5 A review of part-of-speech tagging technology

In this chapter, I will examine a variety of work to date in the field of part-of-speech (POS) tagging technology, looking in particular at a range of different POS tag disambiguation techniques. As I will outline in the following chapter, my part-of-speech tagger for Urdu is based on the use of disambiguation rules devised by a linguist, which I claim to be the optimum technique for the purpose; the goal of this chapter is to justify that claim. To this end, I will go into depth on previous research into a wide range of disambiguation techniques, since a full understanding of those techniques is a necessary prerequisite for justifying my choice. Having reviewed the field, I will discuss the factors that influence the choice of a technique, demonstrating that the rule-based approach to tag disambiguation is the most appropriate to the problem at hand. In particular I will demonstrate that comparisons between the performance of different taggers is highly problematic, and thus that performance is not a criterion which can be used to justify the choice of a disambiguation technique.

There are three main sub-tasks that a full automatic tagging system must accomplish:

> "segmentation of the text into tokens;
>
> assignment of potential tags to tokens, usually resulting in ambiguity;
>
> determination of the contextual appropriateness of each potential tag, usually in order to remove the less appropriate tags and thus resolve the ambiguity."
>
> van Halteren and Voutilainen (1999: 109)

A further conceptual division which is frequently encountered is the splitting

of the second subtask into a lexical phase, where words are looked up in a lexicon and given the tags listed in their lexicon entry, and an analysis phase, where potential tags are assigned based on morphological "hints" given by the spelling of the token (this is sometimes referred to as "suffix analysis").

This conceptual division is frequently an actual division in terms of the software used. This is, for example, apparent in the CLAWS part-of-speech tagging system (Garside, Leech and Sampson 1987). It is also is apparent from the fact that many individual studies in the literature discuss techniques or software that address only one of these subtasks.

This chapter is principally concerned with justifying the choice of the technique used for the subtask of determining contextual appropriateness – otherwise referred to as *disambiguation*. Each technique depends on a model of language use which differs in some way and "[i]t is in these models, and hence within this subtask, that the highest variation exists" (van Halteren and Voutilainen 1999: 110).

Accordingly, I postpone discussion of work done previously in the areas of segmentation of text into tokens and assignment of potential tags, and will consider now the range of disambiguation techniques[1], as a preparatory step towards the selection of a rule-based methodology (see section 5.7).

5.1 A proposed typology of disambiguation methodologies

It is notable that despite the considerable range of techniques employed in part-of-speech disambiguation, the contextual information analysed by a

[1] This review inevitably revisits a number of studies discussed in chapter 2. However, the focus there was on their categorisation schemes (tagsets), whereas here it is on their disambiguation methods.

disambiguation algorithm is typically minimal. As will be demonstrated by the discussion in the remainder of this chapter, preceding or following words, or the tags that these words have, are the only information utilised to any great degree in disambiguation. No use is made of more abstract syntactic concepts such as the noun phrase, the verbal group, or the clause[2]. This is true almost by definition: an approach to morphosyntactic categories that made use of higher-level structures would probably considered a form of parsing rather than part-of-speech disambiguation *per se*. But as Brill, among others, has pointed out, restricting disambiguation to use of this minimal information can be highly effective:

> Corpus-based methods are often able to succeed while ignoring the true complexities of language, banking on the fact that complex linguistic phenomena can often be indirectly observed through simple epiphenomena. For example, one could accurately assign a part-of-speech tag to a the word *race* in (1-3) without any reference to phrase structure or constituent movement: one would only have to realize that, usually, a word one or two words to the right of a modal is a verb and not a noun. An exception to this generalisation arises when the word is also one word to the right of a determiner.
>
> (1) He will race/VERB the car.
> (2) He will not race/VERB the car.
> (3) When will the race/NOUN end?
>
> Brill (1995: 544)

It is worth pointing out, of course, that non-corpus-based approaches to disambiguation utilise exactly the same information in the input data as the corpus-based method suggested by Brill. Before proceeding to an examination of assorted

[2] An exception here is Constraint Grammar (see 5.2.2), which makes use of clause boundaries.

disambiguation techniques, corpus-based and non-corpus-based, it is worth establishing exactly what types of techniques have been developed and on what basis they differ from one another.

It is possible to group models of language used in disambiguation in two ways. Firstly, do the linguistic generalisations in the model derive from the grammatical knowledge of a linguist or from a corpus of texts? Secondly, are these linguistic generalisations expressed as rules or as probabilities? Combining these two classifications, four logically possible disambiguation methodologies exist, as shown in the diagram below.

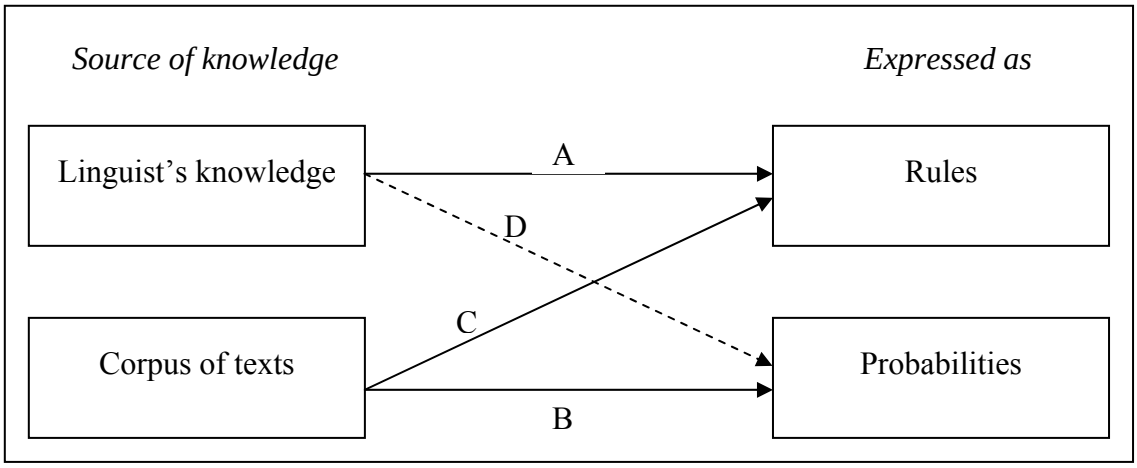

Fig. 5.1

This is, however, an idealisation in terms of what actual tagging systems have been developed. While any given methodology can be captured in this typology, a system could easily be based on a combination of methodologies, allowing for more types of system than the basic four. For example, the CLAWS system contains modules that utilise different methodologies. Its CHAINPROBS module, for instance, is type B, and its IDIOMTAG module is type A. See 5.3.2.2 and 5.6.3 for full discussion of this system.

In type A methods, the linguist's knowledge is expressed as "rules". Typically these rules are used by the tagging program to alter the tags associated with the text in

such a way as to reduce the ambiguity. Systems of this sort were the earliest to be developed, in the 1960s and 1970s, although major advances were made in the 1990s. They are discussed in section 5.2 below. Type B methods, by contrast, typically gather corpus-derived data on the frequencies in which different sequences of tags occur. They then use this data to decide which of the possible tags given to a word is most likely given the surrounding tags, employing some statistical model such as a Markov model. Chronologically, these *probabilistic* or *stochastic* methods (described in section 5.3) were the second to develop, principally in the late 1970s and 1980s.

The most recent approach to part-of-speech disambiguation techniques to be developed is that involving generally applicable machine-learning techniques such as neural networks. These are strictly speaking type B methods, since the linguistic information in the model derives from a corpus and is stored as a large number of numerical parameters. However, the parameters are not tag sequence frequencies, and the variety of models used differ greatly from Markov models. Thus I discuss such approaches separately in section 5.5.

The methodology identified in the diagram as type C, where corpus-based rules are utilised, is identified principally with the work of Eric Brill[3] through the 1990s, who calls this approach "transformation-based error-driven learning". While, as the diagram above illustrates, this methodology bears a strong resemblance to elements of the rule-based method and the stochastic method, the way in which these two approaches are hybridised effectively make this an entirely separate method, and I shall consider it as such; it is discussed in section 5.4.

There are to my knowledge no methodologies of type D (thus the dashed line indicating what would be its mode of operation in the diagram above). It is not hard to

[3] See Brill (1992, 1994, 1995, 1999) and Brill and Pop (1999).

hypothesise how this asymmetry in the range of methodologies comes to be. As McEnery and Wilson (1996: 12) point out, human beings are typically extremely unskilled in estimating frequencies of words, phrase structures and other linguistic phenomena. It would be a perverse disambiguation methodology indeed which, using the knowledge of the human linguist as its source of data, elected to express this knowledge in the form of probabilities – which humans usually get wrong – instead of rules – which they are much more likely to get right. This is not to say that human impressions of linguistic probability have no place in computational linguistics. Early versions of the CLAWS system (Atwell 1987) utilised very approximate human-estimated probabilities within the lexicon. Likewise, McEnery (1995: 201-219) uses a human perception of probability in the context of initial rule-ordering within a computer system to model pragmatic reasoning. However, no full probabilistic model of linguistic knowledge in general or part-of-speech disambiguation in particular has been based upon human-estimated probabilities.

I now proceed to consider in turn rule-based approaches, probabilistic approaches, approaches utilising corpus-derived rules, and approaches based on machine-learning techniques. I will then in section 5.6 review some efforts that have been made firstly to compare, and secondly to synthesise the different methods discussed.

5.2 Rule-based approaches to disambiguation

The basic principle of rule-based approaches is that the knowledge base consists of a set of linguistic generalisations, known most commonly as *rules* or *constraints*. Each rule contains instructions for an operation to be performed, and a

context describing where that rule should be applied. The operation to be performed alters the list of tags associated with an ambiguously-tagged word in such a way that one or more potential tags are eliminated from consideration, reducing the ambiguity. For instance, a rule for an English tagger might state that where one of the potential tags for a word is *infinitive verb*, that reading should be removed *if* the preceding word is not tagged as 1) a modal verb, 2) the primary verb "do", or 3) the infinitive marker "to" (example from Voutilainen 1999c: 230). This rule takes advantage of the known restriction of the infinitive form of the verb in English to these contexts. Different systems have used different computational implementations, which in turn allow different types of rules to be incorporated into the systems.

As can be inferred from the foregoing brief description, taking a "rule-based" approach to disambiguation in tagging does not imply using grammar rules as traditionally formulated by linguists. Disambiguation, rule-based or otherwise, typically makes use of short-range information[4], as mentioned above and as the examples below will make clear. This is a far cry from the theoretical model proposed by many researchers into syntax (compare for instance the clause/phrase structures and processes described by Chomsky[5]).

Rule based approaches are often associated with parsing. For example, Harris' (1962) program was a parser, and Klein and Simmons (1963) depict their rule-based tagging as a preliminary stage to an eventual parsing process. More recently, Hindle utilised a parser in part-of-speech disambiguation (see section 5.4.1). However, there is no necessary link between rule-based approaches and parsing, as demonstrated by

[4] The Constraint Grammar system (see 5.2.2) is an exception here; it does make use of longer-range information.

[5] See for example Chomsky (1957).

Greene and Rubin (1971), whose tagging was not associated particularly with a parser.

Historically, there have been two distinct phases of work on rule-based disambiguation. The earliest work, hinging on the studies of Klein and Simmons (1963) and Greene and Rubin (1971), was begun before any other methodology had been developed, and marked the first attempt to solve the problem of automated POS disambiguation. By contrast, the more recent rule-based approach known as Constraint Grammar, and centring on the work of Karlsson et al. (1995), was undertaken in the light of an intervening decade of research into probabilistic disambiguation, which at that time seemed to provide better results than the early rule-based methods. I discuss the earlier phase in section 5.2.1 below, and move on to a discussion of Constraint Grammar in 5.2.2.

5.2.1 Early work with rule-based disambiguation approaches

Although not the earliest work in the field (see Harris 1962[6], Joshi and Hopely 1997), the study of Klein and Simmons (1963) is frequently cited as such, since their program, called CGC, was the inspiration for later taggers, including that of Greene and Rubin (1971). It should be noted that both Klein and Simmons and Greene and Rubin worked exclusively on English.

Like later taggers, CGC utilises a lexical lookup stage[7] followed by a suffix

[6] Harris' work is not discussed further here, since his system was primarily a parser.

[7] In this case, the lexicon consisted of 2000 items, split for computational purposes into several smaller units. At this time, due to hardware limitations, the working memory available to a program was very much smaller than it is today.

lookup stage to assign an initial set of one or more tags to each word in a sentence. Ambiguity is thus introduced into the analysis. It is removed by the application of rules in the form of what Klein and Simmons refer to as a "context frame test".

This test is worthy of some brief discussion, since its means of operation is somewhat different to that of the typical "rule" described above. The CGC program includes a "context triad frame table", which lists the possible tag sequences which may occur between two unambiguously tagged words. Each such ambiguous sequence encountered is looked up in this table, and the allowable tag sequences compared to the potential tags previously given to the ambiguous words. So if one of the tag sequences suggested for a two-word stretch of ambiguity was NOUN – VERB, but VERB was not among the potential tags for the second word, this sequence would be eliminated from consideration.

Despite what can be seen in retrospect as a unique algorithm, the basic principle of CGC was the same as that of later rule-based systems: "the basis for computing grammar codes [i.e. tags] for the words in a sentence is one of successively reducing the number of combinatorial choices of word-class codes" (Klein and Simmons 1973: 342) – that is, one of reducing ambiguity.

Klein and Simmons report that the CGC system was capable of handling sequences of up to three ambiguously-tagged words using 500 entries in the context triad frame table (out of 2700 that could conceivably be programmed using their formalism). They report a 90% accuracy rate using a set of 30 tags.

The TAGGIT program described by Greene and Rubin (1971) uses an initial stage of lexical lookup and suffix analysis, like CGC. However, TAGGIT is able to handle many more complex exceptions than CGC for such typographic phenomena as amounts of money, words containing apostrophes, formulae, capitalised words,

hyphenised words, numbers, etc. TAGGIT's lexicon is also larger.

Yet the disambiguation method, referred to by Greene and Rubin as the "Context Frame Test", differed somewhat from CGC's. Firstly, the scope of the "context frame rules" was greater. They could refer to up to three consecutive ambiguous words with one or two non-ambiguous words before and/or afterwards, although Greene and Rubin decided that the rules should examine only one ambiguity at a time. The rules could also either select a tag from the list of possible options or remove a tag from that list.

Greene and Rubin's first rules were based on intuition. To take two random examples, they wrote rules based on the principles that a verb following a modal auxiliary is an infinitive rather than present tense, and that nominative pronouns do not follow prepositions. They then added rules suggested by a program that had analysed some manually disambiguated data. These latter rules, it should be noted, were capable of introducing errors, i.e. one can conceive of sentences which would theoretically be incorrectly tagged by these rules. Thus, the reduction of ambiguity had to be "paid for" in potential reduction in accuracy of the analysis – a trade-off which is found in many later systems.

It should also be noted that the rules in TAGGIT are applied *in order*, from the most specific to the least specific. "Specific" here refers to the amount of context, in terms of surrounding tags, the rule contains. The more context the rule contains, the more specific that rule is.

Greene and Rubin (1971: 40) report that TAGGIT disambiguated 77% of the

words it processed. The remaining ambiguity was later removed by hand[8]; this contrasts with the CGC system, in which all ambiguities were, in principle, capable of being removed. This distinction, between disambiguation processes which give each word only a single tag, and processes which allow some ambiguity to remain to prevent the system rejecting a correct tag, persists to this day. The latter approach is sometimes referred to as "n-best" or "k-best" tagging (e.g. by Voutilainen 1999b), because the *n* tags are returned which seem best to the disambiguation process on the basis of the information it has. N-best tagging has been commonly associated with rule-based approaches, whereas the full disambiguation approach has been found more commonly in probabilistic approaches[9]. This is perhaps because rules tend to operate by rejecting unwanted potential tags one by one, so it is entirely plausible that when all rules have been applied, some words might still have two or more tags (Brill 1999: 257). But probabilistic methods, as we will see, actively choose a single tag from among the candidates. However, as Brill (1999) also points out, it is computationally equally possible for probabilistic methods to give a word more than one tag, so this correlation should by no means be regarded as a logical necessity.

5.2.2 Work in the Constraint Grammar framework

Part-of-speech tagging in the Constraint Grammar (CG) framework has been

[8] Both Greene and Rubin's work, and the subsequent manual disambiguation, were done in the context of work on the Brown Corpus (see Francis and Kučera 1982). The task of tagging the Brown Corpus was indeed the primary motivation for the creation of TAGGIT.

[9] A few further examples of this include Constraint Grammar (rule-based, does not fully disambiguate), as discussed in the following section, and systems such as CLAWS or the Xerox tagger (probabilistic, disambiguate fully), discussed in 5.3.2.

undertaken principally by Fred Karlsson and colleagues at the University of Helsinki. It is fair to say that CG represents the state of the art in rule-based disambiguation at the time of writing. CG is both a "language-independent formalism" (Karlsson 1995a: 1) and a specific algorithm and system for text analysis. However, it should be pointed out that work has been done which, while similar to CG in being based on rules written by human analysts, are not strictly within the CG paradigm's system. For example, Koskenniemi (1990) describes a system which uses rules in a very similar way, but the system is implemented computationally as a finite state machine.

It should also be noted that the CG framework is not just a POS disambiguation methodology. It is a more general approach which also includes parsing[10] (see Anttila 1995). As discussed above, it is characteristic of many rule-based approaches that they link part-of-speech disambiguation to parsing, and CG is no exception in this. In fact, tagging and parsing may said to be inseparable in the CG framework (see for instance Karlsson 1995a: 11-14, Karlsson 1995b: 41-42).

The CG approach, which constituted a return to earlier rule-based methods, was to some extent inspired by perceived inadequacies of the probabilistic approach which had been developed in the meantime (see 5.3 below):

> First, it is not obvious that stochastic algorithms could qualify as genuine language-independent formalisms … Second, the error rates of the probabilistic approaches seem to remain fairly high […] if substantial precision lowering is not allowed … there seem to be no easily accessible ways of diagnosing errors and trying to improve the performance of a large completed probabilistic system. Finally, it has not yet been conclusively

[10] Note that Karlsson et al. (1995) frequently use the term *parsing* to refer to what I call "part-of-speech tagging", as well as to the less superficial syntactic analysis that the word more customarily refers to.

> shown how successfully a primarily probabilistic approach would carry over into full-scale analysis of unedited text, including grammatical labelling.
>
> Karlsson (1995a: 7)

However, Karlsson is also at pains to point out that the CG approach does not totally reject any role at all for aspects of a more probabilistic approach (1995a: 5, 9), as the "heuristic constraints" within CG may be seen as probabilistic. These heuristic constraints (a constraint is essentially a rule) are to be applied when "linguistic constraints" fail to disambiguate fully (see also below). This is in contrast to a number of other rule-based analysis systems (cited by Karlsson 1995a: 5) which do wholly exclude probabilistic elements[11]. However, heuristic constraints as described by Karlsson may in practice be better described as "unreliable rules" than "probabilistic" in the usual sense.

Since the CG formalism embraces parsing as well as word-class analysis, it follows that a full CG system is not only a tagger but also a parser. The closest thing to a "tagger" *per se* in the CG model may be the "morphological disambiguation module" described by Voutilainen (1995) and specified by Karlsson (1995b: 42, 44) as a single component within the overall CG system. As with the rule-based disambiguation systems described in the previous section, this module applies after one or more analyses have been given to each word by a lexicon and a morphological analyser[12]. These analyses are not necessarily "tags" in the same sense as the word has been used elsewhere. Some analyses consist of multiple tags, e.g. PRON NOM SG3 SUBJ for a third-person subjective pronoun. Other analyses exclude distinctions

[11] The systems that Karlsson lists are parsers rather than taggers; therefore, no further discussion is devoted to them here.

[12] In the CG system, these two processes are realised as a single computational unit – for example, for English, the ENGTWOL program (Heikkilä 1995).

many other taggers would try to make. For example, there is just one tag, PCP1, for words ending in "-ing", whether nouns, adjectives or verbs. However, it is *not* the case that this distinction is ignored in the CG system altogether (which would be simply a matter of tagset composition). Rather, making the distinction between nominal, adjectival and verbal usages "is entirely left to the syntactic part of the [CG] system" (Voutilainen 1995: 167). In other words, part of the tagging task is put off to a parsing stage. While this is a reasonable approach if all data put through the system is to be both tagged and parsed, it is not a possible solution if tagging is required without parsing. For this reason, it may not be entirely appropriate to consider the CG morphological disambiguation module, in isolation from the syntactic parsing module, to be the direct analogue of other disambiguation systems. Nonetheless, for now I will proceed under the assumption that they are comparable.

Within CG, all linguistic analysis, including part-of-speech disambiguation, is performed by the application of rules, or, as they are typically called, *constraints*. Constraints are characterised by Voutilainen (1999c) as rules that "perform operation X on target Y in context Z". The operation may be SELECT (delete all analyses but one, which is taken to be correct) or REMOVE (an incorrect analysis)[13]. Contexts can refer to words or analyses on words (which may be required to be non-ambiguous if the linguist wishes) preceding or following the target word by any distance. They may also refer to clause boundaries, which are marked up by another module of the overall CG system (Karlsson 1995b: 42-43, 64). This is a key difference from earlier systems: the CGC system looked only at words immediately adjacent to ambiguous items, and TAGGIT likewise used a very restricted local context. It is not however clear to what degree the superior performance of CG systems is due to this extended context scope.

[13] See below for an example of the actual CG rule formalism.

Note also that a single CG constraint may specify multiple contexts where its operation may be applied.

How are the rules in a CG disambiguation system derived? Voutilainen (1999c: 226) asserts that "writing a grammar can be a straightforward and, contrary to common belief, fast process." He suggests an approach to rule-writing which is based on a combination of intuition and trial-and-error. The grammarian devises a new rule by examining ambiguities in the output of the system as it stands, applies the rule, and then observes whether or not the overall result is an improvement in the output[14].

The 1,100 constraints used in the CG system for part-of-speech disambiguation in English are described by Voutilainen (1995: 219) as partially expressing "no more than 23 grammatical generalisations, given as prose statements." These generalisations include[15]:

- *To the right of an infinitive marker, there is an infinitive.*
- *A sentence contains at least one (potentially coordinated) main clause.*
- *Determiners agree in number with their heads.*

The third of these statements, for example, would be represented in the formalism by constraints such as these:

[14] There is more similarity here than might be supposed between the CG approach and Eric Brill's approach, discussed in section 5.4 below. In fact, the only noteworthy difference is that the process of data examination, rule creation, and rule testing is performed by a computer program in Brill's approach, and by a human being in the CG approach.

[15] The full set of generalisations, and the constraints deriving from them, including those given here as examples, are described in detail by Voutilainen (1995: 219-268).

(@w=0 DET-SG (1C PL-HEAD))

(@w=0 DET-PL (*1C NON-MOD-HEAD *L)
(NOT *L PL-HEAD))

These constraints could be expressed in prose as, respectively, "a singular-only determiner in the following sentence is to be discarded if the next word is a nominative plural" and "A plural determiner reading is to be discarded unless it is followed by a nominal head in the plural". Note that the second of these constraints exemplifies the use of multiple contexts in a single rule. It should also be noted that none of these constraints add any analysis that is not there in the output from the previous stage of the tagging process. They simply remove superfluous analyses provided by the lexicon/morphological analyser unit.

A distinction is made between constraints which are entirely reliable and constraints, referred to as "heuristic", which do not necessarily work all the time; these are stored separately in the list of constraints and applied after the reliable constraints in a separate cycle (Karlsson 1995b: 68).

Disambiguation programs using the CG formalism typically do not perform full disambiguation, so their performance is evaluated in terms of *precision* and *recall* rather than *accuracy*[16]. Voutilainen (1999c: 219) reports that for a CG-based disambiguation system, "[r]eaching a recall of well above 99% is not particularly difficult, but reaching a precision of well above 95% at the same time may require a considerable effort." That is to say, it is easy to prevent the correct tag or set of tags being rejected by the constraints, but difficult to simultaneously ensure the rejection

[16] For a discussion of the many incompatible measures of tagger performance that have been used in the literature, see section 5.6.1.

of all incorrect tags. However, full disambiguation of 95% of rules represents a substantial advance over the 77% disambiguation rate achieved by the earlier TAGGIT system (Greene and Rubin 1971). To put precise figures on this improvement, the version of the EngCG tagger using 1,100 constraints[17] achieved a recall of at least 99.7% with 93-97% of words fully disambiguated (reported in Voutilainen 1995: 186).

While a majority of the published literature concerns the application of the CG methodology to the tagging of English, the approach has been applied to several other languages, for instance French (Chanod and Tapanainen 1995a, 1995b); Voutilainen (1999c: 242) refers to work done on two other unspecified European languages.

5.3 Probabilistic approaches to disambiguation

The basic principle of probabilistic approaches[18] is that statistical information concerning the frequency with which sequences of tags occur is gathered from long stretches of running text. This data is used to deduce which of the optional analyses of an ambiguously tagged word is the more likely to be correct. For instance, acquiring frequency statistics (or "training") on a tagged corpus of English, a system might discover that the tag for a subject pronoun is followed by the tag for a verb 70% of the time, the tag for an adverb 29% of the time, and the tag for a noun 1% of the time. If

[17] Voutilainen (1999c: 225) reports a later version of the EngCG tagger, EngCG-2, which has 3,744 disambiguation constraints, and performs better than the results cited here. However, it is not entirely specified how much better this more recent performance is.

[18] These approaches are commonly called "probabilistic" or "stochastic". I will use both terms interchangeably.

that system, during the course of tagging, then encounters a word following a subject pronoun that was ambiguously tagged as either noun or verb, it can use its statistical knowledge to deduce that the verb tag is most likely to be correct.

In practice, a model as primitive as my example here would be incapable of handling long sequences of ambiguous tokens and would be unlikely to perform particularly well. Thus, modern stochastic taggers utilise a mathematically more sophisticated approach known as a Markov model. Markov models allow the calculation of the probabilities of different tag sequences by combining different tag transition probabilities. The mathematics of Markov models are discussed in some detail by Charniak et al. (1993).

The most immediate advantage of a stochastic system over rule-based systems is that the linguist does not have to write an effective set of rules to produce an effective system. As Brill (1992) puts it, "[t]he appeal of stochastic techniques over traditional rule-based techniques comes from the ease with which the necessary statistics can be automatically acquired and the fact that very little handcrafted knowledge need be built into the system". Probabilistic systems also represented a step forward in accuracy over early rule-based taggers. A further advantage is that they were in general more widely applicable (although later rule-based methodologies such as Constraint Grammar redressed the balance in these latter respects):

> The strength of the corpus-based approach is that, through probabilistic predictions, it is able to deal with any kind of English language text which is presented to it: it is eminently robust. Its weakness is that the very reliance on probability admits the possibility of error.
>
> Leech (1987: 3)

As mentioned at the outset of this chapter, probabilistic approaches were not,

ultimately, employed in the Urdu tagger described in the following chapter. However, this decision was made in the light of a full understanding of the probabilistic approach to disambiguation, and cannot be justified without a preliminary discussion of this approach. Therefore, in this section, I will consider at length some examples of Markov model taggers, some which require tagged training data and some which do not. However, before discussing modern stochastic disambiguation techniques (section 5.3.2), I will briefly summarise the earliest attempts to use probabilities in morphosyntactic disambiguation.

5.3.1 Early work with probabilistic approaches

Probabilistic approaches to part-of-speech tagging increased greatly in popularity during the 1980s. However, some work in the area was done prior to 1980. An example is Bahl and Mercer (1976), whose program was trained using 40,000 words of hand-tagged text and a technique based on the Viterbi algorithm[19].

Of much greater importance historically, however, is the work of Stolz et al. (1965), whose often-overlooked work is contemporary with the earliest rule-based approach (Klein and Simmons 1963) and precedes by some years the work of Greene and Rubin (1971). It should therefore be noted that probabilistic and rule-based approaches are of equally venerable provenance.

Stolz et al. describe a system called WISSYN, which tags words with one of a

[19] While Bahl and Mercer provide rather few details of their methodology, references made to their work in the later literature (for example, Abney 1997) suggest that their research was along similar lines to that of later Markov model-based stochastic taggers. Therefore, I will postpone discussion of exactly how such taggers operate until the following section, which deals with more recent implementations of the Markov model approach.

set of 18 grammatical classes (for the most part defined distributionally). After performing lexical lookup and morphological analysis[20], a *probability phase* utilised "empirically derived sets of conditional probabilities to predict the grammatical form class of words from given syntactic environments"[21]. The probability tables "were generated separately by a set of computer programs operating on text which had previously been grammatically coded by a number of human experts". This 28,000-word corpus[22] was made up of samples of undergraduate creative writing.

During training, all strings of tags up to 5 items in length (within the bounds of a single sentence) were tabulated. Then, probabilities for the different parts of speech were calculated using 1,2,3 and 4 items as predictors before, after and bridging the predicted unit. During tagging, WISSYN looked up the longest pre-string (i.e. tags before the target word) it could find in its table, the longest bridging string it could find, and the longest post-string. Then it would multiply the probabilities given by each of these predicting strings for each possible part-of-speech. Compared to a Markov model, this means in essence that the probabilities of chains of tags were learned in training rather than being calculated from bigram probabilities at runtime. When more than one "blank" (word lacking a tag – effectively, noun-verb-adjective-adverb ambiguities) occurred sequentially, the system examined the blanks at both ends of the sequence. Having solved the easier of the two, it then treated that annotation as 100% certain (which was clearly *not* the case) and used it as context to

[20] Prior to the stochastic disambiguation, there is also what Stolz et al. refer to as an "ad hoc" phase in which eight types of ambiguities are eliminated by the application of structural rules.

[21] Although Stolz et al. call their process a "prediction" process, their description makes clear that their algorithm performs what would today be called disambiguation.

[22] "Corpus" is indeed the word used by Stolz et al., somewhat unusually in a time when corpus linguistics was generally out of favour (see McEnery and Wilson 1996).

resolve non-peripheral blanks.

Stolz et al. were unable to perform large-scale tests, as they lacked a hand-tagged corpus as a standard for comparison. However they did run tests on a few thousand words of text, both of student writing (from within and without the set of training data) and of newspaper text. The accuracy rates that they report from this experiment range from 91.4 to 92.8%.

Stolz et al. note that instances where the probabilistic portion of their program had to deal with sequences of "blanks" were a source of many errors. This was because, if the first blank were resolved incorrectly, it would reduce the accuracy of the probabilities generated for remaining blanks in the sequence. They suggest as an improvement expanding the model to estimate pairs of words at a time. Indeed, estimating probabilities for multiple ambiguously-tagged words is effectively what later stochastic taggers using Markov models would achieve.

5.3.2 Later work on probabilistic taggers using Markov models

Subsequent work on probabilistic disambiguation has focussed on the use of a single statistical method, the Markov model, which has been used in many statistical disambiguation modules or systems with few variations. After giving an overview of the theoretical basis underlying Markov models (section 5.3.2.1), I will go on to consider two methods by which a tag may be selected for an ambiguous token using a Markov model (sections 5.3.2.2 and 5.3.2.3); in each case, I will illustrate the discussion by reference to an influential early tagger using a Markov model. Subsequently, I will consider the role of lexical probabilities in Markov models (section 5.3.2.4), the processes by which the parameters of a Markov model can be

acquired and smoothed (section 5.3.2.5), and some variations on the Markov model approach which have been implemented in various stochastic taggers (section 5.3.2.6). Finally, in 5.2.3.7 I will briefly consider the success rates that have been reported by researchers working with probabilistic taggers of this sort.

5.3.2.1 Markov models: an overview

In probabilistic POS tagging, a Markov model estimates the probability of a chain of tags, given empirically-derived tag transition probabilities. By comparing the likelihoods of possible tag sequences for a sequence of ambiguous tokens, the likeliest, and hopefully correct, sequence can be identified. Such a model represents a more minimal use of contextual information than the rule-based approach discussed above (or, for that matter, the more context-rich probabilistic approach of Stolz et al.). Marshall (1987) rationalises the limited context by analogy to the TAGGIT system, in which rules that used only one tag to specify their context (although up to four are allowed) account for 25% of the rules on the list but for about 80% of the rule *applications*:

> This disproportionate usage of minimally specified contexts suggested that a more effective method of tag disambiguation might be produced by a system which exploited only the relationship between adjacent tags…
>
> Marshall (1987: 44)

Early work on Markov models was undertaken by Bahl and Mercer (1976). However, my discussion in this section will focus on a small set of stochastic taggers developed since 1980 whose operation has been described in some depth in the published literature and which have, in consequence, been influential in the field.

A Markov model is made up of the following parameters:

> a set of *states*; a set of *transitions*, each going from an initial state to a final state; for each state, a probability distribution for all transitions leaving that state; a set of *output symbols*, that can be emitted when in a state (or transition); for each state (or transition), a probability distribution for all symbols that can be emitted.
>
> El-Beze and Merialdo (1999: 264)

In terms of POS tagging, the states are the tags, the symbols that they output are their associated words, and the transitions from one state to another make up the sequence of tags allocated to a sentence (see diagram below).

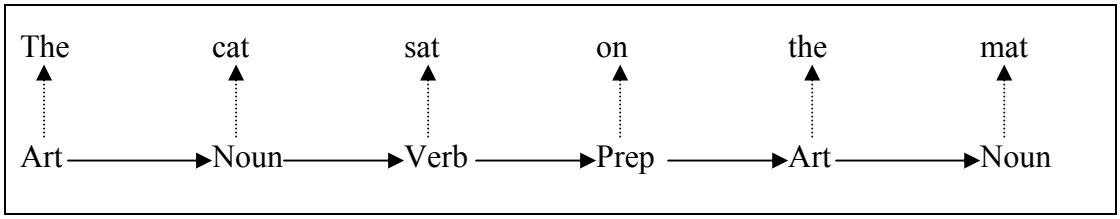

Fig. 5.2

When tagging, a Markov model system knows what output symbols (words) were produced, but not what states (tags) produced them. For this reason, it is common for this type of Markov model to be called a "hidden Markov model", since the transitions are "hidden" from view[23]. The problem is to determine which sequence of tags is most likely to have produced the observed words. The parameters required to fully define the model are thus a) for each state (tag), the probability of a transition

[23] The expression "hidden Markov model" is not used entirely consistently in the literature. Cutting et al. (1992: 133) suggest that taggers trained on tagged data use a Markov model, whereas taggers trained on untagged data use a hidden Markov model (see also 5.3.2.5). By contrast, El-Beze and Merialdo (1999) and Charniak et al. (1993) use the term "hidden Markov model" – or the common abbreviation HMM – for both varieties. To prevent confusion, I will henceforth avoid the expression altogether.

occurring to each state; and b) for each state (tag), the probability of it producing each of its possible outputs (words).

The latter set of parameters is often expressed as the probability of the word (w) given the tag (t), P(w|t)[24]. Essentially this involves the (obviously false) assumption that the only factor influencing the probabilities of one word or another occurring is the tag. The former set of parameters is in principle infinite, since the transition probabilities from any one tag may depend on the entire chain of tags that the model has moved through since the start of the sequence. For example, the probability that the final tag of the sentence above is "Noun" might in principle be different depending on whether the first tag of the sequence is "Art" or "Pron". Since this makes the parameters effectively impossible to estimate, a further assumption is made: that the probability of a tag occurring is determined only by the previous *n*-1 tags[25], where *n* is a small value, typically 2 or 3. This means that the other set of parameters for the model is typically either $P(t_i|t_{i-1}, t_{i-2})$ or just $P(t_i|t_{i-1})$ – the probability of tag A given the preceding tag B or the two preceding tags B and C (El-Beze and Merialdo 1999: 271-272; see also Charniak et al. 1993). Although this assumption too is unjustified, in practice its effect is to restrict Markov model taggers

[24] Throughout I use standard probability notation. P(x) denotes the probability (expressed as a fraction of 1 or a percentage) of event *x* occurring. P(x) = 1 is a certain event, P(x) = 0 an impossible event. x_i is the *i*th in a series of events of type *x*. P(x|a) denotes the probability of *x* occurring if we know that some other event *a* has already occurred. P(x|a,b) denotes the probability of *x* occurring if we know that two other events *a* and *b* have already occurred.

[25] Thus, taggers using Markov models are often called "N-gram taggers" because they look at transition probabilities over *n* words only. Systems where *n* = 2 are often called "bigram taggers", or, more rarely, "digram taggers"; when *n* = 3, the result is a "trigram tagger". Bigram taggers seem to be the more common; an example of a trigram tagger is that described by Merialdo (1994).

to use only local context in disambiguation – which is the case for all taggers (see the quotation from Brill at the start of this chapter).

The process of training a Markov model for POS tagging consists of estimating the parameters of the model, based on an examination of corpus data. Disambiguation using a Markov model consists of applying those probabilities to the task of choosing a single tag from a set of potential tags. Rather than continue to discuss this process in the abstract, I will now illustrate this discussion of Markov models with reference to two particular tagging algorithms and the approaches they take to selecting a tag for an ambiguous token.

5.3.2.2 Selecting an appropriate tag: additive probabilities in CHAINPROBS

The CLAWS1[26] tagging system, developed at the University of Lancaster in the 1980s, utilises a Markov model in its disambiguation module. This module consists of a program called CHAINPROBS, described by Marshall (1987). In this section, I will discuss the procedure by which CHAINPROBS selects a tag from the set presented to it by the word-analysis module of CLAWS[27]. This process is

[26] The system in general is referred to as CLAWS (Constituent Likelihood Automatic Word-Tagging System). CLAWS1 is the earliest version of this software (see Garside 1987), implemented as a set of programs. The current version, CLAWS4, will be considered below (5.6.3) as an example of a hybrid tagger.

[27] This program is called WORDTAG and performs analysis of lexical lookup followed by the application of an algorithm performing morphological analysis based on a 720-item "suffix list" (Garside 1987: 35-38). A rule-based module called IDIOMTAG is also applied prior to the file being passed to CHAINPROBS (Blackwell 1987).

illustrative in general of how this problem is dealt with by a Markov model.

A Markov model disambiguator such as CHAINPROBS resolves ambiguity in chains of ambiguously tagged words. This contrasts with rule-based methods and the early probabilistic methods of Stolz et al. (1965), where only one word at a time is dealt with. In the case of CHAINPROBS, these "chains" were usually 10 words or less in length, since using the CLAWS tagset[28] punctuation is tagged unambiguously.

There are a number of possible sequences of tags for any chain of ambiguously tagged words. These sequences are often referred to as "paths" through the Markov model. For instance, the phrase "the substance dissolves quickly" might receive the tags "the_A substance_N_V dissolves_N_V quickly_R" from a process of lexical lookup and morphological analysis[29]. Given these potential tags, there are four possible paths: A–N–N–R, A–N–V–R, A–V–N–R, and A–V–V–R. Obviously, the number of possible paths becomes exponentially greater as the length of the chain and the number of tags per word increase. Essentially, CHAINPROBS utilises the Markov model parameters acquired during training to calculate the probability of each possible path and then choose the most probable path.

The Markov model probability of a path, in isolation, is the product of the probabilities of all the transitions that make it up, including the transitions from and to the unambiguous words that begin and end the path. If we assume that P(N|A) = 0.6, P(N|N) = 0.3, and P(R|N) = 0.2, then the probability of the first of the four possible paths given above is 0.036. However, this only represents the probability of the path

[28] See 2.1.2.1 for details of this tagset.

[29] I assume a very basic tagset here: N = noun, V = verb, A = article, R = adverb. While the method described here is that of Marshall (1987), this particular example, including the transition probabilities that follow, has been invented purposefully for this description.

A–N–N–R occurring as opposed to *any* four-tag path beginning in A. We have more information than this: we know that the actual path must be one of the four sequences allowed by the sets of potential tags. Therefore, the next stage undertaken by CHAINPROBS (Marshall 1987: 46-47) is to allocate to each potential path a probability which is equal to its probability in isolation expressed as a proportion of the sum of the probabilities in isolation of all the potential paths. To put it another way:

$$\text{P(path given the potential tags)} = \frac{\text{P(path in isolation)}}{\text{sum of P(path in isolation) for all possible sequences}}$$

It might be assumed that for each word in the ambiguous sequence, the correct tag is the tag for that word which occurs on the most probable path. However, this is not what CHAINPROBS does. Instead, the program does what Marshall (1987: 47-48) describes as assigning each tag an "additive probability" as a "cross-check intended to estimate the likelihood that a given tag selection for a word is correct independently of the correctness of tags selected for surrounding words". The additive probability is equal to the sum of the probabilities in isolation of paths on which that tag appears divided by the sum of the probabilities in isolation of all paths, or:

$$\text{Additive P(Tag A on Word B)} = \frac{\text{sum of P(path in isolation) for all sequences with tag A on word B}}{\text{sum of P(path in isolation) for all possible paths}}$$

The tag actually selected by CHAINPROBS is the one with the highest

additive probability[30], although as Marshall (1987: 48) explains, this is usually the same as the tag on the most probable path, for obvious reasons. This approach to tag selection using a Markov model is characterised by El-Beze and Merialdo (1999: 282) and Merialdo (1994: 157) as *Maximum Likelihood* tagging. El-Beze and Merialdo suggest, however, that this approach is inferior to a contrasting approach using the Viterbi algorithm, which will be discussed in the following section.

5.3.2.3 Selecting an appropriate tag: the Viterbi algorithm in VOLSUNGA

The Viterbi algorithm (see Jelinek 1985: 576-577) is a technique for choosing the most probable path without actually calculating the probability of each path through the ambiguously tagged sequences of words. Merialdo (1994: 157) suggests that the Viterbi algorithm has frequently been used in preference to Maximum Likelihood tagging for three reasons. Firstly, it is easier to implement; secondly it cannot produce in its output sequences of tags which are not possible in the model, which is theoretically possible in Maximum Likelihood tagging; thirdly the Viterbi algorithm gives the best analysis of the sentence as a whole, which Merialdo describes as "linguistically appealing". An example of a system which uses a Viterbi-type technique is the VOLSUNGA system of DeRose (1988)[31]; the taggers described by Jelinek (1985), Merialdo (1994), and Cutting et al. (1992) are other examples.

[30] While the effect of CHAINPROBS disambiguation is to choose a preferred tag, its actual action is to put the tag options for each ambiguous word in order of descending additive probability. This has the advantage that the probabilities of the non-selected tags can be preserved in the output.

[31] De Marcken (1990) discusses some possibly more successful variants on DeRose's algorithm which will not be considered further here.

DeRose's motivation for using a different technique to CLAWS was the inefficiency of CHAINPROBS in terms of computer memory and processing speed (see DeRose 1988: 34).

Unlike CHAINPROBS, VOLSUNGA defines the correct tag as the one appearing on the most probable or "optimal" path, defined simply as the path "whose component collocations multiply out to the highest probability". The technique used by DeRose to identify this path without evaluating the probability of every path is essentially the same as the Viterbi algorithm as described by El-Beze and Merialdo (1999: 280-281) and others, although DeRose does not specify that he is using a Viterbi technique. I shall use DeRose's work to exemplify this algorithm, as his system is early and relatively simple.

VOLSUNGA avoids calculating the probability of every path by discarding at each stage those paths which can be seen to be non-optimal. This is best explained by example[32]. Consider the following sentence:

Words:	V	W	X	Y	Z
Tags:	V1	W1	X1	Y1	Z1
		W2	X2	Y2	
		W3	X3	Y3	

The number of possible paths through this system is 27, each of which would have a probability made up of the product of four transitions. VOLSUNGA does not calculate each of these. Instead, non-optimal paths are dropped as processing progresses through the sentence. The first stage of processing calculates three paths: V1 to W1, V1 to W2, V1 to W3 (these are simply the transition probabilities for those tags). At this stage no path can be discounted. At the next stage, the number of

[32] This example is adapted slightly from DeRose (1988: 35).

possible paths rises to nine. For example, P(V1–W1–X1) = P(V1–W1) × P(W1–X1). However, there is only one optimal path from V1 to each of the three potential tags on X. That is, of the paths V1–W1–X1, V1–W2–X1, V1–W3–X1, one will have the greatest probability, and likewise for the paths leading to X2 and X3. This means that six of the possible paths can be discounted. At the next stage, nine paths are generated again (one from each of the X tags to each of the Y tags), and again six can be discarded leaving only the optimal paths from V1 to Y1, Y2 and Y3. These three paths alone are extended to Z1, which, being unambiguous, allows a single path to be chosen as optimal.

A calculation akin to this is performed by all taggers that utilise the Viterbi algorithm. A key feature of both the Viterbi algorithm and Maximum Likelihood tagging is that the final disambiguation is done across an entire sentence. Thus, even though the initial parameters are based on N-grams with very low *n*, the multiplication of transition probabilities within the Markov model means that, in effect, a great deal of context may be taken into account in disambiguation.

5.3.2.4 The use of lexical probabilities in Markov models

The transition probabilities discussed in the calculations of the optimal path in the two preceding sections constitute one set of parameters of the Markov model, which for each tag state ($t_i \mid t_{i-n+1, \ldots,} t_{i-1}$) where n is most often 2 or 3. The other set of parameters discussed above, lexical probabilities, state P(w|t), that is, the probability of a given tag generating a particular word. The implementation of these probabilities is in theory simple: at each stage, the transition probability is multiplied by the lexical

probability[33] (Charniak et al. 1993: 3). However, this is not necessarily exactly what happens in actual tagging systems. Neither Marshall (1987) nor DeRose (1988) uses lexical probabilities in this straightforward way.

CHAINPROBS uses very approximate human-estimated lexical probabilities to reduce the likelihood of potential sequences that would result in a word being given a tag deemed improbable for that word. This was achieved by the use of "rarity markers" which, if they appear on a potential sequence, reduce the likelihood of that sequence to one half, for rare analyses, and one eighth, for very rare analyses (see Marshall 1987: 46-47 and Atwell 1987: 59). VOLSUNGA by contrast utilises empirically-derived numerical lexical probabilities: DeRose reports that implementing this part of the procedure improved performance from 92-93% to 95-97%. However, DeRose also found that using lexical probabilities[34] "as-is" caused performance to degrade, so VOLSUNGA does not allow them to exceed a given ceiling. Furthermore, Charniak et al. (1993) point out that many Markov model taggers, including VOLSUNGA, actually use P(t|w) rather than P(w|t). The latter is correct in view of the Markov model assumption that words are generated by their tags, although the former appears more intuitive given that, during tagging, we know what the word is but not what its tag is. But based on a comparison of P(w|t) and P(t|w) in otherwise identical disambiguation systems, Charniak et al. (1993) report that using P(w|t) improves results.

[33] It is typical for the lexical probability parameters to be stored in the lexicon in the form of a probability associated with each tag in a word's entry: they are transferred onto the text along with the tags and later utilised by the Markov model disambiguator.

[34] Lexical probabilities are referred to by DeRose as "relative tag probabilities".

5.3.2.5 Acquiring Markov model parameters

The previous sections have discussed the application of the Markov model parameters, which capture the probabilities of a word being associated with a given tag and of one tag following another tag. However, until now little has been said of the training of such models – i.e. the process by which these parameters are estimated in the first place.

Since the parameters represent probabilities, the only realistic way[35] to estimate them is by automated statistical analysis of a large body of data – to wit, a corpus. The majority of recent probabilistic taggers have been able to utilise a corpus already marked up with accurate POS tags. In many cases, the Brown Corpus, originally tagged by a combination of Greene and Rubin's (1971) TAGGIT program and human analysts, has been used for this purpose. Both CLAWS and VOLSUNGA initially based their parameters on the Brown Corpus, as did Church (1988), Charniak et al. (1993), and others. Lexical probabilities can be estimated by creating a list of (some of) the words in the corpus and for each one, listing the tags that occur with that word in the corpus and the frequencies with which they go occur. Transition probabilities can be estimated from the frequencies with which the various sequences of two tags (for a bigram model) or three tags (for a trigram model) occur in the corpus.

However, these basic empirically-derived parameters are frequently insufficient to produce a highly accurate disambiguator. If some linguistically possible (albeit rare) sequence of tags or word-tag combination happens not to occur in the training corpus, the probability parameter for its occurrence would be zero – in

[35] The human-estimated lexical probabilities in CLAWS are an exception here.

the model, it could never occur, although in the language it could. It is often necessary to *smooth* the parameters, i.e. to apply some formula to the parameters which alters them in such a way as to reduce this difficulty. A basic smoothing technique is to add the same small positive value to any entry in a table of tag transition probabilities which contains a zero. Marshall (1987: 54) reports using this technique. More complicated approaches to smoothing are discussed by Church (1988: 141-142), El-Beze and Merialdo (1999: 276-278) and Charniak et al. (1993: 3-4).

A greater difficulty in estimating the parameters arises when the training data is not tagged. This makes it impossible to directly acquire the transition probabilities based on sequences of tags in the training corpus. However, it is possible to use a procedure called the Baum-Welch algorithm (or the Forward-Backward) algorithm, which takes initial estimates of the parameters, which need not necessarily be particularly good, and by applying these probabilities in conjunction with an untagged corpus, computes improved estimates for the parameters it has used. This process then iterates many times (El-Beze and Merialdo 1999: 268-269). This contrasts with training on tagged data, which can be accomplished in a single pass of the corpus. One study which applied the Baum-Welch to tag disambiguation at an early date was that of Jelinek (1985). However, a more recent example of such a tagger is that of Cutting et al. (1992). This tagger is known as the Xerox Tagger after its creators' affiliation and although developed for English, it has been used with other languages (for instance, with Spanish by Sánchez León and Nieto Serrano 1997).

Cutting et al. (1992) use iterations of the Baum-Welch algorithm to acquire estimates for the requisite parameters. However, they also use the idea of *ambiguity classes*[36]. These are classes of words which receive the same set of possible tags

[36] The same notion is referred to as *equivalence classes* by Kupiec (1992).

during the phase of lexical lookup and/or morphological analysis. Within an ambiguity class, words are assumed to have the same behaviour. This allows lexical probabilities to be estimated even for words which occur very infrequently in the training data, although this assumption is clearly invalid given that an ambiguity class such as noun-verb must contain such diverse items as *dog* and *see* (Kupiec 1992: 228).

Training with untagged data has been evaluated by Merialdo (1994), who concludes that a model so trained is not as effective as one trained on tagged data, even if the amount of tagged data is relatively small. However, he also concludes that untagged data can be usefully applied to improving the estimates of a Markov model's parameters acquired from an small amount of tagged data.

5.3.2.6 Some variations within Markov model disambiguation

There are a large number of minor variations between different implementations of the Markov model disambiguation technique[37]. In this section, I will discuss a few of the most significant, including the contrast between bigram and trigram taggers, and the use of Markov models in N-best tagging.

The essential difference between using bigrams and trigrams is that the latter has *S* times as many parameters to be estimated, where *S* is the size of the tagset. This requires more training data, since the average frequency of each trigram is so much

[37] There are also some more significant variations involving models further away from the mainstream of stochastic taggers discussed here. An example is Schütze and Singer (1994), who discuss a variant which they call a *variable memory Markov model*. They report an accuracy of 95.81%, claiming however that "[w]hile the learning algorithm of a VMM is efficient and the resulting tagging algorithm is very simple, the accuracy achieved is rather moderate."

lower than the average frequency of the equivalent bigram; Weischedel et al. (1993: 363) suggest that as a rule of thumb, "the training set needs to be large enough to contain on average ten instances of each type of tag sequence that occurs". However, there does not appear to be any general agreement about whether calculating Markov model path probabilities using trigrams produces a notable improvement over bigrams, or on whether the improvement is worth the necessary extra data.

On the one hand, several studies into part-of-speech disambiguation algorithms have made use of trigrams. Merialdo's (1994) tagger, which he uses to compare training on tagged and untagged text, is a trigram tagger, as is Church's (1988) system. El-Beze and Merialdo (1999: 272) suggest that "trigram models are usually superior to bigram ones" with the proviso that sufficient training data must be available. On the other hand, other studies have questioned the use of trigrams. Weischedel et al. (1993) test a bigram and trigram version of their Markov model tagger called POST, and report that the trigram model has a lower error rate but is slower at processing time. Charniak et al. (1993)suggest that "[e]xperimentation has shown that [the trigram model] offers a slight improvement, but not a great deal." De Marcken (1990) goes so far as to suggest that the trigram approach taken by Church is wasteful of data.

Some systems have actually used both bigrams and trigrams. The CHAINPROBS module of CLAWS is based primarily on bigrams, but utilises some three-tag sequences (Marshall 1987: 54-55; Atwell 1987: 59-60) De Marcken (1990: 245) likewise comes to the conclusion that "it is more efficient to use digrams in general and only mark special cases for trigrams, which would reduce space and learning requirements substantially", in comparison to a full trigram model.

Another variation in the Markov model, implemented by Church (1988), is

that the form of the Viterbi algorithm he uses works backwards from the end of the sentence, rather than forwards from the beginning as is more common[38]. However, according to Charniak et al. (1993), "it is easy to show that this has no effect on results".

The final variant that I will consider is an alteration to the normal Markov model approach which allows more than one tag to be retained or selected by the disambiguator, making the Markov model tagger into an n-best tagger (see page 237 above). De Marcken (1990) describes an implementation which performs n-best tagging, based on a modification of DeRose's (1988) algorithm. Weischedel et al. (1993: 366-368) also implement n-best tagging, using a different method. In both cases, this involves allowing the tagger to return not just the most probable tag but any tag which is probable enough. "Probable enough" is defined by some kind of probability threshold; tags which are more likely than the threshold are retained, all others are removed. This additional parameter, the threshold[39], can be set higher or lower depending on what trade-off between accuracy and recall is required (see de Marcken 1990: 245). As with all n-best taggers, the longer the list of tags returned for each token, the greater the chance that the correct tag is on that list.

[38] That the "forwards" version is more common may possibly be for reasons of psychological plausibility.

[39] CHAINPROBS is another example of a system which utilises a probability threshold (actually a set of thresholds), but it is used in rather a different way. If the probability of the most likely tag is greater than the threshold set for CHAINPROBS, then the other tags are deleted; otherwise, they are retained and the selected tag is indicated by its position at the beginning of the list of tags. See Marshall (1987: 51).

5.3.2.7 The performance of Markov model taggers

It can be difficult to evaluate with any degree of certainty the comparability of success rates in the literature (see also section 5.6). For example, Garside (1987) reports a success rate of 96-97% for CLAWS1, but since CHAINPROBS is not the only disambiguation module in the CLAWS system, the result is not immediately comparable with results cited for other modules discussed in the foregoing section which used a Markov model as their sole disambiguator. In short, the performance of CLAWS is not wholly due to CHAINPROBS (see also 5.6.3 below).

However, in general, the performances reported for the taggers discussed in this section ranges between 95% and 97%. Charniak et al. (1993) summarise the performance of the field. Having pointed out that "[o]ne can get 90% of the tags correct just by picking the most likely tag for each word", i.e. without even using a Markov model, they conclude that "[i]mproving the model to include bigrams of tags increases the accuracy to the 95% level… Improvement beyond this level is possible but it gets much harder." They suggest ultimately that raising accuracy to 97% or beyond may require more lexical information than most taggers use. Reports of Markov model taggers achieving any better accuracy than 97% are unquestionably very rare.

5.4 Approaches utilising corpus-derived rules

It is convenient to characterise the approaches discussed in this section as utilising rules "based on corpus data". But it would not be accurate to assume that researchers working on the rule-based approaches discussed earlier did not use

empirical data to inform their opinions about the grammatical rules they wrote. Klein and Simmons (1963) report that some of the rules used in the CGC system were based on hand analysis of samples of empirical data, and others were derived automatically. Likewise, Greene and Rubin used an automatic process to derive some of the rules used in TAGGIT:

> … a 900-sentence subset of the Brown University Corpus was tagged [ambiguously by lexical lookup], and its ambiguities were resolved manually; then a program was run which produced and sorted all possible context frame rules which would have been necessary to perform this disambiguation automatically.
>
> Greene and Rubin (1971: 32-36)

Similarly, Voutlainen (1999c: 226) specifically points out that using a benchmark corpus to analyse the success of rules under design is of use when compiling rules in the Constraint Grammar framework. However, what distinguishes the approaches discussed below is that they extend the role of the computer in formulating and evaluating the rules so far that the human analyst, and their intuitions about grammar, are excluded from the process. While Eric Brill's transformation-based approach is perhaps today the most widely recognised example of a system using automatically derived rules, it followed an earlier approach based on a parser. I will now discuss the early work of Hindle in this parser-based approach before moving on to consider, in section 5.4.2, Brill's work.

5.4.1 The parser-based approach

Working at a time when the superiority of probabilistic models over rule-

based methods in disambiguation was assumed to have been demonstrated (i.e. before Constraint Grammar), Hindle (1989: 119) opined that "[n]o algorithm, symbolic [i.e. rule-based] or otherwise, will succeed in large scale processing of natural text unless it can acquire some of the needed knowledge from samples of naturally occurring text". Thus, Hindle sought to acquire rules automatically from a corpus of pre-tagged text, in a program that was part of a parser. Again, we see the tendency for taggers based on rules to be associated with parsers[40].

Hindle's program works by going through the tagged training corpus and assigning a tag to every word in turn according to its bank of rules. If the tag thus assigned is incorrect, the algorithm adds a new rule to the bank, such that the new set of rules would have produced the correct tag as given in the training corpus. The performance of the new rule is then monitored by the algorithm. If its general effect is to cause errors rather than prevent them, it would be deactivated.

It should be noted that this is not a disambiguation method, but rather a tag generation method: the tags are *produced* by the application of the rules.

Hindle reports that this relatively simple algorithm produced 35,000 rules[41] by means of 5 iterations through the training corpus. Applying these rules to test texts then achieved a score of 97% accuracy, which was a noticeable improvement on what was then attainable using a rule-based approach.

The embedding of this tagger in a parser may be a disadvantage if one merely

[40] In fact, it should be noted that the system described by Hindle is *not* a tagger, but a parser which performs some tagging as part of the parsing process. The fact that the tagset used was relatively small, with only 46 tags, is perhaps related to its use within a parser.

[41] This may seem a large number compared to the quantities of rules produced by human analysts (typically a few thousand). However, the conceptually possible set of rules is reported by Hindle as being "on the order of 10^9"; thus, 35,000 rules stored represents an immense narrowing of the field.

wanted to tag text without parsing it. It is also a problem in that the feasibility of applying Hindle's approach depended not only on the existence of a training corpus (which many empirical methodologies require) but also on the existence of a suitable parser. However, Hindle makes use of the parser and its analyses to improve the accuracy of the tagging. For example, the rules generated could use in their context statements grammatical relations, rather than just the tag(s) on the previous and following words. For instance, a rule that depended on the co-occurrence in English of the verb *have* and the past participle form of a lexical verb might normally have trouble dealing with phrases such as "has immediately left", where an adverb separates the auxiliary and main verbs. However, since the parser "knew" the underlying structure, Hindle's rules could use the sequence *have* / *past participle* in its tagging rules regardless of intervening adverbs. Another way in which Hindle capitalised on the intertwining of tagger and parser was by using the parser to restrict the rule-acquisition software from devising rules about contexts that were not syntactically linked. The result of this restriction was the creation of fewer rules (about 12,000). However, accuracy with this smaller set of syntactically restricted rules was actually greater.

Thus we see that using the syntactic knowledge in a parser, it was possible for Hindle to perform rule-based tagging with automatically acquired rules to a high degree of accuracy. As described in the next section, Brill's transformation-based approach dispenses with the need to use a parser to acquire rules automatically, and ultimately allows untagged training data to be used, as in the training of a hidden Markov model.

5.4.2 The transformation-based approach

5.4.2.1 Tagging by transformation-based error-driven learning

Brill (1995) calls his approach "transformation-based error-driven learning". Transformations, like constraints, are a type of rule; but whereas constraints specify what analyses should be removed from a list of possibilities (see section 5.2.2 above), transformations change one analysis into another. These transformations are automatically learnt by an algorithm which devises the rules one by one in such a way that at each stage the number of errors left by the system is minimised.

The transformation-based error-driven learning approach can apply to problems other than tagging. Brill (1995) reports applying it to prepositional phrase attachment disambiguation, syntactic parsing, and speech sound generation and recognition. Most notably, he applies it to the separate subtask within tagging of assigning an initial tag or tags to words not recognised by the tagger's lexicon (Brill 1994, 1995: 558-561). However, its earliest application was in the use of contextual information to perform part-of-speech tagging (see Brill 1992). Thus, in the discussion that follows, I will refer exclusively to the method as applied to tagging, although it should be understood that the same approach is applicable in an analogous way to other aspects of automated text annotation.

A key fact about Brill's approach is that it is not a disambiguation technique *per se*. Rather, it is what I shall refer to as an "improvement" technique. It takes text which is unambiguously tagged, but with many errors, as its input, and reduces the number of errors. By contrast, disambiguation techniques proper take ambiguously tagged input and reduce the ambiguity. However, the two techniques perform parallel

functions within their respective tagging systems.

5.4.2.2 The advantages of the transformation-based approach

Brill (1992, 1995) argues for the superiority of the corpus-derived transformation approach over both the approaches to which it is close kin – those using hand-written rules and corpus-derived probabilities. For example, he makes the following comment on rule-based taggers[42]:

> …the rules in rule-based systems are usually difficult to construct and are typically not very robust… [the transformation-based tagger] overcomes the limitations common in rule-based approaches to language processing: it is robust, and the rules are automatically acquired.
>
> Brill (1992: 152)

More developed is Brill's argument for the advantages of his approach over stochastic taggers. Firstly, he suggests (1992: 152) that the rules acquired by his system require less storage space than the tables of statistics learned by taggers using a Markov model[43]. He also suggests that finding and implementing improvements to the system is easier in the case of a rule-based tagger than a stochastic tagger; and that his tagger is more amenable to being used with a different tagset, a different genre of texts, or a different language. For example, he points out (1992: 154) that the idiom-list used in CLAWS[44], which is key to the system's high success rate, would have to

[42] It should be noted that these comments are early and may not take into account the advances made in the 1990s by work in the Constraint Grammar framework (see section 5.2.2 above).

[43] The amount of disk space required by a system was clearly more of an issue in the early 1990s than it is today.

[44] See 5.3.2.3 above and also 5.6.3 below.

be rewritten to use the system with a different tagset or language.

However, the major advantage of corpus-derived rules identified by Brill is that the information acquired by his system is qualitatively different from that acquired by a stochastic tagger:

> Although corpus-based approaches have been successful in many different areas of natural language processing, it is often the case that these methods capture the linguistic information they are modelling indirectly in large opaque tables of statistics. This can make it difficult to analyse, understand and improve the ability of these approaches to model underlying linguistic behaviour.
>
> Brill (1995: 543)

In other words, it is impossible to tell what information about language a stochastic system is getting right and what it is getting wrong. This is because the n-gram probabilities that stochastic systems store are nothing like linguistic knowledge as linguists formulate it. By contrast, the transformations learned in Brill's system benefit from what Brill (1992: 152) refers to as "the perspicuity of a small set of meaningful rules" which the linguist can easily recognise and analyse.

Brill has also (1995: 548-551) compared tagging using lists of transformations favourably to the use of decision trees, a machine-learning technique which, Brill reports, can in some cases provide the same sets of classifications as transformations derived by Brill's method (see also section 5.5 below).

5.4.2.3 A summary of Brill's algorithm

I will now proceed to explain briefly the operation of Brill's learning system

(shown in the diagram below)[45]. Like Markov model taggers, Brill's tagger requires pre-tagged text to learn from, but this pre-tagged corpus is used as a reference point for evaluating the performance of the rules rather than as a source for statistics. The system starts with an untagged version of the corpus, which is run through an *initial state annotator*. This is any program which assigns a single tag, accurate or not, to each word. Brill reports success using an initial state annotator which simply assigns the tag which is most frequent for a given word, as stored in a lexicon. However, the "initial state" can also be the output from another system (such as a stochastic tagger – this allows different taggers to be chained together). The tagged corpus (the "truth") is compared by the learner program to this initially-tagged corpus, and "[a]n ordered list of transformations is learned that can be applied to the output of the initial-state annotator to make it better resemble the *truth*" (Brill 1995: 545).

[45] For a full description, see Brill (1995). Brill (1992) describes a slightly earlier version of the same technology, with particular attention to its application to part-of-speech tagging. Brill (1992) uses slightly different terminology to Brill (1995), calling the rules used in the system "patches" rather than "transformations". In this summary I follow the usage of Brill (1995).

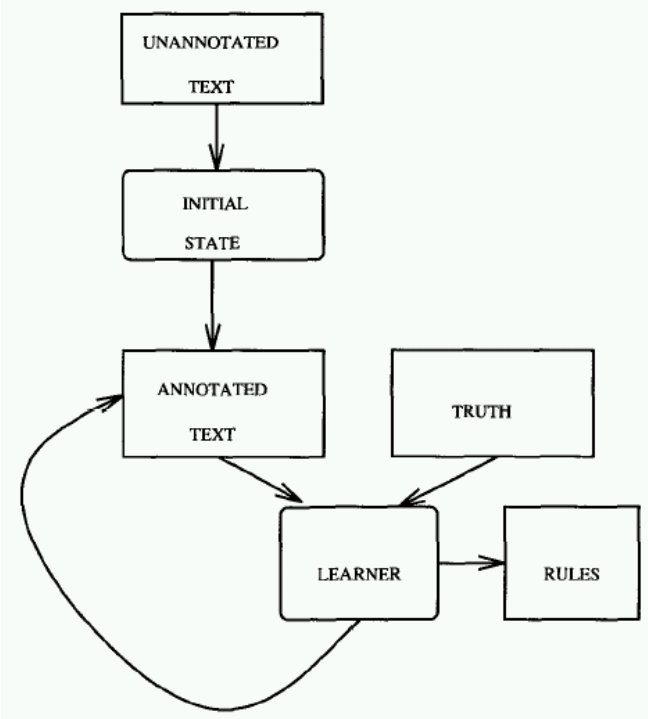

Fig. 5.3: Brill's learning system (taken from Brill 1995: 545)

One transformation is learned each time the algorithm cycles through the learner. As Brill (1995: 553) puts it, "[t]o learn a transformation, the learner, in essence, tries out every transformation, and counts the number of tagging errors after each one is applied." The measure of success of a transformation is the reduction of errors in the annotated corpus as compared to the truth. On each cycle, the transformation with the best score is retained, applied to the training corpus, and added to the bottom of the list of transformations which have been learnt so far. Thus, the list of transformations is ordered from the most effective (at the top) to the least effective (at the bottom). Learning ends when no transformation can be found which improves the tagging of the corpus (i.e. corrects more errors than it creates).

Brill makes the point that "transformations towards the end of the list contribute very little to accuracy" (1995: 557). In one experiment he reports, 447 transformations were learned, giving an accuracy of 97.2%. However, the first 200 transformations on the list applied separately gave an accuracy of 97%, and the first

100 rules gave 96.8%.

The transformations learnt can be very similar to the linguistic information inserted into rule-based taggers or modules of taggers. For instance, Brill (1992: 154) notes that his tagger automatically acquired a rule to tag the phrases like *as old as* (qualifier – adjective – subordinating conjunction). This rule was very similar to an entry in the CLAWS idiom-list (effectively a rule-based module).

The implementation of the tagger itself, as opposed to the learner, is very simple: first, untagged text is passed through the initial-state annotator, and then the learned transformations are applied to the text in the same order that they were learned. Tagging is thus effectively the same as learning[46], except that there is no need to evaluate the transformations since during learning they have established their effectiveness.

5.4.2.4 The form of transformations

Brill's transformations are made up of a rewrite rule and a triggering environment. In the case of part-of-speech tagging, this means that transformations can be paraphrased as *change tag A to tag B if tag A occurs in context C*. An example given by Brill (1995: 545) of this is *Change MODAL to NOUN if the preceding word is DETERMINER*. This would, for instance, correct the tagging of a phrase such as "The can rusted" where the initial state annotator has tagged "can" as a modal verb.

[46] Roche and Schabes (1995) suggest that Brill's algorithm is computationally slower than is desirable for a tagger. However, they report a method to convert a set of rules acquired by Brill's method into a finite state transducer which can apply all the rules at once. This implementation is much faster than Brill's tagger, although it uses more memory, but is functionally identical to it.

In the learning process, the transformation learner has available *templates* for possible contexts with which to create transformations. At the point in the learning algorithm described above where the learner must try out every transformation, the set of possible transformations is equal to the set of "all possible instantiations of transformation templates" (Brill 1995: 553).

In the form of the tagger which Brill (1995) refers to as "nonlexicalised" – basically identical to the tagger presented in Brill (1992) – transformation context templates refer only to the tags on the three words preceding and the three words following the target word. Extensions described in Brill (1994) add further rule templates, which may now refer to the actual words as well as their tags. Brill (1995) refers to this as the "lexicalised" form of the tagger[47].

Some examples of the context templates, rendered as prose, are as follows:

The preceding (following) word is tagged z.
One of the three preceding (following) words is tagged z.
The preceding (following) word is tagged z and the word two before (after) is tagged w.
The preceding (following) word is w.
The word two before (after)is w.
The current word is w and the preceding (following) word is tagged z.
The current word is w, the preceding (following) word is w2 and the preceding (following) tag is t.
(all examples from Brill 1995: 553, 556)

Obviously, in other applications of transformation-based error-driven learning, the templates are different: for instance, the unknown-word initial-state tagger[48] uses templates referring to the characters that make up the target word.

[47] It should be noted that Brill (1995: 557) finds that "the addition of lexicalised transformations [to the tagger] did not result in a much greater improvement in performance" – in his experiment, the error rate only fell slightly, improving accuracy from 97% to 97.2%.

[48] See 5.4.2.1.

Within the tagging system, the transformations are expressed in a linear format which lists sequentially the tag to be changed, what it is to be changed to, the context template to apply and the parameter of that the context takes. So the transformation mapping the tag TO (infinitive marker *to*) to IN[49] (preposition) when the word preceded an article is realised as TO IN NEXT-TAG AT, and the transformation changing a verb tag to a noun one or two words after an article is realised as VB NN PREV-1-OR-2-TAG AT (Brill 1992: 153). The relation of these transformations to the generalisations that a linguist might make on the basis of their own knowledge is clear.

The nature of the templates is not necessarily crucial to the performance of the tagger. Brill (1992: 154) points out that the presence of bad templates will have no impact as long as effective templates – such as the very broad ones cited above – are retained as well: "If a template is bad, then no rules which are instantiations of that template will appear in the final list of [transformations] learned by the tagger." In effect, the templates are suggestions by the programmer as to what type of rules the learner might like to come up with; the algorithm is free to ignore these hints if they are unhelpful.

5.4.2.5 Extensions to Brill's basic method

Brill's tagger as initially designed invariably returns a single tag for each word, and requires a tagged corpus to use as training data. However, later developments in the transformation-based approach mean that both these features can be avoided if so desired.

[49] Brill's tagger uses the Brown Corpus tagset (see 2.1.1).

5.4.2.5.1 N-best tagging in the transformation-based approach

As explained above (page 237), n-best or k-best taggers return more than one tag for each word, trading off improved recall for lower precision. At first sight, this would appear incompatible with the transformation-based approach, which takes fully disambiguated text as its input. However, Brill (1995: 561-562) discusses a method to make a transformation-based tagger return n-best tags, which in effect re-inserts ambiguity after the non-ambiguous stage of improvement using transformations.

This is achieved by using the tagger described above as the initial-state annotator for a second system. This latter system learns transformations based on a different set of templates: the same contexts are used but the operation triggered adds an alternative tag to a word rather than changing a single tag. Transformations are rated on the value obtained by dividing the number of errors they correct (the improvement in recall) by the number of tags added altogether (the fall in precision). That is, at any point in training, the best rule is the one that corrects the most errors for the smallest cost in terms of extra ambiguity. Brill reports that when 250 rules were learned, recall rose to 99.1%, with a mean 1.5 tags per word (with fewer rules, recall was lower but precision was higher).

5.4.2.5.2 Unsupervised training of a transformation-based model

Brill and Pop (1999) describe a method to acquire a set of transformations for

tagging without the use of any tagged training data[50]. This approach is referred to as unsupervised learning, as opposed to the supervised learning discussed above[51]. The only linguistic input required is a lexicon listing the allowable tags for each word. This lexicon need not be generated from a corpus (which generation would itself necessitate a tagged corpus) but "could be extracted from an on-line dictionary or through morphological and distributional analysis." Every word in the text is given all these tags as their initial state annotation, and a set of transformations is learned as before.

However, because the initial annotation is ambiguous, the transformations reduce the number of possible tags rather than changing one tag to another. This means that the unsupervised tagger uses a disambiguation process, like the rule-based approaches discussed in section 5.2, rather than an improvement process like Brill's earlier tagger. These transformations change the tag from X to Y in context C, where X is a set of tags rather than single tags and Y is a single tag which is one of the tags in X (Brill and Pop 1999: 32). The context templates in this version of the tagger are more restricted than in the supervised tagger, referring only to the previous word, the following word, or the tag on either.

It is clearly trivial to generate transformations according to these templates. But evaluating the transformations in the absence of a tagged reference corpus is more challenging. Brill and Pop's approach is to use as a reference of correctness those

[50] Since rules based on a linguist's knowledge do not require a tagged corpus, and hidden Markov models can be trained on untagged data, this extension to Brill's model means that all three approaches discussed so far can be undertaken with or without pre-tagged data. Contrast neural networks and other machine-learning approaches as discussed in section 5.5 below, which do require tagged data.

[51] The "supervision" by the human user of the system consists of making the initial markup of the training data: the tagger's use of this data constitutes operating under human supervision.

words in the untagged training corpus that were unambiguously tagged by the initial state annotation or have been disambiguated by previously learned transformations. A transformation which selects a tag in a given context is evaluated on the basis of how often that tag appears on unambiguously tagged words in that context. The function that measures this compares the frequency on unambiguous words of the selected tags to that of the tags in the ambiguity set that are not selected:

> A good transformation for removing the part of speech ambiguity of a word is one for which one of the possible tags appears much more frequently as measured by unambiguously tagged words than all others in the context, after adjusting for differences in relative frequency between the different tags.
>
> Brill and Pop (1999: 33)

Where the lexicon used contained all the words in the test texts[52], the unsupervised-learning tagger achieved 95 to 96% accuracy based on 120-350 thousand words of data. A strength of this model is that the learner did not over-train itself. This is a danger with the training of stochastic models on untagged text using the Baum-Welch algorithm (Brill and Pop 1999: 34).

As well as the unsupervised model, Brill and Pop discuss a "weakly-supervised" model, using a small amount of tagged text and a large amount of untagged text as training data. Two sets of transformations are learnt. When tagging, those based on unsupervised learning are applied after the initial state annotator but before the supervised transformations. This combined approach achieved 96.8% accuracy with 88,200 words of tagged data – an improvement on what could be

[52] Unknown words would be dealt with in this version of the model by assigning them all open-class tags.

attained if this amount of tagged data were used without any untagged data.

5.5 General machine learning approaches to disambiguation

The previous section discussed Brill's transformation-based error-driven learning. This is a machine-learning algorithm designed with part-of-speech tagging and similar applications in mind. However, other, more generally applicable methods of machine learning, not necessarily originally designed for linguistic applications, have also been utilised in performing part-of-speech disambiguation.

The information which is learned from the training data is stored in the form of numerical parameters: thus in terms of the typology outlined in 5.1 above, machine learning approaches fall into the same category as Markov model disambiguation algorithms. However, the model which is constructed is typically very different from the matrices of transition probabilities that form the basis of Markov modelling (Daelemans 1999: 303).

There exists a wide variety of machine-learning techniques. Those which have been utilised in part-of-speech tagging include case-based learning, decision tree induction and neural networks (an overview of all of these is given by Daelemans 1999). Given the diversity of these models, I will not attempt to discuss them all in depth. Rather, I will briefly discuss the overall principles of machine learning, and then exemplify work in this field by looking in detail at work on part-of-speech tagging using neural networks.

5.5.1 Overview

The underlying rationale of all machine learning is that the system's capacity to perform a given task is derived from examination of a set of examples of inputs to the system and the corresponding outputs that the system would optimally produce. As Daelemans explains:

> Conceptually, a learning system consists of a *performance component* which achieves a specific task (given an input, it produces an output) and a *learning component* which modifies the performance component on the basis of its experience in such a way that performance of the system… improves… To achieve its task, the performance component uses an internal representation. The task of the learning component may therefore be construed as a search in the space of possible representations for a representation that is optimal for performing the mapping.
>
> Daelemans (1999: 287)

In case-based learning, cases encountered in the training data are stored in memory and new situations in the input are handled on the basis of the most similar remembered situation. In part-of-speech tagging, the case base might consist of words, with associated context and the correct tags. Much depends on the weightings given to different features. An example of case-based learning applied to part-of-speech assignation is the Kenmore system of Cardie (1996), which performs both part-of-speech[53] and semantic tagging; another example is the MBT system described by Daelemans et al. (1996). Decision tree induction, by contrast, does not retain any of the training data. Rather, a tree is derived automatically from the data, with output

[53] Cardie's system only distinguishes eighteen parts of speech; thus it may not be wholly comparable to other modern taggers which handle dozens or hundreds of morphosyntactic categories.

tags on its leaves and nodes containing tests that choose the tag. For example, a node might ask "is the word *the*?", leading either to a leaf with an article tag or to further nodes, depending on the answer (example from Magerman 1995[54]). An example of a system using decision trees is that of Black et al. (1992), who report modest improvements on the accuracy achieved by Markov model taggers. A problem in both case-based learning and decision tree induction is *feature relevance* (Daelemans 1999: 293-298): how does the system measure the similarity of one situation to another, or decide what features of the word to ask questions about?

I will now proceed to discuss in greater detail part-of-speech disambiguation using neural networks.

5.5.2 The application of neural networks in disambiguation

A neural network[55] is a learning system whose architecture consists of two or more interconnected layers of processing units, as shown in the diagram below. Each unit may be activated or not activated. The activations of the bottom layer correspond to the input to the system, and the activations of the top layer indicate the output. The optional intermediate layers are called "hidden" because while they contribute to the processing, they do not connect to the world outside the system at all. The activation of each unit propagates to other units via that links connecting them. The parameters of the system are the weights given to the links, and the activation values of the units.

[54] Reported in Daelemans (1999).

[55] The type of neural network that has been used in disambiguation is the *multilayer perceptron network* (Schmid 1994: 2, Daelemans 1999:). Therefore, I will not discuss other types of network and references to neural networks in this section refer solely to multilayer perceptron networks.

Thus, the activation of non-input units depends upon the activations of the units to which they are connected and the weightings of the links connecting them.

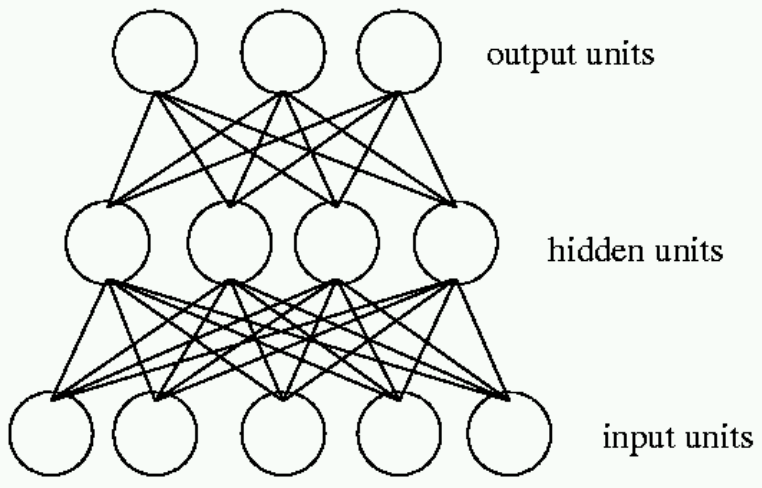

Fig. 5.4: A 3-layer neural network (figure from Schmid 1994)

The training of a neural network consists of iteratively adjusting the weights of the connections and the activation values to produce a system which produces the correct output for the training data:

> During learning, the set of examples is repeatedly examined to find an optimal set of connection weights between layers of simple units. The training material is thus abstracted into a set of numeric weights which is then used to predict the output of new input patterns.
>
> Daelemans (1999: 300)

Prior to their application to part-of-speech disambiguation, neural networks had been successfully used in speech recognition (reported by Schmid 1994). Their use in tagging dates mainly from the late 1980s and the work of Benello, Mackie and Anderson (1989)[56]; subsequent systems using neural networks include Nakamura et

[56] Benello, Mackie and Anderson (1989) report that earlier linguistic neural network systems dealt with made-up data and handled unrestricted data poorly.

al.'s (1990) NETgram system[57] and the Net-Tagger system of Schmid (1994).

The input to a neural network disambiguation system is the ambiguously tagged word, and some amount of context. For each word examined by the network, there is a set of units in the input layer equal in number to the number of tags in the tagset. The input units corresponding to the tag or tags marked up on the word examined are activated. The output layer then indicates the tag the system has chosen. The amount of context used varies greatly between different systems. Benello, Mackie and Anderson (1989) used as their input the four unambiguously tagged words preceding the target word and one ambiguously tagged word following it. Nakamura et al.'s (1990) system allows the scope of the system to be varied, taking one, two or three words before the target word as the context, but *no* words after it. Schmid (1994) also experimented with different context scopes, reporting that three words before the target word and two words afterwards were optimal (to wit, using more context did not improve the system, using less only worsened it slightly).

It can thus be seen that in general, neural networks use a wider context than a Markov model can, based on bigram or trigram transition probabilities. This is because the number of parameters that must be estimated for a Markov model is equal to the size of the tagset raised to the power of the length of the sequences examined. Thus, the amount of training data needed to accurately estimate (say) six-gram[58] probabilities for a tagset of (say) 100 – giving one trillion transition probabilities –

[57] As Daelemans (1999) points out, the NETgram system generates or predicts the part-of-speech of the target word based on the preceding parts-of-speech, and is therefore not a disambiguator *per se*: it does not take a set of potential analyses of the target word as part of its input, as the other two systems described in this section.

[58] Six words is the size of the window taken into account by Benello, Mackie and Anderson's (1989) neural network system.

would be astronomical. For a neural network, the number of parameters is much smaller, since they are what Benello, Mackie and Anderson refer to as "complex conditional probabilities" rather than "simple first-order transition statistics". Therefore wider contexts can realistically be taken into account. However, neural network taggers still use surface-level linguistic information only. They do not use any knowledge of syntactic structure. In this they resemble nearly all the disambiguation methodologies discussed in this chapter.

There is also a great deal of diversity among different systems concerning the structure of the network used. Benello, Mackie and Anderson's (1989) network has 236 input nodes, 236 hidden nodes in a single hidden layer, and 88 output units to represent the output. Nakamura et al.'s NETgram has a variable structure to match its variable context scope, but the basic bigram model has 89 input units, 89 output units, and 16 on each of its two hidden layers. By contrast, Schmid (1994) does not use a hidden layer at all, on the grounds that while these make networks more powerful they can also make them prone to over-training. He reports that the Net-Tagger does not, in any case, benefit from having a hidden layer. Thus, his system consists solely of a number of output units equal to the size of the tagset, and an equal number of input units for each word in the context scope. A key feature of these systems is that the parts-of-speech marked on preceding words in the input to the net are themselves the output of an application of the network.

The performance of neural network taggers[59] appears comparable to that achieved by rule-based and stochastic approaches[60]. Benello, Mackie and Anderson

[59] In the context of neural networks, "performance" is sometimes referred to as *generalisation ability* – i.e. how well the network can "generalise" to handle data that was not in the training set.

[60] A exception may be Nakamura et al., who report only 86.9% accuracy.

report an accuracy of 94.7%. Schmid's Net-Tagger achieves 96.22%, which, Schmid reports, is better than a comparable Markov model tagger that he tested[61]. It should be noted however that the amount of training data required to achieve these results varies. Benello, Mackie and Anderson used a maximum of about 15,500 words; by contrast Schmid trained his model on some 2 million words. It is therefore not certain that these results are entirely comparable.

A question on which there is so far no consensus is the extent to which neural networks and the knowledge within them can be described as "linguistic". Benello, Mackie and Anderson (1989) suggest that neural networks are probabilistic models just like Markov models, and not linguistic in the same sense as rule-based models like Constraint Grammar or transformation-based learning: "if pressed, one could perhaps say the network uses 'fuzzy' reasoning, but it is unlikely that such a thing as a specific linguistic rule could be said to exist in the network." By contrast, Nakamura et al. (1990) report that "[t]he results of analysing the hidden layer after training showed that the word categories were classified into some linguistically significant groups, that is to say, the NETgram learns a linguistic structure."

5.6 Combining and comparing disambiguation methods

In order to choose a disambiguation method to apply to the problem at hand – the tagging of Urdu – it would be helpful to have a reliable means of comparing the different approaches discussed in the four previous sections. However, this is far from

[61] Schmid (1994: 7) does report that Net-Tagger runs much more slowly than the stochastic taggers to which he compared it; however, in this age of continually faster computer hardware this is unlikely to mitigate strongly against the use of neural networks in tagging.

a trivial task; Abney (1997) offers the caveat that "[a]s concerns accuracy figures… it is good to remember the maxim, 'there are lies, damned lies, and statistics'." In this section, I will firstly look at some of the factors that make the comparison of disambiguation algorithms difficult in the first place (5.6.1), and continue by reviewing a study that has attempted to get past these difficulties (5.6.2). Finally, I will review some studies that have taken the further step of attempting to combine different approaches to disambiguation into a single tagging system (5.6.3).

5.6.1 The difficulty of comparing different taggers

There are two fundamental difficulties in comparing the success rates of different disambiguation methods. Firstly, there is no universally agreed system for rating the performance of a tagger. Secondly, it is impossible to be certain that any differences found between two taggers that use different disambiguation methods are actually due to the difference in disambiguation method.

Probably the most widely used system of assessing the performance of a tagger is to look at the percentage of tokens which are tagged correctly. This system is used by Charniak et al. (1993) as the basis for their comparison of different Markov model equations, for instance. This measure of performance is often called "accuracy" or sometimes "correctness" (e.g. by van Halteren 1999b) in contrast to another common system which uses the two measures of "recall" and "precision", as defined by Voutilainen (1995: 172). The precision is the number of tags correctly assigned, expressed as a percentage of all the tags assigned. The recall is the percentage of

tokens which are given the correct tag[62]. Clearly, accuracy is a measure geared to taggers that produce non-ambiguous output, whereas precision/recall is geared to n-best taggers. However, van Halteren (1999b: 82) suggests that accuracy can be used to measure the performance of an n-best tagger if accompanied by the additional measure of ambiguity (equal to the number of tags divided by the number of tokens, and thus, ideally, 1)[63].

It should not be supposed that accuracy/ambiguity are in practice the same as precision/recall. For example, the definition of precision is such that it can never be higher than recall – in fact, the two are the same, and identical to accuracy, if applied to a tagger which assigns one tag only to each token (Voutilainen 1995: 172). In contrast, the ambiguity of such a tagger would be 1 – the optimal score. Therefore, one problem in comparing the published success rates of different taggers is that the figures describing those success rates are in many cases not directly comparable.

The second, greater difficulty in comparison is that factors other than the use of a different disambiguation method might account for differences in performance between two taggers. Taggers frequently use tagsets of different sizes. One tagset may contain distinctions that are difficult to make[64] (thus degrading the tagger's

[62] A slightly different definition, which however amounts to the same thing mathematically, is given by van Halteren (1999b: 82).

[63] It should be noted that despite the prevalence of performance measures based on counts of tags or tokens, it has been suggested (e.g. by van Haltern 1999b: 85 and Abney 1997: 121; see also Merialdo 1994: 156) that the sentence may in fact be a more relevant unit of error. In many applications that might use tagged text – for instance, parsing, or machine translation – a single wrongly tagged word could disrupt the analysis of an entire sentence.

[64] An example of such a distinction given by Abney (1997: 121) is the distinction between gerunds and present participles in English.

performance) which another does not. Even tagsets of the same size may very frequently encode different sets of distinctions. A notion of the impact of the tagset may be gleaned from the results of the AMALGAM project (Atwell et al. 2000), which made a comparative evaluation of different tagsets for English. Towards this end a single tagger (Brill's) was retrained to tag each of six tagsets. The accuracy rates achieved varied from 97.43% to 90.59%. Clearly the tagset is a very significant factor in determining tagger accuracy.

Taggers also use different lexicons and different morphological analysers to provide the initial set of tags to be disambiguated. Taggers are also trained on different datasets[65] (if training is required) and tested and utilised on different data. Together with the variations between tagsets, these factors means that the disambiguation systems in different taggers are dealing with differing amounts of ambiguity in input texts of differing types – so to treat the tagger score (whether accuracy or precision/recall) as an assessment of the efficacy of the disambiguation methodology that it uses is, at the very least, suspect[66].

In summary, as Abney (1997: 121) points out, "a fair comparison of [tagging] techniques is only possible if they are applied to the same task. In this respect, virtually none of the reported tagger error rates are comparable."

To this it might be added that a fair comparison of two disambiguation methodologies would necessarily require that the taggers began with the same

[65] McEnery et al. (1997: 221) give some experimental evidence that the size and, possibly, composition (in terms of text-type) of the training data has an effect on tagger performance. Merialdo (1994: 161) also reports results indicating that size of the training dataset has an impact.

[66] Another factor is that many contemporary taggers are constantly being improved over time. If this is the case, the reported performance of a system at the time of its development may be inferior to the performance it is in practice capable of achieving at the current moment.

investment into the inputting of linguistic information by a human being (e.g. in the form of rules or manually tagged text). This factor may be critical, because the time of an analyst is liable to be the most expensive requirement of a tagging system. A tagger which can perform 95% accurate tagging based on analyses that took a researcher one week might well be considered superior, for practical purposes, to one which performs 99% accurate tagging based on a year's worth of analyst time. This is particularly applicable when considering the suitability of a disambiguation methodology for a language which has not yet been tagged[67].

For these reasons, it is highly problematic, if not impossible, to make a meaningful comparison of the performance of different disambiguation methodologies based solely on their success rates as reported in the literature, as any difference in these rates cannot be attributed with certainty to the difference in disambiguation method alone.

5.6.2 Enabling a meaningful comparison of disambiguation methodologies

In spite of the difficulties outlined in the previous section, efforts have been made to make meaningful the comparison of taggers using significantly different disambiguation methods. A good example of such a study is that of Chanod and Tapanainen (1995a), who compared the performance on French text of a Constraint Grammar tagger (see section 5.2.2 above) with that of the Xerox tagger of Cutting et al. (1992) (see section 5.3.2.5 above). Rather than being a straight "try-out" of pre-

[67] Analyst time is not so precious a commodity in the case of languages like English, where vast quantities of text analysed either by hand or by previously developed taggers are available freely or at minimal expense.

existing tagger programs, this study took each tagger in its untrained state – i.e., with no established lists of constraints or tag transition probabilities.

Because a new implementation of the taggers was being developed, the same tagset, tokeniser, lexicon and morphological analyser[68] could be used in both taggers – thus the comparison is of the disambiguation method alone. Furthermore, Chanod and Tapanainen allocated one month of research time to the development of each of the two taggers. Thus, they evade a number of the difficulties discussed above with regards to making a fair comparison.

Since the Xerox tagger does not utilise tagged training data, the research time on this tagger was used to fine-tune the tagger by writing "biases" – initial values for the transition probabilities before training occurs. By contrast, for the CG system the time was spent writing and testing constraints, of which there were ultimately 75.

Chanod and Tapanainen found that the constraint-based system performed better. In the more successful of two tests of these systems, the CG system was 98.7% accurate (in this case, no ambiguity remained in the analysis) and the probabilistic system was 96.8% accurate. A combination of the systems performed less well than the constraint-based system alone (see also 5.6.3 below on combination tagging systems). Furthermore, Chanod and Tapanainen point out that to write good biases for the probabilistic tagger is very difficult as "it is hard to predict the effect of tuning the parameters of the system, whereas the constraint-based tagger is very straightforward to correct." Thus Chanod and Tapanainen conclude that the constraint-based rules perform better than the Markov model disambiguation system, contain more linguistic information, and are more easily written.

[68] See Chanod and Tapanainen (1995b).

5.6.3 Combining different tagging methodologies: hybrid taggers

With respect to the typology outlined at the start of this chapter, a "hybrid tagger"[69] may be defined as a system which incorporates two or more disambiguation modules based on two or more of the methodologies outlined in the preceding sections. Each disambiguation module may itself be classified as a single type, but the system as a whole cannot.

The creation of a hybrid tagger may be a matter of expediency to improve the functioning of what would otherwise be a single-type system. This is the case, for example, in Stolz et al.'s (1965) system, which employs an "ad hoc" phase (essentially, a rule-based module) to improve the accuracy of a basically probabilistic system. A tagger which might be classified as a hybrid in another sense is that of Brill (1992, 1995), since Brill's methodology requires the use of an initial state annotator which is frequently a probabilistic disambiguation module[70].

However, a more prototypical example of a hybrid system would be the CLAWS system. As explained in 5.3.2.2 above, the core of the CLAWS system is a Markov model disambiguation module. However, even CLAWS1, the earliest version of the system, incorporated elements which in terms of my typology are unquestionably rule-based. So although CLAWS has usually been described in the

[69] I have adopted the term "hybrid tagger" from Garside, Leech and McEnery (1997); however, I will use it throughout this chapter in accordance with the definition given here.

[70] Brill (1995: 552) reports a version of his tagger whose initial state annotator allots to each token the tag that occurs with it most frequently. This initial state annotator can be viewed as a probabilistic disambiguator which utilises lexical probabilities only and not transition probabilities (see section 5.3.2.4). However, Brill has also (1995: 545) implemented a version where the initial state annotator was a full Markov model tagger; this implementation of Brill's algorithm is an even clearer hybrid.

literature as a stochastic tagger (e.g. by Voutilainen 1995: 269), in the terms of the typology used in this chapter, it was from its inception a hybrid tagger.

The rule-based component of CLAWS1 is implemented as a program called IDIOMTAG, which alters the tagging after the WORDTAG lexical analysis component has initially assigned potential tags, but before the text is passed to the stochastic disambiguator (Garside 1987: 39-40). The initial motivation for this program was the inability of the Markov model in CHAINPROBS to deal with multi-token words and fixed idioms:

> IDIOMTAG, as its name suggests, searches the text for specific sequences of words, or words and tags (but not tags alone), whose syntactic role in combination differs from the syntactic role played by the same words in other contexts… In some cases the correct (i.e. "idiomatic") tag is among the choices offered by WORDTAG, and the role of IDIOMTAG is merely to facilitate the process of disambiguation and to forestall possible errors by CHAINPROBS… In other cases, IDIOMTAG inserts tags which were never considered by WORDTAG, because they apply exclusively in the context of one or more idioms.
>
> Blackwell (1987: 111-112)

Blackwell gives the tagging of "first" as an ordinal number in the phrase "at first sight" as an example of the former type of action by IDIOMTAG, and the tagging of "to" as an adverb in the phrase "to and fro" as an example of the latter type. These idioms are stored in a list. It is possible to conceive of the entries in this list as the equivalent of rules with a very particular context. Certainly, it would be possible in the majority of rule-based disambiguation methodologies (including the formalism I describe in the following chapter) to write a rule that would have the same effect on the text as any given CLAWS idiom. In the case of "to and fro", a rule

would have to state that all tags on a word should be removed and replaced with an adverb tag *if* that word was "to", the subsequent word was "and", and the word after that was "fro". Since the application of these rules results in reduced ambiguity, IDIOMTAG can be classified as a rule-based disambiguation module which is linked in serial to a stochastic disambiguation module (CHAINPROBS).

It should also be noted that the current version of CLAWS, CLAWS4, uses additional rule-based elements as well as the idiom-tagger, and the idiom-tagger itself has grown in scope (see Garside and Smith 1997). In CLAWS4, the idiom-tagger and other non-stochastic modules can correct the output from probabilistic processing as well as altering its input. Thus CLAWS4 is a hybrid to an even greater extent than CLAWS1.

The rule-based and stochastic elements of CLAWS developed together, within the context of the CLAWS system. By contrast, Tapanainen and Voutilainen (1994) describe an experiment which sought to combine rule-based and stochastic disambiguation modules which were developed completely separately, by different researchers, and which were originally intended to function as the sole disambiguation routine within their respective systems. The two systems were the EngCG rule-based tagger and the Xerox Markov model tagger (see sections 5.2.2 and 5.3.2.5 above respectively). These two taggers have complementary strengths: EngCG is rarely wrong but does not disambiguate fully, whereas the Xerox tagger always disambiguates fully but is less reliable in terms of accuracy. Since these are independent systems, combining them poses some problems. For example, the EngCG tagger uses a tagset effectively twice as large as the one used by the Xerox tagger. Rather than attempt to link the two systems in serial as elements of a single

system[71], Tapanainen and Voutilainen run them in parallel on the same text and then align the outputs by constructing a mapping between the tagsets and allowing the Xerox tagger to resolve the ambiguities left by the EngCG system. They report an accuracy rate of 98.5%, a result that they describe as "better than other state-of-the-art part-of-speech disambiguators".

5.7 Selecting an approach for disambiguation in Urdu texts

The choice of disambiguation methodologies available for the Urdu tagger can, in the light of what has thus far been discussed in this chapter, be summarised as follows:

- Rule-based disambiguation;
- Probabilistic disambiguation using a Markov model trained on tagged data;
- Probabilistic disambiguation using a Markov model trained on untagged data;
- Tagging using transformation-based error-driven learning[72];
- Disambiguation using some machine-learning technique, e.g. neural networks;
- Disambiguation using some hybrid of the above approaches.

I will now outline and justify my decision to use a rule-based methodology for tag disambiguation in the Urdu tagger on two types of desiderata. In 5.7.2 I will

[71] Note that this is the approach taken by the CLAWS modules, and also (unsuccessfully) by the combination system of Chanod and Tapanainen (1995a) – see also section 5.6.2 above.

[72] As discussed earlier (page 267), Brill's transformation-based technique does not, strictly speaking, perform disambiguation. However, it fulfils the same function.

discuss factors influencing the decision that are specific to this application. Prior to that, in the following section, I will discuss more general factors which mitigate in favour of or against one or more of the methodologies listed above.

5.7.1 General factors

It is first of all possible from general considerations to exclude from consideration a disambiguation methodology based on general machine learning techniques. Daelemans (1999: 303-304) suggests that these methods have several advantages over statistical methods – including their requiring less training data, having fewer parameters, taking more context into account, and often training faster. However, these potential advantages are outweighed by another factor highlighted by Daelemans: they are a very new technology whose effectiveness has yet to be fully evaluated.

> With the availability of only a relatively small body of empirical data and theoretical analysis on the applicability of inductive machine learning techniques to tagging, it is too early for strong conclusions… Compared to the well-developed theoretical and empirical foundations of statistical approaches to tagging, the machine learning approach to this problem has only just started.
>
> Daelemans (1999: 303-304)

It would lie beyond the scope of this thesis to attempt to evaluate and compare the efficacy of the great variety of machine learning techniques[73]. For this reason, and

[73] Not only are there a number of different techniques, for instance decision trees versus neural networks, but there are also variations within individual approaches. For instance even within the neural network technique, several different types of network have been employed (see 5.5.2).

since the study of this type of tag disambiguation is not as well developed as other approaches, it would seem unwise to utilise machine learning techniques in the tagging of Urdu.

It is not possible to exclude any of the other methodologies on the general grounds of their own strength or weakness. As outlined in section 5.6.1 above, comparison of different tagging methodologies is an inherently difficult task, as differences in performance reported for different taggers cannot be reliably attributed to the different disambiguation methodologies used by those taggers.

In any case, the performance rates that are reported are frequently very close to one another. Markov model taggers generally achieve an accuracy of up to 97% (see section 5.3.2.7); Brill (1995) reports a very similar performance using a transformation-based approach; and Voutilainen (1995: 186-187) reports a performance using the rule-based Constraint Grammar methodology of 99.7-100% / 93-97% in terms of recall/precision. Even assuming comparability of results, this would not be a vast difference. As it is, it is doubtful that much can be concluded from these figures regarding the inherent superiority of any of these methods to one another.

Voutilainen's (1995) argument that the Constraint Grammar system outperforms stochastic taggers is supported to some degree by the results of Chanod and Tapanainen's (1995a) exercise in comparing tagger methodologies (see section 5.6.2). However the difference is still not a great one (a 1.9% difference in accuracy rates). It would not be supportable to rule out using a probabilistic methodology on the basis of such a difference.

This being the case, I will move on to discuss the remaining methodologies in the light of factors pertaining specifically to this project and the tagging of Urdu.

5.7.2 Factors specific to this application in the tagging of Urdu

Three factors specific to the task undertaken will be considered here: the nature of the Urdu language, the nature of the tagset which I am using, and practical restrictions on what can conceivably be accomplished by this study.

5.7.2.1 The Urdu language

Perhaps surprisingly, it is not possible to eliminate any of the candidate tagging methodologies on the basis of typological features of Urdu. It *is* possible to rule out simply retraining some existing tagger (e.g. that of Brill 1995 or Cutting et al. 1992) on these grounds. Because Urdu is written in Perso-Arabic, the texts in question are coded in Unicode. The taggers discussed in the review above require ASCII text.

However, beyond this fairly superficial feature of the language, no further decisions can be made on the basis of, for example, Urdu's status as a fairly highly inflected language or its SOV word order. Sánchez León and Nieto Serrano (1997: 163-164) suggest that the potentially freer word order of morphologically rich languages could lead to such language having greater contextual ambiguity – i.e., making it harder to "guess" the tag of a word based the adjacent tags. Sánchez León and Nieto Serrano discuss this in the context of a Markov model tagger, and argue that, on the basis of this, morphologically rich languages would require a higher-order model than languages like English. This might suggest that for a language like Urdu a probabilistic model would be unsuitable, since it would require a high-order model which in turn would necessitate a much greater quantity of training data than is

available[74]. However, the problems associated with a freer word order cannot be assumed to apply only to Markov model taggers. There is no reason why a freer word order should not bedevil a rule-based approach just as much. As Brill (1995: 544) points out, all disambiguation techniques utilise the same kind of superficial information (see the quotation from Brill on page 229). Therefore the probabilistic approach cannot be ruled out solely on linguistic grounds.

5.7.2.2 The nature of the tagset

The tagset outlined in Chapter 3 contains approximately 380 tags. This is by most standards quite large, more than twice as large as most English tagsets. It has been suggested by Tapanainen and Voutilainen (1994) that Markov model taggers operate better with small tagsets, whereas rule-based approaches operate better with tagsets which are as large as possible. However, this is far from a universal conclusion.

It had previously been assumed that all taggers would perform better with a smaller tagset. For instance, Eklund (1993) states that "the fewer the tags, the more 'accurate' the output will be, due to the lack of more subtle subcategories." Subsequently, work by Sánchez León and Nieto Serrano (1997), using Spanish tagsets ranging from 40 to 475 tags, has demonstrated that with a Markov model tagger, a larger tagset improves performance if the model has the appropriate biases. The same result is reported by Smith (1997) in a comparison of the CLAWS system's performance with two English tagsets.

[74] See section 5.3.2.6 for a discussion of why higher-order models require more training data, and 4.5 for a discussion of the limitations on the amount of training data available.

Therefore, although there is still no firm agreement on the matter, it would seem that a fine-grained tagset is beneficial to both rule-based and probabilistic approaches, and the size of the tagset being used in this study is thus of no great use in deciding which methodology to employ.

5.7.2.3 Practical restrictions

The Urdu tagger described in this thesis was created by a single researcher, working in limited time and with limited resources. Although ideally one would select the optimal methodology without regard to these limitations, in practice if a working system is to be built practical restrictions on time and resources must be taken into account.

On practical grounds such as these, it is possible to exclude from consideration a hybrid of two methodologies. *A priori*, a hybrid tagger might be supposed to combine the best features of several methodologies. However, on grounds of time constraints a hybrid tagger can be ruled out. It is beyond the scope of this study to undertake, in effect, the creation of two completely different disambiguation systems[75]. Tapanainen and Voutilainen (1994) avoid this problem by creating their hybrid tagger from two pre-existing taggers. In the case of Urdu, there are no pre-existing taggers, and so the hybrid tagger remains an impracticability. Of course, my

[75] It may also be noted that there is no consensus on the best way to create a hybrid tagger. The work on CLAWS (see section 5.6.3) suggests that different types of disambiguation systems working in serial can be highly effective. However, the experiment performed by Chanod and Tapanainen (1995a) suggest exactly the opposite, that linking disambiguators together in serial is actually detrimental (see also section 5.6.2). On the other hand, Tapanainen and Voutilanen (1994) report good results running two taggers in parallel (see 5.6.3).

work in this study in creating one tagger opens the possibility of a hybrid approach being taken subsequently in the course of future research.

However, a much greater practical restriction is the amount of manually tagged data that is available. As outlined in the previous chapter, the total amount of manually tagged text available in this study was 90,000 words. This total was the result of limitations of time and money. However, it is typical for disambiguation methodologies which require training on pre-tagged data – those based on Markov models and on transformation-based error-driven learning – to be trained on much larger quantities of data than this. Many studies have used the tagged Brown Corpus (1 million words) as training and test data[76], and quantities of text on this scale may be considered typical. But the amount of data available in this study is less than one tenth of that amount. As discussed in section 4.5, less than 50,000 words are available. Nor is there sufficient data to treat the spoken and written languages separately, as would seem prudent. This was done, for example, in the case of CLAWS4, which was trained separately on spoken and written data to produce two separate probability matrices (Garside and Smith 1997: 115).

It must be assumed that without the quantities of training data available to the researchers who have produced impressive results with Markov models, those results could not be replicated by the Urdu tagger. The same applies to a transformation-based tagger. This is a factor which mitigates strongly against the use of either of these methodologies. This conclusion may also be justified in terms of the tagset size. As a general rule, the larger the tagset, the more data is needed to train a Markov

[76] For example, Charniak et al. (1993), Kupiec (1992), Church (1988), Brill and Pop (1999), and Cutting et al. (1992) all utilise the Brown Corpus, or a significant percentage of it, as their training data.

model since individual tags occur less frequently (see 5.3.2.6). The Urdu tagset used here is very large, making the lack of a large quantity of training data more problematic.

Some Markov model taggers can train on untagged data (see 5.3.2.5). However, a lexicon containing the possible tags for at least a significant proportion of the words is still necessary. In the case of Urdu, no such lexicon exists. As lexicons are typically derived from tagged corpora, the need for tagged data is not obviated in this way. Admittedly it would be possible, if difficult, to prepare one by hand. However it is not clear that this would be a more productive use of time and resources than additional manual tagging, given that such an effort could only be undertaken by a native speaker with full knowledge of the tagset.

By comparison, when disambiguation is performed using rules created by a linguist, this restriction does not apply. There is no need for a vast quantity of tagged data. Some tagged text is still required, as a benchmark is required against which to test the system, but a much smaller amount will suffice here than is required to train a Markov model.

On this basis, it is possible to rule out the use of a Markov model or a transformation-based tagging system for the disambiguation module of the Urdu tagger discussed in this thesis. The only remaining option, which has therefore been selected, is disambiguation based on hand-crafted rules.

5.8 Concluding remarks

At the outset of this chapter, I aimed to examine work conducted in the field of part-of-speech tag disambiguation techniques. To this end I have reviewed rule-based

approaches to disambiguation (particularly the Constraint Grammar framework), probabilistic approaches (particularly using Markov models), approaches using corpus-derived rules (particularly Brill's methodology of transformation-based error-driven learning) and machine learning approaches (particularly those using neural networks). I then reviewed the comparison of different methodologies, concluding that such a comparison is highly problematic. Therefore, one cannot justifiably select a disambiguation methodology based on its performance relative to other methodologies, because we have no reliable means of establishing what the difference may be.

I therefore moved on to discuss factors which influence the choice of an approach to disambiguation. Linguistic factors, such as the nature of Urdu grammar, or the size of the tagset being used, have been shown to give little help in selecting one methodology as more likely to succeed than another. However, the choice of methodology is very easily made on the grounds of practical restrictions on what may be done within the context of this project. As has been explained, the only approach which does not encounter some great practical difficulty is that utilising hand-crafted rules[77]. The justification for my choice of this approach is thus largely pragmatic, as has been outlined.

At this point, then, the preliminaries for the tagger – the tagset, the tagging manual, manually tagged text, and the choice of a methodology for disambiguation – are all in place. Therefore, in the next chapter, I move on to discuss the tagger experiment itself.

[77] It may be noted in passing that this method has an additional advantage: hand-crafted rules can be linguistically meaningful, and so some things about the structure of Urdu may be learnt in the process of creating and testing the rules.