CHAPTER 9

Part-of-Speech Tagging

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”) 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 Greek, Latin, and most European languages for the next 2000 years.

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, 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. Nonetheless, eight isn’t very many and more recent part-of-speech tagsets have many more word classes, like the 45 tags used by the Penn Treebank (\?).

Parts-of-speech (also known as POS, word classes, or syntactic categories) are useful because of the large amount of information they give about a word and its neighbors. Knowing whether a word is a noun or a verb tells us a lot about likely neighboring words (nouns are preceded by determiners and adjectives, verbs by nouns) and about the syntactic structure around the word (nouns are generally part of noun phrases), which makes part-of-speech tagging an important component of syntactic parsing (Chapter 12). Parts of speech are useful features for finding named
entities like people or organizations in text and other information extraction tasks (Chapter 18). Parts-of-speech influence the possible morphological affixes and so can influence stemming for informational retrieval, and can help in summarization for improving the selection of nouns or other important words from a document. A word’s part of speech is important for producing pronunciations in speech synthesis and recognition. The word content, for example, is pronounced CONTENT when it is a noun and content when it is an adjective (Chapter 26).

This chapter focuses on computational methods for assigning parts-of-speech to words, part-of-speech tagging. After summarizing English word classes and the standard Penn tagset, we introduce two algorithms for tagging: the Hidden Markov Model (HMM) and the Maximum Entropy Markov Model (MEMM).

9.1 (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 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).

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.

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. 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).

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. 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).

The verb class includes most of the words referring 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 (?; ?; ?).

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”.

The final open class form, adverbs, is rather a hodge-podge, both semantically and formally. In the following sentence from ? (?) all the italicized words are adverbs:

Unfortunately, John walked home extremely slowly 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 “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 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.

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
- **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 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.

When a verb and a particle behave as a single syntactic and/or semantic unit, we call the combination a phrasal verb. Phrasal verbs cause widespread problems with natural language processing because they often behave as a semantic unit with a non-compositional meaning—one that is not predictable from the distinct 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’.
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 21). 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.

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 subordinated 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**.

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

A closed class subtype of English verbs are the **auxiliary** verbs. Cross-linguistically, auxiliaries 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, 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* is used, for example, to 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. The modals are used to mark the mood associated with the event or action depicted by the main verb: *can* indicates ability or possibility, *may* indicates permission or possibility, *must* indicates necessity. In addition to the perfect *have* mentioned above, there is a modal verb *have* (e.g., *I have to go*), which is common in spoken English.

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.

### 9.2 The Penn Treebank Part-of-Speech Tagset

While there are many lists of parts-of-speech, most modern language processing on English uses the 45-tag Penn Treebank tagset (*?), shown in Fig. 9.1. This tagset has been used to label a wide variety of corpora, including the Brown corpus, the Wall Street Journal corpus, and the Switchboard corpus.
### The Penn Treebank Part-of-Speech Tagset

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
<th>Example</th>
<th>Tag</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>coordin. conjunction</td>
<td><em>and, but, or</em></td>
<td>SYM</td>
<td>symbol</td>
<td><em>+%, &amp;</em></td>
</tr>
<tr>
<td>CD</td>
<td>cardinal number</td>
<td><em>one, two</em></td>
<td>TO</td>
<td>“to”</td>
<td><em>to</em></td>
</tr>
<tr>
<td>DT</td>
<td>determiner</td>
<td><em>a, the</em></td>
<td>UH</td>
<td>interjection</td>
<td><em>ah, oops</em></td>
</tr>
<tr>
<td>EX</td>
<td>existential ‘there’</td>
<td><em>there</em></td>
<td>VB</td>
<td>verb base form</td>
<td><em>eat</em></td>
</tr>
<tr>
<td>FW</td>
<td>foreign word</td>
<td><em>mea culpa</em></td>
<td>VBD</td>
<td>verb past tense</td>
<td><em>ate</em></td>
</tr>
<tr>
<td>IN</td>
<td>preposition/sub-conj</td>
<td><em>of, in, by</em></td>
<td>VBG</td>
<td>verb gerund</td>
<td><em>eating</em></td>
</tr>
<tr>
<td>JJ</td>
<td>adjective</td>
<td><em>yellow</em></td>
<td>VBN</td>
<td>verb past participle</td>
<td><em>eaten</em></td>
</tr>
<tr>
<td>JJR</td>
<td>adj., comparative</td>
<td><em>bigger</em></td>
<td>VBP</td>
<td>verb non-3sg pres</td>
<td><em>eat</em></td>
</tr>
<tr>
<td>JJS</td>
<td>adj., superlative</td>
<td><em>wildest</em></td>
<td>VBZ</td>
<td>verb 3sg pres</td>
<td><em>eats</em></td>
</tr>
<tr>
<td>LS</td>
<td>list item marker</td>
<td><em>1, 2, One</em></td>
<td>WDT</td>
<td>wh-determiner</td>
<td><em>which, that</em></td>
</tr>
<tr>
<td>MD</td>
<td>modal</td>
<td><em>can, should</em></td>
<td>WP</td>
<td>wh-pronoun</td>
<td><em>what, who</em></td>
</tr>
<tr>
<td>NN</td>
<td>noun, sing. or mass</td>
<td><em>llama</em></td>
<td>WPS</td>
<td>possessive wh-</td>
<td><em>whose</em></td>
</tr>
<tr>
<td>NNS</td>
<td>noun, plural</td>
<td><em>llamas</em></td>
<td>WRB</td>
<td>wh-adverb</td>
<td><em>how, where</em></td>
</tr>
<tr>
<td>NNP</td>
<td>proper noun, sing.</td>
<td><em>IBM</em></td>
<td>$</td>
<td>dollar sign</td>
<td><em>$</em></td>
</tr>
<tr>
<td>NNPS</td>
<td>proper noun, plural</td>
<td><em>Carolinas</em></td>
<td>#</td>
<td>pound sign</td>
<td><em>#</em></td>
</tr>
<tr>
<td>PDT</td>
<td>predeterminer</td>
<td><em>all, both</em></td>
<td>’</td>
<td>left quote</td>
<td><em>’ or ‘</em></td>
</tr>
<tr>
<td>POS</td>
<td>possessive ending</td>
<td>’s</td>
<td>”</td>
<td>right quote</td>
<td><em>’ or ”</em></td>
</tr>
<tr>
<td>PRP</td>
<td>personal pronoun</td>
<td><em>I, you, he</em></td>
<td>(</td>
<td>left parenthesis</td>
<td>[*], {, &lt;</td>
</tr>
<tr>
<td>PRPS</td>
<td>possessive pronoun</td>
<td><em>your, one’s</em></td>
<td>)</td>
<td>right parenthesis</td>
<td>*], }, &gt;</td>
</tr>
<tr>
<td>RB</td>
<td>adverb</td>
<td><em>quickly, never</em></td>
<td>,</td>
<td>comma</td>
<td>,</td>
</tr>
<tr>
<td>RBR</td>
<td>adverb, comparative</td>
<td><em>faster</em></td>
<td>.</td>
<td>sentence-final punc</td>
<td>. ! ?</td>
</tr>
<tr>
<td>RBS</td>
<td>adverb, superlative</td>
<td><em>fastest</em></td>
<td>:</td>
<td>mid-sentence punc</td>
<td>: ; ... – -</td>
</tr>
<tr>
<td>RP</td>
<td>particle</td>
<td><em>up, off</em></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 9.1  Penn Treebank part-of-speech tags (including punctuation).

Parts-of-speech are generally represented by placing the tag after each word, delimited by a slash, as in the following examples:

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

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

(9.3) 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 (9.1) 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 (9.2) 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 (9.3) shows the segmentation of the possessive morpheme ’s a passive construction, ‘were reported’, in which reported is marked 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 like the Treebank corpora 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 (see Section 9.7 for other languages). 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.

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 the Switchboard
corpus the tag TO is reserved for the infinitive use of to, while the preposition use is
tagged IN:

Well/UH ,/ 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, be-
cause the Penn 45 tags were was collapsed from a larger 87-tag tagset, the original
Brown tagset, some potential useful distinctions were lost. The Penn tagset was de-
dsigned 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

Tagging algorithms assume that words have been tokenized before tagging. 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 multipart words. The Treebank tagset as-
sumes 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/N. The C5 tagset for the British National Cor-
pus., by contrast, allow prepositions like “in terms of” to be treated as a single word
by adding numbers to each tag, as in in/I31 terms/I32 of/I33.

9.3 Part-of-Speech Tagging

tagging

Part-of-speech tagging (tagging for short) is the process of assigning a part-of-
speech marker to each word in an input text. Because tags are generally also applied
to punctuation, tokenization 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 punc-
tuation (such as in abbreviations like e.g. and etc.)

The input to a tagging algorithm is a sequence of words and a tagset, and the
output is a sequence of tags, a single best tag for each word as shown in the examples
on the previous pages.

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, the word book can be a verb (book that flight) or a noun (as in 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 problem of POS-tagging is to **resolve** these ambiguities, choosing the proper tag for the context. Part-of-speech tagging is thus one of the many **disambiguation** tasks in language processing.

How hard is the tagging problem? And how common is tag ambiguity? Fig. 9.2 shows the answer for the Brown and WSJ corpora tagged using the 45-tag Penn tagset. Most word types (80-86%) are unambiguous; that is, they have only a single tag (Janet is always NNP, funniest JJS, and hesitantly RB). But the ambiguous words, although accounting for only 14-15% of the vocabulary, are some of the most common words of English, and hence 55-67% of word tokens in running text are ambiguous. Note the large differences across the two genres, especially in token frequency. Tags in the WSJ corpus are less ambiguous, presumably because this newspaper’s specific focus on financial news leads to a more limited distribution of word usages than the more general texts combined into the Brown corpus.

<table>
<thead>
<tr>
<th>Types</th>
<th>WSJ</th>
<th>Brown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unambiguous</td>
<td>44,432</td>
<td>45,799</td>
</tr>
<tr>
<td>Ambiguous</td>
<td>7,025</td>
<td>8,050</td>
</tr>
<tr>
<td><strong>Tokens</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unambiguous</td>
<td>577,421</td>
<td>384,349</td>
</tr>
<tr>
<td>Ambiguous</td>
<td>711,780</td>
<td>786,646</td>
</tr>
</tbody>
</table>

Figure 9.2 The amount of tag ambiguity for word types in the Brown and WSJ corpora, from the Treebank-3 (45-tag) tagging. These statistics include punctuation as words, and assume words are 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

Still, even many of the ambiguous tokens are easy to disambiguate. This is because the different 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. 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).

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 on a human-labeled test set. One commonly used test set is sections 22-24 of the WSJ corpus. If we train on the rest of the WSJ corpus and test on that test set, 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 a number of statistical algo-
rithms including HMMs, MEMMs and other log-linear models, perceptrons, and probably also rule-based systems—see the discussion at the end of the chapter. See Section 9.7 on other languages and genres.

9.4 HMM Part-of-Speech Tagging

In this section we introduce the use of the Hidden Markov Model for part-of-speech tagging. The HMM defined in the previous chapter was quite powerful, including a learning algorithm—the Baum-Welch (EM) algorithm—that can be given unlabeled data and find the best mapping of labels to observations. However when we apply HMM to part-of-speech tagging we generally don’t use the Baum-Welch algorithm for learning the HMM parameters. Instead HMMs for part-of-speech tagging are trained on a fully labeled dataset—a set of sentences with each word annotated with a part-of-speech tag—setting parameters by maximum likelihood estimates on this training data.

Thus the only algorithm we will need from the previous chapter is the Viterbi algorithm for decoding, and we will also need to see how to set the parameters from training data.

9.4.1 The basic equation of HMM Tagging

Let’s begin with a quick reminder of the intuition of HMM decoding. The goal of HMM decoding is to choose the tag sequence that is most probable given the observation sequence of \( n \) words \( w^n \):

\[
\hat{t}^n = \arg\max_{t^n} P(t^n | w^n)
\]  

(9.4)

by using Bayes’ rule to instead compute:

\[
\hat{t}^n = \arg\max_{t^n} \frac{P(w^n | t^n)P(t^n)}{P(w^n)}
\]  

(9.5)

Furthermore, we simplify Eq. 9.5 by dropping the denominator \( P(w^n) \):

\[
\hat{t}^n = \arg\max_{t^n} P(w^n | t^n)P(t^n)
\]  

(9.6)

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:

\[
P(w^n | t^n) \approx \prod_{i=1}^n P(w_i | t_i)
\]  

(9.7)

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:

\[
P(t^n_1) \approx \prod_{i=1}^n P(t_i | t_{i-1})
\]  

(9.8)
Plugging the simplifying assumptions from Eq. 9.7 and Eq. 9.8 into Eq. 9.6 results in the following equation for the most probable tag sequence from a bigram tagger, which as we will soon see, correspond to the emission probability and transition probability from the HMM of Chapter 7.

\[
P_{i}^{n} = \arg\max_{t_{n}^{1}} P(t_{n}^{1}|w_{n}^{1}) \approx \arg\max_{t_{n}^{1}} \prod_{i=1}^{n} P(w_{i}|t_{i}) P(t_{i}|t_{i-1})
\]  

(9.9)

### 9.4.2 Estimating probabilities

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

In HMM tagging, rather than using the full power of HMM EM learning, the probabilities are estimated just by counting on a tagged training corpus. For this example we’ll use the tagged WSJ corpus. The tag transition probabilities \(P(t_{i}|t_{i-1})\) represent the probability of a tag 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. The maximum likelihood estimate of a transition probability is computed 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

\[
P(t_{i}|t_{i-1}) = \frac{C(t_{i-1},t_{i})}{C(t_{i-1})}
\]

(9.10)

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

\[
P(VB|MD) = \frac{C(MD,VB)}{C(MD)} = \frac{10471}{13124} = .80
\]

(9.11)

The 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

\[
P(w_{i}|t_{i}) = \frac{C(t_{i},w_{i})}{C(t_{i})}
\]

(9.12)

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

\[
P(will|MD) = \frac{C(MD,will)}{C(MD)} = \frac{4046}{13124} = .31
\]

(9.13)

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 *will*?” That would be the posterior \(P(MD|will)\). Instead, \(P(will|MD)\) answers the slightly counterintuitive question “If we were going to generate a MD, how likely is it that this modal would be *will*?”

The two kinds of probabilities from Eq. 9.9, the transition (prior) probabilities like \(P(VB|MD)\) and the emission (likelihood) probabilities like \(P(will|MD)\), correspond to the \(A\) transition probabilities, and \(B\) observation likelihoods of the HMM. Figure 9.3 illustrates some of the the \(A\) transition probabilities for three states in an HMM part-of-speech tagger; the full tagger would have one state for each tag.

Figure 9.4 shows another view of these three states from an HMM tagger, focusing on the word likelihoods \(B\). Each hidden state is associated with a vector of likelihoods for each observation word.
9.4.3 Working through an example

Let’s now work through an example of computing the best sequence of tags that corresponds to the following sequence of words

\[(9.14)\] Janet will back the bill

The correct series of tags is:

\[(9.15)\] Janet/NNP will/MD back/VB the/DT bill/NN

Let the HMM be defined by the two tables in Fig. 9.5 and Fig. 9.6.

Figure 9.5 lists the \(a_{ij}\) probabilities for transitioning between the hidden states (part-of-speech tags).

Figure 9.6 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 9.5  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.

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

Figure 9.6  Observation likelihoods $B$ computed from the WSJ corpus without smoothing.

<table>
<thead>
<tr>
<th>Janet</th>
<th>will</th>
<th>back</th>
<th>the</th>
<th>bill</th>
</tr>
</thead>
<tbody>
<tr>
<td>NNP</td>
<td>0.000032</td>
<td>0</td>
<td>0</td>
<td>0.000048</td>
</tr>
<tr>
<td>MD</td>
<td>0</td>
<td>0.308431</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>VB</td>
<td>0</td>
<td>0.000028</td>
<td>0.000672</td>
<td>0</td>
</tr>
<tr>
<td>JJ</td>
<td>0</td>
<td>0</td>
<td>0.000340</td>
<td>0.000097</td>
</tr>
<tr>
<td>NN</td>
<td>0</td>
<td>0.000200</td>
<td>0.000223</td>
<td>0.000006</td>
</tr>
<tr>
<td>RB</td>
<td>0</td>
<td>0</td>
<td>0.010446</td>
<td>0</td>
</tr>
<tr>
<td>DT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.506099</td>
</tr>
</tbody>
</table>

Figure 9.7  A schematic of the tagging task for the sample sentence, showing the ambiguities for each word and the correct tag sequence as the highlighted path through the hidden states.

Figure 9.7 shows a schematic of the possible tags for each word and the correct final tag sequence.

For the reader’s convenience Fig. 9.8 repeats the pseudocode for the Viterbi algorithm from Chapter 7. 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.

The algorithm first creates $N = 5$ state columns, the first for the observation of the first word Janet, the second for will, and so on. 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 start state) and the observation probability (of the first word); the reader should find this in Fig. 9.9.
**Chapter 9 • Part-of-Speech Tagging**

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

<table>
<thead>
<tr>
<th>create a path probability matrix viterbi[N+2,T]</th>
</tr>
</thead>
<tbody>
<tr>
<td>for each state s from 1 to N do</td>
</tr>
<tr>
<td>viterbi[s,1]← a_{0,s} * b_s(o_1) ; initialization step</td>
</tr>
<tr>
<td>backpointer[s,1]← 0</td>
</tr>
<tr>
<td>for each time step t from 2 to T do</td>
</tr>
<tr>
<td>for each state s from 1 to N do</td>
</tr>
<tr>
<td>viterbi[s,t]← max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t) ; recursion step</td>
</tr>
<tr>
<td>backpointer[s,t]← argmax_{s'=1}^{N} viterbi[s',t-1] * a_{s',s}</td>
</tr>
<tr>
<td>viterbi[q_F,T]← max_{s=1}^{N} viterbi[s,T] * a_s,q_F ; termination step</td>
</tr>
<tr>
<td>backpointer[q_F,T]← argmax_{s=1}^{N} viterbi[s,T] * a_s,q_F ; termination step</td>
</tr>
<tr>
<td>return the backtrace path by following backpointers to states back in time from backpointer[q_F,T]</td>
</tr>
</tbody>
</table>

**Figure 9.8** 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 that assigns maximum likelihood to the observation sequence. Note that states 0 and $q_F$ are non-emitting.

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 $t$, we compute the value $viterbi[s,t]$ by taking the maximum over the extensions of all the paths that lead to the current cell, using the following equation:

$$ v_t(j) = \max_{i=1}^{N} v_{t-1}(i) a_{ij} b_j(o_t) $$  \hfill (9.16)

Recall from Chapter 7 that the three factors that are multiplied in Eq. 9.16 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 $j$

In Fig. 9.9, each cell of the trellis in the column for the word **Janet** 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 **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. Next, each cell in the column gets updated with the maximum probability path from the previous column. 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 transition probability, and the (trivial) max is taken. In this case the final value, 0.000002772, comes from the NNP state at the previous column. The reader should fill in the rest of the trellis in Fig. 9.9 and backtrace to reconstruct the correct state sequence NNP MD VB DT NN. (Exercise 9.??).
9.4.4 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:

$$P(t_n) \approx \prod_{i=1}^{n} P(t_i|t_{i-1})$$  \hspace{1cm} (9.17)

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

$$P(t_n) \approx \prod_{i=1}^{n} P(t_i|t_{i-1},t_{i-2})$$  \hspace{1cm} (9.18)

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, state-of-the-art HMM taggers like \( ? \) have a number of other advanced features. Once 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:

\[
\hat{t}_i^n = \arg\max_{t_i^n} P(t_i^n | w_1^n) \approx \arg\max_{t_i^n} \left[ \prod_{i=1}^{n} P(w_i | t_i) P(t_i | t_{i-1}, t_{i-2}) \right] P(t_{n+1} | t_n) \tag{9.19}
\]

In tagging any sentence with Eq. 9.19, 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. 9.19 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. 9.20:

\[
P(t_i | t_{i-1}, t_{i-2}) = \frac{C(t_{i-2}, t_{i-1}, t_i)}{C(t_{i-2}, t_{i-1})} \tag{9.20}
\]

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 \( P(\text{TO}|\text{PRP,VB}) \) from this frequency, we still could rely on the bigram probability \( P(\text{TO}|\text{VB}), \) or even the unigram probability \( P(\text{TO}). \) The maximum likelihood estimation of each of these probabilities can be computed from a corpus with the following counts:

\begin{align*}
\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{9.21} \\
\text{Bigrams} \quad \hat{P}(t_i | t_{i-1}) &= \frac{C(t_{i-1}, t_i)}{C(t_{i-1})} \tag{9.22} \\
\text{Unigrams} \quad \hat{P}(t_i) &= \frac{C(t_i)}{N} \tag{9.23}
\end{align*}

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:

\[
P(t_i | t_{i-1}, t_{i-2}) = \lambda_3 \hat{P}(t_i | t_{i-1}, t_{i-2}) + \lambda_2 \hat{P}(t_i | t_{i-1}) + \lambda_1 \hat{P}(t_i) \tag{9.24}
\]

We require \( \lambda_1 + \lambda_2 + \lambda_3 = 1 \), ensuring that the resulting \( P \) is a probability distribution. These \( \lambda \)s are generally set by an algorithm called deleted interpolation (?): we successively delete each trigram from the training corpus and choose the \( \lambda \)s so
as to maximize the likelihood of the rest of the corpus. The deletion helps to set the \( \lambda \)s in such a way as to generalize to unseen data and not overfit the training corpus. Figure 9.10 gives a deleted interpolation algorithm for tag trigrams.

```
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 \( C(t_1, t_2, t_3) > 0 \)
  depending on the maximum of the following three values
  case \( \frac{C(t_1, t_2, t_3)}{C(t_1, t_2)} - 1 \): increment \( \lambda_3 \) by \( C(t_1, t_2, t_3) \)
  case \( \frac{C(t_2, t_3)}{C(t_2)} - 1 \): increment \( \lambda_2 \) by \( C(t_1, t_2, t_3) \)
  case \( \frac{C(t_1)}{N} - 1 \): 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 9.10 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 total number of tokens in the corpus. After ? (?).

### 9.4.5 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 wordshape: 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 tend to be adjectives (JJ), and so on. One way to take advantage of this is to store for each final letter sequence (for simplicity referred to as word suffixes) the statistics of which tag they were associated with in training. The method of ? (?) and ? (?), for example, considers suffixes of up to ten letters, computing for each suffix of length \( i \) the probability of the tag \( t_i \) given the suffix letters:

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

(9.25)

They use back-off to smooth these probabilities with successively shorter and shorter suffixes. To capture the fact that unknown words are unlikely to be closed-class words like prepositions, we can compute suffix probabilities only from the
training set for words whose frequency in the training set is \( \leq 10 \), or alternately can train suffix probabilities only on open-class words. To take advantage, separate suffix tries are kept for capitalized and uncapsulated words.

Finally, because Eq. 9.25 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 \( P(t_i|l_{n-1} \ldots l_n) \)).

In addition to using capitalization information for unknown words, 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. 9.21, 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 uncapsulated version of each tag, essentially doubling the size of the tagset.

Combining all these features, a state-of-the-art trigram HMM like that of also has a tagging accuracy of 96.7% on the Penn Treebank.

### 9.5 Maximum Entropy Markov Models

We turn now to a second sequence model, the **maximum entropy Markov model** or **MEMM**. The MEMM is a sequence model adaptation of the MaxEnt (multinomial logistic regression) classifier. Because it is based on logistic regression, the MEMM is a **discriminative sequence model**. By contrast, the HMM is a **generative sequence model**.

In an HMM to compute the best tag sequence that maximizes \( P(T|W) \) we rely on Bayes’ rule and the likelihood \( P(W|T) \):

\[
\hat{T} = \arg\max_T P(T|W) = \arg\max_T P(W|T)P(T) = \arg\max_T \prod_i P(\text{word}_i|\text{tag}_i) \prod_i P(\text{tag}_i|\text{tag}_{i-1}) \tag{9.26}
\]

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

\[
\hat{T} = \arg\max_T P(T|W) = \arg\max_T \prod_i P(t_i|w_i, t_{i-1}) \tag{9.27}
\]

We could do this by training a logistic regression classifier to compute the single probability \( P(t_i|w_i, t_{i-1}) \). Fig. 9.11 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).

#### 9.5.1 Features in a MEMM

Oops. We lied in Eq. 9.27. We actually don’t build MEMMs that condition just on \( w_i \) and \( t_{i-1} \). In fact, an MEMM conditioned on just these two features (the observed
word and the previous tag), as shown in Fig. 9.11 and Eq. 9.27 is no more accurate than the generative HMM model and in fact may be less accurate.

The reason to use a discriminative sequence model is that discriminative models make it easier to incorporate a much wider variety of features. Because in HMMs all computation is based on the two probabilities $P(\text{tag}|\text{tag})$ and $P(\text{word}|\text{tag})$, 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. We saw in the previous section that it was possible to model capitalization or word endings by cleverly fitting in probabilities like $P(\text{capitalization}|\text{tag})$, $P(\text{suffix}|\text{tag})$, and so on into an HMM-style model. But 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 and, as we’ll see, there are lots more features we can add. Figure 9.12 shows a graphical intuition of some of these additional 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:

$$\langle t_i, w_{i-2} \rangle, \langle t_i, w_{i-1} \rangle, \langle t_i, w_i \rangle, \langle t_i, w_{i+1} \rangle, \langle t_i, w_{i+2} \rangle$$

$$\langle t_i, t_{i-1} \rangle, \langle t_i, t_{i-2} \rangle, \langle t_{i-1}, t_{i-2} \rangle,$$

$$\langle t_i, t_{i+1} \rangle, \langle t_i, w_{i+1} \rangle, \langle t_i, w_i, w_{i+1} \rangle.$$  \hspace{1cm} (9.28)

Recall from Chapter 6 that feature templates are used to automatically populate the set of features from every instance in the training and test set. Thus our example $\text{Janet/NNP will/MD back/VB the/DT bill/NN}$, when $w_i$ is the word $\text{back}$, would generate the following features:
\( t_i = \text{VB} \) and \( w_{i-2} = \text{Janet} \)
\( t_i = \text{VB} \) and \( w_{i-1} = \text{will} \)
\( t_i = \text{VB} \) and \( w_i = \text{back} \)
\( t_i = \text{VB} \) and \( w_{i+1} = \text{the} \)
\( t_i = \text{VB} \) and \( w_{i+2} = \text{bill} \)
\( t_i = \text{VB} \) and \( t_{i-1} = \text{MD} \)
\( t_i = \text{VB} \) and \( t_{i-1} = \text{MD} \) and \( t_{i-2} = \text{NNP} \)
\( t_i = \text{VB} \) and \( w_i = \text{back} \) and \( w_{i+1} = \text{the} \)

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 \( \leq 4 \))
- \( w_i \) contains a particular suffix (from all suffixes of length \( \leq 4 \))
- \( w_i \) 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 \( \text{CFC-12} \))
- \( w_i \) is upper case and followed within 3 words by Co., Inc., etc.

**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 \( \text{X.X.X.} \) and DC10-30 would map to \( \text{XXdd-dd.} \). 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 \( \text{Xd-d} \) but I.M.F would still map to \( \text{X.X.X.} \). For example the word \textit{well-dressed} would generate the following non-zero valued feature values:

\[
\begin{align*}
\text{prefix}(w_i) &= \text{w} \\
\text{prefix}(w_i) &= \text{we} \\
\text{prefix}(w_i) &= \text{wel} \\
\text{prefix}(w_i) &= \text{well} \\
\text{suffix}(w_i) &= \text{ssed} \\
\text{suffix}(w_i) &= \text{sed} \\
\text{suffix}(w_i) &= \text{ed} \\
\text{suffix}(w_i) &= \text{d} \\
\text{has-hyphen}(w_i) &= \text{x-x-x} \\
\text{word-shape}(w_i) &= \text{x-x-x-x-x-x-x} \\
\text{short-word-shape}(w_i) &= \text{x-x}
\end{align*}
\]

Feature for known words, like the templates in Eq. 9.28, 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 rare 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.

Given this large set of features, the most likely sequence of tags is then computed by a MaxEnt model that combines these features of the input word \( w_i \), its neighbors within \( l \) words \( w_{i-1}^{i+l} \), and the previous \( k \) tags \( t_{i-k}^{i-1} \) as follows:
\[ T = \arg\max_T P(T|W) \]
\[ = \arg\max_T \prod_i P(t_i|w_i^{i-1}, t_{i-1}^{i-1}) \]
\[ = \arg\max_T \prod_i \exp \left( \sum_{i'} w_{i'} f_i (t_i, w_i^{i-1}, t_{i-1}^{i-1}) \right) \]
\[ \sum_{i' \in \text{tagset}} \exp \left( \sum_{i} w_i f_i (t_i', w_i^{i-1}, t_{i-1}^{i-1}) \right) \]  
(9.29)

### 9.5.2 Decoding and Training MEMMs

We’re now ready to see how to use the MaxEnt classifier to solve the decoding problem by finding the most likely sequence of tags described in Eq. 9.29.

The simplest way to turn the MaxEnt classifier into a sequence model is to build a local classifier that classifies each word left to right, making a hard classification of the first word in the sentence, then a hard decision on the 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. 9.13.

```plaintext
function GREDY MEMM DECODING(words W, model P) returns tag sequence T
for i = 1 to length(W)
  \( \hat{t}_i = \arg\max_{t_i} P(t_i | w_i^{i-1}, t_{i-1}^{i-1}) \)
```

Figure 9.13 In greedy decoding we make a hard decision to choose the best tag left to right.

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 cannot temper its decision with information from future decisions. Although greedy algorithm is very fast, and we do use it in some applications when it has sufficient accuracy, in general this hard decision causes sufficient drop in performance that we don’t use it.

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

Let’s see an example. For pedagogical purposes, let’s assume for this example 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. ?? that the recursive step of the Viterbi equation computes the Viterbi value of time \( t \) for state \( j \) as

\[ v_t(j) = \max_{i=1}^{N} v_{t-1}(i) a_{ij} b_j(o_t); \quad 1 \leq j \leq N, 1 < t \leq T \]  
(9.30)

which is the HMM implementation of
\[ v_t(j) = \max_{i=1}^N \, v_{t-1}(i) \, P(s_j|s_i) \, P(o_t|s_j) \quad 1 \leq j \leq N, \, 1 < t \leq T \quad (9.31) \]

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

\[ v_t(j) = \max_{i=1}^N \, v_{t-1}(i) \, P(s_j|s_i, o_t) \quad 1 \leq j \leq N, \, 1 < t \leq T \quad (9.32) \]

Figure 9.14 shows an example of the Viterbi trellis for an MEMM applied to the ice-cream task from Section ???. Recall that the task is figuring out the hidden weather (hot or cold) from observed numbers of ice creams eaten in Jason Eisner’s diary. Figure 9.14 shows the abstract Viterbi probability calculation, assuming that we have a MaxEnt model that computes \( P(s_j|s_{i-1}, o_i) \) for us.

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 train the weights so as maximize the log-likelihood of the training corpus. As with logistic regression, regularization is important, and all modern systems use L1 or L2 regularization.

### 9.6 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 deci-
sions 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 $t_{i+2}$.

Adding bidirectionality has another useful advantage. MEMMs have a theoretical
weakness, referred to alternatively as the label bias or observation bias problem
(?, ?). These are names for situations when one source of information is ignored be-
cause it is explained away by another source. Consider an example from (?), 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 $P(t_{\text{will}}|s)$ prefers the modal, and because $P(\text{TO}|\text{to}, t_{\text{will}})$
is so close to 1 regardless of $t_{\text{will}}$ the model cannot make use of the transition prob-
ability 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 im-
portance of the previous NN tag for predicting TO. Bidirectionality helps the model
by making the link between TO available when tagging the NN.

One way to implement bidirectionality is to switch to a much more powerful
model called a Conditional Random Field or CRF, which we will introduce in
Chapter 18. But CRFs are much more expensive computationally than MEMMs and
don’t work any better for tagging, and so are not generally used for this task.

Instead, other ways are generally used to add bidirectionality. The Stanford tag-
ger uses a bidirectional version of the MEMM called a cyclic dependency network
(?).

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 tag 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). Multiple-pass
decoding is available in publicly available toolkits like the SVMTool system (?), a
tagger that applies an SVM classifier instead of a MaxEnt classifier at each position,
but similarly using Viterbi (or greedy) decoding to implement a sequence model.

9.7 Part-of-Speech Tagging for Other Languages

The HMM and MEMM speech tagging algorithms have been applied to tagging in
many languages besides English. For languages similar to English, the methods
work well as is; tagger accuracies for German, for example, are close to those for
English. Augmentations become necessary when dealing with highly inflected or
agglutinative 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 (?), while a 10 million word to-
token corpus of Turkish contains four times as many word types as a similarly sized
English corpus (?). Large vocabularies mean many unknown words, and these un-
known words cause significant performance degradations in a wide variety of lan-
guages (including Czech, Slovene, Estonian, and Romanian) (?).

Highly inflectional languages also have much more information than English
coded in word morphology, like case (nominative, accusative, genitive) or gender
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 ızin has three possible morphological/part-of-speech tags and meanings (?):

1. Yerdeki ızin temizlenmesi gerek.  
   iz + Noun+A3sg+Pnon+Gen  
   The trace on the floor should be cleaned.

2. Üzerinde parmak ızin kalmış  
   iz + Noun+A3sg+P2sg+Nom  
   Your fingerprint is left on (it).

3. İçeri girmek için ızin alınan 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, 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 (using a method from Chapter 3, or an extensive dictionary) 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.

Different problems occur with languages like Chinese in which words are not segmented in the writing system. For Chinese part-of-speech tagging word segmentation (Chapter 2) is therefore generally applied before tagging. It is also possible to build sequence models that do joint segmentation and tagging. 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, although 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. One standard unknown feature for Chinese is to build a dictionary in which each character is listed with a vector of each part-of-speech tags that it occurred with in any word in the training set. The vectors of each of the characters in a word are then used as a feature in classification (?).

9.8 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 that are often highly frequent, generally act as function words, and can be 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.
Two common approaches to sequence modeling are a generative approach, HMM tagging, and a discriminative approach, MEMM tagging.

The probabilities in HMM taggers are estimated, not using EM, but directly by maximum likelihood estimation on hand-labeled training corpora. The Viterbi algorithm is used to find the most likely tag sequence.

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 for the sentence.

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 (?), although earlier systems had used part-of-speech information in dictionaries. TDAP used 14 hand-written 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, whose implementation essentially corresponded a cascade of finite-state transducers, was reimplemented (?; ?).

The Computational Grammar Coder (CGC) of ? (?) 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 CGC algorithm reported 90% accuracy on applying a 30-tag tagset to a corpus of articles.

The TAGGIT tagger (?) was based on the ? (?) system, using the same architecture but increasing the size of the dictionary and the size of the tagset to 87 tags. TAGGIT was applied to the Brown corpus and, according to ? (?, 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 list of potential parts-of-speech and in the second stage large lists of hand-written disambiguation rules winnow down this list to a single part of speech for each word.

Soon afterwards the alternative probabilistic architectures began to be developed. Probabilities were used in tagging by ? (?) and a complete probabilistic tagger with Viterbi decoding was sketched by ? (?). The Lancaster-Oslo/Bergen (LOB) corpus, a British English equivalent of the Brown corpus, was tagging in the early 1980’s with the CLAWS tagger (?; ?; ?), a probabilistic algorithm that can be viewed as a simplified approximation to the HMM tagging approach. 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 \( P(\text{tag}|\text{word}) > .10 \).

? (?) developed an algorithm that was almost the HMM approach, including the
use of dynamic programming, although computing a slightly different probability: $P(t|w)P(w)$ instead of $P(w|t)P(w)$. The same year, the probabilistic PARTS tagger of ? (?), (?) was probably the first implemented HMM tagger, described correctly in ? (?), although ? (?) 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 (?; ?; ?). ? (?) showed that fully unsupervised EM didn’t work well for the tagging task and that reliance on hand-labeled data was important. ? (?) showed the importance of the most frequent tag baseline; the 92.3% number we give above was from ? (?). See ? (?) for many implementation details of a state-of-the-art HMM tagger.

? (?) 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 ? (?) system checked all final letter suffixes of lengths 1-5. The probabilistic formulation we described for HMMs comes from ? (?). The unknown word features described on page 18 come mainly from (?), with augmentations from ? (?) and ? (?)

State of the art taggers are based on a number of models developed just after the turn of the last century, including (?) which used the the perceptron algorithm, ? (?) using a bidirectional log-linear model, and (?) using SVMs. HMM (?; ?) and MEMM tagger accuracies are likely just a tad lower.

An alternative modern formalism, the English Constraint Grammar systems (?; ?; ?), uses a two-stage formalism much like the very early taggers from the 1950s and 1960s. A very large morphological analyzer with tens of thousands of English word stems entries is used to return all possible parts-of-speech for a word, using a rich feature-based set of tags. 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 large set of 3,744 constraints are then applied to the input sentence to rule out parts-of-speech that are inconsistent with the context. For example here’s one 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**

**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 combination of the extensive morphological analyzer and carefully written constraints leads to a very high accuracy for the constraint grammar algorithm (?). ? (?) investigates the remaining 2.7% of errors in a state-of-the-art tagger, the bidirectional MEMM-style model described above (?). 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.

The algorithms presented in the chapter rely heavily on in-domain training data
hand-labeled by experts. Much recent work in part-of-speech tagging focuses on ways to relax this assumption. Unsupervised algorithms for part-of-speech tagging cluster words into part-of-speech-like classes (\( \mathcal{C}_1 \), \( \mathcal{C}_2 \), \( \mathcal{C}_3 \), \( \mathcal{C}_4 \)) ; see ? (?) for a summary. Many algorithms focus on combining labeled and unlabeled data, for example by co-training (\( \mathcal{C}_5 \), \( \mathcal{C}_6 \)). Assigning tags to text from very different genres like Twitter text can involve adding new tags for URLs (URL), username mentions (USR), retweets (RT), and hashtags (HT), normalization of non-standard words, and bootstrapping to employ unsupervised data (?).

Readers interested in the history of parts-of-speech should consult a history of linguistics such as ? (?) or ? (?), particularly the article by ? (?) in the latter. ? (?) and ? (?) give a detailed summary of the provenance and makeup of the Brown and other tagsets.

Exercises

9.1 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

9.2 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 gefilte 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.

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

9.4 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.

9.5 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 from this chapter and Chapter 7 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.
9.6 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.


