Part-of-Speech Tagging Using Progol

James Cussens

Oxford University Computing Laboratory
Wolfson Building, Parks Road
Oxford OX1 3QD, UK
Tel: +44 1865 283520 Fax: +44 1865 273839
james.cussens@comlab.ox.ac.uk

Abstract. A system for 'tagging' words with their part-of-speech (POS) tags is constructed. The system has two components: a lexicon containing the set of possible POS tags for a given word, and rules which use a word's context to *eliminate* possible tags for a word. The Inductive Logic Programming (ILP) system Progol is used to induce these rules in the form of definite clauses. The final theory contained 885 clauses. For background knowledge, Progol uses a simple grammar, where the tags are terminals and predicates such as `nounp` (noun phrase) are non-terminals. Progol was altered to allow the caching of information about clauses generated during the induction process which greatly increased efficiency. The system achieved a per-word accuracy of 96.4% on known words drawn from sentences without quotation marks. This is on a par with other tagging systems induced from the same data [5, 2, 4] which all have accuracies in the range 96–97%. The per-sentence accuracy was 49.5%.

1 Introduction

In part-of-speech disambiguation or 'tagging', the aim is to classify word tokens according to their part of speech. This is non-trivial because many words are ambiguous. For example, in the sentence "Jane sat on the table", "table" is a noun, yet in the sentence "James decided to table the motion", "table" is a verb. Once disambiguation has been carried out, the word can be 'tagged' with its correct part of speech (POS) tag. As Daelemans et al [5] note: "Automatic tagging is useful for a number of applications: as a preprocessing stage to parsing, in information retrieval, in text to speech systems, in corpus linguistics, etc."

A variety of approaches, both inductive and non-inductive, have been used to attack the tagging problem. In [7] a Hidden Markov model (HMM) was used to create the well-known Xerox tagger [4] which correctly tagged 96% of word instances on a subset of the Brown corpus.

In [2], Brill induces a rule-based tagger based on transformations. Words are initially tagged with their most common tag according to the training corpus. If a word is not in the corpus, a tag is guessed according to a number of features of the word such as capitalisation. Brill then considers 11 rule templates dictating when a word should have its tag changed. For example, the first rule template

is: "Change tag a to tag b when the preceding (following) word is tagged z." Lexicalized rules are included such as: "Change tag a to tag b when one of the two preceding (following) words is w." With this approach 267 transformation rules were induced, and an accuracy of 96.5% was achieved on a subset of the Wall Street Journal

A notable non-inductive tagger is EngCG, the English Constraint Grammar, which is a "rule-based framework for morphological disambiguation and shallow syntactic parsing, where the rules are hand-coded by a linguistic expert" [8]. EngCG achieved a per-word accuracy of 99.7%, albeit with each word receiving on average 1.04–1.09 tags, so some ambiguity remains [9].

Daelemans et al [5] take a memory-based approach to the tagging problem, implemented via *IGTree*, a tree-based approach for efficient indexing and searching of large case base. Together with a method for handling unknown words, Daelemans et al achieved an accuracy of 96.4% overall, and 96.7% on known words. (Known words are those that have occurred at least once in the training data.) Training was performed on 2 million words from the Wall Street Journal, and testing on the remaining one million.

Tapanainen and Voutilainen [9] *combined* the inductively constructed Xerox tagger and the hand-crafted EngCG to produce a system which achieved an accuracy of 98.5% with no remaining ambiguities on test data of 26,711 words of newspaper text taken from the Bank of English corpus from the University of Birmingham. The motivation for such a combination was that when EngCG performs a disambiguation it is almost always correct. However, EngCG does not disambiguate all words, so these remaining ambiguities are passed to the Xerox tagger, which always makes some decision.

There are two dichotomies in the work just described: that between inductive and non-inductive approaches and that between statistical and rule-based approaches. EngCG, the only non-inductive approach described, does have the best accuracy figures, but was "developed and debugged over several years". All the inductive approaches have accuracies around 96%, but these systems can rapidly be produced from pre-tagged training corpora.

A useful discussion of the relative merits of statistical and rule-based approaches can be found in [1]. Those rule-based taggers tend to be more compact, faster and more comprehensible. A HMM tagger has two notable advantages. Firstly, it can be trained using *untagged* text, although better performance is possible using tagged text. Secondly, HMM taggers can perform tagging on a sentence-by-sentence basis, using the entire sequence of words in the sentence to find the most likely sequence of tags. Such a system should be able to cope when the correct tag can only be decided with reference to words some distance away: the problem of *long-distance dependency*. Rule-based approaches which consider a small window of words on either side of the word to be tagged (the *focus word*), will be in trouble in such cases.

Given the complementary strengths of the various approaches just outlined, some sort of hybrid system, such as that in [9], is attractive. Here, we report on the application of the Inductive Logic Programming (ILP) system Progol [6]

to the tagging problem. Since the brunt of the work involves inducing rules, in the form of definite clauses, for *eliminating* possible tags, our work falls mainly into the inductive+rule-based category. However, we employ a simple statistical method for those ambiguities left unresolved by the elimination rules and employ hand-crafted (and fairly complex) background knowledge. This means that the complete tagging system is hybrid in nature.

The plan of this report is as follows. In Section 2, we describe the two components of our tagging system. Section 3 describes the core of the work: the use of Progol to induce tag elimination rules. In Section 4, we describe how Progol was altered to handle large data sets in a reasonable amount of time. Results and conclusions are in Sections 5 and 6.

2 Combining lexical-statistical information and tag elimination rules

2.1 Creating a lexicon

The data used in this work was the 3 million word tagged Wall Street Journal corpus[1]. The first tagged sentence from the corpus is: Pierre_NP Vinken_NP ,_, 61_CD years_NNS old_JJ ,_, will_MD join_VB the_DT board_NN as_IN a_DT nonexecutive_JJ director_NN Nov._NP 29_CD ._. So that tags could be represented as Prolog constants, without the need for messy quoting, the original tags were translated into strings of lower-case letters. The tag set used in the corpus, together with the Prolog translations and frequencies, is given in Table 1.

After splitting the corpus into 2/3 train and 1/3 test set, the first step taken was to create a *lexicon* which records the frequency with which words in the training set are given different tags. An excerpt from the lexicon is given in Fig 1. Despite its simplicity, the lexicon is the most powerful single element of our tagging system, since it gives us (according to the training data) the set of possible tags for a given word: the word's *ambiguity class*. In many cases there is, according to the training data, only one possible tag for a word.

The entry for "New" in Fig 1 shows the problem of rare tags: out of 2574 occurrences "New" is tagged as `nps` (plural proper noun) only twice. From such statistics one might infer that these two occurrences are due to noise in the training set, Also, intuitively, "New" is not plural. However, the tagging here is correct, both occurrences are part of the noun phrase "New Yorkers", where "New" is correctly tagged as a plural noun. On the other hand the `nn` tagging *is* incorrect.

Noise-handling is a particularly important issue when one is chasing accuracies in excess of 96%. In this work, we simply deleted a tag from a word's ambiguity class if it occurred less than 5% of the time. This crude approach is a weakness, since, for example, the correct `nps` and even the `jj` tagging for "New" were deleted. Future work will concentrate on improved noise-handling, but there are some advantages to this approach. Firstly, most incorrect tags are

[1] ACL Data Collection Initiative CD-ROM 1, September 1991

Table 1. Penn Treebank Part of Speech Tags.

Tag	Prolog	Meaning	Frequency
#	pnd	£	538
$	dlr	"$"	21242
"	lqt	" " " or " ' "	23789
"	rqt	" " " or " ' "	23427
(	lpn	"("	4647
)	rpn	")"	4552
,	cma	","	163227
.	stp	"."	127640
:	cln	";" or ":"	8806
CC	cc	Coordinating Conjunction	67833
CD	cd	Cardinal number	112565
DT	dt	Determiner	263125
EX	ex	Existential "there"	2826
FW	fw	Foreign word	705
IN	in	Preposition or Subordinating conjunction	321687
JJ	jj	Adjective	197118
JJR	jjr	Adjective, comparative	10574
JJS	jjs	Adjective, superlative	6104
LS	ls	List item marker	109
MD	md	Modal	31516
NN	nn	Noun, singular or mass	424605
NNS	nns	Noun, plural	192883
NP	np	Proper noun, singular	304396
NPS	nps	Proper noun, plural	8778
PDT	pdt	Predeterminer	1075
POS	pos	Possessive ending	28090
PP	pp	Personal pronoun	56094
PP$	ppz	Possessive pronoun	27378
RB	rb	Adverb	99788
RBR	rbr	Adverb, comparative	5482
RBS	rbs	Adverb, superlative	1436
RP	rp	Particle	5391
SYM	sym	Symbol	91
TO	to	"To"	72202
UH	uh	Interjection	328
VB	vb	Verb, base form	86049
VBD	vbd	Verb, past tense	96504
VBG	vbg	Verb, gerund or present participle	48325
VBN	vbn	Verb, past participle	65663
VBP	vbp	Verb, non 3rd-person, singular present	40099
VBZ	vbz	Verb, 3rd person singular present	70394
WDT	wdt	Wh-determiner	14300
WP	wp	Wh-pronoun	7771
WP$	wpz	Possessive wh-pronoun	587
WRB	wrb	Wh-adverb	6829

```
New nps 2 jj 61 nn 1 np 2501
...
bears vbz 16 nns 10
beast nn 1
beasties nns 1
beasts nns 3
beat vbd 11 vbp 4 jj 1 nn 5 vb 30
beat-up jj 1
```

Fig. 1. An excerpt from the lexicon (before noise elimination).

eliminated, secondly, those correct tags that are deleted are necessarily rare so their deletion will not cause a substantial decrease in accuracy.

2.2 The tagging system

Our basic approach combines features of [8] and [9]. As in [8], we learn rules for *eliminating* possible tags, but our possible tags are taken from the lexicon, not the EngCG. As in [9], we use a statistical approach if and only if the rule-based approach does not resolve all ambiguities, except here, rather than use the Xerox tagger, we simply choose the most frequent possible tag for a word according to the lexicon.

The central issue of inducing tag elimination rules is detailed in Section 3. Here, by way of motivation, we show, via an example, how the completed system operates. The system works on a sentence-by-sentence basis. Suppose the sentence to be tagged was:

> A House-Senate conference last week accepted the provision with no discussion of the potential cost to the government.

The lexicon then replaces each word with its ambiguity class. Note that for this to be possible the word must appear in the lexicon; the word must be one of the 66,024 'known' words in the lexicon. Consequently, our system is restricted to sentences containing only known words. Future work will use morphological analysis of unknown words to overcome this restriction.

Each tag in a non-singleton ambiguity class is paired with the relative frequency with which it appeared in the lexicon after noise pre-processing. This gives us:

```
[[[dt,0.947],[np,0.053]], [[jj,0.268],[np,0.732]], nn, jj, nn,
[[vbn,0.523],[vbd,0.477]], dt, nn, in, [[dt,0.873],[rb,0.127]],
nn, in, dt, [[jj,0.784],[nn,0.216]], [[nn,0.790],[vb,0.210]],
to, dt, nn, stp]
```

We then use induced clauses to eliminate tags. The first elimination is that of the `vbn` (past participle) tagging for "accepted". This is due to the following induced clause:

```
rmv(A,B,vbn) :- dt(B,C), nounp1(A,D).
```

Here `A` is the left context of "accepted" and `B` is the right context. `dt(B,C)` holds if `B` starts with a `dt` (determiner) and the remainder of `B` is `C`. This rule eliminates the possibility that "accepted" is a past participle because it is sandwiched between the noun phrase "week" and the determiner "the". Note that vbn is the more likely tag for "accepted" without contextual information, so here the elimination rules have (correctly) overridden the lexical-statistical information.

Using the following rule, the `rb` (adverb) tagging for "no" is eliminated because "no" is followed by the noun "discussion":

```
rmv(A,B,rb) :- noun(B,C).
```

The vb (infinitive) tag for "cost" has also been eliminated, because it is followed by "to" and preceded by "potential" which may be a noun or an adjective. If "potential" is a noun and hence, in our grammar, a noun phrase, it allows the following rule to fire:

```
rmv(A,B,vb) :- to(B,C), nounp1(A,D).
```

If instead "potential" is an adjective, then the following rule fires:

```
rmv(A,B,vb) :- adjp1(A,C).
```

Either way the `vb` tagging for "cost" is eliminated.

Since no more tag elimination rules can fire, the lexical statistics take over and eliminate the very unlikely `np` tagging for "A". Returning to the rules, this allows us to knock out the np tagging for "House-Senate", since the following rule can now fire.

```
rmv(A,B,np) :- dt(A,C), cnoun(B,D), nounp(D,E), sverb(E,F)
```

The final ambiguity caused by the word "potential" is then resolved using the lexical statistical information, giving us the (correct) final sequence of tags:

```
[dt, jj, nn, jj, nn, vbd, dt, nn, in, dt, nn, in, dt, jj, nn,
to, dt, nn, stp]
```

3 Eliminating tags by partial parsing

3.1 Generating examples for Progol

It was decided to learn rules for eliminations on a tag by tag basis. In other words, Progol learnt rules for eliminating `cc, cd, dt`, etc from separate data sets, one data set for each tag. Positive and negative examples were generated as follows. The corpus was scanned a sentence at a time. For each word which has an ambiguous tag according to the lexicon, one negative example, and one or more positive examples of tag elimination were created. The negative example records which tag it would not be correct to eliminate, i.e. the correct tag. The

positive examples record which tags might have been correct, since they are members of the word's ambiguity class, but are, in fact, incorrect in the current context. The example contains the tag to be removed and its entire left and right context. The left context is reversed so that the start of the tag list representing the left context contains those tags nearest to the focus word. For example, the second sentence from the corpus produces the positive and negative examples given in Table 2.

Table 2. Creating examples from a sentence.

Mr. Vinken is chairman of Elsevier N.V., the **Dutch** publishing group.
`:- rmv([dt,cma,np,np,in,nn,vbz,np,np],[vbg,nn,stp],np).`
`rmv([dt,cma,np,np,in,nn,vbz,np,np],[vbg,nn,stp],jj).`
`rmv([dt,cma,np,np,in,nn,vbz,np,np],[vbg,nn,stp],nps).`
Mr. Vinken is chairman of Elsevier N.V., the Dutch **publishing** group.
`:- rmv([np,dt,cma,np,np,in,nn,vbz,np,np],[nn,stp],vbg).`
`rmv([np,dt,cma,np,np,in,nn,vbz,np,np],[nn,stp],nn).`

Since it was possible to create vast numbers of examples (at least for the more important tags), we could afford to be quite restrictive about which examples to include in training. Firstly, only sequences which (i) began with a word with an initial capital letter, (ii) ended in a full stop and (iii) *contained no quotes* were used. This had the effect of filtering out some noisy sentences but meant that certain sorts of sentence were never used to generate training examples. The no-quote restriction is a considerable one, and further work will have to drop it. It was used in this preliminary work since it made it easier to identify where sentences began and ended and because the lack of embedded speech in sentences simplified their structure.

For many tags, this produced many thousands of examples (positive and negative). In the future, it may be possible to develop Progol to cope with such vast example sets. Although Progol was altered to speed up learning on large data sets, it was still thought necessary here to sample from the full training set for each tag. So in this work we used only the first 6000 examples found for each tag. Details of the various training sets so produced can be found in [3].

3.2 Using a partial grammar as background knowledge

A subset of the background knowledge is given in Fig 2. The background predicates fall into three sets: 'tag' predicates, such as `cc/2`, forward parsing predicates and backward parsing predicates. The background knowledge defines a grammar where the 43 POS tags are terminals and there are 22 nonterminals defining such grammatical constructs as noun phrase (`nounp`) and adjectival phrase (`adjp`). There is also a number of 'utility predicates' such as `adjp_rest` which are used for efficiency.

```
cc([cc|S],S).    cd([cd|S],S).    dt([dt|S],S).
ex([ex|S],S).    fw([fw|S],S).    in([in|S],S).

cncy([pnd|S],S) :- !.  cncy([dlr|S],S).

cds([cd|X],S2) :- cd(S1,X), (X=S2 ; cds(X,S2)).

noun([nn|S],S) :- !.  noun([np|S],S) :- !.
noun([nns|S],S) :- !. noun([nps|S],S).

snp(L1,L2) :- noun(L1,X), (X=L2 ; snp(X,L2)).

dtz([dt|S],S) :- !.  dtz([ppz|S],S).

nounp(L1,L2) :- noun(L1,X), !, (X=L2 ; snp(X,L2)). % "Pierre Vinken"
nounp([pp|S],S) :- !. % "us, him"
nounp(L1,L2) :- dtz(L1,L3), !, (snp(L3,L2) ; adjp(L3,L4), snp(L4,L2)).
nounp(L1,L2) :- cncy(L1,L3), !, cds(L3,L2). % "$ 20 billion "
nounp([cd|Y],L2) :- !, (X=Y ; cds(Y,X)),  (X=L2 ; snp(X,L2)). % "20, 20%"
nounp(L1,L2) :- adjp(L1,L3), snp(L3,L2). % "green men"

nounp1([pp|S],S) :- !. % "us, him"
nounp1([cd|S1],S2) :- !, (S3=S1 ; cds(S1,S3)), (S3=S2 ; cncy(S3,S2)).
nounp1(S1,S2) :- snp(S1,S3), nounp1_rest(S3,S2).

nounp1_rest(S,S).
nounp1_rest(S1,S2) :- dtz(S1,S2), !.
nounp1_rest([cd|S1],S2) :- !, (S1=S2 ; cds(S1,S2)).
nounp1_rest(S1,S2) :- adjp1(S1,S3), (S3=S2 ; dtz(S3,S2)).
```

Fig. 2. A subset of the background knowledge.

Because the terminals of the grammar are the tags which do not represent all the information there is about the underlying words, the grammar so defined is very 'sloppy'. For example, the definition of simple verb phrase `svp` allows that the tag sequence `[vbz,vbg]` constitutes a simple verb phrase. This means that in any case where a gerund or past participle follows a 3rd person singular present verb, then this is counted as a simple verb phrase. *The singular present verb is not constrained to be an auxiliary verb.* The tag set used here does not distinguish auxiliary verbs. For example, the phrase "is wandering" will be translated, via the lexicon, into `[vbz,vbg]` (neither word is ambiguously tagged in the lexicon, although "wandering" can be a noun). It will then be correctly recognised as a simple verb phrase. However, the fragment "wanders being" which is nonsense, will also be marked as a simple verb phrase.

The grammar defined in the background knowledge is clearly over-general, but we are not using it to determine whether sentences are grammatical, or to

generate new sentences. We have chosen the definitions of the 22 background predicates in the hope that these will be relevant to the task of tag elimination. The clauses produced show this to be the case.

As well as the terminals and the 'forward parsing' predicates we have defined 4 predicates for parsing backwards. We parse the left context of the focus word backwards on the grounds that the most useful part of the context is that nearest to the focus word. Consequently, we work from right-to-left through the left context, i.e. backwards. This can best be done by reversing the the context to the left of the focus word, and then parsing taking the reversal into account. Consequently, a simple verb phrase in the left context can be a sequence `[vbg,vbz]` ("walking is").

Although all background predicates were defined for all Progol runs, not all were allowed to appear in bodies of induced tag elimination clauses. The basic set of 38 background predicates that were allowed to appear in clause bodies can be found in [3]. The intuition for doing this was that for many of the tags, the missing background predicates were irrelevant. For example, knowing whether a particular tag was an `nn, nns, np` or `nps` was thought not to be that helpful in eliminating the possibility that a certain word could be a `dt`—it is enough to know that it is a noun or indeed a noun phrase. When learning elimination rules for certain tags: `jj, jjr, jjs, nn, nns, np, nps, vb, vbg, vbn, vbp, vbz` it was judged that extra background predicates should be allowed to appear in clause bodies. The details are given in [3].

4 Speeding up Progol with caching

The combination of four factors meant that inducing a theory for tag elimination involved a heavy computational burden. These are: the size of the example sets, the complexity and size of the background knowledge, the non-heuristic nature of Progol's search and the fact that positive examples covered by induced clauses were not removed from the training set. Given the scale of the induction problem a more heuristically based ILP approach has attractions. However the very high accuracy figures required motivate a complete search ILP algorithm, such as Progol. Although this can lead to long computation times (e.g. 19.8 hours for `rp`), computation still comprised a very small fraction of the total time committed to the problem. (The Prolog implementation of the Progol algorithm, called P-Progol, was used in all the work reported here. P-Progol is available, for academic research, via anonymous ftp from `ftp.comlab.ox.ac.uk` in the directory `pub/Packages/ILP`.)

In Progol, a 'bottom' clause is produced from a 'seed' positive example, which is the most specific clause which covers that example, given the declarative bias. Progol then carries out an *admissible* top-down search of the subsumption lattice of clauses which subsume the bottom clause (and are in the hypothesis language). Progol is *guaranteed* to find a clause which has maximal compression in this lattice. In this work the following search strategy was employed: once a positive example became covered by an induced clause it was removed from the

set of possible seeds. However, it was *not* removed from the example set. This ensured that the correct positive (P) and negative cover (N) of induced clauses is always available, allowing a training set estimate of the accuracy of a clause: $|P|/(|P|+|N|)$. This was required to exceed 0.95 for all induced clauses.

The computational problem was tackled in two ways. Firstly, search constraints were used: clauses were constrained to have at most 5 literals and cover at least 15 positive examples ($|P| > 15$). This last constraint plays a dual role. Firstly, it ensures that our crude and generally over-optimistic clause accuracy estimate is unlikely to be too inaccurate. But it is also particularly useful in Progol's top-down search; if a clause covers less than 15 positive examples, then so will all its refinements and so the search can be pruned at this clause. One final syntactic constraint pruned out clauses with redundant literals: for example, if a clause contains the literals `adjp(B,C), jjs(B,C)` then `adjp(B,C)` is redundant since all superlative adjectives are also adjectival phrases.

These constraints alone were insufficient to allow learning in a reasonable amount of time. To speed up learning further, we traded space for time by caching the positive and negative cover of clauses which Progol produced during its search. If a clause appears again in a subsequent search, we can avoid doing any theorem-proving by simply recovering its cover from the cache. This approach was extended by having a 'prune cache'. If a clause is such that, for any search in which it appears, the search can be pruned at that clause, then it is added to the prune cache. For each clause produced by Progol during its search, we then check to see if it is in the prune cache, and prune the search if it is. (Details of the caching method are in [3]. Caching is now implemented in the ftp-able version of P-Progol.)

Caching brought a large speed-up. A few comparative experiments have been done to measure this speed-up. In one experiment we induced the sub-theory for eliminating `rp`, with and without caching. With caching Progol took 19.8 hours, which is the longest time for any theory. Without caching, Progol took 312 hours and then stalled, for reasons unknown. This means we have a speed up factor in excess of 15.75. However, even for small data sets the improvements are significant, for the `rbs` theory (1784 examples) Progol without caching took 17,403 seconds. With caching, the same theory was induced in 1,620 seconds, a speed-up factor of 10.74. Using a very recent version of Progol with additional efficiency improvements the `rbs` theory was induced in only 641 seconds, which is over 27 times quicker than the original non-caching version of Progol.

5 Results

5.1 Computation times

Details of the computation times for inducing tag elimination theories for each tag are given in Table 3. The n/a entries for `jj`, `nn`, `nns`, `np` and `nps` are because the theories for these tags were produced over a number of runs, where many of the same clauses were found in the different runs. The "Searches" column records

the number of bottom clauses and hence subsumption lattices searches undertaken. The "Mean" column gives the mean time taken over all these searches. All times are in seconds.

Table 3. Theory size and search computation times for each tag.

Tag	Pos	Neg	Searches	Mean	Search time	Total time (est.)	Clauses
cc	1192	338	539	10.61	5717	6417	28
cd	780	2312	780	26.47	20650	21664	0
dt	3825	2175	385	53.83	20724	21225	42
ex	182	660	101	2.85	288	419	2
fw	331	162	294	7.31	2148	2530	2
in	2292	3708	1186	20.75	24605	26146	22
jj	3216	2784	n/a	n/a	n/a	n/a	47
jjr	2624	3376	1101	49.06	54012	55444	39
jjs	879	1068	426	7.98	3401	3955	19
md	209	102	35	0.19	7	52	4
nn	3025	2975	n/a	n/a	n/a	n/a	61
nns	2266	3734	n/a	n/a	n/a	n/a	48
np	2647	3353	n/a	n/a	n/a	n/a	31
nps	3773	2227	n/a	n/a	n/a	n/a	47
pdt	3231	470	21	58.41	1227	1254	14
pos	448	448	21	110.05	2311	2338	6
pp	511	168	32	2.32	74	116	18
ppz	152	511	84	0.41	35	144	2
rb	4182	1818	1305	19.63	25622	27318	87
rbr	3850	2150	1727	21.08	36400	38645	45
rbs	1064	720	250	4.04	1009	1334	25
rp	4539	1461	1883	36.31	68371	71259	48
rqt	1043	0	1	4.65	5	6	1
stp	11	1	10	0.01	0	13	0
sym	11	18	10	0.01	0	13	0
uh	195	33	42	0.21	9	64	3
vb	3566	2434	180	44.09	7935	8169	35
vbd	3294	2706	545	37.35	20354	21062	20
vbg	3192	2808	1589	22.24	35335	37401	44
vbn	3087	2913	881	28.92	25482	26627	57
vbp	4328	1672	222	38.39	8522	8811	25
vbz	4925	1075	462	62.90	29061	29661	83
wdt	4433	1567	5	115.83	579	586	3
wp	10	5	10	0.01	0	13	0
Total	73313	51952	14127	n/a	n/a	n/a	908

5.2 Structure of the induced theory

Because of the search strategy used, it is possible to have redundant clauses in the induced sub-theories. We call a clause redundant if all the positive examples it covers are covered by other clauses in the theory. Removing such clauses make the theory simpler and may reduce the negative cover, rendering it more likely to be accurate.[2] It does not specialise the theory according to the training data. Redundant clauses were removed by a companion program to P-Progol called T-Reduce. T-Reduce reduced the final theory from the original 908 clauses to 885 clauses. The two big reductions were for the `nn` subtheory (from 61 to 54 clauses) and `nps` (from 47 to 41 clauses). This was probably because these theories were produced over several runs leading to large overlap over clauses. (T-Reduce is available from Ashwin Srinivasan at the same address as the current author.)

One of the features of ILP is that the first-order representation often allows an easily intelligible induced theory. Examining the theory induced here, an extract of which is shown in Fig 3, we see that each individual clause *is* easy to understand, although not all are intuitive. So, although, it seems intuitive that no word preceding "to" should be a co-ordinating conjunction (second rule) rules such as

```
rmv(A,B,in) :-  cma(A,C), cds(C,D), nounp1(D,E), cln(E,F).
```

are not so obvious.

The number of 2,3,4 and 5 literal clauses is 103, 251, 276 and 254 respectively. The number of clauses of depth 0,1,2,3 and 4 is 1, 242, 322, 228 and 92 respectively. The induction of a large number of depth 4 and 5 clauses is significant. This shows that our approach performs disambiguation based on long-distance dependencies, which was one of the motivations for using ILP. The most commonly appearing background predicates are: `nounp` (15.4%), `nounp1` (12.1%), `in` (10.4%), `noun` (8.0%) `dt` (6.5%), `cma` (5.2%), `vp` (4.0%), `vp1` (3.1%), `adjp` (3.1%) and `advp` (2.9%). The importance of complex background predicates such as `nounp` demonstrate the utility of background knowledge for the tagging problem.

Taking the theory as a whole, there is the intelligibility problem of the sheer number of clauses. In future work, we will induce both tag elimination *and* tag identification rules which may lead to a more compact theory. However, it should be noted that overriding goal in tagging is to produce a tagging system of very high accuracy, and given the complexity of the problem it is probably over-optimistic to hope that accurate theories will also be simple ones. (The full (unreduced) theory and extensive analysis of the results can be found in [3]. The reduced theory is available from the author on request.)

5.3 Accuracy on test data

The central results of this work are summarised in Tables 4-7. These show the results of testing the tagging system on an independent test set of 5000 sen-

[2] In fact, accuracy was only increased from 106718/110716 to 106719/110716!

```
rmv(A,B,cc) :- nounp(B,C), in(A,D), noun(D,E), in(E,F).
rmv(A,B,cc) :- to(B,C).

rmv(A,B,dt) :- in(B,C), noun(C,D), nounp(D,E), vp1(A,F).
rmv(A,B,dt) :- in(B,C), nounp(C,D), vp(D,E), vp1(A,F).
rmv(A,B,dt) :- noun(B,C), noun(C,D), vp1(A,E), advp(E,F).
rmv(A,B,dt) :- nounp(B,C), advp(C,D), vp(D,E), vp1(A,F).
rmv(A,B,dt) :- nounp(B,C), noun(C,D), pos(D,E), nounp(E,F).

rmv(A,B,in) :-  cma(A,C), cds(C,D), nounp1(D,E), cln(E,F).
rmv(A,B,in) :-  dt(A,C), in(C,D).

rmv(A,B,jj) :-  cc(A,C), prnoun(B,D).
rmv(A,B,jj) :-  cma(B,C), vp(C,D).
rmv(A,B,jj) :-  dt(A,C), in(B,D), dt(D,E).

rmv(A,B,nn) :-  advp(A,C), nounp1(C,D).
rmv(A,B,nn) :-  cc(A,C), dt(B,D).
rmv(A,B,nn) :-  cc(B,C), nounp(C,D), prnoun(A,E).

rmv(A,B,rp) :-  dt(B,C), nounp1(A,D).
rmv(A,B,rp) :-  in(A,C).
rmv(A,B,rp) :-  in(B,C), adjp(C,D), noun(D,E), nounp1(A,F).

rmv(A,B,vb) :-  cc(B,C), vp(C,D).
rmv(A,B,vb) :-  cma(A,C), nounp1(C,D), nounp1(D,E).
```

Fig. 3. Subset of theory of tag elimination.

tences consisting of 110,716 words. This preliminary system is only applicable to sentences starting with a capital letter, ending with a full stop, not containing quotes and which are composed entirely of words which have occurred at least once in the training set. Only sentences which met these restrictions were included in the test set.

As well as overall accuracy figures, we give accuracies on just the ambiguous words and also accuracies excluding lexical errors. A lexical error occurs when the correct tag for a word is not in the lexicon. This can arise either because of our crude noise-handling approach or simply because not all of the correct tags for a word were seen in the training data. We also compare tagging using the tag elimination theory with tagging just using the lexical statistics. The per-word difference between 94.1% and 96.4% given in Table 5 may seem insignificant. But when we compare on a sentence by sentence basis we see (in Table 7) that the rules increase accuracy from 30.4% to 49.5%.

Finally, in Table 8, we show the results of testing the tag elimination theory in isolation. As expected, the theory is seriously under-general: only 62% of tags that should be eliminated are eliminated. However, tags are very rarely

Table 4. Per-word tagging accuracy.

	Ambiguous				Overall			
Lexical errors?	Correct	Total	Lex-Errs	Acc	Correct	Total	Lex-Errs	Acc
Included	19495	22956	247	84.9%	106719	110716	783	96.4%
Excluded	19495	22709	0	85.8%	106719	109933	0	97.1%

Table 5. Per-word tagging accuracy with and without rules.

	Ambiguous				Overall			
Theory	Correct	Total	Lex-Errs	Acc	Correct	Total	Lex-Errs	Acc
Full	19495	22956	247	84.9%	106719	110716	783	96.4%
Empty	16937	22956	247	73.8%	104161	110716	783	94.1%

incorrectly eliminated. In this application we are *not* invoking the Closed World Assumption as is often done in ILP. If we can not prove that a tag should be eliminated, we do not assume it is the correct tag; instead we ask the lexicon to do the necessary disambiguation. This accounts for the large discrepancy between an accuracy of 77.4% for the rules in isolation and a system accuracy of 96.4%.

6 Conclusions and future work

The most important figure is that of overall 96.4% per-word tagging accuracy. Given our restrictions it is difficult to compare with other approaches, given the absence of many 'known' words results. Daelemans et al [5] achieve 96.7% on known words, but do not have our 'no quotes' restrictions. We conjecture that including sentences with quotes will not cause us too many problems, indeed will increase accuracy since quotes themselves are unambiguous and such sentences generally include the word "said" which is almost always a `vbd`. We conclude that the system produced here achieved a respectable result for a first application of Progol to the tagging problem, but that better results using an unrestricted system are required in the future.

Apart from the reasonable accuracy achieved, important features of this work include the combination of induced clauses and lexical statistics, the capturing of long-term dependencies, the construction of a reasonably comprehensible theory and the successful incorporation of caching.

Table 6. System tagging accuracy for sentences.

Lexical errors?	Correct	Total	Lex-Errs	Acc
Included	2475	5000	698	49.5%
Excluded	2475	4302	0	57.6%

Table 7. System tagging accuracy for sentences with and without rules.

Theory	Correct	Total	Lex-Errs	Acc
Full	2475	5000	698	49.5%
Empty	1520	5000	698	30.4%

Table 8. Contingency table for complete theory

	Actual		
Predicted↓	Pos	Neg	Total
Pos	3439	146	3585
Neg	2111	4304	6415
Total	5550	4450	10000

True positives = 62.0%
True negatives = 96.7%
Overall accuracy = 77.4%

Having said this, our approach could be improved in a number of ways. We finish by listing, in order of importance, necessary future work.

Unknown words At present, we have *no* method of dealing with words that have not occurred in the training set. This means that the present system is not yet a practical tagging system. There is nothing to stop us "bolting on" a sensible method of dealing with unknown words which takes an unknown word and returns a tag ambiguity class. Brill [2] and Daelemans et al [5] both have methods for dealing with unknown words; these techniques could be incorporated into future work.

Noise handling Noisy data is particularly problematic when one is aiming for accuracies of over 96%. [8] used two linguists to pore over the training data to weed out noise. Our crude method of noise-handling is the single biggest cause of inaccuracy. Future work should use a Bayesian approach to take into account the absolute frequency with which a tag appears for a particular word, rather than the simply the proportion (relative frequency) with which it appears.

Combining rules and lexical statistics Our system uses two sources of information to tag a word:

1. The frequencies with which the word has been given particular tags, irrespective of context.
2. The context of the word in terms of the tags of the other words in the sentence.

Essentially, we are trying to find $\arg\max_{tag} P(tag|word, context)$. At present, we use the lexicon to estimate $P(tag|word)$ and have a collection of rules, such that if a rule of the form $remove(tag) \leftarrow context$ is present, then we assume, sometimes incorrectly, that $P(tag|context) < 0.05$ and also that if this inequality holds then *tag* has a lower probability than any other tag. If no elimination rule covers the word then we simply use $P(tag|word)$ to estimate $P(tag|word, context)$. A better approach would be to *label* induced clauses with estimates of their accuracy and to allow (labelled) clauses with

much lower estimated accuracy to be included in the final theory, essentially using induced clauses as the structural component of a suitable probability distribution.

Including quotes The avoidance of sentences with quotes is a non-essential restriction and is primarily a matter of convenience. This should be dropped in future work.

Acknowledgements This work was supported by ESPRIT IV Long Term Research Project ILP II (No. 20237). The author would like to thank Walter Daelemans, Ashwin Srinivasan, David Page, Stephen Muggleton and two anonymous reviewers.

References

1. Steven Abney. Part-of-speech tagging and partial parsing. In Ken Church, Steve Young, and Gerrit Bloothooft, editors, *Corpus-Based Methods in Language and Speech*. Kluwer, Dordrecht, 1996.
2. Eric Brill. Some advances in transformation-based part of speech tagging. In *AAAI94*, 1994.
3. J. Cussens. Part-of-speech disambiguation using ILP. Technical Report PRG-TR-25-96, Oxford University Computing Laboratory, 1996.
4. Doug Cutting, Julian Kupiec, Jan Pedersen, and Penelope Sibun. A practical part-of-speech tagger. In *Third Conference on Applied Natural Linguistic Processing (ANLP-92)*, pages 133–140, 1992.
5. W. Daelemans, J. Zavrel, P. Berck, and S. Gillis. MBT: A memory-based part of speech tagger-generator. In *Proceedings of the Fourth Workshop on Very Large Corpora,*, pages 14–27, Copenhagen, 1996.
6. S. Muggleton. Inverse entailment and Progol. *New Generation Computing Journal*, 13:245–286, 1995.
7. Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–285, February 1989.
8. Christer Samuelsson, Pasi Tapanainen, and Atro Voutilainen. Inducing constraint grammars. In Laurent Miclet and Colin de la Higuera, editors, *Grammatical Inference: Learning Syntax from Sentences*, volume 1147 of *Lecture Notes in Artificial Intelligence*, pages 146–155. Springer, 1996.
9. Pasi Tapanainen and Atro Voutilainen. Tagging accurately – Don't guess if you know. In *Proc. ANLP94*, 1994.