Chapter 5
Part-of-Speech Tagging

Conjunction Junction, what's your function?
Bob Dorough, *Schoolhouse Rock, 1973*

A gnostic was seated before a grammarian. The grammarian said, 'A word must be one of three things: either it is a noun, a verb, or a particle.' The gnostic tore his robe and cried, "Alas! Twenty years of my life and striving and seeking have gone to the winds, for I laboured greatly in the hope that there was another word outside of this. Now you have destroyed my hope.' Though the gnostic had already attained the word which was his purpose, he spoke thus in order to arouse the grammarian.
Rumi (1207–1273), *The Discourses of Rumi*, Translated by A. J. Arberry

Dionysius Thrax of Alexandria (*c.* 100 B.C.), or perhaps someone else (exact authorship being understandably difficult to be sure of with texts of this vintage), wrote a grammatical sketch of Greek (a "*technē*") which summarized the linguistic knowledge of his day. This work is the direct source of an astonishing proportion of our modern linguistic vocabulary, including among many other words, *syntax*, *diphthong*, *clitic*, and *analogy*. Also included are a description of eight **parts-of-speech**: noun, verb, pronoun, preposition, adverb, conjunction, participle, and article. Although earlier scholars (including Aristotle as well as the Stoics) had their own lists of parts-of-speech, it was Thrax's set of eight which became the basis for practically all subsequent part-of-speech descriptions of Greek, Latin, and most European languages for the next 2000 years.

parts-of-speech

Schoolhouse Rock was a popular series of 3-minute musical animated clips first aired on television in 1973. The series was designed to inspire kids to learn multiplication tables, grammar, and basic science and history. The Grammar Rock sequence, for example, included songs about parts-of-speech, thus bringing these categories into the realm of popular culture. As it happens, Grammar Rock was remarkably traditional in its grammatical notation, including exactly eight songs about parts-of-speech. Although the list was slightly modified from Thrax's original, substituting adjective and interjection for the original participle and article, the astonishing durability of the parts-of-speech through two millenia is an indicator of both the importance and the transparency of their role in human language.

tagsets

More recent lists of parts-of-speech (or **tagsets**) have many more word classes; 45 for the Penn Treebank (Marcus et al., 1993), 87 for the Brown corpus (Francis, 1979; Francis and Kučera, 1982), and 146 for the C7 tagset (Garside et al., 1997).

POS

The significance of parts-of-speech (also known as **POS**, **word classes**, **morphological classes**, or **lexical tags**) for language processing is the large amount of information they give about a word and its neighbors. This is clearly true for major categories, (**verb** versus **noun**), but is also true for the many finer distinctions. For example these tagsets distinguish between possessive pronouns (*my*, *your*, *his*, *her*, *its*) and personal pronouns (*I*, *you*, *he*, *me*). Knowing whether a word is a possessive pronoun or a personal pronoun can tell us what words are likely to occur in its vicinity (possessive pronouns are likely to be followed by a noun, personal pronouns by a verb). This can be useful in a language model for speech recognition.

A word's part-of-speech can tell us something about how the word is pronounced. As Ch. 8 will discuss, the word *content*, for example, can be a noun or an adjective. They are pronounced differently (the noun is pronounced *CONtent* and the adjective *conTENT*). Thus knowing the part-of-speech can produce more natural pronunciations in a speech synthesis system and more accuracy in a speech recognition system. (Other pairs like this include *OBject* (noun) and *obJECT* (verb), *DIScount* (noun) and *disCOUNT* (verb); see Cutler (1986)).

Parts-of-speech can also be used in stemming for informational retrieval (IR), since knowing a word's part-of-speech can help tell us which morphological affixes it can take, as we saw in Ch. 3. They can also enhance an IR application by selecting out nouns or other important words from a document. Automatic assignment of part-of-speech plays a role in parsing, in word-sense disambiguation algorithms, and in shallow parsing of texts to quickly find names, times, dates, or other named entities for the information extraction applications discussed in Ch. 22. Finally, corpora that have been marked for parts-of-speech are very useful for linguistic research. For example, they can be used to help find instances or frequencies of particular constructions.

This chapter focuses on computational methods for assigning parts-of-speech to words (**part-of-speech tagging**). Many algorithms have been applied to this problem, including hand-written rules (**rule-based tagging**), probabilistic methods (**HMM tagging** and **maximum entropy tagging**), as well as other methods such as **transformation-based tagging** and **memory-based tagging**. We will introduce three of these algorithms in this chapter: rule-based tagging, HMM tagging, and transformation-based tagging. But before turning to the algorithms themselves, let's begin with a summary of English word classes, and of various tagsets for formally coding these classes.

5.1 (Mostly) English Word Classes

Until now we have been using part-of-speech terms like **noun** and **verb** rather freely. In this section we give a more complete definition of these and other classes. Traditionally the definition of parts-of-speech has been based on syntactic and morphological function; words that function similarly with respect to what can occur nearby (their "syntactic distributional properties"), or with respect to the affixes they take (their morphological properties) are grouped into classes. While word classes do have tendencies toward semantic coherence (nouns do in fact often describe "people, places or things", and adjectives often describe properties), this is not necessarily the case, and in general we don't use semantic coherence as a definitional criterion for parts-of-speech.

closed class
open class

Parts-of-speech can be divided into two broad supercategories: **closed class** types and **open class** types. Closed classes are those that have relatively fixed membership.

For example, prepositions are a closed class because there is a fixed set of them in English; new prepositions are rarely coined. By contrast nouns and verbs are open classes because new nouns and verbs are continually coined or borrowed from other languages (e.g., the new verb *to fax* or the borrowed noun *futon*). It is likely that any given speaker or corpus will have different open class words, but all speakers of a language, and corpora that are large enough, will likely share the set of closed class words. Closed class words are also generally **function words** like *of*, *it*, *and*, or *you*, which tend to be very short, occur frequently, and often have structuring uses in grammar.

function words

There are four major open classes that occur in the languages of the world; **nouns**, **verbs**, **adjectives**, and **adverbs**. It turns out that English has all four of these, although not every language does.

Noun

Noun is the name given to the syntactic class in which the words for most people, places, or things occur. But since syntactic classes like **noun** are defined syntactically and morphologically rather than semantically, some words for people, places, and things may not be nouns, and conversely some nouns may not be words for people, places, or things. Thus nouns include concrete terms like *ship* and *chair*, abstractions like *bandwidth* and *relationship*, and verb-like terms like *pacing* as in *His pacing to and fro became quite annoying*. What defines a noun in English, then, are things like its ability to occur with determiners (*a goat*, *its bandwidth*, *Plato's Republic*), to take possessives (*IBM's annual revenue*), and for most but not all nouns, to occur in the plural form (*goats*, *abaci*).

proper nouns
common nouns

Nouns are traditionally grouped into **proper nouns** and **common nouns**. Proper nouns, like *Regina*, *Colorado*, and *IBM*, are names of specific persons or entities. In English, they generally aren't preceded by articles (e.g., *the book is upstairs*, but *Regina is upstairs*). In written English, proper nouns are usually capitalized.

count nouns
mass nouns

In many languages, including English, common nouns are divided into **count nouns** and **mass nouns**. Count nouns are those that allow grammatical enumeration; that is, they can occur in both the singular and plural (*goat/goats*, *relationship/relationships*) and they can be counted (*one goat*, *two goats*). Mass nouns are used when something is conceptualized as a homogeneous group. So words like *snow*, *salt*, and *communism* are not counted (i.e., **two snows* or **two communisms*). Mass nouns can also appear without articles where singular count nouns cannot (*Snow is white* but not **Goat is white*).

verb

The **verb** class includes most of the words referring to actions and processes, including main verbs like *draw*, *provide*, *differ*, and *go*. As we saw in Ch. 3, English verbs have a number of morphological forms (non-3rd-person-sg (*eat*), 3rd-person-sg (*eats*), progressive (*eating*), past participle (*eaten*)). A subclass of English verbs called **auxiliaries** will be discussed when we turn to closed class forms.

auxiliaries

While many researchers believe that all human languages have the categories of noun and verb, others have argued that some languages, such as Riau Indonesian and Tongan, don't even make this distinction (Broschart, 1997; Evans, 2000; Gil, 2000).

The third open class English form is adjectives; semantically this class includes many terms that describe properties or qualities. Most languages have adjectives for the concepts of color (*white*, *black*), age (*old*, *young*), and value (*good*, *bad*), but there are languages without adjectives. In Korean, for example, the words corresponding to English adjectives act as a subclass of verbs, so what is in English an adjective 'beautiful' acts in Korean like a verb meaning 'to be beautiful' (Evans, 2000).

adverbs

The final open class form, **adverbs**, is rather a hodge-podge, both semantically and formally. For example Schachter (1985) points out that in a sentence like the following, all the italicized words are adverbs:

Unfortunately, John walked *home extremely slowly yesterday*

locative

degree

manner

temporal

What coherence the class has semantically may be solely that each of these words can be viewed as modifying something (often verbs, hence the name "adverb", but also other adverbs and entire verb phrases). **Directional adverbs** or **locative adverbs** (*home*, *here*, *downhill*) specify the direction or location of some action; **degree adverbs** (*extremely*, *very*, *somewhat*) specify the extent of some action, process, or property; **manner adverbs** (*slowly*, *slinkily*, *delicately*) describe the manner of some action or process; and **temporal adverb** describe the time that some action or event took place (*yesterday*, *Monday*). Because of the heterogeneous nature of this class, some adverbs (for example temporal adverbs like *Monday*) are tagged in some tagging schemes as nouns.

The closed classes differ more from language to language than do the open classes. Here's a quick overview of some of the more important closed classes in English, with a few examples of each:

- **prepositions:** on, under, over, near, by, at, from, to, with
- **determiners:** a, an, the
- **pronouns:** she, who, I, others
- **conjunctions:** and, but, or, as, if, when
- **auxiliary verbs:** can, may, should, are
- **particles:** up, down, on, off, in, out, at, by,
- **numerals:** one, two, three, first, second, third

prepositions

Prepositions occur before noun phrases; semantically they are relational, often indicating spatial or temporal relations, whether literal (*on it*, *before then*, *by the house*) or metaphorical (*on time*, *with gusto*, *beside herself*). But they often indicate other relations as well (*Hamlet was written by Shakespeare*, and [from Shakespeare] "*And I did laugh sans intermission an hour by his dial*"). Fig. 5.1 shows the prepositions of English according to the CELEX on-line dictionary (Baayen et al., 1995), sorted by their frequency in the COBUILD 16 million word corpus of English. Fig. 5.1 should not be considered a definitive list, since different dictionaries and tagsets label word classes differently. Furthermore, this list combines prepositions and particles.

particle

phrasal verb

A **particle** is a word that resembles a preposition or an adverb, and is used in combination with a verb. When a verb and a particle behave as a single syntactic and/or semantic unit, we call the combination a **phrasal verb**. Phrasal verbs can behave as a semantic unit; thus they often have a meaning that is not predictable from the separate meanings of the verb and the particle. Thus *turn down* means something like 'reject', *rule out* means 'eliminate', *find out* is 'discover', and *go on* is 'continue'; these are not meanings that could have been predicted from the meanings of the verb and the particle independently. Here are some examples of phrasal verbs from Thoreau:

of	540,085	through	14,964	worth	1,563	pace	12
in	331,235	after	13,670	toward	1,390	nigh	9
for	142,421	between	13,275	plus	750	re	4
to	125,691	under	9,525	till	686	mid	3
with	124,965	per	6,515	amongst	525	o'er	2
on	109,129	among	5,090	via	351	but	0
at	100,169	within	5,030	amid	222	ere	0
by	77,794	towards	4,700	underneath	164	less	0
from	74,843	above	3,056	versus	113	midst	0
about	38,428	near	2,026	amidst	67	o'	0
than	20,210	off	1,695	sans	20	thru	0
over	18,071	past	1,575	circa	14	vice	0

Figure 5.1 Prepositions (and particles) of English from the CELEX on-line dictionary. Frequency counts are from the COBUILD 16 million word corpus.

So I *went on* for some days cutting and hewing timber...
Moral reform is the effort to *throw off* sleep...

Particles don't always occur with idiomatic phrasal verb semantics; here are more examples of particles from the Brown corpus:

...she had turned the paper *over*.
He arose slowly and brushed himself *off*.
He packed *up* his clothes.

We show in Fig. 5.1 a list of single-word particles from Quirk et al. (1985). Since it is extremely hard to automatically distinguish particles from prepositions, some tagsets (like the one used for CELEX) do not distinguish them, and even in corpora that do (like the Penn Treebank) the distinction is very difficult to make reliably in an automatic process, so we do not give counts.

aboard	aside	besides	forward(s)	opposite	through
about	astray	between	home	out	throughout
above	away	beyond	in	outside	together
across	back	by	inside	over	under
ahead	before	close	instead	overhead	underneath
alongside	behind	down	near	past	up
apart	below	east, etc.	off	round	within
around	beneath	eastward(s),etc.	on	since	without

Figure 5.2 English single-word particles from Quirk et al. (1985).

determiners
articles

A closed class that occurs with nouns, often marking the beginning of a noun phrase, is the **determiners**. One small subtype of determiners is the **articles**: English has three articles: *a*, *an*, and *the*. Other determiners include *this* (as in *this chapter*) and *that* (as in *that page*). *A* and *an* mark a noun phrase as indefinite, while *the* can mark it as definite; definiteness is a discourse and semantic property that will be discussed in Ch. 21. Articles are quite frequent in English; indeed *the* is the most frequently occurring word in most corpora of written English. Here are COBUILD statistics, again out of 16 million words:

the: 1,071,676 a: 413,887 an: 59,359

conjunctions

Conjunctions are used to join two phrases, clauses, or sentences. Coordinating conjunctions like *and*, *or*, and *but*, join two elements of equal status. Subordinating conjunctions are used when one of the elements is of some sort of embedded status. For example *that* in *"I thought that you might like some milk"* is a subordinating conjunction that links the main clause *I thought* with the subordinate clause *you might like some milk*. This clause is called subordinate because this entire clause is the "content" of the main verb *thought*. Subordinating conjunctions like *that* which link a verb to its argument in this way are also called **complementizers**. Ch. 12 and Ch. 16 will discuss complementation in more detail. Table 5.3 lists English conjunctions.

complementizers

and	514,946	yet	5,040	considering	174	forasmuch as	0
that	134,773	since	4,843	lest	131	however	0
but	96,889	where	3,952	albeit	104	immediately	0
or	76,563	nor	3,078	providing	96	in as far as	0
as	54,608	once	2,826	whereupon	85	in so far as	0
if	53,917	unless	2,205	seeing	63	inasmuch as	0
when	37,975	why	1,333	directly	26	insomuch as	0
because	23,626	now	1,290	ere	12	insomuch that	0
so	12,933	neither	1,120	notwithstanding	3	like	0
before	10,720	whenever	913	according as	0	neither nor	0
though	10,329	whereas	867	as if	0	now that	0
than	9,511	except	864	as long as	0	only	0
while	8,144	till	686	as though	0	provided that	0
after	7,042	provided	594	both and	0	providing that	0
whether	5,978	whilst	351	but that	0	seeing as	0
for	5,935	suppose	281	but then	0	seeing as how	0
although	5,424	cos	188	but then again	0	seeing that	0
until	5,072	supposing	185	either or	0	without	0

Figure 5.3 Coordinating and subordinating conjunctions of English from CELEX. Frequency counts are from COBUILD (16 million words).

Pronouns
Personal
Possessive
Wh

Pronouns are forms that often act as a kind of shorthand for referring to some noun phrase or entity or event. **Personal pronouns** refer to persons or entities (*you*, *she*, *I*, *it*, *me*, etc.). **Possessive pronouns** are forms of personal pronouns that indicate either actual possession or more often just an abstract relation between the person and some object (*my, your, his, her, its, one's, our, their*). **Wh-pronouns** (*what, who, whom, whoever*) are used in certain question forms, or may also act as complementizers (*Frieda, who I met five years ago* . . .). Table 5.1 shows English pronouns, again from CELEX.

auxiliary

A closed class subtype of English verbs are the **auxiliary** verbs. Crosslinguistically, auxiliaries are words (usually verbs) that mark certain semantic features of a main verb, including whether an action takes place in the present, past or future (tense), whether it is completed (aspect), whether it is negated (polarity), and whether an action is necessary, possible, suggested, desired, etc. (mood).

copula
modal

English auxiliaries include the **copula** verb *be*, the two verbs *do* and *have*, along with their inflected forms, as well as a class of **modal verbs**. *Be* is called a copula because it connects subjects with certain kinds of predicate nominals and adjectives (*He is a duck*). The verb *have* is used for example to mark the perfect tenses (*I have gone*, *I had gone*), while *be* is used as part of the passive (*We were robbed*), or progressive (*We are leaving*) constructions. The modals are used to mark the mood associated with the event or action depicted by the main verb. So *can* indicates ability or possibility,

it	199,920	how	13,137	yourself	2,437	no one	106
I	198,139	another	12,551	why	2,220	wherein	58
he	158,366	where	11,857	little	2,089	double	39
you	128,688	same	11,841	none	1,992	thine	30
his	99,820	something	11,754	nobody	1,684	summat	22
they	88,416	each	11,320	further	1,666	suchlike	18
this	84,927	both	10,930	everybody	1,474	fewest	15
that	82,603	last	10,816	ourselves	1,428	thyself	14
she	73,966	every	9,788	mine	1,426	whomever	11
her	69,004	himself	9,113	somebody	1,322	whosoever	10
we	64,846	nothing	9,026	former	1,177	whomsoever	8
all	61,767	when	8,336	past	984	wherefore	6
which	61,399	one	7,423	plenty	940	whereat	5
their	51,922	much	7,237	either	848	whatsoever	4
what	50,116	anything	6,937	yours	826	whereon	2
my	46,791	next	6,047	neither	618	whoso	2
him	45,024	themselves	5,990	fewer	536	aught	1
me	43,071	most	5,115	hers	482	howsoever	1
who	42,881	itself	5,032	ours	458	thrice	1
them	42,099	myself	4,819	whoever	391	wheresoever	1
no	33,458	everything	4,662	least	386	you-all	1
some	32,863	several	4,306	twice	382	additional	0
other	29,391	less	4,278	theirs	303	anybody	0
your	28,923	herself	4,016	wherever	289	each other	0
its	27,783	whose	4,005	oneself	239	once	0
our	23,029	someone	3,755	thou	229	one another	0
these	22,697	certain	3,345	’un	227	overmuch	0
any	22,666	anyone	3,318	ye	192	such and such	0
more	21,873	whom	3,229	thy	191	whate’er	0
many	17,343	enough	3,197	whereby	176	whenever	0
such	16,880	half	3,065	thee	166	whereof	0
those	15,819	few	2,933	yourselves	148	whereto	0
own	15,741	everyone	2,812	latter	142	whereunto	0
us	15,724	whatever	2,571	whichever	121	whichsoever	0

Figure 5.4 Pronouns of English from the CELEX on-line dictionary. Frequency counts are from the COBUILD 16 million word corpus.

may indicates permission or possibility, *must* indicates necessity, and so on. Fig. 5.1 gives counts for the frequencies of the modals in English. In addition to the perfect *have* mentioned above, there is a modal verb *have* (e.g., *I <u>have</u> to go*), which is very common in spoken English. Neither it nor the modal verb *dare*, which is very rare, have frequency counts because the CELEX dictionary does not distinguish the main verb sense (*I <u>have</u> three oranges*, *He <u>dared</u> me to eat them*), from the modal sense (*There <u>has</u> to be some mistake*, *<u>Dare</u> I confront him?*), from the non-modal auxiliary verb sense (*I <u>have</u> never seen that*).

English also has many words of more or less unique function, including **interjections** (*oh, ah, hey, man, alas, uh, um*), **negatives** (*no, not*), **politeness markers** (*please, thank you*), **greetings** (*hello, goodbye*), and the existential **there** (*<u>there</u> are two on the table*) among others. Whether these classes are assigned particular names or lumped together (as interjections or even adverbs) depends on the purpose of the labeling.

interjections
negatives
politeness markers

can	70,930	might	5,580	shouldn't	858
will	69,206	couldn't	4,265	mustn't	332
may	25,802	shall	4,118	'll	175
would	18,448	wouldn't	3,548	needn't	148
should	17,760	won't	3,100	mightn't	68
must	16,520	'd	2,299	oughtn't	44
need	9,955	ought	1,845	mayn't	3
can't	6,375	will	862	dare, have	???

Figure 5.5 English modal verbs from the CELEX on-line dictionary. Frequency counts are from the COBUILD 16 million word corpus.

5.2 Tagsets for English

The previous section gave broad descriptions of the kinds of syntactic classes that English words fall into. This section fleshes out that sketch by describing the actual tagsets used in part-of-speech tagging, in preparation for the various tagging algorithms to be described in the following sections.

There are a small number of popular tagsets for English, many of which evolved from the 87-tag tagset used for the Brown corpus (Francis, 1979; Francis and Kučera, 1982). The Brown corpus is a 1 million word collection of samples from 500 written texts from different genres (newspaper, novels, non-fiction, academic, etc.) which was assembled at Brown University in 1963–1964 (Kučera and Francis, 1967; Francis, 1979; Francis and Kučera, 1982). This corpus was tagged with parts-of-speech by first applying the TAGGIT program and then hand-correcting the tags.

Besides this original Brown tagset, two of the most commonly used tagsets are the small 45-tag Penn Treebank tagset (Marcus et al., 1993), and the medium-sized 61 tag C5 tagset used by the Lancaster UCREL project's CLAWS (the Constituent Likelihood Automatic Word-tagging System) tagger to tag the British National Corpus (BNC) (Garside et al., 1997). We give all three of these tagsets here, focusing on the smallest, the Penn Treebank set, and discuss difficult tagging decisions in that tag set and some useful distinctions made in the larger tagsets.

The Penn Treebank tagset, shown in Fig. 5.2, has been applied to the Brown corpus, the Wall Street Journal corpus, and the Switchboard corpus among others; indeed, perhaps partly because of its small size, it is one of the most widely used tagsets. Here are some examples of tagged sentences from the Penn Treebank version of the Brown corpus (we will represent a tagged word by placing the tag after each word, delimited by a slash):

(5.1) The/DT grand/JJ jury/NN commented/VBD on/IN a/DT number/NN of/IN other/JJ topics/NNS ./.

(5.2) **There/EX** are/VBP 70/CD children/NNS **there/RB**

(5.3) Although/IN preliminary/JJ findings/NNS were/VBD **reported/VBN** more/RBR than/IN a/DT year/NN ago/IN ,/, the/DT latest/JJS results/NNS appear/VBP in/IN today/NN **'s/POS** New/NNP England/NNP Journal/NNP of/IN Medicine/NNP ,/,

Example (5.1) shows phenomena that we discussed in the previous section; the de-

Tag	Description	Example	Tag	Description	Example
CC	Coordin. Conjunction	*and, but, or*	SYM	Symbol	*+,%, &*
CD	Cardinal number	*one, two, three*	TO	“to”	*to*
DT	Determiner	*a, the*	UH	Interjection	*ah, oops*
EX	Existential ‘there’	*there*	VB	Verb, base form	*eat*
FW	Foreign word	*mea culpa*	VBD	Verb, past tense	*ate*
IN	Preposition/sub-conj	*of, in, by*	VBG	Verb, gerund	*eating*
JJ	Adjective	*yellow*	VBN	Verb, past participle	*eaten*
JJR	Adj., comparative	*bigger*	VBP	Verb, non-3sg pres	*eat*
JJS	Adj., superlative	*wildest*	VBZ	Verb, 3sg pres	*eats*
LS	List item marker	*1, 2, One*	WDT	Wh-determiner	*which, that*
MD	Modal	*can, should*	WP	Wh-pronoun	*what, who*
NN	Noun, sing. or mass	*llama*	WP$	Possessive wh-	*whose*
NNS	Noun, plural	*llamas*	WRB	Wh-adverb	*how, where*
NNP	Proper noun, singular	*IBM*	$	Dollar sign	*$*
NNPS	Proper noun, plural	*Carolinas*	#	Pound sign	*#*
PDT	Predeterminer	*all, both*	“	Left quote	‘ or “
POS	Possessive ending	*’s*	”	Right quote	’ or ”
PRP	Personal pronoun	*I, you, he*	(	Left parenthesis	[, (, {, <
PRP$	Possessive pronoun	*your, one’s*	)	Right parenthesis	],), }, >
RB	Adverb	*quickly, never*	,	Comma	,
RBR	Adverb, comparative	*faster*	.	Sentence-final punc	. ! ?
RBS	Adverb, superlative	*fastest*	:	Mid-sentence punc	: ; ... – -
RP	Particle	*up, off*			

Figure 5.6 Penn Treebank part-of-speech tags (including punctuation).

terminers *the* and *a*, the adjectives *grand* and *other*, the common nouns *jury*, *number*, and *topics*, the past tense verb *commented*. Example (5.2) shows the use of the EX tag to mark the existential *there* construction in English, and, for comparison, another use of *there* which is tagged as an adverb (RB). Example (5.3) shows the segmentation of the possessive morpheme *’s*, and shows an example of a passive construction, ‘were reported’, in which the verb *reported* is marked as a past participle (VBN), rather than a simple past (VBD). Note also that the proper noun *New England* is tagged NNP. Finally, note that since *New England Journal of Medicine* is a proper noun, the Treebank tagging chooses to mark each noun in it separately as NNP, including *journal* and *medicine*, which might otherwise be labeled as common nouns (NN).

Some tagging distinctions are quite hard for both humans and machines to make. For example prepositions (IN), particles (RP), and adverbs (RB) can have a large overlap. Words like *around* can be all three:

(5.4) Mrs./NNP Shaefer/NNP never/RB got/VBD **around/RP** to/TO joining/VBG

(5.5) All/DT we/PRP gotta/VBN do/VB is/VBZ go/VB **around/IN** the/DT corner/NN

(5.6) Chateau/NNP Petrus/NNP costs/VBZ **around/RB** 250/CD

Making these decisions requires sophisticated knowledge of syntax; tagging manuals (Santorini, 1990) give various heuristics that can help human coders make these decisions, and that can also provide useful features for automatic taggers. For example two heuristics from Santorini (1990) are that prepositions generally are associated with a following noun phrase (although they also may be followed by prepositional phrases), and that the word *around* is tagged as an adverb when it means “approximately”. Furthermore, particles often can either precede or follow a noun phrase object, as in the following examples:

(5.7) She told off/RP her friends

(5.8) She told her friends off/RP.

Prepositions, on the other hand, cannot follow their noun phrase (* is used here to mark an ungrammatical sentence, a concept which we will return to in Ch. 12):

(5.9) She stepped off/IN the train

(5.10) *She stepped the train off/IN.

Another difficulty is labeling the words that can modify nouns. Sometimes the modifiers preceding nouns are common nouns like *cotton* below, other times the Treebank tagging manual specifies that modifiers be tagged as adjectives (for example if the modifier is a hyphenated common noun like *income-tax*) and other times as proper nouns (for modifiers which are hyphenated proper nouns like *Gramm-Rudman*):

(5.11) cotton/NN sweater/NN

(5.12) income-tax/JJ return/NN

(5.13) the/DT Gramm-Rudman/NP Act/NP

Some words that can be adjectives, common nouns, or proper nouns, are tagged in the Treebank as common nouns when acting as modifiers:

(5.14) Chinese/NN cooking/NN

(5.15) Pacific/NN waters/NNS

A third known difficulty in tagging is distinguishing past participles (VBN) from adjectives (JJ). A word like *married* is a past participle when it is being used in an eventive, verbal way, as in (5.16) below, and is an adjective when it is being used to express a property, as in (5.17):

(5.16) They were married/VBN by the Justice of the Peace yesterday at 5:00.

(5.17) At the time, she was already married/JJ.

Tagging manuals like Santorini (1990) give various helpful criteria for deciding how ‘verb-like’ or ‘eventive’ a particular word is in a specific context.

The Penn Treebank tagset was culled from the original 87-tag tagset for the Brown corpus. This reduced set leaves out information that can be recovered from the identity of the lexical item. For example the original Brown and C5 tagsets include a separate tag for each of the different forms of the verbs *do* (e.g. C5 tag “VDD” for *did* and “VDG” for *doing*), *be*, and *have*. These were omitted from the Treebank set.

Certain syntactic distinctions were not marked in the Penn Treebank tagset because Treebank sentences were parsed, not merely tagged, and so some syntactic information is represented in the phrase structure. For example, the single tag IN is used for both prepositions and subordinating conjunctions since the tree-structure of the sentence disambiguates them (subordinating conjunctions always precede clauses, prepositions precede noun phrases or prepositional phrases). Most tagging situations, however, do not involve parsed corpora; for this reason the Penn Treebank set is not specific enough for many uses. The original Brown and C5 tagsets, for example, distinguish prepositions (IN) from subordinating conjunctions (CS), as in the following examples:

(5.18) **after/CS** spending/VBG a/AT day/NN at/IN the/AT Brown/NP Palace/NN

(5.19) **after/IN** a/AT wedding/NN trip/NN to/IN Corpus/NP Christi/NP ./.

The original Brown and C5 tagsets also have two tags for the word *to*; in Brown the infinitive use is tagged TO, while the prepositional use as IN:

(5.20) **to/TO** give/VB priority/NN **to/IN** teacher/NN pay/NN raises/NNS

Brown also has the tag NR for adverbial nouns like *home*, *west*, *Monday*, and *tomorrow*. Because the Treebank lacks this tag, it has a much less consistent policy for adverbial nouns; *Monday*, *Tuesday*, and other days of the week are marked NNP, *tomorrow*, *west*, and *home* are marked sometimes as NN, sometimes as RB. This makes the Treebank tagset less useful for high-level NLP tasks like the detection of time phrases.

Nonetheless, the Treebank tagset has been the most widely used in evaluating tagging algorithms, and so many of the algorithms we describe below have been evaluated mainly on this tagset. Of course whether a tagset is useful for a particular application depends on how much information the application needs.

5.3 Part-of-Speech Tagging

tagging

Part-of-speech tagging (or just **tagging** for short) is the process of assigning a part-of-speech or other syntactic class marker to each word in a corpus. Because tags are generally also applied to punctuation, tagging requires that the punctuation marks (period, comma, etc) be separated off of the words. Thus **tokenization** of the sort described in Ch. 3 is usually performed before, or as part of, the tagging process, separating commas, quotation marks, etc., from words, and disambiguating end-of-sentence punctuation (period, question mark, etc) from part-of-word punctuation (such as in abbreviations like *e.g.* and *etc.*)

The input to a tagging algorithm is a string of words and a specified tagset of the kind described in the previous section. The output is a single best tag for each word. For example, here are some sample sentences from the ATIS corpus of dialogues about air-travel reservations that we will discuss in Ch. 12. For each we have shown a potential tagged output using the Penn Treebank tagset defined in Fig. 5.2 on page 131:

(5.21) Book/VB that/DT flight/NN ./.

(5.22) Does/VBZ that/DT flight/NN serve/VB dinner/NN ?/.

The previous section discussed some tagging decisions that are difficult to make for humans. Even in these simple examples, automatically assigning a tag to each

ambiguous

word is not trivial. For example, *book* is **ambiguous**. That is, it has more than one possible usage and part-of-speech. It can be a verb (as in *book that flight* or *to book the suspect*) or a noun (as in *hand me that book*, or *a book of matches*). Similarly *that* can be a determiner (as in *Does that flight serve dinner*), or a complementizer (as in *I thought that your flight was earlier*). The problem of POS-tagging is to **resolve** these

resolve

Tag	Description	Example
(	opening parenthesis	(, [
)	closing parenthesis	),]
*	negator	not n’t
,	comma	,
–	dash	–
.	sentence terminator	. ; ? !
:	colon	:
ABL	pre-qualifier	quite, rather, such
ABN	pre-quantifier	half, all,
ABX	pre-quantifier, double conjunction	both
AP	post-determiner	many, next, several, last
AT	article	a the an no a every
BE/BED/BEDZ/BEG/BEM/BEN/BER/BEZ		be/were/was/being/am/been/are/is
CC	coordinating conjunction	and or but either neither
CD	cardinal numeral	two, 2, 1962, million
CS	subordinating conjunction	that as after whether before
DO/DOD/DOZ		do, did, does
DT	singular determiner,	this, that
DTI	singular or plural determiner	some, any
DTS	plural determiner	these those them
DTX	determiner, double conjunction	either, neither
EX	existential there	there
HV/HVD/HVG/HVN/HVZ		have, had, having, had, has
IN	preposition	of in for by to on at
JJ	adjective	
JJR	comparative adjective	better, greater, higher, larger, lower
JJS	semantically superlative adj.	main, top, principal, chief, key, foremost
JJT	morphologically superlative adj.	best, greatest, highest, largest, latest, worst
MD	modal auxiliary	would, will, can, could, may, must, should
NN	(common) singular or mass noun	time, world, work, school, family, door
NN$	possessive singular common noun	father’s, year’s, city’s, earth’s
NNS	plural common noun	years, people, things, children, problems
NNS$	possessive plural noun	children’s, artist’s parent’s years’
NP	singular proper noun	Kennedy, England, Rachel, Congress
NP$	possessive singular proper noun	Plato’s Faulkner’s Viola’s
NPS	plural proper noun	Americans, Democrats, Chinese,
NPS$	possessive plural proper noun	Yankees’, Gershwins’ Earthmen’s
NR	adverbial noun	home, west, tomorrow, Friday, North,
NR$	possessive adverbial noun	today’s, yesterday’s, Sunday’s, South’s
NRS	plural adverbial noun	Sundays Fridays
OD	ordinal numeral	second, 2nd, twenty-first, mid-twentieth
PN	nominal pronoun	one, something, nothing, anyone, none,
PN$	possessive nominal pronoun	one’s someone’s anyone’s
PP$	possessive personal pronoun	his their her its my our your
PP$$	second possessive personal pronoun	mine, his, ours, yours, theirs
PPL	singular reflexive personal pronoun	myself, herself
PPLS	plural reflexive pronoun	ourselves, themselves
PPO	objective personal pronoun	me, us, him
PPS	3rd. sg. nominative pronoun	he, she, it
PPSS	other nominative pronoun	I, we, they
QL	qualifier	very, too, most, quite, almost, extremely
QLP	post-qualifier	enough, indeed
RB	adverb	
RBR	comparative adverb	later, more, better, longer, further
RBT	superlative adverb	best, most, highest, nearest
RN	nominal adverb	here, then

Figure 5.7 First part of original 87-tag Brown corpus tagset (Francis and Kučera, 1982).

Tag	Description	Example
RP	adverb or particle	across, off, up
TO	infinitive marker	to
UH	interjection, exclamation	well, oh, say, please, okay, uh, goodbye
VB	verb, base form	make, understand, try, determine, drop
VBD	verb, past tense	said, went, looked, brought, reached kept
VBG	verb, present participle, gerund	getting, writing, increasing
VBN	verb, past participle	made, given, found, called, required
VBZ	verb, 3rd singular present	says, follows, requires, transcends
WDT	wh- determiner	what, which
WP$	possessive wh- pronoun	whose
WPO	objective wh- pronoun	whom, which, that
WPS	nominative wh- pronoun	who, which, that
WQL	how	
WRB	wh- adverb	how, when

Figure 5.8 Rest of 87-tag Brown corpus tagset (Francis and Kučera, 1982).

ambiguities, choosing the proper tag for the context. Part-of-speech tagging is thus one of the many **disambiguation** tasks we will see in this book.

disambiguation

How hard is the tagging problem? The previous section described some difficult tagging decisions; how common is tag ambiguity? It turns out that most words in English are unambiguous; i.e., they have only a single tag. But many of the most common words of English are ambiguous (for example *can* can be an auxiliary (‘to be able’), a noun (‘a metal container’), or a verb (‘to put something in such a metal container’)). In fact, DeRose (1988) reports that while only 11.5% of English word types in the Brown corpus are ambiguous, over 40% of Brown tokens are ambiguous. Fig. 5.10 shows the number of word types with different levels of part-of-speech ambiguity from the Brown corpus. We show these computations from two versions of the tagged Brown corpus, the original tagging done at Brown by Francis and Kučera (1982), and the Treebank-3 tagging done at the University of Pennsylvania. Note that despite having more coarse-grained tags, the 45-tag corpus unexpectedly has more ambiguity than the 87-tag corpus.

Luckily, it turns out that many of the 40% ambiguous tokens are easy to disambiguate. This is because the various tags associated with a word are not equally likely. For example, *a* can be a determiner, or the letter *a* (perhaps as part of an acronym or an initial). But the determiner sense of *a* is much more likely.

rule-based
stochastic taggers

Most tagging algorithms fall into one of two classes: **rule-based** taggers and **stochastic taggers** taggers. Rule-based taggers generally involve a large database of hand-written disambiguation rules which specify, for example, that an ambiguous word is a noun rather than a verb if it follows a determiner. The next section will describe a sample rule-based tagger, **EngCG**, based on the Constraint Grammar architecture of Karlsson et al. (1995b).

Stochastic taggers generally resolve tagging ambiguities by using a training corpus to compute the probability of a given word having a given tag in a given context. Sec. 5.5 describes the Hidden Markov Model or **HMM tagger**.

HMM tagger

Finally, Sec. 5.6 will describe an approach to tagging called the **transformation-based tagger** or the **Brill tagger**, after Brill (1995). The Brill tagger shares features of both tagging architectures. Like the rule-based tagger, it is based on rules which determine when an ambiguous word should have a given tag. Like the stochastic tag-

Brill tagger

Tag	Description	Example
AJ0	adjective (unmarked)	*good, old*
AJC	comparative adjective	*better, older*
AJS	superlative adjective	*best, oldest*
AT0	article	*the, a, an*
AV0	adverb (unmarked)	*often, well, longer, furthest*
AVP	adverb particle	*up, off, out*
AVQ	wh-adverb	*when, how, why*
CJC	coordinating conjunction	*and, or*
CJS	subordinating conjunction	*although, when*
CJT	the conjunction *that*	
CRD	cardinal numeral (except *one*)	*3, twenty-five, 734*
DPS	possessive determiner	*your, their*
DT0	general determiner	*these, some*
DTQ	wh-determiner	*whose, which*
EX0	existential *there*	
ITJ	interjection or other isolate	*oh, yes, mhm*
NN0	noun (neutral for number)	*aircraft, data*
NN1	singular noun	*pencil, goose*
NN2	plural noun	*pencils, geese*
NP0	proper noun	*London, Michael, Mars*
ORD	ordinal	*sixth, 77th, last*
PNI	indefinite pronoun	*none, everything*
PNP	personal pronoun	*you, them, ours*
PNQ	wh-pronoun	*who, whoever*
PNX	reflexive pronoun	*itself, ourselves*
POS	possessive *’s* or *’*	
PRF	the preposition *of*	
PRP	preposition (except *of*)	*for, above, to*
PUL	punctuation – left bracket	(or [
PUN	punctuation – general mark	. ! , : ; - ? ...
PUQ	punctuation – quotation mark	‘ ’ ”
PUR	punctuation – right bracket	) or]
TO0	infinitive marker *to*	
UNC	unclassified items (not English)	
VBB	base forms of *be* (except infinitive)	*am, are*
VBD	past form of *be*	*was, were*
VBG	-ing form of *be*	*being*
VBI	infinitive of *be*	
VBN	past participle of *be*	*been*
VBZ	-s form of *be*	*is, ’s*
VDB/D/G/I/N/Z	form of *do*	*do, does, did, doing, to do, etc.*
VHB/D/G/I/N/Z	form of *have*	*have, had, having, to have, etc.*
VM0	modal auxiliary verb	*can, could, will, ’ll*
VVB	base form of lexical verb (except infin.)	*take, live*
VVD	past tense form of lexical verb	*took, lived*
VVG	-ing form of lexical verb	*taking, living*
VVI	infinitive of lexical verb	*take, live*
VVN	past participle form of lex. verb	*taken, lived*
VVZ	-s form of lexical verb	*takes, lives*
XX0	the negative *not* or *n’t*	
ZZ0	alphabetical symbol	*A, B, c, d*

Figure 5.9 UCREL’s C5 tagset for the British National Corpus (Garside et al., 1997).

gers, it has a machine-learning component: the rules are automatically induced from a previously tagged training corpus.

		87-tag Original Brown		45-tag Treebank Brown	
Unambiguous (1 tag)		**44,019**		**38,857**	
Ambiguous (2–7 tags)		**5,490**		**8844**	
Details:	2 tags	4,967		6,731	
	3 tags	411		1621	
	4 tags	91		357	
	5 tags	17		90	
	6 tags	2	(*well*, *beat*)	32	
	7 tags	2	(*still*, *down*)	6	(*well*, *set*, *round*, *open*, *fit*, *down*)
	8 tags			4	(*'s*, *half*, *back*, *a*)
	9 tags			3	(*that*, *more*, *in*)

Figure 5.10 The amount of tag ambiguity for word types in the Brown corpus, from the ICAME release of the original (87-tag) tagging and the Treebank-3 (45-tag) tagging. Numbers are not strictly comparable because only the Treebank segments *'s*. An earlier estimate of some of these numbers is reported in DeRose (1988).

5.4 Rule-Based Part-of-Speech Tagging

The earliest algorithms for automatically assigning part-of-speech were based on a two-stage architecture (Harris, 1962; Klein and Simmons, 1963; Greene and Rubin, 1971). The first stage used a dictionary to assign each word a list of potential parts-of-speech. The second stage used large lists of hand-written disambiguation rules to winnow down this list to a single part-of-speech for each word.

Modern rule-based approaches to part-of-speech tagging have a similar architecture, although the dictionaries and the rule sets are vastly larger than in the 1960's. One of the most comprehensive rule-based approaches is the Constraint Grammar approach (Karlsson et al., 1995a). In this section we describe a tagger based on this approach, the **EngCG** tagger (Voutilainen, 1995, 1999).

EngCG

The EngCG ENGTWOL lexicon is based on the two-level morphology described in Ch. 3, and has about 56,000 entries for English word stems (Heikkilä, 1995), counting a word with multiple parts-of-speech (e.g., nominal and verbal senses of *hit*) as separate entries, and not counting inflected and many derived forms. Each entry is annotated with a set of morphological and syntactic features. Fig. 5.4 shows some selected words, together with a slightly simplified listing of their features, as used in rule writing.

Most of the features in Fig. 5.4 are relatively self-explanatory; SG for singular, -SG3 for other than third-person-singular. NOMINATIVE means non-genitive and PCP2 means past participle. PRE, CENTRAL, and POST are ordering slots for determiners (predeterminers (*all*) come before determiners (*the*): *all the president's men*). NOINDEFDETERMINER means that words like *furniture* do not appear with the indefinite determiner *a*. SV, SVO, and SVOO specify the **subcategorization** or **complementation** pattern for the verb. Subcategorization will be discussed in Ch. 12 and Ch. 16, but briefly SV means the verb appears solely with a subject (*nothing occurred*); SVO with a subject and an object (*I showed the film*); SVOO with a subject and two complements: *She showed her the ball*.

subcategorization
complementation

Word	POS	Additional POS features
smaller	ADJ	COMPARATIVE
fast	ADV	SUPERLATIVE
that	DET	CENTRAL DEMONSTRATIVE SG
all	DET	PREDETERMINER SG/PL QUANTIFIER
dog’s	N	GENITIVE SG
furniture	N	NOMINATIVE SG NOINDEFDETERMINER
one-third	NUM	SG
she	PRON	PERSONAL FEMININE NOMINATIVE SG3
show	V	PRESENT -SG3 VFIN
show	N	NOMINATIVE SG
shown	PCP2	SVOO SVO SV
occurred	PCP2	SV
occurred	V	PAST VFIN SV

Figure 5.11 Lexical entries in the ENGTWOL lexicon (Voutilainen, 1995; Heikkilä, 1995).

In the first stage of the tagger, each word is run through the two-level lexicon transducer and the entries for all possible parts-of-speech are returned. For example the phrase *Pavlov had shown that salivation* . . . would return the following list (one line per possible tag, with the correct tag shown in boldface):

Pavlov	**PAVLOV N NOM SG PROPER**
had	**HAVE V PAST VFIN SVO**
	HAVE PCP2 SVO
shown	**SHOW PCP2 SVOO SVO SV**
that	ADV
	PRON DEM SG
	DET CENTRAL DEM SG
	CS
salivation	**N NOM SG**
. . .	

EngCG then applies a large set of constraints (as many as 3,744 constraints in the EngCG-2 system) to the input sentence to rule out incorrect parts-of-speech. The boldfaced entries in the table above show the desired result, in which the simple past tense tag (rather than the past participle tag) is applied to *had*, and the complementizer (CS) tag is applied to *that*. The constraints are used in a negative way, to eliminate tags that are inconsistent with the context. For example one constraint eliminates all readings of *that* except the ADV (adverbial intensifier) sense (this is the sense in the sentence *it isn’t that odd*). Here’s a simplified version of the constraint:

```
ADVERBIAL-THAT RULE
Given input: “that”
if
  (+1 A/ADV/QUANT); /* if next word is adj, adverb, or quantifier */
  (+2 SENT-LIM);       /* and following which is a sentence boundary, */
  (NOT -1 SVOC/A); /* and the previous word is not a verb like */
                   /* ‘consider’ which allows adjs as object complements */
then eliminate non-ADV tags
else eliminate ADV tag
```

The first two clauses of this rule check to see that the *that* directly precedes a sentence-final adjective, adverb, or quantifier. In all other cases the adverb reading is eliminated. The last clause eliminates cases preceded by verbs like *consider* or *believe* which can take a noun and an adjective; this is to avoid tagging the following instance of *that* as an adverb:

I consider that odd.

Another rule is used to express the constraint that the complementizer sense of *that* is most likely to be used if the previous word is a verb which expects a complement (like *believe*, *think*, or *show*), and if *that* is followed by the beginning of a noun phrase, and a finite verb.

This description oversimplifies the EngCG architecture; the system also includes probabilistic constraints, and also makes use of other syntactic information we haven't discussed. The interested reader should consult Karlsson et al. (1995b) and Voutilainen (1999).

5.5 HMM Part-of-Speech Tagging

The use of probabilities in tags is quite old; probabilities in tagging were first used by Stolz et al. (1965), a complete probabilistic tagger with Viterbi decoding was sketched by Bahl and Mercer (1976), and various stochastic taggers were built in the 1980s (Marshall, 1983; Garside, 1987; Church, 1988; DeRose, 1988). This section describes a particular stochastic tagging algorithm generally known as the Hidden Markov Model or HMM tagger. Hidden Markov Models themselves will be more fully introduced and defined in Ch. 6. In this section, we prefigure Ch. 6 a bit by giving an initial introduction to the Hidden Markov Model as applied to part-of-speech tagging.

Bayesian inference

Use of a Hidden Markov Model to do part-of-speech-tagging, as we will define it, is a special case of **Bayesian inference**, a paradigm that has been known since the work of Bayes (1763). Bayesian inference or Bayesian classification was applied successfully to language problems as early as the late 1950s, including the OCR work of Bledsoe in 1959, and the seminal work of Mosteller and Wallace (1964) on applying Bayesian inference to determine the authorship of the Federalist papers.

In a classification task, we are given some observation(s) and our job is to determine which of a set of classes it belongs to. Part-of-speech tagging is generally treated as a sequence classification task. So here the observation is a sequence of words (let's say a sentence), and it is our job to assign them a sequence of part-of-speech tags.

For example, say we are given a sentence like

(5.23) Secretariat is expected to **race** tomorrow.

What is the best sequence of tags which corresponds to this sequence of words? The Bayesian interpretation of this task starts by considering all possible sequences of classes—in this case, all possible sequences of tags. Out of this universe of tag sequences, we want to choose the tag sequence which is most probable given the observation sequence of n words w_1^n. In other words, we want, out of all sequences of n

ˆ

tags t_1^n the single tag sequence such that $P(t_1^n|w_1^n)$ is highest. We use the hat notation ˆ to mean "our estimate of the correct tag sequence".

(5.24)
$$\hat{t}_1^n = \operatorname*{argmax}_{t_1^n} P(t_1^n|w_1^n)$$

The function $\text{argmax}_x f(x)$ means "the x such that $f(x)$ is maximized". Eq. 5.24 thus means, out of all tag sequences of length n, we want the particular tag sequence t_1^n which maximizes the right-hand side. While (5.24) is guaranteed to give us the optimal tag sequence, it is not clear how to make the equation operational; that is, for a given tag sequence t_1^n and word sequence w_1^n, we don't know how to directly compute $P(t_1^n|w_1^n)$.

The intuition of Bayesian classification is to use Bayes' rule to transform (5.24) into a set of other probabilities which turn out to be easier to compute. Bayes' rule is presented in (5.25); it gives us a way to break down any conditional probability $P(x|y)$ into three other probabilities:

(5.25)
$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

We can then substitute (5.25) into (5.24) to get (5.26):

(5.26)
$$\hat{t}_1^n = \operatorname*{argmax}_{t_1^n} \frac{P(w_1^n|t_1^n)P(t_1^n)}{P(w_1^n)}$$

We can conveniently simplify 5.26 by dropping the denominator $P(w_1^n)$. Why is that? Since we are choosing a tag sequence out of all tag sequences, we will be computing $\frac{P(w_1^n|t_1^n)P(t_1^n)}{P(w_1^n)}$ for each tag sequence. But $P(w_1^n)$ doesn't change for each tag sequence; we are always asking about the most likely tag sequence for the same observation w_1^n, which must have the same probability $P(w_1^n)$. Thus we can choose the tag sequence which maximizes this simpler formula:

(5.27)
$$\hat{t}_1^n = \operatorname*{argmax}_{t_1^n} P(w_1^n|t_1^n)P(t_1^n)$$

To summarize, the most probable tag sequence $\hat{t}_1^n$ given some word string w_1^n can be computed by taking the product of two probabilities for each tag sequence, and choosing the tag sequence for which this product is greatest. The two terms are the

prior probability
likelihood

prior probability of the tag sequence $P(t_1^n)$), and the **likelihood** of the word string $P(w_1^n|t_1^n)$:

(5.28)
$$\hat{t}_1^n = \operatorname*{argmax}_{t_1^n} \overbrace{P(w_1^n|t_1^n)}^{\text{likelihood}} \; \overbrace{P(t_1^n)}^{\text{prior}}$$

Unfortunately, (5.28) is still too hard to compute directly. HMM taggers therefore make two simplifying assumptions. The first assumption is that the probability of a word appearing is dependent only on its own part-of-speech tag; that it is independent of other words around it, and of the other tags around it:

(5.29) $$P(w_1^n|t_1^n) \approx \prod_{i=1}^{n} P(w_i|t_i)$$

The second assumption is that the probability of a tag appearing is dependent only on the previous tag, the **bigram** assumption we saw in Ch. 4:

(5.30) $$P(t_1^n) \approx \prod_{i=1}^{n} P(t_i|t_{i-1})$$

Plugging the simplifying assumptions (5.29) and (5.30) into (5.28) results in the following equation by which a bigram tagger estimates the most probable tag sequence:

(5.31) $$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n|w_1^n) \approx \underset{t_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})$$

Eq. 5.31 contains two kinds of probabilities, tag transition probabilities and word likelihoods. Let's take a moment to see what these probabilities represent. The tag transition probabilities, $P(t_i|t_{i-1})$, represent the probability of a tag given the previous tag. For example, determiners are very likely to precede adjectives and nouns, as in sequences like *that/DT flight/NN* and *the/DT yellow/JJ hat/NN*. Thus we would expect the probabilities $P(\mathrm{NN}|\mathrm{DT})$ and $P(\mathrm{JJ}|\mathrm{DT})$ to be high. But in English, adjectives don't tend to precede determiners, so the probability $P(\mathrm{DT}|\mathrm{JJ})$ ought to be low.

We can compute the maximum likelihood estimate of a tag transition probability $P(\mathrm{NN}|\mathrm{DT})$ by taking a corpus in which parts-of-speech are labeled and counting, out of the times we see DT, how many of those times we see NN after the DT. That is, we compute the following ratio of counts:

(5.32) $$P(t_i|t_{i-1}) = \frac{C(t_{i-1},t_i)}{C(t_{i-1})}$$

Let's choose a specific corpus to examine. For the examples in this chapter we'll use the Brown corpus, the 1 million word corpus of American English described earlier. The Brown corpus has been tagged twice, once in the 1960's with the 87-tag tagset, and again in the 1990's with the 45-tag Treebank tagset. This makes it useful for comparing tagsets, and is also widely available.

In the 45-tag Treebank Brown corpus, the tag DT occurs 116,454 times. Of these, DT is followed by NN 56,509 times (if we ignore the few cases of ambiguous tags). Thus the MLE estimate of the transition probability is calculated as follows:

(5.33) $$P(NN|DT) = \frac{C(DT,NN)}{C(DT)} = \frac{56,509}{116,454} = .49$$

The probability of getting a common noun after a determiner, .49, is indeed quite high, as we suspected.

The word likelihood probabilities, $P(w_i|t_i)$, represent the probability, given that we see a given tag, that it will be associated with a given word. For example if we were to see the tag VBZ (third person singular present verb) and guess the verb that is likely to have that tag, we might likely guess the verb *is*, since the verb *to be* is so common in English.

We can compute the MLE estimate of a word likelihood probability like $P(\text{is}|\text{VBZ})$ again by counting, out of the times we see VBZ in a corpus, how many of those times the VBZ is labeling the word *is*. That is, we compute the following ratio of counts:

$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)} \tag{5.34}$$

In Treebank Brown corpus, the tag VBZ occurs 21,627 times, and VBZ is the tag for *is* 10,073 times. Thus:

$$P(is|VBZ) = \frac{C(VBZ, is)}{C(VBZ)} = \frac{10,073}{21,627} = .47 \tag{5.35}$$

For those readers who are new to Bayesian modeling note that this likelihood term is not asking "which is the most likely tag for the word *is*". That is, the term is not $P(\text{VBZ}|\text{is})$. Instead we are computing $P(\text{is}|\text{VBZ})$. The probability, slightly counterintuitively, answers the question "If we were expecting a third person singular verb, how likely is it that this verb would be *is*?".

We have now defined HMM tagging as a task of choosing a tag-sequence with the maximum probability, derived the equations by which we will compute this probability, and shown how to compute the component probabilities. In fact we have simplified the presentation of the probabilities in many ways; in later sections we will return to these equations and introduce the deleted interpolation algorithm for smoothing these counts, the trigram model of tag history, and a model for unknown words.

But before turning to these augmentations, we need to introduce the decoding algorithm by which these probabilities are combined to choose the most likely tag sequence.

5.5.1 Computing the most-likely tag sequence: An example

The previous section showed that the HMM tagging algorithm chooses as the most likely tag sequence the one that maximizes the product of two terms; the probability of the sequence of tags, and the probability of each tag generating a word. In this section we ground these equations in a specific example, showing for a particular sentence how the correct tag sequence achieves a higher probability than one of the many possible wrong sequences.

We will focus on resolving the part-of-speech ambiguity of the word *race*, which can be a noun or verb in English, as we show in two examples modified from the Brown and Switchboard corpus. For this example, we will use the 87-tag Brown corpus tagset, because it has a specific tag for *to*, TO, used only when *to* is an infinitive; prepositional uses of *to* are tagged as IN. This will come in handy in our example.[1]

In (5.36) *race* is a verb (VB) while in (5.37) *race* is a common noun (NN):

(5.36) Secretariat/NNP is/BEZ expected/VBN to/TO **race**/VB tomorrow/NR

[1] The 45-tag Treebank-3 tagset does make this distinction in the Switchboard corpus but not, alas, in the Brown corpus. Recall that in the 45-tag tagset time adverbs like *tomorrow* are tagged as NN; in the 87-tag tagset they appear as NR.

(5.37) People/NNS continue/VB to/TO inquire/VB the/AT reason/NN for/IN the/AT **race**/NN for/IN outer/JJ space/NN

Let's look at how *race* can be correctly tagged as a VB instead of an NN in (5.36). HMM part-of-speech taggers resolve this ambiguity globally rather than locally, picking the best tag sequence for the whole sentence. There are many hypothetically possible tag sequences for (5.36), since there are other ambiguities in the sentence (for example *expected* can be an adjective (JJ), a past tense/preterite (VBD) or a past participle (VBN)). But let's just consider two of the potential sequences, shown in Fig. 5.12. Note that these sequences differ only in one place; whether the tag chosen for *race* is VB or NN.

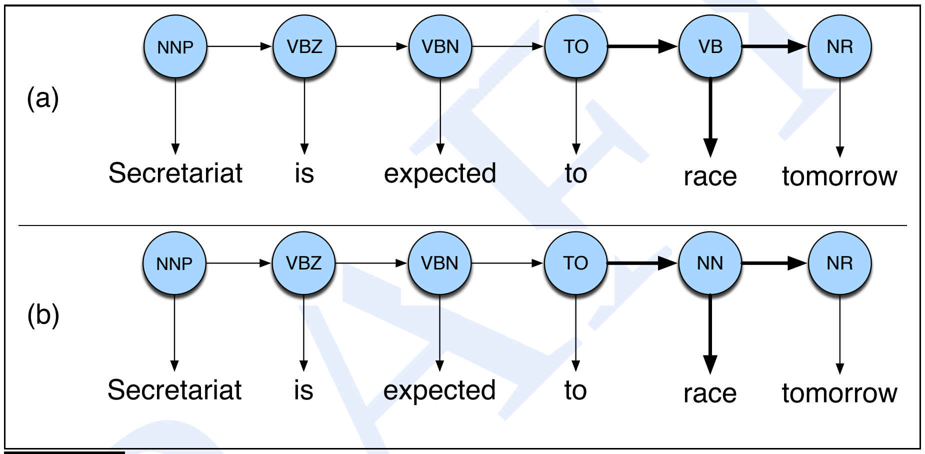


Figure 5.12 Two of the possible sequences of tags corresponding to the Secretariat sentence, one of them corresponding to the correct sequence, in which *race* is a VB. Each arc in these graphs would be associated with a probability. Note that the two graphs differ only in 3 arcs, hence in 3 probabilities.

Almost all the probabilities in these two sequences are identical; in Fig. 5.12 we have highlighted in boldface the three probabilities that differ. Let's consider two of these, corresponding to $P(t_i|t_{i-1})$ and $P(w_i|t_i)$. The probability $P(t_i|t_{i-1})$ in Fig. 5.12a is $P(\text{VB}|\text{TO})$, while in Fig. 5.12b the transition probability is $P(\text{NN}|\text{TO})$.

The tag transition probabilities $P(\text{NN}|\text{TO})$ and $P(\text{VB}|\text{TO})$ give us the answer to the question "How likely are we to expect a verb (noun) given the previous tag?" As we saw in the previous section, the maximum likelihood estimate for these probabilities can be derived from corpus counts.

Since the (87-tag Brown tagset) tag TO is used only for the infinitive marker *to*, we expect that only a very small number of nouns can follow this marker (as an exercise, try to think of a sentence where a noun can follow the infinitive marker use of *to*). Sure enough, a look at the (87-tag) Brown corpus gives us the following probabilities, showing that verbs are about 500 times as likely as nouns to occur after TO:

$$\begin{aligned} P(\text{NN}|\text{TO}) &= .00047 \\ P(\text{VB}|\text{TO}) &= .83 \end{aligned}$$

Let's now turn to $P(w_i|t_i)$, the lexical likelihood of the word *race* given a part-of-speech tag. For the two possible tags VB and NN, these correspond to the probabilities $P(\text{race}|\text{VB})$ and $P(\text{race}|\text{NN})$. Here are the lexical likelihoods from Brown:

$$\begin{aligned} P(\text{race}|\text{NN}) &= .00057 \\ P(\text{race}|\text{VB}) &= .00012 \end{aligned}$$

Finally, we need to represent the tag sequence probability for the following tag (in this case the tag NR for *tomorrow*):

$$\begin{aligned} P(\text{NR}|\text{VB}) &= .0027 \\ P(\text{NR}|\text{NN}) &= .0012 \end{aligned}$$

If we multiply the lexical likelihoods with the tag sequence probabilities, we see that the probability of the sequence with the VB tag is higher and the HMM tagger correctly tags *race* as a VB in Fig. 5.12 despite the fact that it is the less likely sense of *race*:

$$\begin{aligned} P(\text{VB}|\text{TO})P(\text{NR}|\text{VB})P(\text{race}|\text{VB}) &= .00000027 \\ P(\text{NN}|\text{TO})P(\text{NR}|\text{NN})P(\text{race}|\text{NN}) &= .00000000032 \end{aligned}$$

5.5.2 Formalizing Hidden Markov Model taggers

Now that we have seen the equations and some examples of choosing the most probable tag sequence, we show a brief formalization of this problem as a Hidden Markov Model (see Ch. 6 for the more complete formalization).

The HMM is an extension of the finite automata of Ch. 3. Recall that a finite automaton is defined by a set of states, and a set of transitions between states that are taken based on the input observations. *weighted* A **weighted finite-state automaton** is a simple augmentation of the finite automaton in which each arc is associated with a probability, indicating how likely that path is to be taken. The probability on all the arcs leaving a node must sum to 1. *Markov chain* A **Markov chain** is a special case of a weighted automaton in which the input sequence uniquely determines which states the automaton will go through. Because they can't represent inherently ambiguous problems, a Markov chain is only useful for assigning probabilities to unambiguous sequences.

While the Markov chain is appropriate for situations where we can see the actual conditioning events, it is not appropriate in part-of-speech tagging. This is because in part-of-speech tagging, while we observe the words in the input, we do *not* observe the part-of-speech tags. Thus we can't condition any probabilities on, say, a previous part-of-speech tag, because we cannot be completely certain exactly which tag applied to the previous word. *Hidden Markov Model* A **Hidden Markov Model** (**HMM**) allows us to talk about both *observed* events (like words that we see in the input) and *hidden* events (like part-of-speech tags) that we think of as causal factors in our probabilistic model.

HMM An **HMM** is specified by the following components:

$Q = q_1 q_2 \dots q_N$	a set of N **states**
$A = a_{11} a_{12} \dots a_{n1} \dots a_{nn}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \dots o_T$	a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, ..., v_V$.
$B = b_i(o_t)$	A sequence of **observation likelihoods:**, also called **emission probabilities**, each expressing the probability of an observation o_t being generated from a state i.
q_0, q_F	a special **start state** and **end (final) state** which are not associated with observations, together with transition probabilities $a_{01} a_{02} .. a_{0n}$ out of the start state and $a_{1F} a_{2F} \dots a_{nF}$ into the end state.

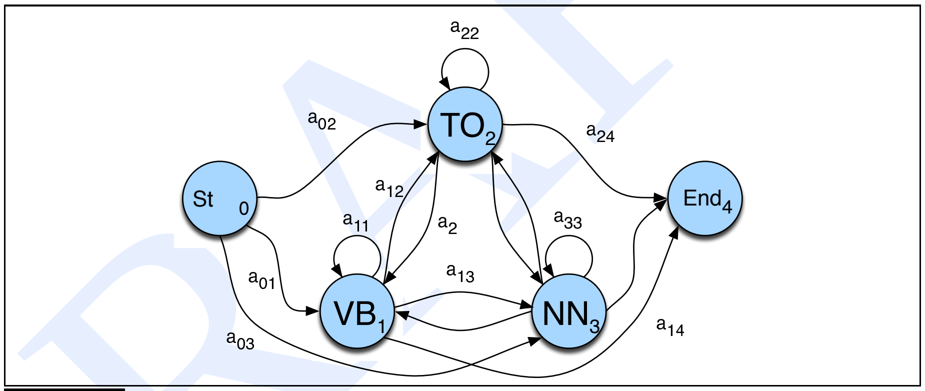


Figure 5.13 The Markov chain corresponding to the hidden states of the HMM. The A transition probabilities are used to compute the prior probability.

An HMM thus has two kinds of probabilities; the A transition probabilities, and the B observation likelihoods, corresponding respectively to the **prior** and **likelihood** probabilities that we saw in equation (5.31). Fig. 5.13 illustrates the prior probabilities in an HMM part-of-speech tagger, showing 3 sample states and some of the A transition probabilities between them. Fig. 5.14 shows another view of an HMM part-of-speech tagger, focusing on the word likelihoods B. Each hidden state is associated with a vector of likelihoods for each observation word.

5.5.3 The Viterbi Algorithm for HMM Tagging

For any model, such as an HMM, that contains hidden variables, the task of determining which sequence of variables is the underlying source of some sequence of observations is called the **decoding** task. The **Viterbi** algorithm is perhaps the most common decoding algorithm used for HMMs, whether for part-of-speech tagging or for speech recognition. The term **Viterbi** is common in speech and language processing, but this

decoding

Viterbi

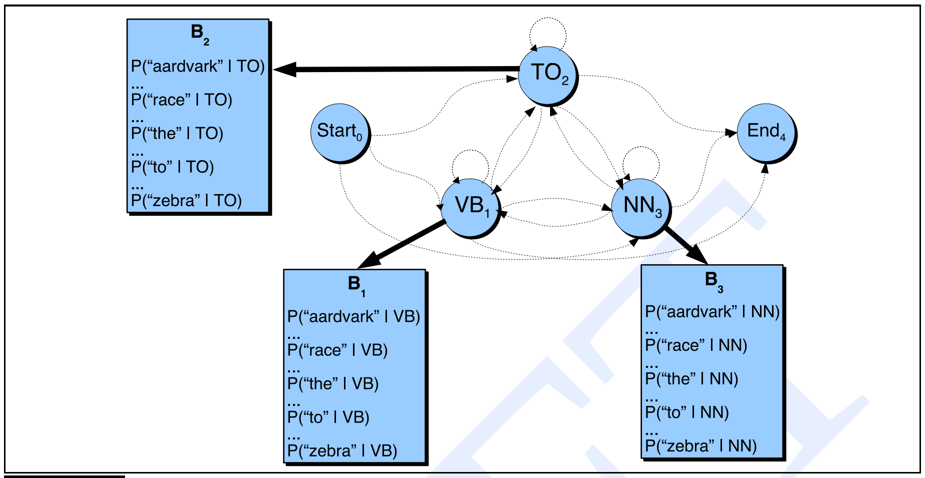


Figure 5.14 The B observation likelihoods for the HMM in the previous figure. Each state (except the non-emitting Start and End states) is associated with a vector of probabilities, one likelihood for each possible observation word.

is really a standard application of the classic **dynamic programming** algorithm, and looks a lot like the **minimum edit distance** algorithm of Ch. 3. The Viterbi algorithm was first applied to speech and language processing in the context of speech recognition by Vintsyuk (1968), but has what Kruskal (1983) calls a 'remarkable history of multiple independent discovery and publication'; see the History section at the end of Ch. 6 for more details.

The slightly simplified version of the Viterbi algorithm that we will present takes as input a single HMM and a set of observed words $O = (o_1 o_2 o_3 \ldots o_T)$ and returns the most probable state/tag sequence $Q = (q_1 q_2 q_3 \ldots q_T)$, together with its probability.

Let the HMM be defined by the two tables in Fig. 5.15 and Fig. 5.16. Fig. 5.15 expresses the a_{ij} probabilities, the *transition* probabilities between hidden states (i.e. part-of-speech tags). Fig. 5.16 expresses the $b_i(o_t)$ probabilities, the *observation* likelihoods of words given tags.

	VB	**TO**	**NN**	**PPSS**
<s>	.019	.0043	.041	.067
VB	.0038	.035	.047	.0070
TO	.83	0	.00047	0
NN	.0040	.016	.087	.0045
PPSS	.23	.00079	.0012	.00014

Figure 5.15 Tag transition probabilities (the a array, $p(t_i|t_{i-1})$) computed from the 87-tag Brown corpus without smoothing. The rows are labeled with the conditioning event; thus $P(PPSS|VB)$ is .0070. The symbol <s> is the start-of-sentence symbol.

Fig. 5.17 shows pseudocode for the Viterbi algorithm. The Viterbi algorithm sets up a probability matrix, with one column for each observation t and one row for each state in the state graph. Each column thus has a cell for each state q_i in the single combined automaton for the four words.

	I	**want**	**to**	**race**
VB	0	.0093	0	.00012
TO	0	0	.99	0
NN	0	.000054	0	.00057
PPSS	.37	0	0	0

Figure 5.16 Observation likelihoods (the *b* array) computed from the 87-tag Brown corpus without smoothing.

```
function VITERBI(observations of len T, state-graph of len N) returns best-path

  create a path probability matrix viterbi[N+2,T]
  for each state s from 1 to N do                              ;initialization step
      $viterbi[s,1] \leftarrow a_{0,s} * b_s(o_1)$
      $backpointer[s,1] \leftarrow 0$
  for each time step t from 2 to T do                          ;recursion step
    for each state s from 1 to N do
      $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
      $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s}$
  $viterbi[q_F,T] \leftarrow \max_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$              ; termination step
  $backpointer[q_F,T] \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T] * a_{s,q_F}$      ; termination step
   return the backtrace path by following backpointers to states back in time from
 $backpointer[q_F,T]$
```

Figure 5.17 Viterbi algorithm for finding optimal sequence of tags. Given an observation sequence and an HMM $\lambda = (A,B)$, the algorithm returns the state-path through the HMM which assigns maximum likelihood to the observation sequence. Note that states 0 and q_F are non-emitting.

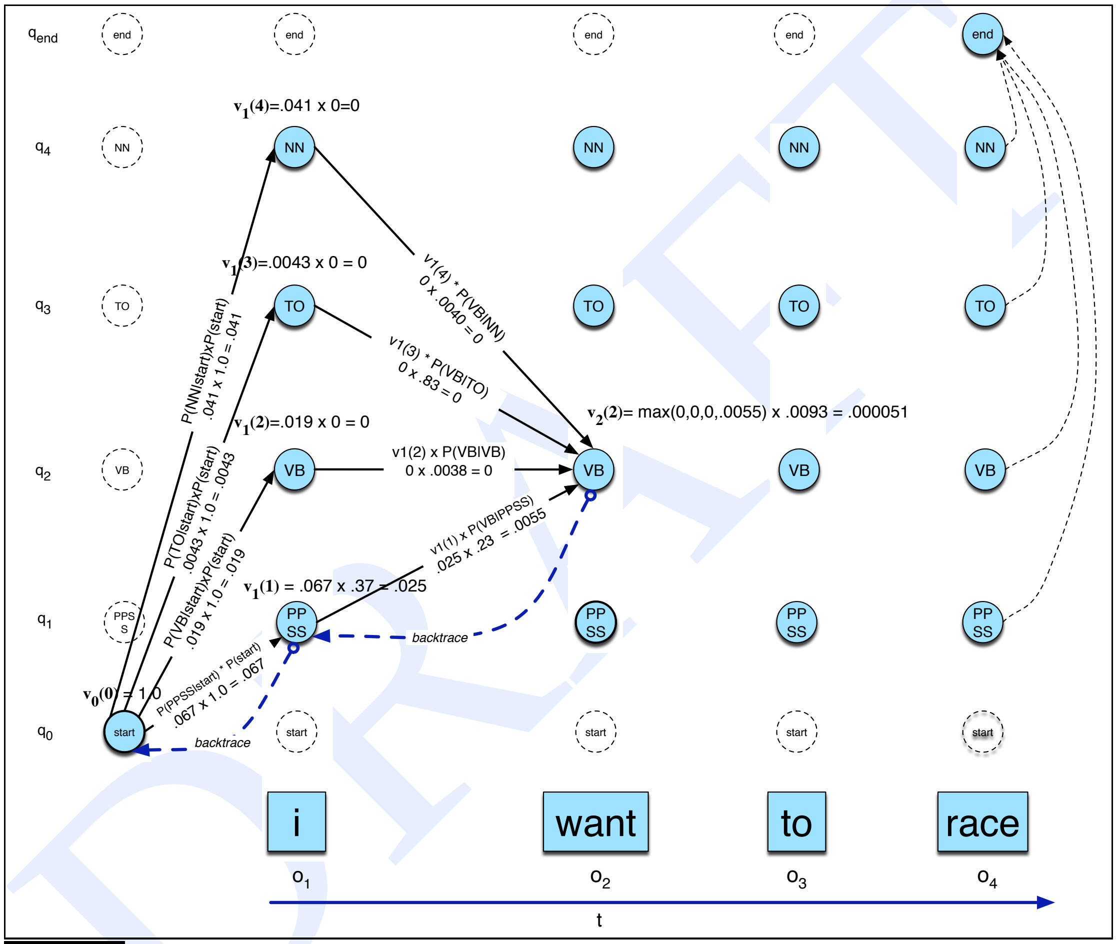
The algorithm first creates N or four state columns. The first column corresponds to the observation of the first word *i*, the second to the second word *want*, the third to the third word *to*, and the fourth to the fourth word *race*. We begin in the first column by setting the viterbi value in each cell to the product of the transition probability (into it from the state state) and the observation probability (of the first word); the reader should find this in Fig. 5.18.

Then we move on, column by column; for every state in column 1, we compute the probability of moving into each state in column 2, and so on. For each state q_j at time t, the value $viterbi[s,t]$ is computed by taking the maximum over the extensions of all the paths that lead to the current cell, following the following equation:

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t) \tag{5.38}$$

The three factors that are multiplied in Eq. 5.38 for extending the previous paths to compute the Viterbi probability at time t are:

$v_{t-1}(i)$	the **previous Viterbi path probability** from the previous time step
a_{ij}	the **transition probability** from previous state q_i to current state q_j
$b_j(o_t)$	the **state observation likelihood** of the observation symbol o_t given the current state j



q_end
q_4
q_3
q_2
q_1
q_0
end
NN
TO
VB
PPSS
start
v_1(4)=.041 x 0=0
v_1(3)=.0043 x 0 = 0
v_1(2)=.019 x 0 = 0
v_1(1) = .067 x .37 = .025
v_0(0) = 1.0
v_2(2)= max(0,0,0,.0055) x .0093 = .000051
P(NN|start)xP(start) .041 x 1.0 = .041
P(TO|start)xP(start) .0043 x 1.0 = .0043
P(VB|start)xP(start) .019 x 1.0 = .019
P(PPSS|start) * P(start) .067 x 1.0 = .067
v1(4) * P(VB|NN) 0 x .0040 = 0
v1(3) * P(VB|TO) 0 x .83 = 0
v1(2) x P(VB|VB) 0 x .0038 = 0
v1(1) x P(VB|PPSS) .025 x .23 = .0055
backtrace
backtrace
i
want
to
race
o_1
o_2
o_3
o_4
t

Figure 5.18 The entries in the individual state columns for the Viterbi algorithm. Each cell keeps the probability of the best path so far and a pointer to the previous cell along that path. We have only filled out columns 0 and 1 and one cell of column 2; the rest is left as an exercise for the reader. After the cells are filled in, backtracing from the *end* state, we should be able to reconstruct the correct state sequence PPSS VB TO VB.

In Fig. 5.18, each cell of the trellis in the column for the word *I* is computed by multiplying the previous probability at the start state (1.0), the transition probability from the start state to the tag for that cell, and the observation likelihood of the word *I* given the tag for that cell. As it turns out, three of the cells are zero (since the word *I* can be neither NN, TO nor VB). Next, each cell in the *want* column gets updated with the maximum probability path from the previous column. We have shown only the value for the VB cell. That cell gets the max of four values; as it happens in this case, three of them are zero (since there were zero values in the previous column). The remaining value is multiplied by the relevant transition probability, and the (trivial) max is taken. In this case the final value, .000051, comes from the PPSS state at the previous column.

The reader should fill in the rest of the trellis in Fig. 5.18, and backtrace to reconstruct the correct state sequence PPSS VB TO VB.

5.5.4 Extending the HMM algorithm to trigrams

We mentioned earlier that HMM taggers in actual use have a number of sophistications not present in the simplified tagger as we have described it so far. One important missing feature has to do with the tag context. In the tagger described above, we assume that the probability of a tag appearing is dependent only on the previous tag:

$$P(t_1^n) \approx \prod_{i=1}^{n} P(t_i|t_{i-1}) \tag{5.39}$$

Most modern HMM taggers actually use a little more of the history, letting the probability of a tag depend on the two previous tags:

$$P(t_1^n) \approx \prod_{i=1}^{n} P(t_i|t_{i-1},t_{i-2}) \tag{5.40}$$

In addition to increasing the window before a tagging decision, state-of-the-art HMM taggers like Brants (2000) let the tagger know the location of the end of the sentence by adding dependence on an end-of-sequence marker for t_{n+1}. This gives the following equation for part-of-speech tagging:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n|w_1^n) \approx \underset{t_1^n}{\operatorname{argmax}} \left[\prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1},t_{i-2}) \right] P(t_{n+1}|t_n) \tag{5.41}$$

In tagging any sentence with (5.41), three of the tags used in the context will fall off the edge of the sentence, and hence will not match regular words. These tags, t_{-1}, t_0, and t_{n+1}, can all be set to be a single special 'sentence boundary' tag which is added to the tagset. This requires that sentences passed to the tagger have sentence boundaries demarcated, as discussed in Ch. 3.

There is one large problem with (5.41); data sparsity. Any particular sequence of tags t_{i-2}, t_{i-1}, t_i that occurs in the test set may simply never have occurred in the training set. That means we cannot compute the tag trigram probability just by the maximum likelihood estimate from counts, following Eq. 5.42:

$$P(t_i|t_{i-1},t_{i-2}) = \frac{C(t_{i-2},t_{i-1},t_i)}{C(t_{i-2},t_{i-1})} : \tag{5.42}$$

Why not? Because many of these counts will be zero in any training set, and we will incorrectly predict that a given tag sequence will never occur! What we need is a way to estimate $P(t_i|t_{i-1},t_{i-2})$ even if the sequence t_{i-2},t_{i-1},t_i never occurs in the training data.

The standard approach to solve this problem is to estimate the probability by combining more robust, but weaker estimators. For example, if we've never seen the tag sequence PRP VB TO, so we can't compute P(TO|PRP,VB) from this frequency, we still could rely on the bigram probability P(TO|VB), or even the unigram probability P(TO). The maximum likelihood estimation of each of these probabilities can be computed from a corpus via the following counts:

$$\text{Trigrams} \quad \hat{P}(t_i|t_{i-1},t_{i-2}) = \frac{C(t_{i-2},t_{i-1},t_i)}{C(t_{i-2},t_{i-1})} \tag{5.43}$$

$$\text{Bigrams} \quad \hat{P}(t_i|t_{i-1}) = \frac{C(t_{i-1},t_i)}{C(t_{i-1})} \tag{5.44}$$

$$\text{Unigrams} \quad \hat{P}(t_i) = \frac{C(t_i)}{N} \tag{5.45}$$

How should these three estimators be combined in order to estimate the trigram probability $P(t_i|t_{i-1},t_{i-2})$? The simplest method of combination is linear interpolation. In linear interpolation, we estimate the probability $P(t_i|t_{i-1}t_{i-2})$ by a weighted sum of the unigram, bigram, and trigram probabilities:

$$P(t_i|t_{i-1}t_{i-2}) = \lambda_1\hat{P}(t_i|t_{i-1}t_{i-2})+\lambda_2\hat{P}(t_i|t_{i-1})+\lambda_3\hat{P}(t_i) \tag{5.46}$$

deleted interpolation

We require $\lambda_1+\lambda_2+\lambda_3=1$, insuring that the resulting P is a probability distribution. How should these λs be set? One good way is **deleted interpolation**, developed by Jelinek and Mercer (1980). In deleted interpolation, we successively delete each trigram from the training corpus, and choose the λs so as to maximize the likelihood of the rest of the corpus. The idea of the deletion is to set the λs in such a way as to generalize to unseen data and not overfit the training corpus. Fig. 5.19 gives the Brants (2000) version of the deleted interpolation algorithm for tag trigrams.

Brants (2000) achieves an accuracy of 96.7% on the Penn Treebank with a trigram HMM tagger. Weischedel et al. (1993) and DeRose (1988) have also reported accuracies of above 96% for HMM tagging. (Thede and Harper, 1999) offer a number of augmentations of the trigram HMM model, including the idea of conditioning word likelihoods on neighboring words and tags.

The HMM taggers we have seen so far are trained on hand-tagged data. Kupiec (1992), Cutting et al. (1992a), and others show that it is also possible to train an HMM tagger on unlabeled data, using the EM algorithm that we will introduce in Ch. 6. These taggers still start with a dictionary which lists which tags can be assigned to which words; the EM algorithm then learns the word likelihood function for each tag, and the tag transition probabilities. An experiment by Merialdo (1994), however, indicates that with even a small amount of training data, a tagger trained on hand-tagged data worked better than one trained via EM. Thus the EM-trained "pure HMM" tagger is

```
function DELETED-INTERPOLATION(corpus) returns $\lambda_1, \lambda_2, \lambda_3$

  $\lambda_1 \leftarrow 0$
  $\lambda_2 \leftarrow 0$
  $\lambda_3 \leftarrow 0$
  foreach trigram $t_1, t_2, t_3$ with $f(t_1, t_2, t_3) > 0$
    depending on the maximum of the following three values
      case $\frac{C(t_1,t_2,t_3)-1}{C(t_1,t_2)-1}$: increment $\lambda_3$ by $C(t_1, t_2, t_3)$
      case $\frac{C(t_2,t_3)-1}{C(t_2)-1}$: increment $\lambda_2$ by $C(t_1, t_2, t_3)$
      case $\frac{C(t_3)-1}{N-1}$: increment $\lambda_1$ by $C(t_1, t_2, t_3)$
    end
  end
  normalize $\lambda_1, \lambda_2, \lambda_3$
  return $\lambda_1, \lambda_2, \lambda_3$
```

Figure 5.19 The deleted interpolation algorithm for setting the weights for combining unigram, bigram, and trigram tag probabilities. If the denominator is 0 for any case, we define the result of that case to be 0. N is the total number of tokens in the corpus. After Brants (2000).

probably best suited to cases where no training data is available, for example when tagging languages for which there is no previously hand-tagged data.

5.6 Transformation-Based Tagging

Transformation-Based Learning

Transformation-Based Tagging, sometimes called Brill tagging, is an instance of the **Transformation-Based Learning** (TBL) approach to machine learning (Brill, 1995), and draws inspiration from both the rule-based and stochastic taggers. Like the rule-based taggers, TBL is based on rules that specify what tags should be assigned to what words. But like the stochastic taggers, TBL is a machine learning technique, in which rules are automatically induced from the data. Like some but not all of the HMM taggers, TBL is a supervised learning technique; it assumes a pre-tagged training corpus.

Samuel et al. (1998) offer a useful analogy for understanding the TBL paradigm, which they credit to Terry Harvey. Imagine an artist painting a picture of a white house with green trim against a blue sky. Suppose most of the picture was sky, and hence most of the picture was blue. The artist might begin by using a very broad brush and painting the entire canvas blue. Next she might switch to a somewhat smaller white brush, and paint the entire house white. She would just color in the whole house, not worrying about the brown roof, or the blue windows or the green gables. Next she takes a smaller brown brush and colors over the roof. Now she takes up the blue paint on a small brush and paints in the blue windows on the house. Finally she takes a very fine green brush and does the trim on the gables.

The painter starts with a broad brush that covers a lot of the canvas but colors a lot of areas that will have to be repainted. The next layer colors less of the canvas, but also makes fewer "mistakes". Each new layer uses a finer brush that corrects less of the picture, but makes fewer mistakes. TBL uses somewhat the same method as this painter. The TBL algorithm has a set of tagging rules. A corpus is first tagged using the broadest rule, that is, the one that applies to the most cases. Then a slightly more specific rule is chosen, which changes some of the original tags. Next an even narrower rule, which changes a smaller number of tags (some of which might be previously changed tags).

5.6.1 How TBL Rules Are Applied

Let's look at one of the rules used by Brill's (1995) tagger. Before the rules apply, the tagger labels every word with its most-likely tag. We get these most-likely tags from a tagged corpus. For example, in the Brown corpus, *race* is most likely to be a noun:

$$\begin{aligned} P(\text{NN}|\text{race}) &= .98 \\ P(\text{VB}|\text{race}) &= .02 \end{aligned}$$

This means that the two examples of *race* that we saw above will both be coded as NN. In the first case, this is a mistake, as NN is the incorrect tag:

(5.47) is/VBZ expected/VBN to/TO race/**NN** tomorrow/NN

In the second case this *race* is correctly tagged as an NN:

(5.48) the/DT race/**NN** for/IN outer/JJ space/NN

After selecting the most-likely tag, Brill's tagger applies its transformation rules. As it happens, Brill's tagger learned a rule that applies exactly to this mistagging of *race*:

Change NN *to* VB *when the previous tag is* TO

This rule would change *race/NN* to *race/VB* in exactly the following situation, since it is preceded by *to/TO*:

(5.49) expected/VBN to/TO race/NN → expected/VBN to/TO race/VB

5.6.2 How TBL Rules Are Learned

Brill's TBL algorithm has three major stages. It first labels every word with its most-likely tag. It then examines every possible transformation, and selects the one that results in the most improved tagging. Finally, it then re-tags the data according to this rule. The last two stages are repeated until some stopping criterion is reached, such as insufficient improvement over the previous pass. Note that stage two requires that TBL knows the correct tag of each word; that is, TBL is a supervised learning algorithm.

The output of the TBL process is an ordered list of transformations; these then constitute a "tagging procedure" that can be applied to a new corpus. In principle the set of possible transformations is infinite, since we could imagine transformations such as "transform NN to VB if the previous word was "IBM" and the word "the" occurs between 17 and 158 words before that". But TBL needs to consider every possible transformation, in order to pick the best one on each pass through the algorithm. Thus the algorithm needs a way to limit the set of transformations. This is done by designing a small set of **templates** (abstracted transformations). Every allowable transformation is an instantiation of one of the templates. Brill's set of templates is listed in Fig. 5.20. Fig. 5.21 gives the details of this algorithm for learning transformations.

templates

The preceding (following) word is tagged **z**.
The word two before (after) is tagged **z**.
One of the two preceding (following) words is tagged **z**.
One of the three preceding (following) words is tagged **z**.
The preceding word is tagged **z** and the following word is tagged **w**.
The preceding (following) word is tagged **z** and the word
    two before (after) is tagged **w**.

Figure 5.20 Brill's (1995) templates. Each begins with *"Change tag* **a** *to tag* **b** *when: ..."*. The variables **a**, **b**, **z**, and **w** range over parts-of-speech.

At the heart of Fig. 5.21 are the two functions GET_BEST_TRANSFORMATION and GET_BEST_INSTANCE. GET_BEST_TRANSFORMATION is called with a list of potential templates; for each template, it calls GET_BEST_INSTANCE. GET_BEST_-INSTANCE iteratively tests every possible instantiation of each template by filling in specific values for the tag variables **a**, **b**, **z**, and **w**.

In practice, there are a number of ways to make the algorithm more efficient. For example, templates and instantiated transformations can be suggested in a data-driven manner; a transformation-instance might only be suggested if it would improve the tagging of some specific word. The search can also be made more efficient by pre-indexing the words in the training corpus by potential transformation. Roche and Schabes (1997a) show how the tagger can also be speeded up by converting each rule into a finite-state transducer and composing all the transducers.

Fig. 5.6.2 shows a few of the rules learned by Brill's original tagger.

5.7 Evaluation and Error Analysis

The probabilities in a statistical model like an HMM POS-tagger come from the corpus it is trained on. We saw in Sec. 4.3 that in order to train statistical models like taggers or N-grams, we need to set aside a **training set**. The design of the training set or **training corpus** needs to be carefully considered. If the training corpus is too specific to the task or domain, the probabilities may be too narrow and not generalize well to tagging sentences in very different domains. But if the training corpus is too general, the probabilities may not do a sufficient job of reflecting the task or domain.

For evaluating N-grams models, we said in Sec. 4.3 that we need to divide our corpus into a distinct training set, test set, and a second test set called a development test set. We train our tagger on the training set. Then we use the **development test set** (also called a **devtest** set) to perhaps tune some parameters, and in general decide what the best model is. Then once we come up with what we think is the best model,

development test set
devtest

```
function TBL(corpus) returns transforms-queue
 INTIALIZE-WITH-MOST-LIKELY-TAGS(corpus)
 until end condition is met do
  templates ← GENERATE-POTENTIAL-RELEVANT-TEMPLATES
  best-transform ← GET-BEST-TRANSFORM(corpus, templates)
  APPLY-TRANSFORM(best-transform, corpus)
  ENQUEUE(best-transform-rule, transforms-queue)
 end
 return(transforms-queue)

function GET-BEST-TRANSFORM(corpus, templates) returns transform
  for each template in templates
    (instance, score) ← GET-BEST-INSTANCE(corpus, template)
    if (score > best-transform.score) then best-transform ← (instance, score)
  return(best-transform)

function GET-BEST-INSTANCE(corpus, template) returns transform
  for from-tag ← from $tag_1$ to $tag_n$ do
   for to-tag ← from $tag_1$ to $tag_n$ do
     for pos ← from 1 to corpus-size do
       if (correct-tag(pos) == to-tag && current-tag(pos) == from-tag)
            num-good-transforms(current-tag(pos−1))++
       elseif (correct-tag(pos)==from-tag && current-tag(pos)==from-tag)
            num-bad-transforms(current-tag(pos−1))++
     end
     best-Z ← $\text{ARGMAX}_t$(num-good-transforms(t) - num-bad-transforms(t))
      if(num-good-transforms(best-Z) - num-bad-transforms(best-Z)
                   > best-instance.score) then
       best.rule ← "Change tag from from-tag to to-tag if prev tag is best-Z"
       best.score ← num-good-transforms(best-Z) - num-bad-transforms(best-Z)
 return(best)

procedure APPLY-TRANSFORM(transform, corpus)
 for pos ← from 1 to corpus-size do
  if (current-tag(pos)==best-rule-from)
          && (current-tag(pos−1)==best-rule-prev))
     current-tag(pos) ← best-rule-to
```

Figure 5.21 The Brill (1995) TBL algorithm for learning to tag. GET_BEST_INSTANCE would have to change for transformation templates other than *"Change tag from X to Y if previous tag is Z"*.

we run it on the (hitherto unseen) test set to see its performance. We might use 80% of our data for training, and save 10% each for devtest and test. Why do we need a development test set distinct from the final test set? Because if we used the final test set to compute performance for all our experiments during our development phase, we would be tuning the various changes and parameters to this set. Our final error rate on

#	Change tags From	To	Condition	Example
1	NN	VB	Previous tag is TO	to/TO race/NN → VB
2	VBP	VB	One of the previous 3 tags is MD	might/MD vanish/VBP → VB
3	NN	VB	One of the previous 2 tags is MD	might/MD not reply/NN → VB
4	VB	NN	One of the previous 2 tags is DT	
5	VBD	VBN	One of the previous 3 tags is VBZ	

Figure 5.22 The first 20 nonlexicalized transformations from Brill (1995).

the test set would then be optimistic: it would underestimate the true error rate.

The problem with having a fixed training set, devset, and test set is that in order to save lots of data for training, the test set might not be large enough to be representative. Thus a better approach would be to somehow use **all** our data both for training and test. How is this possible? The idea is to use **crossvalidation**. In crossvalidation, we randomly choose a training and test set division of our data, train our tagger, and then compute the error rate on the test set. Then we repeat with a different randomly selected training set and test set. We do this sampling process 10 times, and then average these 10 runs to get an average error rate. This is called **10-fold crossvalidation**.

crossvalidation

10-fold crossvalidation

The only problem with cross-validation is that because all the data is used for testing, we need the whole corpus to be blind; we can't examine any of the data to suggest possible features, and in general see what's going on. But looking at the corpus is often important for designing the system. For this reason it is common to create a fixed training set and test set, and then to do 10-fold crossvalidation inside the training set, but compute error rate the normal way in the test set.

Once we have a test set, taggers are evaluated by comparing their labeling of the test set with a human-labeled **Gold Standard** test set, based on **accuracy**: the percentage of all tags in the test set where the tagger and the Gold standard agree. Most current tagging algorithms have an accuracy of around 96–97% for simple tagsets like the Penn Treebank set. These accuracies are for words and punctuation; the accuracy for words only would be lower.

How good is 97%? Since tagsets and tasks differ, the performance of tags can be compared against a lower-bound **baseline** and an upper-bound **ceiling**. One way to set a ceiling is to see how well humans do on the task. Marcus et al. (1993), for example, found that human annotators agreed on about 96–97% of the tags in the Penn Treebank version of the Brown corpus. This suggests that the Gold Standard may have a 3-4% margin of error, and that it is meaningless to get 100% accuracy, (modeling the last 3% would just be modeling noise). Indeed Ratnaparkhi (1996) showed that the tagging ambiguities that caused problems for his tagger were exactly the ones that humans had labeled inconsistently in the training set. Two experiments by Voutilainen (1995, p. 174), however, found that when humans were allowed to discuss tags, they reached consensus on 100% of the tags.

baseline

ceiling

Human Ceiling: When using a human Gold Standard to evaluate a classification algorithm, check the agreement rate of humans on the standard.

The standard **baseline**, suggested by Gale et al. (1992b) (in the slightly different context of word-sense disambiguation), is to choose the **unigram most-likely tag** for each ambiguous word. The most-likely tag for each word can be computed from a hand-tagged corpus (which may be the same as the training corpus for the tagger being evaluated).

Most Frequent Class Baseline: Always compare a classifier against a baseline at least as good as the most frequent class baseline (assigning each token to the class it occurred in most often in the training set).

Tagging algorithms since Harris (1962) incorporate this tag frequency intuition. Charniak et al. (1993) showed that this baseline algorithm achieves an accuracy of 90–91% on the 87-tag Brown tagset; Toutanova et al. (2003) showed that a more complex version, augmented with an unknown word model, achieved 93.69% on the 45-tag Treebank tagset.

When comparing models it is important to use statistical tests (introduced in any statistics class or textbook for the social sciences) to determine if the difference between two models is significant. Cohen (1995) is a useful reference which focuses on statistical research methods for artificial intelligence. Dietterich (1998) focuses on statistical tests for comparing classifiers. When statistically comparing sequence models like part-of-speech taggers, it is important to use **paired tests**. Commonly used paired tests for evaluating part-of-speech taggers include the **Wilcoxon signed-rank test**, **paired t-tests**, versions of matched t-tests such as the Matched-Pair Sentence Segment Word Error (**MAPSSWE**) test originally applied to speech recognition word error rate, and the **McNemar test** .

paired tests

Wilcoxon signed-rank test

paired t-tests

MAPSSWE

McNemar test

5.7.1 Error Analysis

In order to improve any model we need to understand where it went wrong. Analyzing the error in a classifier like a part-of-speech tagger is done via a **confusion matrix**, or **contingency table**. A confusion matrix for an N-way classification task is an N-by-N matrix where the cell (x,y) contains the number of times an item with correct classification x was classified by the model as y. For example, the following table shows a portion of the confusion matrix from the HMM tagging experiments of Franz (1996). The row labels indicate correct tags, column labels indicate the tagger's hypothesized tags, and each cell indicates percentage of the overall tagging error. Thus 4.4% of the total errors were caused by mistagging a VBD as a VBN. Common errors are highlighted.

	IN	**JJ**	**NN**	**NNP**	**RB**	**VBD**	**VBN**
IN	-	.2			.7		
JJ	.2	-	**3.3**	2.1	1.7	.2	**2.7**
NN		**8.7**	-				.2
NNP	.2	**3.3**	**4.1**	-	.2		
RB	**2.2**	2.0	.5		-		
VBD		.3	.5			-	**4.4**
VBN		**2.8**				**2.6**	-

The confusion matrix above, and related error analyses in Franz (1996), Kupiec (1992), and Ratnaparkhi (1996), suggest that some major problems facing current taggers are:

1. **NN versus NNP versus JJ:** These are hard to distinguish prenominally. Distinguishing proper nouns is especially important for information extraction and machine translation.
2. **RP versus RB versus IN:** All of these can appear in sequences of satellites immediately following the verb.
3. **VBD versus VBN versus JJ:** Distinguishing these is important for partial parsing (participles are used to find passives), and for correctly labeling the edges of noun-phrases.

Error analysis like this is a crucial part of any computational linguistic application. Error analysis can help find bugs, find problems in the training data, and, most important, help in developing new kinds of knowledge or algorithms to use in solving problems.

5.8 Advanced Issues in Part-of-Speech Tagging

5.8.1 Practical Issues: Tag Indeterminacy and Tokenization

Tag indeterminacy arises when a word is ambiguous between multiple tags and it is impossible or very difficult to disambiguate. In this case, some taggers allow the use of multiple tags. This is the case in both the Penn Treebank and in the British National Corpus. Common tag indeterminacies include adjective versus preterite versus past participle (JJ/VBD/VBN), and adjective versus noun as prenominal modifier (JJ/NN). Given a corpus with these indeterminate tags, there are 3 ways to deal with tag indeterminacy when training and scoring part-of-speech taggers:

1. Somehow replace the indeterminate tags with only one tag.
2. In testing, count a tagger as having correctly tagged an indeterminate token if it gives either of the correct tags. In training, somehow choose only one of the tags for the word.
3. Treat the indeterminate tag as a single complex tag.

The second approach is perhaps the most sensible, although most previous published results seem to have used the third approach. This third approach applied to the Penn Treebank Brown corpus, for example, results in a much larger tagset of 85 tags instead of 45, but the additional 40 complex tags cover a total of only 121 word instances out of the million word corpus.

Most tagging algorithms assume a process of tokenization has been applied to the tags. Ch. 3 discussed the issue of tokenization of periods for distinguishing sentence-final periods from word-internal period in words like *etc.*. An additional role for tokenization is in word splitting. The Penn Treebank and the British National Corpus split contractions and the *'s*-genitive from their stems:

would/MD n't/RB
children/NNS 's/POS

Indeed, the special Treebank tag POS is used only for the morpheme *'s* which must be segmented off during tokenization.

Another tokenization issue concerns multi-part words. The Treebank tagset assumes that tokenization of words like *New York* is done at whitespace. The phrase *a New York City firm* is tagged in Treebank notation as five separate words: *a/DT New/NNP York/NNP City/NNP firm/NN*. The C5 tagset, by contrast, allow prepositions like "*in terms of*" to be treated as a single word by adding numbers to each tag, as in *in/II31 terms/II32 of/II33*.

5.8.2 Unknown Words

words people
never use —
could be
only I
know them
Ishikawa Takuboku 1885–1912

All the tagging algorithms we have discussed require a dictionary that lists the possible parts-of-speech of every word. But the largest dictionary will still not contain every possible word, as we saw in Ch. 7. Proper names and acronyms are created very often, and even new common nouns and verbs enter the language at a surprising rate. Therefore in order to build a complete tagger we cannot always use a dictionary to give us $p(w_i|t_i)$. We need some method for guessing the tag of an unknown word.

The simplest possible unknown-word algorithm is to pretend that each unknown word is ambiguous among all possible tags, with equal probability. Then the tagger must rely solely on the contextual POS-trigrams to suggest the proper tag. A slightly more complex algorithm is based on the idea that the probability distribution of tags over unknown words is very similar to the distribution of tags over words that occurred only once in a training set, an idea that was suggested by both Baayen and Sproat (1996) and Dermatas and Kokkinakis (1995). These words that only occur once are known as **hapax legomena** (singular **hapax legomenon**). For example, unknown words and *hapax legomena* are similar in that they are both most likely to be nouns, followed by verbs, but are very unlikely to be determiners or interjections. Thus the likelihood $P(w_i|t_i)$ for an unknown word is determined by the average of the distribution over all singleton words in the training set. This idea of using "things we've seen once" as an estimator for "things we've never seen" will prove useful in the Good-Turing algorithm of Ch. 4.

hapax legomena

Most unknown-word algorithms, however, make use of a much more powerful source of information: the morphology of the words. For example, words that end in *-s* are likely to be plural nouns (NNS), words ending with *-ed* tend to be past participles (VBN), words ending with *able* tend to be adjectives (JJ), and so on. Even if we've never seen a word, we can use facts about its morphological form to guess its part-of-speech. Besides morphological knowledge, orthographic information can be very helpful. For example words starting with capital letters are likely to be proper nouns (NP). The presence of a hyphen is also a useful feature; hyphenated words in the Treebank version of Brown are most likely to be adjectives (JJ). This prevalence of JJs is caused by the labeling instructions for the Treebank, which specified that prenominal modifiers should be labeled as JJ if they contained a hyphen.

How are these features combined and used in part-of-speech taggers? One method is to train separate probability estimators for each feature, assume independence, and multiply the probabilities. Weischedel et al. (1993) built such a model, based on four specific kinds of morphological and orthographic features. They used 3 inflectional endings (*-ed*, *-s*, *-ing*), 32 derivational endings (such as *-ion*, *-al*, *-ive*, and *-ly*), 4 values of capitalization depending on whether a word is sentence-initial (+/- capitalization, +/- initial) and whether the word was hyphenated. For each feature, they trained maximum likelihood estimates of the probability of the feature given a tag from a labeled training set. They then combined the features to estimate the probability of an unknown word by assuming independence and multiplying:

(5.50) $$P(w_i|t_i) = p(\text{unknown-word}|t_i) * p(\text{capital}|t_i) * p(\text{endings/hyph}|t_i)$$

Another HMM-based approach, due to Samuelsson (1993) and Brants (2000), generalizes this use of morphology in a data-driven way. In this approach, rather than pre-selecting certain suffixes by hand, all final letter sequences of all words are considered. They consider such suffixes of up to ten letters, computing for each suffix of length i the probability of the tag t_i given the suffix:

(5.51) $$P(t_i|l_{n-i+1}\ldots l_n)$$

These probabilities are smoothed using successively shorter and shorter suffixes. Separate suffix tries are kept for capitalized and uncapitalized words.

In general, most unknown word models try to capture the fact that unknown words are unlikely to be closed-class words like prepositions. Brants models this fact by only computing suffix probabilities from the training set for words whose frequency in the training set is ≤ 10. In the HMM tagging model of Thede and Harper (1999), this fact is modeled instead by only training on open-class words.

Note that (5.51) gives an estimate of $p(t_i|w_i)$; since for the HMM tagging approach we need the likelihood $p(w_i|t_i)$, this can be derived from (5.51) using Bayesian inversion (i.e. using Bayes rule and computation of the two priors $P(t_i)$ and $P(t_i|l_{n-i+1}\ldots l_n)$).

In addition to using capitalization information for unknown words, Brants (2000) also uses capitalization information for tagging known words, by adding a capitalization feature to each tag. Thus instead of computing $P(t_i|t_{i-1},t_{i-2})$ as in (5.44), he actually computes the probability $P(t_i,c_i|t_{i-1},c_{i-1},t_{i-2},c_{i-2})$. This is equivalent to having a capitalized and uncapitalized version of each tag, essentially doubling the size of the tagset.

A non-HMM based approach to unknown word detection was that of Brill (1995) using the TBL algorithm, where the allowable templates were defined orthographically (the first N letters of the words, the last N letters of the word, etc.).

Most recent approaches to unknown word handling, however, combine these features in a third way: by using maximum entropy (**MaxEnt**) models such as the **Maximum Entropy Markov Model** (**MEMM**) first introduced by Ratnaparkhi (1996) and McCallum et al. (2000), and which we will study in Ch. 6. The maximum entropy approach is one a family of log-linear approaches to classification in which many features are computed for the word to be tagged, and all the features are combined in a model based on multinomial logistic regression. The unknown word model in the tagger of Toutanova et al. (2003) uses a feature set extended from Ratnaparkhi (1996), in which each feature represents a property of a word, including features like:

word contains a number
word contains an upper-case letter
word contains a hyphen
word is all upper-case
word contains a particular prefix (from the set of all prefixes of length ≤ 4)
word contains a particular suffix (from the set of all prefixes of length ≤ 4)
word is upper-case and has a digit and a dash (like *CFC-12*)
word is upper-case and followed within 3 words by Co., Inc., etc

Toutanova et al. (2003) found this last feature, implementing a simple company name detector, to be particularly useful. 3 words by a word like Co. or Inc. Note that the Ratnaparkhi (1996) model ignored all features with counts less than 10.

Loglinear models have also been applied to Chinese tagging by Tseng et al. (2005b). Chinese words are very short (around 2.4 characters per unknown word compared with 7.7 for English), but Tseng et al. (2005b) found that morphological features nonetheless gave a huge increase in tagging performance for unknown words. For example for each character in an unknown word and each POS tag, they added a binary feature indicating whether that character ever occurred with that tag in any training set word. There is also an interesting distributional difference in unknown words between Chinese and English. While English unknown words tend to be proper nouns (41% of unknown words in WSJ are NP), in Chinese the majority of unknown words are common nouns and verbs (61% in the Chinese TreeBank 5.0). These ratios are similar to German, and seem to be caused by the prevalence of compounding as a morphological device in Chinese and German.

5.8.3 Part-of-Speech Tagging for Other Languages

As the previous paragraph suggests, part-of-speech tagging algorithms have all been applied to many other languages as well. In some cases, the methods work well without large modifications; Brants (2000) showed the exact same performance for tagging on the German NEGRA corpus (96.7%) as on the English Penn Treebank. But a number of augmentations and changes become necessary when dealing with highly inflected or agglutinative languages.

One problem with these languages is simply the large number of words, when compared to English. Recall from Ch. 3 that agglutinative languages like Turkish (and to some extent mixed agglutinative-inflectional languages like Hungarian) are those in which words contain long strings of morphemes, where each morpheme has relatively few surface forms, and so it is often possible to clearly see the morphemes in the surface text. For example Megyesi (1999) gives the following typical example of a Hungarian word meaning “of their hits”:

(5.52) találataiknak

talál	*-at*	*-a*	*-i*	*-k*	*-nak*
hit/find	nominalizer	his	poss.plur	their	dat/gen

"of their hits"

Similarly, the following list, excerpted from Hakkani-Tür et al. (2002), shows a few of the words producible in Turkish from the root *uyu-*, 'sleep':

uyuyorum	'I am sleeping'	uyuyorsun	'you are sleeping'
uyuduk	'we slept'	uyumadan	'without sleeping'
uyuman	'your sleeping'	uyurken	'while (somebody) is sleeping'
uyutmak	'to cause someone to sleep'	uyutturmak	'to cause someone to cause another person to sleep'

These productive word-formation processes result in a large vocabulary for these languages. Oravecz and Dienes (2002), for example, show that a quarter-million word corpus of English has about 19,000 different words (i.e. word types); the same size corpus of Hungarian has almost 50,000 different words. This problem continues even with much larger corpora; note in the table below on Turkish from Hakkani-Tür et al. (2002) that the vocabulary size of Turkish is far bigger than that of English and is growing faster than English even at 10 million words.

Corpus Size	**Vocabulary Size**	
	Turkish	**English**
1M words	106,547	33,398
10M words	417,775	97,734

The large vocabulary size seems to cause a significant degradation in tagging performance when the HMM algorithm is applied directly to agglutinative languages. For example Oravecz and Dienes (2002) applied the exact same HMM software (called 'TnT') that Brants (2000) used to achieve 96.7% on both English and German, and achieved only 92.88% on Hungarian. The performance on known words (98.32%) was comparable to English results; the problem was the performance on unknown words: 67.07% on Hungarian, compared to around 84-85% for unknown words with a comparable amount of English training data. Hajič (2000) notes the same problem in a wide variety of other languages (including Czech, Slovene, Estonian, and Romanian); the performance of these taggers is hugely improved by adding a dictionary which essentially gives a better model of unknown words. In summary, one difficulty in tagging highly inflected and agglutinative languages is tagging of unknown words.

A second, related issue with such languages is the vast amount of information that is coded in the morphology of the word. In English, lots of information about syntactic function of a word is represented by word order, or neighboring function words. In highly inflectional languages, information such as the case (nominative, accusative, genitive) or gender (masculine, feminine) is marked on the words themselves, and word order plays less of a role in marking syntactic function. Since tagging is often used a preprocessing step for other NLP algorithms such as parsing or information extraction, this morphological information is crucial to extract. This means that a part-of-speech tagging output for Turkish or Czech needs to include information about the case and gender of each word in order to be as useful as parts-of-speech without case or gender are in English.

For this reason, tagsets for agglutinative and highly inflectional languages are usually much larger than the 50-100 tags we have seen for English. Tags in such enriched tagsets are sequences of morphological tags rather than a single primitive tag. Assigning tags from such a tagset to words means that we are jointly solving the problems of part-of-speech tagging and morphological disambiguation. Hakkani-Tür et al. (2002) give the following example of tags from Turkish, in which the word *izin* has three possible morphological/part-of-speech tags (and meanings):

1. Yerdeki **izin** temizlenmesi gerek. iz + `Noun+A3sg+Pnon+Gen`
   The trace on the floor should be cleaned.
2. Üzerinde parmak **izin** kalmiş iz + `Noun+A3sg+P2sg+Nom`
   Your finger **print** is left on (it).
3. Içeri girmek için **izin** alman gerekiyor. izin + `Noun+A3sg+Pnon+Nom`
   You need a **permission** to enter.

Using a morphological parse sequence like `Noun+A3sg+Pnon+Gen` as the part-of-speech tag greatly increases the number of parts-of-speech, of course. We can see this clearly in the morphologically tagged MULTEXT-East corpora, in English, Czech, Estonian, Hungarian, Romanian, and Slovene (Dimitrova et al., 1998; Erjavec, 2004). Hajič (2000) gives the following tagset sizes for these corpora:

Language	Tagset Size
English	139
Czech	970
Estonian	476
Hungarian	401
Romanian	486
Slovene	1033

With such large tagsets, it is generally necessary to perform morphological analysis on each word to generate the list of possible morphological tag sequences (i.e. the list of possible part-of-speech tags) for the word. The role of the tagger is then to disambiguate among these tags. The morphological analysis can be done in various ways. The Hakkani-Tür et al. (2002) model of Turkish morphological analysis is based on the two-level morphology we introduced in Ch. 3. For Czech and the MULTEXT-East languages, Hajič (2000) and Hajič and Hladká (1998) use a fixed external dictionary for each language which compiles out all the possible forms of each word, and lists possible tags for each wordform. The morphological parse also crucially helps address the problem of unknown words, since morphological parsers can accept unknown stems and still segment the affixes properly.

Given such a morphological parse, various methods for the tagging itself can be used. The Hakkani-Tür et al. (2002) model for Turkish uses a Markov model of tag sequences. The model assigns a probability to sequences of tags like

`izin+Noun+A3sg+Pnon+Nom`

by computing tag transition probabilities from a training set. Other models use similar techniques to those for English. Hajič (2000) and Hajič and Hladká (1998), for example, use a log-linear exponential tagger for the MULTEXT-East languages, Oravecz and Dienes (2002) and Džeroski et al. (2000) use the TnT HMM tagger (Brants, 2000), and so on.

5.8.4 Combining Taggers

The various part-of-speech tagging algorithms we have described can also be combined. The most common approach to tagger combination is to run multiple taggers in parallel on the same sentence, and then combine their output, either by voting or by training another classifier to choose which tagger to trust in a given context. Brill and Wu (1998), for example, combined unigram, HMM, TBL, and maximum-entropy taggers by voting via a higher-order classifier, and showed a small gain over the best of the four classifiers. In general, this kind of combination is only useful if the taggers have complementary errors, and so research on combination often begins by checking to see if the errors are indeed different from different taggers. Another option is to combine taggers in series. Hajič et al. (2001) apply this option for Czech, using the rule-based approach to remove some of the impossible tag possibilities for each word, and then an HMM tagger to choose the best sequence from the remaining tags.

5.9 Advanced: The Noisy Channel Model for Spelling

The Bayesian inference model introduced in Sec. 5.5 for tagging has another interpretation: as an implementation of the **noisy channel** model, a crucial tool in speech recognition and machine translation.

In this section we introduce this noisy channel model and show how to apply it to the task of correcting spelling errors. The noisy channel model is used in Microsoft Word and in many search engines, and in general is the most widely used algorithm for correcting any kind of single-word spelling error, including **non-word spelling errors** and for **real-word spelling errors**.

Recall that non-word spelling errors are those which are not English words (like *recieve* for *receive*), and we can **detect** these by simply looking for any word not in a dictionary. We saw in Sec. 3.10 that candidate corrections for some spelling errors could be found by looking for words that had a small **edit distance** to the misspelled word.

The Bayesian models we have seen in this chapter, and the noisy channel model, will give us a better way to find these corrections. Furthermore, we'll be able to use the noisy channel model for **contextual spell checking**, which is the task of correcting **real-word spelling errors** like the following:

They are leaving in about fifteen *minuets* to go to her house.
The study was conducted mainly *be* John Black.

Since these errors have real words, we can't find them by just flagging words not in the dictionary, and we can't correct them just using edit distance alone. But note that words around the candidate correction *in about fifteen minutes* make it a much more probable word sequence than the original *in about fifteen minuets*. The noisy channel model will implement this idea via N-gram models.

noisy channel The intuition of the **noisy channel** model (see Fig. 5.23) is to treat the misspelled word as if a correctly-spelled word had been 'distorted' by being passed through a noisy

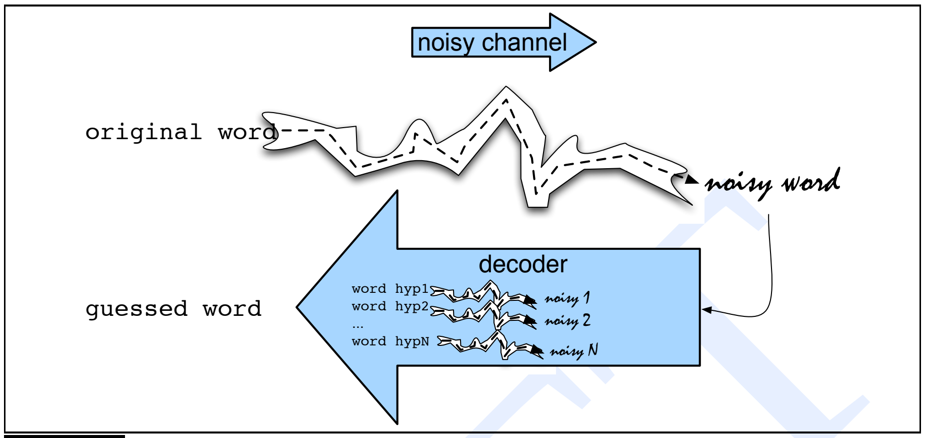


Figure 5.23 In the noisy channel model, we imagine that the surface form we see is actually a 'distorted' form of an original word passed through a noisy channel. The decoder passes each hypothesis through a model of this channel and picks the word that best matches the surface noisy word.

communication channel. This channel introduces "noise" in the form of substitutions or other changes to the letters which makes it hard to recognize the "true" word. Our goal is then to build a model of the channel. Given this model, we then find the true word by taking every word of the language, passing each word through our model of the noisy channel, and seeing which one comes the closest to the misspelled word.

This noisy channel model, like the HMM tagging architecture we saw earlier, is a special case of **Bayesian inference**. We see an observation O (a misspelled word) and our job is to find the word w which generated this misspelled word. Out of all possible words in the vocabulary V we want to find the word w such that $P(w|O)$ is highest, or:

Bayesian

V

$$\hat{w} = \underset{w \in V}{\operatorname{argmax}} P(w|O) \tag{5.53}$$

As we saw for part-of-speech tagging, we will use Bayes rule to turn the problem around (and note that, as for tagging, we can ignore the denominator):

$$\hat{w} = \underset{w \in V}{\operatorname{argmax}} \frac{P(O|w)P(w)}{P(O)} = \underset{w \in V}{\operatorname{argmax}} P(O|w)\,P(w) \tag{5.54}$$

To summarize, the noisy channel model says that we have some true underlying word w, and we have a noisy channel which modifies the word into some possible misspelled surface form. The probability of the noisy channel producing any particular observation sequence O is modeled by $P(O|w)$. The probability distribution over possible hidden words is modeled by $P(w)$. The most probable word $\hat{w}$ given that we've seen some observed misspelling O can be computed by taking the product of the word prior $P(w)$ and the observation likelihood $P(O|w)$ and choosing the word for which this product is greatest.

Let's apply the noisy channel approach to correcting non-word spelling errors. This approach was first suggested by Kernighan et al. (1990); their program, `correct`, takes words rejected by the Unix `spell` program, generates a list of potential correct words, ranks them according to Eq. 5.54, and picks the highest-ranked one. We'll apply the algorithm to the example misspelling *acress*. The algorithm has two stages: *proposing candidate corrections* and *scoring the candidates*.

In order to propose candidate corrections Kernighan et al. make the reasonable (Damerau, 1964) simplifying assumption that the correct word will differ from the misspelling by a single insertion, deletion, substitution, or transposition. The list of candidate words is generated from the typo by applying any single transformation which results in a word in a large on-line dictionary. Applying all possible transformations to *acress* yields the list of candidate words in Fig. 5.24.

		Transformation			
		Correct	**Error**	**Position**	
Error	**Correction**	**Letter**	**Letter**	**(Letter #)**	**Type**
acress	actress	t	–	2	deletion
acress	cress	–	a	0	insertion
acress	caress	ca	ac	0	transposition
acress	access	c	r	2	substitution
acress	across	o	e	3	substitution
acress	acres	–	2	5	insertion
acress	acres	–	2	4	insertion

Figure 5.24 Candidate corrections for the misspelling *acress* and the transformations that would have produced the error (after Kernighan et al. (1990)). "–" represents a null letter.

The second stage of the algorithm scores each correction by Eq. 5.54. Let t represent the typo (the misspelled word), and let c range over the set C of candidate corrections. The most likely correction is then:

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \; \overbrace{P(t|c)}^{\text{likelihood}} \; \overbrace{P(c)}^{\text{prior}} \tag{5.55}$$

The prior probability of each correction $P(c)$ is the language model probability of the word c in context; for in this section for pedagogical reasons we'll make the simplifying assumption that this is the unigram probability $P(c)$, but in practice in spelling correction this is extended to trigram or 4-gram probabilities. Let's use the corpus of Kernighan et al. (1990), which is the 1988 AP newswire corpus of 44 million words. Since in this corpus the word *actress* occurs 1343 times out of 44 million, the word *acres* 2879 times, and so on, the resulting unigram prior probabilities are as follows:

c	**freq(c)**	**p(c)**
actress	1343	.0000315
cress	0	.000000014
caress	4	.0000001
access	2280	.000058
across	8436	.00019
acres	2879	.000065

How can we estimate $P(t|c)$? It is very difficult to model the actual channel perfectly (i.e. computing the exact probability that a word will be mistyped) because it would require knowing who the typist was, whether they were left-handed or right-handed, and many other factors. Luckily, it turns out we can get a pretty reasonable estimate of $p(t|c)$ just by looking at simple local context factors, because the most important factors predicting an insertion, deletion, transposition are the identity of the correct letter itself, how the letter was misspelled, and the surrounding context. For example, the letters *m* and *n* are often substituted for each other; this is partly a fact about their identity (these two letters are pronounced similarly and they are next to each other on the keyboard), and partly a fact about context (because they are pronounced similarly, they occur in similar contexts). Kernighan et al. (1990) used a simple model of this sort. They estimated e.g. $p(acress|across)$ just using the number of times that the letter *e* was substituted for the letter *o* in some large corpus of errors. This is represented by a **confusion matrix**, a square 26×26 matrix which represents the number of times one letter was incorrectly used instead of another. For example, the cell labeled $[o,e]$ in a substitution confusion matrix would give the count of times that *e* was substituted for *o*. The cell labeled $[t,s]$ in an insertion confusion matrix would give the count of times that *t* was inserted after *s*. A confusion matrix can be computed by coding a collection of spelling errors with the correct spelling and then counting the number of times different errors occurred (Grudin, 1983). Kernighan et al. (1990) used four confusion matrices, one for each type of single error:

confusion matrix

- del$[x,y]$ contains the number of times in the training set that the characters *xy* in the correct word were typed as *x*.
- ins$[x,y]$ contains the number of times in the training set that the character *x* in the correct word was typed as *xy*.
- sub$[x,y]$ the number of times that *x* was typed as *y*.
- trans$[x,y]$ the number of times that *xy* was typed as *yx*.

Note that they chose to condition their insertion and deletion probabilities on the previous character; they could also have chosen to condition on the following character. Using these matrices, they estimated $p(t|c)$ as follows (where c_p is the *p*th character of the word *c*):

$$P(t|c) = \begin{cases} \frac{\text{del}[c_{p-1},c_p]}{\text{count}[c_{p-1}c_p]}, \text{ if deletion} \\ \frac{\text{ins}[c_{p-1},t_p]}{\text{count}[c_{p-1}]}, \text{ if insertion} \\ \frac{\text{sub}[t_p,c_p]}{\text{count}[c_p]}, \text{ if substitution} \\ \frac{\text{trans}[c_p,c_{p+1}]}{\text{count}[c_p c_{p+1}]}, \text{ if transposition} \end{cases} \quad (5.56)$$

Fig. 5.25 shows the final probabilities for each of the potential corrections; the unigram prior is multiplied by the likelihood (computed using Eq. 5.56 and the confusion matrices). The final column shows the “normalized percentage”.

This implementation of the Bayesian algorithm predicts *acres* as the correct word (at a total normalized percentage of 45%), and *actress* as the second most likely word.

c	**freq(c)**	**p(c)**	**p(t|c)**	**p(t|c)p(c)**	**%**
actress	1343	.0000315	.000117	3.69×10^{-9}	**37%**
cress	0	.000000014	.00000144	2.02×10^{-14}	**0%**
caress	4	.0000001	.00000164	1.64×10^{-13}	**0%**
access	2280	.000058	.000000209	1.21×10^{-11}	**0%**
across	8436	.00019	.0000093	1.77×10^{-9}	**18%**
acres	2879	.000065	.0000321	2.09×10^{-9}	**21%**
acres	2879	.000065	.0000342	2.22×10^{-9}	**23%**

Figure 5.25 Computation of the ranking for each candidate correction. Note that the highest ranked word is not *actress* but *acres* (the two lines at the bottom of the table), since *acres* can be generated in two ways. The *del*[], *ins*[], *sub*[], and *trans*[] confusion matrices are given in full in Kernighan et al. (1990).

Unfortunately, the algorithm was wrong here: The writer's intention becomes clear from the context: *...was called a "stellar and versatile* **acress** *whose combination of sass and glamour has defined her...*". The surrounding words make it clear that *actress* and not *acres* was the intended word. This is the reason that in practice we use trigram (or larger) language models in the noisy channel model, rather thaan unigrams. Seeing whether a **bigram** model of $P(c)$ correctly solves this problem is left as Exercise 10 for the reader.

The algorithm as we have described it requires hand-annotated data to train the confusion matrices. An alternative approach used by Kernighan et al. (1990) is to compute the matrices by iteratively using this very spelling error correction algorithm itself. The iterative algorithm first initializes the matrices with equal values; thus any character is equally likely to be deleted, equally likely to be substituted for any other character, etc. Next the spelling error correction algorithm is run on a set of spelling errors. Given the set of typos paired with their corrections, the confusion matrices can now be recomputed, the spelling algorithm run again, and so on. This clever method turns out to be an instance of the important **EM** algorithm (Dempster et al., 1977) that we will discuss in Ch. 6.

5.9.1 Contextual Spelling Error Correction

real-word error detection

As we mentioned above, the noisy channel approach can also be applied to detect and correct **real-word spelling errors,** errors that result in an actual word of English. This can happen from typographical errors (insertion, deletion, transposition) that accidentally produce a real word (e.g., *there* for *three*), or because the writer substituted the wrong spelling of a homophone or near-homophone (e.g., *dessert* for *desert*, or *piece* for *peace*). The task of correcting these errors is also called **context-sensitive spell correction**. A number of studies suggest that between of 25% and 40% of spelling errors are valid English words (Kukich, 1992); some of Kukich's examples include:

context-sensitive spell correction

They are leaving in about fifteen *minuets* to go to her house.
The design *an* construction of the system will take more than a year.
Can they *lave* him my messages?
The study was conducted mainly *be* John Black.

We can extend the noisy channel model to deal with real-word spelling errors by generating a *candidate spelling set* for every word in a sentence (Mays et al., 1991). The candidate set includes the word itself, plus every English word that would be generated from the word by either typographical modifications (letter insertion, deletion, substitution), or from a homophone list. The algorithm then chooses the spelling for each word that gives the whole sentence the highest probability. That is, given a sentence $W = \{w_1, w_2, \ldots, w_k, \ldots, w_n\}$, where w_k has alternative spelling w'_k, w''_k, etc., we choose the spelling among these possible spellings that maximizes $P(W)$, using the N-gram grammar to compute $P(W)$.

More recent research has focused on improving the channel model $P(t|c)$, such as by incorporating phonetic information, or allowing more complex errors (Brill and Moore, 2000; Toutanova and Moore, 2002). The most important improvement to the language model $P(c)$ is to use very large contexts, for example by using the very large set of 5-grams publicly released by Google in 2006 (Franz and Brants, 2006). See Norvig (2007) for a nice explanation and Python implementation of the noisy channel model; the end of the chapter has further pointers.

5.10 Summary

This chapter introduced the idea of **parts-of-speech** and **part-of-speech tagging**. The main ideas:

- Languages generally have a relatively small set of **closed class** words, which are often highly frequent, generally act as **function words**, and can be very ambiguous in their part-of-speech tags. Open class words generally include various kinds of **nouns**, **verbs**, **adjectives**. There are a number of part-of-speech coding schemes, based on **tagsets** of between 40 and 200 tags.
- **Part-of-speech tagging** is the process of assigning a part-of-speech label to each of a sequence of words. Rule-based taggers use hand-written rules to distinguish tag ambiguity. HMM taggers choose the tag sequence which maximizes the product of word likelihood and tag sequence probability. Other machine learning models used for tagging include maximum entropy and other log-linear models, decision trees, memory-based learning, and transformation-based learning.
- The probabilities in HMM taggers are trained on hand-labeled training corpora, combining different N-gram levels using deleted interpolation, and using sophisticated unknown word models.
- Given an HMM and an input string, the Viterbi algorithm is used to decode the optimal tag sequence.
- Taggers are evaluated by comparing their output from a test set to human labels for that test set. Error analysis can help pinpoint areas where a tagger doesn't perform well.

Bibliographical and Historical Notes

The earliest implemented part-of-speech assignment algorithm may have been part of the parser in Zellig Harris's Transformations and Discourse Analysis Project (TDAP), which was implemented between June 1958 and July 1959 at the University of Pennsylvania (Harris, 1962). Previous natural language processing systems had used dictionaries with part-of-speech information for words, but have not been described as performing part-of-speech disambiguation. As part of its parsing, TDAP did part-of-speech disambiguation via 14 hand-written rules, whose use of part-of-speech tag sequences prefigures all the modern algorithms, and which were run in an order based on the relative frequency of tags for a word. The parser/tagger was reimplemented recently and is described by Joshi and Hopely (1999) and Karttunen (1999), who note that the parser was essentially implemented (in a very modern way) as a cascade of finite-state transducers.

Soon after the TDAP parser was the Computational Grammar Coder (CGC) of Klein and Simmons (1963). The CGC had three components: a lexicon, a morphological analyzer, and a context disambiguator. The small 1500-word lexicon included exceptional words that could not be accounted for in the simple morphological analyzer, including function words as well as irregular nouns, verbs, and adjectives. The morphological analyzer used inflectional and derivational suffixes to assign part-of-speech classes. A word was run through the lexicon and morphological analyzer to produce a candidate set of parts-of-speech. A set of 500 context rules were then used to disambiguate this candidate set, by relying on surrounding islands of unambiguous words. For example, one rule said that between an ARTICLE and a VERB, the only allowable sequences were ADJ-NOUN, NOUN-ADVERB, or NOUN-NOUN. The CGC algorithm reported 90% accuracy on applying a 30-tag tagset to articles from the Scientific American and a children's encyclopedia.

The TAGGIT tagger (Greene and Rubin, 1971) was based on the Klein and Simmons (1963) system, using the same architecture but increasing the size of the dictionary and the size of the tagset (to 87 tags). For example the following sample rule, which states that a word x is unlikely to be a plural noun (NNS) before a third person singular verb (VBZ):

$$x \text{ VBZ} \rightarrow \textit{not} \text{ NNS}$$

TAGGIT was applied to the Brown corpus and, according to Francis and Kučera (1982, p. 9), "resulted in the accurate tagging of 77% of the corpus" (the remainder of the Brown corpus was tagged by hand).

In the 1970s the Lancaster-Oslo/Bergen (LOB) corpus was compiled as a British English equivalent of the Brown corpus. It was tagged with the CLAWS tagger (Marshall, 1983, 1987; Garside, 1987), a probabilistic algorithm which can be viewed as an approximation to the HMM tagging approach. The algorithm used tag bigram probabilities, but instead of storing the word-likelihood of each tag, tags were marked either as *rare* ($P(\text{tag}|\text{word}) < .01$) *infrequent* ($P(\text{tag}|\text{word}) < .10$), or *normally frequent* ($P(\text{tag}|\text{word}) > .10$), The probabilistic PARTS tagger of Church (1988) was very close to a full HMM tagger. It extended the CLAWS idea to assign full lexical probabilities to each word/tag combination, and used Viterbi decoding to find a tag sequence. Like the CLAWS tagger, however, it stored the probability of the tag given the word:

$$P(\text{tag}|\text{word}) * P(\text{tag}|\text{previous } n \text{ tags}) \quad (5.57)$$

rather than using the probability of the word given the tag, as an HMM tagger does:

$$P(\text{word}|\text{tag}) * P(\text{tag}|\text{previous } n \text{ tags}) \quad (5.58)$$

Later taggers explicitly introduced the use of the Hidden Markov Model, often with the EM training algorithm (Kupiec, 1992; Merialdo, 1994; Weischedel et al., 1993), including the use of variable-length Markov models (Schütze and Singer, 1994).

Most recent tagging algorithms, like the HMM and TBL approaches we have discussed, are machine-learning classifiers which estimate the best tag-sequence for a sentence given various features such as the current word, neighboring parts-of-speech or words, and unknown word features such as orthographic and morphological features. Many kinds of classifiers have been used to combine these features, including decision trees (Jelinek et al., 1994; Magerman, 1995a), maximum entropy models (Ratnaparkhi, 1996), other log-linear models (Franz, 1996), memory-based learning (Daelemans et al., 1996), and networks of linear separators (SNOW) (Roth and Zelenko, 1998). Most machine learning models seem to achieve relatively similar performance given similar features, roughly 96-97% on the Treebank 45-tag tagset on the Wall Street Journal corpus. As of the writing of this chapter, the highest performing published model on this WSJ Treebank task is a log-linear tagger that uses information about neighboring words as well as tags, and a sophisticated unknown-word model, achieving 97.24% accuracy (Toutanova et al., 2003). Most such models are supervised, although there is beginning to be work on unsupervised models (Schütze, 1995; Brill, 1997; Clark, 2000; Banko and Moore, 2004; Goldwater and Griffiths, 2007).

Readers interested in the history of parts-of-speech should consult a history of linguistics such as Robins (1967) or Koerner and Asher (1995), particularly the article by Householder (1995) in the latter. Sampson (1987) and Garside et al. (1997) give a detailed summary of the provenance and makeup of the Brown and other tagsets. More information on part-of-speech tagging can be found in van Halteren (1999).

Algorithms for spelling error detection and correction have existed since at least Blair (1960). Most early algorithm were based on similarity keys like the Soundex algorithm discussed in the exercises on page 82 (Odell and Russell, 1922; Knuth, 1973). Damerau (1964) gave a dictionary-based algorithm for error detection; most error-detection algorithms since then have been based on dictionaries. Damerau also gave a correction algorithm that worked for single errors. Most algorithms since then have relied on dynamic programming, beginning with Wagner and Fischer (1974). Kukich (1992) is the definitive survey article on spelling error detection and correction. Modern algorithms are based on statistical or machine learning algorithm, following e.g., Kashyap and Oommen (1983) and Kernighan et al. (1990). Recent approaches to spelling include extensions to the noisy channel model (Brill and Moore, 2000; Toutanova and Moore, 2002) as well as many other machine learning architectures such as Bayesian classifiers, (Gale et al., 1993; Golding, 1997; Golding and Schabes, 1996), decision lists (Yarowsky, 1994), transformation based learning (Mangu and Brill, 1997) latent semantic analysis (Jones and Martin, 1997) and Winnow (Golding and Roth, 1999). Hirst and Budanitsky (2005) explore the use of word relatedness; see Ch. 20. Noisy channel spelling correction is used in a number of commercial applications, including the Microsoft Word contextual spell checker.

Exercises

5.1 Find one tagging error in each of the following sentences that are tagged with the Penn Treebank tagset:

a. I/PRP need/VBP a/DT flight/NN from/IN Atlanta/NN
b. Does/VBZ this/DT flight/NN serve/VB dinner/NNS
c. I/PRP have/VB a/DT friend/NN living/VBG in/IN Denver/NNP
d. Can/VBP you/PRP list/VB the/DT nonstop/JJ afternoon/NN flights/NNS

5.2 Use the Penn Treebank tagset to tag each word in the following sentences from Damon Runyon's short stories. You may ignore punctuation. Some of these are quite difficult; do your best.

a. It is a nice night.
b. This crap game is over a garage in Fifty-second Street...
c. ...Nobody ever takes the newspapers she sells ...
d. He is a tall, skinny guy with a long, sad, mean-looking kisser, and a mournful voice.
e. ...I am sitting in Mindy's restaurant putting on the gefillte fish, which is a dish I am very fond of, ...
f. When a guy and a doll get to taking peeks back and forth at each other, why there you are indeed.

5.3 Now compare your tags from the previous exercise with one or two friend's answers. On which words did you disagree the most? Why?

5.4 Now tag the sentences in Exercise 2 using the more detailed Brown tagset in Fig. 5.2.

5.5 Implement the TBL algorithm in Fig. 5.21. Create a small number of templates and train the tagger on any POS-tagged training set you can find.

5.6 Implement the "most-likely tag" baseline. Find a POS-tagged training set, and use it to compute for each word the tag which maximizes $p(t|w)$. You will need to implement a simple tokenizer to deal with sentence boundaries. Start by assuming all unknown words are NN and compute your error rate on known and unknown words. Now write at least 5 rules to do a better job of tagging unknown words, and show the difference in error rates.

5.7 Recall that the Church (1988) tagger is not an HMM tagger since it incorporates the probability of the tag given the word:

$$P(\text{tag}|\text{word}) * P(\text{tag}|\text{previous } n \text{ tags}) \tag{5.59}$$

rather than using the likelihood of the word given the tag, as an HMM tagger does:

$$P(\text{word}|\text{tag}) * P(\text{tag}|\text{previous } n \text{ tags}) \tag{5.60}$$

As a gedanken-experiment, construct a sentence, a set of tag transition probabilities, and a set of lexical tag probabilities that demonstrate a way in which the HMM tagger can produce a better answer than the Church tagger, and another example in which the Church tagger is better.

5.8 Build an HMM tagger. This requires (1) that you have implemented the Viterbi algorithm from this chapter and Ch. 6, (2) that you have a dictionary with part-of-speech information and (3) that you have either (a) a part-of-speech-tagged corpus or (b) an implementation of the Forward Backward algorithm. If you have a labeled corpus, train the transition and observation probabilities of an HMM tagger directly on the hand-tagged data. If you have an unlabeled corpus, train using Forward Backward.

5.9 Now run your algorithm on a small test set that you have hand-labeled. Find five errors and analyze them.

5.10 Compute a bigram grammar on a large corpus and reestimate the spelling correction probabilities shown in Fig. 5.25 given the correct sequence ... *was called a "stellar and versatile* **acress** *whose combination of sass and glamour has defined her...* ". Does a bigram grammar prefer the correct word *actress*?

5.11 Read Norvig (2007) and implement one of the extensions he suggests to his Python noisy channel spell checker.