Recognising Clauses Using Symbolic and Machine Learning Approaches

Benjamin C. Hachey

Supervisor: Mirella Lapata

Master of Science in Speech and Language Processing
Department of Theoretical and Applied Linguistics
University of Edinburgh

September 2002

Abstract

Clauses are important for a variety of NLP tasks such as predicting phrasing in text-to-speech synthesis and inferring text alignment for machine translation (Ejerhed 1988, Leffa 1998, Papageorgiou 1997). The Computational Natural Language Learning 2001 shared task (Sang & Déjean 2001) set the goal of identifying clause boundaries in text using machine learning methods. Systems created for the task predicted a label for each word specifying the number of clauses starting and ending at that position in the sentence without differentiating between clause types.

This work extends that of the shared task in several ways: (1) performance bounds are explored, (2) an attempt is made to distinguish 'main' and 'subordinate' clauses, and (3) Winnow and maximum entropy, model classes proven effective in similar domains yet not previously employed for the task, are applied to the problem.

Acknowledgements

Big thanks to Mirella Lapata for her careful guidance and thoughtful support throughout this project, to Chad Cumby for his help with the FeX feature extractor, and finally, to Anna Pakarinen for her diligent journeys through various proofs.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Benjamin C. Hachey*)

Table of Contents

List of Figures

List of Tables

LIST OF ABBREVIATIONS

ALLiS	-	Architecture for Learning Linguistic Structure
CASS	-	Cascaded Analysis of Syntactic Structure
CMTAG	-	Clause Marker TAGging
CoNLL	-	Computational Natural Language Learning
DI93	-	DAGENS INDUSTRI 1993
GIS	-	Generalised Iterative Scaling
HMM	-	Hidden Markov Model
LSTM	-	Long Short-Term Memory
NLP	-	Natural Language Processing
POS	-	Part of Speech
SNoW	-	Sparse Network of Winnows
SUC	-	Stockholm Umeå Corpus
TiMBL	-	Tilburg Memory-Based Learner
TTS	-	Text-to-speech (Synthesis)
WSJ	-	Wall Street Journal

Chapter 1

Introduction

This chapter introduces the problem of clause recognition. First, we define clauses for the purpose of this work. Next, we motivate the the task of automatic clause recognition by presenting some applications in natural language processing (NLP) that benefit from information about clause structure. Finally, we overview the approach to recognising clauses that will be taken here.

1.1 What is a clause?

Clauses are integral to sentence structure. Loos, Anderson, Day, Jordan & Wingate (1999) provide the following formal linguistic definition of a clause:[1]

> A clause is a grammatical unit that includes, at minimum, a predicate and an explicit or implied subject, and expresses a proposition.

This is somewhat vague and does not give a very good idea of the structural ambiguity of clauses, especially to a student of prescriptive grammar. Therefore, we begin with a look at the three types of sentence common to pedagogical grammars that correspond to clause structure. We will present simple, compound, and then complex sentences.

The basic sentential building block is the simple sentence or independent clause. This is informally described as consisting of a subject and a verb and expressing a

[1]The quotation come from a book that is published online. The URL for the page containing the definition is <http://www.sil.org/linguistics/GlossaryOfLinguisticTerms/WhatIsAClause.htm>.

```
(M
   (M It wasn't clear
      (S how
         (S the ownership would stack up
         under the new plan S) S) M)
   , but
   (M employees would keep more than 50% M)
. M)
```

Figure 1.1: Clause example from the Penn Treebank

complete thought. Independent clauses along with dependent clauses are the building blocks used to create multiple sentences, capable of expressing sophisticated actions and thoughts. Quirk, Greenbaum, Leech & Svartik (1985) identify two major devices for building multiple sentences from clauses: coordination and subordination.

Generally speaking, compound sentences are those built through coordination and complex sentences are those built through subordination. Compound sentences contain two or more clauses joined by coordinators (e.g., *but* in Figure 1.1). Complex sentences contain an independent clause with one or more dependent clauses joined by subordinating conjunctions (e.g., *how* in Figure 1.1 and *that* in Figure 1.2). The sense of clauses used in this thesis refers to independent and dependent clauses mentioned above as well as the sentences they combine to create.

Figure 1.1 is an example of a multiple sentence from the Penn Treebank (Marcus, Santorini & Marcinkiewicz 1993) that contains examples of simple, compound, and complex sentences. This illustrates the recursive nature of clause structure. This recursive structure makes identification more difficult than part of speech or chunk tags (e.g., Brill 1994, Sang 2000), whose structure, as it is commonly formulated for automatic recognition, is not embedded.

Following Penn Treebank style guidelines (Bies, Ferguson, Katz & MacIntyre 1995), coordinated clauses are grouped under a clause of the same type. There is always one main clause which may consist as well of two or more coordinated clauses. In the case of Figure 1.1, the first of the coordinated main clauses (*It wasn't clear*

```
(M(S
      (S What
         (S 's more S)S),
   the losses
      (S they and the others caused S)
   "are
      (S just what
         (S we are stumbling over S)S)S),"
says Mr. Stapf,
   (S adding
      (S that
         (S the majority of misdeeds probably go
            (S undetected S)S)S)S)
. M)
```

Figure 1.2: Clause example from the Penn Treebank

how the ownership would stack up under the new plan) is a complex clause. There is a subordinator ‘how’ that sets off the independent clause (*It wasn’t clear*) from the dependent clause (*how the ownership would stack up*). Again following Treebank bracketing practice, the subordinate clause is broken down into a clause including the subordinator and a clause that does not include the subordinator. The second of the coordinated main clauses (*employees would keep more than 50%*) is a simple clause.

Now that we have a notion of clause structure and it’s embedded nature, we take a look at a sentence with truly complex clause structure. The sentence in Figure 1.2 has five levels of embedding and contains examples of a number of specific difficulties encountered in recovering Penn Treebank clause structure from text.

The sentence begins with a subordinate clause (*What’s more, the losses they and the others caused “are just what we are stumbling over*) containing (1) a dependent simple clause that contains an empty subject position and is difficult to construe as expressing a proposition (*’s more*), (2) dependent clauses that aren’t introduced by a subordinator or relative pronoun (e.g., *they and the others caused*), and (3) a dependent clause that is introduced by an adverb modifying the relative pronoun (*just what we are* *stumbling over*).

The second dependent clause attached directly under the top-level of the sentence (*adding that the majority of misdeeds probably go undetected*) contains three further levels of embedding. This clause (1) starts with a verb (exhibiting, once more, an empty or trace subject), and (2) contains a clause consisting of a single word (*undetected*).

1.2 What good are clauses?

Clauses have been postulated as being integral to linguistic processing and have been reported to be useful information for a variety of NLP tasks such as text-to-speech synthesis, machine translation, question answering, and the automatic induction of subcategorisation frames (e.g., Ejerhed 1988, Leffa 1998, Papageorgiou 1997, Abney 1996). In this section, we will first take a brief look at motivation for a clause-by-clause processing of multiple sentences by humans from the realm of theoretical linguistics. Then, we will present several applications in NLP that employ clause structure in performing their tasks.

Ejerhed (1988) summarises work by Jarvella (1971) and Jarvella & Pisoni (1970) that shows human recall of spoken complex sentences improves when there are pauses at sentence and clause boundaries. This evidence led Jarvella (1971) to propose a clause-by-clause view of processing within sentences. In related work, Ejerhed (1996) summarises phonetics work reported by Strangert, Ejerhed & Huber (1993). This study showed significant correlation between the placement of clause boundaries, intonation unit boundaries, silent intervals in speech, and perceived pauses by listeners.

Drawing on this theoretical work, Ejerhed (1988) describes two approaches to identifying basic, non-recursive clauses in unrestricted English text for improving the detection of large prosodic units in text-to-speech synthesis (TTS). The author claims that clause boundaries are prerequisite for the placement of prosodic events that make a significant contribution to the naturalness and intelligibility of synthesised speech. These tonal minor phrases correspond to a phrase accent, having a boundary tone, and, especially in slow speech, a pause at the end of the phrase. At the time of publication, the clause identification algorithm was reported as a significant improvement of the punctuation-based assignment of tonal minor phrases in the Bell Labs TTS system. No perceptual test results are described, however, regarding the intelligibility or naturalness of synthesised speech with and without this clause information.

Papageorgiou (1997) and Leffa (1998) describe clause segmenting algorithms in the context of machine translation. The first study uses clause information to aid the task of automatic sub-sentence alignment of parallel corpora. Results comparing alignment scores with and without the benefit of the clause recognition algorithm are not reported. However, clause alignment was shown to correct errors in clause identification (Papageorgiou 2002). In the case of the second study, clause boundary information is reported to help produce better syntax in the output of the translation program. The reported translation system using clause boundaries (in an informal comparison on sentences with difficult clause structure) shows a definite improvement over a state-of-the-art system lacking the benefit of such syntactic well-formedness and comprehensibility information.

Ejerhed (1988, p.220) also suggests that clauses can be seen as an intermediate step in full parsing:

> It is a hypothesis of the author's current clause-by-clause processing theory, that a unit corresponding to the basic clause is a stable and easily recognizable surface unit and that it is also an important partial result and building block in the construction of a richer linguistic representation that encompasses syntax as well as semantics and discourse structure.

This possibility is further explored by a number of researchers (e.g., Ejerhed & Church 1983, Abney 1991, Daelemans 1995, Ratnaparkhi 1998, Brants 1999). In a recent study, Sang (2002) implements a full parser as a sequence of classifiers built to recognise arbitrary phrases treating clause recognition as just such an intermediate step. The best reported configuration obtained a $F_{\beta=1}$ score (see Equation 2.6) on the full parsing task of 80.49; significantly below scores for state-of-the-art full parser (around 90). Some improvements are suggested, but it is not clear whether it is promising to pursue the approach further.

1.3 Approaches

The present work, although for the most part a machine learning project, can be seen as exploring the merits of symbolic and machine learning approaches to the task of clause recognition. Symbolic approaches are those where knowledge is explicitly encoded by the developer. In learning approaches, a representation (i.e., linear separator or probability distribution in the case of the learners we employ) of the knowledge is induced from a large amount of data. In this section, we take a brief look at how the problem of clause recognition is cast as a symbolic task and as a machine learning task.

1.3.1 Clause recognition with regular languages

Different symbolic approaches to identifying clauses have been reported in the literature including finite state grammars and stepwise, modular algorithms. Section 2.1 in the next chapter describes several previous approaches in some detail. For now, suffice it to say that these clause languages could be written as regular expressions.

Regular expressions are an algebraic formulae whose values are patterns consist-predefineding of sets of strings, called the language of the expression. The benefit is that languages described by a regular expression can be directly translated to a recogniser in the form of a finite state automata.

All of the symbolic approaches in the next chapter define patterns of words, or parts of speech, or other formal indicators of clause boundaries. Thus, whether they are originally implemented as such or not, they can be converted to deterministic finite state automata. Although, additional resources in some of the methods add complexity, these are nevertheless very efficient methods.

1.3.2 Clause recognition as a classification task

Motta & Lu (2000, p. 2) describe classification as “...finding one or more admissible solutions out of a predefined solution space, which explain the features of a given set of observables, in accordance with a given match criterion....” This highlights several items that define a given application:

1. The *features of a given set of observables*, which we will write as $\vec{x}$, constitute the input to the classifier specifying the active features for a given observation. In our application, this can be seen as a mapping from the text corpus to a format suitable for the software we employ.

2. The *solution space* is the set of possible answers $\mathcal{C}$ that our system is meant to choose among. These are referred to as the classes or categories that each observation belongs to.

3. The *match criterion* is defined by the model class that we choose. Chapter 4 will elaborate on the theory and architecture of the winnow and maximum entropy classifiers used here.

Training examples consist of pairs $(\vec{x}, c)$ specifying the correct class label of a given observation. The training procedure selects a single classifier (by setting the parameter values) from the chosen model class. Test examples $\vec{x}$ are then evaluated using the trained representation and the match criterion selects the best class.

The basic features available will be the lexical item, its part-of-speech tag, and its chunk tag (described in Section B.4) from a given sentence position and its neighbours. Predicted labels from previous phases are also available for phases 2 and 3. Phases 1 and 2 predict for each word whether any clause starts or ends at that position. The third stage predicts how many clauses start and end at the positions predicted in the first two phases. Thus, the three phases of this application constitute individual classification tasks with the following sets of categories (defined with Perl-style regular expressions):

- PHASE 1: $\mathcal{C} = \{c \mid /[SX]/\}$
- PHASE 2: $\mathcal{C} = \{c \mid /[EX]/\}$
- PHASE 3: $\mathcal{C} = \{c \mid /(\backslash(S) * \backslash * (S\backslash)) * /\}$

S and X labels in the first phase represent, respectively, positions that start and do not start a clause. E and X labels in the second phase represent positions that end and do not end a clause. And, phase three labels ($(S*$, $*$, $*S)$, $(S*S)$, $(S(S*$, and etcetera) define the number of embedded starts $($S and ends $S)$ at each position; the solitary $*$ label corresponding to no starts or ends at a given position.

Examples of the data encoding along with a description of the derivation from the Penn Treebank can be found in Appendix B. Details about the model classes and the software implementations will be given in Chapter 4.

1.4 Thesis outline

The first part of this dissertation describes previous work on recognising clauses. Issues not addressed in the previous work introduced in Chapter 2 motivate this thesis. Chapter 3 presents performance bounds including a new symbolic baseline that serves to better contextualise the machine learning work within the field of natural language processing as a whole. As mentioned above, Chapter 4 describes the classification approaches for Winnow and maximum entropy. Features and feature encoding are discussed in Chapter 5. Chapter 6 presents experiments with the two classification methods. Finally, chapter 7 summarises the results and achievements of this thesis, discusses shortcomings, and proposes future work.

Chapter 2

Previous Work

This chapter presents previous work on clause recognition. In Section 1, symbolic approaches are considered where knowledge is explicitly encoded by the developer. These stand in contrast to machine learning systems where a representation (e.g., linear separator, probability distribution in the case of the learners we employ) of the knowledge is induced from a large amount of data intended to be representative of the natural phenomena. Machine learning approaches are presented in Section 2. Finally, the motivation for the main part of this work is presented: issues not treated in the CoNLL-2001 shared task.

2.1 Symbolic approaches

Several symbolic approaches to clause recognition have been reported in the literature. We focus here on those reported by Ejerhed (1988, 1996), Papageorgiou (1997), and Leffa (1998).

Ejerhed (1996) describes one of the first studies focusing on identifying clauses in text. The study compares a finite state, rule-base approach and a n-gram, stochastic approach to the task of recognising *basic clauses*, defined as non-recursive clauses corresponding roughly to our simple sentences or independent clauses from the first section of Chapter 1. The algorithms attempt to recover all clauses, but do not attempt attachment disambiguation. Thus, no hierarchical dependency structure is assigned and it cannot reproduce the structures in Figures 1.1 and 1.2.

Rather, an end-to-end segmenting is done where one clause ends where the next begins. Pre-processing on the text consists of tagging with Church's (1988) part-of-speech (POS) tagger and identification of basic non-embedded noun phrases (cf Ejerhed 1987, Church 1988). The same stochastic POS program is used for both the regular expression and the stochastic clause recognition methods, but the respective methods were also used for noun phrase recognition. The stochastic program was trained from 60 texts from the Brown corpus (Francis & Kucera 1964) that were run through the regular expression method and hand-corrected.

The systems were evaluated on different test corpora. The regular expression approach was tested on texts from the *reportage* (political, sports, society, etc.), *learned* (natural sciences, medicine, mathematics, etc), and *general fiction* (novels and short stories) portions of the Brown corpus. The stochastic approach was tested on 4 stories of Associated Press newswire text. Both test corpora contained just over 300 test cases (the former had 308 clauses and the latter, 304). Ejerhed (1988) reports error rate e defined as

$$e = \frac{u+o+w}{t} \tag{2.1}$$

where u is the number of under-recognition errors, o is the number of over-recognition errors, w is the number of wrong place errors, and t is the number of clauses in the gold standard. By this measure, the regular expression method scored 13% and the stochastic method scored 6.5%. *Precision*, *recall*, and $F_{\beta=1}$ scores have also been computed from the results reported and appear in Table 2.1.

Ejerhed (1996) reports on another study in the recovery of basic clauses from text "based on the hypothesis that the language of simple clauses is *k*-testable." The notion is defined by Salomaa (1981):

> Where *k* is a degree of locality specifying how many adjacent squares must be scanned to determine if a word w is in the language L, L is said to be *k*-testable.

A new regular expression algorithm is described where the clause segmentation component comes from the notion of *k*-testability. The author considers the input sentences to be 4-testable, which essentially says that a context of only 4 words around a given potential clause start need be scanned to determine if that position is a clause start or not. Clause ends are simply the positions before a clause start or the end of a sentence.

Other differences from the earlier study lie in the corpus and the pre-processing. Two Swedish corpora are employed. The first test set comes from a portion of the Stockholm Umeå Corpus (SUC) which is an excerpt from a novel and is manually tagged with POS information. The second test set comes from the DAGENS INDUSTRI 1993 (DI93) corpus, which consists of financial news text and is automatically assigned POS tags for the task (cf Åström 1998). The same error rate measure *e* (defined in Equation 2.1 above) is reported with scores of 4.1% for the SUC corpus and 9.2% for the DI93 corpus. See Table 2.1 for *precision* and *recall*.

Papageorgiou (1997) describes a symbolic method for identifying embedded clauses inspired by Abney's (1991) Cascaded Analysis of Syntactic Structure (CASS) parser. The approach is surface oriented and stepwise and trades depth of analysis for efficiency, addressing the issue of scalability without large effects on speed. Clause analysis is done in four stages. In a pre-processing stage, the input is partitioned into tokens. Next, Brill's (1994) tagger is used to predict POS tags. Following this is the Clause Marker TAGging (CMTaG) module, which reduces ambiguity of and adding annotations to possible clause markers. For example, an attempt is made in this module to extend parts of speech of more than one orthographic word for subordinators such as 'so as', 'in order to', 'as if', and etcetera. Finally, a partial parsing module creates the clause structure.

Once again the results reported are for English. This time the test corpus is ten regulation documents (e.g. European Commission rules on export refunds and standard import values) from the CELEX database. Papageorgiou (1997) reports a success rate *s* score which we believe to be defined as

$$s = \frac{t - (u + o + w)}{t} \tag{2.2}$$

or, the complement (i.e., $s = 1 - e$) of Ejerehed's error rate defined above. A 93%

AUTHOR		*Precision*	*Recall*	$F_{\beta=1}$
Ejerhed (1988)	*Regular expression*	87.01	98.89	92.57
	Stochastic	95.07	96.01	95.54
Ejerhed (1996)	*SUC corpus*	100.00	95.80	97.85
	DI93 corpus	98.80	90.70	94.58
Papageorgiou (1997)		95.44	93.06	94.23
Leffa (1998)		–	95.00	–
Sang & Déjean (2001)	*Best CoNLL-2001*	84.82	73.28	78.63
Sang & Déjean (2001)	*CoNLL-2001 Baseline*	98.44	31.48	47.71

Table 2.1: Interpreted standard scores from symbolic systems

success rate is reported in the paper. A summary of *precision*, *recall*, and $F_{\beta=1}$ scores can be found in Table 2.1.

Leffa (1998) describes the final symbolic system we will report on. The approach uses indicators of coordination and subordination (e.g., coordinate conjunctions, relativisers) along with valence information about relevant verbs to identify embedded clauses. Thus, this algorithm departs from the simple, efficient finite state approach, in that processing is done left-to-right looking for formal indicators of clause boundaries and from each verb outward to identify which noun phrases in surrounding the verb should be grouped into a clause with it. This also requires another resource in addition to rules about formal indicators. Namely, a verb lexicon containing valence information must be available. Clauses are first segmented and then identified as one of nominal or adverbial.

Pre-processing consists of a symbolic POS tagger (POS lookup followed by disambiguation based on the context where there are multiple tag possibilities), and identification of nominal groups. The clause identification algorithm itself is divided into three phases as follows.

1. identify clause initiators (e.g., column 4 of Table 2.2)

2. identify clause terminators (e.g., column 5 of Table 2.2)

3. identify full clauses (e.g., column 6 of Table 2.2)

This division of sub-tasks forms the basis for the symbolic baseline and the classification approach described respectively in Section 1 of Chapter 3 and Section 2.2 of this chapter.

500 sentences with more than one clause drawn randomly from an unnamed corpus of expository text were selected for testing. The scoring metric reports correctly classified clauses c. Nothing specific is mentioned about over-recognition, under-recognition, or wrong-place errors. However, we believe this measure to be the same as *recall*, defined as

$$c = recall = \frac{tp}{tp + fn} \tag{2.3}$$

where tp is the number of clauses correctly classified and fn is the number of clauses missed by the algorithm and $tp + fn$ is equivalent to t in the equations above. By this measure, the overall clause recognition performance is 95%.

Some weaknesses of the methods above and the comparison thereof ought to be pointed out here. First, standard measures are not reported. We have attempted to compute *precision*, *recall*, and $F_{\beta=1}$ scores from the information in the papers, but these must not be considered absolute. Second, the methods above cannot really be compared because the test data is different in every case. Furthermore, the preparation and resources incorporated also differ between the approaches. The symbolic and machine learning experiments reported in the following chapters use standard measures and identical resources and test data.

2.2 Machine learning approaches

The Computational Natural Language Learning 2001 (CoNLL-2001) shared task (Sang & Déjean 2001)[1] provided a classification framework in which different learning approaches to clause recognition can be readily compared. Nevertheless, the framework

[1]CoNLL-2000 shared task definition and related software as well as performance figures for the various systems are found at <http://lcg-www.uia.ac.be/conll2000/chunking/>.

can be used equally for machine learning and statistical approaches. In the following, we will first consider why a full parsinFurthermoreg approach to identifying clauses may not be desirable. Next, we will elaborate on how clause recognition can be cast as a machine learning task. The first application of machine learning to embedded clauses that we know of is described by Orăsan (2000). However, we will concentrate on the CoNLL-2001 systems and results.

2.2.1 Full parsing

One approach to clause recognition would be to pursue a full parsing-style system. Indeed, one could use an existing state-of-the-art parser (e.g., Charniak 2000, Collins 1999, Ratnaparkhi 1999) and interpret clause labels from this. As we describe in the next chapter, average clause label $F_{\beta=1}$ score of Collins's (1999) parser is 90.26. This is considerably better than the best system from the CoNLL-2001 shared task, which achieved an $F_{\beta=1}$ score of 78.63 predicting clauses. However, full parsing does have it's drawbacks.

There are several reasons why we might not want to use a full parser. First of all, a classification framework is faster and easier to implement as we can readily apply a number of machine learning algorithms. This makes it easily portable to new domains and languages. the stored representation is generally more efficient and classifiers are considerably faster at run time. This being the case, it is desirable to find an approach that removes this extra overhead for tasks such as question answering where full parsing is not necessarily needed and efficiency is a priority. Thus, for the work here, we will cast clause recognition as a classification problem.

2.2.2 Classification revisited

Section 3.1 in the first chapter described how clause recognition can be cast as a classification task. Here, we will discuss in detail how such an approach can be implemented.

The first thing to determine is the exact definition of a clause that will be used. The CoNLL-2001 data is derived from the Wall Street Journal portion of the Penn Treebank (sections 15-18 forming the training data, section 20 forming the development data,

PHASE 3					
PHASE 2					
PHASE 1					
WORD	POS TAG	CHUNK TAG	START	END	FULL
It	PRP	C-NP	S	X	(S(S*
was	VBD	B-VP	X	X	*
n't	RB	O	X	X	*
clear	JJ	C-ADJP	X	X	*
how	WRB	C-ADVP	S	X	(S*
the	DT	B-NP	S	X	(S*
ownership	NN	E-NP	X	X	*
would	MD	B-VP	X	X	*
stack	VB	E-VP	X	X	*
up	RP	C-PRT	X	X	*
under	IN	C-PP	X	X	*
the	DT	B-NP	X	X	*
new	JJ	I-NP	X	X	*
plan	NN	E-NP	X	E	*S)S)S)
COMMA	COMMA	O	X	X	*
but	CC	O	X	X	*
employees	NNS	C-NP	S	X	(S*
would	MD	B-VP	X	X	*
keep	VB	E-VP	X	X	*
more	JJR	B-NP	X	X	*
than	IN	I-NP	X	X	*
50	CD	I-NP	X	X	*
%	NN	E-NP	X	E	*S)
.	.	O	X	E	*S)

Table 2.2: Example sentence from Penn Treebank in task format.

and section 21, the test data). Therefore, the definition of a clause is, for the most part, the Treebank definition (cf Bies et al. 1995). The `chunklink` (Bucholz 2000) and `clauselink` scripts are used to derive the data format for the task. This is described in detail in the appendix on data preparation. In short most of the Treebank *S* labels (i.e., *S, SBAR, SBARQ, SINV,* and *SQ*) are converted to start, end, and embedded clause labels for the respective phases described below.

Table 2.2 illustrates the data format for the example sentence from Figure 1.2.[2] The task is broken into the three phases laid out by Leffa (1998). In the first phase, features derived from POS tags, chunk tags, and words are used to predict start labels. In the second phase, end labels are predicted using the same features as the first phase together with the predicted start labels. Finally, the third phase uses features derived from POS tags, chunk tags, words, and predicted start and end labels to predict the embedded clause structure of the sentence.

Pre-processing for the CoNLL-2001 shared task consists of assigning POS and chunk tags to each token. The task uses Treebank-compatible POS and chunk labels, however, Brill's (1994) POS tagger and Sang's (2000) chunker are used in place of the hand-corrected Treebank labels to simulate the situation in which this information is not available. This introduces noise before the clause identification stage that precludes the possibility of perfect performance in recognising clauses.

One other key point of the shared task is that it introduced a standard paradigm under which clause recognition approaches can be qualitatively compared. The same data was available to all entrants and the following measures were used.

$$precision = p = \frac{tp}{tp + fn} \tag{2.4}$$

$$recall = r = \frac{tp}{tp + tn} \tag{2.5}$$

tp, tn, and fn are defined in Table 2.3. *Precision* is a measure of the correctness of the chosen items and *recall* is a measure of the number of correct items chosen. Either of these two measure are not sufficient, however, as *precision* does not give any penalty

[2] A step-by-step description of the derivation of the format for the example sentence in Table 2.2 from the Treebank format is found in Appendix B.

	YES is correct	NO is correct
YES is assigned	tp	fn
NO is assigned	fp	tn

Table 2.3: Confusion matrix for evaluating classifiers

for choosing nothing and *recall* does not penalise choosing everything. In other words, *precision* doesn't account for under-recognition and *recall* doesn't account for over-recognition. We attempt to balance these scoring errors by optimising the $F_{\beta=1}$ score.

$$F_{\beta=1} = \frac{(\beta^2+1)*p*r}{(\beta^2*p+r)} = \frac{2*p*r}{p+r} \tag{2.6}$$

The baseline for the task was computed by simply assigning a clause start to the beginning of every sentence and a clause end to the end of every sentence. This achieved a 47.71 $F_{\beta=1}$ score for full segmentation. The score for clause starts and ends are 53.34 and 65.34 respectively.

2.2.3 CoNLL-2001 systems

A number of different classification methods were used for the CoNLL-2001 shared task. These were comprised of boosting decision trees and decision graphs, neural networks, memory-based learning, statistical, and symbolic learning. Results for the various systems are reported in Table 2.4.[3] The following gives a brief description of the different approaches.

Patrick & Goyal (2001) applied the Adaboost algorithm (Freund & Schapire 1995, Schapire 1999) to decision graphs (an extension of decision trees where nodes are allowed to have more than one parent (Oliver & Wallace 1991, Wallace & Patrick 1993)). Their previous experience had shown that "boosting improves classification performance, for increased errors in data, and for increases in missing data." The

[3]Numbers reported are the updated in a revised version of the introduction (Sang & Déjean 2001) available on the shared task web site (<http://cnts.uia.ac.be/conll2001/clauses/>).

System		Starts	Ends	Full
Adaboost	(Carreras & Már.)	91.72	89.22	78.63
HMMs	(Molina & Pla)	87.74	78.61	68.12
MBL	(Sang 2001)	88.82	82.28	67.79
Adaboost	(Patrick & Goyal)	87.27	81.76	66.17
ALLiS	(Déjean)	87.43	65.47	62.77
Connectionist	(Hammerton)	–	–	50.42
Baseline		53.34	65.34	47.71

Table 2.4: $F_{\beta=1}$ scores for the CoNLL-2001 systems

system appears to have suffered from poor encoding of the linguistic features, however, as it was reported to have difficulties with sentence beginnings and endings (Sang & Déjean 2001).

Hammerton (2001) applied a new connectionist architecture incorporating a memory mechanism to the shared task. Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber 1997) enables recurrent neural networks to remember information over long time periods, which should allow them to incorporate information from early in the input sequence into the decision. This is intensive to train, though, so only the first 1000 sentences were used, which creates an obvious disadvantage.

Déjean (2001) employed his symbolic learner, Architecture for Learning Linguistic Structure (ALLiS) (Déjean 2000), to the shared task. The algorithm is based on theory refinement and tries to improve existing grammars by adding such detail as contextualisation and lexicalisation to account for predictions mistakes from the current grammar. A confidence measure is used to stop such extensions when the accuracy of the rule or the frequency of the sequence is high enough.

Sang (2001) applied the memory-based learner, Tilburg Memory-Based Learner (TiMBL) (Daelemans 2000), to the shared task. Memory-based learners are descendants of the k-nearest neighbours algorithm and get their name from the fact that they keep some representation of the training data in memory. Classification for new data, then, is based on it's similarity to feature-value pair sets in the stored training data. The system is unique in employing voting, which offers slight improvements, and head information for chunk elements.

Molina & Pla (2001) report on their system for the shared task, which uses a hidden Markov model (HMM) (Rabiner & Juang 1986). For this approach, the problem consists of solving the following equation:

$$\arg\max_{o_1 \ldots o_T} \left(\prod_{j=1}^{T} P(o_j | o_{j-1}, o_{j-2}) \cdot P(i_j | o_j)\right) \tag{2.7}$$

where $P(o_j|o_{j-1},o_{j-2})$ represents the transition probabilities between states and $P(i_j|o_j)$ represents the output probability for the current observation. The input i_j is taken to be POS and chunk tag tuples for the current word, and the output set O is the set of possible clause tags for the given stage.

Carreras & Màrquez (2001*b*) describe the system that produced the best results for the shared task. They applied Adaboost to decision trees with a modification (Schapire & Singer 1998, Carreras & Màrquez 2001*a*) that allows "the algorithm to work in a higher dimensional feature space that contains conjunctions of simple features." They employed features representing long-distance relationships and patterns of clause indicators in a sentence. They also took a unique approach to segmenting that provided them with results in the final phase with an $F_{\beta=1}$ score more than 10 points higher than the next best system. The details of the features and segmenting algorithm are described in Chapter 5.

In follow up work to the CoNLL-2001 shared task Sang (2002), explored the difference between his system and the system of Carreras & Màrquez. Three main things were found to give Carreras & Màrquez their advantage over the other systems: algorithm, features, and segmentation. Their unique approach to clause segmenting was a great advantage in the third phase. In all task phases, the algorithm and the features served them well.

In Sang's (2002) follow-up work, the features from Carreras & Màrquez (2001*b*) improved scores for the memory-based learner with further improvements from when a feature selection technique (forward sequential selection starting with zero features). Conversely, the scores of Carreras & Màrquez (2001*b*) went down when they ran an experiment using the features of Sang & Déjean's (2001) memory-based learner.

The Carreras & Màrquez scores were still better when the systems were using the same features, though, which indicates that the learning method also played a role in the superiority of their system. As mentioned earlier, Adaboost has been shown to make some learners more robust in the face of noisy and sparse data. Furthermore, the fact that the decision trees are modified to handle higher dimensional feature space seems to have aided the system in all phases, giving it the best results. Apart from this, the choice of features improved system scores.

Finally, the Carreras & Màrquez system performed far above the rest in the third and final phase. This is due to their novel approach to segmentation, which consisted of three phases:

1. predict whether each start should actually be two starts (S$\mapsto$SS?)
2. predict whether each double start should actually be three starts (SS$\mapsto$SSS?)
3. obtain confidence values for clause candidates and select full clauses

More than three clause starts is not predicted because the data for those is too sparse. After confidence values are estimated, full clauses are chosen, one-by-one, through a greedy selection of the clause candidate with the highest confidence value. After a clause is chosen, all the remaining candidates that are incompatible due to crossing brackets are removed. The process is repeated until no candidates remain. Our approach to segmentation (presented in detail in Section 2.3.2 of Chapter 6) is very similar.

2.3 Where do we go from here?

There are a couple of interesting issues that weren't addressed in the CoNLL-2001 shared task. First, the shared task did not distinguish any different types of clauses. Second, the CoNLL-2001 shared task provided a very simple baseline that does not go very far toward contextualising performance of a classification approach amongst the field as a whole.

First, we attempt to distinguish clause types. We concentrate on the difference between main and subordinate rather than the various more detailed levels of information in the Treebank. Appendix B describes in detail how this distinction was recovered from the Penn Treebank.

Second, machine learning methods are only motivated if the have some intrinsic advantage over symbolic methods. The obvious and generally most important is performance. However, we must also consider other issues such as memory and run time complexity, the time and labour necessary to produce a system for a new domain, and how domain specific an implementation should be.

Symbolic algorithms may or may not be less complex than machine learning algorithms. Often, symbolic algorithms (e.g., Leffa 1998) incorporate resources not made available in the CoNLL-2001 shared task. However, the new symbolic baseline, presented in the next section, is considerably less complex than any machine learning approach.

Also, less effort was put into the symbolic algorithm than was put into feature extraction and encoding for the machine learning methods. And, where implementations in new domains of machine learning approaches depend on the availability of training data in those domains, symbolic methods only need annotated corpora for testing.

It would be interesting to see how a symbolic approach would perform on the same data with the same preparation. A portion of the work reported in the next section goes some way toward this end. A symbolic baseline is estimated based on a simplified version of Leffa's (1998) clause recognition algorithm.

Finally, none of the systems applied Winnow or maximum entropy, methods that have provided state-of-the-art results on other NLP tasks such as spelling correction, POS tagging, chunking, prepositional phrase attachment, and full parsing (e.g., Golding & Roth 1999, Ratnaparkhi 1996, Zhang, Demerau & Johnson 2001, McLauchlan 2001, Ratnaparkhi 1999) and are well-suited to natural language problems for reasons presented in Chapter 4. In summary, the following work seeks to do the following three things not addressed in the CoNLL-2001 shared task: distinguish main and subordinate clauses, better contextualise the machine learning results amongst the relative merits of symbolic methods, apply new learning methods to the task.

2.4 Summary

In this chapter, we have given an overview of previous work on the task of clause recognition. We began by describing several symbolic approaches. These are all reported to perform well in the context of the systems they were built for. However, they do not report standard measures, they all work on somewhat different definitions of a clause, and the incorporated different resources. Next, we reported on machine learning approaches, focusing on the CoNLL-2001 shared task. Several issues not addressed in the shared task are the basis for this thesis: (1) the baseline does not in any way reflect the relative merits of symbolic methods, (2) clauses are recognised without making a distinction as to what type of clause it is, and (3) machine learning methods proven effective in similar domains were not applied to the task. The work reported in the following chapters addresses these issues.

Chapter 3

Performance Bounds

This chapter reports on work that involves finding performance bounds for the task of clause recognition. Primarily, this consists of a new symbolic baseline meant to provide a more reasonable lower bound. Next, several possible upper bounds are discussed.

3.1 Baseline

As mentioned in the previous chapter, the baseline for the CoNLL-2001 shared task is exceedingly simple. In this section, we describe an approach to finding a more reasonable baseline. The idea is that machine learning methods should be better contextualised within the field of NLP as a whole. Ejerhed (1988), for instance found that though the stochastic system performed better, the symbolic system performed nearly as well and did not suffer from over-recognition problems. The baseline of the CoNLL-2001 shared task is the simplest possible symbolic approach where every sentence in the data is considered to correspond to a single clause.

3.1.1 Version I

The symbolic approach we pursue is a simplified version of Leffa's (1998) algorithm. The system described here does not use a lexicon or information about verb subcategorisation. In this sense it is more like a finite state grammar of formal indicators of

```
for each word
  if the word is a (clause-level) conjunction
    or subordinator
    then mark as clause initiator
  if the word is a subject-less verb
    then mark as clause initiator
  if the word is the subject of a verb
    then mark as clause initiator
for each word
  if the word is followed by a clause initiator and
    (the phrase preceding it contains a verb)
    then mark as clause terminator
for each clause initiator
  if verb phrase and clause terminator follow
    then mark as a clause
```

Figure 3.1: Symbolic clause identification algorithm from Leffa (1998).

subordination similar to the work of Ejerhed (1988) and Ejerhed (1996).

The three phases of the algorithm as presented by Leffa (1998) are outlined in Figure 3.1. The first two phases go through word-by-word looking at the context for indicators of coordination and subordination such as subordinating conjunctions and subject-less verb phrases to see if the word starts or ends a clause. The third phase considers clause initiators as the processing unit.

It is apparent from a brief inspection of this algorithm that it produces a segmentation similar to the end-to-end clauses of Ejerhed (1988) and Ejerhed (1996). Though, it focuses on dependent clauses and does identify some embedded structure. Since our data is based on the Penn Treebank and the fully embedded CoNLL-2001 conversion, this segmentation approach does not work very well for all phases. However, the approach to identifying clause starts was successful in raising the $F_{\beta=1}$ score more than twenty points.

IDENTIFYING CLAUSE STARTS

CLUES	*accuracy*	*precision*	*recall*	$F_{\beta=1}$
CoNLL-2001 baseline	92.81	98.44	36.58	53.34
Subordinating Conjunctions (W)				
Possessive wh-pronoun (WP$)	92.83	98.45	36.76	53.53
Wh-pronoun (WP)	92.96	97.56	38.25	54.95
Wh-determiner (WDT)	93.25	98.49	40.56	57.46
Wh-adverb (WRB)	92.96	97.83	38.16	54.90
All Ws	93.57	97.20	43.98	60.56
After Ws				
Possessive wh-pronoun (WP$)	92.80	98.03	36.60	53.30
Wh-pronoun (WP)	93.03	98.14	38.65	55.46
Wh-determiner (WDT)	93.25	98.44	40.58	57.47
Wh-adverb (WRB)	93.00	97.96	38.51	55.29
All but WP$	93.66	97.76	44.59	61.24
Coordinating conjunction	92.46	86.61	38.85	53.64
B-SBAR	93.45	97.76	42.70	59.43
B-VP				
+Past tense verb (VBD)	90.13	59.45	38.18	46.50
+*Gerund/present part (VBG)*	92.95	92.92	40.27	56.19
+*to (TO)*	93.27	95.00	42.27	58.51
B-NP...B-VP	91.23	60.37	63.91	62.09
All italicised	93.29	64.91	87.57	74.56

IDENTIFYING CLAUSE ENDS

CLUES	*accuracy*	*precision*	*recall*	$F_{\beta=1}$
CoNLL-2001 baseline	95.64	98.44	48.90	65.34
Before clause start	92.25	53.95	52.73	53.34
Before sentence end	95.92	77.99	71.73	74.73

FULL SEGMENTATION

CLUES	*accuracy*	*precision*	*recall*	$F_{\beta=1}$
CoNLL-2001 baseline		98.44	31.48	47.71
New S & Es	86.81	79.24	48.35	53.27

Table 3.1: Version I experimental results.

A summary of *accuracy* (computed as $tp + tn/tp + fp + tn + fn$), *precision*, *recall*, and $F_{\beta=1}$ scores with different formal indicators of coordination and subordination is found in Table 3.1. *B-SBAR*, *B-VP*, and *B-NP* are chunk labels. They correspond to the beginnings of a subordinate clause, a verb phrase, and a noun phrase respectively.

The highest $F_{\beta=1}$ score on clause starts was achieved using the following formal indicators of clausal conjunction (corresponding to those italicised in the Table 3.1):

- the beginning of every sentence (as in the CoNLL-2001 baseline).
- subordinating conjunctions.
- the position after a subordinating conjunction.
- the start of a subordinate clause chunk (*B-SBAR*).
- the start of a verb phrase chunk (*B-VP*) that
  - begins with a gerund/present participle (*VBG*), or
  - begins with to (*TO*).
- the start of a noun phrase (*B-NP*) followed by a verb phrase.

This gives an improvement of more than 20 points over the CoNLL-2001 baseline $F_{\beta=1}$ score.

In the case of subordinating conjunctions, the position before the word is checked to see if it is a preposition. If it is, the clause start is assigned to the previous position. Otherwise, a clause start is assigned to every subordinating conjunction. In the case of the position after a subordinating conjunction, a clause start is assigned every time. Coordination is difficult to recover due to the ambiguity concerning what level of the parse is being coordinated (i.e., the noun phrases immediately adjacent to the conjunction or the clauses they may be contained within). Our simplified approach is too noisy so there is no effort to recover coordinated clauses in the best configuration

Every start of an *SBAR* chunk is also assigned a clause start. For the beginning of a *VP* chunk, a clause start is assigned if the named POS tag is also present at that position (*VBG* or *TO* for the best configuration). Finally, a clause start is assigned to the beginning of a noun phrase (*B-NP*) if there is a following verb phrase without another noun phrase intervening. This was the noisiest indicator, bringing the *precision* down considerably. But it raised the *recall* enough to significantly raise the $F_{\beta=1}$ score (nearly 10 points when used alone). This is still lower than all of the CoNLL-2001 systems but Patrick & Goyal (2001). However, *recall* is higher that all but Carreras & Màrquez (2001*b*).

Interestingly, Ejerhed (1988) and Papageorgiou (1997) claim that an important difference between symbolic and machine learning approaches is that symbolic approaches tend to suffer only from under-recognition whereas learning approaches suffer from under *and* over-recognition errors. However, using the noun phrase followed by a verb phrase indicator shows that it is a matter of design whether a symbolic approach suffers only from lack of coverage and does not exhibit the kinds of *precision* problems a machine learning approach tends to have.

The highest end score is not as interesting, although it did raise the $F_{\beta=1}$ score some 10 points from the baseline. However, it is achieved by assigning a clause end to the penultimate sentence position if there is a clause start followed by a verb phrase before it with no intervening clause end, which is already achieved by the CoNLL-2001 baseline phase 3 algorithm. This keeps all clause starts and eliminating extra clause ends, putting extra clause ends at the penultimate sentence position where necessary.

Clause segmenting improves only a few points. This is because the approach is the same as the CoNLL-2001 baseline mentioned above and the clause ends aren't adding any information. This first version of the algorithm was used to experiment with formal indicators and determine which are reliable clues for identifying clause boundaries. For this reason, it does not distinguish main and subordinate clauses. Rather it predicts a single clause label for all phases to be directly comparable to the CoNLL-2001 baseline.

3.1.2 Version II

The second version of the baseline extends the first to distinguish main and subordinate clauses. The clues are those found to work the best in Version I. Distinguishing main and subordinate clauses is done in a very straightforward manner. As coordination was

SYSTEM	*Precision*	*Recall*	$F_{\beta=1}$
Main clauses	98.44	87.45	92.62
Subordinate clauses	27.25	39.23	32.16
Overall	79.24	48.35	53.27

Table 3.2: Results distinguishing main (*M*) and subordinate (*S*) clauses.

found to be difficult to recover, main clause labels are assigned only to entire sentences (i.e., an attempt is made only to recover main clauses at the very top level). All other identified clauses are considered subordinate. Results for this version are found in Table 3.2.

Overall results are the same as the clause labels are in the same places. The difference is that some of them now are labelled as main and the rest are labelled as subordinate. Considering our definition of a main clause here, the large difference in performance on main and subordinate clauses reflect in part the fact that embedded clauses are much more difficult to recover than top-level clauses. They also reflect on our simple definition of main clauses as simple and compound clauses at the top level.

3.2 Upper bounds

Finding upper bounds for tasks in natural language processing is a means to identifying good algorithms. Without some idea of what's possible in terms of performance, there is no way to qualify good performance. It is also a means for determining the difficulty of the task. The main approach we took to identifying an upper bound lies in the difference between state-of-the-art full parsers and a classification approach to the clause recognition task. Two other possible upper bounds are also discussed: voting and human performance.

TYPE	*precision*	*recall*	$F_{\beta=1}$
S	90.96	91.21	91.08
SBAR	88.87	87.81	88.34
SINV	88.46	83.64	85.98
SQ	66.67	53.33	58.26
SBARQ	88.89	66.67	76.19
All	90.37	90.16	90.26

Table 3.3: Performance of Collins's (1999) parser on Penn Treebank clause labels.

3.2.1 Full parser

The first approach we took to identifying an upper bound was to look at the performance of a state-of-the-art, statistical full parser on the task. For this, we used Collins's (1999) parser as it achieves $F_{\beta=1}$ of around 90, scores reported are from the Penn Treebank, and scores are reported for different clause types from the Penn Treebank. These are summarised in Table 3.3.

The first five rows are the reported scores for the Treebank clause labels. The final row contains the average performance on clauses weighted by a sample frequency. This is calculated equally for *precision*, *recall*, and $F_{\beta=1}$ as

$$\sum_{t \in \mathcal{T}} \frac{|t|}{|\mathcal{T}|} \cdot score_t \tag{3.1}$$

where $\mathcal{T}$ is the set of all Treebank clause labels and $score_t$ is the performance of the parser on label t. This prevents the low scores on the rarer clauses from overshadowing better scores on the more frequent ones. This score is some 15 points better than the performance of the best CoNLL-2001 system.

The different Treebank clause labels have no direct correlation to main and subordinate clauses. Perhaps, a better way to do this would be to run the parser on the same data used here. We could have then converted the results into the CoNLL-2001 shared task format and used the same evaluation script. However, this would have required writing a new data conversion script and we leave this to future work.

SYSTEM	FULL
Carreras & Már.	78.63
Molina & Pla	66.79
Tjong Kim Sang	66.67
Vote	*67.50*
Déjean	62.77
New baseline	53.27
Hammerton	50.42
CoNLL baseline	47.71
Patrick & Goyal	18.49

Table 3.4: Performance of majority vote.

3.2.2 Other upper bounds

We also propose two other approaches to finding upper bounds for the clause recognition task. First we'll talk a bit about voting as a performance bound. Then we'll talk about human performance, perhaps the most intuitive choice of a performance goal for any system meant to carry out a task that humans do so naturally.

3.2.2.1 Voting

Different forms of system combination have been widely reported to increase the performance of weak learners and compensate for noisy data (e.g., Dietterich 2000, Sang, Daelemans, Déjean, Koeling, Krymolowski, Punyakanok & Roth 2000, van Halteren, Zavrel & Daelemans 1998). Thus, we thought to use a majority vote over the CoNLL-2001 system results as a performance goal for an individual learner.

Simple majority voting is effective when the errors in the different classifiers are uncorrelated. Given that the CoNLL-2001 systems all had considerably different designs and employed different learners, it seems promising. Results of voting are found in Table 3.4 along with a summary of CoNLL-2001 system performances.

Simple majority voting over the third phase results did not improve the performance. This is because the individual results are indeed correlated. For instance, the best system performed well above the others because it had a more sophisticated clause segmentation approach capable of predicting multiple clause starts. Enough of the other systems only predict one clause start where there are multiple to make this be the majority in most cases where multiple starts should have been predicted. This suggests that weighted voting may have been a better approach.

Since only the system output for the final test data is available, we would have had to split this to obtain tuning data for weighted voting. A random sample could be taken of out the test results to use for this purpose. However, this would have changed the make-up of the test set somewhat making the comparison less warranted.

To gain further insight into the poor scores of the voted results, we also calculated pairwise agreement using the kappa statistic to estimate system correlation. The kappa statistic is an index of agreement commonly used to rate scorer reliability that accounts for chance agreement. The value lies on the interval $[-1, 1]$ where a score of -1 indicates total disagreement, a score of 0 indicates chance agreement, and a score of 1 indicates total agreement.

Total pairwise agreement rounded to the fourth decimal point between the nine system output files available for phase 3 of the clause recognition task[1] is 0.7194. This indicates a high level of agreement and enforces the conclusion that voting over the third phase results is not a good approach as the output from the various systems is correlated.

3.2.2.2 Human performance

Finally, we considered evaluating human performance on the clause recognition task. This would consist of giving the participant a set of instructions describing the Treebank style and leaving the performance of the task beyond that to intuition. However, as perhaps we should have gleaned from the size of the the Treebank bracketing guidelines (Bies et al. 1995), the clause structure turned out to be too counter-intuitive to realistically estimate human performance. Again, we leave this to future work.

[1]These files can be found on the CoNLL-2001 shared task web site at <http://cnts.uia.ac.be/conll2001/clauses/>.

3.3 Summary

This chapter discussed different upper and lower bounds for the clause recognition task. We presented a simple symbolic baseline based on the work of Leffa (1998) that represents a considerable improvement over the CoNLL-2001 baseline for the first two phases and a significant, if not nearly as impressive, improvement in the third and final phase. We also looked at the performance of Collins's (1999) parser on clause constituents, which indicates that a classification approach to clause recognition still has a way to go in terms of performance. Finally, we suggested a couple of other upper bounds for the task: voting and human performance.

Chapter 4

Learning Approaches

Chapter 1 described how clause segmentation can be cast as a classification problem. This chapter describes the model classes used in the experiments described in Chapter 6. First, we present the Winnow algorithm and the Sparse Network of Winnows (SNoW) software (Carlson, Cumby, Rosen & Roth 1999).[1] Second, a maximum entropy approach employing the openNLP maxent package (Baldridge, Morton & Bierner 2001)[2] is described.

4.1 Winnow

As we will see, the Winnow algorithm seems an obvious choice for the task of clause recognition as it is efficient in learning linear separators for problems with many irrelevant attributes. More importantly, it is capable of finding reasonable linear bounds even when the data is not linearly separable. Furthermore, Winnow proved very successful in follow up work (Zhang et al. 2001) to the CoNLL-2000 chunking task. In fact, the best scores reported on the task are from a regularised version of the Winnow algorithm guaranteed to converge when the data is no perfect linear separator exists through the sample space. The approach we take here is not regularised, however, re-

[1]The SNoW software and the Fex feature extractor along with relevant papers are available from <http://l2r.cs.uiuc.edu/ danr/snow.html>.

[2]The openNLP maximum entropy software is open source and can be acquired from <http://maxent.sourceforge.net/>.

sults are comparable to the mean CoNLL-2001 score on the clause identification task (see Table 6.4).

4.1.1 Theory

Winnow is an online mistake-driven algorithm which learns a linear separator in the feature space (Dagan, Karov & Roth 1997). The algorithm is designed for efficiency in separating out irrelevant attributes (Littlestone 1988). Indeed, the Winnow mistake bound grows linearly with the number of relevant features and only logarithmically with the total number of features. Winnow makes no assumptions (e.g., about independence) on the features where other (typically Bayesian) methods do. Furthermore, as mentioned above, Winnnow is fairly successful even when no perfect separator exists. This subsection discusses the intuitive idea behind Winnow before the algorithm is presented in the next subsection.

Winnow is an as online learner because learning takes place in a sequence of trials where a trial is the prediction and subsequent update carried out for each labelled example extracted from the training set (Dagan et al. 1997). Online algorithms first make a prediction and then use the correct label as feedback. For Winnow, this feedback determines whether the feature weight should be increased or decreased in the update stage for each trial.

Winnow is also said to be mistake-driven. This is because it falls into the class of algorithms that update their hypothesis only when a mistake is made in the prediction phase of a given trial. Feature weights are discounted for negative examples (a label predicted when it is not meant to be) and promoted for positive examples (a label not predicted when it is present in the training example). This makes the algorithm more sensitive to relations between features (Dagan et al. 1997).

Finally, we mentioned above that SNoW learns a linear separator. Linear algorithms such as SNoW are thus named because they are methods for fitting a hyperplane to n-dimensional data which separates the positive from the negative examples. That is they learn a linear function of n dimensions of the general form

$$a_{\vec{w}}(\vec{x}) = \sum_i (w_i^t)^{x_i} \tag{4.1}$$

where x_i is 1 if the i^{th} feature is present/active in the input and 0 otherwise; and w_i^t is the weight for the i^{th} feature in the function corresponding to the target t. The result of this function for a particular example is a number which can be compared to a threshold to determine whether it represents a positive or negative prediction. In summary, we learn a linear function for each class label t and compare this with a threshold θ_t to determine if that target is predicted to be active by the system. Since the classes are not binary for all of our tasks, SNoW incorporates a decision support mechanism that chooses the class with the highest activation. Also, confidence values can be obtained for a given target and example.

4.1.2 Architecture

SNoW is a sparse network because it implements the Winnow representation in an efficient manner with single feature nodes connecting to all target nodes. Furthermore, the linear function is sparse in that each target is evaluated as a "function of a (small) subset of all the features known to the system." That is,

1. Input features are only allocated in the implementation if they are active in the input example.

2. A non-zero weight exists between a feature and a target node only if the feature has been active in an example labelled with the target node.

The parameters, update rules, and evaluation function for the SNoW implementation are summarised in Table 4.1. The following is a brief description of the learning algorithm.

The weight vector $\vec{w}$ is initialised with equal weights (θ/d where d is the average number of active features in a document making the weights close to θ.) For each labelled example in the training data, SNoW evaluates the linear function for each target using the current weight vector. If a given target is wrongly predicted, it is considered to be a negative example for that label. The appropriate update rule from Table 4.1 is then applied if the the prediction is incorrect. If 1 is wrongly predicted, all weights are promoted (i.e., multiplied by the promotion parameter α). If 0 is wrongly predicted, all weights are multiplied instead by the demotion parameter β.

PARAMETERS	
Name	Description
$\mathcal{A}_t = \{i_1, ..., i_m\}$	The active feature set.
w_i^t	The weight on the edge connecting the i^{th} feature to the target node t.
θ_t	The threshold at target t.
$\alpha > 1$	The promotion parameter.
$0 < \beta < 1$	The demotion parameter.

TRAINING	
Condition	Update
$\Sigma_{i \in \mathcal{A}_t} w_i^t \leq \theta_t$	$\forall i \in \mathcal{A}_t, w_i^t \leftarrow \alpha \cdot w_i^t$
$\Sigma_{i \in \mathcal{A}_t} w_i^t > \theta_t$	$\forall i \in \mathcal{A}_t, w_i^t \leftarrow \beta \cdot w_i^t$

TESTING
A linear unit is active given a set of active features iff $\Sigma_{i \in \mathcal{A}_t} w_i^t > \theta_t$. A decision support mechanism determines the dominant active target node.

Table 4.1: SNoW learning architecture

Once a model is learned, we can query the system with examples to see what label is predicted by the learned linear function. When presented with an example, the information present is propagated through all the subnetworks. This will produce 'weights' for each label. The decision support mechanism is then called to choose the target whose subnetwork has the highest activity.

4.2 Maximum entropy

Maximum entropy (Berger, Della Pietra & Della Pietra 1996) is another obvious choice for the task of recognising clauses. Again, it is a method robust to high-dimensional feature spaces. It makes no independence assumptions about the features and provides a convenient framework for the integration of linguistic knowledge. Furthermore, as mentioned in Chapter 2, it has shown state-of-the-art results in other NLP tasks such as POS tagging and full parsing.

4.2.1 Theory

The idea behind maximum entropy is simple: we want to model all and only the information present in the empirical distribution–in our case, the training corpus described in Appendix B. We go about this by finding the most uniform model subject to the set of constraints in the training data. This begs several questions:

1. Why do we want the most uniform model?
2. What are the constraints?
3. How do we measure uniformity? boolean

Answering these questions will lead us to the principle of maximum entropy and give us the concepts to introduce the learning algorithm in the next subsection.

Why do we want the most uniform model? It is common in statistical NLP to make simplifying independence assumptions. This will make a model converge to the wrong answer, but makes statistical modelling of complex phenomena (e.g., natural language) a tractable problem. The maximum entropy framework avoids this modelling error. It makes as few assumptions as possible by minimising the distance between the learned model and the distribution that assumes nothing. Thus, the search for the most uniform model. It ensures that we do not "add constraints to the model that are not justified by the empirical evidence available to us (Manning & Schütze 1999, p. 589)."

What are the constraints? Constraints are relations, encoded in features, that are recovered from the data. It is important to note here that the notion of feature used in the context of maximum entropy is slightly different from the common notion of a feature as referring only to some characteristic of the object being modelled. Maximum entropy features are functions that map input pairs of traditional features (i.e., items representing the existence of a single pieces of evidence) and class labels to values representing whether or not the pair is present in the current example: $f : \vec{\mathcal{X}} \times \mathcal{C} \mapsto \{0,1\}$.

(4.2)

$$f_j(\vec{x},c) = \begin{cases} 1 & \text{if } c = \text{S and SentStart=1} \\ 0 & \text{otherwise} \end{cases}$$

Example 4.2 is a feature from the clause recognition task. c is the class label of the example and $\vec{x}$ is the evidence. It says that the feature defined by the evidence $\vec{x}$ and the class label c is true if $c = \mathrm{S}$ and the current position is the beginning of a sentence.

How do we measure uniformity? The answer to this question is the basis of our model class and provides its name. Manning & Schütze (1999, p. 61) describe entropy (or self-information) as a measure of the average uncertainty of a random variable. This can be written as

$$H(p) = -\sum p \cdot \log p \tag{4.3}$$

Where the log probability $\log p(x)$ is the information content of the event, $H(p)$ can be seen as the expected value of the amount of information in the variable. As the probability distribution gets closer to uniform, $H(p)$ goes to 1. Thus, we maximise $H(p)$ to find the model that is closest to uniform and therefore assumes nothing about that which is unknown.

We stated earlier that we want the model closest to uniform given the observed set of constraints. To ensure that the model is also closest to the empirical distribution, we require that

$$E_{p_*} f_i = E_{\tilde{p}} f_i \tag{4.4}$$

That is, we restrict our attention to those models which agree with the training sample on 'how often the output of the process should exhibit the feature f_i (Berger et al. 1996).'

Which brings us to the principle of maximum entropy. Where the set of possible models is defined as $\mathcal{P} = \left\{p | E_{p_*} f_i = E_{\tilde{p}} f_i \text{ for } i \in \{1, 2, ..., k\}\right\}$, Berger et al. (1996, p. 8) state the principle as follows:

> To select a model from a set $\mathcal{P}$ of allowed probability distributions, choose the model $p_* \in \mathcal{P}$ with maximum entropy $H(p)$:

$$p_* = \arg\max_{p \in \mathcal{P}} H(p) \tag{4.5}$$

This formalises exactly what we've described intuitively above. Namely, the solution is the model with the highest entropy from the subset of all possible distributions consisting of those that are subject to the observed constraints.

PARAMETERS	
Name	Description
n	iteration
$\vec{x}$	possible context
c	possible label
C	sum-of-features constant
α_i	feature 'weight' for f_i

TRAINING	
Condition	Update
until parameters converge	$p^{(n)}(\vec{x},c) = \frac{1}{Z}\prod_{i=1}^{K+1}(\alpha^{(n)})^{f_i(\vec{x},c)}$ Compute $E_{p^{(n)}}$ for all $1 \le i \le K+1$ $\alpha_i^{(n+1)} = \alpha_i^{(n)}\left(\frac{E_{\tilde{p}} f_i}{E_{p^{(n)}} f_i}\right)^{\frac{1}{C}}$

TESTING
$c = \arg\max_{c \in C} p_*(\vec{x},c)$

Table 4.2: openNLP maxent learning architecture

4.2.2 Architecture

It can be shown (e.g., Ratnaparkhi 1997) that the p_* distribution defined in equation 4.5 must be loglinear of the following form:

$$p_*(\vec{x},c) = \frac{1}{Z}\prod_{i=1}^{k}\alpha_i^{f_i(\vec{x},c)},\ 0 < \alpha_i < \infty \tag{4.6}$$

where Z is a normalising constant and α is the weight for feature f_i. Now we have the pieces we need to describe the generalised iterative scaling algorithm (Manning & Schütze 1999, Ratnaparkhi 1997) used in the openNLP maxent package. Table 4.2 summarises the parameters, updates, and evaluation function for maximum entropy.

The openNLP maxent package used here employs the Generalise Iterative Scaling (GIS) algorithm to find the maximum entropy distribution subject to the observed constraints. The GIS algorithm requires that the sum of the features be equal to some constant C defined to be the greatest possible feature sum (Manning & Schütze 1999, Ratnaparkhi 1997). This involves adding an extra 'correction' feature which is why the $p^{(n)}$ update formula requires the product from 1 to $K+1$.[3] Darroch & Ratcliff (1972) give a proof of convergence.

In training, then, the GIS algorithm iterates over the steps outlined for the re-estimation of of the parameters α_i.

1. compute $p^{(n)}$ for each example in the training set
2. compute the expected value according to the model $p^{(n)}$ of f_i for each feature where $E_{p^{(n)}} f_i$ is approximated by $\frac{1}{N} \sum_{j=1}^{N} \sum_c p(c|\vec{x}_j) f_i(\vec{x}_j, c)$
3. update the 'weight' parameters

This terminates when the change in log-likelihood is negligible or after a specified number of iterations. In the results reported here, we will use a fixed number of 100 iterations.

In testing, we choose the class label that maximises the probability of the the final model p_*. This is computed with the learned features weights α_i and the model from Equation 4.6.

4.3 Summary

This chapter presented the model classes that we use in our experiments with clause recognition. Winnow and maximum entropy are good choices for the task as they avoid making unwarranted independence assumptions and are robust in high-dimensional feature spaces. Furthermore, both of the implementations offer the option of getting a confidence value for a target label given a test example. This will prove important in our approach to clause segmentation described in Section 6.2.3.2.

[3] In a somewhat confusing bit of notation, this sum-of-features constant is different than $\mathcal{C}$, which represents the set of all labels. They *can* be told apart however as the set of all labels will be written in a script font.

Chapter 5

Features and Encoding

As mentioned earlier, choosing relevant features is integral to a classification approach. The appendix on feature extraction discusses the mechanics of feature extraction and encoding employed in this thesis. In this chapter, we talk a little bit about feature encoding and summarise the set of features we used for the experiments reported in the next chapter. Throughout this chapter we will use the sentence in Table 5.1 where the first column corresponds to the sentence position.

5.1 Features

The main set of features used in our experiments are those described by Carreras & Màrquez (2001*b*). These provided their system a distinct advantage over the others and improved scores in the follow-up work of Sang & Déjean (2001). These are divided into three categories below: word, POS, and chunk window; sentence features; and sentence patterns.

5.1.1 Word, POS, and chunk window

The first set of features are those immediately available from the data: the word, POS, and chunk tokens for each sentence position and in the context to either side. For our experiments, a symmetrical window is defined by the amount of positions to be considered on either side of the current sentence position. Thus a window size of 1 in-

SENT POS	WORD	POS	CHUNK	START
1	Confidence	NN	C-NP	S
2	in	IN	C-PP	X
3	the	DT	B-NP	X
4	pound	NN	E-NP	X
5	is	VBZ	B-VP	X
6	widely	RB	I-VP	X
7	expected	VBN	I-VP	X
8	to	TO	I-VP	X
9	take	VB	E-VP	X
10	another	DT	B-NP	X
11	sharp	JJ	I-NP	X
12	dive	NN	E-NP	X
13	if	IN	C-SBAR	S
14	trade	NN	B-NP	S
15	figures	NNS	E-NP	X
16	for	IN	C-PP	X
17	September	NNP	C-NP	X
18	COMMA	COMMA	O	X
19	due	JJ	C-ADJP	X
20	for	IN	C-PP	X
21	release	NN	C-NP	X
22	tomorrow	NN	C-NP	X
23	COMMA	COMMA	O	X
24	fail	VBP	B-VP	X
25	to	TO	I-VP	X
26	show	VB	E-VP	X
27	a	DT	B-NP	X
28	substantial	JJ	I-NP	X
29	improvement	NN	E-NP	X
30	from	IN	C-PP	X
31	July	NNP	B-NP	X
32	and	CC	I-NP	X
33	August	NNP	E-NP	X
34	’s	POS	B-NP	X
35	near-record	JJ	I-NP	X
36	deficits	NNS	E-NP	X
37	.	.	O	X

Table 5.1: The corpus sentence used as an example throughout this chapter.

dicates that features for the word, POS, and/or chunk labels (as specified) are included from one position to either side of the current position as well as those at the current position.

The encoding is aimed at keeping the example files as small as possible. Therefore, word, POS, and chunk tokens are represented by `0`, `1`, and `2` respectively. The relative position is specified for each window feature by a `-` or `+` indicating whether it is to the left or right of the current position. This is followed by an integer which indicates how many positions to the right or left it is from the current position.

For example, if we specify a word, POS, and chunk window of 1 for the second position of the example sentence, the following training/testing example will be generated.

(5.1)
```
0-1=Confidence 1-1=NN 2-1=C-NP 0+1=the 1+1=DT 2+1=B-NP 0+-0=in
1+-0=IN 2+-0=C-PP X
```

The first three features are the word, POS tag, and chunk label for the previous position. The next three features are the same for the position following 'in'. The seventh, eight, and ninth correspond to the current position. And, the final token is the target label.

These features capture constraints of the immediate context. This kind of immediate context is successful for applications like POS tagging and spelling correction where the most relevant information is found close at hand. However, for the task of identifying the embedded phrase structure of clauses, features such as whether or not there is a verb phrase anywhere before the current position or whether there is a verb phrase followed by a period after the current position may provide more useful information. Such features are described in the following subsections.

5.1.2 Sentence features

The second category of features used in the main experiments is referred to as sentence features. These are an attempt to capture long-distance dependencies. Sentence feature items correspond to things that have some part in the definition of a clause. Four features are generated for each sentence feature item specified: (1) whether or not it can be found on the left, (2) number of occurrences to the left, (3) whether or not it can be found on the right, and (4) number of occurrences to the right.

For the corpus sentence from Table 5.1, if we specify a verb phrase sentence feature for the 13th position along with a word, POS, and chunk window of 1, the following example will be generated.

(5.2)
```
0-1=dive 1-1=NN 2-1=I-NP 0+1=trade 1+1=NN 2+1=B-NP 0+-0=if
1+-0=IN 2+-0=B-SBAR ExL-EC-VP=0 CL-EC-VP=0 ExL-BC-VP=1
CL-BC-VP=1 S
```

The features `ExL-EC-VP=0`, `CL-EC-VP=0`, `ExL-BC-VP=1`, and `CL-BC-VP=1` are added to the window features and correspond in order to the descriptions 1-4 listed above.

5.1.3 Sentence patterns

The third and final category of features is referred to as sentence patterns. These are an attempt capture generalisations about the order of sentence features that the plain existence and count features miss. A sentence pattern consisting of the items specified is generated for the left context and the right context with respect to the current position.

Once again looking at the 13th position of the sentence in Table 5.1, with verb phrases, commas, coordinate conjunctions, and full stops specified to be part of the sentence pattern, we get the following example.

(5.3)
```
0-1=dive 1-1=NN 2-1=I-NP 0+1=trade 1+1=NN 2+1=B-NP 0+-0=if
1+-0=IN 2+-0=B-SBAR BP0=B-VP_E-VP_COMMA
AP0=COMMA_COMMA_B-VP_E-VP_CC_.  S
```

The `BP0=B-VP_E-VP_COMMA` feature expresses the fact that there is verb phrase beginning followed by a verb phrase end followed by a comma from the beginning of the sentence to the current position. The second sentence pattern feature (encoded in the examples above as `AP0=COMMA_COMMA_B-VP_E-VP_CC_.`) is the pattern from the current position to the end. It specifies that there is a comma followed by a second comma followed by a the beginning of a verb phrase followed by the end of a verb phrase followed by a coordinate conjunction followed by a period.

	SNoW	MAXENT
multiple	89.68	90.43
single	89.87	90.80

Table 5.2: Results of experiments with number of sentence boundary features.

5.2 Encoding

Besides the features used, the manner in which the features are represented also affects scores. To illustrate, this section presents a couple of experiments with encoding of sentence boundaries. Sentence boundaries make up a category of features that are not part of the main experiments, but whose representation affected scores in several early experiments.

Table 5.2 compares the $F_{\beta=1}$ scores of having multiple sentence boundary features to the results of having only one sentence boundary feature representing in the clause start phase whether the current position is the beginning of a sentence and in the clause end phase whether the current position is the end of a sentence. This was run with a word, POS, and chunk window of 1 to either side.

The *multiple* row corresponds to there being a sentence boundary feature for each window item specified (i.e., word, POS, and chunk). An example for a position at the beginning of a sentence is encoded as

(5.4)
```
0+1=in 1+1=IN 2+1=C-PP 0+-0=Confidence 1+-0=NN 2+-0=C-NP
0_BegSent=1 0_BSExRVP=1 1_BegSent=1 1_BSExRVP=1 2_BegSent=1
2_BSExRVP=1 S
```

where the numbers `0`, `1`, and `2` at the beginning of the `BegSent=1` and `BSExRVP=1` features corresponds to word, POS and chunk labels respectively. The *single* row corresponds to an encoding where only one sentence boundary feature is generated for each begin and end position. This is encoded as

(5.5)
```
0+1=in 1+1=IN 2+1=C-PP 0+-0=Confidence 1+-0=NN 2+-0=C-NP
BegSent=1 BSExRVP=1 S
```

	SNoW	MAXENT
simple	86.66	88.50
start-end	89.37	90.16
start-end-vp	89.87	90.80

Table 5.3: Results of sentence boundary encoding experiments.

where the extra sentence boundary features have been removed. As Table 5.2 illustrates, both algorithms perform better with the encoding that eliminates the redundant features.

Further preliminary experiments explored other options in encoding sentence boundaries. Table 5.3 compares these $F_{\beta=1}$ scores. Again this was run with a word, POS, and chunk window of 1 to either side.

The *simple* row corresponds to having no explicit sentence boundary feature, which is encoded as

(5.6) `0+1=in 1+1=IN 2+1=C-PP 0+-0=Confidence 1+-0=NN 2+-0=C-NP S`

The *start-end* row corresponds to having a single explicit sentence boundary feature. `BegSent=1` is added to the evidence above giving:

(5.7)
```
0+1=in 1+1=IN 2+1=C-PP 0+-0=Confidence 1+-0=NN 2+-0=C-NP
BegSent=1 S
```

Finally, *start-end-vp* corresponds to the addition of a single crossed feature where there is a verb phrase in the sentence. This adds `BSExRVP=1` to the evidence for a sentence start position from the *start-end* encoding

(5.8)
```
0+1=in 1+1=IN 2+1=C-PP 0+-0=Confidence 1+-0=NN 2+-0=C-NP
BegSent=1 BSExRVP=1 S
```

The `BSExRVP=1` feature is intended to capture the situation where there is no clause in a sentence such as article titles. In these cases, there should be no full, main clause marked at the top level. This seems to capture the intended phenomena, showing an improvement roughly proportional to occurrence in the corpus.

In conclusion, several early experiments comparing pure sentence boundary features with crossed features representing the fact that there is a sentence boundary at the current position and a verb phrase in the sentence illustrate that feature encoding affects classifier scores. The sentence boundary encoding we adopted for the work in this thesis is the latter.

5.3 Summary

This chapter has presented the features that are the basis for the development experiments described in the Chapter 6. In addition to the word, part-of-speech, and chunk context directly available in the data, we employ sentence features and sentence patterns intended to capture long-distance dependencies and generalisations about the order of clause indicators in the sentence. We also described an early experiment with the encoding of sentence boundaries.

Chapter 6

Experimental Results

This chapter reports on the optimisation experiments carried out with Winnow and maximum entropy classifiers on the features described in the last chapter. The first section describes the experimental design for the three phases of the clause recognition task (starts, ends, and full segmentation). The second section describes the implementation and reports on the final results.

6.1 Experiment design

Experiments were run on each of the three main feature types described in the previous chapter for each of the steps of the three-phase approach to clause recognition first described by Leffa (1998) and subsequently used by Sang & Déjean (2001) in formulating the CoNLL-2001 shared task. Table 6.1 contains a key of the parameters used to represent the feature items of each feature type. In addition to these, already-predicted S- and E-points are used when they are available in later stages for feature window, sentence features, and sentence patterns.

For each phase, we first tested all combinations of window size from zero to three with window features. Next, using the optimised window, we ran separate experiments for sentence features and sentence patterns. First, each possible sentence feature element defined in the previous section was tested singly. The sentence feature elements that improved the score were then tested in conjunction to see if they further improved

WINDOW FEATURES	
PARAM	DESCRIPTION
w,p,c	word, part-of-speech tag, chunk tag
0..3	size of symmetrical window to either side of current position

SENTENCE FEATURES		
PARAM	CORPUS	DESCRIPTION
v,w,d	[BEC]-VP WP$ WP	verb phrase, (possessive) *wh*-pronoun
q,n,o,c,m,l,p	“ ” ( ) , : .	words whose POS is punctuation
t	that	the word ‘that’

SENTENCE PATTERNS		
PARAM	CORPUS	DESCRIPTION
v	[BEC]-VP	begin, end, and close of verb phrases
w	{WP,WP$,WDT,WRB}	subordinating conjunctions
q,n,o,c,m,l,p	“ ” ( ) , : .	words whose POS is punctuation
j	CC	coordinating conjunction
t	that+POS	the word ‘that’ plus its part of speech

Table 6.1: Feature items parameter key for window, sentence features, and sentence patterns.

the score. Finally, we tested different combinations of sentence patterns items. These combinations were chosen on the basis of what we thought would be effectual in the given phase and, where they are the same across the feature types, pattern items that improved scores as feature items.

6.2 Implementation and results

In this section we report on the experiments in each phase and describe the implementations and final configurations. We consider the three phases in order: clause starts, clause ends, clause segmenting. The first two phases are straight-forward. The third phase, however, is further divided into two subtasks. Finally, we report on the approach and scores taken here to distinguishing main and subordinate clauses. A summary of these results is found in Table 6.2.

SPARSE NETWORK OF WINNOWS

PHASE	FEATURE SET	*Precision*	*Recall*	$F_{\beta=1}$
1. starts	wpc–v–vwnocmplqjt–1	87.02/88.39	85.60/82.12	86.30/85.14
2. ends	wpc–snt–svmlj–1	88.22/88.07	75.76/73.93	81.51/80.38
3. segment	swc–e–enmp–0	76.82/77.74	60.99/58.73	68.00/66.91

OPENNLP MAXIMUM ENTROPY

PHASE	FEATURE SET	*Precision*	*Recall*	$F_{\beta=1}$
1. starts	wpc–v–vlj–1	93.69/91.64	85.20/82.43	89.25/86.79
2. ends	pc–s–svwmt–2	90.48/88.39	84.21/83.06	87.23/85.64
3. segment	sewpc–sevnmp–evmp–1	84.05/81.64	70.19/67.57	76.50/73.94

Table 6.2: Development/Test scores summary on optimised feature set. Features are encoded according to the parameter key in Table 6.1 as WindowFeatures–SentenceFeatures–SentencePatterns–WindowSize.

6.2.1 Predicting clause starts

This subsection reports on experiments with phase 1 of the clause recognition task; that is, experiments predicting clause starts. A position that begins a single clause and a position that begins multiple clauses are considered equal in this phase. Start positions are tagged with a `S` label and non-start positions are tagged with a `X` label.

In the experiments for this phase, we consider all combinations of window size and window features as described above. The window is centred around the position under consideration. For sentence features, we generate count and existence features to the right and left of the position under consideration as described in the previous chapter. Finally, for the sentence patterns experiment, we generate a pattern from the start of the sentence to the current position and from the current position to the end of the sentence.

differentThe final configuration for clause starts employs a word, chunk, and POS window of 1 position to either side of that under consideration for both Winnow and maximum entropy. The only sentence feature item that showed significant improvement is the verb phrase. The two systems did optimise for different sentence patterns.

The optimised version generates sentence patterns of verb phrases, subordinating conjunctions, coordinating conjunctions, the word that plus its POS, open parenthesis, closes parenthesis, comma, period, colon, open quote and close quote punctuation parts of speech. The final maximum entropy version generates features for verb phrases, colons, and coordinating conjunctions.

6.2.2 Predicting clause ends

This subsection presents phase two experiments for the clause recognition task. Here, a classifier is trained to recognise sentence positions where one or more clauses are terminated. As was the case with clause starts in the Section 6.2.1, single and multiple ends are considered equal in this phase. End positions are tagged with a `E` label and non-end positions are tagged with a `X` label.

Experiments are identical to those described for sentence starts, except that predicted sentence starts are included in all feature types. The final configuration for the clause end classifier was in all features for the Winnow and maximum entropy predictors.

For the Winnow configuration, we generate word, chunk, and part-of-speech features for a window size of one. Predicted starts, end quotes, and the word ‘that’ are generated for sentence features. Sentence patterns incorporate predicted starts, verb phrases, commas, colons, and coordinating conjunctions.

The final maximum entropy configuration uses POS and chunk window features two to either side. Only predicted starts are used for sentence features. Finally, sentence patterns are generated from predicted starts, verb phrases, subordinating conjunctions, commas, and the word ‘that’ with its part of speech.

6.2.3 Clause segmenting experiments

The most interesting part of our approach is the method used for segmenting clauses. Clause segmenting was split up into two sub-phases. In the first sub-phase, a classifier was trained to predict which clause starts should be double starts. The second subphase did the actual segmenting. A predictor was trained to recognise full clauses and the weights obtained from the classifier were then used to inform a segmenting algorithm.

This follows the approach of Carreras & Màrquez (2001*b*). However, they had another phase between the two described here. The extra phase predicted which double starts should be changed to triple starts. They found the data to sparse to predict more than triple starts. In our experiments, a reliable predictor could not be trained on the amount of data available for predicting triple from double starts.

6.2.3.1 Predicting double starts

In this sub-phase of segmenting, we trained a classifier to predict when a start position predicted in the first phase has two or more starts. Obviously, there is considerably less training data for this as we are not considering every sentence position but only those where a start has been predicted. As mentioned above, Carreras & Màrquez (2001*b*) have another phase which predicts which double starts should be incremented to triple starts. However, we found the data to be too sparse to train an effective double-to-triple classifier. In this sub-phase, positions that start a single clause are tagged with `S` and positions starting two or more are tagged with `SS`.

The features are a bit different for predicting double starts. We still ran experiments on all combinations of window features and window size centred around the predicted start under consideration. Sentence features once again consider count and existence features to the left and right of the current predicted start. For sentence patterns, however, we now generate a pattern for each possible clause (one pattern from the current predicted start under consideration to each predicted end to the right) as well as the patterns from the beginning to the current position and from the current position to the end.

The Winnow configuration for this sub-task consists of a 0-sized feature window of predicted starts, predicted ends, words, and chunk tags. The optimised maximum entropy configuration generates predicted starts, words, and chunk tags for a window size of one. Neither model class benefited from sentence features or sentence patterns. $F_{\beta=1}$ scores on the test set for this sub-phase are 85.78 for Winnow and 87.37 for maximum entropy. This is enough to improve the performance of the segmenting

```
if there is a candidate corresponding to whole sentence
    then
    remove from heap and add to parse
    remove any candidates with crossing boundaries
    if a double start was not predicted in the previous sub-phase
        then remove any candidates starting at same position
while the candidate heap is non-empty
    remove top heap item and add to parse
    remove any candidates with crossing boundaries
    if a double start was not predicted in the previous sub-phase
        then remove any candidates starting at same position
```

Figure 6.1: Clause segmentation algorithm.

algorithm described below.

6.2.3.2 Segmenting algorithm

As mentioned above the segmenting algorithm relies on confidence values for clause candidates. These confidence values are obtained by training a classifier to recognise whether a candidate is a clause or not. The features for the segmenting experiments were drawn from word, POS, and chunk windows, sentence features, and sentence patterns.

The sentence is subsequently parsed into clauses based on the confidence level of each candidate being a clause. At each iteration, the clause with the highest confidence value for being a clause is added to the final parse. Following this, all candidates that cross brackets with the selected clause are removed from the heap. This is also where the double starts come in; we remove from the heap any clauses starting at the same position if a double start was not predicted there. This algorithm is summarised in Figure 6.1.

This assumes that the candidates and their weights are available. In our implementation, they are read from a file and matched with their confidence values in the output from the classifier and stored in a searchable heap data structure. The first half of the algorithm summarised in Figure 6.1 ensures that if the full sentence is a clause candidate, then it is always marked as a clause. This raised $F_{\beta=1}$ scores nearly 20 points in early segmentation experiments.

The experiments training a classifier to recognise clauses were based on the utility of the resulting confidence values in the clause segmenting. As in the sub-phase predicting double starts, the features used here are a bit different. First, there are two more sentence features: existence and count features are generated before, inside, and following each clause candidate. Sentence patterns are generated only from the beginning of the clause candidate to the end. Window features are centred on the clause candidate start position.

The final Winnow configuration for this phase consists of a feature window of size zero using predicted starts, words, and chunk tags. Only predicted ends are considered for sentence features. Sentence patterns are generated using predicted ends, end quotes, commas, and periods. The optimised maximum entropy configuration uses all sentence feature items in a size one window. Sentence features are generated from predicted starts, predicted ends, verb phrases, end quotes, commas, and periods. Sentence patterns representing the clause candidate incorporate predicted ends, verb phrases, commas, and periods.

One final note: Carreras & Màrquez (2001*b*) chose a multi-phase approach to predicting multiple starts because they were using a binary predictor. Since the Winnow and maximum entropy software we employ handles more than two targets, a further experiment could train a single classifier to predict the number of starts at a given position. We leave this to future work.

6.2.4 Main vs. subordinate

The approach to distinguishing main and subordinate clauses is very similar to that taken in the symbolic algorithm of Chapter 3. Again, all top-level clauses are assigned a main label. The difference here is that a main label is also assigned to clauses embedded one level under the first if there is more than one clause at this level. This is intended to capture top-level compound sentences. As was the case with the symbolic algorithm in Chapter 3 (see Table 3.2), we expect the overall score to be the same as we

SPARSE NETWORK OF WINNOWS

SYSTEM	*Precision*	*Recall*	$F_{\beta=1}$
Main clauses	84.39	87.39	85.86
Subordinate clauses	60.05	34.37	43.72
Total	77.74	58.73	66.91

OPENNLP MAXIMUM ENTROPY

SYSTEM	*Precision*	*Recall*	$F_{\beta=1}$
Main clauses	84.42	86.20	85.30
Subordinate clauses	67.38	46.62	55.11
Total	81.64	67.57	73.94

NEW SYMBOLIC BASELINE

SYSTEM	*Precision*	*Recall*	$F_{\beta=1}$
Main clauses	98.44	87.45	92.62
Subordinate clauses	27.25	39.23	32.16
Total	79.24	48.35	53.27

Table 6.3: SNoW and openNLP maxent phase 3 scores for main and subordinate clauses.

are simply assigning a main or subordinate label to each of the clauses and not adding or removing any labels. Table 6.3 summarises these results.

The results show the same pattern as we saw in the results from the symbolic baseline (reproduced in the table). As in the symbolic algorithm, the large difference in performance between main and subordinate clauses reflects our simple definition of a main clause and the relative ease of identifying top-level clauses compared to embedded clauses.

One interesting point to note is that main clauses have more of a problem with over-recognition, characterised by *precision* being lower than *recall*. Subordinate clauses, on the other hand, are characterised by *recall* sores being lower than *precision*, thus exhibiting more of a problem with under-recognition. This is the opposite of the symbolic baseline where main clauses exhibit more of a problem with under-recognition and subordinate with over-recognition

System		Starts	Ends	Full
Adaboost	(Carreras & Már.)	91.72	89.22	78.63
maxent	*Current work*	*86.79*	*85.64*	*73.94*
HMMs	(Molina & Pla)	87.74	78.61	68.12
MBL	(Sang)	88.82	82.28	67.79
SNoW	*Current work*	*85.14*	*80.38*	*66.91*
Adaboost	(Patrick & Goyal)	87.27	81.76	66.17
ALLiS	(Déjean)	87.43	65.47	62.77
Connectionist	(Hammerton)	-	-	50.42
New baseline	Current work	74.56	74.73	53.27
Baseline	CoNLL-2001	53.34	65.34	47.71

Table 6.4: $F_{\beta=1}$ scores for the CoNLL-2001 systems

6.3 Discussion

The performance of both Winnow and maximum entropy are quite good. Collectively they show results comparable to or better than the bulk of the CoNLL-2001 systems with scores for maximum entropy approaching those of Carreras & Màrquez's (2001*b*) system, which achieved $F_{\beta=1}$ scores more than 10 points higher than the second best system in assigning fully-segmented, embedded clause structure. Table 6.4 repeats the results of the CoNLL-2001 systems ordered by scores on the final phase with the results from our Winnow and maximum entropy approaches in italics.

The Winnow results for segmentation perform on a level with the bulk of the systems. Ranked among the CoNLL-2001 system results, it is the 4th best with as many systems performing below as above. Nevertheless, the results are somewhat disappointing. We suspect that this was due at least in part to Winnow being more sensitive to the noise introduced in the POS and chunk tagging stages of the data preparation.

To test the hypothesis that Winnow is more sensitive to noisy data, we ran SNoW on the test data with the Treebank POS tags and Treebank-derived chunk labels substituted for the automatically assigned ones. These tests show that the SNoW $F_{\beta=1}$ score for the final phase jumps from 66.91 to 71.71 with the Treebank tags. The maximum entropy scores change a bit less from 73.94 to 76.58.

As expected, the Winnow results improved more with the correct POS and chunk labels than did the maximum entropy results. However, the difference is not striking and does not account for the Winnow performance. We assume that the remaining difference lies in the fact that Winnow is a linear separator algorithm. Although it generally performs well even where no perfect separator exists, it does not provide a framework for modelling all of the dependencies as does maximum entropy.

The maximum entropy results are considerably more impressive. The performance is second only to the boosted decision trees of Carreras & Màrquez (2001*b*) both in predicting clause ends and in full segmenting. The final, segmenting score, though some 5 points short of the best system, achieves a $F_{\beta=1}$ score some 5 points better than the second-best system from the CoNLL-2001 shared task.

6.4 Summary

In this chapter, we have presented and discussed the experimental results. Both Winnow and maximum entropy showed good results, with maximum entropy substantially outperforming the bulk of the CoNLL-2001 systems achieving results second only to Carreras & Màrquez (2001*b*). We hypothesised that the difference between the two learners is due to the fact that Winnow is more sensitive to noise. An experiment showed that while this may account for some of the difference, it does not explain it away. Maximum entropy has clearly shown better results for this task.

Chapter 7

Conclusions

The previous chapters introduced the problem of clause recognition, discussed previous approaches to the task, explored performance bounds, and presented experiments predicting embedded clause structure using Winnow and maximum entropy. In this chapter, we summarise the results and achievements of this thesis, discuss shortcomings of the work, and suggest future work on the problem of clause recognition.

7.1 Results and achievements

There are several results from this thesis that stand to be pointed out. The first contribution is the exploration of performance bounds including a new symbolic baseline. Secondly, main and subordinate clauses are distinguished. Finally, Winnow and maximum entropy models are created and compared to the other learners employed for the CoNLL-2001 shared task. In the rest of this section we will discuss these achievements in some detail

7.1.1 Performance bounds

The new symbolic baseline reported in Chapter 3 better contextualises machine learning approaches to clause recognition among a common division in the current state of NLP. A new symbolic baseline was created that serves to better contextualise the machine learning results on the task among the field of computational linguistics as a whole.

This also brings up the idea of ‘supervision’ in learning. We characterised symbolic approaches earlier as consisting of knowledge explicitly encoded by the developer. However, the machine learning approaches inherit a good deal of knowledge in the form of the features designed for the task.

The new baseline performance is slightly better than the connectionist system (Hammerton 2001) in identifying clause starts and is significantly better than the Déjean’s (2001) symbolic learner at identifying clause ends (see Table 6.4). Apart from that, the rest of the systems are still above baseline in all phases.

It is unfortunate that we cannot compare the results of the previous symbolic approaches to the machine learning approaches. As we mentioned in Chapter 2, the test data as well as preparation and resources incorporated differ among all previous work. We suspect that the the machine learning methods would perform at least on a par with the symbolic approaches applied to the same task definition with the same resources.

7.1.2 Winnow and maximum entropy

The application of Winnow and maximum entropy have reinforced their suitability for natural language tasks, which are often characterised by complex feature spaces. Both methods are robust in high-dimensional feature spaces and avoid making unwarranted assumptions common to other learning approaches. The Sparse Network of Winnows software achieved final results that perform on a level with most of the CoNLL-2001 systems while the maximum entropy results are second only to those of Carreras & Màrquez (2001*b*).

This work also suplements other work in reinforcing the importance of feature extraction. Sang (2002) describes experiments with a memory-based learner and Carreras & Màrquez’s (2001*b*) feature set. Using these linguistically-motivated features that capture aspects of the data relevant to recognising clauses, improvements were achieved with the memory-based learner. This enforces the rather intuitive notion (exhibited as well in our results) that performance on a given task is affected both by the learner and the features chosen to represent the task. This form can be seen as symbolic methods being used to inform machine learning tasks. Indeed, it is this sort of explicit encoding of knowledge that makes it impossible to call any machine learning task truly unsupervised.

In this application to the task of recognising clauses, maximum entropy performed better than Winnow. This is due to a couple of important characteristics of the two model classes. First, maximum entropy is better at modelling relationships between features. Second, Winnow is more susceptible to noise in the data introduced through human error in annotation and mistakes from the POS and chunk taggers.

7.1.3 Distinguishing main and subordinate

We also proposed to distinguish main and subordinate clauses in our final system. We did this using simple heuristics for identifying main clauses among the clause segmentation predicted in phase 3. However, separating the main clause performance from the subordinate clause performance helps in identifying where the difficulty lies in recognising clause structure.

Low *recall* relative to *precision* in the subordinate clause scores for the machine learning methods indicate under-recognition problems. Low *precision* relative to *recall* in the main clause scores indicate over-recogniiton problems. Also, much higher scores in recognising main clauses reflect the simple definition of main adopted for this work and the relative difficulty of identifying embedded clauses.

7.2 Problems and weaknesses

This section discusses several weaknesses of the work reported in this thesis. We consider weaknesses in defining the task and in the data. Namely, we will address the lack of qualitative analysis, the simple definition of the distinction between main and subordinate mentioned above, noise from earlier stages in preparing the data for the task, and the size of the corpus.

7.2.1 Task definition

One weakness of the shared task is that complexity of algorithms are not discussed. In a sense, this does not bear directly on the engineering focus of the task; the concentration is on scores and not implementation. We have gone a short way toward considering the relative complexity of the algorithms in talking about symbolic and machine learning methods in Chapter 2. However, this work still suffers some lack of qualitative evaluation.

A weakness alluded to throughout this thesis in the various sections on main and subordinate clauses (1.2, 3.1.2, and 6.2.4) is the fact that our definition of main clauses (only top-level simple and coordinated clauses) is perhaps too simple. A more interesting distinction may have been between independent and dependent clauses as defined in Chapter 1. This alternative definition is also the basis a novel approach to clause segmentation suggested in the Section 7.3.1.

7.2.2 Data issues

Another weakness lies in the stacked nature of the approach to the task. Scores approaching the upper bounds (let alone perfect scores) are not possible due to noise from the automatic POS and chunk tagging stages performed in preparation for the clausing task. These two phases of imperfect predictors employed in preparing the data for the task each introduce noise. Brill (1994) reports accuracy scores around 97% for his POS tagger and Sang et al. (2000) reports *precision*, *recall*, and $F_{\beta=1}$ scores of 94.04, 91.00, and 92.50. So, the annotations we start with are imperfect.

Furthermore, the gold standard is based on human annotations of the Penn Treebank. While these annotations have for the most part been checked and at least double checked, inter-annotator disagreement is quoted at between 1% and 3% for POS tags (Marcus et al. 1993). Though we couldn't find numbers, Treebank parses are likely to be considerably noisier than this.

Finally, the corpus we're using is fairly small. In a recent study, Banko & Brill (2001) showed that the performance of memory-based, Winnow, perceptron, and naive Bayes learners were still increasing at 1,000,000,000 words of annotated material in a confusion set disambiguation task. However, using the same data as the CoNLL-2001 shared task allows the results to be compared directly.

7.3 Future work

We mentioned several tasks left to previous work in Chapters 3 and 6: improvement of symbolic approach, weighted voting, and expanding the prediction of multiple starts from a binary to a multi-label task. Here, we will talk a bit more about some other possible extensions: other features that may have been incorporated, feature selection methods, and a new approach to the task of clause recognition.

7.3.1 Other features

First, it might be interesting to perform experiments with more features derived from the corpus or obtained from outside resources. Though they would most likely offer only small improvement, the following are a few possibilities.

In a study on chunking, Osborne (2000) used part-of-word features including the first two letters and the last four letters of the current word. The features helped the system cope with sparse statistics by using orthography to approximate morphology. Although this may not improve our scores as it gives information similar to POS tags, further experiments could be performed on such part-of-word features. Another possibility is phrasal head information. Sang (2001) replaces chunk labels with phrasal head information in his CoNLL-2001 clause recognition system giving a bit of an improvement in recognising clause ends.

More interesting, perhaps, but also adding to the overall complexity of the system, would be to incorporate outside resources. For ideas, we can look to the symbolic systems described in Chater 2. For example, Leffa (1998) uses verb subcategorisation information. Though we have the advantage of chunk information, including valence information for verbs may help the learning algorithms cope with the errors introduced in the chunking stage.

7.3.2 Feature selection

Although Winnow and Maximum Entropy are robust to irrelevant features, the optimisation experiments presented in Chapter 6 illustrate that both could benefit from an feature selection method. Feature selection refers to any approach to finding the items from the feature space that are most valuable to the classification task at hand. Sang (2002), for instance, start with zero features and employ a forward sequential selection process through the clause recognition feature space of Carreras & Màrquez (2001*b*). This provides an $F_{\beta=1}$ score improvement for predicting clause starts in the development set from 90.52 to 91.82. We might also try feature combination methods such as principal components analysis (Jolliffe 1986) or multiple discriminants analysis (Duda & Hart 1973).

7.3.3 New approach

Finally, though different measures and different resources are used in earlier work on non-embedded clauses (Ejerhed 1988, Ejerhed 1996), it seems to be the case that concentrating on basic clauses simplifies the task. If this is indeed the case, a better approach may be to begin by predicting an end-to-end segmenting where subordinate clauses are not yet attached to their parent clause. Then, we could use an attachment disambiguation similar to that used for PP attachment (e.g., Hindle & Rooth 1993, Ratnaparkhi & Roukos 1994, Collins & Brooks 1995) where we would suppose that attachment depends on, for instance, the subordinating conjunction, the verb head of the clauses that are possible attachment sites.

Appendix A

Electronic Resources

What follows is a brief description of the code found at the link above and, where relevant, the parameters used.[1]

A.1 Data Preparation

- `makedata` – This script automates the sub tasks of data conversion. It calls the `chunklink_2-2-2000_for_conll.pl` script followed by the `cl2001.pl` script and, finally, combines the resulting clause tags with the Brill (1994) POS tags and the Sang (2000) chunk tags from the CoNLL-2001 shared task.
- `chunklink_2-2-2000_for_conll.pl` (Bucholz 2000) – This script creates the data used to infer main and subordinate clauses.
  - Parameters: `-B BeginEndCombined -N -f -h -H -t`
    - `-B BeginEndCombined` specifies that the chunk labels should specify begin, inside, end, and outside.
    - `-N` specifies that the word number should be suppressed in the output.
    - `-f` specifies that the function should be suppressed in the output.
    - `-h` specifies that the head word should be suppressed in the output.

[1] Links to all the scripts and other software used to complete this work can be found at <http://www.ling.ed.ac.uk/ benjamin/project/index.shtml>.

    * `-H` specifies that the head number should be suppressed in the output.
    * `-t` specifies that the trace information should be suppressed in the output.
  - Available from <http://ilk.kub.nl/ sabine/chunklink/README.html>.

- `cl2001.pl` – This script infers the main and subordinate clause labels from the output of the `chunklink_2-2-2000_for_conll.pl` script.
  - Parameters: `-m`
    * `-m` specifies that the only the phase 3 embedded clause tags should distinguish main and subordinate clauses.

- `pt2fdg_begin-end.pl` – This script converts data from the task format to a format compatible with the fex feature extractor.

- `fex` – This compiled C++ code creates integer features for `SNoW`.
  - Available from <http://l2r.cs.uiuc.edu/ cogcomp/software/fexreg.html>.
  - Parameters: `-p`
    * `-p` that the data column-based format from which `fex` is capable of creating conjunctions, disjunctions, collocations, and sparse collocations of column items. See (Cumby & Yih 2002) for details.

A.2 Evaluation

- `conlleval1, conlleval2, conlleval3` – These scripts are used to compute *accuracy*, *precision*, *recall*, and $F_{\beta=1}$ scores for all results reported.
  - Available from <http://lcg-www.uia.ac.be/conll2001/clauses/bin/>.

A.3 Voting

- `combinefiles.pl` – This script combines the output labels from all of the CoNLL-2001 systems into one file in the format required for the `vote.pl` script.

- `vote.pl` – This script was used to perform a majority vote over the output of the CoNLL-2001 systems.
  - Parameters: `-o`
    - `-o` specifies that the results should be displayed
  - Available from <http://lcg-www.uia.ac.be/ erikt/npcombi/>.
- `balance.pl` – This script balances the embedded clause structure from the voted output.

A.4 Symbolic

- `MS_Cleffa.pl` – This script implements the sybolic clause recogntion algorithm. It can is used separately for all three phases. Each phase expects the predicted labels from the previous phases to be present in the input file.

A.5 Classifiers

- `runExSEF_snow.pl` and `runExSEF_maxent.pl` – These scripts provide a parameterised testing environment. They create the data, train, and test SNoW and maxent depending on the parameters specified.
- `snow` – This compiled C++ code provides training and testing of Winnow classifiers.
  - Available from <http://l2r.cs.uiuc.edu/ danr/snow.html>.
- `snow_pred2text.pl` – This script interprets the SNoW predictions, converting them to the appropriate format for evaluation.
- `openNLP maxent` – This open source java code provides training and testing of maximum entropy classifiers.
  - Available from <http://maxent.sourceforge.net/>.

- `maxent_pred2text.pl` – This script interprets the openNLP maxent predictions, converting them to the appropriate format for evaluation.

Appendix B

Data Preparation

This appendix describes the data used for the machine learning approaches. Section 1 describes the corpus used; Section 2 the formatting. And finally, section 3 explains how the data was created including the information used to distinguish between main and subordinate clauses in the corpus.

Integral to the choice of an encoding for the data is how we cast the problem as a classification task. Chapters 1 and 2 describe how clause recognition can be cast as a tagging problem; here we go into the details of the format of the task. In the following, we will discuss the corpus, and the two scripts used to obtain the data for the work reported here.

B.1 Corpus

The corpus is what is sometimes called expository text (e.g., Leffa 1998). It consists of several sections of newswire from the Wall Street Journal portion of the Penn Treebank (Marcus et al. 1993).[1]

The sections used are based on the CoNLL-2001 shared task, which in turn are based on earlier shared task. Ultimately, this can be traced back to early work on base noun phrase recognition by Ramshaw & Marcus (1995). Sections and word counts are summarised in Table B.1 for the training, development, and test sets. The data

[1]Information about the Penn Treebank is also available online at <http://www.cis.upenn.edu/ treebank/home.html>.

```
( (S
    (S
      (NP-SBJ
        (NP (PRP It) )
        (SBAR (-NONE- *EXP*-1) ))
      (VP (VBD was) (RB n’t)
        (ADJP-PRD (JJ clear) )
        (SBAR-1
          (WHADVP-2 (WRB how) )
          (S
            (NP-SBJ (DT the) (NN ownership) )
            (VP (MD would)
              (VP (VB stack)
                (PRT (RP up) )
                (PP-LOC (IN under)
                  (NP (DT the) (JJ new) (NN plan) ))
                (ADVP-MNR (-NONE- *T*-2) )))))))
    (, ,) (CC but)
    (S
      (NP-SBJ (NNS employees) )
      (VP (MD would)
        (VP (VB keep)
          (NP
            (QP (JJR more) (IN than) (CD 50) )
            (NN %) ))))
    (. .) ))
```

Figure B.1: Example sentence from Penn Treebank

	# words	WSJ section(s)
Train	211727	15-18
Development	47377	20
Test	40039	21

Table B.1: Word count and WSJ sections for training, development, and test data

sets are fairly small, but using them allows the results to be compared to CoNLL-2001 results as well as making the optimising cycle of design, implementation, and evaluation relatively quick.

The format of the data before any conversion to the format used for the clause recognition task is the combined format of the Penn Treebank. Figure B.1 contains an example sentence in this format. The following two sections will show the same example in the intermediate and final converted formats.

We should also mention that word and chunk tags used in the task are not those from the Treebank. Rather, they are assigned by the Brill (1994) part of speech tagger and Sang's (2000) CoNLL-2000 shared task chunking system. This assures that performance rates are estimates for texts where no annotation exists.

B.2 Encoding

The first step in conversion to the task format is running Bucholz's (2000) *chunklink.pl* script. This is run with the following command line arguments for the training set.

```
./bin/chunklink_2-2-2000_for_conll.pl \
      -B BeginEndCombined -N -f -h -H -t \
      ../treebank/combined/wsj/15/wsj_15??.mrg \
      ../treebank/combined/wsj/16/wsj_16??.mrg \
      ../treebank/combined/wsj/17/wsj_17??.mrg \
      ../treebank/combined/wsj/18/wsj_18??.mrg
```

Refer to the appendix on electronic resources for an explanation of the arguments. Further detail can be found in the associated documentation (Bucholz 2000).

The above example creates output as in Table B.2. The first two columns are not used. The next three contain the chunk label, POS tag, and word for token. The last column is a representation of the Treebank parse and is used to infer main and subordinate clauses as explained in the next section.

		CHUNK	POS	WORD	TAG CHAIN
0000	1	C-NP	PRP	It	B-S/B-S/C-NP/C-NP
0000	1	B-VP	VBD	was	I-S/I-S/B-VP
0000	1	O	RB	n’t	I-S/I-S/I-VP
0000	1	C-ADJP	JJ	clear	I-S/I-S/I-VP/C-ADJP
0000	1	C-ADVP	WRB	how	I-S/I-S/I-VP/B-SBAR/C-WHADVP
0000	1	B-NP	DT	the	I-S/I-S/I-VP/I-SBAR/B-S/B-NP
0000	1	E-NP	NN	ownership	I-S/I-S/I-VP/I-SBAR/I-S/E-NP
0000	1	B-VP	MD	would	I-S/I-S/I-VP/I-SBAR/I-S/B-VP
0000	1	E-VP	VB	stack	I-S/I-S/I-VP/I-SBAR/I-S/I-VP
0000	1	C-PRT	RP	up	I-S/I-S/I-VP/I-SBAR/I-S/I-VP/C-PRT
0000	1	C-PP	IN	under	I-S/I-S/I-VP/I-SBAR/I-S/I-VP/B-PP
0000	1	B-NP	DT	the	I-S/I-S/I-VP/I-SBAR/I-S/I-VP/I-PP/B-NP
0000	1	I-NP	JJ	new	I-S/I-S/I-VP/I-SBAR/I-S/I-VP/I-PP/I-NP
0000	1	E-NP	NN	plan	I-S/E-S/E-VP/E-SBAR/E-S/E-VP/E-PP/E-NP
0000	1	O	COMMA	COMMA	I-S
0000	1	O	CC	but	I-S
0000	1	C-NP	NNS	employees	I-S/B-S/C-NP
0000	1	B-VP	MD	would	I-S/I-S/B-VP
0000	1	E-VP	VB	keep	I-S/I-S/I-VP
0000	1	B-NP	JJR	more	I-S/I-S/I-VP/B-NP
0000	1	I-NP	IN	than	I-S/I-S/I-VP/I-NP
0000	1	I-NP	CD	50	I-S/I-S/I-VP/I-NP
0000	1	E-NP	NN	%	I-S/E-S/E-VP/E-NP
0000	1	O	.	.	E-S

Table B.2: Example after chunklink script is run

WORD	POS	CHUNK	START	END	FULL
It	PRP	C-NP	S	X	(M(M*
was	VBD	B-VP	X	X	*
n't	RB	O	X	X	*
clear	JJ	C-ADJP	X	X	*
how	WRB	C-ADVP	S	X	(S*
the	DT	B-NP	S	X	(S*
ownership	NN	E-NP	X	X	*
would	MD	B-VP	X	X	*
stack	VB	E-VP	X	X	*
up	RP	C-PRT	X	X	*
under	IN	C-PP	X	X	*
the	DT	B-NP	X	X	*
new	JJ	I-NP	X	X	*
plan	NN	E-NP	X	E	*S)S)M)
COMMA	COMMA	O	X	X	*
but	CC	O	X	X	*
employees	NNS	C-NP	S	X	(M*
would	MD	B-VP	X	X	*
keep	VB	E-VP	X	X	*
more	JJR	B-NP	X	X	*
than	IN	I-NP	X	X	*
50	CD	I-NP	X	X	*
%	NN	E-NP	X	E	*M)
.	.	O	X	E	*M)

Table B.3: Example after clauselink script is run

B.3 Main and subordinate

The final stage of conversion uses the flattened form of the tree structure in the output from the `chunklink.pl` script to infer main and subordinate labels for the third phase of the embedded clause recognition task.

An an example of the output from this phase can be found in Table B.3. The first through third columns contain words, POS tags, and chunk labels respectively. The next three columns contain labels corresponding to the three phases of the task.

The fourth column contains clause start labels corresponding to the first phase of the clause recognition task. The fifth column contains clause end labels corresponding to the second phase. For both, a single start or end label is assigned for any number of embedded starts and ends at a given point. The sixth column corresponds to the third and final phase and contains the embedded main and subordinate clause labels.

The final conversion script (`clauselink.pl`) looks at the tree structure output from the `chunklink.pl` script to determine main and subordinate clauses. All clauses at the top level of a sentence are considered to be main clauses as are any clauses coordinated at the top level. All other clauses are assigned subordinate labels. After the data scripts were created, they were checked and double checked on random samples of 500 sentences.

Appendix C

Feature Extraction

Feature extraction is a very important part of machine learning techniques, especially in the realm of natural language processing. The types of features chosen and the way they are encoded can have a large effect on the performance of the the predictors used. Furthermore, they interact with the predictors on an individual level as is illustrated in Chapter 6 by the difference between the optimised feature sets of SNoW and maximum entropy.

Two approaches to feature extraction were used. They were tested on the first phase using SNoW. The first method employed FeX, a feature extractor written specifically for applying SNoW to NLP tasks. The second approach is a Perl script designed specifically for our the framework we use for the task of clause recognition. This section describes the two approaches and the reasons for choosing the latter.

Before getting into the details, we should clarify the difference between feature extraction and feature selection. Both Winnow and Maximum entropy can be construed as feature selection methods. They are robust in the face of many features. We can train them on any features that may be relevant and they will learn higher weights for relevant features and lower weights for irrelevant features. When we talk about feature extraction here, we are referring to the process of building examples from the input data. It is these examples that the learning architectures use to create representations of the relationships between features and targets and predict a target given the representation along with the example.

FeX feature extraction

Feats	**0**	**1**	**2**	**3**	**4**
w	0.00	65.69	70.36	71.30	70.14
p	0.00	64.78	65.38	63.35	64.14
c	0.00	65.06	61.54	63.87	63.05
wp	0.00	73.96	76.03	74.54	75.37
wc	0.00	70.45	71.79	73.81	74.12
pc	0.00	73.67	74.53	73.05	74.16
wpc	0.00	76.94	77.96	78.17	76.84

Perl script feature extraction

Feats	**0**	**1**	**2**	**3**	**4**
w	67.82	80.63	78.89	78.29	77.00
p	36.65	74.03	54.70	61.28	60.12
c	18.66	74.84	75.36	74.23	75.67
wp	66.06	84.23	82.42	80.84	78.46
wc	70.51	88.63	86.80	85.71	84.20
pc	59.94	86.75	84.85	84.57	82.08
wpc	68.01	89.43	88.05	88.33	86.59

Table C.1: Results of window size experiment using FeX and a Perl script created for this task.

C.1 FeX

The FeX feature extractor was written for using SNoW on simple NLP tasks. It requires the data to be in a specific format which meant another conversion beyond that described in the appendix on data preparation. Table C.1 (key found in Table 6.1 of Chapter 6) gives FeX and SNoW $F_{\beta=1}$ scores for the window size and basic features experiment reported in the experiments chapter.

A specific limitation of FeX is the fact that it requires a specific input format especially as it only allows three individual kinds of features to be used (although it was suggested that FeX could be extended to do this (Cumby 2002)) where we needed to use word, POS tag, chunk label, and already predicted labels in the later phases.

C.2 Perl script

The second choice for feature extraction was to write a script to do the job. Table C.1 (key found in Table 6.1 of Chapter 6) gives the scores for a feature extraction script written specifically for the task on the same experiment. The numbers speak for themselves here; performance is far better with the Perl script tailored to the task. The results in this thesis are based on the feature extraction using the Perl script.

Bibliography

Abney, S. (1991), ‘Parsing by chunks’, *Principle-based Parsing: Computation and Pscholinguistics* pp. 257–278.

Abney, S. (1996), Partial parsing via finite-state cascades, *in* ‘Proceedings of the 8th European Summer School in Logic, Language and Information (ESSLLI’96)’, Prague, Czech Republic, pp. 8–15.

Åström, M. (1998), A probabilistic tagger for swedish using the SUC tagset, Technical report, Department of Linguistics, University of Umeå.

Baldridge, J., Morton, T. & Bierner, G. (2001), *openNLP maxent software documentation*.
URL: *<http://maxent.sourceforge.net/index.html>*

Banko, M. & Brill, E. (2001), Scaling to very large corpora for natural language disambiguation, *in* ‘Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL 2001)’, Toulouse, France, pp. 26–33.

Berger, A., Della Pietra, S. & Della Pietra, V. (1996), ‘A maximum entropy approach to natural language processing’, *Computational Linguistics* **22**(1), 39–71.

Bies, A., Ferguson, M., Katz, K. & MacIntyre, R. (1995), Bracketing guidelines for Treebank II style Penn Treebank Project, Technical report, University of Pennsylvania.

Brants, T. (1999), Cascaded markon models, *in* ‘Proceedings of the 9th Conference of the European Chapter of the Associations for Computational Linguistics (EACL’99)’, Bergen, Norway, pp. 118–125.

Brill, E. (1994), Some advances in rule-based part of speech tagging, *in* ‘Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)’, Seattle, Washington, USA, pp. 722–727.

Bucholz, S. (2000), *README for Perl script chunklink.pl*.
URL: *<http://ilk.kub.nl/∼sabine/chunklink/README.html>*

Carlson, A., Cumby, C., Rosen, J. & Roth, D. (1999), *SNoW user guide*.
URL: *<http://l2r.cs.uiuc.edu/∼danr/snow.html>*

Carreras, X. & Màrquez, L. (2001*a*), Boosting trees for anti-spam email filtering, *in* 'Proceedings of the 4th Conference on Recent Advances in Natural Language Processing (RANLP-2001)', Tzigov Chark, Bulgaria, pp. 58–64.

Carreras, X. & Màrquez, L. (2001*b*), Boosting trees for clause splitting, *in* 'Proceedings of the 5th Conference on Natural Language Learning (CoNLL-2001) at the 39th Annual Meeting of the Association for Computational Linguistics (ACL 2001)', Toulouse, France, pp. 73–75.

Charniak, E. (2000), A maximum-entropy-inspired parser, *in* 'Proceedings of the 1st Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL-2000)', Seattle, Washington, USA, pp. 132–139.

Church, K. (1988), A stochastic parts program and noun phrase parser for unrestricted text, *in* 'Proceedings of the 2nd Conference on Applied Natural Language Processing (ANLP-88)', Austin, Texas, pp. 136–143.

Collins, M. (1999), Head-Driven Statistical Models for Natural Language Processing, PhD thesis, University of Pennsylvania.

Collins, M. & Brooks, J. (1995), Prepositional attachment through a backed-off model, *in* 'Proceedings of the 3rd Workshop on Very Large Corpora', Somerset, New Jersey, USA, pp. 27–38.

Cumby, C. (2002), Private communication.

Cumby, C. & Yih, W. (2002), *FEX user guide*.
URL: *<http://l2r.cs.uiuc.edu/ cogcomp/FexManual.ps>*

Daelemans, W. (1995), 'Memory-based lexical acquistion and processing', *Springer lecture notes in artificial intelligence: Machine translation and the lexicon* **898**, 85–98.

Daelemans, W. (2000), TiMBL: Tilburg memory-based learner - version 3.0, reference guide, Technical report, ILK, Computational Linguistics, Tilburgh University.
URL: *<http://ilk.kub.nl/downloads/pub/papers/ilk.0201.ps>*

Dagan, I., Karov, Y. & Roth, D. (1997), Mistake-driven learning in text categorization, *in* 'Proceedings of 2nd Conference on Empirical Methods in Natural Language Processing (ENMLP-97)', Providence, Rhode Island, USA, pp. 55–63.

Darroch, J. & Ratcliff, D. (1972), 'Generalized iterative scaling for log-linear models', *Annals of Mathematical Statistics* **43**, 1470–1480.

Déjean, H. (2000), Theory refinement and natural language learning, *in* ‘Proceedings of the 18th international conference on computational linguistics (COLING 2000)’, Saarbrüken, Germany.

Déjean, H. (2001), Using ALLiS for clausing, *in* ‘Proceedings of the 5th Conference on Natural Language Learning (CoNLL-2001) at the 39th Annual Meeting of the Association for Computational Linguistics (ACL 2001)’, Toulouse, France, pp. 64–66.

Dietterich, T. (2000), Ensemble methods in machine learning, *in* ‘Proceedings of the 1st International Workshop on Multiple Classifier Systems (MCS 2000)’, Cagliari, Italy, pp. 1–15.

Duda, R. & Hart, P. (1973), *Pattern classification and scene analysis*, Wiley-Interscience.

Ejerhed, E. (1987), Finding noun phrases and clauses in unrestricted text: On the use of stochastic and finitary methods in text analysis, Technical report, AT&T Bell Labs.

Ejerhed, E. (1988), Finding clauses in unrestricted text by finitary and stochastic methods, *in* ‘Proceedings of the 2nd Conference on Applied Natural Language Processing (ANLP-88)’, Austin, Texas, USA, pp. 219–227.

Ejerhed, E. (1996), Finite state segmentation of discourse into clauses, *in* ‘Proceedings of the 12th European Conference on Artificial Intelligence (ECAI-96) Workshop on Extended Finite State Models of Language’, Budapest, Hungary.

Ejerhed, E. & Church, K. (1983), Finite state parsing, *in* ‘Papers from the 7th Scandinavian Conference of Linguistics’, Helsinki, Finland, pp. 410–432.

Francis, W. & Kucera, H. (1964), *Brown Corpus Manual of Information*.
URL: <*http://khnt.hit.uib.no/icame/manuals/brown*>

Freund, Y. & Schapire, R. (1995), A decision-theoretic generalization of on-line learning and an application to boosting, *in* ‘Proceedings of the 2nd European Conference on Computational Learning Theory (EuroCOLT ’95)’, Berlin, Germany, pp. 23–37.

Golding, A. & Roth, D. (1999), ‘A winnow-based approach to context-sensitive spelling correction’, *Machine Learning* **34**(1-3), 107–130.

Hammerton, J. (2001), Clause identification with long short-term memory, *in* ‘Proceedings of the 5th Conference on Natural Language Learning (CoNLL-2001) at the 39th Annual Meeting of the Association for Computational Linguistics (ACL 2001)’, Toulouse, France, pp. 61–63.

Hindle, D. & Rooth, M. (1993), ‘Structural ambiguity and lexical relations’, *Computational Linguistics* **19**(1), 103–120.

Hochreiter, S. & Schmidhuber, J. (1997), ‘Long short-term memory’, *Neural Computation* **9**(8), 1735–1780.

Jarvella, R. (1971), ‘Syntactic processing of connected speech’, *Journal of Verbal Learning & Verbal Behavior* **10**, 409–416.

Jarvella, R. & Pisoni, D. (1970), ‘The relation between syntactic and perceptual units in speech processing’, *Journal of the Acoustical Society of America* **48**, 84.

Jolliffe, I. (1986), *Principal Component Analysis*, Springer Verlag.

Leffa, V. (1998), Clause processing in complex sentences, *in* ‘Proceedings of the 1st International Conference on Language Resources and Evaluation’, Granada, Spain, pp. 937–943.

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

Loos, E., Anderson, S., Day, D., Jordan, P. & Wingate, J., eds (1999), *Glossary of linguistic terms*, SIL International.
URL: <*http://www.sil.org/linguistics/GlossaryOfLinguisticTerms/Index.htm*>

Manning, C. & Schütze, H. (1999), *Foundations of Statistical Natural Language Processing*, Cambridge: The MIT Press.

Marcus, M., Santorini, B. & Marcinkiewicz, M. (1993), ‘Building a large annotated corpus of english: The Penn Treebank’, *Computational Linguistics* **19**, 313–330.

McLauchlan, M. (2001), Maximum entropy models and prepositional phrase attachment, Master’s thesis, University of Edinburgh.

Molina, A. & Pla, F. (2001), Clause detection using hmm, *in* ‘Proceedings of the 5th Conference on Natural Language Learning (CoNLL-2001) at the 39th Annual Meeting of the Association for Computational Linguistics (ACL 2001)’, Toulouse, France, pp. 70–72.

Motta, E. & Lu, W. (2000), IBROW project ist-1999-19005: Appendix 1: Classification task specification, Technical report, The Open University.

Oliver, J. & Wallace, C. (1991), Inferring decision graphs, *in* ‘Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91) Workshop on Evaluating and Changing Representation in Machine Learning’, Sydney, Australia.

Orăsan, C. (2000), A hybrid method for clause splitting in unrestricted english texts, *in* 'Proceedings of the International Conference on Artificial and Computational Intelligence for Decision, Control and Automation in Engineering and Industrial Applications (ACIDCA'2000)', Monastir, Tunisia, pp. 129–134.

Osborne, M. (2000), Shallow parsing as part-of-speech tagging, *in* 'Proceedings of the 4th Conference on Computational Language Learning (CoNLL-2000) and the 2nd Learning Language in Logic Workshop (LLL-2000)', Lisbon, Portugal.

Papageorgiou, H. (1997), Clause recognition in the framework of alignment, *in* 'Proceedings of the 2nd Conference on Recent Advances in Natural Language Processing (RANLP-97)', Tzigov Chark, Bulgaria, pp. 417–425.

Papageorgiou, H. (2002), Private communication.

Patrick, J. & Goyal, I. (2001), Boosted decision graphs for NLP learning tasks, *in* 'Proceedings of the 5th Conference on Natural Language Learning (CoNLL-2001) at the 39th Annual Meeting of the Association for Computational Linguistics (ACL 2001)', Toulouse, France, pp. 58–60.

Quirk, R., Greenbaum, S., Leech, G. & Svartik, J. (1985), *A comprehensive grammar of the English language*, London: Longman.

Rabiner, L. & Juang, B. (1986), 'An introduction to hidden markov models', *IEEE Acoustics, Speech and Signal Processing (ASSP) Magazine* **3**(1), 4–16.

Ramshaw, L. & Marcus, M. (1995), Text chunking using transformation-based learning, *in* 'Proceedings of the 3rd Workshop on Very Large Corpora at the 33rd Annual Meeting of the Association for Computational Linguistics (ACL 1995)', Somerset, New Jersey, USA, pp. 82–94.

Ratnaparkhi, A. (1996), A maximum entropy model for part-of-speech tagging, *in* 'Proceedings of the 1st Conference on Empirical Methods in Natural Language Processing (EMNLP-96)', Philadelphia, Pennsylvania, USA, pp. 133–142.

Ratnaparkhi, A. (1997), A simple introduction to maximum entropy models for natural language processing, Technical report, Institute for Research in Cognitive Science, University of Pennsylvania.
URL: *ftp://ftp.cis.upenn.edu/pub/ircs/tr/97-08.ps.Z*

Ratnaparkhi, A. (1998), Maximum entropy models for natural language ambiguity resolution, PhD thesis, University of Pennsylvania.

Ratnaparkhi, A. (1999), 'Learning to parse natural language with maximum entropy models', *Machine Learning* **34**(1-3), 151–175.

Ratnaparkhi, A. & Roukos, S. (1994), 'A maximum entropy model for prepositional phrase attachment'.

Salomaa, A. (1981), *Jewels of formal language theory*, Pitman Publishing Ltd, London.

Sang, E. (2000), Text chunking by system combination, *in* 'Proceedings of the 4th Conference on Computational Language Learning (CoNLL-2000) and the 2nd Learning Language in Logic Workshop (LLL-2000)', Lisbon, Portugal, pp. 151–153.

Sang, E. (2001), Memory-based clause identification, *in* 'Proceedings of the 5th Conference on Natural Language Learning (CoNLL-2001) at the 39th Annual Meeting of the Association for Computational Linguistics (ACL 2001)', Toulouse, France, pp. 67–69.

Sang, E. (2002), 'Memory-based shallow parsing', *Journal of Machine Learning Research* **2**, 559–594.

Sang, E., Daelemans, W., Déjean, H., Koeling, R., Krymolowski, Y., Punyakanok, V. & Roth, D. (2000), Applying system combination to base noun phrase identification, *in* 'Proceedings of the 18th International Conference on Computational Linguistics (COLING 2000)', Saarbrücken, Germany, pp. 857–863.

Sang, E. & Déjean, H. (2001), Introduction to the CoNLL-2001 shared task: Clause identification, *in* 'Proceedings of the 5th Conference on Natural Language Learning (CoNLL-2001) at the 39th Annual Meeting of the Association for Computational Linguistics (ACL 2001)', Toulouse, France, pp. 52–57.

Schapire, R. E. (1999), Theoretical views of boosting and applications, *in* 'Proceedings of the 10th International Conference on Algorithmic Learning Theory ALT'99', Tokyo, Japan, pp. 13–25.

Schapire, R. E. & Singer, Y. (1998), Improved boosting algorithms using confidence-rated predictions, *in* 'Proceedings of the 11th Annual Conference on Computational Learning Theory (COLT 1998)', Madison, Wisconsin, USA, pp. 80–91.

Strangert, E., Ejerhed, E. & Huber, D. (1993), Clause structure and prosodic segmentation, *in* 'Papers from the 7th Swedish Phonetics Conference (FONETIK-03)', Uppsala, Sweden, pp. 81–84.

van Halteren, H., Zavrel, J. & Daelemans, W. (1998), Improving data driven wordclass tagging by system combination, *in* 'Proceedings of the joint 17th International Conference on Computational Linguistics (COLING '98) and 36th Annual Meeting of the Association for Computational Linguistics, (ACL 1998)', Montréal, Québec, Canada, pp. 491–497.

Wallace, C. & Patrick, J. (1993), ‘Coding decision trees’, *Machine Learning* **11**, 7–22.

Zhang, T., Demerau, F. & Johnson, D. (2001), Text chunking using regularized winnow, *in* ‘Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL-2001)’, Toulouse, France, pp. 539–546.