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 the words *syntax*, *diphthong*, *clitic*, and *analogy*. Also included are a description of eight *parts of speech*: noun, verb, pronoun, preposition, adverb, conjunction, participle, and article. Although earlier scholars (including Aristotle as well as the Stoics) had their own lists of parts of speech, it was Thrax’s set of eight that became the basis for descriptions of European languages for the next 2000 years. (All the way to the *Schoolhouse Rock* educational television shows of our childhood, which had songs about 8 parts of speech, like the late great Bob Dorough’s *Conjunction Junction*.) The durability of parts of speech through two millennia speaks to their centrality in models of human language.

Proper names are another important and anciently studied linguistic category. While parts of speech are generally assigned to individual words or morphemes, a proper name is often an entire multiword phrase, like the name “Marie Curie”, the location “New York City”, or the organization “Stanford University”. We’ll use the term *named entity* for, roughly speaking, anything that can be referred to with a proper name: a person, a location, an organization, although as we’ll see the term is commonly extended to include things that aren’t entities per se.

Parts of speech (also known as *POS*) and named entities are useful clues to sentence structure and meaning. Knowing whether a word is a noun or a verb tells us about likely neighboring words (nouns in English are preceded by determiners and adjectives, verbs by nouns) and syntactic structure (verbs have dependency links to nouns), making part-of-speech tagging a key aspect of parsing. Knowing if a named entity like *Washington* is a name of a person, a place, or a university is important to many natural language processing tasks like question answering, stance detection, or information extraction.

In this chapter we’ll introduce the task of *part-of-speech tagging*, taking a sequence of words and assigning each word a part of speech like *noun* or *verb*, and the task of *named entity recognition* (NER), assigning words or phrases tags like *PERSON*, *LOCATION*, or *ORGANIZATION*.

Such tasks in which we assign, to each word \( x_i \) in an input word sequence, a label \( y_i \), so that the output sequence \( Y \) has the same length as the input sequence \( X \) are called *sequence labeling* tasks. We’ll introduce classic sequence labeling algorithms, one generative—the Hidden Markov Model (HMM)—and one discriminative—the Conditional Random Field (CRF). In following chapters we’ll introduce modern sequence labelers based on RNNs and Transformers.
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 more complete definitions. While word classes do have semantic tendencies—adjectives, for example, often describe properties and nouns **people**—parts of speech are defined instead based on their grammatical relationship with neighboring words or the morphological properties about their affixes.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADJ</td>
<td>Adjective: noun modifiers describing properties</td>
<td>red, young, awesome</td>
</tr>
<tr>
<td>ADV</td>
<td>Adverb: verb modifiers of time, place, manner</td>
<td>very, slowly, home, yesterday</td>
</tr>
<tr>
<td>NOUN</td>
<td>words for persons, places, things, etc.</td>
<td>algorithm, cat, mango, beauty</td>
</tr>
<tr>
<td>VERB</td>
<td>words for actions and processes</td>
<td>draw, provide, go</td>
</tr>
<tr>
<td>PROPN</td>
<td>Proper noun: name of a person, organization, place, etc.</td>
<td>Regina, IBM, Colorado</td>
</tr>
<tr>
<td>INTJ</td>
<td>Interjection: exclamation, greeting, yes/no response, etc.</td>
<td>oh, um, yes, hello</td>
</tr>
<tr>
<td>ADP</td>
<td>Adposition (Preposition/Postposition): marks a noun’s spacial, temporal, or other relation</td>
<td>in, on, by, under</td>
</tr>
<tr>
<td>AUX</td>
<td>Auxiliary: helping verb marking tense, aspect, mood, etc.</td>
<td>can, may, should, are</td>
</tr>
<tr>
<td>CCONJ</td>
<td>Coordinating Conjunction: joins two phrases/clauses</td>
<td>and, or, but</td>
</tr>
<tr>
<td>DET</td>
<td>Determiner: marks noun phrase properties</td>
<td>a, an, the, this</td>
</tr>
<tr>
<td>NUM</td>
<td>Numeral</td>
<td>one, two, first, second</td>
</tr>
<tr>
<td>PART</td>
<td>Particle: a preposition-like form used together with a verb</td>
<td>up, down, on, off, in, out, at, by</td>
</tr>
<tr>
<td>PRON</td>
<td>Pronoun: a shorthand for referring to an entity or event</td>
<td>she, who, I, others</td>
</tr>
<tr>
<td>SCONJ</td>
<td>Subordinating Conjunction: joins a main clause with a subordinate clause such as a sentential complement</td>
<td>that, which</td>
</tr>
<tr>
<td>PUNCT</td>
<td>Punctuation</td>
<td>, , ()</td>
</tr>
<tr>
<td>SYM</td>
<td>Symbols like $ or emoji</td>
<td>$, %</td>
</tr>
<tr>
<td>X</td>
<td>Other</td>
<td>asdf, qwfg</td>
</tr>
</tbody>
</table>

**Figure 8.1** The 17 parts of speech in the Universal Dependencies tagset (Nivre et al., 2016a). Features can be added to make finer-grained distinctions (with properties like number, case, definiteness, and so on).

Parts of speech fall into two broad categories: **closed class** and **open class**. 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. 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** (including proper nouns), **verbs**, **adjectives**, and **adverbs**, as well as the smaller open class of **interjections**. English has all five, although not every language does.

**Nouns** are words for people, places, or things, but include others as well. Common nouns include concrete terms like cat and mango, abstractions like algorithm and beauty, and verb-like terms like pacing as in *His pacing to and fro became quite annoying*. Nouns in English can occur with determiners (a goat, this bandwidth) take possessives (*IBM’s annual revenue*), and may occur in the plural (goats, abaci).

Many languages, including English, divide common nouns into **count nouns** and **mass nouns**. Count nouns can occur in the singular and plural (goat/goats, relationship/relationships) and can be counted (one goat, two goats). Mass nouns are used when something is conceptualized as a homogeneous group. So snow, salt, and communism are not counted (i.e., *two snows* or *two communisms*). **Proper nouns**, like Regina, Colorado, and IBM, are names of specific persons or entities.
Verbs refer to actions and processes, including main verbs like draw, provide, and go. English verbs have inflections (non-third-person-singular (eat), third-person-singular (eats), progressive (eating), past participle (eaten)). While many scholars 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).

Adjectives often describe properties or qualities of nouns, like 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”.

Adverbs are a hodge-podge. All the italicized words in this example are adverbs:

Actually, I ran home extremely quickly yesterday.

Adverbs generally modify 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).

Interjections (oh, hey, alas, uh, um) are a smaller open class that also includes greetings (hello, goodbye) and question responses (yes, no, uh-huh).

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. A verb and a particle acting as a single unit is called a phrasal verb. The meaning of phrasal verbs is often non-compositional—not predictable from the individual meanings of the verb and the particle. Thus, turn down means ‘reject’, rule out ‘eliminate’, and go on ‘continue’.

Determiners like this and that (this chapter, that page) can mark the start of an English noun phrase. Articles like a, an, and the, are a type of determiner that mark discourse properties of the noun and are quite frequent; the is the most common word in written English, with a and an 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, the subordinating conjunction that in “I thought that you might like some milk” 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.

Pronouns act as a shorthand for referring to an 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 act as complementizers (*Frida, who married Diego…*).

**Auxiliary** verbs mark semantic features of a main verb such as its 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*, forms, as well as modal verbs 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.

An English-specific tagset, the 45-tag Penn Treebank tagset (Marcus et al., 1993), shown in Fig. 8.2, has been used to label many syntactically annotated corpora like the Penn Treebank corpora, so is worth knowing about.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
<th>Example</th>
<th>Tag</th>
<th>Description</th>
<th>Example</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC</td>
<td>coord. conj.</td>
<td><em>and, but, or</em></td>
<td>NNP</td>
<td>proper noun, sing.</td>
<td><em>IBM</em></td>
<td>TO</td>
</tr>
<tr>
<td>CD</td>
<td>cardinal number</td>
<td><em>one, two</em></td>
<td>NNPS</td>
<td>proper noun, plu.</td>
<td><em>Carolinans</em></td>
<td>UH</td>
</tr>
<tr>
<td>DT</td>
<td>determiner</td>
<td><em>a, the</em></td>
<td>NNS</td>
<td>noun, plural</td>
<td><em>llamas</em></td>
<td>VB</td>
</tr>
<tr>
<td>EX</td>
<td>existential ‘there’</td>
<td><em>there</em></td>
<td>PDT</td>
<td>predeterminer</td>
<td><em>all, both</em></td>
<td>VBD</td>
</tr>
<tr>
<td>FW</td>
<td>foreign word</td>
<td><em>mea culpa</em></td>
<td>POS</td>
<td>possessive ending</td>
<td><em>’s</em></td>
<td>VBG</td>
</tr>
<tr>
<td>IN</td>
<td>preposition/</td>
<td><em>of, in, by</em></td>
<td>PRP</td>
<td>personal pronoun</td>
<td><em>I, you, he</em></td>
<td>VBN</td>
</tr>
<tr>
<td>JJ</td>
<td>adjective</td>
<td><em>yellow</em></td>
<td>PRPS</td>
<td>possess. pronoun</td>
<td><em>your, one’s</em></td>
<td>VBP</td>
</tr>
<tr>
<td>JJR</td>
<td>comparative adj</td>
<td><em>bigger</em></td>
<td>RB</td>
<td>adverb</td>
<td><em>quickly</em></td>
<td>VBZ</td>
</tr>
<tr>
<td>JJS</td>
<td>superlative adj</td>
<td><em>wildest</em></td>
<td>RBR</td>
<td>comparative adv</td>
<td><em>faster</em></td>
<td>WDT</td>
</tr>
<tr>
<td>LS</td>
<td>list item marker</td>
<td><em>1, 2, One</em></td>
<td>RBS</td>
<td>superlat. adv</td>
<td><em>fastest</em></td>
<td>WP</td>
</tr>
<tr>
<td>MD</td>
<td>modal</td>
<td><em>can, should</em></td>
<td>RP</td>
<td>particle</td>
<td><em>up, off</em></td>
<td>WPS</td>
</tr>
<tr>
<td>NN</td>
<td>sing or mass noun</td>
<td><em>llama</em></td>
<td>SYM</td>
<td>symbol</td>
<td>+,%,&amp;</td>
<td>WRB</td>
</tr>
</tbody>
</table>

**Figure 8.2** Penn Treebank part-of-speech tags.

Below we show some examples with each word tagged according to both the UD and Penn tags. Notice that the Penn tagset distinguishes tense and participles on verbs, and has a special tag for the existential *there* construction in English. Note that since *New England Journal of Medicine* is a proper noun, both tagsets mark its component nouns as NNP, including *journal* and *medicine*, which might otherwise be labeled as common nouns (NOUN/NN).

(8.1) *There/*PRO/EX are/VERB/VBP 70/NUM/CD children/NOUN/NNS there/ADV/RB /PUNC/.

(8.2) *Preliminary/ADJ/1 results/NOUN/NNS were/AUX/VBD reported/VERB/VBN in/ADP/IN today/NOUN/NN ’s/PART/POS New/PROPN/NNP England/PROPN/NP Journal/PROPN/NNP of/ADP/IN Medicine/PROPN/NNP

### 8.2 Part-of-Speech Tagging

**Part-of-speech tagging** is the process of assigning a part-of-speech to each word in a text. The input is a sequence \(x_1, x_2, \ldots, x_n\) of (tokenized) words and a tagset, and the output is a sequence \(y_1, y_2, \ldots, y_n\) of tags, each output \(y_i\) corresponding exactly to one input \(x_i\), as shown in the intuition in Fig. 8.3.

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.

The accuracy of part-of-speech tagging algorithms (the percentage of test set tags that match human gold labels) is extremely high. One study found accuracies over 97% across 15 languages from the Universal Dependency (UD) treebank (Wu and Dredze, 2019). Accuracies on various English treebanks are also 97% (no matter the algorithm; HMMs, CRFs, BERT perform similarly). This 97% number is also about the human performance on this task, at least for English (Manning, 2011).

<table>
<thead>
<tr>
<th>Types:</th>
<th>WSJ</th>
<th>Brown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unambiguous (1 tag)</td>
<td>44,432 (86%)</td>
<td>45,799 (85%)</td>
</tr>
<tr>
<td>Ambiguous (2+ tags)</td>
<td>7,025 (14%)</td>
<td>8,050 (15%)</td>
</tr>
<tr>
<td>Tokens:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unambiguous (1 tag)</td>
<td>577,421 (45%)</td>
<td>384,349 (33%)</td>
</tr>
<tr>
<td>Ambiguous (2+ tags)</td>
<td>711,780 (55%)</td>
<td>786,646 (67%)</td>
</tr>
</tbody>
</table>

We’ll introduce algorithms for the task in the next few sections, but first let’s explore the task. Exactly how hard is it? Fig. 8.4 shows that most word types (85-86%) are unambiguous (Janet is always NNP, hesitantly is always RB). But the ambiguous words, though accounting for only 14-15% of the vocabulary, are very common, and 55-67% of word tokens in running text are ambiguous. Particularly ambiguous common words include 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 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 useful baseline: 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).
The most-frequent-tag baseline has an accuracy of about 92%\(^1\). The baseline thus differs from the state-of-the-art and human ceiling (97%) by only 5%.

### 8.3 Named Entities and Named Entity Tagging

Part of speech tagging can tell us that words like *Janet, Stanford University*, and *Colorado* are all proper nouns; being a proper noun is a grammatical property of these words. But viewed from a semantic perspective, these proper nouns refer to different kinds of entities: Janet is a person, Stanford University is an organization, and Colorado is a location.

A **named entity** is, roughly speaking, anything that can be referred to with a proper name: a person, a location, an organization. The task of **named entity recognition (NER)** is to find spans of text that constitute proper names and tag the type of the entity. Four entity tags are most common: **PER** (person), **LOC** (location), **ORG** (organization), or **GPE** (geo-political entity). However, the term **named entity** is commonly extended to include things that aren’t entities per se, including dates, times, and other kinds of temporal expressions, and even numerical expressions like prices. Here’s an example of the output of an NER tagger:

Citing high fuel prices, [**ORG United Airlines**] said [**TIME Friday**] it has increased fares by [**MONEY $6**] per round trip on flights to some cities also served by lower-cost carriers. [**ORG American Airlines**], a unit of [**ORG AMR Corp.**], immediately matched the move, spokesman [**PER Tim Wagner**] said. [**ORG United**], a unit of [**ORG UAL Corp.**], said the increase took effect [**TIME Thursday**] and applies to most routes where it competes against discount carriers, such as [**LOC Chicago**] to [**LOC Dallas**] and [**LOC Denver**] to [**LOC San Francisco**].

The text contains 13 mentions of named entities including 5 organizations, 4 locations, 2 times, 1 person, and 1 mention of money. Figure 8.5 shows typical generic named entity types. Many applications will also need to use specific entity types like proteins, genes, commercial products, or works of art.

<table>
<thead>
<tr>
<th>Type</th>
<th>Tag</th>
<th>Sample Categories</th>
<th>Example sentences</th>
</tr>
</thead>
<tbody>
<tr>
<td>People</td>
<td>PER</td>
<td>people, characters</td>
<td><em>Turing</em> is a giant of computer science.</td>
</tr>
<tr>
<td>Organization</td>
<td>ORG</td>
<td>companies, sports teams</td>
<td>The <em>IPCC</em> warned about the cyclone.</td>
</tr>
<tr>
<td>Location</td>
<td>LOC</td>
<td>regions, mountains, seas</td>
<td><em>Mt. Sanitas</em> is in <em>Sunshine Canyon</em>.</td>
</tr>
<tr>
<td>Geo-Political Entity</td>
<td>GPE</td>
<td>countries, states</td>
<td><em>Palo Alto</em> is raising the fees for parking.</td>
</tr>
</tbody>
</table>

**Figure 8.5** A list of generic named entity types with the kinds of entities they refer to.

Named entity tagging is a useful first step in lots of natural language processing tasks. In sentiment analysis we might want to know a consumer’s sentiment toward a particular entity. Entities are a useful first stage in question answering, or for linking text to information in structured knowledge sources like Wikipedia. And named entity tagging is also central to tasks involving building semantic representations, like extracting events and the relationship between participants.

Unlike part-of-speech tagging, where there is no segmentation problem since each word gets one tag, the task of named entity recognition is to find and label **spans** of text, and is difficult partly because of the ambiguity of segmentation; we

\(^1\) In English, on the WSJ corpus, tested on sections 22-24.
need to decide what’s an entity and what isn’t, and where the boundaries are. Indeed, most words in a text will not be named entities. Another difficulty is caused by type ambiguity. The mention JFK can refer to a person, the airport in New York, or any number of schools, bridges, and streets around the United States. Some examples of this kind of cross-type confusion are given in Figure 8.6.

<table>
<thead>
<tr>
<th>Words</th>
<th>IO Label</th>
<th>BIO Label</th>
<th>BIOES Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane Villanueva</td>
<td>I-PER</td>
<td>B-PER</td>
<td>B-PER</td>
</tr>
<tr>
<td>of</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>United Airlines</td>
<td>I-ORG</td>
<td>B-ORG</td>
<td>B-ORG</td>
</tr>
<tr>
<td>Holding</td>
<td>I-ORG</td>
<td>I-ORG</td>
<td>E-ORG</td>
</tr>
<tr>
<td>discussed the</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Chicago</td>
<td>I-LOC</td>
<td>B-LOC</td>
<td>S-LOC</td>
</tr>
<tr>
<td>route</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>.</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
</tbody>
</table>

Figure 8.7 NER as a sequence model, showing IO, BIO, and BIOES taggings.

The standard approach to sequence labeling for a span-recognition problem like NER is BIO tagging (Ramshaw and Marcus, 1995). This is a method that allows us to treat NER like a word-by-word sequence labeling task, via tags that capture both the boundary and the named entity type. Consider the following sentence:

Jane Villanueva of United Airlines Holding, a unit of United Airlines Holding, said the fare applies to the Chicago route.

Figure 8.7 shows the same excerpt represented with BIO tagging, as well as variants called IO tagging and BIOES tagging. BIO tagging we label any token that begins a span of interest with the label B, tokens that occur inside a span are tagged with an I, and any tokens outside of any span of interest are labeled O. While there is only one O tag, we’ll have distinct B and I tags for each named entity class. The number of tags is thus $2n + 1$ tags, where $n$ is the number of entity types. BIO tagging can represent exactly the same information as the bracketed notation, but has the advantage that we can represent the task in the same simple sequence modeling way as part-of-speech tagging: assigning a single label $y_i$ to each input word $x_i$:

We’ve also shown two variant tagging schemes: IO tagging, which loses some information by eliminating the B tag, and BIOES tagging, which adds an end tag E for the end of a span, and a span tag S for a span consisting of only one word. A sequence labeler (HMM, CRF, RNN, Transformer, etc.) is trained to label each token in a text with tags that indicate the presence (or absence) of particular kinds of named entities.
8.4 HMM Part-of-Speech Tagging

In this section we introduce our first sequence labeling algorithm, the Hidden Markov Model, and show how to apply it to part-of-speech tagging. Recall that a sequence labeler is a model whose job is to assign a label to each unit in a sequence, thus mapping a sequence of observations to a sequence of labels of the same length. The HMM is a classic model that introduces many of the key concepts of sequence modeling that we will see again in more modern models.

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

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 8.8](image_url)

More formally, consider a sequence of state variables \( q_1, q_2, \ldots, q_t \). 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.

**Markov Assumption:** \( P(q_t = a|q_1 \ldots q_{t-1}) = P(q_t = a|q_{t-1}) \) (8.3)

Figure 8.8a 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 8.8b shows a Markov chain for assigning a probability to a sequence of words \( w_1 \ldots w_t \). This Markov chain should be familiar; in fact, it represents a bigram language model, with each edge expressing the probability \( p(w_t|w_{t-1}) \)!

Given the two models in Fig. 8.8, 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. \( \sum_{j=1}^{N} a_{ij} = 1 \) \( \forall i \)
- \( \pi = \pi_1, \pi_2, \ldots, \pi_N \) an initial probability distribution over states. \( \pi_i \) 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. 8.8a (with \( \pi = [0.1, 0.7, 0.2] \)) to compute the probability of each of the following sequences:

- (8.4) hot hot hot hot
- (8.5) cold hot cold hot

What does the difference in these probabilities tell you about a real-world weather fact encoded in Fig. 8.8a?

### 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.

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
- \( A = a_{11} a_{12} \ldots a_{N1} \ldots a_{NN} \) a transition probability matrix \( A \), each \( a_{ij} \) representing the probability of moving from state \( i \) to state \( j \), s.t. \( \sum_{j=1}^{N} a_{ij} = 1 \) \( \forall i \)
- \( O = o_1 o_2 \ldots o_T \) a sequence of \( T \) observations, each one drawn from a vocabulary \( V = v_1, v_2, \ldots, v_V \)
- \( B = b_i(o_t) \) a sequence of observation likelihoods, also called emission probabilities, each expressing the probability of an observation \( o_t \) being generated from a state \( q_i \)
- \( \pi = \pi_1, \pi_2, \ldots, \pi_N \) an initial probability distribution over states. \( \pi_i \) 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:

**Markov Assumption:** \( P(q_t|q_1, \ldots, q_{t-1}) = P(q_t|q_{t-1}) \) \hspace{1cm} (8.6)

Second, the probability of an output observation \( o_t \) depends only on the state that produced the observation \( q_t \) and not on any other states or any other observations:

**Output Independence:** \( P(o_t|q_1, \ldots, q_t, o_1, \ldots, o_{t-1}) = P(o_t|q_t) \) \hspace{1cm} (8.7)
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:

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})} \quad (8.8)$$

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

$$P(VB|MD) = \frac{C(MD, VB)}{C(MD)} = \frac{10471}{13124} = .80 \quad (8.9)$$

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

$$P(w_i|t_i) = \frac{C(t_{i}, w_i)}{C(t_{i})} \quad (8.10)$$

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

$$P(will|MD) = \frac{C(MD, will)}{C(MD)} = \frac{4046}{13124} = .31 \quad (8.11)$$

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 $P(MD|will)$. 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 $A$ transition probabilities, and $B$ observation likelihoods of the HMM are illustrated in Fig. 8.9 for three states in an HMM part-of-speech tagger; the full tagger would have one state for each tag.

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**: 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 $Q = q_1q_2q_3...q_T$. 
For part-of-speech tagging, the goal of HMM decoding is to choose the tag sequence \( t_1 \ldots t_n \) that is most probable given the observation sequence of \( n \) words \( w_1 \ldots w_n \):

\[
\hat{t}_{1:n} = \arg\max_{t_1 \ldots t_n} P(t_1 \ldots t_n | w_1 \ldots w_n)
\]  

(8.12)

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

\[
\hat{t}_{1:n} = \arg\max_{t_1 \ldots t_n} \frac{P(w_1 \ldots w_n | t_1 \ldots t_n) P(t_1 \ldots t_n)}{P(w_1 \ldots w_n)}
\]  

(8.13)

Furthermore, we simplify Eq. 8.13 by dropping the denominator \( P(w_1 \ldots w_n) \):

\[
\hat{t}_{1:n} = \arg\max_{t_1 \ldots t_n} P(w_1 \ldots w_n | t_1 \ldots t_n) P(t_1 \ldots t_n)
\]  

(8.14)

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:

\[
P(w_1 \ldots w_n | t_1 \ldots t_n) \approx \prod_{i=1}^{n} P(w_i | t_i)
\]  

(8.15)

The second assumption, the \textit{bigram} assumption, is that the probability of a tag is dependent only on the previous tag, rather than the entire tag sequence:

\[
P(t_1 \ldots t_n) \approx \prod_{i=1}^{n} P(t_i | t_{i-1})
\]  

(8.16)

Plugging the simplifying assumptions from Eq. 8.15 and Eq. 8.16 into Eq. 8.14 results in the following equation for the most probable tag sequence from a bigram tagger:

\[
\hat{t}_{1:n} = \arg\max_{t_1 \ldots t_n} P(t_1 \ldots t_n | w_1 \ldots w_n) \approx \arg\max_{t_1 \ldots t_n} \prod_{i=1}^{n} P(w_i | t_i) P(t_i | t_{i-1})
\]  

(8.17)

The two parts of Eq. 8.17 correspond neatly to the \textit{B emission probability} and \textit{A transition probability} that we just defined above!
8.4.5 The Viterbi Algorithm

The decoding algorithm for HMMs is the Viterbi algorithm shown in Fig. 8.10. As an instance of dynamic programming, Viterbi resembles the dynamic programming minimum edit distance algorithm of Chapter 2.

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 8.11 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 $q_1, \ldots, q_{t-1}$, given the HMM $\lambda$. 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

$$v_t(j) = \max_{q_1, \ldots, q_{t-1}} P(q_1 \ldots q_{t-1}, o_1, o_2 \ldots o_t, q_t = j | \lambda) \quad (8.18)$$

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

$$v_t(j) = \max_{i=1}^N v_{t-1}(i) a_{ij} b_j(o_t) \quad (8.19)$$

The three factors that are multiplied in Eq. 8.19 for extending the previous paths to compute the Viterbi probability at time $t$ are

---

**Figure 8.10** 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.
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. 8.11):

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

<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 8.12 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.

Let the HMM be defined by the two tables in Fig. 8.12 and Fig. 8.13. Figure 8.12 lists the $a_{ij}$ probabilities for transitioning between the hidden states (part-of-speech tags). Figure 8.13 expresses the $b_j(o_t)$ 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 8.14 shows a fleshed-out version of the sketch we saw in Fig. 8.11, the Viterbi lattice for computing the best hidden state sequence for the observation sequence *Janet will back the bill*.

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 $\pi$ transition probability (the start probability for that state $i$, which we get from the $<s>$ entry of Fig. 8.12),
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. 8.14.

Next, each cell in the will column gets updated. For each state, we compute the value \( \text{viterbi}[s,t] \) by taking the maximum over the extensions of all the paths from the previous column that lead to the current cell according to Eq. 8.19. 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. 8.14 and backtrace to see whether or not the Viterbi algorithm returns the gold state sequence NNP MD VB DT NN.

8.5 Conditional Random Fields (CRFs)

While the HMM is a useful and powerful model, it turns out that HMMs need a number of augmentations to achieve high accuracy. For example, in POS tagging as in other tasks, we often run into unknown words: proper names and acronyms are created very often, and even new common nouns and verbs enter the language at a surprising rate. It would be great to have ways to add arbitrary features to help with this, perhaps based on capitalization or morphology (words starting with capital letters are likely to be proper nouns, words ending with -ed tend to be past tense (VBD or VBN), etc.) Or knowing the previous or following words might be a useful feature (if the previous word is the, the current tag is unlikely to be a verb).

Although we could try to hack the HMM to find ways to incorporate some of these, in general it’s hard for generative models like HMMs to add arbitrary features directly into the model in a clean way. We’ve already seen a model for combining arbitrary features in a principled way: log-linear models like the logistic regression model of Chapter 5! But logistic regression isn’t a sequence model; it assigns a class to a single observation.

Luckily, there is a discriminative sequence model based on log-linear models: the conditional random field (CRF). We’ll describe here the linear chain CRF, the version of the CRF most commonly used for language processing, and the one whose conditioning closely matches the HMM.

Assuming we have a sequence of input words \( X = x_1...x_n \) and want to compute a sequence of output tags \( Y = y_1...y_n \). In an HMM to compute the best tag sequence that maximizes \( P(Y|X) \) we rely on Bayes’ rule and the likelihood \( P(X|Y) \):

\[
\hat{Y} = \arg\max_Y p(Y|X) \\
= \arg\max_Y p(X|Y)p(Y) \\
= \arg\max_Y \prod_i p(x_i|y_i) \prod_i p(y_i|y_{i-1}) \quad (8.21)
\]

In a CRF, by contrast, we compute the posterior \( p(Y|X) \) directly, training the CRF
to discriminate among the possible tag sequences:

$$\hat{Y} = \text{argmax}_{Y \in \mathcal{Y}} P(Y|X)$$

(8.22)

However, the CRF does not compute a probability for each tag at each time step. Instead, at each time step the CRF computes log-linear functions over a set of relevant features, and these local features are aggregated and normalized to produce a global probability for the whole sequence.

Let’s introduce the CRF more formally, again using $X$ and $Y$ as the input and output sequences. A CRF is a log-linear model that assigns a probability to an entire output (tag) sequence $Y$, out of all possible sequences $\mathcal{Y}$, given the entire input (word) sequence $X$. We can think of a CRF as like a giant version of what multinomial logistic regression does for a single token. Recall that the feature function $f$ in regular multinomial logistic regression can be viewed as a function of a tuple: a token $x$ and a label $y$ (page ??). In a CRF, the function $F$ maps an entire input sequence $X$ and an entire output sequence $Y$ to a feature vector. Let’s assume we have $K$ features, with a weight $w_k$ for each feature $F_k$:

$$p(Y|X) = \frac{\exp \left( \sum_{k=1}^{K} w_k F_k(X,Y) \right)}{\sum_{Y' \in \mathcal{Y}} \exp \left( \sum_{k=1}^{K} w_k F_k(X,Y') \right)}$$

(8.23)

It’s common to also describe the same equation by pulling out the denominator into a function $Z(X)$:

$$p(Y|X) = \frac{1}{Z(X)} \exp \left( \sum_{k=1}^{K} w_k F_k(X,Y) \right)$$

(8.24)

$$Z(X) = \sum_{Y' \in \mathcal{Y}} \exp \left( \sum_{k=1}^{K} w_k F_k(X,Y') \right)$$

(8.25)

We’ll call these $K$ functions $F_k(X,Y)$ **global features**, since each one is a property of the entire input sequence $X$ and output sequence $Y$. We compute them by decomposing into a sum of **local** features for each position $i$ in $Y$:

$$F_k(X,Y) = \sum_{i=1}^{n} f_k(y_{i-1}, y_i, X, i)$$

(8.26)

Each of these local features $f_k$ in a linear-chain CRF is allowed to make use of the current output token $y_i$, the previous output token $y_{i-1}$, the entire input string $X$ (or any subpart of it), and the current position $i$. This constraint to only depend on the current and previous output tokens $y_i$ and $y_{i-1}$ are what characterizes a **linear chain CRF**. As we will see, this limitation makes it possible to use versions of the efficient Viterbi and Forward-Backwards algorithms from the HMM. A general CRF, by contrast, allows a feature to make use of any output token, and are thus necessary for tasks in which the decision depend on distant output tokens, like $y_{i-4}$. General CRFs require more complex inference, and are less commonly used for language processing.
8.5.1 Features in a CRF POS Tagger

Let’s look at some of these features in detail, since the reason to use a discriminative sequence model is that it’s easier to incorporate a lot of features.²

Again, in a linear-chain CRF, each local feature $f_k$ at position $i$ can depend on any information from: $(y_{i-1}, y_i, X, i)$. So some legal features representing common situations might be the following:

\[
\begin{align*}
\mathbb{1}\{x_i = \text{the}, y_i = \text{DET}\} \\
\mathbb{1}\{y_i = \text{PROPN}, x_{i+1} = \text{Street}, y_{i-1} = \text{NUM}\} \\
\mathbb{1}\{y_i = \text{VERB}, y_{i-1} = \text{AUX}\}
\end{align*}
\]

For simplicity, we’ll assume all CRF features take on the value 1 or 0. Above, we explicitly use the notation $\mathbb{1}\{x\}$ to mean “1 if $x$ is true, and 0 otherwise”. From now on, we’ll leave off the $\mathbb{1}$ when we define features, but you can assume each feature has it there implicitly.

Although the idea of what features to use is done by the system designer by hand, the specific features are automatically populated by using feature templates as we briefly mentioned in Chapter 5. Here are some templates that only use information from $(y_{i-1}, y_i, X, i)$:

\[
\begin{align*}
\langle y_i, x_i \rangle, \langle y_i, y_{i-1}\rangle, \langle y_i, x_{i-1}, x_{i+2}\rangle
\end{align*}
\]

These templates automatically populate the set of features from every instance in the training and test set. Thus for our example Janet/NNP will/MD back/VB the/DT bill/NN, when $x_i$ is the word back, the following features would be generated and have the value 1 (we’ve assigned them arbitrary feature numbers):

$\begin{align*}
f_{3743}: y_i = \text{VB} \text{ and } x_i = \text{back} \\
f_{156}: y_i = \text{VB} \text{ and } y_{i-1} = \text{MD} \\
f_{99732}: y_i = \text{VB} \text{ and } x_{i-1} = \text{will} \text{ and } x_{i+2} = \text{bill}
\end{align*}$

It’s also important to have features that help with unknown words. One of the most important is word shape features, which 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 words in all caps map to X, words with initial-caps map to Xx, DC10-30 would be mapped to Xd-d but I.M.F would still map to X.X.X. Prefix and suffix features are also useful. In summary, here are some sample feature templates that help with unknown words:

\[
\begin{align*}
x_i \text{ contains a particular prefix (perhaps from all prefixes of length } \leq 2) \\
x_i \text{ contains a particular suffix (perhaps from all suffixes of length } \leq 2) \\
x_i \text{'s word shape} \\
x_i \text{'s short word shape}
\end{align*}
\]

For example the word well-dressed might generate the following non-zero valued feature values:

² 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. 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.
prefix\( (x_i) = w \)
prefix\( (x_i) = we \)
suffix\( (x_i) = ed \)
suffix\( (x_i) = d \)
word-shape\( (x_i) = xxxx-xxxxxxx \)
short-word-shape\( (x_i) = x-x \)

The known-word templates 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.

Remember that in a CRF we don’t learn weights for each of these local features \( f_k \). Instead, we first sum the values of each local feature (for example feature \( f_{3743} \)) over the entire sentence, to create each global feature (for example \( F_{3743} \)). It is those global features that will then be multiplied by weight \( w_{3743} \). Thus for training and inference there is always a fixed set of \( K \) features with \( K \) weights, even though the length of each sentence is different.

### 8.5.2 Features for CRF Named Entity Recognizers

A CRF for NER makes use of very similar features to a POS tagger, as shown in Figure 8.15.

- **identity of \( w_i \), identity of neighboring words**
- **embeddings for \( w_i \), embeddings for neighboring words**
- **part of speech of \( w_i \), part of speech of neighboring words**
- **presence of \( w_i \) in a gazetteer**
- \( w_i \) contains a particular prefix (from all prefixes of length \( \leq 4 \))
- \( w_i \) contains a particular suffix (from all suffixes of length \( \leq 4 \))
- **word shape of \( w_i \), word shape of neighboring words**
- **short word shape of \( w_i \), short word shape of neighboring words**

**gazetteer**

One feature that is especially useful for locations is a gazetteer, a list of place names, often providing millions of entries for locations with detailed geographical and political information.\(^3\) This can be implemented as a binary feature indicating a phrase appears in the list. Other related resources like name-lists, for example from the United States Census Bureau\(^4\), can be used, as can other entity dictionaries like lists of corporations or products, although they may not be as helpful as a gazetteer (Mikheev et al., 1999).

The sample named entity token *L’Occitane* would generate the following non-zero valued feature values (assuming that *L’Occitane* is neither in the gazetteer nor the census).

---

\(^3\) [www.geonames.org](http://www.geonames.org)

\(^4\) [www.census.gov](http://www.census.gov)
8.5 CONDITIONAL RANDOM FIELDS (CRFs) 19

\[ \text{prefix}(x_i) = L \quad \text{suffix}(x_i) = \text{tane} \]
\[ \text{prefix}(x_i) = L' \quad \text{suffix}(x_i) = \text{ane} \]
\[ \text{prefix}(x_i) = L'O \quad \text{suffix}(x_i) = \text{ane} \]
\[ \text{prefix}(x_i) = L'Oc \quad \text{suffix}(x_i) = \text{e} \]
\[ \text{word-shape}(x_i) = X'Xxxxxxxx \quad \text{short-word-shape}(x_i) = X'Xx \]

Figure 8.16 illustrates the result of adding part-of-speech tags and some shape information to our earlier example.

<table>
<thead>
<tr>
<th>Words</th>
<th>POS</th>
<th>Short shape</th>
<th>Gazetteer</th>
<th>BIO Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jane</td>
<td>NNP</td>
<td>Xx</td>
<td></td>
<td>B-PER</td>
</tr>
<tr>
<td>Villanueva</td>
<td>NNP</td>
<td>Xx</td>
<td></td>
<td>I-PER</td>
</tr>
<tr>
<td>of</td>
<td>IN</td>
<td>x</td>
<td>0</td>
<td>O</td>
</tr>
<tr>
<td>United</td>
<td>NNP</td>
<td>Xx</td>
<td>0</td>
<td>B-ORG</td>
</tr>
<tr>
<td>Airlines</td>
<td>NNP</td>
<td>Xx</td>
<td>0</td>
<td>I-ORG</td>
</tr>
<tr>
<td>Holding</td>
<td>NNP</td>
<td>Xx</td>
<td>0</td>
<td>I-ORG</td>
</tr>
<tr>
<td>discussed</td>
<td>VBD</td>
<td>x</td>
<td>0</td>
<td>O</td>
</tr>
<tr>
<td>the</td>
<td>DT</td>
<td>x</td>
<td>0</td>
<td>O</td>
</tr>
<tr>
<td>Chicago</td>
<td>NNP</td>
<td>Xx</td>
<td>1</td>
<td>B-LOC</td>
</tr>
<tr>
<td>route</td>
<td>NN</td>
<td>x</td>
<td>0</td>
<td>O</td>
</tr>
</tbody>
</table>

Figure 8.16 Some NER features for a sample sentence, assuming that Chicago and Villanueva are listed as locations in a gazetteer. We assume features only take on the values 0 or 1, so the first POS feature, for example, would be represented as \( 0 \{ \text{POS} = \text{NNP} \} \).

8.5.3 Inference and Training for CRFs

How do we find the best tag sequence \( \hat{Y} \) for a given input \( X \)? We start with Eq. 8.22:

\[
\hat{Y} = \arg\max_{\hat{Y} \in Y} P(Y|X) \\
= \arg\max_{\hat{Y} \in Y} \frac{1}{Z(X)} \exp \left( \sum_{k=1}^{K} w_k f_k(X, Y) \right) \\
= \arg\max_{\hat{Y} \in Y} \exp \left( \sum_{k=1}^{K} w_k \sum_{i=1}^{n} f_k(y_{i-1}, y_i, X, i) \right) \\
= \arg\max_{\hat{Y} \in Y} \sum_{k=1}^{K} w_k \sum_{i=1}^{n} f_k(y_{i-1}, y_i, X, i) \\
= \arg\max_{\hat{Y} \in Y} \sum_{i=1}^{n} \sum_{k=1}^{K} w_k f_k(y_{i-1}, y_i, X, i)
\]

(8.27)  (8.28)  (8.29)  (8.30)

We can ignore the \( \exp \) function and the denominator \( Z(X) \), as we do above, because \( \exp \) doesn’t change the argmax, and the denominator \( Z(X) \) is constant for a given observation sequence \( X \).

How should we decode to find this optimal tag sequence \( \hat{y} \)? Just as with HMMs, we’ll turn to the Viterbi algorithm, which works because, like the HMM, the linear-chain CRF depends at each timestep on only one previous output token \( y_{i-1} \).

Concretely, this involves filling an \( N \times T \) array with the appropriate values, 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 HMM Viterbi have to do only with how we fill each cell. Recall from Eq. 8.19 that the recursive step of the Viterbi equation computes the Viterbi value of time $t$ for state $j$ as

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i) a_{ij} b_{j}(o_t); \quad 1 \leq j \leq N, 1 < t \leq T$$ (8.31)

which is the HMM implementation of

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i) P(s_j|s_i) P(o_t|s_j) \quad 1 \leq j \leq N, 1 < t \leq T$$ (8.32)

The CRF requires only a slight change to this latter formula, replacing the $a$ and $b$ prior and likelihood probabilities with the CRF features:

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i) \sum_{k=1}^{K} w_k f_k(y_{t-1}, y_t, X, t) \quad 1 \leq j \leq N, 1 < t \leq T$$ (8.33)

Learning in CRFs relies on the same supervised learning algorithms we presented for logistic regression. Given a sequence of observations, feature functions, and corresponding outputs, we use stochastic gradient descent to train the weights to maximize the log-likelihood of the training corpus. The local nature of linear-chain CRFs means that the forward-backward algorithm introduced for HMMs in Appendix A can be extended to a CRF version that will efficiently compute the necessary derivatives. As with logistic regression, L1 or L2 regularization is important.

### 8.6 Evaluation of Named Entity Recognition

Part-of-speech taggers are evaluated by the standard metric of **accuracy**. Named entity recognizers are evaluated by **recall**, **precision**, and **F$1$ measure**. Recall that recall is the ratio of the number of correctly labeled responses to the total that should have been labeled; precision is the ratio of the number of correctly labeled responses to the total labeled; and $F$-measure is the harmonic mean of the two.

To know if the difference between the $F_1$ scores of two NER systems is a significant difference, we use the paired bootstrap test, or the similar randomization test (Section ??).

For named entity tagging, the *entity* rather than the word is the unit of response. Thus in the example in Fig. 8.16, the two entities *Jane Villanueva* and *United Airlines Holding* and the non-entity *discussed* would each count as a single response.

The fact that named entity tagging has a segmentation component which is not present in tasks like text categorization or part-of-speech tagging causes some problems with evaluation. For example, a system that labeled *Jane* but not *Jane Villanueva* as a person would cause two errors, a false positive for O and a false negative for I-PER. In addition, using entities as the unit of response but words as the unit of training means that there is a mismatch between the training and test conditions.

### 8.7 Further Details

In this section we summarize a few remaining details of the data and models for part-of-speech tagging and NER, beginning with data. Since the algorithms we have
presented are supervised, having labeled data is essential for training and test. A wide variety of datasets exist for part-of-speech tagging and/or NER. The Universal Dependencies (UD) dataset (Nivre et al., 2016b) has POS tagged corpora in 92 languages at the time of this writing, as do the Penn Treebanks in English, Chinese, and Arabic. OntoNotes has corpora labeled for named entities in English, Chinese, and Arabic (Hovy et al., 2006). Named entity tagged corpora also available in particular domains, such as for biomedical (Bada et al., 2012) and literary text (Bamman et al., 2019).

### 8.7.1 Rule-based Methods

While machine learned (neural or CRF) sequence models are the norm in academic research, commercial approaches to NER are often based on pragmatic combinations of lists and rules, with some smaller amount of supervised machine learning (Chiticariu et al., 2013). For example in the IBM System T architecture, a user specifies declarative constraints for tagging tasks in a formal query language that includes regular expressions, dictionaries, semantic constraints, and other operators, which the system compiles into an efficient extractor (Chiticariu et al., 2018).

One common approach is to make repeated rule-based passes over a text, starting with rules with very high precision but low recall, and, in subsequent stages, using machine learning methods that take the output of the first pass into account (an approach first worked out for coreference (Lee et al., 2017)):

1. First, use high-precision rules to tag unambiguous entity mentions.
2. Then, search for substring matches of the previously detected names.
3. Use application-specific name lists to find likely domain-specific mentions.
4. Finally, apply supervised sequence labeling techniques that use tags from previous stages as additional features.

Rule-based methods were also the earliest methods for part-of-speech tagging. Rule-based taggers like the English Constraint Grammar system (Karlsson et al. 1995, Voutilainen 1999) use a two-stage formalism invented in the 1950s and 1960s: (1) a morphological analyzer with tens of thousands of word stem entries returns all parts of speech for a word, then (2) a large set of thousands of constraints are applied to the input sentence to rule out parts of speech inconsistent with the context.

### 8.7.2 POS Tagging for Morphologically 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 lan-
guages need to label words with case and gender information. Tagsets for morpho-
logically rich languages are therefore sequences of morphological tags rather than a
single primitive tag. Here’s a Turkish example, in which the word \textit{izin} has three
possible morphological/part-of-speech tags and meanings (Hakkani-Tür et al., 2002):

1. Yerdeki \textit{izin} temizlenmesi gerek. 
   \textit{iz} + Noun+A3sg+Pnon+Gen
   \textit{The trace} on the floor should be cleaned.

2. Üzerinde parmak \textit{izin} kalmış 
   iz + Noun+A3sg+P2sg+Nom
   \textit{Your} finger \textit{print} is left on (it).

3. İçeri girmek için \textit{izin} almanız gerekiyor. 
   \textit{izin} + Noun+A3sg+Pnon+Nom
   \textit{You need} \textit{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.

8.8 Summary

This chapter introduced \textbf{parts of speech} and \textbf{named entities}, and the tasks of \textbf{part-
of-speech tagging} and \textbf{named entity recognition}:

- Languages generally have a small set of \textbf{closed class} words that are highly
  frequent, ambiguous, and act as \textbf{function words}, and \textbf{open-class} words like
  \textbf{nouns, verbs, adjectives}. Various part-of-speech \textbf{tagsets} exist, of between 40
  and 200 tags.

- \textbf{Part-of-speech tagging} is the process of assigning a part-of-speech label to
  each of a sequence of words.

- \textbf{Named entities} are words for proper nouns referring mainly to people, places,
  and organizations, but extended to many other types that aren’t strictly entities
  or even proper nouns.

- Two common approaches to \textbf{sequence modeling} are a \textbf{generative} approach,
  HMM tagging, and a \textbf{discriminative} approach, CRF tagging. We will see a
  neural approach in following chapters.

- The probabilities in HMM taggers are estimated by maximum likelihood es-
  timation on tag-labeled training corpora. The Viterbi algorithm is used for
  \textbf{decoding}, finding the most likely tag sequence.

- \textbf{Conditional Random Fields} or \textbf{CRF taggers} train a log-linear model that can
  choose the best tag sequence given an observation sequence, based on features
  that condition on the output tag, the prior output tag, the entire input sequence,
  and the current timestep. They use the Viterbi algorithm for inference, to
  choose the best sequence of tags, and a version of the Forward-Backward
  algorithm (see Appendix A) for training.
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 modern algorithms. The parser was implemented essentially as a cascade of finite-state transducers; see Joshi and Hopely (1999) and Karttunen (1999) for a reimplemention.

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.

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 HMM tagger implementation details, including the extension to trigram contexts, and the use of sophisticated unknown word features; its performance is still close to state of the art taggers.
Log-linear models for POS tagging were introduced by Ratnaparkhi (1996), who introduced a system called MXPOST which implemented a maximum entropy Markov model (MEMM), a slightly simpler version of a CRF. Around the same time, sequence labelers were applied to the task of named entity tagging, first with HMMs (Bikel et al., 1997) and MEMMs (McCallum et al., 2000), and then once CRFs were developed (Lafferty et al. 2001), they were also applied to NER (McCallum and Li, 2003). A wide exploration of features followed (Zhou et al., 2005). Neural approaches to NER mainly follow from the pioneering results of Collobert et al. (2011), who applied a CRF on top of a convolutional net. BiLSTMs with word and character-based embeddings as input followed shortly and became a standard neural algorithm for NER (Huang et al. 2015, Ma and Hovy 2016, Lample et al. 2016) followed by the more recent use of Transformers and BERT.

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 unknown word features described on page 17 come mainly from Ratnaparkhi (1996), with augmentations from Toutanova et al. (2003) and Manning (2011).

State of the art POS taggers use neural algorithms, either bidirectional RNNs or Transformers like BERT; see Chapter 9 and Chapter 11. HMM (Brants 2000; Thede and Harper 1999) and CRF tagger accuracies are likely just a tad lower.

Manning (2011) investigates the remaining 2.7% of errors in a high-performing tagger (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).


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 you can decode a 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.

8.7 Develop a set of regular expressions to recognize the character shape features described on page 17.

8.8 The BIO and other labeling schemes given in this chapter aren’t the only possible one. For example, the B tag can be reserved only for those situations where an ambiguity exists between adjacent entities. Propose a new set of BIO tags for use with your NER system. Experiment with it and compare its performance with the schemes presented in this chapter.

8.9 Names of works of art (books, movies, video games, etc.) are quite different from the kinds of named entities we’ve discussed in this chapter. Collect a list of names of works of art from a particular category from a Web-based source (e.g., gutenberg.org, amazon.com, imdb.com, etc.). Analyze your list and give examples of ways that the names in it are likely to be problematic for the techniques described in this chapter.

8.10 Develop an NER system specific to the category of names that you collected in the last exercise. Evaluate your system on a collection of text likely to contain instances of these named entities.


Chiticariu, L., Y. Li, and F. R. Reiss. 2013. Rule-Based Information Extraction is Dead! Long Live Rule-Based Information Extraction Systems! EMNLP.

Christodouloupolous, C., S. Goldwater, and M. Steedman. 2010. Two decades of unsupervised POS induction: How far have we come? EMNLP.

Church, K. W. 1988. A stochastic parts program and noun phrase parser for unrestricted text. ANLP.

Church, K. W. 1989. A stochastic parts program and noun phrase parser for unrestricted text. ICASSP.


Ma, X. and E. H. Hovy. 2016. End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. ACL.


Ratnaparkhi, A. 1996. A maximum entropy part-of-speech tagger. EMNLP.


