Chapter 20
Computational Lexical Semantics

To get a single right meaning is better than a ship-load of pearls,
To resolve a single doubt is like the bottom falling off the bucket.
Yuen Mei (1785) (translation by Arthur Waley)

The asphalt that Los Angeles is famous for occurs mainly on its freeways. But in the middle of the city is another patch of asphalt, the La Brea tar pits, and this asphalt preserves millions of fossil bones from the last of the Ice Ages of the Pleistocene Epoch. One of these fossils is the *Smilodon*, or sabre-toothed tiger, instantly recognizable by its long canines. Five million years ago or so, a completely different sabre-tooth tiger called *Thylacosmilus* lived in Argentina and other parts of South America. Thylacosmilus was a marsupial where Smilodon was a placental mammal, but had the same long upper canines and, like Smilodon, had a protective bone flange on the lower jaw. The similarity of these two mammals is one of many example of parallel or convergent evolution, in which particular contexts or environments lead to the evolution of very similar structures in different species (Gould, 1980).

The role of context is also important in the similarity of a less biological kind of organism: the word. Suppose we wanted to decide if two words have similar meanings. Not surprisingly, words with similar meanings often occur in similar contexts, whether in terms of corpora (having similar neighboring words or syntactic structures in sentences) or in terms of dictionaries and thesauruses (having similar definitions, or being nearby in the thesaurus hierarchy). Thus similarity of context turns out to be an important way to detect semantic similarity. Semantic similarity turns out to play an important roles in a diverse set of applications including information retrieval, question answering, summarization and generation, text classification, automatic essay grading and the detection of plagiarism.

In this chapter we introduce a series of topics related to computing with word meanings, or **computational lexical semantics**. Roughly in parallel with the sequence of topics in Ch. 19, we'll introduce computational tasks associated with word senses, relations among words, and the thematic structure of predicate-bearing words. We'll see the role of important role of context and similarity of sense in each of these.

We begin with **word sense disambiguation**, the task of examining word tokens in context and determining which sense of each word is being used. WSD is a task with a long history in computational linguistics, and as we will see, is a non-trivial undertaking given the somewhat elusive nature of many word senses. Nevertheless, there are robust algorithms that can achieve high levels of accuracy given certain reasonable assumptions. Many of these algorithms rely on contextual similarity to help choose the proper sense.

This will lead us natural to a consideration of the computation of **word similarity** and other relations between words, including the **hypernym**, **hyponym**, and **meronym** WordNet relations introduced in Ch. 19. We'll introduce methods based purely on corpus similarity, and others based on structured resources such as WordNet.

Finally, we describe algorithms for **semantic role labeling**, also known as **case role** or **thematic role assignment**. These algorithms generally use features extracted from syntactic parses to assign semantic roles such as AGENT, THEME and INSTRUMENT to the phrases in a sentence with respect to particular predicates.

20.1 Word Sense Disambiguation: Overview

Our discussion of compositional semantic analyzers in Ch. 18 pretty much ignored the issue of lexical ambiguity. It should be clear by now that this is an unreasonable approach. Without some means of selecting correct senses for the words in an input, the enormous amount of homonymy and polysemy in the lexicon would quickly overwhelm any approach in an avalanche of competing interpretations.

word sense disambiguation
WSD

The task of selecting the correct sense for a word is called **word sense disambiguation**, or **WSD**. Disambiguating word senses has the potential to improve many natural language processing tasks. As we'll see in Ch. 25, **machine translation** is one area where word sense ambiguities can cause severe problems; others include **question-answering**, **information retrieval**, and **text classification**. The way that WSD is exploited in these and other applications varies widely based on the particular needs of the application. The discussion presented here ignores these application-specific differences and focuses on the implementation and evaluation of WSD systems as a stand-alone task.

In their most basic form, WSD algorithms take as input a word in context along with a fixed inventory of potential word senses, and return the correct word sense for that use. Both the nature of the input and the inventory of senses depends on the task. For machine translation from English to Spanish, the sense tag inventory for an English word might be the set of different Spanish translations. If speech synthesis is our task, the inventory might be restricted to homographs with differing pronunciations such as *bass* and *bow*. If our task is automatic indexing of medical articles, the sense tag inventory might be the set of MeSH (Medical Subject Headings) thesaurus entries. When we are evaluating WSD in isolation, we can use the set of senses from a dictionary/thesaurus resource like WordNet or LDOCE. Fig. 20.1 shows an example for the word *bass*, which can refer to a musical instrument or a kind of fish.[1]

WordNet Sense	Spanish Translation	Roget Category	Target Word in Context
bass^4	lubina	FISH/INSECT	...fish as Pacific salmon and striped **bass** and...
bass^4	lubina	FISH/INSECT	...produce filets of smoked **bass** or sturgeon...
bass^7	bajo	MUSIC	...exciting jazz **bass** player since Ray Brown...
bass^7	bajo	MUSIC	...play **bass** because he doesn't have to solo...

Figure 20.1 Possible definitions for the inventory of sense tags for *bass*.

[1] The WordNet database includes 8 senses; we have arbitrarily selected two for this example; we have also arbitrarily selected one of the many possible Spanish names for fishes which could be used to translate English *sea-bass*.

lexical sample

It is useful to distinguish two variants of the generic WSD task. In the **lexical sample** task, a small pre-selected set of target words is chosen, along with an inventory of senses for each word from some lexicon. Since the set of words and the set of senses is small, **supervised machine learning** approaches are often used to handle lexical sample tasks. For each word, a number of corpus instances (context sentences) can be selected and hand-labeled with the correct sense of the target word in each. Classifier systems can then be trained using these labeled examples. Unlabeled target words in context can then be labeled using such a trained classifier. Early work in word sense disambiguation focused solely on lexical sample tasks of this sort, building word-specific algorithms for disambiguating single words like *line*, *interest*, or *plant*.

all-words

In contrast, in the **all-words** task systems are given entire texts and a lexicon with an inventory of senses for each entry, and are required to disambiguate every content word in the text. The all-words task is very similar to part-of-speech tagging, except with a much larger set of tags, since each lemma has its own set. A consequence of this larger set of tags is a serious data sparseness problem; there is unlikely to be adequate training data for every word in the test set. Moreover, given the number of polysemous words in reasonably-sized lexicons, approaches based on training one classifier per term are unlikely to be practical.

In the following sections we explore the application of various machine learning paradigms to word sense disambiguation. We begin with supervised learning, followed by a section on how systems are standardly evaluated. We then turn to a variety of methods for dealing with the lack of sufficient day for fully-supervised training, including dictionary-based approaches and bootstrapping techniques.

Finally, after we have introduced the necessary notions of distributional word similarity in Sec. 20.7, we return in Sec. 20.10 to the problem of unsupervised approaches to sense disambiguation.

20.2 Supervised Word Sense Disambiguation

If we have data which has been hand-labeled with correct word senses, we can use a **supervised learning** approach to the problem of sense disambiguation. extracting features from the text that are helpful in predicting particular senses, and then training a classifier to assign the correct sense given these features. The output of training is thus a classifier system capable of assigning sense labels to unlabeled words in context.

For **lexical sample** tasks, there are various labeled corpora for individual words, consisting of context sentences labeled with the correct sense for the target word. These include the *line-hard-serve* corpus containing 4,000 sense-tagged examples of *line* as a noun, *hard* as an adjective and *serve* as a verb (Leacock et al., 1993) , and the *interest* corpus with 2,369 sense-tagged examples of *interest* as a noun (Bruce and Wiebe, 1994). The SENSEVAL project has also produced a number of such sense-labeled lexical sample corpora (SENSEVAL-1 with 34 words from the HECTOR lexicon and corpus (Kilgarriff and Rosenzweig, 2000; Atkins, 1993), SENSEVAL-2 and -3 with 73 and 57 target words, respectively (Palmer et al., 2001b; Kilgarriff, 2001)).

semantic concordance

For training **all-word** disambiguation tasks we use a **semantic concordance**, a

corpus in which each open-class word in each sentence is labeled with its word sense from a specific dictionary or thesaurus. One commonly used corpus is SemCor, a subset of the Brown Corpus consisting of over 234,000 words which were manually tagged with WordNet senses (Miller et al., 1993; Landes et al., 1998). In addition, sense-tagged corpora have been built for the SENSEVAL all-word tasks. The SENSEVAL-3 English all-words test data consisted of 2081 tagged content word tokens, from 5,000 total running words of English from the WSJ and Brown corpora (Palmer et al., 2001b).

20.2.1 Extracting Feature Vectors for Supervised Learning

The first step in supervised training is to extract a useful set of features that are predictive of word senses. As Ide and Véronis (1998b) point out, the insight that underlies all modern algorithms for word sense disambiguation was first articulated by Weaver (1955) in the context of machine translation:

> If one examines the words in a book, one at a time as through an opaque mask with a hole in it one word wide, then it is obviously impossible to determine, one at a time, the meaning of the words. [...] But if one lengthens the slit in the opaque mask, until one can see not only the central word in question but also say N words on either side, then if N is large enough one can unambiguously decide the meaning of the central word. [...] The practical question is : "What minimum value of N will, at least in a tolerable fraction of cases, lead to the correct choice of meaning for the central word?"

To extract useful features from such a window, a minimal amount of processing is first performed on the sentence containing the window. This processing varies from approach to approach but typically includes part-of-speech tagging, lemmatization or stemming, and in some cases syntactic parsing to reveal information such as head words and dependency relations. Context features relevant to the target word can then be extracted from this enriched input. A **feature vector** consisting of numeric or nominal values is used to encode this linguistic information as an input to most machine learning algorithms.

feature vector

Two classes of features are generally extracted from these neighboring contexts: collocational features and bag-of-words features. A **collocation** is a word or phrase in a position-specific relationship to a target word (i.e., exactly one word to the right, or exactly 4 words to the left, and so on). Thus **collocational features** encode information about *specific* positions located to the left or right of the target word. Typical features extracted for these context words include the word itself, the root form of the word, and the word's part-of-speech. Such features are effective at encoding local lexical and grammatical information that can often accurately isolate a given sense.

collocation

collocational features

As an example of this type of feature-encoding, consider the situation where we need to disambiguate the word *bass* in the following WSJ sentence:

(20.1) An electric guitar and **bass** player stand off to one side, not really part of the scene, just as a sort of nod to gringo expectations perhaps.

A collocational feature-vector, extracted from a window of two words to the right and left of the target word, made up of the words themselves and their respective parts-of-

speech, i.e.,

$$[w_{i-2}, \text{POS}_{i-2}, w_{i-1}, \text{POS}_{i-1}, w_{i+1}, \text{POS}_{i+1}, w_{i+2}, \text{POS}_{i+2}] \quad (20.2)$$

would yield the following vector:

```
[guitar, NN, and, CC, player, NN, stand, VB]
```

bag-of-words

The second type of feature consists of **bag-of-words** information about neighboring words. A **bag-of-words** means an unordered set of words, ignoring their exact position. The simplest bag-of-words approach represents the context of a target word by a vector of features, each binary feature indicating whether a vocabulary word w does or doesn't occur in the context. This vocabulary is typically preselected as some useful subset of words in a training corpus. In most WSD applications, the context region surrounding the target word is generally a small symmetric fixed size window with the target word at the center. Bag-of-word features are effective at capturing the general topic of the discourse in which the target word has occurred. This, in turn, tends to identify senses of a word that are specific to certain domains. We generally don't use stop-words as features, and may also limit the bag-of-words to only consider a small number of frequently used content words.

For example a bag-of-words vector consisting of the 12 most frequent content words from a collection of *bass* sentences drawn from the WSJ corpus would have the following ordered word feature set:

[*fishing, big, sound, player, fly, rod, pound, double, runs, playing, guitar, band*]

Using these word features with a window size of 10, example (20.1) would be represented by the following binary vector:

```
[0,0,0,1,0,0,0,0,0,0,1,0]
```

We'll revisit the bag-of-words technique in Ch. 23 where we'll see that it forms the basis for the **vector space model** of search in modern search engines.

Most approaches to sense disambiguation use both collocational and bag-of-words features, either by joining them into one long vector, or by building a distinct classifier for each feature type, and combining them in some manner.

20.2.2 Naive Bayes and Decision List Classifiers

Given training data together with the extracted features, any supervised machine learning paradigm can be used to train a sense classifier. We will restrict our discussion here to the naive Bayes and decision list approaches, since they have been the focus of considerable work in word sense disambiguation and have not yet been introduced in previous chapters.

naive Bayes classifier

The **naive Bayes classifier** approach to WSD is based on the premise that choosing the best sense $\hat{s}$ out of the set of possible senses S for a feature vector $\vec{f}$ amounts to choosing the most probable sense given that vector. In other words:

$$\hat{s} = \underset{s \in S}{\text{argmax}} P(s|\vec{f}) \quad (20.3)$$

As is almost always the case, it would be difficult to collect reasonable statistics for this equation directly. To see this, consider that a simple binary bag of words vector defined

over a vocabulary of 20 words would have 2^{20} possible feature vectors. It's unlikely that any corpus we have access to will provide coverage to adequately train this kind of feature vector. To get around this problem we first reformulate our problem in the usual Bayesian manner as follows:

$$\hat{s} = \underset{s \in S}{\text{argmax}} \frac{P(\vec{f}|s)P(s)}{P(\vec{f})} \tag{20.4}$$

Even this equation isn't helpful enough, since the data available that associates specific vectors $\vec{f}$ with each sense s is also too sparse. However, what is available in greater abundance in a tagged training set is information about individual feature-value pairs in the context of specific senses. Therefore, we can make the independence assumption that gives this method its name, and that has served us well in part-of-speech tagging, speech recognition, and probabilistic parsing — **naively** assume that the features are independent of one another. Making this assumption that the features are **conditionally independent given the word sense** yields the following approximation for $P(\vec{f}|s)$:

$$P(\vec{f}|s) \approx \prod_{j=1}^{n} P(f_j|s) \tag{20.5}$$

In other words, we can estimate the probability of an entire vector given a sense by the product of the probabilities of its individual features given that sense. Since $P(\vec{f})$ is the same for all possible senses, it does not effect the final ranking of senses, leaving us with the following formulation of a **naive Bayes classifier for WSD**:

$$\hat{s} = \underset{s \in S}{\text{argmax}} P(s) \prod_{j=1}^{n} P(f_j|s) \tag{20.6}$$

Given this equation, **training** a naive Bayes classifier consists of estimating each of these probabilities. (20.6) first requires an estimate for the prior probability of each sense $P(s)$. We get the maximum likelihood estimate of this probability from the sense-tagged training corpus by counting the number of times the sense s_i occurs and dividing by the total count of the target word w_j (i.e. the sum of the instances of each sense of the word). That is:

$$P(s_i) = \frac{\text{count}(s_i, w_j)}{\text{count}(w_j)} \tag{20.7}$$

We also need to know each of the individual feature probabilities $P(f_j|s)$. The maximum likelihood estimate for these would be:

$$P(f_j|s) = \frac{\text{count}(f_j, s)}{\text{count}(s)} \tag{20.8}$$

Thus, if a collocational feature such as $[w_{i-2} = \text{guitar}]$ occurred 3 times for sense bass^1, and sense bass^1 itself occurred 60 times in training, the MLE estimate is $P(f_j|s) = 0.05$. Binary bag-of-word features are treated in a similar manner; we simply count the number of times a given vocabulary item is present with each of the possible senses and divide by the count for each sense.

Rule		Sense
fish within window	⇒	**bass**^{1}
striped bass	⇒	**bass**^{1}
guitar within window	⇒	**bass**^{2}
bass player	⇒	**bass**^{2}
piano within window	⇒	**bass**^{2}
tenor within window	⇒	**bass**^{2}
sea bass	⇒	**bass**^{1}
play/V *bass*	⇒	**bass**^{2}
river within window	⇒	**bass**^{1}
violin within window	⇒	**bass**^{2}
salmon within window	⇒	**bass**^{1}
on bass	⇒	**bass**^{2}
bass are	⇒	**bass**^{1}

Figure 20.2 An abbreviated decision list for disambiguating the fish sense of bass from the music sense (Yarowsky, 1997).

With the necessary estimates in place, we can assign senses to words in context by applying Eq. 20.6. More specifically, we take the target word in context, extract the specified features, compute $P(s)\prod_{j=1}^{n} P(f_j|s)$ for each sense, and return the sense associated with the highest score. Note that in practice, the probabilities produced for even the highest scoring senses will be dangerously low due to the various multiplications involved; mapping everything to log-space and instead performing additions is the usual solution.

The use of a simple maximum likelihood estimator means that in testing, when a target word cooccurs with a word that it did not cooccur with in training, all of its senses will receive a probability of zero. Smoothing is therefore essential to the whole enterprise. Naive Bayes approaches to sense disambiguation generally use the simple Laplace (add-one or add-k) smoothing discussed in Ch. 4.

One problem with naive Bayes and some other classifiers is that it's hard for humans to examine their workings and understand their decisions. Decision lists and decision trees are somewhat more transparent approaches that lend themselves to inspection. **Decision list classifiers** are equivalent to simple case statements in most programming languages. In a decision list classifier, a sequence of tests is applied to each target word feature vector. Each test is indicative of a particular sense. If a test succeeds, then the sense associated with that test is returned. If the test fails, then the next test in the sequence is applied. This continues until the end of the list, where a default test simply returns the majority sense.

Decision list classifiers

Fig. 20.2 shows a portion of a decision list for the task of discriminating the fish sense of *bass* from the music sense. The first test says that if the word *fish* occurs anywhere within the input context then **bass**^{1} is the correct answer. If it doesn't then each of the subsequent tests is consulted in turn until one returns true; as with case statements a default test that returns true is included at the end of the list.

Learning a decision list classifier consists of generating and ordering individual tests based on the characteristics of the training data. There are a wide number of

methods that can be used to create such lists. In the approach used by Yarowsky (1994) for binary homonym discrimination, each individual feature-value pair constitutes a test. We can measure how much a feature indicates a particular sense by computing the log-likelihood of the sense given the feature. The ratio between the log-likelihoods of the two senses tells us how discriminative a feature is between senses:

$$\left|\text{Log}\left(\frac{P(Sense_1|f_i)}{P(Sense_2|f_i)}\right)\right| \tag{20.9}$$

The decision list is then created from these tests by simply ordering the tests in the list according to the log-likelihood ratio. Each test is checked in order and returns the appropriate sense. This training method differs quite a bit from standard decision list learning algorithms. For the details and theoretical motivation for these approaches see Rivest (1987) or Russell and Norvig (1995).

20.3 WSD Evaluation, Baselines, and Ceilings

Evaluating component technologies like WSD is always a complicated affair. In the long term, we're primarily interested in the extent to which they improve performance in some end-to-end application such as information retrieval, question answering or machine translation. Evaluating component NLP tasks embedded in end-to-end applications is called **extrinsic evaluation**, **task-based** evaluation, **end-to-end** evaluation, or **in vivo** evaluation. It is only with extrinsic evaluation that we can tell if a technology such as WSD is working in the sense of actually improving performance on some real task.

extrinsic evaluation
in vivo

Extrinsic evaluations are much more difficult and time-consuming to implement, however, since they require integration into complete working systems. Furthermore, an extrinsic evaluation may only tell us something about WSD in the context of the application, and may not generalize to other applications.

For these reasons, WSD systems are typically developed and evaluated intrinsically. In **intrinsic** or **in vitro** we treat a WSD component as if it were a stand-alone system operating independently of any given application. In this style of evaluation, systems are evaluated either using exact match **sense accuracy**: the percentage of words that are tagged identically with the hand-labeled sense tags in a test set; or with standard precision and recall measures if systems are permitted to pass on labeling some instances. In general, we evaluate using held out data from the same sense-tagged corpora that we used for training, such as the SemCor corpus discussed above, or the various corpora produced by the SENSEVAL effort.

intrinsic
in vitro
sense accuracy

Many aspects of sense evaluation have been standardized by the SENSEVAL/SEMEVAL efforts (Palmer et al., 2006; Kilgarriff and Palmer, 2000). This framework provides a shared task with training and testing materials along with sense inventories for all-words and lexical sample tasks in a variety of languages.

Whichever WSD task we are performing, we ideally need two additional measures to assess how well we're doing: a baseline measure to tell use how well we're doing as

compared to relatively simple approaches, and a ceiling to tell us how close we are to optimal performance.

most frequent sense

The simplest baseline is to choose the **most frequent sense** for each word (Gale et al., 1992b) from the senses in a labeled corpus. For WordNet, this corresponds to the **take the first sense** heuristic, since senses in WordNet are generally ordered from most-frequent to least-frequent. WordNet sense frequencies come from the SemCor sense-tagged corpus described above.

take the first sense

Unfortunately, many WordNet senses do not occur in SemCor; these unseen senses are thus ordered arbitrarily after those that do. The four WordNet senses of the noun *plant*, for example, are as follows:

Freq	Synset	Gloss
338	plant[1], works, industrial plant	buildings for carrying on industrial labor
207	plant[2], flora, plant life	a living organism lacking the power of locomotion
2	plant[3]	something planted secretly for discovery by another
0	plant[4]	an actor situated in the audience whose acting is rehearsed but seems spontaneous to the audience

The most frequent sense baseline can be quite accurate, and is therefore often used as a default, to supply a word sense when a supervised algorithm has insufficient training data. A second commonly used baseline is the **Lesk algorithm**, discussed in the next section.

Human inter-annotator agreement is generally considered as a ceiling, or upper bound, for sense disambiguation evaluations. Human agreement is measured by comparing the annotations of two human annotators on the same data given the same tagging guidelines. The ceiling (inter-annotator agreement) for many all-words corpora using WordNet-style sense inventories seems to range from about 75% to 80% (Palmer et al., 2006). Agreement on more coarse grained, often binary, sense inventories is closer to 90% (Gale et al., 1992b).

While using hand-labeled test sets is the best current method for evaluation, labeling large amounts of data is still quite expensive. For supervised approaches, we need this data anyhow for training so the effort to label large amounts of data seems justified. But for unsupervised algorithms like those we will discuss in Sec. 20.10, it would be nice to have an evaluation method that avoided hand labeling. The use of **pseudowords** is one such simplified evaluation method (Gale et al., 1992a; Schütze, 1992a). A pseudoword is an artificial word created by concatenating two randomly-chosen words together (e.g., *banana* and *door* to create *banana-door*.) Each occurrence of the two words in the test set is replaced by the new concatenation, creating a new 'word' which is now ambiguous between the senses *banana* and *door*. The 'correct sense' is defined by the original word, and so we can apply our disambiguation algorithm and compute accuracy as usual. In general, pseudowords give an overly optimistic measure of performance, since they are a bit easier to disambiguate than average ambiguous words. This is because the different senses of real words tend to be similar, while pseudowords are generally not semantically similar, acting like homonymous but not polysemous words (Gaustad, 2001). Nakov and Hearst (2003) shows that it is possible to improve the accuracy of pseudoword evaluation by more carefully choosing the pseudowords.

pseudowords

20.4 WSD: Dictionary and Thesaurus Methods

Supervised algorithms based on sense-labeled corpora are the best performing algorithms for sense disambiguation. However, such labeled training data is expensive and limited and supervised approaches fail on words not in the training data. Thus this section and the next describe different ways to get indirect supervision from other sources. In this section, we describe methods for using a dictionary or thesaurus as an indirect kind of supervision; the next section describes bootstrapping approaches.

20.4.1 The Lesk Algorithm

Lesk algorithm

Simplified Lesk

By far the most well-studied dictionary-based algorithm for sense disambiguation is the **Lesk algorithm**, really a family of algorithms that choose the sense whose dictionary gloss or definition shares the most words with the target word's neighborhood. Fig. 20.3 shows the simplest version of the algorithm, often called the **Simplified Lesk** algorithm (Kilgarriff and Rosenzweig, 2000).

```
function SIMPLIFIED LESK(word, sentence) returns best sense of word

  best-sense ← most frequent sense for word
  max-overlap ← 0
  context ← set of words in sentence
  for each sense in senses of word do
    signature ← set of words in the gloss and examples of sense
    overlap ← COMPUTEOVERLAP(signature, context)
    if overlap > max-overlap then
        max-overlap ← overlap
        best-sense ← sense
  end
  return(best-sense)
```

Figure 20.3 The Simplified Lesk Algorithm. The COMPUTEOVERLAP function returns the number of words in common between two sets, ignoring function words or other words on a stop list. The original Lesk algorithm defines the *context* in a more complex way. The *Corpus Lesk* algorithm weights each overlapping word w by its $-\log P(w)$, and includes labeled training corpus data in the *signature*.

As an example of the Lesk algorithm at work, consider disambiguating the word *bank* in the following context:

(20.10) The **bank** can guarantee deposits will eventually cover future tuition costs because it invests in adjustable-rate mortgage securities.

given the following two WordNet senses:

bank[1]	Gloss:	a financial institution that accepts deposits and channels the money into lending activities
	Examples:	"he cashed a check at the bank", "that bank holds the mortgage on my home"
bank[2]	Gloss:	sloping land (especially the slope beside a body of water)
	Examples:	"they pulled the canoe up on the bank", "he sat on the bank of the river and watched the currents"

Sense **bank**[1] has two (non-stop) words overlapping with the context in (20.10): *deposits* and *mortgage*, while sense bank[2] has zero, so sense **bank**[1] is chosen.

There are many obvious extensions to Simplified Lesk. The original Lesk algorithm (Lesk, 1986) is slightly more indirect. Instead of comparing a target word's signature with the context words, the target signature is compared with the signatures of each of the context words. For example, consider Lesk's example of selecting the appropriate sense of *cone* in the phrase *pine cone* given the following definitions for *pine* and *cone*.

pine	1	kinds of evergreen tree with needle-shaped leaves
	2	waste away through sorrow or illness
cone	1	solid body which narrows to a point
	2	something of this shape whether solid or hollow
	3	fruit of certain evergreen trees

In this example, Lesk's method would select **cone**[3] as the correct sense since two of the words in its entry, *evergreen* and *tree*, overlap with words in the entry for *pine*, whereas neither of the other entries have any overlap with words in the definition of *pine*. In general Simplified Lesk seems to work better than original Lesk.

The primary problem with either the original or simplified approaches, however, is that the dictionary entries for the target words are short, and may not provide enough chance of overlap with the context.[2] One remedy is to expand the list of words used in the classifier to include words related to, but not contained in their individual sense definitions. But the best solution, if any sense-tagged corpus data like SemCor is available, is to add all the words in the labeled corpus sentences for a word sense into the signature for that sense. This version of the algorithm, the **Corpus Lesk** algorithm is the best-performing of all the Lesk variants (Kilgarriff and Rosenzweig, 2000; Vasilescu et al., 2004) and is used as a baseline in the SENSEVAL competitions. Instead of just counting up the overlapping words, the **Corpus Lesk** algorithm also applies a weight to each overlapping word. The weight is the **inverse document frequency** or **IDF**, a standard information-retrieval measure to be introduced in Ch. 23. IDF measures how many different 'documents' (in this case glosses and examples) a word occurs in (Ch. 23) and is thus a way of discounting function words. Since function words like *the*, *of*, etc, occur in many documents, their IDF is very low, while the IDF of content words is high. Corpus Lesk thus uses IDF instead of a stoplist.

Corpus Lesk

inverse document frequency

IDF

Formally the IDF for a word i can be defined as

$$\text{idf}_i = \log\left(\frac{Ndoc}{nd_i}\right) \tag{20.11}$$

[2] Indeed, Lesk (1986) notes that the performance of his system seems to roughly correlate with the length of the dictionary entries.

where $Ndoc$ is the total number of 'documents' (glosses and examples) and nd_i is the number of these documents containing word i.

Finally, it is possible to combine the Lesk and supervised approaches, by adding new Lesk-like bag-of-words features. For example, the glosses and example sentences for the target sense in WordNet could be used to compute the supervised bag-of-words features instead of (or in addition to) the words in the SemCor context sentence for the sense (Yuret, 2004).

20.4.2 Selectional Restrictions and Selectional Preferences

One of the earliest knowledge-sources for sense disambiguation is the notion of **selectional restrictions** defined in Ch. 19. For example the verb *eat* might have a restriction that its THEME argument be `[+FOOD]`. In early systems, selectional restrictions were used to rule out senses that violate the selectional restrictions of neighboring words (Katz and Fodor, 1963; Hirst, 1987). Consider the following pair of WSJ examples of the word *dish*:

(20.12) "In our house, everybody has a career and none of them includes washing **dishes**," he says.

(20.13) In her tiny kitchen at home, Ms. Chen works efficiently, stir-frying several simple **dishes**, including braised pig's ears and chicken livers with green peppers.

These correspond to WordNet **dish**^{1} (a piece of dishware normally used as a container for holding or serving food), with hypernyms like *artifact*, and **dish**^{2} (a particular item of prepared food) with hypernyms like *food*.

The fact that we perceive no ambiguity in these examples can be attributed to the selectional restrictions imposed by *wash* and *stir-fry* on their THEME semantic roles. The restrictions imposed by *wash* (perhaps [+WASHABLE]) conflict with **dish**^{2}. The restrictions on *stir-fry* ([+EDIBLE]) conflict with **dish**^{1}. In early systems, the predicate strictly selected the correct sense of an ambiguous argument by eliminating the sense that fails to match one of its selectional restrictions. But such hard constraints have a number of problems. The main problem is that selectional restriction violations often occur in well-formed sentences, either because they are negated as in (20.14), or because selectional restrictions are overstated as in (20.15):

(20.14) But it fell apart in 1931, perhaps because people realized you can't **eat** gold for lunch if you're hungry.

(20.15) In his two championship trials, Mr. Kulkarni **ate** glass on an empty stomach, accompanied only by water and tea.

As Hirst (1987) observes, examples like these often result in the elimination of all senses, bringing semantic analysis to a halt. Modern models thus adopt the view of selectional restrictions as preferences, rather than rigid requirements. Although there have been many instantiations of this approach over the years (e.g., Wilks, 1975c, 1975b, 1978), we'll discuss a member of the popular probabilistic or information-theoretic family of approaches: Resnik's (1997) model of **selectional association**.

selectional preference strength

Resnik first defines the **selectional preference strength** as the general amount of

information that a predicate tells us about the semantic class of its arguments. For example, the verb *eat* tells us a lot about the semantic class of its direct object, since they tend to be edible. The verb *be*, by contrast, tells us less about its direct objects. The selectional preference strength can be defined by the difference in information between two distributions: the distribution of expected semantic classes $P(c)$ (how likely is it that a direct object will fall into class c) and the distribution of expected semantic classes for the particular verb $P(c|v)$ (how likely is it that the direct object of specific verb v will fall into semantic class c). The greater the difference between these distributions, the more information the verb is giving us about possible objects. This difference can be quantified by the **relative entropy** between these two distributions, or **Kullback-Leibler divergence** (Kullback and Leibler, 1951). The Kullback-Leibler or KL divergence $D(P||Q)$ can be used to express the difference between two probability distributions P and Q, and will be discussed further when we discuss word similarity in Eq. 20.50.

relative entropy
Kullback-Leibler divergence

$$D(P||Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)} \tag{20.16}$$

The selectional preference $S_R(v)$ uses the KL divergence to express how much information, in bits, the verb v expresses about the possible semantic class of its argument.

$$\begin{aligned} S_R(v) &= D(P(c|v)||P(c)) \\ &= \sum_c P(c|v) \log \frac{P(c|v)}{P(c)} \end{aligned} \tag{20.17}$$

selectional association

Resnik then defines the **selectional association** of a particular class and verb as the relative contribution of that class to the general selectional preference of the verb:

$$A_R(v,c) = \frac{1}{S_R(p)} P(c|v) \log \frac{P(c|v)}{P(c)} \tag{20.18}$$

The selectional association is thus a probabilistic measure of the strength of association between a predicate and a class dominating the argument to the predicate. Resnik estimates the probabilities for these associations by parsing a corpus, counting all the times each predicate occurs with each argument word, and assuming that each word is a partial observation of all the WordNet concepts containing the word. The following table from Resnik (1996) shows some sample high and low selectional associations for verbs and some WordNet semantic classes of their direct objects.

Verb	Direct Object Semantic Class	Assoc	Direct Object Semantic Class	Assoc
read	WRITING	6.80	ACTIVITY	-.20
write	WRITING	7.26	COMMERCE	0
see	ENTITY	5.79	METHOD	-0.01

Resnik (1998) shows that these selectional associations can be used to perform a limited form of word sense disambiguation. Roughly speaking the algorithm selects as

the correct sense for an argument the one that has the highest selectional association between one of its ancestor hypernyms and the predicate.

While we have presented only the Resnik model of selectional preferences, there are other more recent models, using probabilistic methods and using other relations than just direct object; see the end of the chapter for a brief summary. In general, selectional restriction approaches perform as well as other unsupervised approaches at sense disambiguation, but not as well as Lesk or as supervised approaches.

20.5 Minimally Supervised WSD: Bootstrapping

bootstrapping

Yarowsky algorithm

Both the supervised approach and the dictionary-based approach to WSD require large hand-built resources; supervised training sets in one case, large dictionaries in the other. We can instead use **bootstrapping** algorithms, often called **semi-supervised learning** or **minimally supervised learning**, which need only a very small hand-labeled training set. The most widely emulated bootstrapping algorithm for WSD is the **Yarowsky algorithm** (Yarowsky, 1995).

The goal of the Yarowsky algorithm is to learn a classifier for a target word (in a lexical-sample task). The algorithm is given a small seed-set Λ_0 of labeled instances of each sense, and a much larger unlabeled corpus V_0. The algorithm first trains an initial decision-list classifier on the seed-set Λ_0. It then uses this classifier to label the unlabeled corpus V_0. The algorithm then selects the examples in V_0 that it is most confident about, removes them, and adds them to the training set (call it now Λ_1). The algorithm then trains a new decision list classifier (a new set of rules) on Λ_1, and iterates by applying the classifier to the now-smaller unlabeled set V_1, extracting a new training set Λ_2 and so on. With each iteration of this process, the training corpus grows and the untagged corpus shrinks. The process is repeated until some sufficiently low error-rate on the training set is reached, or until no further examples from the untagged corpus are above threshold.

The key to any bootstrapping approach lies in its ability to create a larger training set from a small set of seeds. This requires an accurate initial set of seeds and a good confidence metric for picking good new examples to add to the training set. The confidence metric used by Yarowsky (1995) is the measure described earlier in Sec. 20.2.2, the log-likelihood ratio of the decision-list rule that classified the example.

One Sense per Collocation

One way to generate the initial seeds is to hand-label a small set of examples (Hearst, 1991). Instead of hand-labeling, it is also possible to use a heuristic to automatically select accurate seeds. Yarowsky (1995) used the **One Sense per Collocation** heuristic, which relies on the intuition that certain words or phrases strongly associated with the target senses tend not to occur with the other sense. Yarowsky defines his seed set by choosing a single collocation for each sense. As an illustration of this technique, consider generating seed sentences for the fish and musical senses of *bass*. Without too much thought, we might come up with *fish* as a reasonable indicator of **bass**[1], and *play* as a reasonable indicator of **bass**[2]. Fig. 20.5 shows a partial result of such a search for the strings "fish" and "play" in a corpus of *bass* examples drawn from the WSJ.

We can also suggest collocates automatically, for example extracting words from

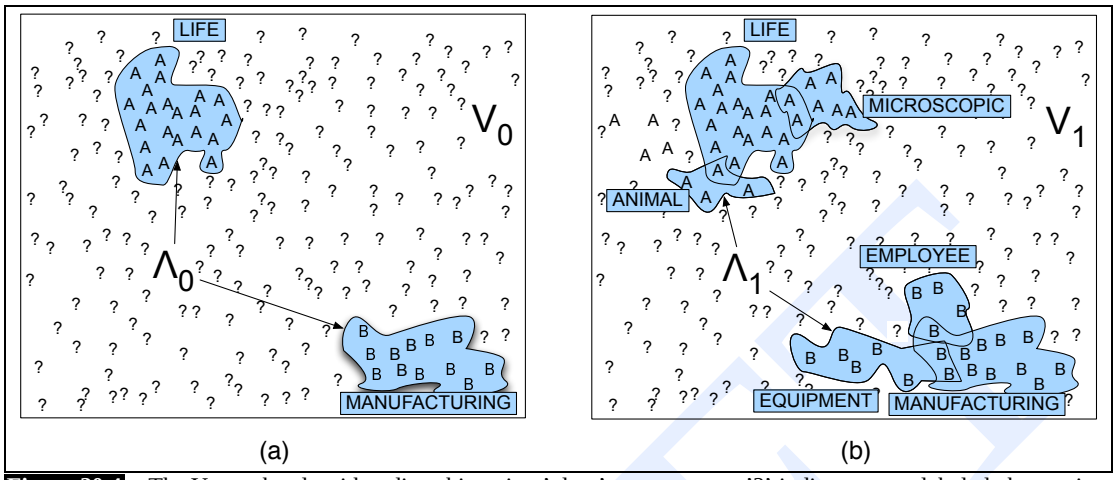

Figure 20.4 The Yarowsky algorithm disambiguating 'plant' at two stages; '?' indicates an unlabeled observation, A and B are observations labeled as SENSE-A or SENSE-B. 'LIFE' indicates observations occur with collocate "life". The initial stage (a) shows only seed sentences Λ_0 labeled by collocates ('life' and 'manufacturing'). An intermediate stage is shown in (b) where more collocates have been discovered ('equipment', 'microscopic', etc) and more instances in V_0 have been moved into Λ_1, leaving a smaller unlabeled set V_1. Figure adapted from Yarowsky (1995).

We need more good teachers – right now, there are only a half a dozen who can **play** the free **bass** with ease.

An electric guitar and **bass play**er stand off to one side, not really part of the scene, just as a sort of nod to gringo expectations perhaps.

When the New Jersey Jazz Society, in a fund-raiser for the American Jazz Hall of Fame, honors this historic night next Saturday, Harry Goodman, Mr. Goodman's brother and **bass play**er at the original concert, will be in the audience with other family members.

The researchers said the worms spend part of their life cycle in such **fish** as Pacific salmon and striped **bass** and Pacific rockfish or snapper.

And it all started when **fish**ermen decided the striped **bass** in Lake Mead were too skinny.

Though still a far cry from the lake's record 52-pound **bass** of a decade ago, "you could fillet these **fish** again, and that made people very, very happy," Mr. Paulson says.

Figure 20.5 Samples of *bass* sentences extracted from the WSJ using the simple correlates *play* and *fish*.

machine readable dictionary entries, and selecting seeds using collocational statistics such as those described in Sec. 20.7 (Yarowsky, 1995).

One Sense Per Discourse

The original Yarowsky algorithm also makes use of a second heuristic, called **One Sense Per Discourse**, based on the work of Gale et al. (1992c), who noticed that a particular word appearing multiple times in a text or discourse often appeared with the same sense. Yarowsky (1995), for example, showed in a corpus of 37,232 examples that every time the word *bass* occurred more than once in a discourse, that it occurred in only the *fish* or only the *music* coarse-grain sense throughout the discourse. The validity of this heuristic depends on the granularity of the sense inventory and is not valid

in every discourse situation; it seems to be true mostly for coarse-grain senses, and particularly for cases of homonymy rather than polysemy (Krovetz, 1998). Nonetheless, it has still been useful in a number of unsupervised and semi-supervised sense disambiguation situations.

20.6 Word Similarity: Thesaurus Methods

We turn now to the computation of various semantic relations that hold between words. We saw in Ch. 19 that such relations include synonymy, antonymy, hyponymy, hypernymy, and meronymy. Of these, the one that has been most computationally developed and has the greatest number of applications is the idea of word **synonymy** and **similarity**.

word similarity
semantic distance

Synonymy is a binary relation between words; two words are either synonyms or not. For most computational purposes we use instead a looser metric of **word similarity** or **semantic distance**. Two words are more similar if they share more features of meaning, or are near-synonyms. Two words are less similar, or have greater semantic distance, if they have fewer common meaning elements. Although we have described them as relations between words, synonymy, similarity, and distance are actually relations between word *senses*. For example of the two senses of *bank*, we might say that the financial sense is similar to one of the senses of *fund* while the riparian sense is more similar to one of the senses of *slope*. In the next few sections of this chapter, we will need to compute these relations over both words and senses.

The ability to compute word similarity is a useful part of many language understanding applications. In **information retrieval** or **question answering** we might want to retrieve documents whose words have similar meanings to the query words. In **summarization**, **generation**, and **machine translation**, we need to know whether two words are similar to know if we can substitute one for the other in particular contexts. In **language modeling**, we can use semantic similarity to cluster words for class-based models. One interesting class of applications for word similarity is automatic grading of student responses. For example algorithms for **automatic essay grading** use word similarity to determine if an essay is similar in meaning to a correct answer. We can also use word-similarity as part of an algorithm to *take* an exam, such as a multiple-choice vocabulary test. Automatically taking exams is useful in test designs in order to see how easy or hard a particular multiple-choice question or exam is.

There are two classes of algorithms for measuring word similarity. This section focuses on **thesaurus-based** algorithms, in which we measure the distance between two senses in an on-line thesaurus like WordNet or MeSH. The next section focuses on **distributional** algorithms, in which we estimate word similarity by finding words that have similar distributions in a corpus.

The thesaurus-based algorithms use the structure of the thesaurus to define word similarity. In principle we could measure similarity using any information available in a thesaurus (meronymy, glosses, etc). In practice, however, thesaurus-based word similarity algorithms generally use only the hypernym/hyponym (*is-a* or subsumption) hierarchy. In WordNet, verbs and nouns are in separate hypernym hierarchies, so a

thesaurus-based algorithm for WordNet can thus only compute noun-noun similarity, or verb-verb similarity; we can't compare nouns to verbs, or do anything with adjectives or other parts of speech.

word relatedness

Resnik (1995) and Budanitsky and Hirst (2001) draw the important distinction between **word similarity** and **word relatedness**. Two words are similar if they are near-synonyms, or roughly substitutable in context. Word relatedness characterizes a larger set of potential relationships between words; antonyms, for example, have high relatedness, but low similarity. The words *car* and *gasoline* are very related, but not similar, while *car* and *bicycle* are similar. Word similarity is thus a subcase of word relatedness. In general, the five algorithms we describe in this section do not attempt to distinguish between similarity and semantic relatedness; for convenience we will call them *similarity* measures, although some would be more appropriately described as relatedness measures; we return to this question in Sec. 20.8.

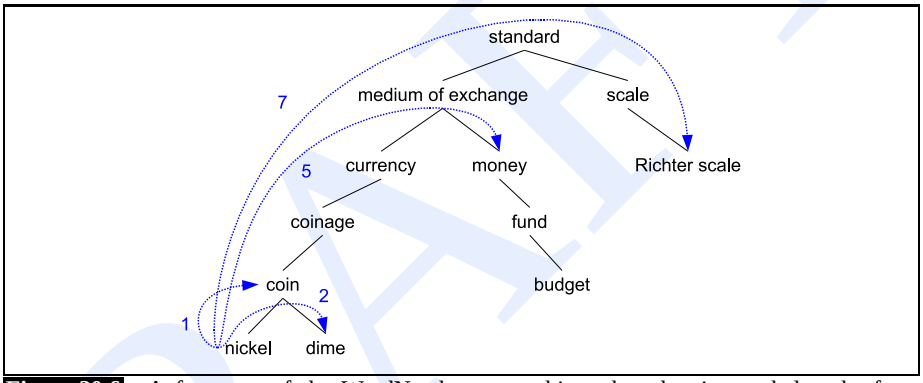

Figure 20.6 A fragment of the WordNet hypernym hierarchy, showing path lengths from *nickel* to *coin* (1), *dime* (2), *money* (5), and *Richter scale* (7).

The oldest and simplest thesaurus-based algorithms are based on the intuition that the shorter the **path** between two words or senses in the graph defined by the thesaurus hierarchy, the more similar they are. Thus a word/sense is very similar to its parents or its siblings, and less similar to words that are far away in the network. This notion can be operationalized by measuring the number of edges between the two concept nodes in the thesaurus graph. Fig. 20.6 shows an intuition; the concept *dime* is most similar to *nickel* and *coin*, less similar to *money*, and even less similar to *Richter scale*. Formally, we specify path length as follows:

$$\text{pathlen}(c_1, c_2) = \text{the number of edges in the shortest path in the thesaurus graph between the sense nodes } c_1 \text{ and } c_2$$

Path-based similarity can be defined just as the path length, often with a log transform (Leacock and Chodorow, 1998), resulting in the following common definition of **path-length based similarity**:

path-length based similarity

$$\text{sim}_{\text{path}}(c_1, c_2) = -\log \text{pathlen}(c_1, c_2) \tag{20.19}$$

For most applications, we don't have sense-tagged data, and thus we need our algorithm to give us the similarity between words rather than between senses or concepts.

For any of the thesaurus-based algorithms, following Resnik (1995), we can approximate the correct similarity (which would require sense disambiguation) by just using the pair of senses for the two words that results in maximum sense similarity. Thus based on sense similarity we can define **word similarity** as follows:

word similarity

$$\text{wordsim}(w_1, w_2) = \max_{\substack{c_1 \in \text{senses}(w_1) \\ c_2 \in \text{senses}(w_2)}} \text{sim}(c_1, c_2) \tag{20.20}$$

The basic path-length algorithm makes the implicit assumption that each link in the network represents a uniform distance. In practice, this assumption is not appropriate. Some links (for example those that are very deep in the WordNet hierarchy) often seem to represent an intuitively narrow distance, while other links (e.g., higher up in the WordNet hierarchy) represent an intuitively wider distance. For example, in Fig. 20.6, the distance from *nickel* to *money* (5) seems intuitively much shorter than the distance from *nickel* to an abstract word *standard*; the link between *medium of exchange* and *standard* seems wider than that between, say, *coin* and *coinage*.

It is possible to refine path-based algorithms with normalizations based on depth in the hierarchy (Wu and Palmer, 1994), but in general we'd like an approach which lets us represent the distance associated with each edge independently.

A second class of thesaurus-based similarity algorithms attempts to offer just such a fine-grained metric. These **information content word similarity** algorithms still rely on the structure of the thesaurus, but also add probabilistic information derived from a corpus.

information content

Using similar notions to those we introduced earlier to define soft selectional restrictions, let's first define $P(c)$, following Resnik (1995), as the probability that a randomly selected word in a corpus is an instance of concept c (i.e., a separate random variable, ranging over words, associated with each concept). This implies that $P(root) = 1$, since any word is subsumed by the root concept. Intuitively, the lower a concept in the hierarchy, the lower its probability. We train these probabilities by counting in a corpus; each word in the corpus counts as an occurrence of each concept that contains it. For example, in Fig. 20.6 above, an occurrence of the word *dime* would count toward the frequency of *coin*, *currency*, *standard*, etc. More formally, Resnik computes $P(c)$ as follows:

$$P(c) = \frac{\sum_{w \in \text{words}(c)} count(w)}{N} \tag{20.21}$$

where $\text{words}(c)$ is the set of words subsumed by concept c, and N is the total number of words in the corpus that are also present in the thesaurus.

Fig. 20.7, from Lin (1998b), shows a fragment of the WordNet concept hierarchy augmented with the probabilities $P(c)$.

We now need two additional definitions. First, following basic information theory, we define the information content (IC) of a concept c as:

$$\text{IC}(c) = -\log P(c) \tag{20.22}$$

lowest common subsumer
LCS

Second, we define the **lowest common subsumer** or **LCS** of two concepts:

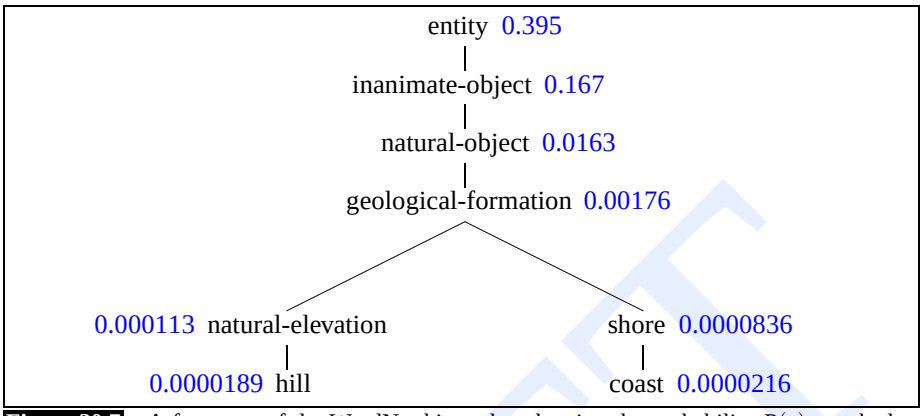

Figure 20.7 A fragment of the WordNet hierarchy, showing the probability $P(c)$ attached to each content, adapted from a figure from Lin (1998b)

$\text{LCS}(c_1, c_2)$ = the lowest common subsumer, i.e., the lowest node in the hierarchy that subsumes (is a hypernym of) both c_1 and c_2

There are now a number of ways to use the information content of a node in a word similarity metric. The simplest way was first proposed by Resnik (1995). We think of the similarity between two words as related to their common information; the more two words have in common, the more similar they are. Resnik proposes to estimate the common amount of information by the **information content of the lowest common subsumer of the two nodes**. More formally, the **Resnik similarity** measure is:

Resnik similarity

$$\text{sim}_{\text{resnik}}(c_1, c_2) = -\log P(\text{LCS}(c_1, c_2)) \tag{20.23}$$

Lin (1998b) extended the Resnik intuition by pointing out that a similarity metric between objects A and B needs to do more than measure the amount of information in common between A and B. For example, he pointed out that in addition, the more **differences** between A and B, the less similar they are. In summary:

- **commonality**: the more information A and B have in common, the more similar they are.
- **difference**: the more differences between the information in A and B, the less similar they are

Lin measures the commonality between A and B as the information content of the proposition that states the commonality between A and B:

$$\text{IC(Common(A,B))} \tag{20.24}$$

He measures the difference between A and B as

$$\text{IC(description(A,B))} - \text{IC(common(A,B))} \tag{20.25}$$

where description(A,B) describes A and B. Given a few additional assumptions about similarity, Lin proves the following theorem:

Similarity Theorem: The similarity between A and B is measured by the ratio between the amount of information needed to state the commonality of A and B and the information needed to fully describe what A and B are:

$$\text{sim}_{\text{Lin}}(A,B) = \frac{\log P(\text{common(A,B)})}{\log P(\text{description(A,B)})} \tag{20.26}$$

Applying this idea to the thesaurus domain, Lin shows (in a slight modification of Resnik's assumption) that the information in common between two concepts is twice the information in the lowest common subsumer $\text{LCS}(c_1,c_2)$. Adding in the above definitions of the information content of thesaurus concepts, the final **Lin similarity** function is:

Lin similarity

$$\text{sim}_{\text{Lin}}(c_1,c_2) = \frac{2 \times \log P(LCS(c_1,c_2))}{\log P(c_1) + \log P(c_2)} \tag{20.27}$$

For example, using sim_{lin}, Lin (1998b) shows that the similarity between the concepts of *hill* and *coast* from Fig. 20.7 is:

$$\text{sim}_{\text{Lin}}(\text{hill},\text{coast}) = \frac{2 \times \log P(\text{geological-formation})}{\log P(\text{hill}) + \log P(\text{coast}))} = 0.59 \tag{20.28}$$

Jiang-Conrath distance

A very similar formula, **Jiang-Conrath distance** (Jiang and Conrath, 1997) (although derived in a completely different way from Lin, and expressed as a distance rather than similarity function) has been shown to work as well or better than all the other thesaurus-based methods:

$$\text{dist}_{\text{JC}}(c_1,c_2) = 2 \times \log P(\text{LCS}(c_1,c_2)) - (\log P(c_1) + \log P(c_2)) \tag{20.29}$$

dist_{jc} can be transformed into a similarity by taking the reciprocal.

Finally, we describe a **dictionary-based** method, an extension of the Lesk algorithm for word-sense disambiguation described in Sec. 20.4.1. We call this a dictionary rather than a thesaurus method because it makes use of glosses, which are in general a property of dictionaries rather than thesauri (although WordNet does have glosses). Like the Lesk algorithm, the intuition of this **Extended Gloss Overlap**, or **Extended Lesk** measure (Banerjee and Pedersen, 2003) is that two concepts/senses in a thesaurus are similar if their glosses contain overlapping words. We'll begin by sketching an overlap function for two glosses. Consider these two concepts, with their glosses:

Extended Gloss Overlap
Extended Lesk

- *drawing paper:* paper that is specially prepared for use in drafting
- *decal:* the art of transferring designs from specially prepared paper to a wood or glass or metal surface.

For each n-word phrase that occurs in both glosses, Extended Lesk adds in a score of n^2 (the relation is non-linear because of the Zipfian relationship between lengths of phrases and their corpus frequencies; longer overlaps are rare so should be weighted more heavily). Here the overlapping phrases are *paper* and *specially prepared*, for a total similarity score of $1^2 + 2^2 = 5$.

Given such an overlap function, when comparing two concepts (synsets), Extended Lesk not only looks for overlap between their glosses, but also between the glosses of the senses which are hypernyms, hyponyms, meronyms, and other relations of the two concepts. For example if we just considered hyponyms, and defined gloss(hypo(A)) as the concatenation of all the glosses of all the hyponym senses of A, the total relatedness between two concepts A and B might be:

$$\begin{aligned}\text{similarity(A,B)} = \;& \text{overlap(gloss(A), gloss(B))} \\ & + \text{overlap(gloss(hypo(A)), gloss(hypo(B)))} \\ & + \text{overlap(gloss(A), gloss(hypo(B)))} \\ & + \text{overlap(gloss(hypo(A)),gloss(B))}\end{aligned}$$

Let RELS be the set of possible WordNet relations whose glosses we compare; assuming a basic overlap measure as sketched above, we can then define the **Extended Lesk** overlap measure as:

$$\text{sim}_{\text{eLesk}}(c_1,c_2) = \sum_{r,q \in \text{RELS}} \text{overlap}(\text{gloss}(r(c_1)), \text{gloss}(q(c_2))) \tag{20.30}$$

$$\begin{aligned}\text{sim}_{\text{path}}(c_1,c_2) &= -\log \text{pathlen}(c_1,c_2) \\ \text{sim}_{\text{Resnik}}(c_1,c_2) &= -\log P(\text{LCS}(c_1,c_2)) \\ \text{sim}_{\text{Lin}}(c_1,c_2) &= \frac{2 \times \log P(LCS(c_1,c_2))}{\log P(c_1) + \log P(c_2)} \\ \text{sim}_{\text{jc}}(c_1,c_2) &= \frac{1}{2 \times \log P(\text{LCS}(c_1,c_2)) - (\log P(c_1) + \log P(c_2))} \\ \text{sim}_{\text{eLesk}}(c_1,c_2) &= \sum_{r,q \in \text{RELS}} \text{overlap}(\text{gloss}(r(c_1)), \text{gloss}(q(c_2)))\end{aligned}$$

Figure 20.8 Five thesaurus-based (and dictionary-based) similarity measures.

Fig. 20.8 summarizes the five similarity measures we have described in this section The publicly available `Wordnet::Similarity` package implementing all these and other thesaurus-based word similarity measures is described in Pedersen et al. (2004).

Evaluating Thesaurus-based Similarity Which of these similarity measures is best? Word similarity measures have been evaluated in two ways. One intrinsic method is to compute the correlation coefficient between word similarity scores from an algorithm and word similarity ratings assigned by humans; such human ratings have been obtained for 65 word pairs by Rubenstein and Goodenough (1965), and 30 word pairs by Miller and Charles (1991). Another more extrinsic evaluation method is to embed the similarity measure in some end application like detection of **malapropisms** (real-word spelling errors) (Budanitsky and Hirst, 2006; Hirst and Budanitsky, 2005), or other

NLP applications like word-sense disambiguation (Patwardhan et al., 2003; McCarthy et al., 2004) and evaluate its impact on end-to-end performance. All of these evaluations suggest that all the above measures perform relatively well, and that of these, Jiang-Conrath similarity and Extended Lesk similarity are two of the best approaches, depending on the application.

20.7 Word Similarity: Distributional Methods

The previous section showed how to compute similarity between any two senses in a thesaurus, and by extension between any two words in the thesaurus hierarchy. But of course we don't have such thesauri for every language. Even for languages where we do have such resources, thesaurus-based methods have a number of limitations. The obvious limitation is that thesauri often lack words, especially new or domain-specific words. In addition, thesaurus-based methods only work if rich hyponymy knowledge is present in the thesaurus. While we have this for nouns, hyponym information for verbs tends to be much sparser, and doesn't exist at all for adjectives and adverbs. Finally, it is more difficult with thesaurus-based methods to compare words in different hierarchies, such as nouns with verbs.

For these reasons, methods which can automatically extract synonyms and other word relations from corpora have been developed. In this section we introduce such **distributional** methods, which can be applied directly to supply a word relatedness measure for NLP tasks. Distributional methods can also be used for **automatic thesaurus generation** for automatically populating or augmenting on-line thesauruses like WordNet with new synonyms and, as we will see in Sec. 20.8, with other relations like hyponymy and meronymy.

The intuition of distributional methods is that the meaning of a word is related to the distribution of words around it; in the famous dictum of Firth (1957), "You shall know a word by the company it keeps!". Consider the following example, modified by Lin (1998a) from (?):

(20.31) A bottle of *tezgüino* is on the table.
Everybody likes *tezgüino*.
Tezgüino makes you drunk.
We make *tezgüino* out of corn.

The contexts in which *tezgüino* occurs suggest that it might be some kind of fermented alcoholic drink made from corn. The distributional method tries to capture this intuition by representing features of the context of *tezgüino* that might overlap with features of similar words like *beer*, *liquor*, *tequila*, and so on. For example such features might be occurs before *drunk* or occurs after *bottle* or is the direct object of *likes*.

feature vector

We can then represent a word w as a **feature vector** just as we saw with the bag-of-words features in Sec. 20.2. For example, suppose we had one binary feature f_i representing each of the N words in the lexicon v_i. The feature means w occurs in the neighborhood of word v_i, and hence takes the value 1 if w and v_i occur in some context window, and 0 otherwise. We could represent the meaning of word w as the feature

vector

$$\vec{w} = (f_1, f_2, f_3, \cdots, f_N)$$

If *w*= *tezgüino*, v_1=*bottle*, v_2=*drunk*, and v_3=*matrix*, the co-occurrence vector for *w* from the corpus above would be:

$$\vec{w} = (1, 1, 0, \cdots)$$

Given two words represented by such sparse feature vectors, we can apply a vector distance measure and say that the words are similar if the two vectors are close by this measure. Fig. 20.7 shows an intuition about vector similarity for the four words *apricot*, *pineapple*, *digital*, and *information*. Based on the meanings of these four words, we would like a metric that shows *apricot* and *pineapple* to be similar, *digital* and *information*, to be similar, and the other four pairings to produce low similarity. For each word, Fig. 20.7 shows a short piece (8 dimensions) of the (binary) word co-occurrence vectors, computed from words that occur within a two-line context in the Brown corpus. The reader should convince themselves that the vectors for *apricot* and *pineapple* are indeed more similar than those of, say, *apricot* and *information*. For pedagogical purposes we've shown the context words that are particularly good at discrimination. Note that since vocabularies are quite large (10,000-100,000 words) and most words don't occur near each other in any corpus, real vectors are quite sparse.

	arts	**boil**	**data**	**function**	**large**	**sugar**	**summarized**	**water**
apricot	0	1	0	0	1	1	0	1
pineapple	0	1	0	0	1	1	0	1
digital	0	0	1	1	1	0	1	0
information	0	0	1	1	1	0	1	0

Figure 20.9 Co-occurrence vectors for four words, computed from the Brown corpus, showing only 8 of the (binary) dimensions (hand-picked for pedagogical purposes to show discrimination). Note that *large* occurs in all the contexts and *arts* occurs in none; a real vector would be extremely sparse.

Now that we have some intuitions, let's move on to examine the details of these measures. Specifying a distributional similarity measure requires that we specify three parameters: (1) how the co-occurrence terms are defined (i.e. what counts as a neighbor), (2) how these terms are weighted (binary? frequency? mutual information?) and (3) what vector distance metric we use (cosine? Euclidean distance?). Let's look at each of these requirements in the next three subsections.

20.7.1 Defining a Word's Co-occurrence Vectors

In our example feature vector, we used the feature *w* occurs in the neighborhood of word v_j. That is, for a vocabulary size *N*, each word *w* had *N* features, specifying whether vocabulary element v_j occurred in the neighborhood. Neighborhoods range from a small window of words (as few as one or two words on either side) to very large windows of $\pm$500 words. In a minimal window, for example, we might have two

features for each word v_j in the vocabulary, word v_k occurs immediately before word w and word v_k occurs immediately after word w.

To keep these contexts efficient, we often ignore very frequent words which tend not to be very discriminative, e.g., function words such as *a*, *am*, *the*, *of*, *1*, *2*, and so on. These removed words are called **stopwords** or the **stoplist**.

stopwords
stoplist

Even with the removal of the stopwords, when used on very large corpora these co-occurrence vectors tend to be very large. Instead of using every word in the neighborhood, Hindle (1990) suggested choosing words that occur in some sort of **grammatical relation** or **dependency** to the target words. Hindle suggested that nouns which bear the same grammatical relation to the same verb might be similar. For example, the words *tea*, *water*, and *beer* are all frequent direct objects of the verb *drink*. The words *senate*, *congress*, *panel*, and *legislature* all tend to be subjects of the verbs *consider*, *vote*, and *approve*.

Hindle's intuition follows from the early work of Harris (1968), who suggested that:

> The meaning of entities, and the meaning of grammatical relations among them, is related to the restriction of combinations of these entities relative to other entities.

There have been a wide variety of realizations of Hindle's idea since then. In general, in these methods each sentence in a large corpus is parsed and a dependency parse is extracted. We saw in Ch. 12 lists of grammatical relations produced by dependency parsers, including noun-verb relations like subject, object, indirect object, and noun-noun relations like genitive, ncomp, and so on. A sentence like the following would result in the set of dependencies shown here:

(20.32) I discovered dried tangerines:

discover (subject I)	I (subj-of discover)
tangerine (obj-of discover)	tangerine (adj-mod dried)
dried (adj-mod-of tangerine)	

Since each word can be in a variety of different dependency relations with other words, we'll need to augment the feature space. Each feature is now a pairing of a word and a relation, so instead of a vector of N features, we have a vector of $N \times R$ features, where R is the number of possible relations. Fig. 20.10 shows a schematic example of such a vector, taken from Lin (1998a), for the word *cell*. As the value of each attribute we have shown the frequency of the feature co-occurring with *cell*; the next section will discuss the use of what values and weights to use for each attribute.

Since full parsing is very expensive, it is common to use a chunker or shallow parser of the type defined in Sec. 13.5, with the goal of extracting only a smaller set of relations like subject, direct object, and prepositional object of a particular preposition (Curran, 2003).

20.7.2 Measures of Association with Context

Now that we have a definition for the features or dimensions of a word's context vector, we are ready to discuss the values that should be associated with those features. These values are typically thought of as **weights** or measures of **association** between each

association

	subj-of, absorb	*subj-of*, adapt	*subj-of*, behave	...	*pobj-of*, inside	*pobj-of*, into	...	*nmod-of*, abnormality	*nmod-of*, anemia	*nmod-of*, architecture	...	*obj-of*, attack	*obj-of*, call	*obj-of*, come from	*obj-of*, decorate	...	*nmod*, bacteria	*nmod*, body	*nmod*, bone marrow
cell	1	1	1		16	30		3	8	1		6	11	3	2		3	2	2

Figure 20.10 Co-occurrence vector for the word *cell*, from Lin (1998a), showing grammatical function (dependency) features. Values for each attribute are frequency counts from a 64-million word corpus, parsed by an early version of MINIPAR.

target word w and a given feature f. In the example in Fig. 20.7, our association measure was a binary value for each feature, 1 if the relevant word had occurred in the context, 0 if not. In the example in Fig. 20.10, we used a richer association measure, the relative frequency with which the particular context feature had co-occurred with the target word.

Frequency, or probability, is certainly a better measure of association than just a binary value; features that occur often with a target word are more likely to be good indicators of the word's meaning. Let's define some terminology for implementing a probabilistic measure of association. For a target word w, each element of its co-occurrence vector is a feature f, consisting of a relation r and a related word w'; we can say $f = (r, w')$. For example, one of the features of the word *cell* in Fig. 20.10 is $f = (r, w')$ =(obj-of, *attack*). The probability of a feature f given a target word w is $P(f|w)$, for which the maximum likelihood estimate is:

$$P(f|w) = \frac{\text{count}(f,w)}{\text{count}(w)} \tag{20.33}$$

Similarly, the maximum likelihood estimate for the joint probability $P(f,w)$ is:

$$P(f,w) = \frac{\text{count}(f,w)}{\sum_{w'} \text{count}(w'))} \tag{20.34}$$

$P(w)$ and $P(f)$ are computed similarly.

Thus if we were to define simple probability as a measure of association it would look as follows:

$$\text{assoc}_{\text{prob}}(w,f) = P(f|w) \tag{20.35}$$

It turns out, however, that simple probability doesn't work as well as more sophisticated association schemes for word similarity.

Why isn't frequency or probability a good measure of association between a word and a context feature? Intuitively, if we want to know what kinds of contexts are shared by *apricot* and *pineapple* but not by *digital* and *information*, we're not going to get good discrimination from words like *the*, *it*, or *they*, which occur frequently with all sorts of words, and aren't informative about any particular word. We'd like context words which are particularly informative about the target word. We, therefore, need

a weighting or measure of association which asks how much more often than chance that the feature co-occurs with the target word. As Curran (2003) points out, such a weighting is what we also want for finding good **collocations**, and so the measures of association used for weighting context words for semantic similarity are exactly the same measure used for finding a word's collocations.

collocations

One of the most important measures of association was first proposed by Church and Hanks (1989, 1990) and is based on the notion of **mutual information**. The **mutual information** between two random variables X and Y is

mutual information

$$I(X,Y) = \sum_x \sum_y P(x,y) \log_2 \frac{P(x,y)}{P(x)P(y)} \tag{20.36}$$

The **pointwise mutual information** (Fano, 1961)[3] is a measure of how often two events x and y occur, compared with what we would expect if they were independent:

pointwise mutual information

$$I(x,y) = \log_2 \frac{P(x,y)}{P(x)P(y)} \tag{20.37}$$

We can apply this intuition to co-occurrence vectors, by defining the pointwise mutual information association between a target word w and a feature f as:

$$\text{assoc}_{\text{PMI}}(w,f) = \log_2 \frac{P(w,f)}{P(w)P(f)} \tag{20.38}$$

The intuition of the PMI measure is that the numerator tells us how often we observed the two words together (assuming we compute probability using MLE as above). The denominator tells us how often we would **expect** the two words to co-occur assuming they each occurred independently, so their probabilities could just be multiplied. Thus the ratio gives us an estimate of how much more the target and feature co-occur than we expect by chance.

Since f is itself composed of two variables r and w', there is a slight variant on this model, due to Lin (1998a), that breaks down the expected value for $P(f)$ slightly differently; we'll call it the **Lin association measure** $\text{assoc}_{\text{Lin}}$, not to be confused with the WordNet measure sim_{Lin} that we discussed in the previous section:

Lin association measure

$$\text{assoc}_{\text{Lin}}(w,f) = \log_2 \frac{P(w,f)}{P(w)P(r|w)P(w'|w)} \tag{20.39}$$

For both $\text{assoc}_{\text{PMI}}$ and $\text{assoc}_{\text{Lin}}$, we generally only use the feature f for a word w if the assoc value is positive, since negative PMI values (implying things are co-occurring *less often* than we would expect by chance) tend to be unreliable unless the training corpora are enormous (Dagan et al., 1993; Lin, 1998a). In addition, when we are using the assoc-weighted features to compare two target words, we only use features that co-occur with both target words.

Fig. 20.11 from Hindle (1990) shows the difference between raw frequency counts and PMI-style association, for some direct objects of the verb *drink*.

[3] Fano actually used the phrase *mutual information* to refer to what we now call *pointwise mutual information*, and the phrase *expectation of the mutual information* for what we now call *mutual information*; the term *mutual information* is still often used to mean *pointwise mutual information*.

Object	Count	PMI assoc	Object	Count	PMI assoc
bunch beer	2	12.34	wine	2	9.34
tea	2	11.75	water	7	7.65
Pepsi	2	11.75	anything	3	5.15
champagne	4	11.75	much	3	5.15
liquid	2	10.53	it	3	1.25
beer	5	10.20	<SOME AMOUNT>	2	1.22

Figure 20.11 Objects of the verb *drink*, sorted by PMI, from Hindle (1990).

t-test

One of the most successful association measures for word similarity attempts to capture the same intuition as mutual information, but uses the **t-test** statistic to measure how much more frequent the association is than chance. This measure was proposed for collocation-detection by Manning and Schütze (1999, Chapter 5) and then applied to word similarity by Curran and Moens (2002), Curran (2003).

The t-test statistic computes the difference between observed and expected means, normalized by the variance. The higher the value of t, the more likely we can reject the null hypothesis that the observed and expected means are the same.

(20.40)
$$t = \frac{\bar{x} - \mu}{\sqrt{\frac{s^2}{N}}}$$

When applied to association between words, the null hypothesis is that the two words are independent, and hence $P(f,w) = P(f)P(w)$ correctly models the relationship between the two words. We want to know how different the actual MLE probability $P(f,w)$ is from this null hypothesis value, normalized by the variance. Note the similarity to the comparison with the product model in the PMI measure above. The variance s^2 can be approximated by the expected probability $P(f)P(w)$ (see Manning and Schütze (1999)). Ignoring N (since it is constant), the resulting t-test association measure from Curran (2003) is thus:

(20.41)
$$\text{assoc}_{\text{t-test}}(w,f) = \frac{P(w,f) - P(w)P(f)}{\sqrt{P(f)P(w)}}$$

See the history section for a summary of various other weighting factors that have been tested on word similarity.

20.7.3 Defining similarity between two vectors

From the previous sections we can now compute a co-occurrence vector for a target word, with each co-occurrence feature weighted by an association measure, giving us a distributional definition of the meaning of a target word.

To define similarity between two target words v and w, we need a measure for taking two such vectors and giving a measure of vector similarity. Perhaps the simplest two measures of vector distance are the Manhattan and Euclidean distance. Fig. 20.12 shows a graphical intuition for Euclidean and Manhattan distance between two two-dimensional vectors $\vec{a}$ and $\vec{b}$. The **Manhattan distance**, also known as **Levenshtein distance** or **L1 norm**, is

Manhattan distance
Levenshtein distance
L1 norm

$$\text{distance}_{\text{manhattan}}(\vec{x},\vec{y}) = \sum_{i=1}^{N} |x_i - y_i| \tag{20.42}$$

L2 norm

The **Euclidean distance**, also called the **L2 norm**, was introduced in Ch. 9:

$$\text{distance}_{\text{euclidean}}(\vec{x},\vec{y}) = \sqrt{\sum_{i=1}^{N} (x_i - y_i)^2} \tag{20.43}$$

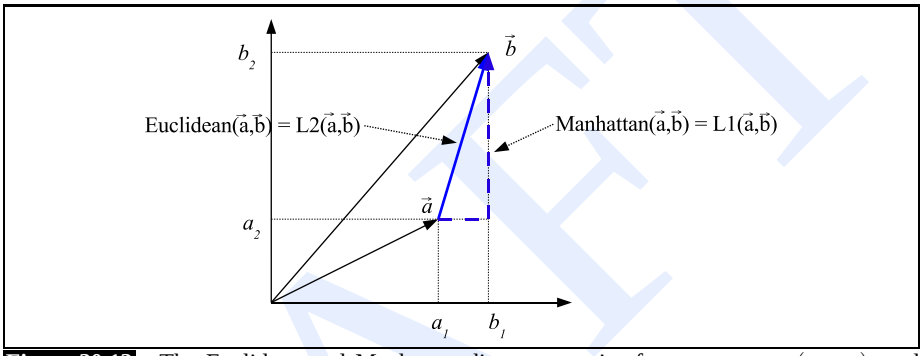

Figure 20.12 The Euclidean and Manhattan distance metrics for vectors $a = (a_1, a_2)$, and $b = (b_1, b_2)$, just to give the reader a graphical intuition about the idea of distance between vectors; these particular metrics are generally not used for word similarity. See Ch. 9 for more on distance metrics.

Although the Euclidean and Manhattan distance metrics provide a nice geometric intuition for vector similarity and distance, these measures are rarely used for word similarity. This is because both measures turn out to be very sensitive to extreme values. Instead of these simple distance metrics, word similarity is based on closely related metrics from **information retrieval** and from **information theory**. The information retrieval methods seem to work better for word similarity, so we'll define a number of these in this section.

Let's begin with the intuition for a similarity metric in Fig. 20.7, in which the similarity between two binary vectors was just the number of features the two words had in common. If we assume a feature vector is a **binary vector**, we can define such a similarity metric as follows, using the **dot product** or **inner product** operator from linear algebra:

binary vector
dot product
inner product

$$\text{sim}_{\text{dot-product}}(\vec{v},\vec{w}) = \vec{v}\cdot\vec{w} = \sum_{i=1}^{N} v_i \times w_i \tag{20.44}$$

In most cases, though, as we saw in the previous section, the values of our vector are not binary. Let's assume for the rest of this section that the entries in the co-occurrence vector are the **association** values between the target words and each of the features. In other words, let's define the vector for a target word $\vec{w}$ with N features $f_1..f_N$ as:

$$\vec{w} = (\text{assoc}(w, f_1), \text{assoc}(w, f_2), \text{assoc}(w, f_3), \ldots, \text{assoc}(w, f_N)) \tag{20.45}$$

Now we can apply $\text{sim}_{\text{dot-product}}$ to vectors with values defined as associations, to get the dot-product similarity between weighted values. This raw dot-product, however, has a problem as a similarity metric: it favors **long** vectors. The **vector length** is defined as:

vector length

$$|\vec{v}| = \sqrt{\sum_{i=1}^{N} v_i^2} \tag{20.46}$$

A vector can be longer because it has more non-zero values, or because each dimension has a higher value. Both of these facts will increase the dot product. It turns out that both of these can occur as a by-product of word frequency. A vector from a very frequent word will have more non-zero co-occurrence association values, and will probably have higher values in each (even using association weights that control somewhat for frequency). The raw dot product thus favors frequent words.

We need to modify the dot product to normalize for the vector length. The simplest way is just to divide the dot product by the lengths of each of the two vectors. This **normalized dot product** turns out to be the same as the cosine of the angle between the two vectors. The **cosine** or normalized dot product similarity metric is thus:

normalized dot product
cosine

$$\text{sim}_{\text{cosine}}(\vec{v},\vec{w}) = \frac{\vec{v}\cdot\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{i=1}^{N} v_i \times w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}} \tag{20.47}$$

Because we have transformed the vectors to unit length, the cosine metric, unlike Euclidean or Manhattan distance, is no longer sensitive to long vectors from high-frequency words. The cosine value ranges from 1 for vectors pointing in the same direction, through 0 for vectors which are orthogonal (share no common terms), to -1 for vectors pointing in opposite directions, although in practice values tend to be positive.

Let's discuss two more similarity measures derived from information retrieval. The **Jaccard** (Jaccard, 1908, 1912) (also called **Tanimoto** or **min/max** (Dagan, 2000)) measure was originally designed for binary vectors. It was extended by Grefenstette (1994) to vectors of weighted associations as follows:

Jaccard
Tanimoto
min/max

$$\text{sim}_{\text{Jaccard}}(\vec{v},\vec{w}) = \frac{\sum_{i=1}^{N} \min(v_i, w_i)}{\sum_{i=1}^{N} \max(v_i, w_i)} \tag{20.48}$$

The numerator of the Grefenstette/Jaccard function uses the min function, essentially computing the (weighted) number of overlapping features (since if either vector has a zero association value for an attribute, the result will be zero). The denominator can be viewed as a normalizing factor.

Dice

A very similar measure, the **Dice** measure, was similarly extended from binary vectors to vectors of weighted associations; one extension from Curran (2003) uses the Jaccard numerator, but uses as the denominator normalization factor the total weighted value of non-zero entries in the two vectors.

$$\text{sim}_{\text{Dice}}(\vec{v},\vec{w}) = \frac{2 \times \sum_{i=1}^{N} \min(v_i, w_i)}{\sum_{i=1}^{N} (v_i + w_i)} \tag{20.49}$$

$$\text{assoc}_{\text{prob}}(w,f) = P(f|w) \quad (20.35)$$
$$\text{assoc}_{\text{PMI}}(w,f) = \log_2 \frac{P(w,f)}{P(w)P(f)} \quad (20.38)$$
$$\text{assoc}_{\text{Lin}}(w,f) = \log_2 \frac{P(w,f)}{P(w)P(r|w)P(w'|w)} \quad (20.39)$$
$$\text{assoc}_{\text{t-test}}(w,f) = \frac{P(w,f)-P(w)P(f)}{\sqrt{P(f)P(w)}} \quad (20.41)$$

$$\text{sim}_{\text{cosine}}(\vec{v},\vec{w}) = \frac{\vec{v}\cdot\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{i=1}^{N} v_i \times w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}} \quad (20.47)$$
$$\text{sim}_{\text{Jaccard}}(\vec{v},\vec{w}) = \frac{\sum_{i=1}^{N} \min(v_i,w_i)}{\sum_{i=1}^{N} \max(v_i,w_i)} \quad (20.48)$$
$$\text{sim}_{\text{Dice}}(\vec{v},\vec{w}) = \frac{2\times\sum_{i=1}^{N} \min(v_i,w_i)}{\sum_{i=1}^{N} (v_i+w_i)} \quad (20.49)$$
$$\text{sim}_{\text{JS}}(\vec{v}||\vec{w}) = D(\vec{v}|\frac{\vec{v}+\vec{w}}{2}) + D(\vec{w}|\frac{\vec{v}+\vec{w}}{2}) \quad (20.52)$$

Figure 20.13 Defining word similarity: measures of association between a target word w and a feature $f = (r,w')$ to another word w', and measures of vector similarity between word co-occurrence vectors $\vec{v}$ and $\vec{w}$.

Finally, there is a family of information-theoretic distributional similarity measures, (Pereira et al., 1993; Dagan et al., 1994, 1999; Lee, 1999), also based on the conditional probability association measure $P(f|w)$. The intuition of these models is that two vectors $\vec{v}$ and $\vec{w}$ are similar to the extent that their probability distributions $P(f|w)$ and $P(f|v)$ are similar. The basis of comparing two probability distributions P and Q is the

KL divergence

Kullback-Leibler divergence or **KL divergence** or **relative entropy** (Kullback and Leibler, 1951) :

$$D(P||Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)} \quad (20.50)$$

Unfortunately, the KL-divergence is undefined when $Q(x) = 0$ and $P(x) \neq 0$, which is a problem since these word distribution vectors are generally quite sparse. One alternative (Lee, 1999) is to use the

Jenson-Shannon divergence

Jenson-Shannon divergence, which represents the divergence of each distribution from the mean of the two, and doesn't have this problem with zeros:

$$JS(P||Q) = D(P|\frac{P+Q}{2}) + D(Q|\frac{P+Q}{2}) \quad (20.51)$$

Rephrased in terms of vectors $\vec{v}$ and $\vec{w}$,

$$\text{sim}_{\text{JS}}(\vec{v}||\vec{w}) = D(\vec{v}|\frac{\vec{v}+\vec{w}}{2}) + D(\vec{w}|\frac{\vec{v}+\vec{w}}{2}) \quad (20.52)$$

Fig. 20.13 summarizes the measures of association and of vector similarity that we have designed. See the history section for a summary of other vector similarity measures.

Finally, let's look at some of the results of distributional word similarity. The following are the ten most similar words to the different parts of speech of *hope* and *brief*, derived using the online dependency-based similarity tool (Lin, 2007); this tool defines the co-occurrence vector using all minipar grammatical relations, uses the $\text{assoc}_{\text{Lin}}$ measure of association, and a vector similarity metric from Lin (1998a).

- **hope (N):** optimism 0.141, chance 0.137, expectation 0.137, prospect 0.126, dream 0.119, desire 0.118, fear 0.116, effort 0.111, confidence 0.109, promise 0.108
- **hope (V):** would like 0.158, wish 0.140, plan 0.139, say 0.137, believe 0.135, think 0.133, agree 0.130, wonder 0.130, try 0.127, decide 0.125,
- **brief (N):** legal brief 0.139, affidavit 0.103, filing 0.0983, petition 0.0865, document 0.0835, argument 0.0832, letter 0.0786, rebuttal 0.0778, memo 0.0768, article 0.0758
- **brief (A):** lengthy 0.256, hour-long 0.191, short 0.174, extended 0.163, frequent 0.163, recent 0.158, short-lived 0.155, Prolonged 0.149, week-long 0.149, occasional 0.146

20.7.4 Evaluating Distributional Word Similarity

Distributional similarity can be evaluated in the same ways as thesaurus-based similarity; we can compare intrinsically to human similarity scores, or we can evaluate it extrinsically as part of end-to-end applications. Besides word sense disambiguation and malapropism detection, similarity measures have been used as a part of systems for the grading of exams and essays(Landauer et al., 1997), or taking TOEFL multiple-choice exams (Landauer and Dumais, 1997; Turney et al., 2003).

Distributional algorithms are also often evaluated in a third intrinsic way: by comparison with a gold-standard thesaurus. This comparison can be direct with a single thesaurus (Grefenstette, 1994; Lin, 1998a) or by using precision and recall measure against an ensemble of thesauri (Curran and Moens, 2002). Let S be the set of words that are defined as similar in the thesaurus, by being in the same synset, or perhaps sharing the same hypernym, or being in the hypernym-hyponym relation. Let S' be the set of words that are classified as similar by some algorithm. We can define precision and recall as:

$$\text{precision} = \frac{|S \cap S'|}{|S'|} \quad \text{recall} = \frac{|S \cap S'|}{|S|} \tag{20.53}$$

Curran (2003) evaluated a number of distributional algorithms using comparison with thesauri and found that the Dice and Jaccard methods performed best as measures of vector similarity, while t-test performed best as a measure of association. Thus the best metric weighted the associations with t-test, and then used either Dice or Jaccard to measure vector similarity.

20.8 Hyponymy and other word relations

Similarity is only one kind of semantic relation between words. As we discussed in Ch. 19, WordNet and MeSH both include **hyponymy/hypernymy**, as do many thesauruses for other languages, such as CiLin for Chinese (?). WordNet also includes **antonymy**, **meronymy**, and other relations. Thus if we want to know if two senses are related by one of these relations, and the senses occur in WordNet or MeSH, we can just look them up. But since many words are not in these resources, it is important to be able to learn new hypernym and meronym relations automatically.

Much work on automatic learning of word relations is based on a key insight first articulated by Hearst (1992), that the presence of certain lexico-syntactic patterns can indicate a particular semantic relationship between two nouns. Consider the following sentence extracted by Hearst from the Groliers encyclopedia:

(20.54) 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**

(20.55) $$NP_0 \textit{ such as } NP_1\{,NP_2\ldots,(and|or)NP_i\}, i \geq 1$$

implies the following semantics

(20.56) $$\forall NP_i, i \geq 1, \text{hyponym}(NP_i, NP_0)$$

allowing us to infer

(20.57) $$\text{hyponym}(\text{Gelidium}, \text{red algae})$$

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

Figure 20.14 Hand-built lexico-syntactic patterns for finding hypernyms (Hearst, 1992, 1998)

Fig. 20.14 shows five patterns Hearst (1992, 1998) suggested for inferring the hyponym relation; we've shown NP_H as the parent/hyponym. There are a number of other attempts to extract different WordNet relations using such patterns; see the history section for more details.

Of course, the coverage of such pattern-based methods is limited by the number and accuracy of the available patterns. Unfortunately, once the obvious examples have been found, the process of creating patterns by hand becomes a difficult and slow process. Fortunately, we've already seen the solution to this kind of problem. We can find new patterns using **bootstrapping** methods that are common in information extraction (Riloff, 1996; Brin, 1998), and are also key to the Yarowsky method described earlier in Sec. 20.5.

The key insight for the use of bootstrapping in relational pattern discovery is that with a large corpus we can expect that words involved in a relation to show up with many different patterns that express that same relation. Therefore, in theory at least, we need only start with a small number of precise patterns to acquire a set of seed words involved in a given relation. These words can then be used to query a large corpus for sentences containing both terms in some kind of dependency relation; new patterns can then be extracted from these new sentences. The process can be repeated until the pattern set is large enough.

As an example of this process, consider the terms "red algae" and "Gelidium" discovered earlier using Hearst's simple pattern set. Among the results of a simple Google search using these as query terms is the following example:

(20.58) One example of a red algae is Gelidium.

Removing the seed words from such a sentence and replacing them with simple wildcards is the crudest kind of pattern generation. In this case, submitting the pattern "One example of a * is *" to Google currently yields nearly 500,000 hits, including the following example:

(20.59) One example of a boson is a photon.

We can also extract slightly more sophisticated patterns by parsing the extracted sentences and putting wildcards into the parse tree.

The key to the success of bootstrapping approaches is to avoid the *semantic drift* that tends to occur as part of repeated applications of bootstrapping. The further we get from the original set of seed words or patterns the more likely it is that we'll come across patterns with meanings quite different from what we set out to discover. We'll see methods for dealing with this drift when we discuss bootstrapping for information extraction in Ch. 22.

An alternative to bootstrapping is to use large lexical resources like WordNet as a source of training information, in which each WordNet hypernym/hyponym pair tells us something about kinds of words are in this relation, and we train a classifier to help find new words that exhibit this relation.

This hyponym learning algorithm of Snow et al. (2005), for example, relies on WordNet to help learn large numbers of weak hyponym patterns, and then combine them in a supervised classifier in 4 steps:

1. Collect all pairs of WordNet noun concepts c_i, c_j that are in the hypernym / hyponym relation.
2. For each noun pair, collect all sentences (in a 6 million word corpus) in which both nouns occur.
3. Parse the sentences and automatically extract every possible Hearst-style lexico-syntactic pattern from the parse tree
4. Use the large set of patterns as features in an logistic regression classifier
5. Given a pair of nouns in the test set, extract features and use the classifier to determine if the noun pair is related by the hypernym/hyponym relation or not.

Four of the new patterns automatically learned by this algorithm include:

NP_H like NP	NP_H called NP
NP is a NP_H	NP, a NP_H (appositive):

Snow et al. (2005) then showed good hypernym detection performance by using each of these patterns as a weak feature combined by a logistic regression classifier.

Another way to use WordNet to help address the hypernym problem is to model the task as choosing the place to insert unknown words into an otherwise complete hierarchy. It is possible to do this without using lexico-syntactic patterns. For example, we can use a similarity classifier (using distributional information, or morphological information) to find the words in the hierarchy that are most similar to an unknown word, using an approach like K-Nearest-Neighbors, and insert the new word there (Tseng, 2003). Or we can treat the task of hypernym labeling as a labeling task like named-entity tagging. Ciaramita and Johnson (2003) take this approach, using as tags 26 **supersenses**, from the 26 broad-category 'lexicographer class' labels from WordNet (*person*, *location*, *event*, *quantity*, etc). They use features such as surrounding part-of-speech tags, word bigram and trigram features, spelling and morphological features, and apply a multiclass perceptron classifier.

supersenses

Finding **meronyms** seems to be harder than hyponyms; here are some examples from Girju et al. (2003):

(20.60) The car's mail messenger is busy at work in the <PART>mail car</PART> as the <WHOLE>train</WHOLE> moves along.

(20.61) Through the open <PART>side door</PART> of the <WHOLE>car</WHOLE>, moving scenery can be seen.

Meronyms are hard to find because the lexico-syntactic patterns that characterize them are very ambiguous. For example the two most common patterns indicating meronymy are the English genitive constructions [NP_1 of NP_2] and [NP_1's NP_2], which also express many other meanings such as *possession*; see Girju et al. (2003, 2006) for discussion and possible algorithms.

Learning individual relations between words is an important component of the general task of **thesaurus induction**. In thesaurus induction, we combine our estimates of word similarity with our hypernym or other relations to build an entire ontology or thesaurus. For example the two-step thesaurus induction algorithm of Caraballo (1999, 2001) first applies a bottom-up **clustering** algorithm to group together semantically similar words into an unlabeled word hierarchy. Recall from Sec. 20.10 that in agglomerative clustering, we start by assigning each word its own cluster. New clusters are then formed in a bottom-up fashion by successively merging the two clusters that are most similar; we can use any metric for semantic similarity, such as one of the distributional metrics described in the previous section. In the second step, given the unlabeled hierarchy, the algorithm uses a pattern-based hyponym classifier to assign a hypernym label to each cluster of words. See the history section for more recent work on thesaurus induction.

thesaurus induction

20.9 Semantic Role Labeling

semantic role labeling

The final task we'll discuss in this chapter links word meanings with sentence meanings. This is the task of **semantic role labeling**, sometimes called **thematic role labeling**, **case role assignment** or even **shallow semantic parsing**. Semantic role labeling is the task of automatically finding the **semantic roles** for each predicate in a sentence. More specifically, that means determining which constituents in a sentence are semantic arguments for a given predicate, and then determining the appropriate role for each of those arguments. Semantic role labeling has the potential to improve performance in any language understanding task, although to date its primary applications have been in question answering and information extraction.

Current approaches to semantic role labeling are based on supervised machine learning and hence require access to adequate amounts of training and testing materials. Over the last few years, both the FrameNet and PropBank resources discussed in Ch. 19 have played this role. That is, they have been used to specify what counts as a predicate, to define the set of roles used in the task and to provide training and test data. The SENSEVAL-3 evaluation used Framenet, while the CONLL evaluations in 2004 and 2005 were based on PropBank.

The following examples show the different representations from the two efforts. Recall that FrameNet (20.62) employs a large number of frame-specific frame elements as roles, while PropBank (20.63) makes use of a smaller number of numbered argument labels which can be interpreted as verb-specific labels.

(20.62)
[You]	can't	[blame]	[the program]	[for being unable to identify it]
COGNIZER		TARGET	EVALUEE	REASON

(20.63)
[The San Francisco Examiner]	issued	[a special edition]	[yesterday]
ARG0	TARGET	ARG1	ARGM-TMP

A simplified semantic role labeling algorithm is sketched in Fig. 20.15. Following the very earliest work on semantic role analysis (Simmons, 1973), most work on semantic role labeling begins by parsing the sentence. Publicly available broad-coverage parsers (such as Collins (1996) or Charniak (1997)) are typically used to assign a parse to the input string. Fig. 20.16 shows a parse of (20.63) above. The resulting parse is then traversed to find all predicate-bearing words. For each of these predicates the tree is again traversed to determine which role, if any, each constituent in the parse plays with respect to that predicate. This judgment is made by first characterizing the constituent as a set of features with respect to the predicate. A classifier trained on an appropriate training set is then passed this feature set and makes the appropriate assignment.

Let's look in more detail at the simple set of features suggested by Gildea and Jurafsky (2000, 2002), which have been incorporated into most role-labeling systems. We'll extract them for the first *NP* in Fig. 20.16, the *NP-SBJ* constituent *the San Francisco Examiner*.

- The governing **predicate**, in this case the verb *issued*. For PropBank, the predicates are always verbs; FrameNet also has noun and adjective predicates. The

```
function SEMANTICROLELABEL(words) returns labeled tree

    parse ← PARSE(words)
    for each predicate in parse do
        for each node in parse do
            featurevector ← EXTRACTFEATURES(node, predicate, parse)
            CLASSIFYNODE(node, featurevector, parse)
```

Figure 20.15 A generic semantic role labeling algorithm. The CLASSIFYNODE component can be a simple 1-of-N classifier which assigns a semantic role (or NONE for non-role constituents). CLASSIFYNODE can be trained on labeled data such as FrameNet or PropBank.

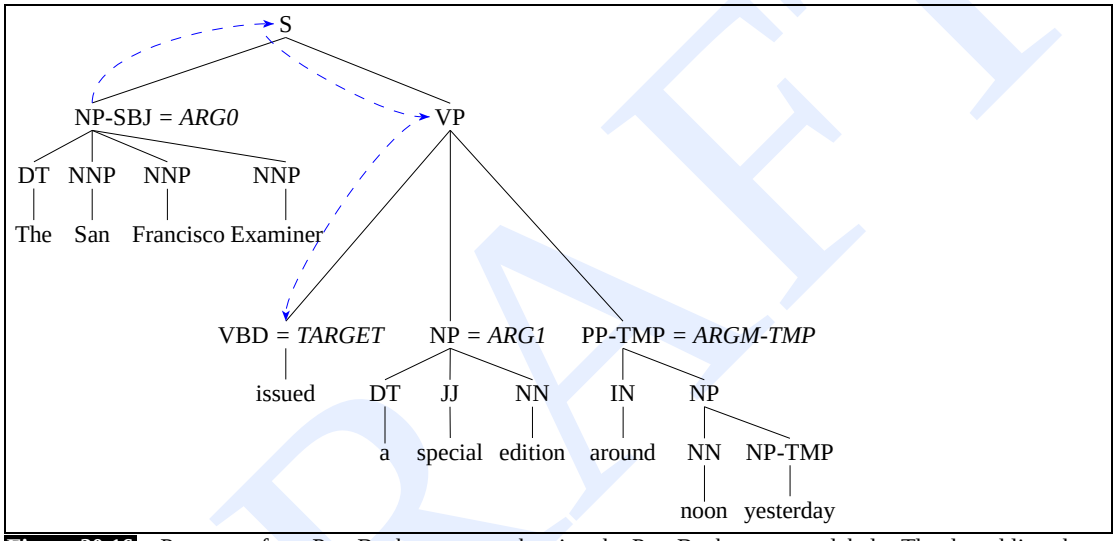

Figure 20.16 Parse tree for a PropBank sentence, showing the PropBank argument labels. The dotted line shows the **path** feature NP↑S↓VP↓VBD for ARG0, the NP-SBJ constituent *the San Francisco Examiner*.

predicate is a crucial feature, since both PropBank and FrameNet labels are defined only with respect to a particular predicate.

- The **phrase type** of the constituent, in this case *NP* (or *NP-SBJ*). This is simply the name of the parse node which dominates this constituent in the parse tree. Some semantic roles tend to appear as *NP*s, others as *S* or *PP*, and so on.
- The **head word** of the constituent, *Examiner*. The head word of a constituent can be computed using standard head rules, such as those given in Ch. 12 in Fig. 12.13. Certain head words (e.g. pronouns) place strong constraints on the possible semantic roles they are likely to fill.
- The **head word part-of-speech** of the constituent, *NNP*.
- The **path** in the parse tree from the constituent to the predicate. This path is marked by the dotted line in Fig. 20.16. Following (Gildea and Jurafsky, 2000), we can use a simple linear representation of the path, NP↑S↓VP↓VBD. ↑ and ↓ represent upward and downward movement in the tree respectively. The path is

very useful as a compact representation of many kinds of grammatical function relationships between the constituent and the predicate.

- The **voice** of the clause in which the constituent appears, in this case **active** (as contrasted with **passive**). Passive sentences tend to have strongly different linkings of semantic roles to surface form than active ones.
- The binary **linear position** of the constituent with respect to the predicate, either **before** or **after**.
- The **sub-categorization** of the predicate. Recall from Ch. 12 that the subcategorization of a verb is the set of expected arguments that appear in the verb phrase. We can extract this information by using the phrase structure rule that expands the immediate parent of the predicate; VP → NP PP for the predicate in Fig. 20.16.

Many other features are generally extracted by semantic role labeling systems, such as named entity tags (it is useful to know if a constituent is a LOCATION or PERSON, for example), or more complex versions of the path features (the upward or downward halves, whether particular nodes occur in the path), the rightmost or leftmost words of the constituent, and so on.

We now have a set of observations like the following example, each with a vector of features; we have shown the features in the order described above (recall that most observations will have the value NONE rather than e.g., ARG0, since most constituents in the parse tree will not bear a semantic role):

ARG0: [issued, NP, Examiner, NNP, NP↑S↓VP↓VBD, active, before, VP → NP PP]

Just as we saw for word sense disambiguation, we can divide these observations into a training and a test set, use the training examples in any supervised machine learning algorithm, and build a classifier. SVM and Maximum Entropy classifiers have yielded good results on this task on standard evaluations. Once trained, the classifier can be used on unlabeled sentences to propose a role for each constituent in the sentence. More precisely, an input sentence is parsed and a procedure similar to that described earlier for training is employed.

Instead of training a single stage classifier, some role labeling algorithms do classification in multiple stages for efficiency:

- **Pruning:** to speed up execution, some constituents are eliminated from consideration as possible roles, based on simple rules
- **Identification:** a binary classification of each node as an ARG to be labeled or a NONE.
- **Classification:** a one-of-N classification of all the constituents that were labeled as ARG by the previous stage.

There are a number of complications that all semantic role labeling systems need to deal with. Constituents in FrameNet and PropBank are required to be non-overlapping. Thus if a system incorrectly labels two overlapping constituents as arguments, it needs to decide which of the two is correct. Additionally, the semantic roles of constituents are not independent; since PropBank does not allow multiple identical arguments, labeling one constituent as an ARG0 would greatly increase the probability of another

constituent being labeled ARG1. Both these problems can be addressed by the two-stage approaches based on lattice or *N*-best rescoring discussed in Ch. 9: having the classifier assign multiple labels to each constituent, each with a probability, and using a second global optimization pass to pick the best label sequence.

Instead of using parses as input, it is also possible to do semantic role labeling directly from raw (or part-of-speech tagged) text by applying the chunking techniques used for named entity extraction or partial parsing. Such techniques are particularly useful in domains such as bioinformatics where it is unlikely that syntactic parsers trained on typical newswire text will perform well.

Finally, semantic role labeling systems have been generally evaluated by requiring that each argument label must be assigned to the exactly correct word sequence or parse constituent. Precision, recall, and F-measure can then be computed. A simple rule-based system can be used as a baseline, for example tagging the first NP before the predicate as ARG0 and the first NP after the predicate as ARG1, and switching these if the verb phrase is passive.

20.10 Advanced: Unsupervised Sense Disambiguation

Let's briefly return to the WSD task. It is expensive and difficult to build large corpora in which each word is labeled for its word sense. For this reason, unsupervised approaches to sense disambiguation are an exciting and important research area.

In unsupervised approaches, we don't use human-defined word senses. Instead, the set of 'senses' of each word are created automatically from the instances of each word in the training set. Let's introduce a simplified version of the methods of Schütze's (Schütze, 1992b, 1998) on unsupervised sense disambiguation. In Schütze's method, we first represent each instance of a word in the training set by distributional context feature-vectors that are a slight generalization of the feature vectors we defined in Sec. 20.7. (It is for this reason that we turned to unsupervised sense disambiguation only after introducing word similarity.)

As in Sec. 20.7 we will represent a word w as a vector based on frequencies of its neighboring words. For example for a given target word (type) w, we might select 1000 words that occur most frequently within 25 words of any instance of w. These 1000 words become the dimension of the vector. Let's define f_i to mean the frequency with which word i occurs in the context of word w. We define the word vector $\vec{w}$ (for a given token (observation) of w) as:

$$\vec{w} = (f_1, f_2, f_3, \cdots, f_{1000})$$

So far this is just a version of the distributional context we saw in Sec. 20.7. We can also use a slightly more complex version of the distributional context. For example Schuetze defines the **context vector** of a word w not as this first-order vector, but instead by its **second order co-occurrence**. That is, the context vector for a word w is built by taking each word x in the context of w, for each x computing its word vector $\vec{x}$, and then taking the centroid (average) of the vectors $\vec{x}$.

Let's see how we use these context vectors (whether first-order or second-order) in unsupervised sense disambiguation of a word w. In training, we'll need only 3 steps:

1. For each token w_i of word w in a corpus, compute a context vector $\vec{c}$.
2. Use a **clustering algorithm** to **cluster** these word token context vectors $\vec{c}$ into a predefined number of groups or clusters. Each cluster defines a sense of w.
3. Compute the **vector centroid** of each cluster. Each vector centroid $\vec{s}_j$ is a **sense vector** representing that sense of w.

Since this is an unsupervised algorithm we won't have names for each of these 'senses' of w; we just refer to the jth sense of w.

Now how do we disambiguate a particular token t of w? Again we have three steps:

1. Compute a context vector $\vec{c}$ for t as discussed above.
2. Retrieve all sense vectors s_j for w.
3. Assign t to the sense represented by the sense vector s_j that is closest to t.

All we need is a clustering algorithm, and a distance metrics between vectors. Fortunately, clustering is a well-studied problem with a wide number of standard algorithms that can be applied to inputs structured as vectors of numerical values (Duda and Hart, 1973). A frequently used technique in language applications is known as **agglomerative clustering**. In this technique, each of the N training instances is initially assigned to its own cluster. New clusters are then formed in a bottom-up fashion by successively merging the two clusters that are most similar. This process continues until either a specified number of clusters is reached, or some global goodness measure among the clusters is achieved. In cases where the number of training instances makes this method too expensive, random sampling can be used on the original training set (Cutting et al., 1992b) to achieve similar results.

agglomerative clustering

How can we evaluate unsupervised sense disambiguation approaches? As usual, the best way is to do extrinsic or in vivo evaluation, in which the WSD algorithm is embedded in some end-to-end system. Intrinsic evaluation can also be useful, though, if we have some way to map the automatically derived sense classes into some hand-labeled gold standard set, so that we can compare a hand-labeled test set with a set labeled by our unsupervised classifier. One way of doing this mapping is to map each sense cluster to a pre-defined sense by choosing the sense that (in some training set) has the most word tokens overlapping with the cluster. Another is to consider all pairs of words in the test set, testing for each whether both the system and the hand-labeling put both members of the pair in the same cluster or not.

Bibliographical and Historical Notes

Word sense disambiguation traces its roots to some of the earliest applications of digital computers. We saw above Warren Weaver's (1955) suggestion to disambiguate a word by looking at a small window around it, in the context of machine translation. Other notions first proposed in this early period include the use of a thesaurus for

disambiguation (Masterman, 1957), supervised training of Bayesian models for disambiguation (Madhu and Lytel, 1965), and the use of clustering in word sense analysis (Sparck Jones, 1986).

An enormous amount of work on disambiguation has been conducted within the context of early AI-oriented natural language processing systems. While most natural language analysis systems of this type exhibited some form of lexical disambiguation capability, a number of these efforts made word sense disambiguation a larger focus of their work. Among the most influential efforts were the efforts of Quillian (1968) and Simmons (1973) with semantic networks, the work of Wilks with *Preference Semantics* Wilks (1975c, 1975b, 1975a), and the work of Small and Rieger (1982) and Riesbeck (1975) on word-based understanding systems. Hirst's ABSITY system (Hirst and Charniak, 1982; Hirst, 1987, 1988), which used a technique based on semantic networks called marker passing, represents the most advanced system of this type. As with these largely symbolic approaches, most connectionist approaches to word sense disambiguation have relied on small lexicons with hand-coded representations (Cottrell, 1985; Kawamoto, 1988).

Considerable work on sense disambiguation has been conducted in the areas of Cognitive Science and psycholinguistics. Appropriately enough, it is generally described using a different name: lexical ambiguity resolution. Small et al. (1988) present a variety of papers from this perspective.

The earliest implementation of a robust empirical approach to sense disambiguation is due to Kelly and Stone (1975) who directed a team that hand-crafted a set of disambiguation rules for 1790 ambiguous English words. Lesk (1986) was the first to use a machine readable dictionary for word sense disambiguation. Wilks et al. (1996) describe extensive explorations of the use of machine readable dictionaries. The problem of dictionary senses being too fine-grained or lacking an appropriate organization has been addressed with models of clustering word senses Dolan (1994), Peters et al. (1998), Chen and Chang (1998), Mihalcea and Moldovan (2001), Agirre and de Lacalle (2003), Chklovski and Mihalcea (2003), Palmer et al. (2004), McCarthy (2006), Navigli (2006), Snow et al. (2007); corpora with clustered word senses for training clustering algorithms include Palmer et al. (2006) and **OntoNotes** (Hovy et al., 2006).

OntoNotes

Modern interest in supervised machine learning approaches to disambiguation began with Black (1988), who applied decision tree learning to the task. The need for large amounts of annotated text in these methods led to investigations into the use of bootstrapping methods (Hearst, 1991; Yarowsky, 1995). The problem of how to weigh and combine disparate sources of evidence is explored in Ng and Lee (1996), McRoy (1992), and Stevenson and Wilks (2001).

Among the semi-supervised methods, more recent models of selectional preference include Li and Abe (1998), Ciaramita and Johnson (2000), McCarthy and Carroll (2003), Light and Greiff (2002). Diab and Resnik (2002) give a semi-supervised algorithm for sense disambiguation based on aligned parallel corpora in two languages. For example, the fact that the French word *catastrophe* might be translated as English *disaster* in one instance and *tragedy* in another instance can be used to disambiguate the senses of the two English words (i.e. to choose senses of *disaster* and *tragedy* that are similar). Abney (2002, 2004) explores the mathematical foundations of the Yarowsky algorithm and its relation to co-training. The most-frequent-sense heuristic

is an extremely powerful one, but requires large amounts of supervised training data. McCarthy et al. (2004) propose an unsupervised way to automatically estimate the most frequent sense, based on the thesaurus similarity metrics defined in Sec. 20.6.

The earliest attempt to use clustering in the study of word senses is due to Sparck Jones (1986). Zernik (1991) successfully applied a standard information retrieval clustering algorithm to the problem, and provided an evaluation based on improvements in retrieval performance. More extensive recent work on clustering can be found in Pedersen and Bruce (1997) and Schütze (1997, 1998).

A few algorithms have attempted to exploit the power of mutually disambiguating all the words in a sentence, either by multiple passes (Kelly and Stone, 1975) to take advantage of easily disambiguated words, or by parallel search (Cowie et al., 1992; Veronis and Ide, 1990).

Recent work has focused on ways to use the web for training data for word sense disambiguation, either unsupervised (Mihalcea and Moldovan, 1999) or by using volunteers to label data (Chklovski and Mihalcea, 2002).

Resnik (2006) describes potential applications of WSD. One recent application has been to improve machine translation Chan et al. (2007), Carpuat and Wu (2007).

Agirre and Edmonds (2006) is a comprehensive edited volume that summarizes the state of the art in WSD. Ide and Veronis (1998a) provide a comprehensive review of the history of word sense disambiguation up to 1998. Ng and Zelle (1997) provide a more focused review from a machine learning perspective. Wilks et al. (1996) describe dictionary and corpus experiments, along with detailed descriptions of very early work.

The models of distributional word similarity we discussed arose out of research in linguistics and psychology of the 1950's. The idea that meaning was related to distribution of words in context was widespread in linguistic theory of the 1950's; even before the well-known Firth (1957) and Harris (1968) dictums discussed earlier, Joos (1950) stated that

> the linguist's 'meaning' of a morpheme...is by definition the set of conditional probabilities of its occurrence in context with all other morphemes'

The related idea that the meaning of a word could be modeled as a point in a Euclidean space, and that the similarity of meaning between two words could be modeled as the distance between these points, was proposed in psychology by Osgood et al. (1957). The application of these ideas in a computational framework was first made by Sparck Jones (1986), and became a core principle of information retrieval, from whence it came into broader use in speech and language processing.

There are a wide variety of other weightings and methods for word similarity. The largest class of methods not discussed in this chapter are the variants to and details of the **information-theoretic** methods like Jensen-Shannon divergence, KL-divergence and α-skew divergence that we briefly introduced (Pereira et al., 1993; Dagan et al., 1994, 1999; Lee, 1999, 2001); there are also other metrics from Hindle (1990) and Lin (1998a). Alternative paradigms include the **co-occurrence retrieval** model (Weeds, 2003; Weeds and Weir, 2005). Manning and Schütze (1999, Chapter 5 and 8) give collocation measures and other related similarity measures. A commonly used weighting is **weighted mutual information** (Fung and McKeown, 1997) in which the pointwise mutual information is weighted by the joint probability. In information retrieval the **TF/IDF** weight is widely used, as we will see in Ch. 23. See Dagan (2000), Mo-

weighted mutual information

hammad and Hirst (2005), Curran (2003) and Weeds (2003) for good summaries of distributional similarity.

Latent Semantic Indexing
LSA

An alternative vector space model of semantic similarity, **Latent Semantic Indexing** (LSI) or **Latent Semantic Analysis** (**LSA**), uses **singular value decomposition** to reduce the dimensionality of the vector space with the intent of discovering higher-order regularities (Deerwester et al., 1990). We have already discussed Schütze (1992b), another semantic similarity model based on singular value decomposition.

There is a wide variety of recent literature on other lexical relations and thesaurus induction. The use of distributional word similarity for thesaurus induction was explored systematically by Grefenstette (1994). A wide variety of distributional clustering algorithms have been applied to the task of discovering groupings of semantically similar words, including hard clustering (Brown et al., 1992), soft clustering (Pereira et al., 1993), as well as new algorithms like **Clustering By Committee** (CBC) (Lin and Pantel, 2002). For particular relations, Lin et al. (2003) applied hand-crafted patterns to find **antonyms**, with the goal of improving synonym-detection. The distributional word similarity algorithms from Sec. 20.7 often incorrectly assign high similarity to antonyms. Lin et al. (2003) showed that words appearing in the patterns *from X to Y* or *either X or Y* tended to be antonyms. Girju et al. (2003, 2006) show improvements in **meronym** extraction by learning generalizations about the semantic superclasses of the two nouns. Chklovski and Pantel (2004) used hand-built patterns to extract fine-grained relations between verbs such as **strength**. Much recent work has focused on thesaurus induction by combining different relation extractors. Pantel and Ravichandran (2004), for example, extend Caraballo's algorithm for combining similarity and hyponymy information, while Snow et al. (2006) integrate multiple relation extractors to compute the most probable thesaurus structure. Recent work on similarity focuses on the use of the Web, for example relying on Wikipedia Strube and Ponzetto (2006), Gabrilovich and Markovitch (2007); this Web-based work is also closely related to unsupervised information extraction; see Ch. 22 and references like Etzioni et al. (2005).

While not as old a field as word similarity or sense disambiguation, semantic role labeling has a long history in computational linguistics. The earliest work on semantic role labeling (Simmons, 1973) first parsed a sentence using an ATN parser. Each verb then had a set of rules specifying how the parse should be mapped to semantic roles. These rules mainly made reference to grammatical functions (subject, object, complement of specific prepositions), but also checked constituent-internal features such as the animacy of head nouns.

Statistical work in the area revived in 2000 after the FrameNet and PropBank project had created databases large enough and consistent enough to make training and testing possible. Many popular features used for role labeling are defined in Gildea and Jurafsky (2002), Chen and Rambow (2003), Surdeanu et al. (2003), Xue and Palmer (2004), Pradhan et al. (2003, 2005).

To avoid the need for huge labeled training sets, recent work has focused on unsupervised approaches for semantic role labeling (Swier and Stevenson, 2004).

The semantic labeling work described above focuses on labeling each sentence token in a corpus with semantic roles. An alternative approach to semantic role labeling focuses on lexicon learning, using unsupervised learning on a corpus to learn the kinds of semantic classes a verb can belong to in terms of its possible semantic roles or

argument alternation patterns (Stevenson and Merlo, 1999; Schulte im Walde, 2000; Merlo and Stevenson, 2001; Merlo et al., 2001; Grenager and Manning, 2006).

Exercises

20.1 Collect a small corpus of example sentences of varying lengths from any newspaper or magazine. Using WordNet, or any standard dictionary, determine how many senses there are for each of the open-class words in each sentence. How many distinct combinations of senses are there for each sentence? How does this number seem to vary with sentence length?

20.2 Using WordNet, or a standard reference dictionary, tag each open-class word in your corpus with its correct tag. Was choosing the correct sense always a straightforward task. Report on any difficulties you encountered.

20.3 Using the same corpus, isolate the words taking part in all the verb-subject and verb-object relations. How often does it appear to be the case that the words taking part in these relations could be disambiguated using only information about the words in the relation?

20.4 Between the words *eat* and *find* which would you expect to be more effective in selectional restriction-based sense disambiguation? Why?

20.5 Using your favorite dictionary, simulate the Original Lesk word overlap disambiguation algorithm described on page 659 on the phrase *Time flies like an arrow*. Assume that the words are to be disambiguated one at a time, from left to right, and that the results from earlier decisions are used later in the process.

20.6 Build an implementation of your solution to the previous exercise. Using WordNet, implement the Original Lesk word overlap disambiguation algorithm described on page 659 on the phrase *Time flies like an arrow*.

20.7 Implement and experiment with a decision-list sense disambiguation system. As a model, use the kinds of features shown in Fig. 20.2. Use one of the publicly available decision-list packages like WEKA (or see Russell and Norvig (1995) for more details on implementing decision-list learning yourself). To facilitate evaluation of your system, you should obtain one of the freely available sense-tagged corpora.

20.8 Evaluate two or three of the similarity methods from the publicly available `Wordnet::Similarity` package (Pedersen et al., 2004). You might do this by hand-labeling some word pairs with similarity scores and seeing how well the algorithms approximate your hand labels.

20.9 Implement a distributional word similarity algorithm that can take different measures of association and different measures of vector similarity. Now evaluate two measures of association and two measures of vector similarity from Fig. 20.13. Again, you might do this by hand-labeling some word pairs with

similarity scores and seeing how well the algorithms approximate your hand labels.