComb [**BW98**] stands for a tagger that works by combination of a statistical trigram-based tagger, TBL and ME.

Comparing to all the individual taggers we observe that our approach reports the highest accuracy, and that it is comparable to that of TComb obtained by the combination of three taggers[10]. This is encouraging, since we have improved an

[9]Chapter 7 presents a more exhaustive comparison between these, and other, relevant approaches to POS tagging and ours.

[10]Note that the results obtained by the other successful tagger-by-combination mentioned in the introduction [**HZD98**] has not been included here. The reason is that it was evaluated on the LOB corpus instead of the WSJcorpus.

Tagger	Train	Test	Overall	Known	Unknown	Ambig.
TBL	950 Kw	150 Kw	96.6%	—	82.2%	—
ME	963 Kw	193 Kw	96.5%	—	86.2%	—
SPATTER	~975 Kw	~47 Kw	96.5%	—	—	—
IGTREE	2,000 Kw	200 Kw	96.4%	96.7%	90.6%	—
TComb	1,100 Kw	265 Kw	97.2%	—	—	—
STT^{+}(CPD+ENS)	998 Kw	175 Kw	97.2%	97.5%	84.5%	92.8%

TABLE 7. Comparison of different taggers on the WSJ corpus

individual POS tagger which could be further introduced as a better component in an ensemble of taggers.

Unfortunately, the performance on unknown words is difficult to compare. On the one hand, many authors do not provide the corresponding figures. On the other hand, it strongly depends on the used lexicon. For instance, IGTREE does not include in the lexicon the numbers appearing in the training set, and, so, any number in the test set is considered unknown (they report an unusually high percentage of unknown words: 5.5% compared to our 2.25%). The fact that numbers are very easy to recognize could explain their outstanding results on tagging unknown words.

In order to support this thesis we tested a 10–nearest neighbour algorithm on the unknown words of the test set, using exactly the same features as IGTREEs (following the description in [**DZBG96**]), and weighting these features with the Information Gain measure. Note that the IGTREE approach is a way to compress and index a set of learning instances (this is why it is presented as a memory–based learning method), and, therefore, the performance should be similar to that of the k-nn algorithm. Using the same training set of 22,793 examples than in our experiments, the k-nn algorithm obtained an accuracy of 77.21%, which is slightly below than the accuracy of a single decision tree tested alone (outside the taggers) and without any type of boosting (78.36%).

Finally, note that ME also reports a higher percentage of unknown words, 3.2%, while TBL says nothing about this issue.

3. Discussion of Results and Further Work

In this chapter, we have applied several ML techniques for constructing ensembles of classifiers to address the most representative and/or difficult cases of ambiguity within a decision–tree–based English POS tagger. As a result, the overall accuracy has been significantly improved. Comparing to other approaches, we see that our tagger performs better on the WSJ corpus and under the open vocabulary assumption, than a number of state–of–the–art POS taggers, and similar to another approach based on the combination of several taggers.

The cost of this improvement has been quantified in terms of storage requirement and speed of the resulting enriched taggers. Of course, there exists a clear tradeoff between accuracy and efficiency which should be resolved on the basis of the user needs. Another factor that should be mentioned is that, although all proposed techniques are fully automatic, the construction of appropriate ensembles requires a significant human and computational effort.

More generally, one may think that, after all the involved effort, the achieved improvement seems small. It is true that, up to the present, the results that he have

obtained by combining several classifiers are not as impressive as some previously published works in other domains. On this particular, we think that we are moving very close to the best achievable results using fully statistically–based techniques, with a limited amount of contextual information, and ignoring long–distance dependencies, and semantic and pragmatic knowledge.

We think that the key is in the treatment of exceptions and infrequent linguistic phenomena. The infrequent events have, in isolation, a very low statistical significance, and so they are very difficult to acquire with automatic machine learning algorithms (which are statistical in nature), especially in the presence of a certain amount of noise[11]. It is our belief that some kind of specific human linguistic knowledge should be jointly considered in order to achieve the next qualitative step.

All in all, we think that the work of this chapter can be considered a valuable starting point on the application of classifier combination to POS tagging, however, it is clear that further study should be devoted in many directions. Some of them have been already mentioned, other of our interest are listed below, and finally some remaining lines are suggested in the following section 4.1.

- There are several points regarding the methods used for constructing the ensembles of decision trees, and the way they are combined and included in the taggers that are not enough mature. As we have mentioned all along the chapter, some of the implementation decisions are primarily arbitrary, some methods are applied in the most naive and straightforward way, etc. Apart from this fact, some of the already applied methods for constructing ensembles could be jointly considered in a mixed approach.
- We are now quite interested on experimenting with the inclusion of our tagger as a component in an ensemble of preexisting taggers, in the style of **[BW98, HZD98]**.
- Regarding the combination step, we are interested on testing more sophisticated methods for combining the results of individual classifiers. We think of weighted voting rules, or about introducing some intermediate steps of meta–learning (in the style of stacked generalization **[Wol92]**), in order to learn the best way of combining the components. The construction of such a meta–classifier could involve the use of alternative learning algorithms to complement the decision trees.
- In some of the applied methods, the information used to acquire each tree of the ensemble is fundamentally the same. As a consequence, the obtained trees have a low variance and their outcomes are highly correlated, leaving few room for improvement in the combination approach. Therefore, once an ensemble of decision trees is constructed it would be useful to be able to evaluate how 'different' are the trees of the ensembles in terms of redundancy, inconsistency, complementarity, etc. This would allow us to eliminate highly–correlated members, to predict whether it is worthwhile to include this ensemble or not, etc.
- Finally we will be interested on testing other methods for constructing ensembles that naturally applies to decision trees, e.g. injecting randomness.

[11] Certainly, the noise is a handicap in our domain of work. See, for instance, the paper by Ratnaparkhi **[Rat96]** for some comments and evaluation on the labelling inconsistencies of the WSJ corpus due to the different criteria applied by the annotators.

4. Methods that did not Work

Apart from the methods described in section 2 we tested a number of other methods for constructing ensembles of decision trees, which are explained below.

4.1. Boosting. Like bagging, boosting algorithms perform an iterative subsampling of the training set of examples to generate multiple hypothesis. The difference here is that boosting defines a probability distribution $p_i(\mathbf{x})$ over the training examples, instead of assuming a uniform distribution, and that successive steps are not independent. Initially, $p_1(\mathbf{x})$ is assigned a uniform distribution. In each iteration i, it draws a training set of size N by sampling with replacement according to the probability distribution $p_i(\mathbf{x})$. The learning algorithm is then used to acquire a classifier h_i. The error rate of this classifier on the training examples —weighted according to $p_i(\mathbf{x})$— is calculated and used to adjust the probability distribution on the training examples for the next iteration, $p_{i+1}(\mathbf{x})$. The weight adjustment is oriented to place more weight on those training examples that were misclassified by h_i and less weight on examples that were correctly classified. In subsequent iterations, therefore, more difficult learning problems are progressively constructed. In the combination, the vote of classifier h_i is weighted according to its error rate.

The recent literature contains many references regarding boosting algorithms and their application. The original work can be found in the papers by Freund et al. [**FS95, FS96**], in which the ADABOOST algorithm and some variants are introduced and tested as a way to boost the performance of 'weak' learners. 'arcx4', by Breiman[**Bre98a**], and 'Mini Boosting' by Quinlan[**Qui98**] are some simplified variants of ADABOOST. Other variations and improvements of ADABOOST can be found in the following references [**MD97, SS98b, FISS98, TZ98, MBB98**].

In our case, we used the ADABOOST algorithm following the description in [**SFBL97**], but instead of resampling from the set of examples according to a probability distribution, we implemented a new version of the decision-tree induction algorithm that deals with weighted examples —in a similar way to that followed by Quinlan [**Qui96a**] to adapt ADABOOST to C4.5— Some authors note that working with weighted examples is better than resampling when boosting decision trees.

The results of ADABOOST on our domain were disappointing. From the 18 ambiguity classes , only in four it was observed a significant change in accuracy, two positive and two negative (the rest were also tied seven to seven). Additionally, the loss of the negative cases was far more significant than the gain of the positive, and, therefore, the average result on all domains was slightly negative, and so performing clearly worse than bagging.

This is quite surprising since there are many empirical studies in the recent ML literature showing that ADABOOST outperforms the other resampling methods, and bagging in particular. See, for instance, the following references: [**Qui96a, MO97, Die98b, BK99**].

One possible explanation for such bad results is the noise present in the training corpus, since many authors [**Qui96a, MO97, Die98a**] have observed that boosting produces severe degradation on some data sets, especially in the presence of noise. For instance, Dietterich [**Die98a**] performs a comparison between bagging, boosting, and randomization using decision trees, in which is shown that boosting

is clearly the best method in noiseless data sets, but becomes the worst when injecting a 5% of wrongly classified examples. In this particular experiment, a 5% of classification noise, makes boosting to loss all the initial gain, while higher levels of noise make it even counterproductive. The same study concludes that bagging is the most error–tolerant method.

In our case, the percentage of mislabelled words in the WSJ corpus is around 2–3%, which would probably be higher on the ambiguous words, and, therefore, such an explanation could be valid. However, this hypothesis should be further verified by incrementally adding artificial noisy examples in the training data and observing the behaviour of bagging compared to boosting.

There are many other papers studying the behaviour of boosting. For instance, in **[Bre98a]** boosting is primarily described as a variance–reducing procedure, and therefore a low effectiveness should be expected when a low–variance (stable) algorithm is boosted. Schapire et al. **[SFBL97]** argued that stability in itself may not be sufficient to predict boosting's failure, and they characterize two situations in which boosting might fail, which are: (1) there is insufficient training data relative to the complexity of the base classifiers, and (2) the training errors of the base classifiers (weighted according to the current probability distribution of examples) become too large to quickly.

The first clearly does not apply to our domain, which is the set of most frequent ambiguity classes. However we empirically observed that the second happened in our experiments, indicating that the learning algorithm does not adapt very well to the most 'difficult' examples, i.e, those in which more weight is placed and which are the more important to calculate the error rates of successive base classifiers.

Bauer and Kohavi **[BK99]** performed a very exhaustive empirical study around bagging, boosting and their application using decision trees. One of the conclusions of the study is that the iterative algorithm for reweighting instances used by boosting emphasizes not only 'hard areas' but also outliers and noise

Joining the two previous paragraphs mean that although the learning process is focused to learn these kinds of examples, the acquired decision trees generally fail to classify them. This is not strange if they were majority noisy (and therefore contradictory) examples. Otherwise, it would indicate some limitation of the learning algorithm or, more probably, that the set of used features is not rich enough to properly describe all examples of the domain.

This is very interesting since a manually study of the highly emphasized examples in the final distribution should provide useful insight about difficult examples, infrequent events, and noise in the training set. The first two would be useful to determine which type of information is needed to resolve the most difficult cases, perhaps treating them separately. The third would be useful to apply some kind of pre–process to filter out noisy examples from training data.

This is an issue that we plan to devote further investigation.

4.2. Pairwise Coupling Classification with Correcting Classifiers. In this method, that we will call PWC-CC, the ensemble of classifiers is constructed by recasting multiclass problems into a set of binary classification problems, so it belongs to the family of methods that manipulate the output targets. More particularly, the decomposition scheme, converts a K–class problem into $\frac{1}{2}K(K-1)$ 2–class problems, one for each pair of classes. The bipartition for the pair (i, j) focuses on the separation of class y_i from class y_j. The learning algorithm is then

used for acquiring one classifier h_{ij} for each of these 2-class problems. Additionally, a new set of classifiers is obtained to prevent that a certain classifier h_{ij} could take part in the classification of new examples not belonging to y_i nor y_j. So, each classifier h_{ij} is paired with a *correcting* binary classifier, trained to separate classes y_i and y_j from all the rest. Provided that classifier outputs are expressed as probabilities, the final combination is performed by averaging the weighted results of all classifiers.

We implemented the PWC-CC method according to the description given in [MM98], and we applied it on the eight multiclass domains appearing in the eighteen ambiguity classes of our interest.

The results of PWC-CC were comparable to those of bagging. A significant improvement in accuracy was achieved in six of the eight multiclass problems. In the other two problems, the difference was also slightly positive. The global error reduction was of 10.12% (11.78% in the significant cases). The method was not finally used in the experiments because it brought no significant improvement to the other methods, it only applies in a reduced number of ambiguity classes, and it introduces some complexity in the algorithm that uses the acquired decision trees (the tagger proper) due to the introduction of the reconstruction scheme.

4.3. Partitioning of Large Training Sets. In the style of Chan and Stolfo [CS95], we tested the appropriateness of addressing the ambiguity classes with many training examples by dividing the whole training set into a number of relatively small disjoint subsets and combining the corresponding individually induced classifiers. For that, we selected the five largest ambiguity classes —with a number of examples over 17,000— and we estimated the right size of the subsets for each case, with the idea that subsets have to be big enough to obtain fairly accurate individual classifiers. In this way the number of combined trees in each domain varied from 4 to 10.

The ensembles constructed by partitioning the training set into disjoint subsets performed better than any individual tree in each of the five data sets (and in any of the combinations, varying the size of the subsets[12]). Unfortunately, the improvement was not enough to compensate the accuracy loss of the individual trees, which are grown from relatively small training sets. Compared to the trees acquired with all the available examples, the ensembles performed better in three cases (1 significant) and worse in the remaining two (1 significant). Therefore the improvement should be considered negligible, and the effort probably worthless.

4.4. Resampling Training Data for Unknown Words. We applied another method for constructing a set of decision trees for the `unknown-word` ambiguity class. It is based on the fact that the procedure for collecting, from the whole training set, the training examples for that class —explained in detail in section 2.1.1— can be seen as a kind of subsampling method, and thus it can be iterated many times, in the style of bagging, to construct more training sets (which will have a high degree of overlapping).

Following this idea, we repeated the extraction process six times, collecting training sets of about 22,500 examples each (recall that the number of examples

[12]The size of the disjoint subsets was relevant to the accuracy of the ensemble (with a positive bias to the ensembles with few, and, so, more accurate, trees), suggesting that an effort in estimating an adequate size is required in each case.

selected is about 2.5% of the 1 million word training corpus). More precisely, the six sets sum 136,435 examples, from which only 39,272 were actually different.

With the purpose of further complementing these highly redundant sets we collected six additional training sets, but now using the straightforward procedure (i.e., any occurrence of an open category is an example) preserving the relative proportions of the 20 tags for unknown words.

Then, we constructed 12 decision trees —say, $A_1, \ldots, A_6$ for the trees corresponding to the first six training sets, and $B_1, \ldots, B_6$ for the remaining— and we made several experiments combining them. In particular, we test the effect of incrementally adding A-trees, B-trees, and both, to the ensemble.

The best result was obtained with the combination $\{A_1, A_2, A_3, A_4, A_5, B_1\}$, which allowed to reduce the error rate from 20.87% (obtained with the best single tree) to 17.81%, indicating that this method is potentially useful, and that "B-type" examples can positively complement the training set if they are introduced in a lower proportion.

The achieved error reduction is significant, however the reduction obtained by more simple methods introduced in section 2.3 was even higher: Recall that the combination of functions for feature selection allowed the construction of an ensemble that reported an error rate of 16.86%. Additionally, note that we only report the best result here, and that in order to make this method applicable we should establish a procedure for automatically tuning the parameters involved. These two facts were the main cause for not including the presented method in the general experimental setting.