CHAPTER

8 Part-of-Speech Tagging

Dionysius Thrax of Alexandria (*c.* 100 B.C.), or perhaps someone else (it was a long time ago), wrote a grammatical sketch of Greek (a "*technē*") that summarized the linguistic knowledge of his day. This work is the source of an astonishing proportion of modern linguistic vocabulary, including words like *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 that became the basis for practically all subsequent part-of-speech descriptions of most European languages for the next 2000 years.

parts of speech

Schoolhouse Rock was a series of popular animated educational television clips from the 1970s. Its Grammar Rock sequence included songs about exactly 8 parts of speech, including the late great Bob Dorough's *Conjunction Junction*:

> *Conjunction Junction, what's your function?*
> *Hooking up words and phrases and clauses...*

Although the list of 8 was slightly modified from Thrax's original, the astonishing durability of the parts of speech through two millennia is an indicator of both the importance and the transparency of their role in human language.[1]

POS

Parts of speech (also known as **POS**, **word classes**, or **syntactic categories**) are useful because they reveal a lot about a word and its neighbors. Knowing whether a word is a **noun** or a **verb** tells us about likely neighboring words (nouns are preceded by determiners and adjectives, verbs by nouns) and syntactic structure (nouns are generally part of noun phrases), making part-of-speech tagging a key aspect of parsing (Chapter 13). Parts of speech are useful features for labeling **named entities** like people or organizations in **information extraction** (Chapter 18), or for coreference resolution (Chapter 22). A word's part of speech can even play a role in speech recognition or synthesis, e.g., the word *content* is pronounced *CONtent* when it is a noun and *conTENT* when it is an adjective.

This chapter introduces parts of speech, and then introduces two algorithms for **part-of-speech tagging**, the task of assigning parts of speech to words. One is generative— Hidden Markov Model (HMM)—and one is discriminative—the Maximum Entropy Markov Model (MEMM). Chapter 9 then introduces a third algorithm based on the recurrent neural network (RNN). All three have roughly equal performance but, as we'll see, have different tradeoffs.

8.1 (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. While word classes do have semantic tendencies—adjectives, for example, often describe

[1] Nonetheless, eight isn't very many and, as we'll see, recent tagsets have more.

properties and nouns *people*— parts of speech are traditionally defined instead based on syntactic and morphological function, grouping words that have similar neighboring words (their **distributional** properties) or take similar affixes (their morphological properties).

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 with relatively fixed membership, such as prepositions—new prepositions are rarely coined. By contrast, nouns and verbs are open classes—new nouns and verbs like *iPhone* or *to fax* are continually being created or borrowed. Any given speaker or corpus may have different open class words, but all speakers of a language, and sufficiently large corpora, likely share the set of closed class words. Closed class words are 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 word

Four major open classes occur in the languages of the world: **nouns**, **verbs**, **adjectives**, and **adverbs**. English has all four, although not every language does.

noun

The syntactic class **noun** includes the words for most people, places, or things, but others as well. 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 noun

Open class nouns fall into two classes. **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.

common noun
count noun
mass noun

The other class, **common nouns**, are divided in many languages, including English, into **count nouns** and **mass nouns**. Count nouns allow grammatical enumeration, occurring 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

Verbs refer to actions and processes, including main verbs like *draw*, *provide*, and *go*. English verbs have inflections (non-third-person-sg (*eat*), third-person-sg (*eats*), progressive (*eating*), past participle (*eaten*)). 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) .

adjective

The third open class English form is **adjectives**, a class that includes many terms for 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".

adverb

The final open class form, **adverbs**, is rather a hodge-podge in both form and meaning. In the following all the italicized words are adverbs:

Actually, I ran *home extremely quickly yesterday*

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 "ad-

verb", 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 adverbs** describe the time that some action or event took place (*yesterday*, *Monday*). Because of the heterogeneous nature of this class, some adverbs (e.g., temporal adverbs like *Monday*) are tagged in some tagging schemes as nouns.

locative
degree
manner
temporal

The closed classes differ more from language to language than do the open classes. Some of the important closed classes in English include:

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

preposition

Prepositions occur before noun phrases. Semantically they often indicate spatial or temporal relations, whether literal (*on it*, *before then*, *by the house*) or metaphorical (*on time*, *with gusto*, *beside herself*), but often indicate other relations as well, like marking the agent in *Hamlet was written by Shakespeare*. A **particle** resembles a preposition or an adverb and is used in combination with a verb. Particles often have extended meanings that aren't quite the same as the prepositions they resemble, as in the particle *over* in *she turned the paper over*.

particle

A verb and a particle that act as a single syntactic and/or semantic unit are called a **phrasal verb**. The meaning of phrasal verbs is often problematically **non-compositional**—not predictable from the distinct meanings of the verb and the particle. Thus, *turn down* means something like 'reject', *rule out* 'eliminate', *find out* 'discover', and *go on* 'continue'.

phrasal verb

A closed class that occurs with nouns, often marking the beginning of a noun phrase, is the **determiner**. One small subtype of determiners is the **article**: English has three articles: *a*, *an*, and *the*. Other determiners include *this* and *that* (*this chapter*, *that page*). *A* and *an* mark a noun phrase as indefinite, while *the* can mark it as definite; definiteness is a discourse property (Chapter 23). Articles are quite frequent in English; indeed, *the* is the most frequently occurring word in most corpora of written English, and *a* and *an* are generally right behind.

determiner
article

Conjunctions 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 has some 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**.

conjunctions
complementizer

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

pronoun
personal
possessive
wh

complementizers (*Frida, who married Diego...*).

auxiliary

A closed class subtype of English verbs are the **auxiliary** verbs. Cross-linguistically, auxiliaries mark semantic features of a main verb: 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, or desired (mood). 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* can mark the perfect tenses (*I have gone*, *I had gone*), and *be* is used as part of the passive (*We were robbed*) or progressive (*We are leaving*) constructions. Modals are used to mark the mood associated with the event depicted by the main verb: *can* indicates ability or possibility, *may* permission or possibility, *must* necessity. There is also a modal use of *have* (e.g., *I have to go*).

copula

modal

English also has many words of more or less unique function, including **interjections** (*oh, hey, alas, uh, um*), **negatives** (*no, not*), **politeness markers** (*please, thank you*), **greetings** (*hello, goodbye*), and the existential **there** (*there are two on the table*) among others. These classes may be distinguished or lumped together as interjections or adverbs depending on the purpose of the labeling.

interjection

negative

8.2 The Penn Treebank Part-of-Speech Tagset

An important tagset for English is the 45-tag Penn Treebank tagset (Marcus et al., 1993), shown in Fig. 8.1, which has been used to label many corpora. In such labelings, parts of speech are generally represented by placing the tag after each word, delimited by a slash:

Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coordinating conjunction	*and, but, or*	PDT	predeterminer	*all, both*	VBP	verb non-3sg present	*eat*
CD	cardinal number	*one, two*	POS	possessive ending	*'s*	VBZ	verb 3sg pres	*eats*
DT	determiner	*a, the*	PRP	personal pronoun	*I, you, he*	WDT	wh-determ.	*which, that*
EX	existential 'there'	*there*	PRP$	possess. pronoun	*your, one's*	WP	wh-pronoun	*what, who*
FW	foreign word	*mea culpa*	RB	adverb	*quickly*	WP$	wh-possess.	*whose*
IN	preposition/ subordin-conj	*of, in, by*	RBR	comparative adverb	*faster*	WRB	wh-adverb	*how, where*
JJ	adjective	*yellow*	RBS	superlatv. adverb	*fastest*	$	dollar sign	*$*
JJR	comparative adj	*bigger*	RP	particle	*up, off*	#	pound sign	*#*
JJS	superlative adj	*wildest*	SYM	symbol	+,%, &	“	left quote	‘ or “
LS	list item marker	*1, 2, One*	TO	“to”	*to*	”	right quote	’ or ”
MD	modal	*can, should*	UH	interjection	*ah, oops*	(	left paren	[, (, {, <
NN	sing or mass noun	*llama*	VB	verb base form	*eat*	)	right paren	],), }, >
NNS	noun, plural	*llamas*	VBD	verb past tense	*ate*	,	comma	,
NNP	proper noun, sing.	*IBM*	VBG	verb gerund	*eating*	.	sent-end punc	. ! ?
NNPS	proper noun, plu.	*Carolinas*	VBN	verb past part.	*eaten*	:	sent-mid punc	: ; ... – -

Figure 8.1 Penn Treebank part-of-speech tags (including punctuation).

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

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

(8.3) Preliminary/JJ findings/NNS were/VBD **reported/VBN** in/IN today/NN **'s/POS** New/NNP England/NNP Journal/NNP of/IN Medicine/NNP ./.

Example (8.1) shows the determiners *the* and *a*, the adjectives *grand* and *other*, the common nouns *jury*, *number*, and *topics*, and the past tense verb *commented*. Example (8.2) 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 (8.3) shows the segmentation of the possessive morpheme *'s*, and a passive construction, 'were reported', in which *reported* is tagged as a past participle (VBN). 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).

Corpora labeled with parts of speech are crucial training (and testing) sets for statistical tagging algorithms. Three main tagged corpora are consistently used for training and testing part-of-speech taggers for English. The **Brown** corpus is a million words of samples from 500 written texts from different genres published in the United States in 1961. The **WSJ** corpus contains a million words published in the Wall Street Journal in 1989. The **Switchboard** corpus consists of 2 million words of telephone conversations collected in 1990-1991. The corpora were created by running an automatic part-of-speech tagger on the texts and then human annotators hand-corrected each tag.

Brown
WSJ
Switchboard

There are some minor differences in the tagsets used by the corpora. For example in the WSJ and Brown corpora, the single Penn tag TO is used for both the infinitive *to* (*I like to race*) and the preposition *to* (*go to the store*), while in Switchboard the tag TO is reserved for the infinitive use of *to* and the preposition is tagged IN:

Well/UH ,/, I/PRP ,/, I/PRP want/VBP **to/TO** go/VB **to/IN** a/DT restaurant/NN

Finally, there are some idiosyncracies inherent in any tagset. For example, because the Penn 45 tags were collapsed from a larger 87-tag tagset, the **original Brown tagset**, some potentially useful distinctions were lost. The Penn tagset was designed for a treebank in which sentences were parsed, and so it leaves off syntactic information recoverable from the parse tree. Thus for example the Penn tag IN is used for both subordinating conjunctions like *if, when, unless, after*:

after/IN spending/VBG a/DT day/NN at/IN the/DT beach/NN

and prepositions like *in, on, after*:

after/IN sunrise/NN

Words are generally tokenized before tagging. The Penn Treebank and the British National Corpus split contractions and the *'s*-genitive from their stems:[2]

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

The Treebank tagset assumes that tokenization of multipart words like *New York* is done at whitespace, thus tagging. *a New York City firm* as *a/DT New/NNP York/NNP City/NNP firm/NN*.

Another commonly used tagset, the Universal POS tag set of the Universal Dependencies project (Nivre et al., 2016), is used when building systems that can tag many languages. See Section 8.7.

[2] Indeed, the Treebank tag POS is used only for *'s*, which must be segmented in tokenization.

8.3 Part-of-Speech Tagging

part-of-speech tagging

Part-of-speech tagging is the process of assigning a part-of-speech marker to each word in an input text.[3] The input to a tagging algorithm is a sequence of (tokenized) words and a tagset, and the output is a sequence of tags, one per token.

ambiguous

Tagging is a **disambiguation** task; words are **ambiguous** —have more than one possible part-of-speech—and the goal is to find the correct tag for the situation. For example, *book* can be a verb (*book that flight*) or a noun (*hand me that book*). *That* can be a determiner (*Does that flight serve dinner*) or a complementizer (*I thought that your flight was earlier*). The goal of POS-tagging is to **resolve** these ambiguities, choosing the proper tag for the context. How common is tag ambiguity? Fig. 8.2 shows that most word types (85-86%) are unambiguous (*Janet* is always NNP, *funniest* JJS, and *hesitantly* RB). But the ambiguous words, though accounting for only 14-15% of the vocabulary, are very common words, and hence 55-67% of word tokens in running text are ambiguous.[4]

ambiguity resolution

		WSJ	Brown
Types:			
Unambiguous	(1 tag)	44,432 (**86%**)	45,799 (**85%**)
Ambiguous	(2+ tags)	7,025 (**14%**)	8,050 (**15%**)
Tokens:			
Unambiguous	(1 tag)	577,421 (**45%**)	384,349 (**33%**)
Ambiguous	(2+ tags)	711,780 (**55%**)	786,646 (**67%**)

Figure 8.2 Tag ambiguity for word types in Brown and WSJ, using Treebank-3 (45-tag) tagging. Punctuation were treated as words, and words were kept in their original case.

Some of the most ambiguous frequent words are *that*, *back*, *down*, *put* and *set*; here are some examples of the 6 different parts of speech for the word *back*:

earnings growth took a **back/JJ** seat
a small building in the **back/NN**
a clear majority of senators **back/VBP** the bill
Dave began to **back/VB** toward the door
enable the country to buy **back/RP** about debt
I was twenty-one **back/RB** then

Nonetheless, many words are easy to disambiguate, because their different tags aren't equally likely. For example, *a* can be a determiner or the letter *a*, but the determiner sense is much more likely. This idea suggests a simplistic **baseline** algorithm for part-of-speech tagging: given an ambiguous word, choose the tag which is **most frequent** in the training corpus. This is a key concept:

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).

accuracy

How good is this baseline? A standard way to measure the performance of part-of-speech taggers is **accuracy**: the percentage of tags correctly labeled (matching

[3] Tags are also applied to punctuation, so tagging assumes tokenizing of commas, quotation marks, etc., and disambiguating end-of-sentence periods from periods inside words (*e.g.*, *etc.*).

[4] Note the large differences across the two genres, especially in token frequency. Tags in the WSJ corpus are less ambiguous; its focus on financial news leads to a more limited distribution of word usages than the diverse genres of the Brown corpus.

human labels on a test set). If we train on the WSJ training corpus and test on sections 22-24 of the same corpus the most-frequent-tag baseline achieves an accuracy of 92.34%. By contrast, the state of the art in part-of-speech tagging on this dataset is around 97% tag accuracy, a performance that is achievable by most algorithms (HMMs, MEMMs, neural networks, rule-based algorithms). See Section 8.7 on other languages and genres.

8.4 HMM Part-of-Speech Tagging

sequence model

In this section we introduce the use of the Hidden Markov Model for part-of-speech tagging. The HMM is a **sequence model**. A sequence model or **sequence classifier** is a model whose job is to assign a label or class to each unit in a sequence, thus mapping a sequence of observations to a sequence of labels. An HMM is a probabilistic sequence model: given a sequence of units (words, letters, morphemes, sentences, whatever), it computes a probability distribution over possible sequences of labels and chooses the best label sequence.

8.4.1 Markov Chains

Markov chain

The HMM is based on augmenting the Markov chain. A **Markov chain** is a model that tells us something about the probabilities of sequences of random variables, *states*, each of which can take on values from some set. These sets can be words, or tags, or symbols representing anything, for example the weather. A Markov chain makes a very strong assumption that if we want to predict the future in the sequence, all that matters is the current state. All the states before the current state have no impact on the future except via the current state. It's as if to predict tomorrow's weather you could examine today's weather but you weren't allowed to look at yesterday's weather.

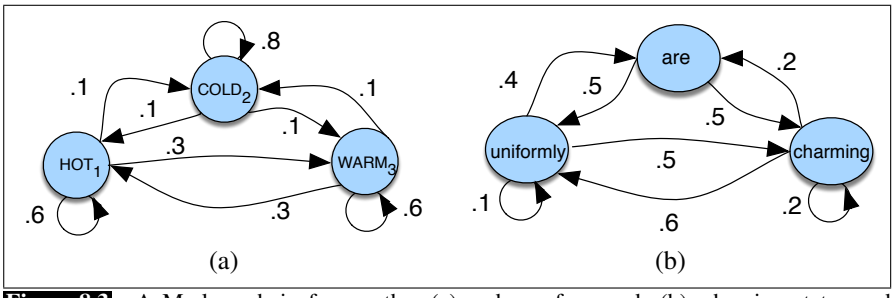

Figure 8.3 A Markov chain for weather (a) and one for words (b), showing states and transitions. A start distribution π is required; setting $\pi = [0.1, 0.7, 0.2]$ for (a) would mean a probability 0.7 of starting in state 2 (cold), probability 0.1 of starting in state 1 (hot), etc.

Markov assumption

More formally, consider a sequence of state variables $q_1, q_2, ..., q_i$. A Markov model embodies the **Markov assumption** on the probabilities of this sequence: that when predicting the future, the past doesn't matter, only the present.

$$\textbf{Markov Assumption:} \quad P(q_i = a|q_1...q_{i-1}) = P(q_i = a|q_{i-1}) \tag{8.4}$$

Figure 8.3a shows a Markov chain for assigning a probability to a sequence of weather events, for which the vocabulary consists of HOT, COLD, and WARM. The

states are represented as nodes in the graph, and the transitions, with their probabilities, as edges. The transitions are probabilities: the values of arcs leaving a given state must sum to 1. Figure 8.3b shows a Markov chain for assigning a probability to a sequence of words $w_1 ... w_n$. This Markov chain should be familiar; in fact, it represents a bigram language model, with each edge expressing the probability $p(w_i|w_j)$! Given the two models in Fig. 8.3, we can assign a probability to any sequence from our vocabulary.

Formally, a Markov chain is specified by the following components:

$Q = q_1 q_2 \ldots q_N$	a set of N **states**
$A = a_{11} a_{12} \ldots a_{n1} \ldots a_{nn}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{n} a_{ij} = 1 \quad \forall i$
$\pi = \pi_1, \pi_2, ..., \pi_N$	an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$

Before you go on, use the sample probabilities in Fig. 8.3a (with $\pi = [0.1, 0.7, 0.2]$) to compute the probability of each of the following sequences:

(8.5) hot hot hot hot
(8.6) cold hot cold hot

What does the difference in these probabilities tell you about a real-world weather fact encoded in Fig. 8.3a?

8.4.2 The Hidden Markov Model

A Markov chain is useful when we need to compute a probability for a sequence of observable events. In many cases, however, the events we are interested in are **hidden**: we don't observe them directly. For example we don't normally observe part-of-speech tags in a text. Rather, we see words, and must infer the tags from the word sequence. We call the tags **hidden** because they are not observed.

hidden

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. An HMM is specified by the following components:

$Q = q_1 q_2 \ldots q_N$	a set of N **states**
$A = a_{11} \ldots a_{ij} \ldots a_{NN}$	a **transition probability matrix** A, each a_{ij} representing the probability of moving from state i to state j, s.t. $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$
$O = o_1 o_2 \ldots o_T$	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 q_i
$\pi = \pi_1, \pi_2, ..., \pi_N$	an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have $\pi_j = 0$, meaning that they cannot be initial states. Also, $\sum_{i=1}^{n} \pi_i = 1$

A first-order hidden Markov model instantiates two simplifying assumptions. First, as with a first-order Markov chain, the probability of a particular state depends only on the previous state:

$$\textbf{Markov Assumption:} \quad P(q_i|q_1...q_{i-1}) = P(q_i|q_{i-1}) \tag{8.7}$$

Second, the probability of an output observation o_i depends only on the state that produced the observation q_i and not on any other states or any other observations:

$$\textbf{Output Independence:} \quad P(o_i|q_1 \ldots q_i, \ldots, q_T, o_1, \ldots, o_i, \ldots, o_T) = P(o_i|q_i) \tag{8.8}$$

8.4.3 The components of an HMM tagger

Let's start by looking at the pieces of an HMM tagger, and then we'll see how to use it to tag. An HMM has two components, the A and B probabilities.

The A matrix contains the tag transition probabilities $P(t_i|t_{i-1})$ which represent the probability of a tag occurring given the previous tag. For example, modal verbs like *will* are very likely to be followed by a verb in the base form, a VB, like *race*, so we expect this probability to be high. We compute the maximum likelihood estimate of this transition probability by counting, out of the times we see the first tag in a labeled corpus, how often the first tag is followed by the second:

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})} \tag{8.9}$$

In the WSJ corpus, for example, MD occurs 13124 times of which it is followed by VB 10471, for an MLE estimate of

$$P(VB|MD) = \frac{C(MD, VB)}{C(MD)} = \frac{10471}{13124} = .80 \tag{8.10}$$

Let's walk through an example, seeing how these probabilities are estimated and used in a sample tagging task, before we return to the algorithm for decoding.

In HMM tagging, the probabilities are estimated by counting on a tagged training corpus. For this example we'll use the tagged WSJ corpus.

The B emission probabilities, $P(w_i|t_i)$, represent the probability, given a tag (say MD), that it will be associated with a given word (say *will*). The MLE of the emission probability is

$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)} \tag{8.11}$$

Of the 13124 occurrences of MD in the WSJ corpus, it is associated with *will* 4046 times:

$$P(will|MD) = \frac{C(MD, will)}{C(MD)} = \frac{4046}{13124} = .31 \tag{8.12}$$

We saw this kind of Bayesian modeling in Chapter 4; recall that this likelihood term is not asking "which is the most likely tag for the word *will*?" That would be the posterior $P(\text{MD}|\text{will})$. Instead, $P(\text{will}|\text{MD})$ answers the slightly counterintuitive question "If we were going to generate a MD, how likely is it that this modal would be *will*?"

The A transition probabilities, and B observation likelihoods of the HMM are illustrated in Fig. 8.4 for three states in an HMM part-of-speech tagger; the full tagger would have one state for each tag.

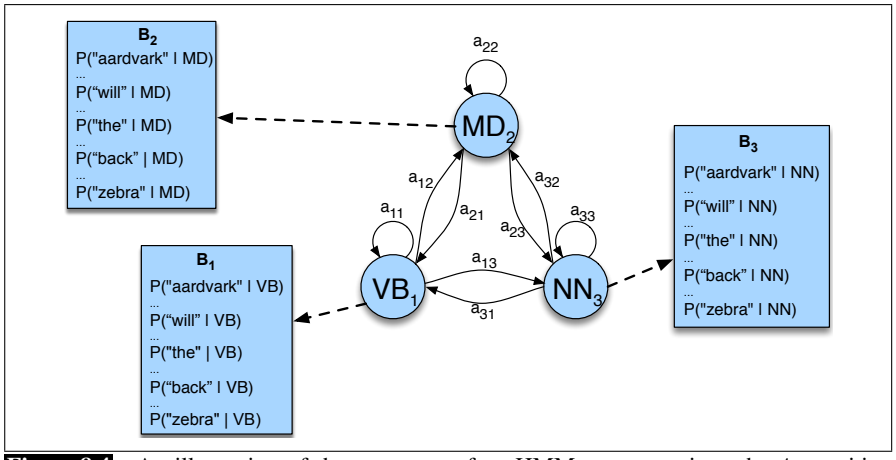

Figure 8.4 An illustration of the two parts of an HMM representation: the A transition probabilities used to compute the prior probability, and the B observation likelihoods that are associated with each state, one likelihood for each possible observation word.

8.4.4 HMM tagging as decoding

For any model, such as an HMM, that contains hidden variables, the task of determining the hidden variables sequence corresponding to the sequence of observations is called **decoding**. More formally,

decoding

> **Decoding**: Given as input an HMM $\lambda = (A,B)$ and a sequence of observations $O = o_1, o_2, ..., o_T$, find the most probable sequence of states $Q = q_1 q_2 q_3 \ldots q_T$.

For part-of-speech tagging, the goal of HMM decoding is to choose the tag sequence t_1^n that is most probable given the observation sequence of n words w_1^n:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n|w_1^n) \tag{8.13}$$

The way we'll do this in the HMM is to use Bayes' rule to instead compute:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} \frac{P(w_1^n|t_1^n)P(t_1^n)}{P(w_1^n)} \tag{8.14}$$

Furthermore, we simplify Eq. 8.14 by dropping the denominator $P(w_1^n)$:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(w_1^n|t_1^n)P(t_1^n) \tag{8.15}$$

HMM taggers make two further simplifying assumptions. The first is that the probability of a word appearing depends only on its own tag and is independent of neighboring words and tags:

$$P(w_1^n|t_1^n) \approx \prod_{i=1}^{n} P(w_i|t_i) \tag{8.16}$$

The second assumption, the **bigram** assumption, is that the probability of a tag is dependent only on the previous tag, rather than the entire tag sequence;

$$P(t_1^n) \approx \prod_{i=1}^{n} P(t_i|t_{i-1}) \tag{8.17}$$

Plugging the simplifying assumptions from Eq. 8.16 and Eq. 8.17 into Eq. 8.15 results in the following equation for the most probable tag sequence from a bigram tagger:

$$\hat{t}_1^n = \operatorname*{argmax}_{t_1^n} P(t_1^n|w_1^n) \approx \operatorname*{argmax}_{t_1^n} \prod_{i=1}^{n} \overbrace{P(w_i|t_i)}^{\text{emission}} \overbrace{P(t_i|t_{i-1})}^{\text{transition}} \tag{8.18}$$

The two parts of Eq. 8.18 correspond neatly to the B **emission probability** and A **transition probability** that we just defined above!

8.4.5 The Viterbi Algorithm

Viterbi algorithm

The decoding algorithm for HMMs is the **Viterbi algorithm** shown in Fig. 8.5. As an instance of **dynamic programming**, Viterbi resembles the dynamic programming **minimum edit distance** algorithm of Chapter 2.

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

create a path probability matrix *viterbi[N,T]*
for each state s **from** 1 **to** N **do** ; initialization step
  $viterbi[s,1] \leftarrow \pi_s * b_s(o_1)$
  $backpointer[s,1] \leftarrow 0$
for each time step t **from** 2 **to** T **do** ; recursion step
  for each state s **from** 1 **to** N **do**
    $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
    $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
$bestpathprob \leftarrow \max_{s=1}^{N} viterbi[s,T]$; termination step
$bestpathpointer \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T]$; termination step
bestpath $\leftarrow$ the path starting at state *bestpathpointer*, that follows backpointer[] to states back in time
return *bestpath*, *bestpathprob*

Figure 8.5 Viterbi algorithm for finding the optimal sequence of tags. Given an observation sequence and an HMM $\lambda = (A,B)$, the algorithm returns the state path through the HMM that assigns maximum likelihood to the observation sequence.

The Viterbi algorithm first sets up a probability matrix or **lattice**, with one column for each observation o_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. Figure 8.6 shows an intuition of this lattice for the sentence *Janet will back the bill*.

Each cell of the lattice, $v_t(j)$, represents the probability that the HMM is in state j after seeing the first t observations and passing through the most probable state sequence $q_1,...,q_{t-1}$, given the HMM λ. The value of each cell $v_t(j)$ is computed by recursively taking the most probable path that could lead us to this cell. Formally, each cell expresses the probability

$$v_t(j) = \max_{q_1,...,q_{t-1}} P(q_1...q_{t-1}, o_1, o_2 \ldots o_t, q_t = j|\lambda) \tag{8.19}$$

We represent the most probable path by taking the maximum over all possible previous state sequences $\max_{q_1,...,q_{t-1}}$. Like other dynamic programming algorithms,

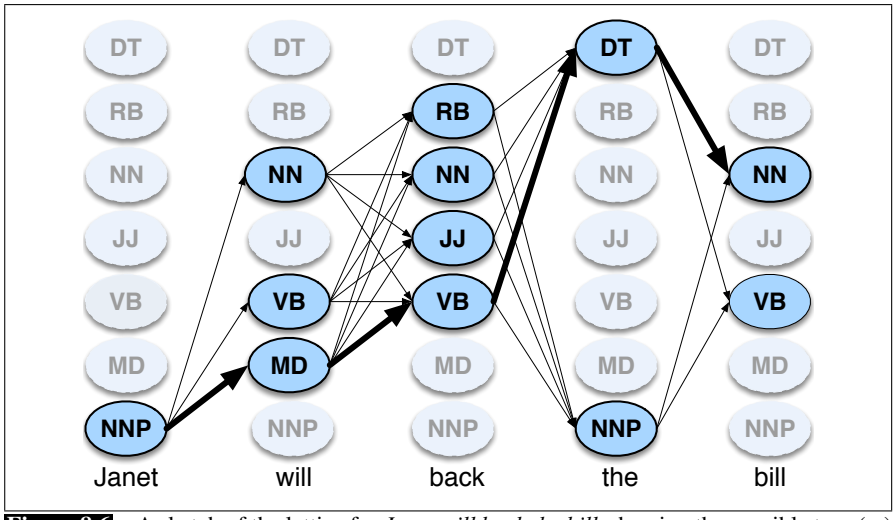

Figure 8.6 A sketch of the lattice for *Janet will back the bill*, showing the possible tags (q_i) for each word and highlighting the path corresponding to the correct tag sequence through the hidden states. States (parts of speech) which have a zero probability of generating a particular word according to the B matrix (such as the probability that a determiner DT will be realized as *Janet*) are greyed out.

Viterbi fills each cell recursively. Given that we had already computed the probability of being in every state at time $t-1$, we compute the Viterbi probability by taking the most probable of the extensions of the paths that lead to the current cell. For a given state q_j at time t, the value $v_t(j)$ is computed as

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t) \tag{8.20}$$

The three factors that are multiplied in Eq. 8.20 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

8.4.6 Working through an example

Let's tag the sentence *Janet will back the bill*; the goal is the correct series of tags (see also Fig. 8.6):

(8.21) Janet/NNP will/MD back/VB the/DT bill/NN

Let the HMM be defined by the two tables in Fig. 8.7 and Fig. 8.8. Figure 8.7 lists the a_{ij} probabilities for transitioning between the hidden states (part-of-speech tags). Figure 8.8 expresses the $b_i(o_t)$ probabilities, the *observation* likelihoods of words given tags. This table is (slightly simplified) from counts in the WSJ corpus. So the word *Janet* only appears as an NNP, *back* has 4 possible parts of speech, and the word *the* can appear as a determiner or as an NNP (in titles like "Somewhere Over the Rainbow" all words are tagged as NNP).

Figure 8.9 shows a fleshed-out version of the sketch we saw in Fig. 8.6, the Viterbi lattice for computing the best hidden state sequence for the observation sequence *Janet will back the bill*.

	NNP	MD	VB	JJ	NN	RB	DT
$<s>$	0.2767	0.0006	0.0031	0.0453	0.0449	0.0510	0.2026
NNP	0.3777	0.0110	0.0009	0.0084	0.0584	0.0090	0.0025
MD	0.0008	0.0002	0.7968	0.0005	0.0008	0.1698	0.0041
VB	0.0322	0.0005	0.0050	0.0837	0.0615	0.0514	0.2231
JJ	0.0366	0.0004	0.0001	0.0733	0.4509	0.0036	0.0036
NN	0.0096	0.0176	0.0014	0.0086	0.1216	0.0177	0.0068
RB	0.0068	0.0102	0.1011	0.1012	0.0120	0.0728	0.0479
DT	0.1147	0.0021	0.0002	0.2157	0.4744	0.0102	0.0017

Figure 8.7 The A transition probabilities $P(t_i|t_{i-1})$ computed from the WSJ corpus without smoothing. Rows are labeled with the conditioning event; thus $P(VB|MD)$ is 0.7968.

	Janet	will	back	the	bill
NNP	0.000032	0	0	0.000048	0
MD	0	0.308431	0	0	0
VB	0	0.000028	0.000672	0	0.000028
JJ	0	0	0.000340	0	0
NN	0	0.000200	0.000223	0	0.002337
RB	0	0	0.010446	0	0
DT	0	0	0	0.506099	0

Figure 8.8 Observation likelihoods B computed from the WSJ corpus without smoothing, simplified slightly.

There are $N = 5$ state columns. We begin in column 1 (for the word *Janet*) by setting the Viterbi value in each cell to the product of the π transition probability (the start probability for that state i, which we get from the $<s>$ entry of Fig. 8.7), and the observation likelihood of the word *Janet* given the tag for that cell. Most of the cells in the column are zero since the word *Janet* cannot be any of those tags. The reader should find this in Fig. 8.9.

Next, each cell in the *will* column gets updated. For each state, we compute the value $viterbi[s,t]$ by taking the maximum over the extensions of all the paths from the previous column that lead to the current cell according to Eq. 8.20. We have shown the values for the MD, VB, and NN cells. Each cell gets the max of the 7 values from the previous column, multiplied by the appropriate transition probability; as it happens in this case, most of them are zero from the previous column. The remaining value is multiplied by the relevant observation probability, and the (trivial) max is taken. In this case the final value, 2.772e-8, comes from the NNP state at the previous column. The reader should fill in the rest of the lattice in Fig. 8.9 and backtrace to see whether or not the Viterbi algorithm returns the gold state sequence NNP MD VB DT NN.

8.4.7 Extending the HMM Algorithm to Trigrams

Practical HMM taggers have a number of extensions of this simple model. One important missing feature is a wider tag context. In the tagger described above the probability of a tag depends only on the previous tag:

$$P(t_1^n) \approx \prod_{i=1}^{n} P(t_i|t_{i-1}) \tag{8.22}$$

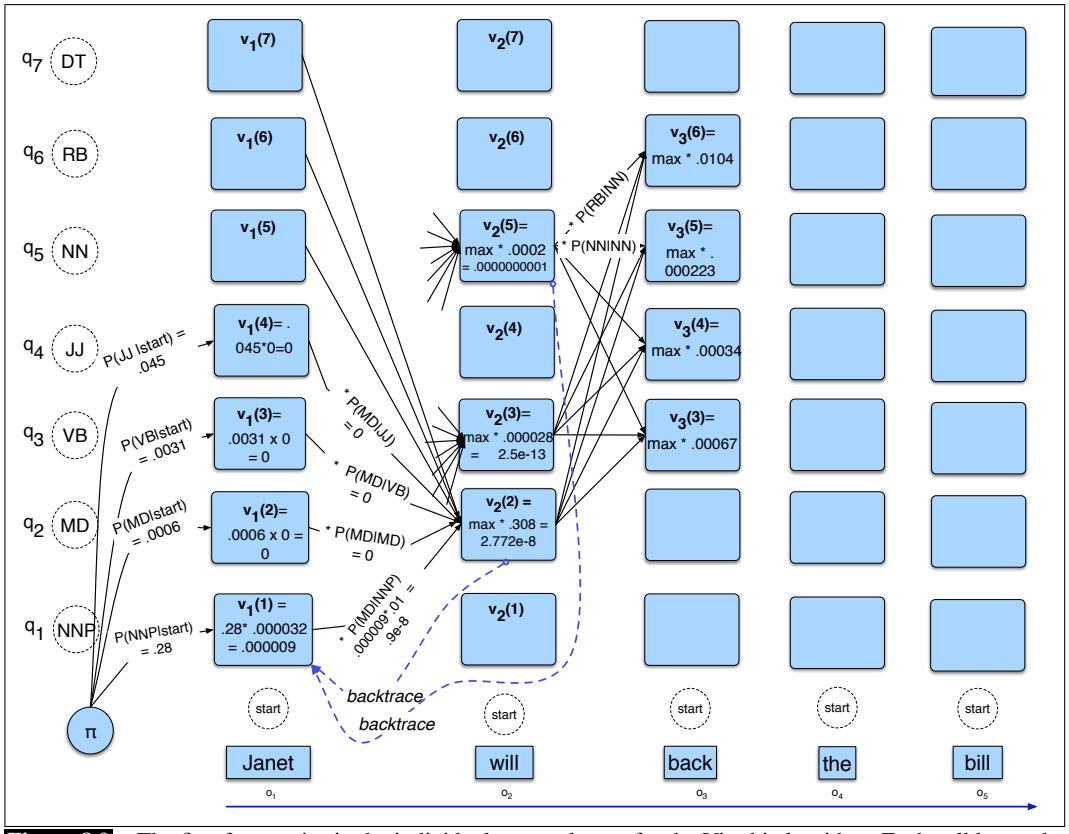

Figure 8.9 The first few entries in the individual state columns for the Viterbi algorithm. Each cell keeps the probability of the best path so far and a pointer to the previous cell along that path. We have only filled out columns 1 and 2; to avoid clutter most cells with value 0 are left empty. The 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 NNP MD VB DT NN.

In practice we use more of the history, letting the probability of a tag depend on the two previous tags:

$$P(t_1^n) \approx \prod_{i=1}^{n} P(t_i|t_{i-1},t_{i-2}) \tag{8.23}$$

Extending the algorithm from bigram to trigram taggers gives a small (perhaps a half point) increase in performance, but conditioning on two previous tags instead of one requires a significant change to the Viterbi algorithm. For each cell, instead of taking a max over transitions from each cell in the previous column, we have to take a max over paths through the cells in the previous two columns, thus considering N^2 rather than N hidden states at every observation.

In addition to increasing the context window, HMM taggers have a number of other advanced features. One is to 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:

$$\hat{t}_1^n = \underset{t_1^n}{\operatorname{argmax}} P(t_1^n|w_1^n) \approx \underset{t_1^n}{\operatorname{argmax}} \left[\prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1},t_{i-2})\right] P(t_{n+1}|t_n) \tag{8.24}$$

In tagging any sentence with Eq. 8.24, 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 that is added to the tagset, which assumes sentences boundaries have already been marked.

One problem with trigram taggers as instantiated in Eq. 8.24 is 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. 8.25:

$$P(t_i|t_{i-1},t_{i-2}) = \frac{C(t_{i-2},t_{i-1},t_i)}{C(t_{i-2},t_{i-1})} \tag{8.25}$$

Just as we saw with language modeling, 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 solving this problem is the same interpolation idea we saw in language modeling: estimate the probability by combining more robust, but weaker estimators. For example, if we've never seen the tag sequence PRP VB TO, and so 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 with the following counts:

$$\text{Trigrams} \quad \hat{P}(t_i|t_{i-1},t_{i-2}) = \frac{C(t_{i-2},t_{i-1},t_i)}{C(t_{i-2},t_{i-1})} \tag{8.26}$$

$$\text{Bigrams} \quad \hat{P}(t_i|t_{i-1}) = \frac{C(t_{i-1},t_i)}{C(t_{i-1})} \tag{8.27}$$

$$\text{Unigrams} \quad \hat{P}(t_i) = \frac{C(t_i)}{N} \tag{8.28}$$

The standard way to combine these three estimators to estimate the trigram probability $P(t_i|t_{i-1},t_{i-2})$ is via 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:

$$P(t_i|t_{i-1}t_{i-2}) = \lambda_3\hat{P}(t_i|t_{i-1}t_{i-2}) + \lambda_2\hat{P}(t_i|t_{i-1}) + \lambda_1\hat{P}(t_i) \tag{8.29}$$

deleted interpolation

We require $\lambda_1 + \lambda_2 + \lambda_3 = 1$, ensuring that the resulting P is a probability distribution. The λs are set by **deleted interpolation** (Jelinek and Mercer, 1980): 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 deletion helps to set the λs in such a way as to generalize to unseen data and not overfit. Figure 8.10 gives a deleted interpolation algorithm for tag trigrams.

8.4.8 Beam Search

When the number of states grows very large, the vanilla Viterbi algorithm is slow. The complexity of the algorithm is $O(N^2T)$; N (the number of states) can be large for trigram taggers, which have to consider every previous pair of the 45 tags, resulting in $45^3 = 91,125$ computations per column. N can be even larger for other applications of Viterbi, for example to decoding in neural networks, as we will see in future chapters.

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

$\lambda_1,\ \lambda_2,\ \lambda_3 \leftarrow 0$
foreach trigram t_1, t_2, t_3 with $C(t_1, t_2, t_3) > 0$
  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 λ_3 by $C(t_1, t_2, t_3)$
    case $\frac{C(t_2,t_3)-1}{C(t_2)-1}$: increment λ_2 by $C(t_1, t_2, t_3)$
    case $\frac{C(t_3)-1}{N-1}$: increment λ_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 8.10 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 number of tokens in the corpus. After Brants (2000).

beam search

One common solution to the complexity problem is the use of **beam search** decoding. In beam search, instead of keeping the entire column of states at each time point t, we just keep the best few hypothesis at that point. At time t this requires computing the Viterbi score for each of the N cells, sorting the scores, and keeping only the best-scoring states. The rest are pruned out and not continued forward to time $t+1$.

beam width

One way to implement beam search is to keep a fixed number of states instead of all N current states. Here the **beam width** β is a fixed number of states. Alternatively β can be modeled as a fixed percentage of the N states, or as a probability threshold. Figure 8.11 shows the search lattice using a beam width of 2 states.

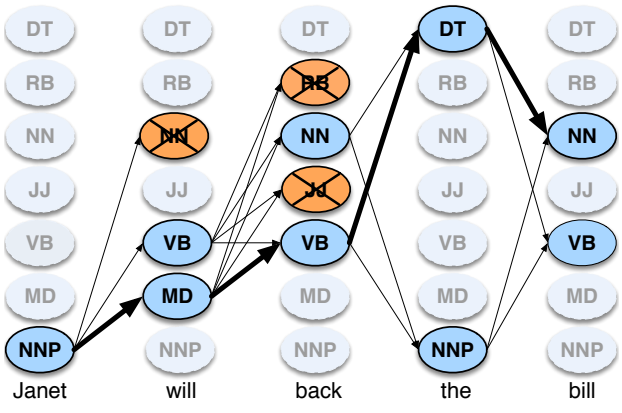

Figure 8.11 A beam search version of Fig. 8.6, showing a beam width of 2. At each time t, all (non-zero) states are computed, but then they are sorted and only the best 2 states are propagated forward and the rest are pruned, shown in orange.

8.4.9 Unknown Words

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

unknown words

To achieve high accuracy with part-of-speech taggers, it is also important to have a good model for dealing with **unknown words**. Proper names and acronyms are created very often, and even new common nouns and verbs enter the language at a surprising rate. One useful feature for distinguishing parts of speech is word shape: words starting with capital letters are likely to be proper nouns (NNP).

But the strongest source of information for guessing the part-of-speech of unknown words is morphology. 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* adjectives (JJ), and so on. We store for each final letter sequence (for simplicity referred to as word *suffixes*) of up to 10 letters the statistics of the tag it was associated with in training. We are thus computing for each suffix of length i the probability of the tag t_i given the suffix letters (Samuelsson 1993, Brants 2000):

$$P(t_i|l_{n-i+1}\dots l_n) \tag{8.30}$$

Back-off is used to smooth these probabilities with successively shorter suffixes. Because unknown words are unlikely to be closed-class words like prepositions, suffix probabilities can be computed only for words whose training set frequency is ≤ 10, or only for open-class words. Separate suffix tries are kept for capitalized and uncapitalized words.

Finally, because Eq. 8.30 gives a posterior estimate $p(t_i|w_i)$, we can compute the likelihood $p(w_i|t_i)$ that HMMs require by using Bayesian inversion (i.e., using Bayes' rule and computation of the two priors $P(t_i)$ and $P(t_i|l_{n-i+1}\dots l_n)$).

In addition to using capitalization information for unknown words, Brants (2000) also uses capitalization for 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 Eq. 8.26, the algorithm 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, doubling the size of the tagset.

Combining all these features, a trigram HMM like that of Brants (2000) has a tagging accuracy of 96.7% on the Penn Treebank, perhaps just slightly below the performance of the best MEMM and neural taggers.

8.5 Maximum Entropy Markov Models

While an HMM can achieve very high accuracy, we saw that it requires a number of architectural innovations to deal with unknown words, backoff, suffixes, and so on. It would be so much easier if we could add arbitrary features directly into the model in a clean way, but that's hard for generative models like HMMs. Luckily, we've already seen a model for doing this: the logistic regression model of Chapter 5! But logistic regression isn't a sequence model; it assigns a class to a single observation. However, we could turn logistic regression into a discriminative sequence model simply by running it on successive words, using the class assigned to the prior word

MEMM

as a feature in the classification of the next word. When we apply logistic regression in this way, it's called the **maximum entropy Markov model** or **MEMM**.[5]

Let the sequence of words be $W = w_1^n$ and the sequence of tags $T = t_1^n$. In an HMM to compute the best tag sequence that maximizes $P(T|W)$ we rely on Bayes' rule and the likelihood $P(W|T)$:

$$\begin{aligned}\hat{T} &= \operatorname*{argmax}_T P(T|W) \\ &= \operatorname*{argmax}_T P(W|T)P(T) \\ &= \operatorname*{argmax}_T \prod_i P(word_i|tag_i) \prod_i P(tag_i|tag_{i-1}) \end{aligned} \tag{8.31}$$

In an MEMM, by contrast, we compute the posterior $P(T|W)$ directly, training it to discriminate among the possible tag sequences:

$$\begin{aligned}\hat{T} &= \operatorname*{argmax}_T P(T|W) \\ &= \operatorname*{argmax}_T \prod_i P(t_i|w_i, t_{i-1}) \end{aligned} \tag{8.32}$$

Consider tagging just one word. A multinomial logistic regression classifier could compute the single probability $P(t_i|w_i, t_{i-1})$ in a different way than an HMM. Fig. 8.12 shows the intuition of the difference via the direction of the arrows; HMMs compute likelihood (observation word conditioned on tags) but MEMMs compute posterior (tags conditioned on observation words).

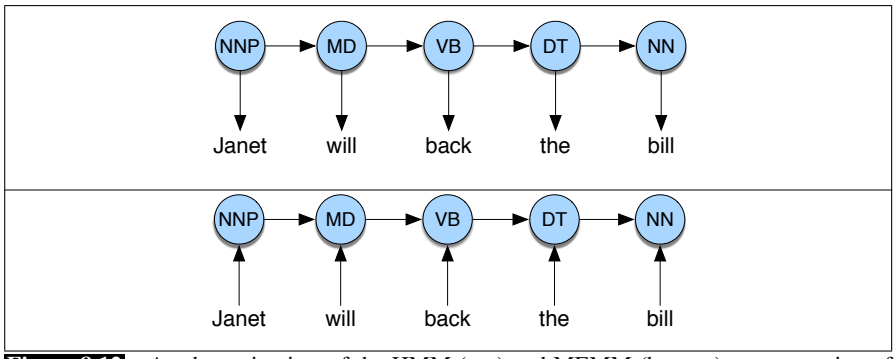

Figure 8.12 A schematic view of the HMM (top) and MEMM (bottom) representation of the probability computation for the correct sequence of tags for the *back* sentence. The HMM computes the likelihood of the observation given the hidden state, while the MEMM computes the posterior of each state, conditioned on the previous state and current observation.

8.5.1 Features in a MEMM

Of course we don't build MEMMs that condition just on w_i and t_{i-1}. The reason to use a discriminative sequence model is that it's easier to incorporate a lot of features.[6] Figure 8.13 shows a graphical intuition of some of these additional features.

[5] 'Maximum entropy model' is an outdated name for logistic regression; see the history section.

[6] Because in HMMs all computation is based on the two probabilities P(tag|tag) and P(word|tag), if we want to include some source of knowledge into the tagging process, we must find a way to encode the knowledge into one of these two probabilities. Each time we add a feature we have to do a lot of complicated conditioning which gets harder and harder as we have more and more such features.

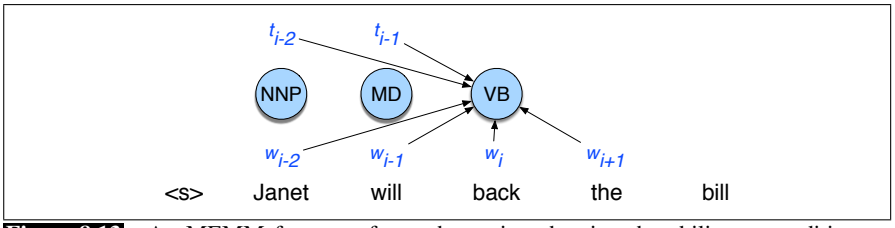

Figure 8.13 An MEMM for part-of-speech tagging showing the ability to condition on more features.

A basic MEMM part-of-speech tagger conditions on the observation word itself, neighboring words, and previous tags, and various combinations, using feature **templates** like the following:

templates

$$\begin{aligned} &\langle t_i, w_{i-2}\rangle, \langle t_i, w_{i-1}\rangle, \langle t_i, w_i\rangle, \langle t_i, w_{i+1}\rangle, \langle t_i, w_{i+2}\rangle \\ &\langle t_i, t_{i-1}\rangle, \langle t_i, t_{i-2}, t_{i-1}\rangle, \\ &\langle t_i, t_{i-1}, w_i\rangle, \langle t_i, w_{i-1}, w_i\rangle \langle t_i, w_i, w_{i+1}\rangle, \end{aligned} \tag{8.33}$$

Recall from Chapter 5 that feature templates are used to automatically populate the set of features from every instance in the training and test set. Thus our example *Janet/NNP will/MD back/VB the/DT bill/NN*, when w_i is the word *back*, would generate the following features:

t_i = VB and w_{i-2} = Janet
t_i = VB and w_{i-1} = will
t_i = VB and w_i = back
t_i = VB and w_{i+1} = the
t_i = VB and w_{i+2} = bill
t_i = VB and t_{i-1} = MD
t_i = VB and t_{i-1} = MD and t_{i-2} = NNP
t_i = VB and w_i = back and w_{i+1} = the

Also necessary are features to deal with unknown words, expressing properties of the word's spelling or shape:

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

word shape

Word shape features are used to represent the abstract letter pattern of the word by mapping lower-case letters to 'x', upper-case to 'X', numbers to 'd', and retaining punctuation. Thus for example I.M.F would map to X.X.X. and DC10-30 would map to XXdd-dd. A second class of shorter word shape features is also used. In these features consecutive character types are removed, so DC10-30 would be mapped to Xd-d but I.M.F would still map to X.X.X. For example the word *well-dressed* would generate the following non-zero valued feature values:

prefix(w_i) = `w`
prefix(w_i) = `we`
prefix(w_i) = `wel`
prefix(w_i) = `well`
suffix(w_i) = `ssed`
suffix(w_i) = `sed`
suffix(w_i) = `ed`
suffix(w_i) = `d`
has-hyphen(w_i)
word-shape(w_i) = `xxxx-xxxxxxx`
short-word-shape(w_i) = `x-x`

Features for known words, like the templates in Eq. 8.33, are computed for every word seen in the training set. The unknown word features can also be computed for all words in training, or only on training words whose frequency is below some threshold. The result of the known-word templates and word-signature features is a very large set of features. Generally a feature cutoff is used in which features are thrown out if they have count < 5 in the training set.

8.5.2 Decoding and Training MEMMs

The most likely sequence of tags is then computed by combining these features of the input word w_i, its neighbors within l words w_{i-l}^{i+l}, and the previous k tags t_{i-k}^{i-1} as follows (using θ to refer to feature weights instead of w to avoid the confusion with w meaning words):

$$\begin{aligned}\hat{T} &= \operatorname*{argmax}_T P(T|W) \\ &= \operatorname*{argmax}_T \prod_i P(t_i|w_{i-l}^{i+l}, t_{i-k}^{i-1}) \\ &= \operatorname*{argmax}_T \prod_i \frac{\exp\left(\sum_j \theta_j f_j(t_i, w_{i-l}^{i+l}, t_{i-k}^{i-1})\right)}{\sum_{t' \in \text{tagset}} \exp\left(\sum_j \theta_j f_j(t', w_{i-l}^{i+l}, t_{i-k}^{i-1})\right)}\end{aligned} \tag{8.34}$$

How should we decode to find this optimal tag sequence $\hat{T}$? The simplest way to turn logistic regression into a sequence model is to build a local classifier that classifies each word left to right, making a hard classification on the first word in the sentence, then a hard decision on the second word, and so on. This is called a **greedy** decoding algorithm, because we greedily choose the best tag for each word, as shown in Fig. 8.14.

greedy

function GREEDY SEQUENCE DECODING(words W, model P) **returns** tag sequence T

for i = 1 **to** *length(W)*
  $\hat{t}_i = \operatorname*{argmax}_{t' \in T} P(t' \mid w_{i-l}^{i+l}, t_{i-k}^{i-1})$

Figure 8.14 In greedy decoding we simply run the classifier on each token, left to right, each time making a hard decision about which is the best tag.

The problem with the greedy algorithm is that by making a hard decision on each word before moving on to the next word, the classifier can't use evidence from future decisions. Although the greedy algorithm is very fast, and occasionally has sufficient accuracy to be useful, in general the hard decision causes too great a drop in performance, and we don't use it.

Viterbi

Instead we decode an MEMM with the **Viterbi** algorithm just as with the HMM, finding the sequence of part-of-speech tags that is optimal for the whole sentence.

For example, assume that our MEMM is only conditioning on the previous tag t_{i-1} and observed word w_i. Concretely, this involves filling an $N \times T$ array with the appropriate values for $P(t_i|t_{i-1},w_i)$, maintaining backpointers as we proceed. As with HMM Viterbi, when the table is filled, we simply follow pointers back from the maximum value in the final column to retrieve the desired set of labels. The requisite changes from the HMM-style application of Viterbi have to do only with how we fill each cell. Recall from Eq. 8.20 that the recursive step of the Viterbi equation computes the Viterbi value of time t for state j as

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, a_{ij}\, b_j(o_t); \quad 1 \le j \le N, 1 < t \le T \tag{8.35}$$

which is the HMM implementation of

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, P(s_j|s_i)\, P(o_t|s_j) \quad 1 \le j \le N, 1 < t \le T \tag{8.36}$$

The MEMM requires only a slight change to this latter formula, replacing the a and b prior and likelihood probabilities with the direct posterior:

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i)\, P(s_j|s_i,o_t) \quad 1 \le j \le N, 1 < t \le T \tag{8.37}$$

Learning in MEMMs relies on the same supervised learning algorithms we presented for logistic regression. Given a sequence of observations, feature functions, and corresponding hidden states, we use gradient descent to train the weights to maximize the log-likelihood of the training corpus.

8.6 Bidirectionality

The one problem with the MEMM and HMM models as presented is that they are exclusively run left-to-right. While the Viterbi algorithm still allows present decisions to be influenced indirectly by future decisions, it would help even more if a decision about word w_i could directly use information about future tags t_{i+1} and t_{i+2}.

label bias
observation bias

Adding bidirectionality has another useful advantage. MEMMs have a theoretical weakness, referred to alternatively as the **label bias** or **observation bias** problem (Lafferty et al. 2001, Toutanova et al. 2003). These are names for situations when one source of information is ignored because it is **explained away** by another source. Consider an example from Toutanova et al. (2003), the sequence *will/NN to/TO fight/VB*. The tag TO is often preceded by NN but rarely by modals (MD), and so that tendency should help predict the correct NN tag for *will*. But the previous transition $P(t_{will}|\langle s\rangle)$ prefers the modal, and because $P(TO|to,t_{will})$ is so close to 1 regardless of t_{will} the model cannot make use of the transition probability and incorrectly chooses MD. The strong information that *to* must have the tag TO has **explained away** the presence of TO and so the model doesn't learn the importance of

the previous NN tag for predicting TO. Bidirectionality helps the model by making the link between TO available when tagging the NN.

CRF

One way to implement bidirectionality is to switch to a more powerful model called a **conditional random field** or **CRF**. The CRF is an undirected graphical model, which means that it's not computing a probability for each tag at each time step. Instead, at each time step the CRF computes log-linear functions over a **clique**, a set of relevant features. Unlike for an MEMM, these might include output features of words in future time steps. The probability of the best sequence is similarly computed by the Viterbi algorithm. Because a CRF normalizes probabilities over all tag sequences, rather than over all the tags at an individual time t, training requires computing the sum over all possible labelings, which makes CRF training quite slow.

Stanford tagger

Simpler methods can also be used; the **Stanford tagger** uses a bidirectional version of the MEMM called a cyclic dependency network (Toutanova et al., 2003).

Alternatively, any sequence model can be turned into a bidirectional model by using multiple passes. For example, the first pass would use only part-of-speech features from already-disambiguated words on the left. In the second pass, tags for all words, including those on the right, can be used. Alternately, the tagger can be run twice, once left-to-right and once right-to-left. In greedy decoding, for each word the classifier chooses the highest-scoring of the tags assigned by the left-to-right and right-to-left classifier. In Viterbi decoding, the classifier chooses the higher scoring of the two sequences (left-to-right or right-to-left). These bidirectional models lead directly into the bi-LSTM models that we will introduce in Chapter 9 as a standard neural sequence model.

8.7 Part-of-Speech Tagging for Morphological Rich Languages

Augmentations to tagging algorithms become necessary when dealing with languages with rich morphology like Czech, Hungarian and Turkish.

These productive word-formation processes result in a large vocabulary for these languages: a 250,000 word token corpus of Hungarian has more than twice as many word types as a similarly sized corpus of English (Oravecz and Dienes, 2002), while a 10 million word token corpus of Turkish contains four times as many word types as a similarly sized English corpus (Hakkani-Tür et al., 2002). Large vocabularies mean many unknown words, and these unknown words cause significant performance degradations in a wide variety of languages (including Czech, Slovene, Estonian, and Romanian) (Hajič, 2000).

Highly inflectional languages also have much more information than English coded in word morphology, like **case** (nominative, accusative, genitive) or **gender** (masculine, feminine). Because this information is important for tasks like parsing and coreference resolution, part-of-speech taggers for morphologically rich languages need to label words with case and gender information. Tagsets for morphologically rich languages are therefore sequences of morphological tags rather than a single primitive tag. Here's a Turkish example, in which the word *izin* has three possible morphological/part-of-speech tags and meanings (Hakkani-Tür et al., 2002):

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. İçeri girmek için **izin** alman gerekiyor. izin + `Noun+A3sg+Pnon+Nom`
   You need **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, and so tagsets can be 4 to 10 times larger than the 50–100 tags we have seen for English. With such large tagsets, each word needs to be morphologically analyzed to generate the list of possible morphological tag sequences (part-of-speech tags) for the word. The role of the tagger is then to disambiguate among these tags. This method also helps with unknown words since morphological parsers can accept unknown stems and still segment the affixes properly.

For non-word-space languages like Chinese, word segmentation (Chapter 2) is either applied before tagging or done jointly. Although Chinese words are on average very short (around 2.4 characters per unknown word compared with 7.7 for English) the problem of unknown words is still large. While English unknown words tend to be proper nouns in Chinese the majority of unknown words are common nouns and verbs because of extensive compounding. Tagging models for Chinese use similar unknown word features to English, including character prefix and suffix features, as well as novel features like the **radicals** of each character in a word. (Tseng et al., 2005).

A standard for multilingual tagging is the Universal POS tag set of the Universal Dependencies project, which contains 16 tags plus a wide variety of features that can be added to them to create a large tagset for any language (Nivre et al., 2016).

8.8 Summary

This chapter introduced **parts of speech** and **part-of-speech tagging**:

- Languages generally have a small set of **closed class** words that are highly frequent, ambiguous, and act as **function words**, and **open-class** words like **nouns**, **verbs**, **adjectives**. Various part-of-speech **tagsets** exist, 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.
- Two common approaches to **sequence modeling** are a **generative** approach, **HMM** tagging, and a **discriminative** approach, **MEMM** tagging. We will see a third, discriminative neural approach in Chapter 9.
- The probabilities in HMM taggers are estimated by maximum likelihood estimation on tag-labeled training corpora. The Viterbi algorithm is used for **decoding**, finding the most likely tag sequence
- Beam search is a variant of Viterbi decoding that maintains only a fraction of high scoring states rather than all states during decoding.
- **Maximum entropy Markov model** or **MEMM taggers** train logistic regression models to pick the best tag given an observation word and its context and the previous tags, and then use Viterbi to choose the best sequence of tags.
- Modern taggers are generally run **bidirectionally**.

Bibliographical and Historical Notes

What is probably the earliest part-of-speech tagger was part of the parser in Zellig Harris's Transformations and Discourse Analysis Project (TDAP), implemented between June 1958 and July 1959 at the University of Pennsylvania (Harris, 1962), although earlier systems had used part-of-speech dictionaries. TDAP used 14 handwritten rules for part-of-speech disambiguation; the use of part-of-speech tag sequences and the relative frequency of tags for a word prefigures all modern algorithms. The parser was implemented essentially as a cascade of finite-state transducers; see Joshi and Hopely (1999) and Karttunen (1999) for a reimplementation.

The Computational Grammar Coder (CGC) of Klein and Simmons (1963) had three components: a lexicon, a morphological analyzer, and a context disambiguator. The small 1500-word lexicon listed only function words and other irregular words. The morphological analyzer used inflectional and derivational suffixes to assign part-of-speech classes. These were run over words to produce candidate parts of speech which were then disambiguated by a set of 500 context rules 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 TAGGIT tagger (Greene and Rubin, 1971) used the same architecture as Klein and Simmons (1963), with a bigger dictionary and more tags (87). TAGGIT was applied to the Brown corpus and, according to Francis and Kučera (1982, p. 9), accurately tagged 77% of the corpus; the remainder of the Brown corpus was then tagged by hand. All these early algorithms were based on a two-stage architecture in which a dictionary was first used to assign each word a set of potential parts of speech, and then lists of handwritten disambiguation rules winnowed the set down to a single part of speech per word.

Soon afterwards probabilistic architectures began to be developed. Probabilities were used in tagging by Stolz et al. (1965) and a complete probabilistic tagger with Viterbi decoding was sketched by Bahl and Mercer (1976). The Lancaster-Oslo/Bergen (LOB) corpus, a British English equivalent of the Brown corpus, was tagged in the early 1980's with the CLAWS tagger (Marshall 1983; Marshall 1987; Garside 1987), a probabilistic algorithm that approximated a simplified HMM tagger. The algorithm used tag bigram probabilities, but instead of storing the word likelihood of each tag, the algorithm marked tags either as *rare* ($P(\text{tag}|\text{word}) < .01$) *infrequent* ($P(\text{tag}|\text{word}) < .10$) or *normally frequent* ($P(\text{tag}|\text{word}) > .10$).

DeRose (1988) developed a quasi-HMM algorithm, including the use of dynamic programming, although computing $P(t|w)P(w)$ instead of $P(w|t)P(w)$. The same year, the probabilistic PARTS tagger of Church (1988), (1989) was probably the first implemented HMM tagger, described correctly in Church (1989), although Church (1988) also described the computation incorrectly as $P(t|w)P(w)$ instead of $P(w|t)P(w)$. Church (p.c.) explained that he had simplified for pedagogical purposes because using the probability $P(t|w)$ made the idea seem more understandable as "storing a lexicon in an almost standard form".

Later taggers explicitly introduced the use of the hidden Markov model (Kupiec 1992; Weischedel et al. 1993; Schütze and Singer 1994). Merialdo (1994) showed that fully unsupervised EM didn't work well for the tagging task and that reliance on hand-labeled data was important. Charniak et al. (1993) showed the importance of the most frequent tag baseline; the 92.3% number we give above was from Abney et al. (1999). See Brants (2000) for many implementation details of an HMM tagger whose performance is still roughly close to state of the art taggers.

Ratnaparkhi (1996) introduced the MEMM tagger, called MXPOST, and the modern formulation is very much based on his work.

The idea of using letter suffixes for unknown words is quite old; the early Klein and Simmons (1963) system checked all final letter suffixes of lengths 1-5. The probabilistic formulation we described for HMMs comes from Samuelsson (1993). The unknown word features described on page 19 come mainly from (Ratnaparkhi, 1996), with augmentations from Toutanova et al. (2003) and Manning (2011).

State of the art taggers use neural algorithms like the sequence models in Chapter 9 or (bidirectional) log-linear models Toutanova et al. (2003). HMM (Brants 2000; Thede and Harper 1999) and MEMM tagger accuracies are likely just a tad lower.

An alternative modern formalism, the English Constraint Grammar systems (Karlsson et al. 1995; Voutilainen 1995; Voutilainen 1999), uses a two-stage formalism much like the early taggers from the 1950s and 1960s. A morphological analyzer with tens of thousands of English word stem entries returns all parts of speech for a word, using a large feature-based tagset. So the word *occurred* is tagged with the options ⟨V PCP2 SV⟩ and ⟨V PAST VFIN SV⟩, meaning it can be a participle (PCP2) for an intransitive (SV) verb, or a past (PAST) finite (VFIN) form of an intransitive (SV) verb. A set of 3,744 constraints are then applied to the input sentence to rule out parts of speech inconsistent with the context. For example here's a rule for the ambiguous word *that* that eliminates all tags except the ADV (adverbial intensifier) sense (this is the sense in the sentence *it isn't that odd*):

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

Manning (2011) investigates the remaining 2.7% of errors in a high-performing tagger, the bidirectional MEMM-style model described above (Toutanova et al., 2003). He suggests that a third or half of these remaining errors are due to errors or inconsistencies in the training data, a third might be solvable with richer linguistic models, and for the remainder the task is underspecified or unclear.

Supervised tagging relies heavily on in-domain training data hand-labeled by experts. Ways to relax this assumption include unsupervised algorithms for clustering words into part-of-speech-like classes, summarized in Christodoulopoulos et al. (2010), and ways to combine labeled and unlabeled data, for example by co-training (Clark et al. 2003; Søgaard 2010).

See Householder (1995) for historical notes on parts of speech, and Sampson (1987) and Garside et al. (1997) on the provenance of the Brown and other tagsets.

Exercises

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

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

8.2 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.

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

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

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

8.5 Build a bigram HMM tagger. You will need a part-of-speech-tagged corpus. First split the corpus into a training set and test set. From the labeled training set, train the transition and observation probabilities of the HMM tagger directly on the hand-tagged data. Then implement the Viterbi algorithm so that you can label an arbitrary test sentence. Now run your algorithm on the test set. Report its error rate and compare its performance to the most frequent tag baseline.

8.6 Do an error analysis of your tagger. Build a confusion matrix and investigate the most frequent errors. Propose some features for improving the performance of your tagger on these errors.

Abney, S. P., Schapire, R. E., and Singer, Y. (1999). Boosting applied to tagging and PP attachment. In *EMNLP/VLC-99*, 38–45.

Bahl, L. R. and Mercer, R. L. (1976). Part of speech assignment by a statistical decision algorithm. In *Proceedings IEEE International Symposium on Information Theory*, 88–89.

Brants, T. (2000). TnT: A statistical part-of-speech tagger. In *ANLP 2000*, 224–231.

Broschart, J. (1997). Why Tongan does it differently. *Linguistic Typology*, *1*, 123–165.

Charniak, E., Hendrickson, C., Jacobson, N., and Perkowitz, M. (1993). Equations for part-of-speech tagging. In *AAAI-93*, 784–789. AAAI Press.

Christodoulopoulos, C., Goldwater, S., and Steedman, M. (2010). Two decades of unsupervised POS induction: How far have we come?. In *EMNLP-10*.

Church, K. W. (1988). A stochastic parts program and noun phrase parser for unrestricted text. In *ANLP 1988*, 136–143.

Church, K. W. (1989). A stochastic parts program and noun phrase parser for unrestricted text. In *ICASSP-89*, 695–698.

Clark, S., Curran, J. R., and Osborne, M. (2003). Bootstrapping pos taggers using unlabelled data. In *CoNLL-03*, 49–55.

DeRose, S. J. (1988). Grammatical category disambiguation by statistical optimization. *Computational Linguistics*, *14*, 31–39.

Evans, N. (2000). Word classes in the world's languages. In Booij, G., Lehmann, C., and Mugdan, J. (Eds.), *Morphology: A Handbook on Inflection and Word Formation*, 708–732. Mouton.

Francis, W. N. and Kučera, H. (1982). *Frequency Analysis of English Usage*. Houghton Mifflin, Boston.

Garside, R. (1987). The CLAWS word-tagging system. In Garside, R., Leech, G., and Sampson, G. (Eds.), *The Computational Analysis of English*, 30–41. Longman.

Garside, R., Leech, G., and McEnery, A. (1997). *Corpus Annotation*. Longman.

Gil, D. (2000). Syntactic categories, cross-linguistic variation and universal grammar. In Vogel, P. M. and Comrie, B. (Eds.), *Approaches to the Typology of Word Classes*, 173–216. Mouton.

Greene, B. B. and Rubin, G. M. (1971). Automatic grammatical tagging of English. Department of Linguistics, Brown University, Providence, Rhode Island.

Hajič, J. (2000). Morphological tagging: Data vs. dictionaries. In *NAACL 2000*. Seattle.

Hakkani-Tür, D., Oflazer, K., and Tür, G. (2002). Statistical morphological disambiguation for agglutinative languages. *Journal of Computers and Humanities*, *36*(4), 381–410.

Harris, Z. S. (1962). *String Analysis of Sentence Structure*. Mouton, The Hague.

Householder, F. W. (1995). Dionysius Thrax, the *technai*, and Sextus Empiricus. In Koerner, E. F. K. and Asher, R. E. (Eds.), *Concise History of the Language Sciences*, 99–103. Elsevier Science.

Jelinek, F. and Mercer, R. L. (1980). Interpolated estimation of Markov source parameters from sparse data. In Gelsema, E. S. and Kanal, L. N. (Eds.), *Proceedings, Workshop on Pattern Recognition in Practice*, 381–397. North Holland.

Joshi, A. K. and Hopely, P. (1999). A parser from antiquity. In Kornai, A. (Ed.), *Extended Finite State Models of Language*, 6–15. Cambridge University Press.

Karlsson, F., Voutilainen, A., Heikkilä, J., and Anttila, A. (Eds.). (1995). *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text*. Mouton de Gruyter.

Karttunen, L. (1999). Comments on Joshi. In Kornai, A. (Ed.), *Extended Finite State Models of Language*, 16–18. Cambridge University Press.

Klein, S. and Simmons, R. F. (1963). A computational approach to grammatical coding of English words. *Journal of the Association for Computing Machinery*, *10*(3), 334–347.

Kupiec, J. (1992). Robust part-of-speech tagging using a hidden Markov model. *Computer Speech and Language*, *6*, 225–242.

Lafferty, J. D., McCallum, A., and Pereira, F. C. N. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML 2001*.

Manning, C. D. (2011). Part-of-speech tagging from 97% to 100%: Is it time for some linguistics?. In *CICLing 2011*, 171–189.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, *19*(2), 313–330.

Marshall, I. (1983). Choice of grammatical word-class without GLobal syntactic analysis: Tagging words in the LOB corpus. *Computers and the Humanities*, *17*, 139–150.

Marshall, I. (1987). Tag selection using probabilistic methods. In Garside, R., Leech, G., and Sampson, G. (Eds.), *The Computational Analysis of English*, 42–56. Longman.

Merialdo, B. (1994). Tagging English text with a probabilistic model. *Computational Linguistics*, *20*(2), 155–172.

Nivre, J., de Marneffe, M.-C., Ginter, F., Goldberg, Y., Hajič, J., Manning, C. D., McDonald, R., Petrov, S., Pyysalo, S., Silveira, N., Tsarfaty, R., and Zeman, D. (2016). Universal Dependencies v1: A multilingual treebank collection. In *LREC*.

Oravecz, C. and Dienes, P. (2002). Efficient stochastic part-of-speech tagging for Hungarian. In *LREC-02*, 710–717.

Ratnaparkhi, A. (1996). A maximum entropy part-of-speech tagger. In *EMNLP 1996*, 133–142.

Sampson, G. (1987). Alternative grammatical coding systems. In Garside, R., Leech, G., and Sampson, G. (Eds.), *The Computational Analysis of English*, 165–183. Longman.

Samuelsson, C. (1993). Morphological tagging based entirely on Bayesian inference. In *9th Nordic Conference on Computational Linguistics NODALIDA-93*. Stockholm.

Schütze, H. and Singer, Y. (1994). Part-of-speech tagging using a variable memory Markov model. In *ACL-94*, 181–187.

Søgaard, A. (2010). Simple semi-supervised training of part-of-speech taggers. In *ACL 2010*, 205–208.

Stolz, W. S., Tannenbaum, P. H., and Carstensen, F. V. (1965). A stochastic approach to the grammatical coding of English. *CACM*, *8*(6), 399–405.

Thede, S. M. and Harper, M. P. (1999). A second-order hidden Markov model for part-of-speech tagging. In *ACL-99*, 175–182.

Toutanova, K., Klein, D., Manning, C. D., and Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *HLT-NAACL-03*.

Tseng, H., Jurafsky, D., and Manning, C. D. (2005). Morphological features help POS tagging of unknown words across language varieties. In *Proceedings of the 4th SIGHAN Workshop on Chinese Language Processing*.

Voutilainen, A. (1995). Morphological disambiguation. In Karlsson, F., Voutilainen, A., Heikkilä, J., and Anttila, A. (Eds.), *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text*, 165–284. Mouton de Gruyter.

Voutilainen, A. (1999). Handcrafted rules. In van Halteren, H. (Ed.), *Syntactic Wordclass Tagging*, 217–246. Kluwer.

Weischedel, R., Meteer, M., Schwartz, R., Ramshaw, L. A., and Palmucci, J. (1993). Coping with ambiguity and unknown words through probabilistic models. *Computational Linguistics*, *19*(2), 359–382.