10 *Part-of-Speech Tagging*

THE ULTIMATE GOAL of research on Natural Language Processing is to parse and understand language. As we have seen in the preceding chapters, we are still far from achieving this goal. For this reason, much research in NLP has focussed on intermediate tasks that make sense of some of the structure inherent in language without requiring complete understanding. One such task is part-of-speech tagging, or simply *tagging*. Tagging is the task of labeling (or tagging) each word in a sentence with its appropriate part of speech. We decide whether each word is a noun, verb, adjective, or whatever. Here is an example of a tagged sentence:

TAGGING

(10.1) The-AT representative-NN put-VBD chairs-NNS on-IN the-AT table-NN.

The part-of-speech tags we use in this chapter are shown in table 10.1, and generally follow the Brown/Penn tag sets (see section 4.3.2). Note that another tagging is possible for the same sentence (with the rarer sense for *put* of an *option to sell*):

(10.2) The-AT representative-JJ put-NN chairs-VBZ on-IN the-AT table-NN.

But this tagging gives rise to a semantically incoherent reading. The tagging is also syntactically unlikely since uses of *put* as a noun and uses of *chairs* as an intransitive verb are rare.

This example shows that tagging is a case of limited syntactic disambiguation. Many words have more than one syntactic category. In tagging, we try to determine which of these syntactic categories is the most likely for a particular use of a word in a sentence.

Tagging is a problem of limited scope: Instead of constructing a complete parse, we just fix the syntactic categories of the words in a sentence.

Tag	Part Of Speech
AT	article
BEZ	the word *is*
IN	preposition
JJ	adjective
JJR	comparative adjective
MD	modal
NN	singular or mass noun
NNP	singular proper noun
NNS	plural noun
PERIOD	. : ? !
PN	personal pronoun
RB	adverb
RBR	comparative adverb
TO	the word *to*
VB	verb, base form
VBD	verb, past tense
VBG	verb, present participle, gerund
VBN	verb, past participle
VBP	verb, non-3rd person singular present
VBZ	verb, 3rd singular present
WDT	*wh*- determiner (*what*, *which*)

Table 10.1 Some part-of-speech tags frequently used for tagging English.

For example, we are not concerned with finding the correct attachment of prepositional phrases. As a limited effort, tagging is much easier to solve than parsing, and accuracy is quite high. Between 96% and 97% of tokens are disambiguated correctly by the most successful approaches. However, it is important to realize that this impressive accuracy figure is not quite as good as it looks, because it is evaluated on a per-word basis. For instance, in many genres such as newspapers, the average sentence is over twenty words, and on such sentences, even with a tagging accuracy of 96% this means that there will be on average over one tagging error per sentence.

Even though it is limited, the information we get from tagging is still quite useful. Tagging can be used in information extraction, question an-

swering, and shallow parsing. The insight that tagging is an intermediate layer of representation that is useful and more tractable than full parsing is due to the corpus linguistics work that was led by Francis and Kučera at Brown University in the 1960s and 70s (Francis and Kučera 1982).

The following sections deal with Markov Model taggers, Hidden Markov Model taggers and transformation-based tagging. At the end of the chapter, we discuss levels of accuracy for different approaches to tagging. But first we make some general comments on the types of information that are available for tagging.

10.1 The Information Sources in Tagging

How can one decide the correct part of speech for a word used in a context? There are essentially two sources of information. One way is to look at the tags of other words in the context of the word we are interested in. These words may also be ambiguous as to their part of speech, but the essential observation is that some part of speech sequences are common, such as AT JJ NN, while others are extremely unlikely or impossible, such as AT JJ VBP. Thus when choosing whether to give an NN or a VBP tag to the word *play* in the phrase *a new play*, we should obviously choose the former. This type of *syntagmatic* structural information is the most obvious source of information for tagging, but, by itself, it is not very successful. For example, Greene and Rubin (1971), an early deterministic rule-based tagger that used such information about syntagmatic patterns correctly tagged only 77% of words. This made the tagging problem look quite hard. One reason that it looks hard is that many content words in English can have various parts of speech. For example, there is a very productive process in English which allows almost any noun to be turned into a verb, for example, *Next, you **flour** the pan*, or, *I want you to **web** our annual report*. This means that almost any noun should also be listed in a dictionary as a verb as well, and we lose a lot of constraining information needed for tagging.

SYNTAGMATIC

These considerations suggest the second information source: just knowing the word involved gives a lot of information about the correct tag. Although *flour* can be used as a verb, an occurrence of *flour* is much more likely to be a noun. The utility of this information was conclusively demonstrated by Charniak et al. (1993), who showed that a 'dumb' tagger

that simply assigns the most common tag to each word performs at the surprisingly high level of 90% correct.[1] This made tagging look quite easy – at least given favorable conditions, an issue to which we shall return. As a result of this, the performance of such a 'dumb' tagger has been used to give a baseline performance level in subsequent studies. And all modern taggers in some way make use of a combination of syntagmatic information (looking at information about tag sequences) and lexical information (predicting a tag based on the word concerned).

Lexical information is so useful because the distribution of a word's usages across different parts of speech is typically extremely uneven. Even for words with a number of parts of speech, they usually occur used as one particular part of speech. Indeed, this distribution is usually so marked that this one part of speech is often seen as basic, with others being derived from it. As a result, this has led to a certain tension over the way the term 'part of speech' has been used. In traditional grammars, one often sees a word in context being classified as something like 'a noun being used as an adjective,' which confuses what is seen as the 'basic' part of speech of the lexeme with the part of speech of the word as used in the current context. In this chapter, as in modern linguistics in general, we are concerned with determining the latter concept, but nevertheless, the distribution of a word across the parts of speech gives a great deal of additional information. Indeed, this uneven distribution is one reason why one might expect statistical approaches to tagging to be better than deterministic approaches: in a deterministic approach one can only say that a word can or cannot be a verb, and there is a temptation to leave out the verb possibility if it is very rare (since doing so will probably lift the level of overall performance), whereas within a statistical approach, we can say that a word has an extremely high *a priori* probability of being a noun, but there is a small chance that it might be being used as a verb, or even some other part of speech. Thus syntactic disambiguation can be argued to be one context in which a framework that allows quantitative information is more adequate for representing linguistic knowledge than a purely symbolic approach.

1. The general efficacy of this method was noted earlier by Atwell (1987).

10.2 Markov Model Taggers

10.2.1 The probabilistic model

In Markov Model tagging, we look at the sequence of tags in a text as a Markov chain. As discussed in chapter 9, a Markov chain has the following two properties:

- **Limited horizon.** $P(X_{i+1} = t^j | X_1, \ldots, X_i) = P(X_{i+1} = t^j | X_i)$
- **Time invariant (stationary).** $P(X_{i+1} = t^j | X_i) = P(X_2 = t^j | X_1)$

That is, we assume that a word's tag only depends on the previous tag (limited horizon) and that this dependency does not change over time (time invariance). For example, if a finite verb has a probability of 0.2 to occur after a pronoun at the beginning of a sentence, then this probability will not change as we tag the rest of the sentence (or new sentences). As with most probabilistic models, the two Markov properties only approximate reality. For example, the Limited Horizon property does not model long-distance relationships like *Wh*-extraction – this was in fact the core of Chomsky's famous argument against using Markov Models for natural language.

Exercise 10.1 [⋆]

What are other linguistic phenomena that are not modeled correctly by Markov chains? Which general property of language is common to these phenomena?

Exercise 10.2 [⋆]

Why is Time Invariance problematic for modeling language?

Following (Charniak et al. 1993), we will use the notation in table 10.2. We use subscripts to refer to words and tags in particular positions of the sentences and corpora we tag. We use superscripts to refer to word types in the lexicon of words and to refer to tag types in the tag set. In this compact notation, we can state the above Limited Horizon property as follows:

$$P(t_{i+1} | t_{1,i}) = P(t_{i+1} | t_i)$$

We use a *training set* of manually tagged text to learn the regularities of tag sequences. The maximum likelihood estimate of tag t^k following

w_i	the word at position i in the corpus
t_i	the tag of w_i
$w_{i,i+m}$	the words occurring at positions i through $i+m$ (alternative notations: $w_i \cdots w_{i+m}$, $w_i, \ldots, w_{i+m}$, $w_{i(i+m)}$)
$t_{i,i+m}$	the tags $t_i \cdots t_{i+m}$ for $w_i \cdots w_{i+m}$
w^l	the l^{th} word in the lexicon
t^j	the j^{th} tag in the tag set
$C(w^l)$	the number of occurrences of w^l in the training set
$C(t^j)$	the number of occurrences of t^j in the training set
$C(t^j, t^k)$	the number of occurrences of t^j followed by t^k
$C(w^l : t^j)$	the number of occurrences of w^l that are tagged as t^j
T	number of tags in tag set
W	number of words in the lexicon
n	sentence length

Table 10.2 Notational conventions for tagging.

t^j is estimated from the relative frequencies of different tags following a certain tag as follows:

$$P(t^k|t^j) = \frac{C(t^j, t^k)}{C(t^j)}$$

For instance, following on from the example of how to tag *a new play*, we would expect to find that $P(\text{NN}|\text{JJ}) \gg P(\text{VBP}|\text{JJ})$. Indeed, on the Brown corpus, $P(\text{NN}|\text{JJ}) \approx 0.45$ and $P(\text{VBP}|\text{JJ}) \approx 0.0005$.

With estimates of the probabilities $P(t_{i+1}|t_i)$, we can compute the probability of a particular tag sequence. In practice, the task is to find the most probable tag sequence for a sequence of words, or equivalently, the most probable state sequence for a sequence of words (since the states of the Markov Model here are tags). We incorporate words by having the Markov Model emit words each time it leaves a state. This is similar to the symbol emission probabilities b_{ijk} in HMMs from chapter 9:

$$P(O_n = k|X_n = s_i, X_{n+1} = s_j) = b_{ijk}$$

The difference is that we can directly observe the states (or tags) if we have a tagged corpus. Each tag corresponds to a different state. We

can also directly estimate the probability of a word being emitted by a particular state (or tag) via Maximum Likelihood Estimation:

$$P(w^l|t^j) = \frac{C(w^l, t^j)}{C(t^j)}$$

Now we have everything in place to find the best tagging $t_{1,n}$ for a sentence $w_{1,n}$. Applying Bayes' rule, we can write:

$$\begin{aligned} \arg\max_{t_{1,n}} P(t_{1,n}|w_{1,n}) &= \arg\max_{t_{1,n}} \frac{P(w_{1,n}|t_{1,n})P(t_{1,n})}{P(w_{1,n})} \\ &= \arg\max_{t_{1,n}} P(w_{1,n}|t_{1,n})P(t_{1,n}) \end{aligned} \tag{10.3}$$

We now reduce this expression to parameters that can be estimated from the training corpus. In addition to the Limited Horizon assumption (10.5), we make two assumptions about words:

- words are independent of each other (10.4), and
- a word's identity only depends on its tag (10.5)

$$\begin{aligned} P(w_{1,n}|t_{1,n})P(t_{1,n}) &= \prod_{i=1}^{n} P(w_i|t_{1,n}) \\ &\quad \times P(t_n|t_{1,n-1}) \times P(t_{n-1}|t_{1,n-2}) \times \cdots \times P(t_2|t_1) \end{aligned} \tag{10.4}$$

$$\begin{aligned} &= \prod_{i=1}^{n} P(w_i|t_i) \\ &\quad \times P(t_n|t_{n-1}) \times P(t_n - 1|t_{n-2}) \times \cdots \times P(t_2|t_1) \end{aligned} \tag{10.5}$$

$$= \prod_{i=1}^{n} [P(w_i|t_i) \times P(t_i|t_{i-1})] \tag{10.6}$$

(We define $P(t_1|t_0) = 1.0$ to simplify our notation.)

Exercise 10.3 [⋆]

These are simplifying assumptions. Give two examples each of phenomena where independence of words (10.4) and independence from previous and following tags (10.5) don't hold.

So the final equation for determining the optimal tags for a sentence is:

$$\hat{t}_{1,n} = \arg\max_{t_{1,n}} P(t_{1,n}|w_{1,n}) = \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1}) \tag{10.7}$$

```
1  for all tags t^j do
2      for all tags t^k do
3          P(t^k|t^j) := C(t^j,t^k) / C(t^j)
4      end
5  end
6  for all tags t^j do
7      for all words w^l do
8          P(w^l|t^j) := C(w^l,t^j) / C(t^j)
9      end
10 end
```

Figure 10.1 Algorithm for training a Visible Markov Model Tagger. In most implementations, a smoothing method is applied for estimating the $P(t^k|t^j)$ and $P(w^l|t^j)$.

	Second tag					
First tag	AT	BEZ	IN	NN	VB	PERIOD
AT	0	0	0	48636	0	19
BEZ	1973	0	426	187	0	38
IN	43322	0	1325	17314	0	185
NN	1067	3720	42470	11773	614	21392
VB	6072	42	4758	1476	129	1522
PERIOD	8016	75	4656	1329	954	0

Table 10.3 Idealized counts of some tag transitions in the Brown Corpus. For example, NN occurs 48636 times after AT.

The algorithm for training a Markov Model tagger is summarized in figure 10.1. The next section describes how to tag with a Markov Model tagger once it is trained.

Exercise 10.4 [⋆]

Given the data in table 10.3, compute maximum likelihood estimates as shown in figure 10.1 for P(AT|PERIOD), P(NN|AT), P(BEZ|NN), P(IN|BEZ), P(AT|IN), and P(PERIOD|NN). Assume that the total number of occurrences of tags can be obtained by summing over the numbers in a row (e.g., 1973+426+187 for BEZ).

Exercise 10.5 [⋆]

Given the data in table 10.4, compute maximum likelihood estimates as shows in figure 10.1 for $P(bear|t^k)$, $P(is|t^k)$, $P(move|t^k)$, $P(president|t^k)$, $P(progress|t^k)$, and $P(the|t^k)$. Take the total number of occurrences of tags from table 10.3.

	AT	BEZ	IN	NN	VB	PERIOD
bear	0	0	10	0	43	0
is	0	10065	0	0	0	0
move	0	0	0	36	133	0
on	0	0	5484	0	0	0
president	0	0	0	382	0	0
progress	0	0	0	108	4	0
the	69016	0	0	0	0	0
.	0	0	0	0	0	48809

Table 10.4 Idealized counts of tags that some words occur within the Brown Corpus. For example, 36 occurrences of *move* are with the tag NN.

Exercise 10.6 [⋆]

Compute the following two probabilities:

- P(AT NN BEZ IN AT NN|The bear is on the move.)
- P(AT NN BEZ IN AT VB|The bear is on the move.)

10.2.2 The Viterbi algorithm

We could evaluate equation (10.7) for all possible taggings $t_{1,n}$ of a sentence of length n, but that would make tagging exponential in the length of the input that is to be tagged. An efficient tagging algorithm is the Viterbi algorithm from chapter 9. To review, the Viterbi algorithm has three steps: (i) initialization, (ii) induction, and (iii) termination and path-readout. We compute two functions $\delta_i(j)$, which gives us the probability of being in state j (= tag j) at word i, and $\psi_{i+1}(j)$, which gives us the most likely state (or tag) at word i given that we are in state j at word $i+1$. The reader may want to review the discussion of the Viterbi algorithm in section 9.3.2 before reading on. Throughout, we will refer to states as tags in this chapter because the states of the model correspond to tags. (But note that this is only true for a bigram tagger.)

The initialization step is to assign probability 1.0 to the tag PERIOD:

$$\delta_1(\text{PERIOD}) = 1.0$$

$$\delta_1(t) = 0.0 \text{ for } t \neq \text{PERIOD}$$

That is, we assume that sentences are delimited by periods and we prepend a period in front of the first sentence in our text for convenience.

comment: Given: a sentence of length n
comment: Initialization
$\delta_1(\text{PERIOD}) = 1.0$
$\delta_1(t) = 0.0$ for $t \neq \text{PERIOD}$
comment: Induction
for $i := 1$ **to** n **step** 1 **do**
    for all tags t^j **do**
        $\delta_{i+1}(t^j) := \max_{1 \leq k \leq T}[\delta_i(t^k) \times P(w_{i+1}|t^j) \times P(t^j|t^k)]$
        $\psi_{i+1}(t^j) := \arg\max_{1 \leq k \leq T}[\delta_i(t^k) \times P(w_{i+1}|t^j) \times P(t^j|t^k)]$
    end
end
comment: Termination and path-readout
$X_{n+1} = \arg\max_{1 \leq j \leq T} \delta_{n+1}(j)$
for $j := n$ **to** 1 **step** -1 **do**
    $X_j = \psi_{j+1}(X_{j+1})$
end
$P(X_1, \ldots, X_n) = \max_{1 \leq j \leq T} \delta_{n+1}(t^j)$

Figure 10.2 Algorithm for tagging with a Visible Markov Model Tagger.

The induction step is based on equation (10.7), where $a_{jk} = P(t^k|t^j)$ and $b_{jkw^l} = P(w^l|t^j)$:

$$\delta_{i+1}(t^j) = \max_{1 \leq k \leq T}[\delta_i(t^k) \times P(w_{i+1}|t^j) \times P(t^j|t^k)], \quad 1 \leq j \leq T$$

$$\psi_{i+1}(t^j) = \arg\max_{1 \leq k \leq T}[\delta_i(t^k) \times P(w_{i+1}|t^j) \times P(t^j|t^k)], \quad 1 \leq j \leq T$$

Finally, termination and read-out steps are as follows, where $X_1, \ldots, X_n$ are the tags we choose for words $w_1, \ldots, w_n$:

$$X_n = \arg\max_{1 \leq j \leq T} \delta_n(t^j)$$

$$X_i = \psi_{i+1}(X_{i+1}), \quad 1 \leq i \leq n-1$$

$$P(X_1, \ldots, X_n) = \max_{1 \leq j \leq T} \delta_{n+1}(t^j)$$

Tagging with a Visible Markov Model tagger is summarized in figure 10.2.

Exercise 10.7 [⋆]

Based on the probability estimates from the previous set of exercises, tag the following sentence using the Viterbi algorithm.

(10.8) The bear is on the move.

Exercise 10.8

Some larger data sets of tag sequence probabilities and some suggested exercises are available on the website.

Terminological note: Markov Models vs. Hidden Markov Models. The reader may have noticed that for the purposes of tagging, the Markov Models in this chapter are treated as Hidden Markov Models. This is because we can observe the states of the Markov Model in training (the tags of the labeled corpus), but we only observe words in applying the Markov Model to the tagging task. We could say that the formalism used in Markov Model tagging is really a mixed formalism. We construct 'Visible' Markov Models in training, but treat them as Hidden Markov Models when we put them to use and tag new corpora.

10.2.3 Variations

Unknown words

We have shown how to estimate word generation probabilities for words that occur in the corpus. But many words in sentences we want to tag will not be in the training corpus. Some words will not even be in the dictionary. We discussed above that knowing the a priori distribution of the tags for a word (or at any rate the most common tag for a word) takes you a great deal of the way in solving the tagging problem. This means that unknown words are a major problem for taggers, and in practice, the differing accuracy of different taggers over different corpora is often mainly determined by the proportion of unknown words, and the smarts built into the tagger that allow it to try to guess the part of speech of unknown words.

The simplest model for unknown words is to assume that they can be of any part of speech (or perhaps only any open class part of speech – that is nouns, verbs, etc., but not prepositions or articles). Unknown words are given a distribution over parts of speech corresponding to that of the lexicon as a whole. While this approach is serviceable in some cases, the loss of lexical information for these words greatly lowers the accuracy of the tagger, and so people have tried to exploit other features of the word and its context to improve the lexical probability estimates for unknown words. Often, we can use morphological and other cues to

Feature	Value	NNP	NN	NNS	VBG	VBZ
unknown word	yes	0.05	0.02	0.02	0.005	0.005
	no	0.95	0.98	0.98	0.995	0.995
capitalized	yes	0.95	0.10	0.10	0.005	0.005
	no	0.05	0.90	0.90	0.995	0.995
ending	*-s*	0.05	0.01	0.98	0.00	0.99
	-ing	0.01	0.01	0.00	1.00	0.00
	-tion	0.05	0.10	0.00	0.00	0.00
	other	0.89	0.88	0.02	0.00	0.01

Table 10.5 Table of probabilities for dealing with unknown words in tagging. For example, $P(\text{unknown word} = \text{yes}|\text{NNP}) = 0.05$ and $P(\text{ending} = \textit{-ing}|\text{VBG}) = 1.0$.

make inferences about a word's possible parts of speech. For example, words ending in *-ed* are likely to be past tense forms or past participles. Weischedel et al. (1993) estimate word generation probabilities based on three types of information: how likely it is that a tag will generate an unknown word (this probability is zero for some tags, for example PN, personal pronouns); the likelihood of generation of uppercase/lowercase words; and the generation of hyphens and particular suffixes:

$$P(w^l|t^j) = \frac{1}{Z} P(\text{unknown word}|t^j) P(\text{capitalized}|t^j) P(\text{endings/hyph}|t^j)$$

where Z is a normalization constant. This model reduces the error rate for unknown words from more than 40% to less than 20%.

Charniak et al. (1993) propose an alternative model which depends both on roots and suffixes and can select from multiple morphological analyses (for example, *do-es* (a verb form) vs. *doe-s* (the plural of a noun)).

Most work on unknown words assumes independence between features. Independence is often a bad assumption. For example, capitalized words are more likely to be unknown, so the features 'unknown word' and 'capitalized' in Weischedel et al.'s model are not really independent. Franz (1996; 1997) develops a model for unknown words that takes dependence into account. He proposes a loglinear model that models *main effects* (the effects of a particular feature on its own) as well as *interactions* (such as the dependence between 'unknown word' and 'capitalized'). For an approach based on Bayesian inference see Samuelsson (1993).

MAIN EFFECTS
INTERACTIONS

Exercise 10.9 [⋆]

Given the (made-up) data in table 10.5 and Weischedel et al.'s model for unknown words, compute $P(\textit{fenestration}|t^k)$, $P(\textit{fenestrates}|t^k)$, $P(\textit{palladio}|t^k)$, $P(\textit{palladios}|t^k)$, $P(\textit{Palladio}|t^k)$, $P(\textit{Palladios}|t^k)$, and $P(\textit{guesstimating}|t^k)$. Assume that NNP, NN, NNS, VBG, and VBZ are the only possible tags. Do the estimates seem intuitively correct? What additional features could be used for better results?

Exercise 10.10 [⋆⋆]

Compute better estimates of the probabilities in table 10.5 from the data on the web site.

Trigram taggers

The basic Markov Model tagger can be extended in several ways. In the model developed so far, we make predictions based on the preceding tag.

BIGRAM TAGGER

This is called a *bigram tagger* because the basic unit we consider is the preceding tag and the current tag. We can think of tagging as selecting the most probable bigram (modulo word probabilities).

We would expect more accurate predictions if more context is taken into account. For example, the tag RB (adverb) can precede both a verb in the past tense (VBD) and a past participle (VBN). So a word sequence like *clearly marked* is inherently ambiguous in a Markov Model with a 'memory' that reaches only one tag back. A *trigram tagger* has a two-tag memory and lets us disambiguate more cases. For example, *is clearly marked* and *he clearly marked* suggest VBN and VBD, respectively, because the trigram "BEZ RB VBN" is more frequent than the trigram "BEZ RB VBD" and because "PN RB VBD" is more frequent than "PN RB VBN." A trigram tagger was described in (Church 1988), which is probably the most cited publication on tagging and got many NLP researchers interested in the problem of part-of-speech tagging.

TRIGRAM TAGGER

Interpolation and variable memory

Conditioning predictions on a longer history is not always a good idea. For example, there are usually no short-distance syntactic dependencies across commas. So knowing what part of speech occurred before a comma does not help in determining the correct part of speech after the comma. In fact, a trigram tagger may make worse predictions than a bigram tagger in such cases because of sparse data problems –

trigram transition probabilities are estimated based on rarer events, so the chances of getting a bad estimate are higher.

One way to address this problem is linear interpolation of unigram, bigram, and trigram probabilities:

$$P(t_i|t_{1,i-1}) = \lambda_1 P_1(t_i) + \lambda_2 P_2(t_i|t_{i-1}) + \lambda_3 P_3(t_i|t_{i-1,i-2})$$

This method of linear interpolation was covered in chapter 6 and how to estimate the parameters λ_i using an HMM was covered in chapter 9.

Some researchers have selectively augmented a low-order Markov model based on error analysis and prior linguistic knowledge. For example, Kupiec (1992b) observed that a first order HMM systematically mistagged the sequence "the bottom of" as "AT JJ IN." He then extended the order-one model with a special network for this construction so that the improbability of a preposition after a "AT JJ" sequence could be learned. This method amounts to manually selecting higher-order states for cases where an order-one memory is not sufficient.

A related method is the Variable Memory Markov Model (VMMM) (Schütze and Singer 1994). VMMMs have states of mixed "length" instead of the fixed-length states of bigram and trigram taggers. A VMMM tagger can go from a state that remembers the last two tags (corresponding to a trigram) to a state that remembers the last three tags (corresponding to a fourgram) and then to a state without memory (corresponding to a unigram). The number of symbols to remember for a particular sequence is determined in training based on an information-theoretic criterion. In contrast to linear interpolation, VMMMs condition the length of memory used for prediction on the current sequence instead of using a fixed weighted sum for all sequences. VMMMs are built top-down by splitting states. An alternative is to build this type of model bottom-up by way of *model merging* (Stolcke and Omohundro 1994a; Brants 1998).

MODEL MERGING

The hierarchical non-emitting Markov model is an even more powerful model that was proposed by Ristad and Thomas (1997). By introducing non-emitting transitions (transitions between states that do not emit a word or, equivalently, emit the empty word ϵ), this model can store dependencies between states over arbitrarily long distances.

Smoothing

Linear interpolation is a way of smoothing estimates. We can use any of the other estimation methods discussed in chapter 6 for smoothing. For

example, Charniak et al. (1993) use a method that is similar to Adding One (but note that, in general, it does not give a proper probability distribution ...):

$$P(t^j|t^{j+1}) = (1-\epsilon)\frac{C(t^{j-1},t^j)}{C(t^{j-1})} + \epsilon$$

Smoothing the word generation probabilities is more important than smoothing the transition probabilities since there are many rare words that will not occur in the training corpus. Here too, Adding One has been used (Church 1988). Church added 1 to the count of all parts of speech listed in the dictionary for a particular word, thus guaranteeing a non-zero probability for all parts of speech t^j that are listed as possible for w^l:

$$P(t^j|w^l) = \frac{C(t^j,w^l)+1}{C(w^l)+K_l}$$

where K_l is the number of possible parts of speech of w^l.

Exercise 10.11 [⋆]

Recompute the probability estimates in exercises 10.4 and 10.5 with Adding One.

Reversibility

We have described a Markov Model that 'decodes' (or tags) from left to right. It turns out that decoding from right to left is equivalent. The following derivation shows why this is the case:

$$\begin{aligned} P(t_{1,n}) &= P(t_1)P(t_{1,2}|t_1)P(t_{2,3}|t_2)\ldots P(t_{n-1,n}|t_{n-1}) \\ &= \frac{P(t_1)P(t_{1,2})P(t_{2,3})\ldots P(t_{n-1,n})}{P(t_1)P(t_2)\ldots P(t_{n-1})} \\ &= P(t_n)P(t_{1,2}|t_2)P(t_{2,3}|t_3)\ldots P(t_{n-1,n}|t_n) \end{aligned} \tag{10.9}$$

Assuming that the probability of the initial and last states are the same (which is the case in tagging since both correspond to the tag PERIOD), 'forward' and 'backward' probability are the same. So it doesn't matter which direction we choose. The tagger described here moves from left to right. Church's tagger takes the opposite direction.

Maximum Likelihood: Sequence vs. tag by tag

As we pointed out in chapter 9, the Viterbi Algorithm finds the most likely sequence of states (or tags). That is, we maximize $P(t_{1,n}|w_{1,n})$. We

could also maximize $P(t_i|w_{1,n})$ for all i which amounts to summing over different tag sequences.

As an example consider sentence (10.10):

(10.10) Time flies like an arrow.

Let us assume that, according to the transition probabilities we've gathered from our training corpus, (10.11a) and (10.11b) are likely taggings (assume probability 0.01), (10.11c) is an unlikely tagging (assume probability 0.001), and that (10.11d) is impossible because transition probability $P(\text{VB}|\text{VBZ})$ is 0.0.

(10.11) a. NN VBZ RB AT NN. $P(\cdot) = 0.01$

b. NN NNS VB AT NN. $P(\cdot) = 0.01$

c. NN NNS RB AT NN. $P(\cdot) = 0.001$

d. NN VBZ VB AT NN. $P(\cdot) = 0$

For this example, we will obtain taggings (10.11a) and (10.11b) as the equally most likely sequences $P(t_{1,n}|w_{1,n})$. But we will obtain (10.11c) if we maximize $P(t_i|w_{1,n})$ for all i. This is because $P(X_2 = \text{NNS}|\textit{Time flies like an arrow}) = 0.011 = P(b) + P(c) > 0.01 = P(a) = P(X_2 = \text{VBZ}|\textit{Time flies like an arrow})$ and $P(X_3 = \text{RB}|\textit{Time flies like an arrow}) = 0.011 = P(a) + P(c) > 0.01 = P(b) = P(X_3 = \text{VB}|\textit{Time flies like an arrow})$.

Experiments conducted by Merialdo (1994: 164) suggest that there is no large difference in accuracy between maximizing the likelihood of individual tags and maximizing the likelihood of the sequence. Intuitively, it is fairly easy to see why this might be. With Viterbi, the tag transitions are more likely to be sensible, but if something goes wrong, we will sometimes get a sequence of several tags wrong; whereas with tag by tag, one error does not affect the tagging of other words, and so one is more likely to get occasional dispersed errors. In practice, since incoherent sequences (like "NN NNS RB AT NN" above) are not very useful, the Viterbi algorithm is the preferred method for tagging with Markov Models.

10.3 Hidden Markov Model Taggers

Markov Model taggers work well when we have a large tagged training set. Often this is not the case. We may want to tag a text from a specialized domain with word generation probabilities that are different from

those in available training texts. Or we may want to tag text in a foreign language for which training corpora do not exist at all.

10.3.1 Applying HMMs to POS tagging

If we have no training data, we can use an HMM to learn the regularities of tag sequences. Recall that an HMM as introduced in chapter 9 consists of the following elements:

- a set of states
- an output alphabet
- initial state probabilities
- state transition probabilities
- symbol emission probabilities

As in the case of the Visible Markov Model, the states correspond to tags. The output alphabet consists either of the words in the dictionary or classes of words as we will see in a moment.

We could randomly initialize all parameters of the HMM, but this would leave the tagging problem too unconstrained. Usually dictionary information is used to constrain the model parameters. If the output alphabet consists of words, we set word generation (= symbol emission) probabilities to zero if the corresponding word-tag pair is not listed in the dictionary (e.g., JJ is not listed as a possible part of speech for *book*). Alternatively, we can group words into word equivalence classes so that all words that allow the same set of tags are in the same class. For example, we could group *bottom* and *top* into the class JJ-NN if both are listed with just two parts of speech, JJ and NN. The first method was proposed by Jelinek (1985), the second by Kupiec (1992b). We write $b_{j.l}$ for the probability that word (or word class) l is emitted by tag j. This means that as in the case of the Visible Markov Model the 'output' of a tag does not depend on which tag (= state) is next.

- **Jelinek's method.**

$$b_{j.l} = \frac{b^{\star}_{j.l} C(w^l)}{\sum_{w^m} b^{\star}_{j.m} C(w^m)}$$

where the sum is over all words w^m in the dictionary and

$$b^{\star}_{j.l} = \begin{cases} 0 & \text{if } t^j \text{ is not a part of speech allowed for } w^l \\ \frac{1}{T(w^l)} & \text{otherwise} \end{cases}$$

where $T(w^j)$ is the number of tags allowed for w^j.

Jelinek's method amounts to initializing the HMM with the maximum likelihood estimates for $P(w^k|t^i)$, assuming that words occur equally likely with each of their possible tags.

- **Kupiec's method.** First, group all words with the same possible parts of speech into 'metawords' u_L. Here L is a subset of the integers from 1 to T, where T is the number of different tags in the tag set:

  $$u_L = \{w^l | j \in L \leftrightarrow t^j \text{is allowed for } w^l\} \quad \forall L \subseteq \{1, \ldots, T\}$$

  For example, if NN = t^5 and JJ = t^8 then $u_{\{5,8\}}$ will contain all words for which the dictionary allows tags NN and JJ and no other tags.

  We then treat these metawords u_L the same way we treated words in Jelinek's method:[2]

  $$b_{j.L} = \frac{b^{\star}_{j.L} C(u_L)}{\sum_{u_{L'}} b^{\star}_{j.L'} C(u_{L'})}$$

  where $C(u_{L'})$ is the number of occurrences of words from $u_{L'}$, the sum in the denominator is over all metawords $u_{L'}$, and

  $$b^{\star}_{j.L} = \begin{cases} 0 & \text{if } j \notin L \\ \frac{1}{|L|} & \text{otherwise} \end{cases}$$

  where $|L|$ is the number of indices in L.

The advantage of Kupiec's method is that we don't fine-tune a separate set of parameters for each word. By introducing equivalence classes, the total number of parameters is reduced substantially and this smaller set can be estimated more reliably. This advantage could turn into a disadvantage if there is enough training material to accurately estimate parameters word by word as Jelinek's method does. Some experiments

2. The actual initialization used by Kupiec is a variant of what we present here. We have tried to make the similarity between Jelinek's and Kupiec's methods more transparent.

D0	maximum likelihood estimates from a tagged training corpus
D1	correct ordering only of lexical probabilities
D2	lexical probabilities proportional to overall tag probabilities
D3	equal lexical probabilities for all tags admissible for a word
T0	maximum likelihood estimates from a tagged training corpus
T1	equal probabilities for all transitions

Table 10.6 Initialization of the parameters of an HMM. D0, D1, D2, and D3 are initializations of the lexicon, and T0 and T1 are initializations of tag transitions investigated by Elworthy.

conducted by Merialdo (1994) suggest that unsupervised estimation of a separate set of parameters for each word introduces error. This argument does not apply to frequent words, however. Kupiec therefore does not include the 100 most frequent words in equivalence classes, but treats them as separate one-word classes.

Training. Once initialization is completed, the Hidden Markov Model is trained using the Forward-Backward algorithm as described in chapter 9.

Tagging. As we remarked earlier, the difference between VMM tagging and HMM tagging is in how we *train* the model, not in how we *tag.* The formal object we end up with after training is a Hidden Markov model in both cases. For this reason, there is no difference when we apply the model in tagging. We use the Viterbi algorithm in exactly the same manner for Hidden Markov Model tagging as we do for Visible Markov Model tagging.

10.3.2 The effect of initialization on HMM training

The 'clean' (i.e., theoretically well-founded) way of stopping training with the Forward-Backward algorithm is the log likelihood criterion (stop when the log likelihood no longer improves). However, it has been shown that, for tagging, this criterion often results in overtraining. This issue was investigated in detail by Elworthy (1994). He trained HMMs from the different starting conditions in table 10.6. The combination of D0 and T0 corresponds to Visible Markov Model training as we described it at the beginning of this chapter. D1 orders the lexical probabilities correctly

(for example, the fact that the tag VB is more likely for *make* than the tag NN), but the absolute values of the probabilities are randomized. D2 gives the same ordering of parts of speech to all words (for example, for the most frequent tag t^j, we would have $P(w|t^j)$ is greater than $P(w|t^k)$ for all other tags t^k). D3 preserves only information about which tags are possible for a word, the ordering is not necessarily correct. T1 initializes the transition probabilities to roughly equal numbers.[3]

CLASSICAL

EARLY MAXIMUM

INITIAL MAXIMUM

Elworthy (1994) finds three different patterns of training for different combinations of initial conditions. In the *classical* pattern, performance on the test set improves steadily with each training iteration. In this case the log likelihood criterion for stopping is appropriate. In the *early maximum* pattern, performance improves for a number of iterations (most often for two or three), but then decreases. In the *initial maximum* pattern, the very first iteration degrades performance.

The typical scenario for applying HMMs is that a dictionary is available, but no tagged corpus as training data (conditions D3 (maybe D2) and T1). For this scenario, training follows the early maximum pattern. That means that we have to be careful in practice not to overtrain. One way to achieve this is to test the tagger on a held-out validation set after each iteration and stop training when performance decreases.

Elworthy also confirms Merialdo's finding that the Forward-Backward algorithm degrades performance when a tagged training corpus (of even moderate size) is available. That is, if we initialize according to D0 and T0, then we get the initial maximum pattern. However, an interesting twist is that if training and test corpus are very different, then a few iterations do improve performance (the early maximum pattern). This is a case that occurs frequently in practice since we are often confronted with types of text for which we do not have similar tagged training text.

In summary, if there is a sufficiently large training text that is fairly similar to the intended text of application, then we should use Visible Markov Models. If there is no training text available or training and test text are very different, but we have at least some lexical information, then we should run the Forward-Backward algorithm for a few iterations. Only when we have no lexical information at all, should we train for a larger number of iterations, ten or more. But we cannot expect good perfor-

3. Exactly identical probabilities are generally bad as a starting condition for the EM algorithm since they often correspond to suboptimal local optima that can easily be avoided. We assume that D3 and T1 refer to approximately equal probabilities that are slightly perturbed to avoid ties.

mance in this case. This failure is not a defect in the forward-backward algorithm, but reflects the fact that the forward-backward algorithm is only maximizing the likelihood of the training data by adjusting the parameters of an HMM. The changes it is using to reduce the cross entropy may not be in accord with our true objective function – getting words assigned tags according to some predefined tag set. Therefore it is not capable of optimizing performance on that task.

Exercise 10.12 [⋆]

When introducing HMM tagging above, we said that random initialization of the model parameters (without dictionary information) is not a useful starting point for the EM algorithm. Why is this the case? What would happen if we just had the following eight parts of speech: preposition, verb, adverb, adjective, noun, article, conjunction, and auxiliary; and randomly initialized the HMM. Hint: The EM algorithm will concentrate on high-frequency events which have the highest impact on log likelihood (the quantity maximized).

How does this initialization differ from D3?

Exercise 10.13 [⋆]

The EM algorithm improves the log likelihood of the model given the data in each iteration. How is this compatible with Elworthy's and Merialdo's results that tagging accuracy often decreases with further training?

Exercise 10.14 [⋆]

The crucial bit of prior knowledge that is captured by both Jelinek's and Kupiec's methods of parameter initialization is which of the word generation probabilities should be zero and which should not. The implicit assumption here is that a generation probability set to zero initially will remain zero during training. Show that this is the case referring to the introduction of the Forward-Backward algorithm in chapter 9.

Exercise 10.15 [⋆⋆]

Get the Xerox tagger (see pointer on website) and tag texts from the web site.

10.4 Transformation-Based Learning of Tags

In our description of Markov models we have stressed at several points that the Markov assumptions are too crude for many properties of natural language syntax. The question arises why we do not adopt more sophisticated models. We could condition tags on preceding words (not just preceding tags) or we could use more context than trigram taggers by going to fourgram or even higher order taggers.

This approach is not feasible because of the large number of parameters we would need. Even with trigram taggers, we had to smooth and interpolate because maximum likelihood estimates were not robust enough. This problem would be exacerbated with models more complex than the Markov models introduced so far, especially if we wanted to condition transition probabilities on words.

We will now turn to transformation-based tagging. One of the strengths of this method is that it can exploit a wider range of lexical and syntactic regularities. In particular, tags can be conditioned on words and on more context. Transformation-based tagging encodes complex interdependencies between words and tags by selecting and sequencing transformations that transform an initial imperfect tagging into one with fewer errors. The training of a transformation-based tagger requires an order of magnitude fewer decisions than estimating the large number of parameters of a Markov model.

Transformation-based tagging has two key components:

- a specification of which 'error-correcting' transformations are admissible
- the learning algorithm

As input data, we need a tagged corpus and a dictionary. We first tag each word in the training corpus with its most frequent tag – that is what we need the dictionary for. The learning algorithm then constructs a ranked list of transformations that transforms the initial tagging into a tagging that is close to correct. This ranked list can be used to tag new text, by again initially choosing each word's most frequent tag, and then applying the transformations. We will now describe these components in more detail.

10.4.1 Transformations

A transformation consists of two parts, a triggering environment and a rewrite rule. Rewrite rules have the form $t^1 \rightarrow t^2$, meaning "replace tag t^1 by tag t^2." Brill (1995a) allows the triggering environments shown in table 10.7. Here the asterisk is the site of the potential rewriting and the boxes denote the locations where a trigger will be sought. For example, line 5 refers to the triggering environment "Tag t^j occurs in one of the three previous positions."

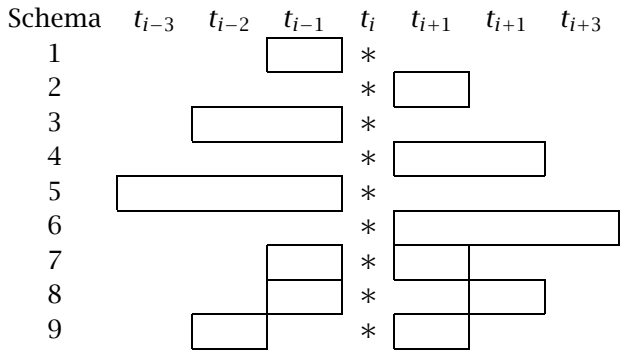

Table 10.7 Triggering environments in Brill's transformation-based tagger. Examples: Line 5 refers to the triggering environment "Tag t^j occurs in one of the three previous positions"; Line 9 refers to the triggering environment "Tag t^j occurs two positions earlier and tag t^k occurs in the following position."

Source tag	Target tag	Triggering environment
NN	VB	previous tag is TO
VBP	VB	one of the previous three tags is MD
JJR	RBR	next tag is JJ
VBP	VB	one of the previous two words is *n't*

Table 10.8 Examples of some transformations learned in transformation-based tagging.

Examples of the type of transformations that are learned given these triggering environments are shown in table 10.8. The first transformation specifies that nouns should be retagged as verbs after the tag TO. Later transformations with more specific triggers will switch some words back to NN (e.g., *school* in *go to school*). The second transformation in table 10.8 applies to verbs with identical base and past tense forms like *cut* and *put*. A preceding modal makes it unlikely that they are used in the past tense. An example for the third transformation is the retagging of *more* in *more valuable player*.

The first three transformations in table 10.8 are triggered by tags. The fourth one is triggered by a word. (In the Penn Treebank words like *don't* and *shouldn't* are split up into a modal and *n't*.) Similar to the second transformation, this one also changes a past tense form to a base form. A preceding *n't* makes a base form more likely than a past tense form.

1 C_0 := corpus with each word tagged with its most frequent tag
3 **for** $k := 0$ **step** 1 **do**
4 $\quad v :=$ the transformation u_i that minimizes $E(u_i(C_k))$
6 $\quad$ **if** $(E(C_k) - E(v(C_k))) < \epsilon$ **then break fi**
7 $\quad C_{k+1} := v(C_k)$
8 $\quad \tau_{k+1} := v$
9 **end**
10 Output sequence: $\tau_1, \ldots, \tau_k$

Figure 10.3 The learning algorithm for transformation-based tagging. C_i refers to the tagging of the corpus in iteration i. E is the error rate.

Word-triggered environments can also be conditioned on the current word and on a combination of words and tags ("the current word is w^i and the following tag is t^j").

There is also a third type of transformation in addition to tag-triggered and word-triggered transformations. *Morphology-triggered transformations* offer an elegant way of integrating the handling of unknown words into the general tagging formalism. Initially, unknown words are tagged as proper nouns (NNP) if capitalized, as common nouns (NN) otherwise. Then morphology-triggered transformations like "Replace NN by NNS if the unknown word's suffix is *-s*" correct errors. These transformations are learned by the same learning algorithm as the tagging transformations proper. We will now describe this learning algorithm.

10.4.2 The learning algorithm

The learning algorithm of transformation-based tagging selects the best transformations and determines their order of application. It works as shown in figure 10.3.

Initially we tag each word with its most frequent tag. In each iteration of the loop, we choose the transformation that reduces the error rate most (line 4), where the error $E(C_k)$ is measured as the number of words that are mistagged in tagged corpus C_k. We stop when there is no transformation left that reduces the error rate by more than a prespecified threshold ϵ. This procedure is a greedy search for the optimal sequence of transformations.

We also have to make two decisions about how to apply the transfor-

mations, that is, how exactly to compute $\tau_i(C_k)$. First, we are going to stipulate that transformations are applied from left to right to the input. Secondly, we have to decide whether transformations should have an immediate or delayed effect. In the case of immediate effect, applications of the same transformation can influence each other. Brill implements delayed-effect transformations, which are simpler. This means that a transformation "A → B if the preceding tag is A" will transform AAAA to ABBB. AAAA would be transformed to ABAB if transformations took effect immediately.

An interesting twist on this tagging model is to use it for *unsupervised* learning as an alternative to HMM tagging. As with HMM tagging, the only information available in unsupervised tagging is which tags are allowable for each word. We can then take advantage of the fact that many words only have one tag and use that as the scoring function for selecting transformations. For example, we can infer that the tagging of *can* in *The can is open* as NN is correct if most unambiguous words in the environment "AT __ BEZ" are nouns with this tag. Brill (1995b) describes a system based on this idea that achieves tagging accuracies of up to 95.6%, a remarkable result for an unsupervised method. What is particularly interesting is that there is no overtraining – in sharp contrast to HMMs which are very prone to overtraining as we saw above. This is a point that we will return to presently.

10.4.3 Relation to other models

Decision trees

Transformation-based learning bears some similarity to decision trees (see section 16.1). We can view a decision tree as a mechanism that labels all leaves that are dominated by a node with the majority class label of that node. As we descend the tree we relabel the leaves of a child node if its label differs from that of the parent node. This way of looking at a decision tree shows the similarity to transformation-based learning where we also go through a series of relabelings, working on smaller and smaller subsets of the data.

In principle, transformation-based learning is strictly more powerful than decision trees as shown by Brill (1995a). That is, there exist classification tasks that can be solved using transformation-based learning that cannot be solved using decision trees. However, it is not clear that

this 'extra power' of transformation-based learning is used in NLP applications.

The main practical difference between the two methods is that the training data are split at each node in a decision tree and that we apply a different sequence of 'transformations' for each node (the sequence corresponding to the decisions on the path from the root to that node). In transformation-based learning, each transformation in the learned transformation list is applied to all the data (leading to a rewriting when the triggering environment is met). As a result, we can directly minimize on the figure of merit that we are most interested in (number of tagging errors in the case of tagging) as opposed to indirect measures like entropy that are used for HMMs and decision trees. If we directly minimized tagging errors in decision tree learning, then it would be easy to achieve 100% accuracy for each leaf node. But performance on new data would be poor because each leaf node would be formed based on arbitrary properties of the training set that don't generalize. Transformation-based learning seems to be surprisingly immune to this form of overfitting (Ramshaw and Marcus 1994). This can be partially explained by the fact that we always learn on the whole data set.

One price we pay for this robustness is that the space of transformation sequences we have to search is huge. A naive implementation of transformation-based learning will therefore be quite inefficient. However, there are ways of searching the space more intelligently and efficiently (Brill 1995a).

Probabilistic models in general

In comparison to probabilistic models (including decision trees), transformation based learning does not make the battery of standard methods available that probability theory provides. For example, no extra work is necessary in a probabilistic model for a 'k-best' tagging – a tagging module that passes a number of tagging hypotheses with probabilities on to the next module downstream (such as the parser).

It is possible to extend transformation-based tagging to 'k-best' tagging by allowing rules of the form "add tag A to B if ..." so that some words will be tagged with multiple tags. However, the problem remains that we don't have an assessment of how likely each of the tags is. The first tag could be 100 times more likely than the next best one in one situation

and all tags could be equally likely in another situation. This type of knowledge could be critical for constructing a parse.

An important characteristic of learning methods is the way prior knowledge can be encoded. Transformation-based tagging and probabilistic approaches have different strengths here. The specification of templates for the most appropriate triggering environments offers a powerful way of biasing the learner towards good generalizations in transformation-based learning. The templates in table 10.7 seem obvious. But they seem obvious only because of what we know about syntactic regularities. A large number of other templates that are obviously inappropriate are conceivable (e.g., "the previous even position in the sentence is a noun").

In contrast, the probabilistic Markov models make it easier to encode precisely what the prior likelihood for the different tags of a word are (for example, the most likely tag is ten times as likely or just one and a half times more likely). The only piece of knowledge we can give the learner in transformation-based tagging is which tag is most likely.

10.4.4 Automata

The reader may wonder why we describe transformation-based tagging in this textbook even though we said we would not cover rule-oriented approaches. While transformation-based tagging has a rule component, it also has a quantitative component. We are somewhat loosely using Statistical NLP in the sense of any corpus-based or quantitative method that uses counts from corpora, not just those that use the framework of probability theory. Transformation-based tagging clearly is a Statistical NLP method in this sense because transformations are selected based on a quantitative criterion.

However, the quantitative evaluation of transformations (by how much they improve the error rate) only occurs during training. Once learning is complete, transformation-based tagging is purely symbolic. That means that a transformation-based tagger can be converted into another symbolic object that is equivalent in terms of tagging performance, but has other advantageous properties like time efficiency.

FINITE STATE TRANSDUCER

This is the approach taken by Roche and Schabes (1995). They convert a transformation-based tagger into an equivalent *finite state transducer*, a finite-state automaton that has a pair of symbols on each arc, one input symbol and one output symbol (in some cases several symbols can be output when an arc is traversed). A finite-state transducer passes over an

input string and converts it into an output string by consuming the input symbols on the arcs it traverses and outputting the output symbols on the same arcs.

LOCAL EXTENSION

The construction algorithm proposed by Roche and Schabes has four steps. First, each transformation is converted into a finite-state transducer. Second, the transducer is converted into its *local extension*. Simply put, the local extension f_2 of a transducer f_1 is constructed such that running f_2 on an input string in one pass has the same effect as running f_1 on each position of the input string. This step takes care of cases like the following. Suppose we have a transducer that implements the transformation "replace A by B if one of the two preceding symbols is C." This transducer will have one arc with the input symbol A and the output symbol B. So for an input sequence like "CAA" we have to run it twice (at the second and third position) to correctly transduce "CAA" to "CBB." The local extension is constructed such that one pass will do this conversion.

In the third step, we compose all transducers into one single transducer whose effect is the same as running the individual transducers in sequence. This single transducer is generally non-deterministic. Whenever this transducer has to keep an event (like "C occurred at position i") in memory it will do this by launching two paths one assuming that a tag affected by a preceding C will occur later, one assuming that no such tag will occur. The appropriate path will be pursued further, the inappropriate path will be 'killed off' at the appropriate position in the string. This type of indeterminism is not efficient, so the fourth step is to convert the non-deterministic transducer into a deterministic one. This is not possible in general since non-deterministic transducers can keep events in memory for an arbitrary long sequence, which cannot be done by deterministic transducers. However, Roche and Schabes show that the transformations used in transformation-based tagging do not give rise to transducers with this property. We can therefore always transform a transformation-based tagger into a deterministic finite-state transducer.

The great advantage of a deterministic finite-state transducer is speed. A transformation-based tagger can take RKn elementary steps to tag a text where R is the number of transformations, K is the length of the triggering environment, and n is the length of the input text (Roche and Schabes 1995: 231). In contrast, finite-state transducers are linear in the length of the input text with a much smaller constant. Basically, we only hop from one state to the next as we read a word, look up its most likely tag (the initial state) and output the correct tag. This makes speeds of

several tens of thousands of words per second possible. The speed of Markov model taggers can be an order of magnitude lower. This means that transducer-based tagging adds a very small overhead to operations like reading the input text from disk and its time demands are likely to be negligible compared to subsequent processing steps like parsing or message understanding.

There has also been work on transforming Hidden Markov models into finite state transducers (Kempe 1997). But, in this case, we cannot achieve complete equivalence since automata cannot perfectly mimic the floating point operations that need to be computed for the Viterbi algorithm.

10.4.5 Summary

The great advantage of transformation-based tagging is that it can condition tagging decisions on a richer set of events than the probabilistic models we looked at earlier. For example, information from the left and right can be used simultaneously and individual words (not just their tags) can influence the tagging of neighboring words. One reason transformation-based tagging can accommodate this richer set of triggering environments is probably that it primarily deals with binary information, which is less complex than probabilities.

It has also been claimed that transformations are easier to understand and modify than the transition and word generation probabilities in probabilistic tagging. However, it can be quite hard to foresee the effect of changing one transformation in a sequence, since complex interactions can occur when several dozen transformations are applied in sequence and each depends on the output of the previous one.

Work on the theoretical foundations of transformation-based tagging is still on-going. For example, the fact that transformation-based learning seems remarkably resistant to overfitting is so far an empirical result that is not well understood.

Even so, both learning and tagging are remarkably simple and intuitive in transformation-based tagging. Whether this simplicity is the principal criterion for choosing between a transformation-based or a probabilistic tagger, or whether the strength of probabilistic models in dealing with uncertainty and certain types of prior knowledge are more important considerations will depend on many factors such as what type of system the tagger is a component of and whether those working on this system are more comfortable with rule-based or probabilistic approaches.

Apart from tagging, transformation-based learning has also been applied to parsing (Brill 1993b), prepositional phrase attachment (Brill and Resnik 1994), and word sense disambiguation (Dini et al. 1998).

Exercise 10.16 [⋆]

Transformation-based learning is a form of greedy search. Is greedy search expected to find the optimal sequence of transformations? What would be alternatives?

Exercise 10.17 [⋆]

Most of the triggering environments in Brill (1995a) refer to preceding context. Why? Would you expect the same tendency for languages other than English?

Exercise 10.18 [⋆]

The set of possible triggering environments for words and tags is different in (Brill 1995a). For example, "one of the three preceding tags is X" is admissible as a triggering environment, but not "one of the three preceding words is X." What might be the reason for this difference? Consider the differences between the sizes of the search spaces for words and tags.

Exercise 10.19 [⋆]

Apart from choosing the most frequent tag as initialization, we can also assign all words to the same tag (say, NN) or use the output of another tagger which the transformation-based tagger can then improve. Discuss relative advantages of different initializations.

Exercise 10.20 [⋆⋆]

Get the Brill tagger (see pointer on website) and tag texts from the website.

10.5 Other Methods, Other Languages

10.5.1 Other approaches to tagging

Tagging has been one of the most active areas of research in NLP in the last ten years. We were only able to cover three of the most important approaches here. Many other probabilistic and quantitative methods have been applied to tagging, including all the methods we cover in chapter 16: neural networks (Benello et al. 1989), decision trees (Schmid 1994), memory-based learning (or k nearest neighbor approaches) (Daelemans et al. 1996), and maximum entropy models (Ratnaparkhi 1996).[4]

4. Ratnaparkhi's tagger, one of the highest performing statistical taggers, is publicly available. See the website.

There has also been work on how to construct a tagged corpus with a minimum of human effort (Brill et al. 1990). This problem poses itself when a language with as yet no tagged training corpus needs to be tackled or when in the case of already tagged languages we encounter text that is so different as to make existing tagged corpora useless.

Finally, some researchers have explored ways of constructing a tag set automatically in order to create syntactic categories that are appropriate for a language or a particular text sort (Schütze 1995; McMahon and Smith 1996).

10.5.2 Languages other than English

We have only covered part-of-speech tagging of English here. It turns out that English is a particularly suitable language for methods that try to infer a word's grammatical category from its position in a sequence of words. In many other languages, word order is much freer, and the surrounding words will contribute much less information about part of speech. However, in most such languages, the rich inflections of a word contribute more information about part of speech than happens in English. A full evaluation of taggers as useful preprocessors for high-level multilingual NLP tasks will only be possible after sufficient experimental results from a wide range of languages are available.

Despite these reservations, there exist now quite a number of tagging studies, at least for European languages. These studies suggest that the accuracy for other languages is comparable with that for English (Dermatas and Kokkinakis 1995; Kempe 1997), although it is hard to make such comparisons due to the incomparability of tag sets (tag sets are not universal, but all encode the particular functional categories of individual languages).

10.6 Tagging Accuracy and Uses of Taggers

10.6.1 Tagging accuracy

Accuracy numbers currently reported for tagging are most often in the range of 95% to 97%, when calculated over all words. Some authors give accuracy for ambiguous words only, in which case the accuracy figures are of course lower. However, performance depends considerably on factors such as the following.

- **The amount of training data available.** In general, the more the better.
- **The tag set.** Normally, the larger the tag set, the more potential ambiguity, and the harder the tagging task (but see the discussion in section 4.3.2). For example, some tag sets make a distinction between the preposition *to* and the infinitive marker *to*, and some don't. Using the latter tag set, one can't tag *to* wrongly.
- **The difference between training corpus and dictionary on the one hand and the corpus of application on the other.** If training and application text are drawn from the same source (for example, the same time period of a particular newspaper), then accuracy will be high. Normally the only results presented for taggers in research papers present results from this situation. If the application text is from a later time period, from a different source, or even from a different genre than the training text (e.g., scientific text vs. newspaper text), then performance can be poor.[5]
- **Unknown words.** A special case of the last point is coverage of the dictionary. The occurrence of many unknown words will greatly degrade performance. The percentage of words not in the dictionary can be very high when trying to tag material from some technical domain.

A change in any of these four conditions will impact tagging accuracy, sometimes dramatically. If the training set is small, the tag set large, the test corpus significantly different from the training corpus, or we are confronted with a larger than expected number of unknown words, then performance can be far below the performance range cited above. It is important to stress that these types of external conditions often have a stronger influence on performance than the choice of tagging method – especially when differences between methods reported are on the order of half a percent.

The influence of external factors also needs to be considered when we evaluate the surprisingly high performance of a 'dumb' tagger which always chooses a word's most frequent tag. Such a tagger can get an accuracy of about 90% in favorable conditions (Charniak et al. 1993). This high number is less surprising when we learn that the dictionary that was used in (Charniak et al. 1993) is based on the corpus of application, the

5. See Elworthy (1994) and Samuelsson and Voutilainen (1997) for experiments looking at performance for different degrees of similarity to the training set.

Brown corpus. Considerable manual effort went into the resources that make it now easy to determine what the most frequent tag for a word in the Brown corpus is. So it is not surprising that a tagger exploiting this dictionary information does well. The automatic tagger that was originally used to preprocess the Brown corpus only achieved 77% accuracy (Greene and Rubin 1971). In part this was due to its non-probabilistic nature, but in large part this was due to the fact that it could not rely on a large dictionary giving the frequency with which words are used in different parts of speech that was suitable for the corpus of application.

Even in cases where we have a good dictionary and the most-frequent-tag strategy works well, it is still important how well a tagger does in the range from 90% correct to 100% correct. For example, a tagger with 97% accuracy has a 63% chance of getting all tags in a 15-word sentence right, compared to 74% for a tagger with 98% accuracy. So even small improvements can make a significant difference in an application.

ENGLISH CONSTRAINT GRAMMAR

One of the best-performing tagging formalisms is non-quantitative: EngCG (*English Constraint Grammar*), developed at the University of Helsinki. Samuelsson and Voutilainen (1997) show that it performs better than Markov model taggers, especially if training and test corpora are not from the same source.[6] In EngCG, hand-written rules are compiled into finite-state automata (Karlsson et al. 1995; Voutilainen 1995). The basic idea is somewhat similar to transformation-based learning, except that a human being (instead of an algorithm) iteratively modifies a set of tagging rules so as to minimize the error rate. In each iteration, the current rule set is run on the corpus and an attempt is made to modify the rules so that the most serious errors are handled correctly. This methodology amounts to writing a small expert system for tagging. The claim has been made that for somebody who is familiar with the methodology, writing this type of tagger takes no more effort than building an HMM tagger (Chanod and Tapanainen 1995), though it could be argued that the methodology for HMM tagging is more easily accessible.

We conclude our remarks on tagging accuracy by giving examples of some of the most frequent errors. Table 10.9 shows some examples of common error types reported by Kupiec (1992b). The example phrases and fragments are all ambiguous, demonstrating that semantic context,

6. The accuracy figures for EngCG reported in the paper are better than 99% vs. better than 95% for a Markov Model tagger, but comparison is difficult since some ambiguities are not resolved by EngCG. EngCG returns a set of more than one tag in some cases.

Correct tag	Tagging error	Example
noun singular	adjective	*an executive order*
adjective	adverb	*more important issues*
preposition	particle	*He ran up a big ...*
past tense	past participle	*loan needed to meet*
past participle	past tense	*loan needed to meet*

Table 10.9 Examples of frequent errors of probabilistic taggers.

or more syntactic context is necessary than a Markov model has access to. Syntactically, the word *executive* could be an adjective as well as a noun. The phrase *more important issues* could refer to a larger number of important issues or to issues that are more important. The word *up* is used as a preposition in *running up a hill*, as a particle in *running up a bill*. Finally, depending on the embedding, *needed* can be a past participle or a past tense form as the following two sentences from (Kupiec 1992b) show:

(10.12) a. The loan needed to meet rising costs of health care.

b. They cannot now handle the loan needed to meet rising costs of health care.

CONFUSION MATRIX

Table 10.10 shows a portion of a *confusion matrix* for the tagger described in (Franz 1995). Each row shows the percentage of the time words of a certain category were given different tags by the tagger. In a way the results are unsurprising. The errors occur in the cases where multiple class memberships are common. Particularly to be noted, however, is the low accuracy of tagging particles, which are all word types that can also act as prepositions. The distinction between particles and prepositions, while real, is quite subtle, and some people feel that it is not made very accurately even in hand-tagged corpora.[7]

10.6.2 Applications of tagging

The widespread interest in tagging is founded on the belief that many NLP applications will benefit from syntactically disambiguated text. Given this

7. Hence, as we shall see in chapter 12, it is often ignored in the evaluation of probabilistic parsers.

Correct Tags	Tags assigned by the tagger							
	DT	IN	JJ	NN	RB	RP	VB	VBG
DT	99.4	.3			.3			
IN	.4	97.5			1.5	.5		
JJ		.1	93.9	1.8	.9		.1	.4
NN			2.2	95.5			.2	.4
RB	.2	2.4	2.2	.6	93.2	1.2		
RP		24.7		1.1	12.6	61.5		
VB			.3	1.4			96.0	
VBG			2.5	4.4				93.0

Table 10.10 A portion of a confusion matrix for part of speech tagging. For each tag, a row of the table shows the percentage of the time that the tagger assigned tokens of that category to different tags. (Thus, in the full confusion matrix, the percentages in each row would add to 100%, but do not do so here, because only a portion of the table is shown.). Based on (Franz 1995).

ultimate motivation for part-of-speech tagging, it is surprising that there seem to be more papers on stand-alone tagging than on applying tagging to a task of immediate interest. We summarize here the most important applications for which taggers have been used.

PARTIAL PARSING

Most applications require an additional step of processing after tagging: *partial parsing*. Partial parsing can refer to various levels of detail of syntactic analysis. The simplest partial parsers are limited to finding the noun phrases of a sentence. More sophisticated approaches assign grammatical functions to noun phrases (subject, direct object, indirect object) and give partial information on attachments, for example, 'this noun phrase is attached to another (unspecified) phrase to the right'.

There is an elegant way of using Markov models for noun phrase recognition (see (Church 1988), but a better description can be found in (Abney 1996a)). We can take the output of the tagger and form a sequence of tag bigrams. For example, NN VBZ RB AT NN would be transformed into NN-VBZ VBZ-RB RB-AT AT-NN. This sequence of tag bigrams is then tagged with five symbols: noun-phrase-beginning, noun-phrase-end, noun-phrase-interior, noun-phrase-exterior (that is, this tag bigram is not part of a noun phrase), and between-noun-phrases (that is, at the position of this tag bigram there is a noun phrase immediately to the right and a noun phrase immediately to the left). The noun phrases

are then all sequences of tags between a noun-phrase-beginning symbol (or a between-noun-phrases symbol) and a noun-phrase-end symbol (or a between-noun-phrases symbol), with noun-phrase-interior symbols in between.

The best known approaches to partial parsing are Fidditch, developed in the early eighties by Hindle (1994), and an approach called "parsing by chunks" developed by Abney (1991). These two systems do not use taggers because they predate the widespread availability of taggers. See also (Grefenstette 1994). Two approaches that are more ambitious than current partial parsers and attempt to bridge the gap between shallow and full parsing are the XTAG system (Doran et al. 1994) and chunk tagging (Brants and Skut 1998; Skut and Brants 1998).

In many systems that build a partial parser on top of a tagger, partial parsing is accomplished by way of regular expression matching over the output of the tagger. For example, a simple noun phrase may be defined as a sequence of article (AT), an arbitrary number of adjectives (JJ) and a singular noun (NN). This would correspond to the regular expression "AT JJ* NN." Since these systems focus on the final application, not on partial parsing we cover them in what follows not under the rubric "partial parsing," but grouped according to the application they are intended for. For an excellent overview of partial parsing (and tagging) see (Abney 1996a).

One important use of tagging in conjunction with partial parsing is for *lexical acquisition*. We refer the reader to chapter 8.

INFORMATION EXTRACTION

Another important application is *information extraction* (which is also referred to as message understanding, data extraction, or text data mining). The goal in information extraction is to find values for the predefined slots of a template. For example, a template for weather reporting might have slots for the type of weather condition (tornado, snow storm), the location of the event (the San Francisco Bay Area), the time of the event (Sunday, January 11, 1998), and what the effect of the event was (power outage, traffic accidents, etc.). Tagging and partial parsing help identify the entities that serve as slot fillers and the relationships between them. A recent overview article on information extraction is (Cardie 1997). In a way one could think of information extraction as like tagging except that the tags are semantic categories, not grammatical parts of speech. However, in practice quite different techniques tend to be employed, because local sequences give less information about semantic categories than grammatical categories.

Tagging and partial parsing can also be applied to finding good in-

dexing terms in information retrieval. The best unit for matching user queries and documents is often not the individual word. Phrases like *United States of America* and *secondary education* lose much of their meaning if they are broken up into words. Information retrieval performance can be improved if tagging and partial parsing are applied to noun phrase recognition and query-document matching is done on more meaningful units than individual terms (Fagan 1987; Smeaton 1992; Strzalkowski 1995). A related area of research is phrase normalization in which variants of terms are normalized and represented as the same basic unit (for example, *book publishing* and *publishing of books*). See (Jacquemin et al. 1997).

QUESTION ANSWERING

Finally, there has been work on so-called *question answering* systems which try to answer a user query that is formulated in the form of a question by returning an appropriate noun phrase such as a location, a person, or a date (Kupiec 1993b; Burke et al. 1997). For example, the question *Who killed President Kennedy?* might be answered with the noun phrase *Oswald* instead of returning a list of documents as most information retrieval systems do. Again, analyzing a query in order to determine what type of entity the user is looking for and how it is related to other noun phrases mentioned in the question requires tagging and partial parsing.

We conclude with a negative result: the best lexicalized probabilistic parsers are now good enough that they perform better starting with untagged text and doing the tagging themselves, rather than using a tagger as a preprocessor (Charniak 1997a). Therefore, the role of taggers appears to be as a fast lightweight component that gives sufficient information for many application tasks, rather than as a desirable preprocessing stage for all applications.

10.7 Further Reading

Early work on modeling natural language using Markov chains had been largely abandoned by the early sixties, partially due to Chomsky's criticism of the inadequacies of Markov models (Chomsky 1957: ch. 3). The lack of training data and computing resources to pursue an 'empirical' approach to natural language probably also played a role. Chomsky's criticism still applies: Markov chains cannot fully model natural language, in particular they cannot model many recursive structures (but cf. Ristad and Thomas (1997)). What has changed is that approaches that

emphasize technical goals such as solving a particular task have become acceptable even if they are not founded on a theory that fully explains language as a cognitive phenomenon.

The earliest 'taggers' were simply programs that looked up the category of words in a dictionary. The first well-known program which attempted to assign tags based on syntagmatic contexts was the rule-based program presented in (Klein and Simmons 1963), though roughly the same idea is present in (Salton and Thorpe 1962). Klein and Simmons use the terms 'tags' and 'tagging,' though apparently interchangeably with 'codes' and 'coding.' The earliest probabilistic tagger known to us is (Stolz et al. 1965). This program initially assigned tags to some words (including all function words) via use of a lexicon, morphology rules, and other ad-hoc rules. The remaining open class words were then tagged using conditional probabilities calculated from tag sequences. Needless to say, this wasn't a well-founded probabilistic model.

Credit has to be given to two groups, one at Brown University, one at the University of Lancaster, who spent enormous resources to tag two large corpora, the Brown corpus and the Lancaster-Oslo-Bergen (LOB) corpus. Both groups recognized how invaluable a corpus annotated with tag information would be for further corpus research. Without these two tagged corpora, progress on part-of-speech tagging would have been hard if not impossible. The availability of a large quantity of tagged data is no doubt an important reason that tagging has been such an active area of research.

The Brown corpus was automatically pre-tagged with a rule-based tagger, TAGGIT (Greene and Rubin 1971). This tagger used lexical information only to limit the tags of words and only applied tagging rules when words in the surrounding context were unambiguously tagged. The output of the tagger was then manually corrected in an effort that took many years and supplied the training data for a lot of the quantitative work that was done later.

One of the first Markov Model taggers was created at the University of Lancaster as part of the LOB tagging effort (Garside et al. 1987; Marshall 1987). The heart of this tagger was the use of bigram tag sequence probabilities, with limited use of higher order context, but the differing probabilities of assigning a word to different parts of speech were handled by ad hoc discounting factors. The type of Markov Model tagger that tags based on both word probabilities and tag transition probabilities was introduced by Church (1988) and DeRose (1988).

During the beginning of the resurgence of quantitative methods in NLP, the level of knowledge of probability theory in the NLP community was so low that a frequent error in early papers is to compute the probability of the next tag in a Markov model as (10.14) instead of (10.13). At first sight, (10.14) can seem more intuitive. After all, we are looking at a word and want to determine its tag, so it is not far-fetched to assume the word as given and the tag as being conditioned on the word.

$$\arg\max_{t_{1,n}} P(t_{1,n}|w_{1,n}) = \prod_{i=1}^{n} [P(w_i|t_i) \times P(t_i|t_{i-1})] \tag{10.13}$$

$$\arg\max_{t_{1,n}} P(t_{1,n}|w_{1,n}) = \prod_{i=1}^{n} [P(t_i|w_i) \times P(t_i|t_{i-1})] \tag{10.14}$$

But, actually, equation (10.14) is not correct, and use of it results in lower performance (Charniak et al. 1993).

While the work of Church and DeRose was key in the resurgence of statistical methods in computational linguistics, work on hidden Markov model tagging had actually begun much earlier at the IBM research centers in New York state and Paris. Jelinek (1985) and Derouault and Merialdo (1986) are widely cited. Earlier references are Bahl and Mercer (1976) and Baker (1975), who attributes the work to Elaine Rich. Other early work in probabilistic tagging includes (Eeg-Olofsson 1985; Foster 1991).

DENOMINAL VERBS

A linguistic discussion of the conversion of nouns to verbs (*denominal verbs*) and its productivity can be found in Clark and Clark (1979). Huddleston (1984: ch. 3) contains a good discussion of the traditional definitions of parts of speech, their failings, and the notion of part of speech or word class as used in modern structuralist linguistics.

10.8 Exercises

Exercise 10.21 [⋆]

Usually, the text we want to tag is not segmented into sentences. An algorithm for identifying sentence boundaries is introduced in chapter 4, together with a general overview of the nitty-gritty of corpus processing.

Could we integrate sentence-boundary detection into the tagging methods introduced in this chapter? What would we have to change? How effective would you expect sentence-boundary detection by means of tagging to be?

Exercise 10.22 [⋆⋆]

Get the MULTEXT tagger (see the website) and tag some non-English text from the website.