CHAPTER

17 Information Extraction

Imagine that you are an analyst with an investment firm that tracks airline stocks. You’re given the task of determining the relationship (if any) between airline announcements of fare increases and the behavior of their stocks the next day. Historical data about stock prices is easy to come by, but what about the airline announcements? You will need to know at least the name of the airline, the nature of the proposed fare hike, the dates of the announcement, and possibly the response of other airlines. Fortunately, these can be all found in news articles like this one:

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

This chapter presents techniques for extracting limited kinds of semantic content from text. This process of information extraction (IE), turns the unstructured information embedded in texts into structured data, for example for populating a relational database to enable further processing.

We begin with the first step in most IE tasks, finding the proper names or named entities in a text. The task of named entity recognition (NER) is to find each mention of a named entity in the text and label its type. What constitutes a named entity type is task specific; people, places, and organizations are common, but gene or protein names (Cohen and Demner-Fushman, 2014) or financial asset classes might be relevant for some tasks. Once all the named entities in a text have been extracted, they can be linked together in sets corresponding to real-world entities, inferring, for example, that mentions of United Airlines and United refer to the same company. This is the joint task of coreference resolution and entity linking which we defer til Chapter 21.

Next, we turn to the task of relation extraction: finding and classifying semantic relations among the text entities. These are often binary relations like child-of, employment, part-whole, and geospatial relations. Relation extraction has close links to populating a relational database.

Finally, we discuss three tasks related to events. Event extraction is finding events in which these entities participate, like, in our sample text, the fare increases...
by United and American and the reporting events said and cite. **Event coreference** (Chapter 21) is needed to figure out which event mentions in a text refer to the same event; in our running example the two instances of increase and the phrase the move all refer to the same event.

To figure out when the events in a text happened we extract **temporal expressions** like days of the week (Friday and Thursday), relative expressions like two days from now or next year and times such as 3:30 P.M.. These expressions must be normalized onto specific calendar dates or times of day to situate events in time. In our sample task, this will allow us to link Friday to the time of United’s announcement, and Thursday to the previous day’s fare increase, and produce a timeline in which United’s announcement follows the fare increase and American’s announcement follows both of those events.

Finally, many texts describe recurring stereotypical events or situations. The task of **template filling** is to find such situations in documents and fill in the template slots. These slot-fillers may consist of text segments extracted directly from the text, or concepts like times, amounts, or ontology entities that have been inferred from text elements through additional processing.

Our airline text is an example of this kind of stereotypical situation since airlines often raise fares and then wait to see if competitors follow along. In this situation, we can identify United as a lead airline that initially raised its fares, $6 as the amount, Thursday as the increase date, and American as an airline that followed along, leading to a filled template like the following.

<table>
<thead>
<tr>
<th>Fare-Raise Attempt:</th>
<th>Lead Airline: United Airlines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount: $6</td>
<td>Effective Date: 2006-10-26</td>
</tr>
<tr>
<td>Follower: American Airlines</td>
<td></td>
</tr>
</tbody>
</table>

## 17.1 Named Entity Recognition

The first step in information extraction is to detect the **entities** in the text. A named entity is, roughly speaking, anything that can be referred to with a proper name: a person, a location, an organization. The term 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 the sample text introduced earlier with the named entities marked:

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.

In addition to their use in extracting events and the relationship between participants, named entities are useful for many other language processing tasks. In
17.1 • Named Entity Recognition

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.

Figure 17.1 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>Turing is a giant of computer science.</td>
</tr>
<tr>
<td>Organization</td>
<td>ORG</td>
<td>companies, sports teams</td>
<td>The IPCC warned about the cyclone.</td>
</tr>
<tr>
<td>Location</td>
<td>LOC</td>
<td>regions, mountains, seas</td>
<td>The Mt. Sanitas loop is in Sunshine Canyon.</td>
</tr>
<tr>
<td>Geo-Political Entity</td>
<td>GPE</td>
<td>countries, states, provinces</td>
<td>Palo Alto is raising the fees for parking.</td>
</tr>
<tr>
<td>Facility</td>
<td>FAC</td>
<td>bridges, buildings, airports</td>
<td>Consider the Golden Gate Bridge.</td>
</tr>
<tr>
<td>Vehicles</td>
<td>VEH</td>
<td>planes, trains, automobiles</td>
<td>It was a classic Ford Falcon.</td>
</tr>
</tbody>
</table>

**Figure 17.1** A list of generic named entity types with the kinds of entities they refer to.

**Named entity recognition** means finding spans of text that constitute proper names and then classifying the type of the entity. Recognition is difficult partly because of the ambiguity of segmentation; we need to decide what’s an entity and what isn’t, and where the boundaries are. 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 Figures 17.2 and 17.3.

<table>
<thead>
<tr>
<th>Name</th>
<th>Possible Categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Washington</td>
<td>Person, Location, Political Entity, Organization, Vehicle</td>
</tr>
<tr>
<td>Downing St.</td>
<td>Location, Organization</td>
</tr>
<tr>
<td>IRA</td>
<td>Person, Organization, Monetary Instrument</td>
</tr>
<tr>
<td>Louis Vuitton</td>
<td>Person, Organization, Commercial Product</td>
</tr>
</tbody>
</table>

**Figure 17.2** Common categorical ambiguities associated with various proper names.

[PER Washington] was born into slavery on the farm of James Burroughs.
[ORG Washington] went up 2 games to 1 in the four-game series.
Blair arrived in [LOC Washington] for what may well be his last state visit.
In June, [GPE Washington] passed a primary seatbelt law.
The [VEH Washington] had proved to be a leaky ship, every passage I made...

**Figure 17.3** Examples of type ambiguities in the use of the name Washington.

17.1.1 NER as Sequence Labeling

The standard algorithm for named entity recognition is as a word-by-word sequence labeling task, in which the assigned tags capture both the boundary and the type. A sequence classifier like an MEMM/CRF or a bi-LSTM is trained to label the tokens in a text with tags that indicate the presence of particular kinds of named entities. Consider the following simplified excerpt from our running example.

[ORG American Airlines], a unit of [ORG AMR Corp.], immediately matched the move, spokesman [PER Tim Wagner] said.
Figure 17.4 shows the same excerpt represented with IOB tagging. In IOB tagging we introduce a tag for the beginning (B) and inside (I) of each entity type, and one for tokens outside (O) any entity. The number of tags is thus $2n + 1$ tags, where $n$ is the number of entity types. IOB tagging can represent exactly the same information as the bracketed notation.

<table>
<thead>
<tr>
<th>Words</th>
<th>IOB Label</th>
<th>IO Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>American</td>
<td>B-ORG</td>
<td>I-ORG</td>
</tr>
<tr>
<td>Airlines</td>
<td>I-ORG</td>
<td>I-ORG</td>
</tr>
<tr>
<td>,</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>a</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>unit</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>of</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>AMR Corp.</td>
<td>B-ORG</td>
<td>I-ORG</td>
</tr>
<tr>
<td>,</td>
<td>I-ORG</td>
<td>I-ORG</td>
</tr>
<tr>
<td>immediately</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>matched</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>the</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>move</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>,</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>spokesman</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Tim Wagner</td>
<td>B-PER</td>
<td>I-PER</td>
</tr>
<tr>
<td>said</td>
<td>I-PER</td>
<td>I-PER</td>
</tr>
<tr>
<td></td>
<td>O</td>
<td>O</td>
</tr>
</tbody>
</table>

We’ve also shown IO tagging, which loses some information by eliminating the B tag. Without the B tag IO tagging is unable to distinguish between two entities of the same type that are right next to each other. Since this situation doesn’t arise very often (usually there is at least some punctuation or other delimiter), IO tagging may be sufficient, and has the advantage of using only $n + 1$ tags.

In the following three sections we introduce the three standard families of algorithms for NER tagging: feature based (MEMM/CRF), neural (bi-LSTM), and rule-based.

### 17.1.2 A feature-based algorithm for NER

- 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
- base-phrase syntactic chunk label of $w_i$ and 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 $\leq 4$)
- $w_i$ is all upper case
- word shape of $w_i$, word shape of neighboring words
- short word shape of $w_i$, short word shape of neighboring words
- presence of hyphen

We've also shown IO tagging, which loses some information by eliminating the B tag. Without the B tag IO tagging is unable to distinguish between two entities of the same type that are right next to each other. Since this situation doesn’t arise very often (usually there is at least some punctuation or other delimiter), IO tagging may be sufficient, and has the advantage of using only $n + 1$ tags.

In the following three sections we introduce the three standard families of algorithms for NER tagging: feature based (MEMM/CRF), neural (bi-LSTM), and rule-based.
The first approach is to extract features and train an MEMM or CRF sequence model of the type we saw for part-of-speech tagging in Chapter 8. Figure 17.5 lists standard features used in such feature-based systems. We’ve seen many of these features before in the context of part-of-speech tagging, particularly for tagging unknown words. This is not surprising, as many unknown words are in fact named entities. Word shape features are thus particularly important in the context of NER. Recall that word shape features are used to represent the abstract letter pattern of the word by mapping lower-case letters to ‘x’, upper-case to ‘X’, numbers to ‘d’, and retaining punctuation. Thus for example I.M.F would map to X.X.X. and DC10-30 would map to XXdd-dd. A second class of shorter word shape features is also used. In these features consecutive character types are removed, so DC10-30 would be mapped to Xd-d but I.M.F would still map to X.X.X. This feature by itself accounts for a considerable part of the success of feature-based NER systems for English news text. Shape features are also particularly important in recognizing names of proteins and genes in biological texts.

For example the named entity token L’Occitane would generate the following non-zero valued feature values:

- prefix(w_i) = L
- suffix(w_i) = tane
- prefix(w_i) = L'
- suffix(w_i) = ane
- prefix(w_i) = L'0
- suffix(w_i) = ne
- prefix(w_i) = L'0c
- suffix(w_i) = e
- word-shape(w_i) = X'Xxxxxxxxx
- short-word-shape(w_i) = X'Xx

A gazetteer is a list of place names, often providing millions of entries for locations with detailed geographical and political information. A related resource is name-lists; the United States Census Bureau also provides extensive lists of first names and surnames derived from its decadal census in the U.S. Similar lists of corporations, commercial products, and all manner of things biological and mineral are also available from a variety of sources. Gazetteer and name features are typically implemented as a binary feature for each name list. Unfortunately, such lists can be difficult to create and maintain, and their usefulness varies considerably. While gazetteers can be quite effective, lists of persons and organizations are not always helpful (Mikheev et al., 1999).

Feature effectiveness depends on the application, genre, media, and language. For example, shape features, critical for English newswire texts, are of little use with automatic speech recognition transcripts, or other non-edited or informally-edited sources, or for languages like Chinese that don’t use orthographic case. The features in Fig. 17.5 should therefore be thought of as only a starting point.

Figure 17.6 illustrates the result of adding part-of-speech tags, syntactic base-phrase chunk tags, and some shape information to our earlier example.

Given such a training set, a sequence classifier like an MEMM can be trained to label new sentences. Figure 17.7 illustrates the operation of such a sequence labeler at the point where the token Corp. is next to be labeled. If we assume a context window that includes the two preceding and following words, then the features available to the classifier are those shown in the boxed area.

---

1 www.geonames.org
2 www.census.gov
17.1.3 A neural algorithm for NER

The standard neural algorithm for NER is based on the bi-LSTM introduced in Chapter 9. Recall that in that model, word and character embeddings are computed for input word $w_i$. These are passed through a left-to-right LSTM and a right-to-left LSTM, whose outputs are concatenated (or otherwise combined) to produce a single output layer at position $i$. In the simplest method, this layer can then be directly passed onto a softmax that creates a probability distribution over all NER tags, and the most likely tag is chosen as $t_i$.

For named entity tagging this greedy approach to decoding is insufficient, since it doesn’t allow us to impose the strong constraints neighboring tokens have on each other (e.g., the tag I-PER must follow another I-PER or B-PER). Instead a CRF layer is normally used on top of the bi-LSTM output, and the Viterbi decoding algorithm is used to decode. Fig. 17.8 shows a sketch of the algorithm.
17.1.4 Rule-based NER

While machine learned (neural or MEMM/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 IBM System T is a text understanding architecture in which a user specifies complex declarative constraints for tagging tasks in a formal query language that includes regular expressions, dictionaries, semantic constraints, NLP operators, and table structures, all of 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, allowing the results of one pass to influence the next. The stages typically first involve the use of rules that have extremely high precision but low recall. Subsequent stages employ more error-prone statistical methods that take the output of the first pass into account.

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

The intuition behind this staged approach is twofold. First, some of the entity mentions in a text will be more clearly indicative of a given entity’s class than others. Second, once an unambiguous entity mention is introduced into a text, it is likely that subsequent shortened versions will refer to the same entity (and thus the same type of entity).

17.1.5 Evaluation of Named Entity Recognition

The familiar metrics of recall, precision, and $F_1$ measure are used to evaluate NER systems. Remember 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 num-
ber of correctly labeled responses to the total labeled; and $F$-measure is the harmonic mean of the two. For named entities, the entity rather than the word is the unit of response. Thus in the example in Fig. 17.6, the two entities Tim Wagner and AMR Corp. and the non-entity said 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 American but not American Airlines as an organization would cause two errors, a false positive for O and a false negative for I-ORG. 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.

17.2 Relation Extraction

Next on our list of tasks is to discern the relationships that exist among the detected entities. Let’s return to our sample airline text:

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 tells us, for example, that Tim Wagner is a spokesman for American Airlines, that United is a unit of UAL Corp., and that American is a unit of AMR. These binary relations are instances of more generic relations such as part-of or employs that are fairly frequent in news-style texts. Figure 17.9 lists the 17 relations used in the ACE relation extraction evaluations and Fig. 17.10 shows some sample relations. We might also extract more domain-specific relation such as the notion of an airline route. For example from this text we can conclude that United has routes to Chicago, Dallas, Denver, and San Francisco.
### Relations Types Examples

<table>
<thead>
<tr>
<th>Relation Type</th>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical-Located</td>
<td>PER-GPE</td>
<td>He was in Tennessee</td>
</tr>
<tr>
<td>Part-Whole-Subsidiary</td>
<td>ORG-ORG</td>
<td>XYZ, the parent company of ABC</td>
</tr>
<tr>
<td>Person-Social-Family</td>
<td>PER-PER</td>
<td>Yoko’s husband John</td>
</tr>
<tr>
<td>Org-AFF-Founder</td>
<td>PER-ORG</td>
<td>Steve Jobs, co-founder of Apple...</td>
</tr>
</tbody>
</table>

**Figure 17.10** Semantic relations with examples and the named entity types they involve.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Classes</th>
<th>Relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>United, UAL, American Airlines, AMR</td>
<td>Tim Wagner is a person</td>
<td>United is a unit of UAL</td>
</tr>
<tr>
<td>Chicago, Dallas, Denver, and San Francisco</td>
<td>Chicago, Dallas, Denver, and San Francisco are places</td>
<td>American is a unit of AMR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tim Wagner works for American Airlines</td>
</tr>
<tr>
<td></td>
<td></td>
<td>United serves Chicago, Dallas, Denver, and San Francisco</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PartOf = {\langle a, b \rangle, \langle c, d \rangle}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OrgAff = {\langle c, e \rangle}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Serves = {\langle a, f \rangle, \langle a, g \rangle, \langle a, h \rangle, \langle a, i \rangle}</td>
</tr>
</tbody>
</table>

**Figure 17.11** A model-based view of the relations and entities in our sample text.

These relations correspond nicely to the model-theoretic notions we introduced in Chapter 15 to ground the meanings of the logical forms. That is, a relation consists of a set of ordered tuples over elements of a domain. In most standard information-extraction applications, the domain elements correspond to the named entities that occur in the text, to the underlying entities that result from co-reference resolution, or to entities selected from a domain ontology. Figure 17.11 shows a model-based view of the set of entities and relations that can be extracted from our running example. Notice how this model-theoretic view subsumes the NER task as well; named entity recognition corresponds to the identification of a class of unary relations.

Sets of relations have been defined for many other domains as well. For example, UMLS, the Unified Medical Language System from the US National Library of Medicine has a network that defines 134 broad subject categories, entity types, and 54 relations between the entities, such as the following:

<table>
<thead>
<tr>
<th>Entity</th>
<th>Relation</th>
<th>Entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Injury</td>
<td>disrupts</td>
<td>Physiological Function</td>
</tr>
<tr>
<td>Bodily Location</td>
<td>location-of</td>
<td>Biologic Function</td>
</tr>
<tr>
<td>Anatomical Structure</td>
<td>part-of</td>
<td>Organism</td>
</tr>
<tr>
<td>Pharmacologic Substance</td>
<td>causes</td>
<td>Pathological Function</td>
</tr>
<tr>
<td>Pharmacologic Substance</td>
<td>treats</td>
<td>Pathologic Function</td>
</tr>
</tbody>
</table>

Given a medical sentence like this one:

(17.1) Doppler echocardiography can be used to diagnose left anterior descending artery stenosis in patients with type 2 diabetes

We could thus extract the UMLS relation:

_Echocardiography, Doppler Diagnoses Acquired stenosis_

Wikipedia also offers a large supply of relations, drawn from infoboxes, structured tables associated with certain Wikipedia articles. For example, the Wikipedia...
infobox for Stanford includes structured facts like state = "California" or president = "Mark Tessier-Lavigne". These facts can be turned into relations like president-of or located-in, or into relations in a metalanguage called RDF (Resource Description Framework). An RDF triple is a tuple of entity-relation-entity, called a subject-predicate-object expression. Here’s a sample RDF triple:

```
subject     predicate     object
Golden Gate Park location San Francisco
```

For example the crowdsourced DBpedia (Bizer et al., 2009) is an ontology derived from Wikipedia containing over 2 billion RDF triples. Another dataset from Wikipedia infoboxes, Freebase (Bollacker et al., 2008), now part of Wikidata (www.wikidata.org), has relations like

- people/person/nationality
- location/location/contains

WordNet or other ontologies offer useful ontological relations that express hierarchical relations between words or concepts. For example WordNet has the is-a or hypernym relation between classes,

- Giraffe is-a ruminant is-a ungulate is-a mammal is-a vertebrate ...

WordNet also has Instance-of relation between individuals and classes, so that for example San Francisco is in the Instance-of relation with city. Extracting these relations is an important step in extending or building ontologies.

There are five main classes of algorithms for relation extraction: hand-written patterns, supervised machine learning, semi-supervised (via bootstrapping and via distant supervision), and unsupervised. We’ll introduce each of these in the next sections.

### 17.2.1 Using Patterns to Extract Relations

The earliest and still common algorithm for relation extraction is lexico-syntactic patterns, first developed by Hearst (1992a). Consider the following sentence:

> Agar is a substance prepared from a mixture of red algae, such as Gelidium, for laboratory or industrial use.

Hearst points out that most human readers will not know what Gelidium is, but that they can readily infer that it is a kind of (a hyponym of) red algae, whatever that is. She suggests that the following lexico-syntactic pattern

```
NP₀ such as NP₁{,NP₂,...,(and|or)NPᵢ}, i ≥ 1
```

implies the following semantics

```
∀ NPᵢ, i ≥ 1, hyponym(NPᵢ, NP₀)
```

allowing us to infer

```
hyponym(Gelidium, red algae)
```

Figure 17.12 shows five patterns Hearst (1992a, 1998) suggested for inferring the hyponym relation; we’ve shown NP₁ as the parent/hyponym. Modern versions of the pattern-based approach extend it by adding named entity constraints. For example if our goal is to answer questions about “Who holds what office in which organization?”, we can use patterns like the following:
Figure 17.12 Hand-built lexico-syntactic patterns for finding hypernyms, using {} to mark optionality (Hearst 1992a, Hearst 1998).

| NP { NP}* {} (and|or) other NP_H | temples, treasuries, and other important civic buildings |
|-----------------------------------|---------------------------------------------------------|
| such NP_H as { NP}* (or|and) NP | red algae such as Gelidium |
| such NP_H as { NP}* (or|and) NP | such authors as Herrick, Goldsmith, and Shakespeare |
| NP_H {} including { NP}* (or|and) NP | common-law countries, including Canada and England |
| NP_H {} especially { NP}* (or|and) NP | European countries, especially France, England, and Spain |

Hand-built patterns have the advantage of high-precision and they can be tailored to specific domains. On the other hand, they are often low-recall, and it’s a lot of work to create them for all possible patterns.

17.2.2 Relation Extraction via Supervised Learning

Supervised machine learning approaches to relation extraction follow a scheme that should be familiar by now. A fixed set of relations and entities is chosen, a training corpus is hand-annotated with the relations and entities, and the annotated texts are then used to train classifiers to annotate an unseen test set.

The most straightforward approach has three steps, illustrated in Fig. 17.13. Step one is to find pairs of named entities (usually in the same sentence). In step two, a filtering classifier is trained to make a binary decision as to whether a given pair of named entities are related (by any relation). Positive examples are extracted directly from all relations in the annotated corpus, and negative examples are generated from within-sentence entity pairs that are not annotated with a relation. In step 3, a classifier is trained to assign a label to the relations that were found by step 2. The use of the filtering classifier can speed up the final classification and also allows the use of distinct feature-sets appropriate for each task. For each of the two classifiers, we can use any of the standard classification techniques (logistic regression, neural network, SVM, etc.)

function FINDRELATIONS(words) returns relations
relations ← nil
entities ← FINDENTITIES(words)
forall entity pairs (e1,e2) in entities do
if RELATED?(e1,e2)
relations ← relations+CLASSIFYRELATION(e1,e2)

Figure 17.13 Finding and classifying the relations among entities in a text.

For the feature-based classifiers like logistic regression or random forests the most important step is to identify useful features. Let’s consider features for clas-
sifying the relationship between *American Airlines* (Mention 1, or M1) and *Tim Wagner* (Mention 2, M2) from this sentence:

(17.5) **American Airlines**, a unit of AMR, immediately matched the move, spokesman **Tim Wagner** said

Useful word features include

- The headwords of M1 and M2 and their concatenation
  
  **American** **Wagner** **American Airlines-Wagner**

- Bag-of-words and bigrams in M1 and M2
  
  **American, Airlines, Tim, Wagner, American Airlines, Tim Wagner**

- Words or bigrams in particular positions
  
  M2: -1 spokesman  
  M2: +1 said

- Bag of words or bigrams between M1 and M2:
  
  a, AMR, of, immediately, matched, move, spokesman, the, unit

- Stemmed versions of the same

Embeddings can be used to represent words in any of these features. Useful named entity features include

- Named-entity types and their concatenation
  
  (M1: ORG, M2: PER, M1M2: ORG-PER)

- Entity Level of M1 and M2 (from the set NAME, NOMINAL, PRONOUN)
  
  M1: NAME [it or he would be PRONOUN]  
  M2: NAME [the company would be NOMINAL]

- Number of entities between the arguments (in this case 1, for AMR)

The **syntactic structure** of a sentence can also signal relationships among its entities. Syntax is often featured by using strings representing **syntactic paths**: the (dependency or constituency) path traversed through the tree in getting from one entity to the other.

- Base syntactic chunk sequence from M1 to M2
  
  **NP NP PP VP NP NP**

- Constituent paths between M1 and M2
  
  **NP ↑ NP ↑ S ↑ S ↓ NP**

- Dependency-tree paths
  
  **Airlines ←subj matched ←comp said →subj Wagner**

Figure 17.14 summarizes many of the features we have discussed that could be used for classifying the relationship between *American Airlines* and *Tim Wagner* from our example text.

Neural models for relation extraction similarly treat the task as supervised classification. One option is to use a similar architecture as we saw for named entity tagging: a bi-LSTM model with word embeddings as inputs and a single softmax classification of the sentence output as a 1-of-N relation label. Because relations often hold between entities that are far part in a sentence (or across sentences), it may be possible to get higher performance from algorithms like convolutional nets (dos Santos et al., 2015) or chain or tree LSTMs (Miwa and Bansal 2016, Peng et al. 2017).

In general, if the test set is similar enough to the training set, and if there is enough hand-labeled data, supervised relation extraction systems can get high accuracies. But labeling a large training set is extremely expensive and supervised
models are brittle: they don’t generalize well to different text genres. For this reason, much research in relation extraction has focused on the semi-supervised and unsupervised approaches we turn to next.

### 17.2.3 Semisupervised Relation Extraction via Bootstrapping

Supervised machine learning assumes that we have lots of labeled data. Unfortunately, this is expensive. But suppose we just have a few high-precision seed patterns, like those in Section 17.2.1, or perhaps a few seed tuples. That’s enough to bootstrap a classifier! Bootstrapping proceeds by taking the entities in the seed pair, and then finding sentences (on the web, or whatever dataset we are using) that contain both entities. From all such sentences, we extract and generalize the context around the entities to learn new patterns. Fig. 17.15 sketches a basic algorithm.

```plaintext
function BOOTSTRAP(Relation R) returns new relation tuples
    tuples ← Gather a set of seed tuples that have relation R
    iterate
        sentences ← find sentences that contain entities in tuples
        patterns ← generalize the context between and around entities in sentences
        newpairs ← use patterns to grep for more tuples
        newpairs ← newpairs with high confidence
        tuples ← tuples + newpairs
    return tuples
```

Suppose, for example, that we need to create a list of airline/hub pairs, and we know only that Ryanair has a hub at Charleroi. We can use this seed fact to discover new patterns by finding other mentions of this relation in our corpus. We search for the terms Ryanair, Charleroi and hub in some proximity. Perhaps we find the following set of sentences:

(17.6) Budget airline Ryanair, which uses Charleroi as a hub, scrapped all weekend flights out of the airport.

(17.7) All flights in and out of Ryanair’s Belgian hub at Charleroi airport were grounded on Friday...

<table>
<thead>
<tr>
<th>M1 headword</th>
<th>airlines (as a word token or an embedding)</th>
</tr>
</thead>
<tbody>
<tr>
<td>M2 headword</td>
<td>Wagner</td>
</tr>
<tr>
<td>Word(s) before M1</td>
<td>NONE</td>
</tr>
<tr>
<td>Word(s) after M2</td>
<td>said</td>
</tr>
<tr>
<td>Bag of words between</td>
<td>{a, unit of, AMR, Inc., immediately, matched, the, move, spokesman}</td>
</tr>
<tr>
<td>M1 type</td>
<td>ORG</td>
</tr>
<tr>
<td>M2 type</td>
<td>PERS</td>
</tr>
<tr>
<td>Concatenated types</td>
<td>ORG-PERS</td>
</tr>
<tr>
<td>Constituent path</td>
<td>NP ↑ NP ↑ S ↑ S ↓ NP</td>
</tr>
<tr>
<td>Base phrase path</td>
<td>NP → NP → PP → NP → VP → NP → NP</td>
</tr>
<tr>
<td>Typed-dependency path</td>
<td>Airlines ←subj matched ←comp said →subj Wagner</td>
</tr>
</tbody>
</table>

Figure 17.14 Sample of features extracted during classification of the &lt;American Airlines, Tim Wagner&gt; tuple; M1 is the first mention, M2 the second.
A spokesman at Charleroi, a main hub for Ryanair, estimated that 8000 passengers had already been affected.

From these results, we can use the context of words between the entity mentions, the words before mention one, the word after mention two, and the named entity types of the two mentions, and perhaps other features, to extract general patterns such as the following:

/ [ORG], which uses [LOC] as a hub /
/ [ORG]'s hub at [LOC] /
/ [LOC] a main hub for [ORG] /

These new patterns can then be used to search for additional tuples.

Bootstrapping systems also assign confidence values to new tuples to avoid semantic drift. In semantic drift, an erroneous pattern leads to the introduction of erroneous tuples, which, in turn, lead to the creation of problematic patterns and the meaning of the extracted relations 'drifts'. Consider the following example:

Sydney has a ferry hub at Circular Quay.

If accepted as a positive example, this expression could lead to the incorrect introduction of the tuple \( \langle \text{Sydney}, \text{CircularQuay} \rangle \). Patterns based on this tuple could propagate further errors into the database.

Confidence values for patterns are based on balancing two factors: the pattern’s performance with respect to the current set of tuples and the pattern’s productivity in terms of the number of matches it produces in the document collection. More formally, given a document collection \( \mathcal{D} \), a current set of tuples \( T \), and a proposed pattern \( p \), we need to track two factors:

- **hits**: the set of tuples in \( T \) that \( p \) matches while looking in \( \mathcal{D} \)
- **finds**: The total set of tuples that \( p \) finds in \( \mathcal{D} \)

The following equation balances these considerations (Riloff and Jones, 1999).

\[
\text{Conf}_{\text{RLogF}}(p) = \frac{\text{hits}_p}{\text{finds}_p} \times \log(\text{finds}_p)
\]

This metric is generally normalized to produce a probability.

We can assess the confidence in a proposed new tuple by combining the evidence supporting it from all the patterns \( P \) that match that tuple in \( \mathcal{D} \) (Agichtein and Gravano, 2000). One way to combine such evidence is the noisy-or technique. Assume that a given tuple is supported by a subset of the patterns in \( P \), each with its own confidence assessed as above. In the noisy-or model, we make two basic assumptions. First, that for a proposed tuple to be false, all of its supporting patterns must have been in error, and second, that the sources of their individual failures are all independent. If we loosely treat our confidence measures as probabilities, then the probability of any individual pattern \( p \) failing is \( 1 - \text{Conf}(p) \); the probability of all of the supporting patterns for a tuple being wrong is the product of their individual failure probabilities, leaving us with the following equation for our confidence in a new tuple.

\[
\text{Conf}(t) = 1 - \prod_{p \in P} (1 - \text{Conf}(p))
\]

Setting conservative confidence thresholds for the acceptance of new patterns and tuples during the bootstrapping process helps prevent the system from drifting away from the targeted relation.
17.2.4 Distant Supervision for Relation Extraction

Although text that has been hand-labeled with relation labels is extremely expensive to produce, there are ways to find indirect sources of training data. The distant supervision method of Mintz et al. (2009) combines the advantages of bootstrapping with supervised learning. Instead of just a handful of seeds, distant supervision uses a large database to acquire a huge number of seed examples, creates lots of noisy pattern features from all these examples and then combines them in a supervised classifier.

For example suppose we are trying to learn the place-of-birth relationship between people and their birth cities. In the seed-based approach, we might have only 5 examples to start with. But Wikipedia-based databases like DBPedia or Freebase have tens of thousands of examples of many relations; including over 100,000 examples of place-of-birth, \(<Edwin Hubble, Marshfield>, <Albert Einstein, Ulm>, etc.,). The next step is to run named entity taggers on large amounts of text—Mintz et al. (2009) used 800,000 articles from Wikipedia—and extract all sentences that have two named entities that match the tuple, like the following:

...Hubble was born in Marshfield...
...Einstein, born (1879), Ulm...
...Hubble’s birthplace in Marshfield...

Training instances can now be extracted from this data, one training instance for each identical tuple \(<relation, entity1, entity2>\). Thus there will be one training instance for each of:

\(<born-in, Edwin Hubble, Marshfield>\)
\(<born-in, Albert Einstein, Ulm>\)
\(<born-year, Albert Einstein, 1879>\)

and so on.

We can then apply feature-based or neural classification. For feature-based classification, standard supervised relation extraction features like the named entity labels of the two mentions, the words and dependency paths in between the mentions, and neighboring words. Each tuple will have features collected from many training instances; the feature vector for a single training instance like \(<born-in, Albert Einstein, Ulm>\) will have lexical and syntactic features from many different sentences that mention Einstein and Ulm.

Because distant supervision has very large training sets, it is also able to use very rich features that are conjunctions of these individual features. So we will extract thousands of patterns that conjoin the entity types with the intervening words or dependency paths like these:

PER was born in LOC
PER, born (XXXX), LOC
PER’s birthplace in LOC

To return to our running example, for this sentence:

\[17.12\] American Airlines, a unit of AMR, immediately matched the move, spokesman Tim Wagner said

we would learn rich conjunction features like this one:

\[M1 = ORG & M2 = PER & nextword="said" & path= NP ↑ NP ↑ S ↑ S ↓ NP\]

The result is a supervised classifier that has a huge rich set of features to use in detecting relations. Since not every test sentence will have one of the training...
relations, the classifier will also need to be able to label an example as *no-relation*. This label is trained by randomly selecting entity pairs that do not appear in any Freebase relation, extracting features for them, and building a feature vector for each such tuple. The final algorithm is sketched in Fig. 17.16.

```plaintext
function Distant Supervision(Database D, Text T) returns relation classifier C
    foreach relation R
        foreach tuple (e1,e2) of entities with relation R in D
            sentences ← Sentences in T that contain e1 and e2
            f ← Frequent features in sentences
            observations ← observations + new training tuple (e1, e2, f, R)
        C ← Train supervised classifier on observations
    return C
```

**Figure 17.16** The distant supervision algorithm for relation extraction. A neural classifier might not need to use the feature set \( f \).

Distant supervision shares advantages with each of the methods we’ve examined. Like supervised classification, distant supervision uses a classifier with lots of features, and supervised by detailed hand-created knowledge. Like pattern-based classifiers, it can make use of high-precision evidence for the relation between entities. Indeed, distance supervision systems learn patterns just like the hand-built patterns of early relation extractors. For example the *is-a* or *hypernym* extraction system of Snow et al. (2005) used hypernym/hyponym NP pairs from WordNet as distant supervision, and then learned new patterns from large amounts of text. Their system induced exactly the original 5 template patterns of Hearst (1992a), but also 70,000 additional patterns including these four:

- NP\(_H\) like NP  *Many hormones like leptin...*
- NP\(_H\) called NP  *...using a markup language called XHTML*
- NP is a NP\(_H\)  *Ruby is a programming language...*
- NP, a NP\(_H\)  *IBM, a company with a long...*

This ability to use a large number of features simultaneously means that, unlike the iterative expansion of patterns in seed-based systems, there’s no semantic drift. Like unsupervised classification, it doesn’t use a labeled training corpus of texts, so it isn’t sensitive to genre issues in the training corpus, and relies on very large amounts of unlabeled data. Distant supervision also has the advantage that it can create training tuples to be used with neural classifiers, where features are not required.

But distant supervision can only help in extracting relations for which a large enough database already exists. To extract new relations without datasets, or relations for new domains, purely unsupervised methods must be used.

### 17.2.5 Unsupervised Relation Extraction

The goal of unsupervised relation extraction is to extract relations from the web when we have no labeled training data, and not even any list of relations. This task is often called *open information extraction* or *Open IE*. In Open IE, the relations are simply strings of words (usually beginning with a verb).

For example, the ReVerb system (Fader et al., 2011) extracts a relation from a sentence \( s \) in 4 steps:
1. Run a part-of-speech tagger and entity chunker over $s$
2. For each verb in $s$, find the longest sequence of words $w$ that start with a verb and satisfy syntactic and lexical constraints, merging adjacent matches.
3. For each phrase $w$, find the nearest noun phrase $x$ to the left which is not a relative pronoun, wh-word or existential “there”. Find the nearest noun phrase $y$ to the right.
4. Assign confidence $c$ to the relation $r = (x,w,y)$ using a confidence classifier and return it.

A relation is only accepted if it meets syntactic and lexical constraints. The syntactic constraints ensure that it is a verb-initial sequence that might also include nouns (relations that begin with light verbs like make, have, or do often express the core of the relation with a noun, like have a hub in):

$$V \mid VP \mid VW*P$$
$$V = \text{verb particle? adv?}$$
$$W = (\text{noun} \mid \text{adj} \mid \text{adv} \mid \text{pron} \mid \text{det})$$
$$P = (\text{prep} \mid \text{particle} \mid \text{inf. marker})$$

The lexical constraints are based on a dictionary $D$ that is used to prune very rare, long relation strings. The intuition is to eliminate candidate relations that don’t occur with sufficient number of distinct argument types and so are likely to be bad examples. The system first runs the above relation extraction algorithm offline on 500 million web sentences and extracts a list of all the relations that occur after normalizing them (removing inflection, auxiliary verbs, adjectives, and adverbs). Each relation $r$ is added to the dictionary if it occurs with at least 20 different arguments.

Finally, a confidence value is computed for each relation using a logistic regression classifier. The classifier is trained by taking 1000 random web sentences, running the extractor, and hand labelling each extracted relation as correct or incorrect. A confidence classifier is then trained on this hand-labeled data, using features of the relation and the surrounding words. Fig. 17.17 shows some sample features used in the classification.

| $(x,r,y)$ covers all words in $s$ |
| the last preposition in $r$ is for |
| the last preposition in $r$ is on |
| $\text{len}(s) \leq 10$ |
| there is a coordinating conjunction to the left of $r$ in $s$ |
| $r$ matches a lone V in the syntactic constraints |
| there is preposition to the left of $x$ in $s$ |
| there is an NP to the right of $y$ in $s$ |

Figure 17.17 Features for the classifier that assigns confidence to relations extracted by the Open Information Extraction system REVERB (Fader et al., 2011).

For example the following sentence:

(17.13) United has a hub in Chicago, which is the headquarters of United Continental Holdings.

has the relation phrases has a hub in and is the headquarters of (it also has has and is, but longer phrases are preferred). Step 3 finds United to the left and Chicago to the right of has a hub in, and skips over which to find Chicago to the left of is the headquarters of. The final output is:
r1: <United, has a hub in, Chicago>
r2: <Chicago, is the headquarters of, United Continental Holdings>

The great advantage of unsupervised relation extraction is its ability to handle a huge number of relations without having to specify them in advance. The disadvantage is the need to map these large sets of strings into some canonical form for adding to databases or other knowledge sources. Current methods focus heavily on relations expressed with verbs, and so will miss many relations that are expressed nominally.

17.2.6 Evaluation of Relation Extraction

Supervised relation extraction systems are evaluated by using test sets with human-annotated, gold-standard relations and computing precision, recall, and F-measure. Labeled precision and recall require the system to classify the relation correctly, whereas unlabeled methods simply measure a system’s ability to detect entities that are related.

Semi-supervised and unsupervised methods are much more difficult to evaluate, since they extract totally new relations from the web or a large text. Because these methods use very large amounts of text, it is generally not possible to run them solely on a small labeled test set, and as a result it’s not possible to pre-annotate a gold set of correct instances of relations.

For these methods it’s possible to approximate (only) precision by drawing a random sample of relations from the output, and having a human check the accuracy of each of these relations. Usually this approach focuses on the tuples to be extracted from a body of text rather than on the relation mentions; systems need not detect every mention of a relation to be scored correctly. Instead, the evaluation is based on the set of tuples occupying the database when the system is finished. That is, we want to know if the system can discover that Ryanair has a hub at Charleroi; we don’t really care how many times it discovers it. The estimated precision $\hat{P}$ is then

$$\hat{P} = \frac{\text{# of correctly extracted relation tuples in the sample}}{\text{total # of extracted relation tuples in the sample}}$$  \hspace{1cm} (17.14)

Another approach that gives us a little bit of information about recall is to compute precision at different levels of recall. Assuming that our system is able to rank the relations it produces (by probability, or confidence) we can separately compute precision for the top 1000 new relations, the top 10,000 new relations, the top 100,000, and so on. In each case we take a random sample of that set. This will show us how the precision curve behaves as we extract more and more tuples. But there is no way to directly evaluate recall.

17.3 Extracting Times

Times and dates are a particularly important kind of named entity that play a role in question answering, in calendar and personal assistant applications. In order to reason about times and dates, after we extract these temporal expressions they must be normalized—converted to a standard format so we can reason about them. In this section we consider both the extraction and normalization of temporal expressions.
17.3.1 Temporal Expression Extraction

Temporal expressions are those that refer to absolute points in time, relative times, durations, and sets of these. **Absolute** temporal expressions are those that can be mapped directly to calendar dates, times of day, or both. **Relative** temporal expressions map to particular times through some other reference point (as in a week from last Tuesday). Finally, **durations** denote spans of time at varying levels of granularity (seconds, minutes, days, weeks, centuries, etc.). Figure 17.18 lists some sample temporal expressions in each of these categories.

<table>
<thead>
<tr>
<th>Absolute</th>
<th>Relative</th>
<th>Durations</th>
</tr>
</thead>
<tbody>
<tr>
<td>April 24, 1916</td>
<td>yesterday</td>
<td>four hours</td>
</tr>
<tr>
<td>The summer of ’77</td>
<td>next semester</td>
<td>three weeks</td>
</tr>
<tr>
<td>10:15 AM</td>
<td>two weeks from yesterday</td>
<td>six days</td>
</tr>
<tr>
<td>The 3rd quarter of 2006</td>
<td>last quarter</td>
<td>the last three quarters</td>
</tr>
</tbody>
</table>

**Figure 17.18** Examples of absolute, relational and durational temporal expressions.

Temporal expressions are grammatical constructions that have temporal **lexical triggers** as their heads. Lexical triggers might be nouns, proper nouns, adjectives, and adverbs; full temporal expressions consist of their phrasal projections: noun phrases, adjective phrases, and adverbial phrases. Figure 17.19 provides examples.

<table>
<thead>
<tr>
<th>Category</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Noun</td>
<td>morning, noon, night, winter, dusk, dawn</td>
</tr>
<tr>
<td>Proper Noun</td>
<td>January, Monday, Ides, Easter, Rosh Hashana, Ramadan, Tet</td>
</tr>
<tr>
<td>Adjective</td>
<td>recent, past, annual, former</td>
</tr>
<tr>
<td>Adverb</td>
<td>hourly, daily, monthly, yearly</td>
</tr>
</tbody>
</table>

**Figure 17.19** Examples of temporal lexical triggers.

Let’s look at the TimeML annotation scheme, in which temporal expressions are annotated with an XML tag, TIMEX3, and various attributes to that tag (Pustejovsky et al. 2005, Ferro et al. 2005). The following example illustrates the basic use of this scheme (we defer discussion of the attributes until Section 17.3.2).

A fare increase initiated <TIMEX3>last week</TIMEX3> by UAL Corp’s United Airlines was matched by competitors over <TIMEX3>the weekend</TIMEX3>, marking the second successful fare increase in <TIMEX3>two weeks</TIMEX3>.

The temporal expression recognition task consists of finding the start and end of all of the text spans that correspond to such temporal expressions. **Rule-based approaches** to temporal expression recognition use cascades of automata to recognize patterns at increasing levels of complexity. Tokens are first part-of-speech tagged, and then larger and larger chunks are recognized from the results from previous stages, based on patterns containing trigger words (e.g., February) or classes (e.g., MONTH). Figure 17.20 gives a fragment from a rule-based system.

**Sequence-labeling approaches** follow the same IOB scheme used for named-entity tags, marking words that are either inside, outside or at the beginning of a TIMEX3-delimited temporal expression with the I, O, and B tags as follows:

* A fare increase initiated last week by UAL Corp’s...
* O O O O B I O O O
Features are extracted from the token and its context, and a statistical sequence labeler is trained (any sequence model can be used). Figure 17.21 lists standard features used in temporal tagging.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token</td>
<td>The target token to be labeled</td>
</tr>
<tr>
<td>Tokens in window</td>
<td>Bag of tokens in the window around a target</td>
</tr>
<tr>
<td>Shape</td>
<td>Character shape features</td>
</tr>
<tr>
<td>POS</td>
<td>Parts of speech of target and window words</td>
</tr>
<tr>
<td>Chunk tags</td>
<td>Base-phrase chunk tag for target and words in a window</td>
</tr>
<tr>
<td>Lexical triggers</td>
<td>Presence in a list of temporal terms</td>
</tr>
</tbody>
</table>

Temporal expression recognizers are evaluated with the usual recall, precision, and \( F \)-measures. A major difficulty for all of these very lexicalized approaches is avoiding expressions that trigger false positives:

(17.15) 1984 tells the story of Winston Smith...
(17.16) ...U2’s classic Sunday Bloody Sunday

### 17.3.2 Temporal Normalization

**Temporal normalization** is the process of mapping a temporal expression to either a specific point in time or to a duration. Points in time correspond to calendar dates, to times of day, or both. Durations primarily consist of lengths of time but may also include information about start and end points. Normalized times are represented with the `VALUE` attribute from the ISO 8601 standard for encoding temporal values (ISO8601, 2004). Fig. 17.22 reproduces our earlier example with the value attributes added in.

![Figure 17.22](image-url)
The encodings for the temporal expressions in our sample text all follow from this date, and are shown here as values for the VALUE attribute.

The first temporal expression in the text proper refers to a particular week of the year. In the ISO standard, weeks are numbered from 01 to 53, with the first week of the year being the one that has the first Thursday of the year. These weeks are represented with the template YYYY-Wnn. The ISO week for our document date is week 27; thus the value for last week is represented as “2007-W26”.

The next temporal expression is the weekend. ISO weeks begin on Monday; thus, weekends occur at the end of a week and are fully contained within a single week. Weekends are treated as durations, so the value of the VALUE attribute has to be a length. Durations are represented according to the pattern Pnx, where n is an integer denoting the length and x represents the unit, as in P3Y for three years or P2D for two days. In this example, one weekend is captured as P1WE. In this case, there is also sufficient information to anchor this particular weekend as part of a particular week. Such information is encoded in the ANCHORTIMEID attribute. Finally, the phrase two weeks also denotes a duration captured as P2W. There is a lot more to the various temporal annotation standards—far too much to cover here. Figure 17.23 describes some of the basic ways that other times and durations are represented. Consult ISO8601 (2004), Ferro et al. (2005), and Pustejovsky et al. (2005) for more details.

<table>
<thead>
<tr>
<th>Unit</th>
<th>Pattern</th>
<th>Sample Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully specified dates</td>
<td>YYYY-MM-DD</td>
<td>1991-09-28</td>
</tr>
<tr>
<td>Weeks</td>
<td>YYYY-Wnn</td>
<td>2007-W27</td>
</tr>
<tr>
<td>Weekends</td>
<td>PnWE</td>
<td>P1WE</td>
</tr>
<tr>
<td>24-hour clock times</td>
<td>HH:MM:SS</td>
<td>11:13:45</td>
</tr>
<tr>
<td>Dates and times</td>
<td>YYYY-MM-DDTHH:MM:SS</td>
<td>1991-09-28T11:00:00</td>
</tr>
<tr>
<td>Financial quarters</td>
<td>Qn</td>
<td>1999-Q3</td>
</tr>
</tbody>
</table>

Figure 17.23 Sample ISO patterns for representing various times and durations.

Most current approaches to temporal normalization are rule-based (Chang and Manning 2012, Strögen and Gertz 2013). Patterns that match temporal expressions are associated with semantic analysis procedures. As in the compositional rule-to-rule approach introduced in Chapter 16, the meaning of a constituent is computed from the meaning of its parts using a method specific to the constituent, although here the semantic composition rules involve temporal arithmetic rather than \( \lambda \)-calculus attachments.

**Fully qualified** date expressions contain a year, month, and day in some conventional form. The units in the expression must be detected and then placed in the correct place in the corresponding ISO pattern. The following pattern normalizes expressions like April 24, 1916.

\[
FQTE \rightarrow \text{Month Date }, \text{Year} \quad \{ \text{Year.val }\text{ – Month.val }\text{ – Date.val} \}
\]

The non-terminals Month, Date, and Year represent constituents that have already been recognized and assigned semantic values, accessed through the \* .val notation. The value of this FQE constituent can, in turn, be accessed as FQTE.val during further processing.

Fully qualified temporal expressions are fairly rare in real texts. Most temporal expressions in news articles are incomplete and are only implicitly anchored, often with respect to the dateline of the article, which we refer to as the document’s
temporal anchor. The values of temporal expressions such as today, yesterday, or tomorrow can all be computed with respect to this temporal anchor. The semantic procedure for today simply assigns the anchor, and the attachments for tomorrow and yesterday add a day and subtract a day from the anchor, respectively. Of course, given the cyclic nature of our representations for months, weeks, days, and times of day, our temporal arithmetic procedures must use modulo arithmetic appropriate to the time unit being used.

Unfortunately, even simple expressions such as the weekend or Wednesday introduce a fair amount of complexity. In our current example, the weekend clearly refers to the weekend of the week that immediately precedes the document date. But this won’t always be the case, as is illustrated in the following example.

(17.17) Random security checks that began yesterday at Sky Harbor will continue at least through the weekend.

In this case, the expression the weekend refers to the weekend of the week that the anchoring date is part of (i.e., the coming weekend). The information that signals this meaning comes from the tense of continue, the verb governing the weekend.

Relative temporal expressions are handled with temporal arithmetic similar to that used for today and yesterday. The document date indicates that our example article is ISO week 27, so the expression last week normalizes to the current week minus 1. To resolve ambiguous next and last expressions we consider the distance from the anchoring date to the nearest unit. Next Friday can refer either to the immediately next Friday or to the Friday following that, but the closer the document date is to a Friday, the more likely it is that the phrase will skip the nearest one. Such ambiguities are handled by encoding language and domain-specific heuristics into the temporal attachments.

17.4 Extracting Events and their Times

The task of event extraction is to identify mentions of events in texts. For the purposes of this task, an event mention is any expression denoting an event or state that can be assigned to a particular point, or interval, in time. The following markup of the sample text on page 19 shows all the events in this text.

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

In English, most event mentions correspond to verbs, and most verbs introduce events. However, as we can see from our example, this is not always the case. Events can be introduced by noun phrases, as in the move and the increase, and some verbs fail to introduce events, as in the phrasal verb took effect, which refers to when the event began rather than to the event itself. Similarly, light verbs such as make, take, and have often fail to denote events; for light verbs the event is often expressed by the nominal direct object (took a flight), and these light verbs just provide a syntactic structure for the noun’s arguments.
Various versions of the event extraction task exist, depending on the goal. For example in the TempEval shared tasks (Verhagen et al. 2009) the goal is to extract events and aspects like their aspectual and temporal properties. Events are to be classified as actions, states, *reporting events* (say, report, tell, explain), perception events, and so on. The aspect, tense, and modality of each event also needs to be extracted. Thus for example the various *said* events in the sample text would be annotated as (class=REPORTING, tense=PAST, aspect=PERFECTIVE).

Event extraction is generally modeled via supervised learning, detecting events via sequence models with IOB tagging, and assigning event classes and attributes with multi-class classifiers. Common features include surface information like parts of speech, lexical items, and verb tense information; see Fig. 17.24.

### Feature Explanation

<table>
<thead>
<tr>
<th>Feature</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character affixes</td>
<td>Character-level prefixes and suffixes of target word</td>
</tr>
<tr>
<td>Nominalization suffix</td>
<td>Character level suffixes for nominalizations (e.g., -tion)</td>
</tr>
<tr>
<td>Part of speech</td>
<td>Part of speech of the target word</td>
</tr>
<tr>
<td>Light verb</td>
<td>Binary feature indicating that the target is governed by a light verb</td>
</tr>
<tr>
<td>Subject syntactic category</td>
<td>Syntactic category of the subject of the sentence</td>
</tr>
<tr>
<td>Morphological stem</td>
<td>Stemmed version of the target word</td>
</tr>
<tr>
<td>Verb root</td>
<td>Root form of the verb basis for a nominalization</td>
</tr>
<tr>
<td>WordNet hypernyms</td>
<td>Hypernym set for the target</td>
</tr>
</tbody>
</table>

Figure 17.24 Features commonly used in both rule-based and machine learning approaches to event detection.

### 17.4.1 Temporal Ordering of Events

With both the events and the temporal expressions in a text having been detected, the next logical task is to use this information to fit the events into a complete timeline. Such a timeline would be useful for applications such as question answering and summarization. This ambitious task is the subject of considerable current research but is beyond the capabilities of current systems.

A somewhat simpler, but still useful, task is to impose a partial ordering on the events and temporal expressions mentioned in a text. Such an ordering can provide many of the same benefits as a true timeline. An example of such a partial ordering is the determination that the fare increase by *American Airlines* came after the fare increase by *United* in our sample text. Determining such an ordering can be viewed as a binary relation detection and classification task similar to those described earlier in Section 17.2. The temporal relation between events is classified into one of the standard set of Allen relations shown in Fig. 17.25 (Allen, 1984), using feature-based classifiers as in Section 17.2, trained on the TimeBank corpus with features like words/embeddings, parse paths, tense and aspect.

The *TimeBank* corpus consists of text annotated with much of the information we’ve been discussing throughout this section (Pustejovsky et al., 2003b). TimeBank 1.2 consists of 183 news articles selected from a variety of sources, including the Penn TreeBank and PropBank collections.

Each article in the TimeBank corpus has had the temporal expressions and event mentions in them explicitly annotated in the TimeML annotation (Pustejovsky et al., 2003a). In addition to temporal expressions and events, the TimeML annotation provides temporal links between events and temporal expressions that specify the nature of the relation between them. Consider the following sample sentence and
Delta Air Lines earnings soared 33% to a record in the fiscal first quarter, bucking the industry trend toward declining profits.

As annotated, this text includes three events and two temporal expressions. The events are all in the occurrence class and are given unique identifiers for use in further annotations. The temporal expressions include the creation time of the article, which serves as the document time, and a single temporal expression within the text.

In addition to these annotations, TimeBank provides four links that capture the temporal relations between the events and times in the text, using the Allen relations from Fig. 17.25. The following are the within-sentence temporal relations annotated for this example.
• Soaring is included in the fiscal first quarter
• Soaring is before 1989-10-26
• Soaring is simultaneous with the bucking
• Declining includes soaring

17.5 Template Filling

Many texts contain reports of events, and possibly sequences of events, that often correspond to fairly common, stereotypical situations in the world. These abstract situations or stories, related to what have been called scripts (Schank and Abelson, 1977), consist of prototypical sequences of sub-events, participants, and their roles. The strong expectations provided by these scripts can facilitate the proper classification of entities, the assignment of entities into roles and relations, and most critically, the drawing of inferences that fill in things that have been left unsaid. In their simplest form, such scripts can be represented as templates consisting of fixed sets of slots that take as values slot-fillers belonging to particular classes. The task of template filling is to find documents that invoke particular scripts and then fill the slots in the associated templates with fillers extracted from the text. These slot-fillers may consist of text segments extracted directly from the text, or they may consist of concepts that have been inferred from text elements through some additional processing.

A filled template from our original airline story might look like the following.

<table>
<thead>
<tr>
<th>FARE-RAISE ATTEMPT:</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEAD AIRLINE:</td>
</tr>
<tr>
<td>AMOUNT:</td>
</tr>
<tr>
<td>EFFECTIVE DATE:</td>
</tr>
<tr>
<td>FOLLOWER:</td>
</tr>
</tbody>
</table>

This template has four slots (LEAD AIRLINE, AMOUNT, EFFECTIVE DATE, FOLLOWER). The next section describes a standard sequence-labeling approach to filling slots. Section 17.5.2 then describes an older system based on the use of cascades of finite-state transducers and designed to address a more complex template-filling task that current learning-based systems don’t yet address.

17.5.1 Machine Learning Approaches to Template Filling

In the standard paradigm for template filling, we are trying to fill fixed known templates with known slots, and also assumes training documents labeled with examples of each template, and the fillers of each slot marked in the text. The is to create one template for each event in the input documents, with the slots filled with text from the document.

The task is generally modeled by training two separate supervised systems. The first system decides whether the template is present in a particular sentence. This task is called template recognition or sometimes, in a perhaps confusing bit of terminology, event recognition. Template recognition can be treated as a text classification task, with features extracted from every sequence of words that was labeled in training documents as filling any slot from the template being detected. The usual set of features can be used: tokens, embeddings, word shapes, part-of-speech tags, syntactic chunk tags, and named entity tags.
The second system has the job of role-filler extraction. A separate classifier is trained to detect each role (LEAD-AIRLINE, AMOUNT, and so on). This can be a binary classifier that is run on every noun-phrase in the parsed input sentence, or a sequence model run over sequences of words. Each role classifier is trained on the labeled data in the training set. Again, the usual set of features can be used, but now trained only on an individual noun phrase or the fillers of a single slot.

Multiple non-identical text segments might be labeled with the same slot label. For example in our sample text, the strings United or United Airlines might be labeled as the LEAD-AIRLINE. These are not incompatible choices and the coreference resolution techniques introduced in Chapter 21 can provide a path to a solution.

A variety of annotated collections have been used to evaluate this style of approach to template filling, including sets of job announcements, conference calls for papers, restaurant guides, and biological texts. Recent work focuses on extracting templates in cases where there is no training data or even predefined templates, by inducing templates as sets of linked events (Chambers and Jurafsky, 2011).

### 17.5.2 Earlier Finite-State Template-Filling Systems

The templates above are relatively simple. But consider the task of producing a template that contained all the information in a text like this one (Grishman and Sundheim, 1995):

Bridgestone Sports Co. said Friday it has set up a joint venture in Taiwan with a local concern and a Japanese trading house to produce golf clubs to be shipped to Japan. The joint venture, Bridgestone Sports Taiwan Co., capitalized at 20 million new Taiwan dollars, will start production in January 1990 with production of 20,000 iron and “metal wood” clubs a month.

The MUC-5 ‘joint venture’ task (the Message Understanding Conferences were a series of U.S. government-organized information-extraction evaluations) was to produce hierarchically linked templates describing joint ventures. Figure 17.27 shows a structure produced by the FASTUS system (Hobbs et al., 1997). Note how the filler of the ACTIVITY slot of the TIE-UP template is itself a template with slots.

<table>
<thead>
<tr>
<th>Tie-up-1</th>
<th>Activity-1:</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELATIONSHIP</td>
<td>COMPANY</td>
</tr>
<tr>
<td>ENTITIES</td>
<td>Bridgestone Sports Taiwan Co.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>JOINT VENTURE</td>
<td>Bridgestone Sports Taiwan Co.</td>
</tr>
<tr>
<td>ACTIVITY</td>
<td>Activity-1</td>
</tr>
<tr>
<td>AMOUNT</td>
<td>NTS$20000000</td>
</tr>
</tbody>
</table>

Figure 17.27 The templates produced by FASTUS given the input text on page 26.

Early systems for dealing with these complex templates were based on cascades of transducers based on hand-written rules, as sketched in Fig. 17.28.

The first four stages use hand-written regular expression and grammar rules to do basic tokenization, chunking, and parsing. Stage 5 then recognizes entities and events with a FST-based recognizer and inserts the recognized objects into the appropriate slots in templates. This FST recognizer is based on hand-built regular expressions like the following (NG indicates Noun-Group and VG Verb-Group), which matches the first sentence of the news story above.
17.6 Summary

This chapter has explored techniques for extracting limited forms of semantic content from texts.

- **Named entities** can be recognized and classified by featured-based or neural sequence labeling techniques.

- **Relations among entities** can be extracted by pattern-based approaches, supervised learning methods when annotated training data is available, lightly supervised bootstrapping methods when small numbers of seed tuples or seed patterns are available, distant supervision when a database of relations is available, and unsupervised or Open IE methods.

- Reasoning about time can be facilitated by detection and normalization of
temporal expressions through a combination of statistical learning and rule-based methods.

- **Events** can be detected and ordered in time using sequence models and classifiers trained on temporally- and event-labeled data like the *TimeBank corpus*.

- **Template-filling** applications can recognize stereotypical situations in texts and assign elements from the text to roles represented as fixed sets of slots.

### Bibliographical and Historical Notes

The earliest work on information extraction addressed the template-filling task in the context of the Frump system (DeJong, 1982). Later work was stimulated by the U.S. government-sponsored MUC conferences (Sundheim 1991, Sundheim 1992, Sundheim 1993, Sundheim 1995). Early MUC systems like CIRCUS system (Lehnert et al., 1991) and SCISOR (Jacobs and Rau, 1990) were quite influential and inspired later systems like FASTUS (Hobbs et al., 1997). Chinchor et al. (1993) describe the MUC evaluation techniques.

Due to the difficulty of porting systems from one domain to another, attention shifted to machine learning approaches.

Early supervised learning approaches to IE (Cardie 1993, Cardie 1994, Riloff 1993, Soderland et al. 1995, Huffman 1996) focused on automating the knowledge acquisition process, mainly for finite-state rule-based systems. Their success, and the earlier success of HMM-based speech recognition, led to the use of sequence labeling (HMMs: Bikel et al. 1997; MEMMs McCallum et al. 2000; CRFs: Lafferty et al. 2001), and a wide exploration of features (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).

Neural algorithms for relation extraction often explore architectures that can handle entities far apart in the sentence: recursive networks (Socher et al., 2012), convolutional nets (dos Santos et al., 2015), or chain or tree LSTMs (Miwa and Bansal 2016, Peng et al. 2017).

Progress in this area continues to be stimulated by formal evaluations with shared benchmark datasets, including the Automatic Content Extraction (ACE) evaluations of 2000-2007 on named entity recognition, relation extraction, and temporal expressions, the **KBP (Knowledge Base Population)** evaluations (Ji et al. 2010, Surdeanu 2013) of relation extraction tasks like slot filling (extracting attributes (‘slots’) like age, birthplace, and spouse for a given entity) and a series of SemEval workshops (Hendrickx et al., 2009).

Semisupervised relation extraction was first proposed by Hearst (1992b), and extended by systems like AutoSlog-TS (Riloff, 1996), DIPRE (Brin, 1998), SNOWBALL (Agichtein and Gravano, 2000), and (Jones et al., 1999). The distant supervision algorithm we describe was drawn from Mintz et al. (2009), who coined the term ‘distant supervision’, but similar ideas occurred in earlier systems like Craven and Kumlien (1999) and Morgan et al. (2004) under the name weakly labeled data, as well as in Snow et al. (2005) and Wu and Weld (2007). Among the many exten-
Exercises

17.1 Develop a set of regular expressions to recognize the character shape features described on page 5.

17.2 The IOB labeling scheme given in this chapter isn’t the only possible one. For example, an E tag might be added to mark the end of entities, or the B tag can be reserved only for those situations where an ambiguity exists between adjacent entities. Propose a new set of IOB tags for use with your NER system. Experiment with it and compare its performance with the scheme presented in this chapter.

17.3 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.

17.4 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.

17.5 Acronym expansion, the process of associating a phrase with an acronym, can be accomplished by a simple form of relational analysis. Develop a system based on the relation analysis approaches described in this chapter to populate a database of acronym expansions. If you focus on English Three Letter Acronyms (TLAs) you can evaluate your system’s performance by comparing it to Wikipedia’s TLA page.

17.6 A useful functionality in newer email and calendar applications is the ability to associate temporal expressions connected with events in email (doctor’s appointments, meeting planning, party invitations, etc.) with specific calendar entries. Collect a corpus of email containing temporal expressions related to event planning. How do these expressions compare to the kinds of expressions commonly found in news text that we’ve been discussing in this chapter?

17.7 Acquire the CMU seminar corpus and develop a template-filling system by using any of the techniques mentioned in Section 17.5. Analyze how well your system performs as compared with state-of-the-art results on this corpus.


