7

A Hybrid Grammatical Tagger: CLAWS4

ROGER GARSIDE and NICHOLAS SMITH

In this chapter we discuss in detail how a piece of software can carry out automatically one important task in corpus annotation. The task is **part-of-speech (POS) tagging** (also called **word-class tagging**, or **grammatical tagging**); that is, assigning to each word in a text its correct part of speech in context. The result of this task, as a form of corpus annotation, was discussed in some detail in Chapter 2. It usually forms a basis for more sophisticated annotation, such as full syntactic parsing or semantic annotation, and it carries out the useful supplementary tasks of splitting up the text into individual words and sentences.

Most current part-of-speech taggers are **probabilistic** or **stochastic** (see, for example, Marshall 1983, Garside *et al.* 1987, Church 1988, DeRose 1991, Cutting *et al.* 1992, Merialdo 1994); that is, they choose a preferred tag for a word by calculating the most likely tag in the context of the word and its immediate neighbours. At the same time, non-probabilistic or **rule-based** taggers (which began with Greene and Rubin 1971) have been making something of a come-back, with the tagging systems discussed by Brill (1992) and Voutilainen (1995: 165–284). In practice it may be that a **hybrid** system, which combines both probabilistic and rule-based approaches, captures the best of both techniques.

Most serviceable taggers today attain an accuracy in the region of 95–98 per cent. However, what is meant by such a figure is open to variant interpretations. It is probably better to avoid drawing conclusions about the quality of tagging software from comparing crude accuracy rates until we know more about the quality of the linguistic distinctions which the tagger makes, and how consistent analysts have been in checking the accuracy of a tagger (see Chapter 17).[1]

One of the earliest probabilistic taggers was **CLAWS** (Constituent Likelihood Automatic Word-tagging System), developed by UCREL at the University of Lancaster (Marshall 1983, Garside *et al.* 1987). This chapter discusses the current incarnation of this piece of software, CLAWS4; this could now be considered to be a hybrid tagger, involving both probabilistic and rule-based elements. It has been designed so that it can be easily adapted to different types of text in different input formats.

7.1 Probabilistic and Rule-based Taggers

In natural language processing by computer up to the late 1970s, part-of-speech tagging was seen as a by-product (and not a very interesting one at that) of full syntactic parsing. However the **TAGGIT** program (Greene and Rubin 1971) introduced the idea of providing a text corpus annotated with part-of-speech information as a useful tool for linguistic research. This was the **Brown Corpus** of one million words of written American English, collected in the early 1960s. The tags assigned were from a set of some 77 tags (the Brown tagset). The basic idea in the TAGGIT program was to associate with each word a set of potential tags, and then use the context to choose the correct one. The mechanism for the initial assignment of tags to a word relied on a lexicon, a word-ending list, and a set of other rules for dealing with capitalized words, hyphenated words, etc.; as we will see, this general type of mechanism is used in CLAWS (and indeed in other probabilistic taggers). The contextual disambiguation was carried out in TAGGIT by means of context-frame rules. A context-frame rule was a rule, designed by a linguist based on observation of data, which specified some information about a potential tag in the context of up to three tags on either side – the rule could specify that the potential tag was the correct one in context, or that the potential tag was impossible in this context (so that one of the other potential tags must be the correct one). All the tags being used for contextual clues in a context frame rule had to be unambiguous, so the context frame rules had to be tried several times, in the hope that disambiguating a tag at some point in a sentence would allow a context frame rule now to be applied to disambiguate another tag in the sentence.

After this pioneering tagger using the rule-based paradigm, interest passed in the early 1980s to probabilistic taggers. The general idea is that, if we have a sequence of words, each with one or more potential tags, then we can choose the most likely sequence of tags by calculating the probability of all possible sequences of tags, and then choosing the sequence with the highest probability. Thus, if we have a sequence of words $w_1, w_2, \dots,$ w_n, the goal of tagging is to select the most likely sequence of tags $t_1, t_2, \dots,$ t_n associated with those words, and we assume that this is the correct sequence. The statistical model which has been most used in POS tagging is that of the **hidden Markov model** (HMM) (see Poritz 1988). We can

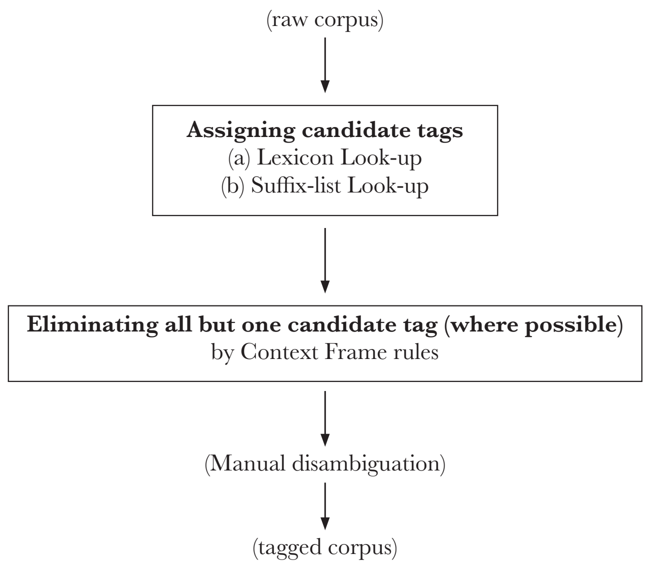

Figure 7.1 Diagram of the TAGGIT system processing the Brown Corpus

directly observe the sequence of words, but we can only estimate the sequence of tags, which is 'hidden' from the observer of the text; hence the term 'hidden Markov model' is appropriate. A HMM enables us to estimate the most likely sequence of tags, making use of observed frequencies of words and tags (in a **training corpus**).

The probability of a tag sequence is generally a function of:

- the probability that one tag follows another; for example, after a determiner tag an adjective tag or a noun tag is quite likely, but a verb base form tag is less likely. So in a sentence beginning *the run*..., the word *run* is more likely to be a noun than a verb base form.
- The probability of a word being assigned a particular tag from the list of all possible tags for the word; for example, the word *over* could be a common noun in certain restricted contexts (of cricket reports), but generally a preposition tag would be overwhelmingly the more likely one.

On page 105 we have the sequence of words $w_1, w_2, \ldots, w_n$ and a possible sequence of tags $t_1, t_2, \ldots, t_n$. The probability of this sequence of tags makes use of estimates of the tag transition probabilities shown as $p_{1,2}$, $p_{2,3}$, etc. in the diagram below. In the simplest case these estimates are derived from frequencies of tag pairs ('bigrams' in the training corpus). The second type of probability that enters into an HMM is that labelled q_1, q_2, etc.

$$\begin{tikzcd}
w_1 & w_2 & w_3 & w_4 & \cdots & w_{n-1} & w_n \\
t_1 \arrow[u, "q_1"'] \arrow[r, "p_{1,2}"'] & t_2 \arrow[u, "q_2"'] \arrow[r, "p_{2,3}"'] & t_3 \arrow[u, "q_3"'] \arrow[r, "p_{3,4}"'] & t_4 \arrow[u, "q_4"'] \arrow[r] & \cdots \arrow[u] \arrow[r] & t_{n-1} \arrow[u, "q_{n-1}"'] \arrow[r, "p_{n-1,n}"'] & t_n \arrow[u, "q_n"']
\end{tikzcd}$$

in the diagram. This is the probability that the word w_i will be associated with the tag t_i. Obviously, with a sufficiently large training corpus, it will be possible to estimate the q_i from the relative frequency with which any particular word and tag associate with each other.

This model is a *first-order* HMM, since the estimates used for the tag transition probabilities are derived from bigrams; that is, we have estimated the likelihood of a particular tag occurring given only the preceding tag. A second-order HMM would use tag transition estimates derived from trigrams; that is, we estimate the likelihood of a particular tag occurring given the preceding *two* tags. This is clearly a more refined estimate of the probability of one tag following another, but we have to calculate a much larger set of estimates and need a correspondingly larger training corpus.

It is clear that the HMM model descibed above is no more than a rather rough approximation to the problem we are trying to solve when we tag a corpus. For one thing, it treats the tag sequence as an abstraction from what the words are; it ignores, for example, the problem of how to deal with idiosyncratic word sequences or multiwords like *as well as*. Secondly, and more notably, it ignores any grammatical constraints on the word-class of a word, apart from constraints derived from its immediate neighbours. Nevertheless, in spite of its manifest theoretical limitations, the HMM approach to tagging is surprisingly successful, and various taggers of this general design result in a 95–97 per cent accuracy rate. The Xerox tagger, which is of this general design, is discussed further in Section 10.2.

After Greene and Rubin's TAGGIT tagger, the next rule-based tagger to attract serious interest was that of Brill (1992, 1994), and this illustrates how the rule-based approach contrasts with the probabilistic approach. Like a probabilistic tagger, Brill's tagger requires a training corpus and, using this, the tagger works by 'automatically recognizing and remedying its weaknesses, thereby incrementally improving its performance' (Brill 1992). The first step is to apply the most likely tag for each word (i.e. 'most likely' without reference to any left or right context, what is called the 'unigram' probability) – for this step, and this step alone, quantitative information is needed. The training corpus, tagged in this way, has a relatively high accuracy (in the region of 90 per cent), and the task then is to improve this result by iteratively applying a set of patching rules of the following form (excerpted from Brill 1992):

change tag *a* to tag *b* when:
1. The preceding (following) word is tagged *z*.
2. The word two before (after) is tagged *z*.
3. One of the two preceding (following) words is tagged *z*.
4. One of the three preceding (following) words is tagged *z*.

These rules, as can be seen, make use of information available in the immediate context of the 'target word', although longer-range rules are also possible. The tagger looks at the application of each rule of this type to the training corpus, and computes the number of errors remedied by its application and the number of new errors introduced. For example, the most successful of Brill's (1992) patches was one which changed the tag 'infinitive marker' to 'preposition' when an article followed (e.g. in *to the*, the word *to* could scarcely be an infinitive marker). The procedure of learning the patches is iterative, and after each run the most successful patch is added to the list of patches. The 'patching rules' are ordered in terms of the net improvement they achieve (i.e. the size of the difference between tags corrected and tags wrongly corrected). If one wishes to tag a new corpus, the patches (after the basic tagging has been done) are applied in order of success-rate; but if a patch changes tag *a* to tag *b*, it applies only if there is some instance in the training corpus of the word in question having the tag *b*.

Brill's rule-based tagger has produced results comparable to HMM taggers, and therefore has challenged the orthodoxy (which had been growing up in the early 1990s) that statistical methods outperform rule-based methods. At the same time, statistical taggers do not seem to be making significant progress towards the goal of 100 per cent success, and this may be because they are lacking in the kinds of grammatical knowledge about language which linguists take for granted. There may be a plateau which probabilistic taggers have reached, and there may be limits to how far one can go without a richer kind of linguistic knowledge.

This may be borne out by the fact that the one notable improvement on the 3–5 per cent error rate claimed up to now has been a grammar-based system, **ENGCG** (the English Constraint Grammar of the Helsinki team of Karlsson and Voutilainen – see Section 3.3.5). Certainly the grammar- or rule-based approach can be taken much further than was previously thought.

In fact many systems are not quite so 'pure' as the above discussion implies, and already there is some combining of strategies. Nearly all probabilistic taggers have sets of heuristic rules or guessers dealing with unknown words, while some rule-based systems use a limited amount of frequency information, as we have seen with Brill's (1992, 1994) system. A systematic attempt to integrate the two approaches is described in Tapanainen and Voutilainen (1994). Their experiment involved tagging a text with both a rule-based tagger (ENGCG – see Section 3.3.5) and a probabilistic tagger (the Xerox tagger – see Section 10.2) and aligning the outputs from the two programs. Where ENGCG succeeds in fully disambiguating a word its analysis is preferred to that of the Xerox tagger; where ambiguity remains in the former, it is resolved by accepting the disambiguation of the latter.

7.2 The CLAWS Tagger

The CLAWS tagger, discussed in detail in the remainder of this chapter, could be considered to be a hybrid tagger, involving both probabilistic and rule-based elements, even in its earliest form (CLAWS1 – Marshall 1983, Garside *et al.* 1987). The probabilistic element was an approximation to a HMM tagger. In one respect it was less that a HMM tagger; instead of using probabilities of word-tag association (the probabilities in the diagram on p. 105), it relied on human judgement of frequency applied to tags for ambiguous words in the lexicon. A three-point scale was used – common, rare (less than 10 per cent of the word occurrences were expected to receive this tag), very rare (less than 1 per cent of the word occurrences were expected to receive this tag). The reason for adopting this expedient was that the training corpus was judged not to be large enough to provide reliable word-tag association statistics.

In another respect, CLAWS1 was already more than a HMM tagger: it contained an embryonic rule-based component, the so-called 'idiomlist' (see Blackwell 1987) which enabled it to carry out exceptional taggings; for example, to tag **multiwords** (see Section 2.2) such as *as for* or *in order that* as single tokens, or to identify common tag sequence constraints which departed from what could be expected using the HMM mechanism (for example *dining room* tagged as NOUN–NOUN rather than ADJECTIVE–NOUN).

The 'idiomlist' component has been enormously expanded in later versions of CLAWS, so that the term idiomlist (never very satisfactory, as it suggests that it searches only for patterns which are linguistic idioms) should be replaced by the term **rule-based component**. This component in current versions of CLAWS not only identifies exceptional sequences (for example multiwords, foreign expressions and complex names of various kinds), it also carries out a significant role in disambiguation, sometimes preempting, sometimes correcting the probabilistic processing of major categories such as infinitivals and past participles, which are liable to cause trouble for a tagger relying on probabilistic resources alone.

The first version of the CLAWS tagging system (which was subsequently named CLAWS1) was developed at Lancaster over the period 1980–83. It was developed as part of a project to assign part-of-speech information to the LOB (Lancaster-Oslo/Bergen) Corpus, a one-million word corpus of British English designed to match the Brown Corpus in size, scope and structure. The tagset used, which became known as the CLAWS1 tagset, was a development of the Brown tagset, using about 135 tags.

The second version of CLAWS (CLAWS2) was developed over the period 1983–86. One special feature of the LOB Corpus was that certain features such as sentence breaks were explicitly marked in the text. CLAWS2 was developed as a tagger which could be run over general text, without explicit mark-up of this kind. The tagset used, the CLAWS2 tagset (with 132 tags), was a revision and refinement of the CLAWS1 tagset, based on experience with using this original tagset.

The development of CLAWS4 began in 1988,[2] and a number of versions of this software have been produced; the latest version (late 1996) being version 17. One development has been the separation of the CLAWS software from the tagset used. As mentioned above, earlier versions of CLAWS were closely designed round a specific tagset. However CLAWS4 was developed to tag the one hundred million word British National Corpus (BNC), and for this two tagsets were to be used:

- a detailed tagset (C7) of 146 tags for a two million word **sampler** corpus, and
- a less refined tagset (C5) of 61 tags for the rest of the corpus.

The opportunity was taken to decouple the program code from the tagset, which is now read in as part of the resources required for a particular tagging task.

7.3 Input Issues

The first version of CLAWS was designed specifically for the LOB Corpus, with its special notation for representing sentence breaks, changes of typeface, special characters, etc. (see Johansson *et al.* 1978). CLAWS2 was designed to cope with text which used normal orthographic conventions, but it soon became clear that CLAWS would have to be able to cope with representation of special characters such as accented letters, and of the structure of a text – for instance, it is fairly common for certain parts of a document not to be text for tagging, and these have to be marked so that they can be ignored.

When CLAWS4 came to be redesigned so that it could be used for the British National Corpus project, it was decided to move over to using SGML (Standard Generalized Mark-up Language: see Section 2.4) to represent all features of a text. This is the normal default assumption in current use of the CLAWS system, although it is possible also to process files in plain ASCII. In a number of recent UCREL projects, a pre-processing program running before CLAWS translates a text from a different format into the standard set of SGML tags and entities used by CLAWS. For example in one project there are a number of special purpose programs to take files of text from a variety of sources (including the World Wide Web) and with a number of different formats to represent emphasis, quotation marks, special characters, etc., and to convert them into the standard format required by CLAWS.

One of the resources read by CLAWS at the beginning of a run is a list of all valid SGML tags and entities together with the action to be taken on recognizing them. It is of course possible to supplement the standard set of tags and/or entities for processing a particular text. If CLAWS encounters a tag or entity not in these tables, it displays an error message. The default action for a valid SGML tag is for CLAWS simply to ignore it, copying it from the input to the output with an associated **null** part-of-speech mark. This would be the normal case, for example, for tags marking text structural divisions (chapters, paragraphs, etc.) or denoting typeshifts (bold, italic, etc.). Alternative actions which can be specified are:

- to ignore all text bracketed by a particular pair of tags – this was not used in the BNC data, but might apply, for example, to editorial notes inserted in a text.
- to treat a particular tag as the start of the taggable text. In the BNC written text was always enclosed within <text> ... </text> markers, and spoken text was enclosed within <stext> ... </stext> markers. If CLAWS reads an SGML-conformant text it always reads up to the first such start tag before processing any text, stopping when it meets the corresponding end tag, and starting to process text again if it meets a further start tag.

One problem with skipping everything up to the first start tag, is what to do with the characters passed over, since we presumably wish to end up with a single text file containing all the original information plus the additional part-of-speech information. Since the first version of CLAWS its output format has been rather restricted; what is referred to as 'vertical output' (see Section 7.6), with a single orthographic unit on each line, and a fixed format of text line reference, word (up to 25 characters), subsidiary information (enclitic markers, error markers, CLAWS decision codes, etc.), and the part-of-speech tags. A number of programs have been written at UCREL (including several editors) which expect text in this format, so it was difficult to change it to a more flexible format. Any data that do not fit within this rigid format, including the SGML **header information** which precedes the text proper in an SGML document (often several thousand characters long in the BNC) are therefore copied to a supplementary free-format output file, and a marker is placed in the normal CLAWS output file indicating their offset and length in the supplementary file. Thus the two files could be merged back together without loss of information, after the post-editing and other post-processing had taken place.

This supplementary file also solved a problem never satisfactorily resolved in earlier versions of CLAWS; what to do about long words. In earlier versions of CLAWS a long word (that is, one longer than 25 characters) was simply truncated, with an error message. Now it is simply inserted in the supplementary file and a suitable pointer inserted in the normal CLAWS output file.

The table of SGML entities (marked by an opening '&' and a closing ';') indicates for each valid entity what class of character it represents, and CLAWS takes an appropriate action for each class. Some of the more commonly encountered classes are:

- accented letters such as &Eacute; (representing an upper case letter) or &ccirc; (representing a lower case ĉ). Since words containing accented letters may be naturalized into English with or without accents, a word containing entities of this class is looked up in the lexicon with and then without the accents. The SGML entity table entry for this class specifies what unaccented character or characters correspond to this accented letter – thus &Eacute; corresponds to 'E', and &oelig; corresponds to 'oe'.
- certain classes of character can be specified as of a type of character to be ignored. Thus, the word Unix&copy; is treated as if it were 'Unix', since the character &copy; (a copyright sign '©') is treated as a character to be ignored.
- a further class of SGML entities, including such entities as &frac13; for the fraction one third, is specified as to be treated as part of a numerical value.

An example of an SGML input text to CLAWS4, illustrating some of the features mentioned above is

```
<text>
The na&iuml;ve cat sat on the <hi rend="italic">Persian</hi> mat.
</text>
```

which represents the sentence 'The naïve cat sat on the *Persian* mat.'

7.4 Tagging Individual Words

In this section we discuss how a set of one or more potential part-of-speech tags is associated with each individual orthographic unit – a word or other sequence of graphic characters considered as a unit.

The assignment of tags is treated as a sequence of tests of the current orthographic unit. If a test succeeds then an appropriate set of tags is assigned; if not, the next test is applied. The sequence of tests is as follows:

1. First a number of tests are carried out for orthographic units of certain special types:
   (a) for long words (i.e. words over 25 characters long), which are treated by default as common nouns
   (b) for truncated words, which are given the unknown tag (FU in C7 or UNC in C5). In the spoken part of the British National Corpus truncated words (e.g. *never* truncated to *nev* at a point where an utterance is interrupted) are bracketed by the SGML tags <trunc> ... </trunc>. In tagging other spoken corpora, such as the **COLT corpus** of London teenage discourse[3] where truncated words are marked by a trailing =-symbol, a pre-processing program is used to map the notation into the BNC format
   (c) and for clitics, such as the *'ll* of *he'll* (see Section 2.2)
2. Next the full word is first looked up in the main CLAWS **lexicon**. The look-up procedure is simplified by converting the word to be looked up into a standard form (all lower case, no abbreviatory full stops). If an entry is found in the lexicon for this word, then it contains a list of potential tags for the word, but the tags are annotated with the type of orthography to be expected if the word is to be allocated this tag. A filter process uses the orthography of the word in the text to retain only the appropriate subset of the tags. Thus a lower-case word would retain only those tags marked as appropriate for lower case, while a word with an initial capital would retain the tags marked as expecting an initial capital (types of proper noun, for instance). But those tags marked as appropriate for lower case are also retained, since it is common to find words of this type capitalized at the beginning of sentences or in headlines. It would have been possible to apply this filtering process more selectively, since, for example, in the BNC, headlines are marked with a special SGML tag. However, the capitalization process

is more widespread than this; furthermore the SGML headline tag has not always been found to be present where expected
3. Tests are carried out for dropped initial *h* and final *g*, succeeding only if the resulting word is found in the lexicon (for example *'ouse* and *anythin'*). This tends not to be very useful for the transcription of the spoken part of the BNC, but it is quite useful in representation of spoken dialogue in written texts.
4. Tests are then carried out for words with a trailing *s*; this is stripped off, and spelling rules are used to obtain a suitable base form, which is then checked against the lexicon. A filtering process retains only those tags consistent with a trailing *s* (plural nouns of various types, and third person singular forms of verbs).
5. Next there are a number of tests for special orthographic units of various sorts – this set of tests has tended to be expanded fairly frequently as new classes of orthographic unit are recognized by the analysts post-editing the output text. This step deals with
   (a) individual letters, numbers of various types, Roman numerals
   (b) words of the form *A/B*. The two portions are looked up in the lexicon, and the common set of tags from the two parts – words *A* and *B* (if any) – are assigned to the word *A/B*
   (c) formulae, recognized by containing a mixture of letters and numbers, or containing special characters like +, are assigned a formula tag (FO in C7).
6. Words containing a **hyphen** are dealt with at this stage (if the word has not already been dealt with, by appearing in the lexicon). There are three main procedures:
   (a) since certain words can appear with or without hyphens, and since extra hyphens can sometimes be inserted because of line-breaks in the text, the hyphens are first removed and the word looked up again in the lexicon
   (b) a second procedure recognizes certain prefixes which can be added to a word without changing its grammatical class. Thus a prefix of this type can be recognized in *counter-attack*, and the appropriate tags extracted from the lexicon entry for *attack*
   (c) finally, a hyphenated word *A-B* is broken into the two parts *A* and *B*, these parts looked up in the lexicon, and an attempt made to construct tags for *A-B* from the tags for *A* and *B*. Thus past tense of verb followed by adverb or preposition can result in adjective (as in *fed-up*).
7. The next step is an attempt to predict the appropriate tags by considering the ending of the word:
   (a) First the word-ending *er* is treated specially. The ending is stripped from the word, and the result looked up in the lexicon. Essentially

verb tags associated with the stem indicate a common (agentive) noun for the word (e.g. *listener*), and adjective tags indicate a comparative adjective (e.g. *odder*)

(b) A list of **word-endings** with associated tags is then searched for the word, and the longest match found. As with searches in the lexicon a filtering process is again used, so that a word with a particular ending could have distinctive tags if it appears with or without an initial capital. (For example, *-man* with a word-initial capital is likely to be a proper noun, but not if the word begins with a small letter – compare *Bowman* and *bowman*)

(c) Finally a trailing *s* is stripped, and the resulting stem looked up in the list of word endings.

8. The final step is a default procedure, if all the above tests have failed. Any orthographic unit reaching this point is allocated the default set of potential tags – noun, verb, adjective, adverb.

7.5 Using Probabilities

The result of the procedures described in the previous section is that each word in the text receives one or more part-of-speech tags. The task of the probabilistic part of CLAWS is to choose a single preferred tag in cases where a word has more than one. In these cases CLAWS in fact ranks all the potential tags, from most likely to least likely, assigning to each a probability of the tag being the correct one. This probability figure can be used to estimate the likelihood of the preferred tag being the correct one, and allows the introduction of **portmanteau tags** (see Section 9.3).

As mentioned in Section 7.1, the basic mechanism used by CLAWS is to estimate the likelihood of tags over a sequence of words starting with a word with a single unambiguous tag, continuing over a sequence of one or more words with more than one potential tag, and finishing again with a word with a single tag. Since punctuation marks are unambiguously tagged in the CLAWS system, in the worst case the sequence of words would be a complete sentence, but it is usually shorter. In principle CLAWS then considers each sequence of possible tags for this sequence of words, estimates the probability of that sequence, and then chooses the sequence with the highest probability. The probability of a sequence is calculated from:

- the conditional probability $P(t \mid t_{-1}\, t_{-2} \ldots)$ of a particular tag t given that the preceding tags were t_{-1}, t_{-2}, etc., and
- the conditional probability $P(w \mid t)$ of a particular word w, given that

the associated tag was t.

Consider an example, where words w_0 and w_4 are unambiguously tagged t_0 and t_4 respectively, and the intervening words w_1 to w_3 have two or more potential tags each:

w_0	w_1	w_2	w_3	w_4
t_0	t_{11}	t_{21}	t_{31}	t_4
	t_{12}	t_{22}	t_{32}	
			t_{33}	

Then the probability of the words w_0 to w_4 being tagged t_0, t_{11}, t_{21}, t_{31}, t_4 is the expression

$$\begin{array}{ll} & P(w_0 | t_0) \times \\ P(t_{11} | t_0 \ldots) & \times\ P(w_1 | t_{11}) \times \\ P(t_{21} | t_{11} \ldots) & \times\ P(w_2 | t_{21}) \times \\ P(t_{31} | t_{21} \ldots) & \times\ P(w_3 | t_{31}) \times \\ P(t_4 | t_{31} \ldots) & \times\ P(w_4 | t_4) \end{array}$$

and a similar expression can be calculated for each possible tag sequence. Establishing the most probable sequence in this way can result in a large amount of calculation, especially for a long sequence of ambiguous words each with several alternative tags. However, there is a procedure, called the **Viterbi alignment**, which can reduce sharply the amount of effort required:

1. We can calculate the probability of the most likely (indeed the only) path from t_0 to each of t_{11} and t_{12}. For example, the probability of the former is $P(w_0 | t_0)$ x $P(t_{11} | t_0 \ldots)$ x $P(w_1 | t_{11})$.
2. We can then calculate the probability of the most likely path from t_0 to each of t_{21} and t_{22}:
   (a) consider the path from t_0 to t_{21}; it goes through either t_{11} or t_{12}. The probability of the path going through t_{11} is the probability of the most likely (the only) path from t_0 to $t_{11} \times P(t_{21} | t_{11} \ldots) \times P(w_2 | t_{21})$
   (b) we can similarly calculate the probability of the path through t_{12}, and then choose the path to t_{21} with the highest probability. We record this highest probability, together with information as to whether the path went through t_{11} or t_{12}
   (c) we use the same mechanism for choosing the most probable path to t_{22}.
3. We can then calculate the probability of the most likely path from t_0 to each of t_{31}, t_{32} and t_{33}. This calculation looks back to the information

stored for the possible paths leading up to word w_2, but no further.

4. We can continue with this **forward** calculation, until we reach the end of the ambiguity. At the end we know the probability of the most likely route, and which was the best choice of the last ambiguous tag (here t_{31}, t_{32} or t_{33}). But the information stored with this tag enables us to find the best choice of the next to last ambiguous tag (here t_{21} or t_{22}).
5. We can make a **backward** pass, extracting the best choice of tag for each word as we go.

The above Viterbi calculation tells us the most likely tag sequence, and what its probability is. We may also want the probabilities of the individual tags. For example, the most likely path in the above example might be t_0, t_{11}, t_{21}, t_{31}, t_4 with a probability of 0.4; but there might be two paths through tag t_{12} each with a probability of 0.3 (and all other paths have negligible probability). Then the individual probability for the tags t_{11} and t_{12} is 0.4 and 0.6. We can calculate this by an extension of the above Viterbi alignment; on the backward pass we make a similar calculation to that on the forward pass, and from this we can calculate the individual probabilities (see Jelinek 1976, 1990).

CLAWS carries out the above calculations for all sequences of one or more ambiguously tagged words, and reorders the tags by decreasing individual probability, but with the tag on the most likely tag sequence first; the first tag in the list is CLAWS' preferred tag for this word. It is possible (as indicated in the example above) for the tag with the highest individual probability not to be the tag on the most likely sequence, but it is rare.

The probability calculation makes use of information about the likelihood of one tag following another. CLAWS4 is set up to allow the likelihood figures to make use of only the preceding tag, $P(t \mid t_{-1})$, or of the two preceding tags, $P(t \mid t_{-1}\ t_{-2})$. Most of the recent work with CLAWS4 has made use of only the bigram statistics $P(t \mid t_{-1})$. As part of the production of the British National Corpus a two-million word **sampler corpus** was constructed, and this was manually post-edited so that only a very small percentage of errors are likely to remain in its tagging. This has been used as **training data**, to generate a set of bigram probabilities of one tag following another. In fact the sampler corpus is made up of one million words of written text and one million words of spoken text, so two separate probability matrices have been generated, one for tagging written material and one for spoken material.

There is a problem with the calculation of probabilities of tag sequences. If a certain tag transition has never been seen in the training data, then any tag sequence containing this transition will have a probability of zero, and will never be considered. A probabilistic tagger works on the principle that all tag sequences are possible, but some are more probable than others. The CLAWS system is therefore set up so that any tag transition which does not occur in the training data is given a very small probability, so the transition is not treated as completely impossible.

The calculation of the best tag sequence also makes use of the probabilities P(w l t), that a particular word is associated with a particular tag. CLAWS in fact stores information about a particular tag being associated with a particular word P(t l w), and uses Bayes' theorem (see Jelinek 1990) to calculate P(w l t). There are two mechanisms in CLAWS4 for supplying these probabilities:

1. As with the bigram information, figures for word-tag associations can be extracted from a corpus of correctly tagged text. Current versions of CLAWS make use of a lexicon induced from the BNC written and spoken sampler corpora, and this has word-tag association figures for all words which appear sufficiently often in the tagged text (see Section 9.2.3).
2. The word-tag association information is likely not to be useful for words which occur only infrequently. Further, it is difficult to arrive at suitable frequency figures for words which have been assigned a set of potential tags as part of some rule-driven procedure, for example for dealing with hyphenated or capitalized words. For this reason CLAWS4 has a second mechanism for indicating crude frequency estimates in cases where good frequencies are not available; in earlier versions of CLAWS *all* word-tag association information was of this type. If good frequency information is unavailable, a linguist can indicate in the lexicon that for a particular word a particular tag is unlikely (nominally less than 10 per cent chance, indicated with a '@'-character) or very unlikely (nominally less than 1 per cent chance, indicated with a '%'-character). Similarly some of the rule-driven procedures deliver frequency estimates of this crude form.

The CLAWS4 probabilities are all obtained by extraction from text corpora which have been corrected by hand. There is another mechanism by which probabilities can be estimated. If we start off with a set of tag transition and word-tag probabilities, and with a corpus of text (without part-of-speech annotation) it is possible to perform an iterative procedure called the **forward-backward algorithm** which adjusts the probabilities a bit at a time in the light of possible tag sequences estimated to occur in this training data. Thus an initial set of estimates of probabilities is adjusted in the light of a quantity of training data of an appropriate type to give a more accurate set of probabilities which can be used on other texts of the same type (see Jelinek 1990 and Chapter 10 for more details of this **self-**

organizing methodology).

7.6 Using Contextual Patterns

Early in the development of the CLAWS1 system, two problems were noticed with the mechanism described above:

1. Some text items are traditionally written as two or more separate orthographic words, but function as a single grammatical unit; obvious examples are multiword prepositions such as *according to* (see multi-words, Section 2.2). The UCREL team came to refer to the tags associated with these multiword units as **ditto-tags**, since a sequence of orthographic words would receive the same tag.
2. There were some segmented patterns of words which the probabilistic mechanisms described above did not handle very well, and which could be handled by searching for a few simple patterns of words and/or tags.

It was therefore decided to write a simple pattern matching module, which would run immediately before the Viterbi alignment procedure. There would be a small number of patterns (in the first version of CLAWS, some 150 patterns) each with an associated action – to insert one or more tags on one or more words matched by the pattern. Although the ditto-tag problem could have been solved by extending the lexicon to include multiword units, it was decided to use the contextual pattern matching module for this task as well.

The contextual pattern matching mechanism has been extensively developed in more recent versions of CLAWS. A pattern to be matched consists of a sequence of two or more elements, to be matched to a sequence of two or more words in the text. An element to be matched can consist of any of the following:

- a word (for example *according*), a regular expression representing a word (for example any word ending with *-ing*), a particular word with initial capitalization (thus a pattern element *Times* would not match *times*), any word with an initial capital (this is useful for matching the open-ended portion of certain types of geographical naming expressions, for instance), or any of a list of similar words (this allows multiple patterns to be encoded more concisely)
- a part-of-speech tag (for example any word assigned a potential adjective tag), a regular expression representing a tag (for example any tag starting with an N, indicating a noun tag of some form), or any of a list of possible tags

- an indication that the match must *fail* – for example, it is possible to search for a pattern of words or tags *not* preceded by some part of the verb *to be*
- an indication that a particular pattern element is optional (a common element useful in correcting verb patterns allows an optional intervening adverb or the word *not* or *n't*)
- an indication that an optional element can be repeated a number of times (it is, for example, possible to indicate that up to three words can occur between the two parts of a pattern).

A rather rebarbative syntax has grown up over the years of CLAWS development for indicating all the above types of pattern element. More recent developments, particularly the Template Tagger described in the next chapter, have cleaned up and extended the syntax to allow more powerful matches than are currently possible with CLAWS. An example would be the possibility of defining a named set of words which could be quoted in a number of different rules; in CLAWS the list of words would have to be written out in each pattern which required them.

There is a problem in the CLAWS contextual pattern matching (which re-occurs in other pattern matching programs; see Sections 8.3.3 and 9.2.5) to do with dealing with the overlap between patterns, or the decision of which is the preferred pattern if several match simultaneously (and given that the application of the actions of one matching pattern might cause other patterns to cease to match). CLAWS has a conceptually simple mechanism for dealing with multiple matches:

- the text is scanned from beginning to end, and each pattern is tried at each word position in the text with all possible structures
- if there are more than one matching patterns starting at a particular point, then a score is calculated for each such pattern and the actions of only the highest scoring pattern are carried out. The score is based on the type of match (for example, an element matching on a word scores higher than a match on a tag, and a pattern most of whose matches are on words scores higher than one most of whose matches are on tags), and then on length of match (a longer pattern scores higher than a shorter one). Thus *as well as* beats *as* Adjective *as* (by the first criterion) and beats *as well* (by the second criterion)
- when a pattern is chosen by the above criterion, then all other patterns commencing at the same point are abandoned. Furthermore, all patterns which begin within the scope of the matching pattern are abandoned, and the pattern matching recommences immediately beyond the matched pattern. Thus it is not possible with this mechanism to recognize a pattern within the scope of another pattern; for example,

a multiword adverb (recognized with a ditto-tag pattern) within a verbal pattern requiring an adverb.

To provide more flexibility the pattern matching in the latest versions of CLAWS is divided into a number of passes. There are two passes before the Viterbi probabilistic disambiguation described in the previous section and two afterwards. The idea is that most multiword units requiring a ditto-tag will receive one in the first pass, and then the results of this pass are available to the second pass. While patterns in the first two passes match *any* potential tag, those in the final two passes match only on the tag preferred by the Viterbi process.

7.7 Conclusions

The result of running CLAWS4 over the text displayed at the end of Section 7.3 is as follows:

```
**6;0;START     NULL
--------------------------
The             AT0
na&iuml;ve      AJ0
cat             NN1
sat             [VVD/91] VVN@/9
on              [PRP/90] AVP@/10
the             AT0
**18;7;hi       NULL
Persian         AJ0
</hi>           NULL
mat             [NN1/99] AJ0@/1 VVB@/0
.               .
**8;26;text     NULL
```

The first column represents the words of the text; items commencing ** indicate a reference to a particular position in the supplementary file (here corresponding to most of the SGML tags). The second column represents the part-of-speech tags assigned by CLAWS, with the preferred tag at the left. Where there is a choice the numbers indicate the percentage likelihood of the individual tag, and the square brackets indicate the preferred tag sequence. SGML tags are given the special part-of-speech mark NULL. The dashed line indicates the insertion by CLAWS of a sentence break. The actual output from CLAWS also includes line reference numbers, tagging decision codes and other subsidiary information.

Currently, CLAWS4 operates with an accuracy rate of some 96–97 per cent across the whole range of texts in the BNC. If manual post-editing is required, an X-Windows-based editor Xanthippe (see Section 13.3.1) provides a user interface onto the (vertical format) text, allowing a correct tag to be promoted to the preferred tag position or a new tag inserted from a panel of options. Other facilities allow the editor with a few key-strokes to insert or delete sentence breaks, split or join words, or modify the ditto-tagging.

A final program in the suite reformats the output from CLAWS, whether post-edited or not, into normal **horizontal running text** with the part-of-speech tags added; for the BNC the tags are represented as SGML entities. At this stage the characters from the supplementary file are inserted into the output, resulting in a text such as:

```
<text>
<s>
<w AT0>The<w AJ0>na&iuml;ve<w NN1>cat<w VVD>sat<w PRP>on
<w AT0>the<hi rend="italic"><w AJ0>Persian</hi><w NN1>mat<c PUN>.
<s>
</text>
```

In the introduction it was stated that most of the BNC was tagged with a C5 tagset of some 61 tags. In fact it was tagged with a slightly larger tagset: the additions were **process tags**, making distinctions which were useful at the disambiguation stage, but not required in the final result – the mapping to remove these extra tags is performed at the final post-processing stage, as is the introduction of **portmanteau tags**; that is where the CLAWS system is unable reliably to decide between two tags, and consequently both tags are assigned to the word output. This final program decides whether a portmanteau tag is appropriate based on the individual word probabilities calculated by the Viterbi processing described in Section 7.5; the issue of portmanteau tags is discussed further in Section 9.3.

This chapter has described the general structure of the current version of the CLAWS tagging system (CLAWS4), incorporating both a basic probabilistic Viterbi process and a supplementary rule-based set of components, capable of assigning part-of-speech marks to general text with an overall accuracy rate in the region of 96–97 per cent. The next chapter describes the Template Tagger, a program which extends the pattern matching techniques of Section 7.6, to insert further grammatical annotation or to correct annotation (such as part-of-speech information) which has already been inserted. Chapter 9 describes in more detail how the CLAWS program was adapted for use in tagging the British National Corpus with its wealth of different text types.

Notes

1. It should be noted, however, that Voutilainen uses a more complex measure of tagging success, derived from information retrieval, calculating two percentage figures known as **precision** and **recall**. Recall is the extent to which all legitimate readings are found in the output of the tagger – allowing, that is, for ambiguous taggings of one word. Precision is the extent to which illegitimate readings are discarded from the output (see Voutilainen 1995: 172). Optimally, both measures should be 100 per cent. Voutilainen (1995: 275) records the following impressive result of one experiment: Recall 99.77 per cent; Precision 95.54 per cent.
2. The name CLAWS3 was applied to a modified version of CLAWS2 which involved attempts at verb subcategorization. It never became a fully developed system.
3. The COLT corpus was collected and transcribed at Bergen (see Haslerud and Stenström 1995). Part of it was incorporated into the BNC as part of the spoken material, but an enhanced version, in a more detailed transcription, is being grammatically tagged at Lancaster, using CLAWS4.