Contents

Chapter 1

Finding the Structure of Words

Otakar Smrž and Hyun-Jo You

Human language is a complicated thing. We use it to express our thoughts, and through language, we receive information and infer its meaning. Linguistic expressions are not unorganized, though. They show structure of different kinds and complexity and consist of more elementary components whose co-occurrence in context refines the notions they refer to in isolation and implies further meaningful relations between them.

Trying to understand language en bloc is not a viable approach. Linguists have developed whole disciplines that look at language from different perspectives and at different levels of detail. The point of morphology, for instance, is to study the variable forms and functions of words, while syntax is concerned with the arrangement of words into phrases, clauses, and sentences. Word structure constraints due to pronunciation are described by phonology, whereas conventions for writing constitute the orthography of a language. The meaning of a linguistic expression is its semantics, and etymology and lexicology cover especially the evolution of words and explain the semantic, morphological, and other links among them.

Words are perhaps the most intuitive units of language, yet they are in general tricky to define. Knowing how to work with them allows, in particular, the development of syntactic and semantic abstractions and simplifies other advanced views on language. Morphology is an essential part of language processing, and in multilingual settings, it becomes even more important.

In this chapter, we explore how to identify words of distinct types in human languages, and how the internal structure of words can be modeled in connection with the grammatical properties and lexical concepts the words should represent. The discovery of word structure is **morphological parsing**.

How difficult can such tasks be? It depends. In many languages, words are delimited in the orthography by whitespace and punctuation. But in many other languages, the writing system leaves it up to the reader to tell words apart or determine their exact phonological forms. Some languages use words whose form need not change much with the varying context; others are highly sensitive about the choice of word forms according to particular syntactic and semantic constraints and restrictions.

1.1 Words and Their Components

Words are defined in most languages as the smallest linguistic units that can form a complete utterance by themselves. The minimal parts of words that deliver aspects of meaning to them are called morphemes. Depending on the means of communication, morphemes are spelled out via graphemes—symbols of writing such as letters or characters—or are realized through phonemes, the distinctive units of sound in spoken language.[1] It is not always easy to decide and agree on the precise boundaries discriminating words from morphemes and from phrases [1, 2].

1.1.1 Tokens

Suppose, for a moment, that words in English are delimited only by whitespace and punctuation [3], and consider Example 1–1:

EXAMPLE 1–1: Will you read the newspaper? Will you read it? I won't read it.

If we confront our assumption with insights from etymology and syntax, we notice two words here: *newspaper* and *won't*. Being a compound word, *newspaper* has an interesting derivational structure. We might wish to describe it in more detail, once there is a lexicon or some other linguistic evidence on which to build the possible hypotheses about the origins of the word. In writing, *newspaper* and the associated concept is distinguished from the isolated *news* and *paper*. In speech, however, the distinction is far from clear, and identification of words becomes an issue of its own.

For reasons of generality, linguists prefer to analyze *won't* as two syntactic words, or tokens, each of which has its independent role and can be reverted to its normalized form. The structure of *won't* could be parsed as *will* followed by *not*. In English, this kind of **tokenization** and **normalization** may apply to just a limited set of cases, but in other languages, these phenomena have to be treated in a less trivial manner.

In Arabic or Hebrew [4], certain tokens are concatenated in writing with the preceding or the following ones, possibly changing their forms as well. The underlying lexical or syntactic units are thereby blurred into one compact string of letters and no longer appear as distinct words. Tokens behaving in this way can be found in various languages and are often called clitics.

In the writing systems of Chinese, Japanese [5], and Thai, whitespace is not used to separate words. The units that are delimited graphically in some way are sentences or clauses. In Korean, character strings are called *eojeol* 'word segment' and roughly correspond to speech or cognitive units, which are usually larger than words and smaller than clauses [6], as shown in Example 1–2:

EXAMPLE 1–2: 학생들에게만 주셨는데
hak.sayng.tul.ey.key.man cwu.syess.nun.te[2]
haksayng-tul-eykey-man cwu-si-ess-nunte
student+*plural*+*dative*+only give+*honorific*+*past*+while
while (he/she) gave (it) only to the students

1. Signs used in sign languages are composed of elements denoted as phonemes, too.
2. We use the Yale romanization of the Korean script and indicate its original characters by dots. Hyphens mark morphological boundaries, and tokens are separated by plus symbols.

Nonetheless, the elementary morphological units are viewed as having their own syntactic status [7]. In such languages, tokenization, also known as **word segmentation**, is the fundamental step of morphological analysis and a prerequisite for most language processing applications.

1.1.2 Lexemes

By the term word, we often denote not just the one linguistic form in the given context but also the concept behind the form and the set of alternative forms that can express it. Such sets are called lexemes or lexical items, and they constitute the lexicon of a language. Lexemes can be divided by their behavior into the lexical categories of verbs, nouns, adjectives, conjunctions, particles, or other parts of speech. The citation form of a lexeme, by which it is commonly identified, is also called its lemma.

When we convert a word into its other forms, such as turning the singular *mouse* into the plural *mice* or *mouses*, we say we inflect the lexeme. When we transform a lexeme into another one that is morphologically related, regardless of its lexical category, we say we derive the lexeme: for instance, the nouns *receiver* and *reception* are derived from the verb *to receive*.

EXAMPLE 1–3: Did you see him? I didn't see him. I didn't see anyone.

Example 1–3 presents the problem of tokenization of *didn't* and the investigation of the internal structure of *anyone*. In the paraphrase *I saw no one*, the lexeme *to see* would be inflected into the form *saw* to reflect its grammatical function of expressing positive past tense. Likewise, *him* is the oblique case form of *he* or even of a more abstract lexeme representing all personal pronouns. In the paraphrase, *no one* can be perceived as the minimal word synonymous with *nobody*. The difficulty with the definition of what counts as a word need not pose a problem for the syntactic description if we understand *no one* as two closely connected tokens treated as one fixed element.

In the Czech translation of Example 1–3, the lexeme *vidět* 'to see' is inflected for past tense, in which forms comprising two tokens are produced in the second and first person (i.e., *viděla jsi* 'you-FEM-SG saw' and *neviděla jsem* 'I-FEM-SG did not see'). Negation in Czech is an inflectional parameter rather than just syntactic and is marked both in the verb and in the pronoun of the latter response, as in Example 1–4:

EXAMPLE 1–4: Vidělas ho? Neviděla jsem ho. Neviděla jsem nikoho.
saw+you-are him? not-saw I-am him. not-saw I-am no-one.

Here, *vidělas* is the contracted form of *viděla jsi* 'you-FEM-SG saw'. The *s* of *jsi* 'you are' is a clitic, and due to free word order in Czech, it can be attached to virtually any part of speech. We could thus ask a question like *Nikoho nevidělas?* 'Did you see no one?' in which the pronoun *nikoho* 'no one' is followed by this clitic.

1.1.3 Morphemes

Morphological theories differ on whether and how to associate the properties of word forms with their structural components [8, 9, 10, 11]. These components are usually called **segments** or **morphs**. The morphs that by themselves represent some aspect of the meaning of a word are called **morphemes** of some function.

Human languages employ a variety of devices by which morphs and morphemes are combined into word forms. The simplest morphological process concatenates morphs one by one, as in *dis-agree-ment-s*, where *agree* is a free lexical morpheme and the other elements are bound grammatical morphemes contributing some partial meaning to the whole word.

In a more complex scheme, morphs can interact with each other, and their forms may become subject to additional phonological and orthographic changes denoted as morphophonemic. The alternative forms of a morpheme are termed allomorphs.

Examples of morphological alternation and phonologically dependent choice of the form of a morpheme are abundant in the Korean language. In Korean, many morphemes change their forms systematically with the phonological context. Example 1-5 lists the allomorphs *-ess-*, *-ass-*, *-pess-* of the temporal marker indicating past tense. The first two alter according to the phonological condition of the preceding verb stem; the last one is used especially for the verb *ha-* 'do'. The appropriate allomorph is merely concatenated after the stem, or it can be further contracted with it, as was *-si-ess-* into *-syess-* in Example 1-2. During morphological parsing, normalization of allomorphs into some canonical form of the morpheme is desirable, especially because the contraction of morphs interferes with simple segmentation:

EXAMPLE 1-5:

	concatenated		contracted		
(a)	보았-	*po-ass-*	봤-	*pwass-*	'have seen'
(b)	가지 었-	*ka.ci-ess-*	가졌-	*ka.cyess-*	'have taken'
(c)	하였-	*ha-yess-*	했-	*hayss-*	'have done'
(d)	되었-	*toy-ess-*	됐-	*twayss-*	'have become'
(e)	놓았-	*noh-ass-*	놨-	*nwass-*	'have put'

Contractions (a, b) are ordinary but require attention because two characters are reduced into one. Other types (c, d, e) are phonologically unpredictable, or lexically dependent. For example, *coh-ass-* 'have been good' may never be contracted, whereas *noh-* and *-ass-* are merged into *nwass-* in (e).

There are yet other linguistic devices of word formation to account for, as the morphological process itself can get less trivial. The concatenation operation can be complemented with infixation or intertwining of the morphs, which is common, for instance, in Arabic. Nonconcatenative inflection by modification of the internal vowel of a word occurs even in English: compare the sounds of *mouse* and *mice*, *see* and *saw*, *read* and *read*.

Notably in Arabic, internal inflection takes place routinely and has a yet different quality. The internal parts of words, called stems, are modeled with root and pattern morphemes. Word structure is then described by templates abstracting away from the root but showing the pattern and all the other morphs attached to either side of it.

EXAMPLE 1-6: `hl stqrO hwh Aljrā}d?`[3] هل ستقرأ هذه الجرائد؟

hal sa-taqraʔu hāḏihi ʔ-ǧarāʔida?

whether will+you-read this the-newspapers?

`hl stqrWhA? ln OqrWhA.` هل ستقرؤها؟ لن أقرأها.

hal sa-taqraʔuhā? lan ʔaqraʔuhā.

whether will+you-read+it? not-will I-read+it.

3. The original Arabic script is transliterated using Buckwalter notation. For readability, we also provide the standard phonological transcription, which reduces ambiguity.

The meaning of Example 1-6 is similar to that of Example 1-1, only the phrase *hāḏihi ʔ-ǧarāʔida* refers to 'these newspapers'. While *sa-taqraʔu* 'you will read' combines the future marker *sa-* with the imperfective second-person masculine singular verb *taqraʔu* in the indicative mood and active voice, *sa-taqraʔuhā* 'you will read it' also adds the cliticized feminine singular personal pronoun in the accusative case.[4]

The citation form of the lexeme to which *taqraʔu* 'you-MASC-SG read' belongs is *qaraʔa*, roughly 'to read'. This form is classified by linguists as the basic verbal form represented by the template *faʕal* merged with the consonantal root *q r ʔ*, where the *f · ʕ · l* symbols of the template are substituted by the respective root consonants. Inflections of this lexeme can modify the pattern *faʕal* of the stem of the lemma into *faʕ* and concatenate it, under rules of morphophonemic changes, with further prefixes and suffixes. The structure of *taqraʔu* is thus parsed into the template *ta-faʕ-aʔu* and the invariant root.

The word *al-ǧarāʔida* 'the newspapers' in the accusative case and definite state is another example of internal inflection. Its structure follows the template *al-faʕāʔil-a* with the root *ǧ r d*. This word is the plural of *ǧarīdah* 'newspaper' with the template *faʕīl-ah*. The links between singular and plural templates are subject to convention and have to be declared in the lexicon.

Irrespective of the morphological processes involved, some properties or features of a word need not be apparent explicitly in its morphological structure. Its existing structural components may be paired with and depend on several functions simultaneously but may have no particular grammatical interpretation or lexical meaning.

The *-ah* suffix of *ǧarīdah* 'newspaper' corresponds with the inherent feminine gender of the lexeme. In fact, the *-ah* morpheme is commonly, though not exclusively, used to mark the feminine singular forms of adjectives: for example, *ǧadīd* becomes *ǧadīdah* 'new'. However, the *-ah* suffix can be part of words that are not feminine, and there its function can be seen as either emptied or overridden [12]. In general, linguistic forms should be distinguished from functions, and not every morph can be assumed to be a morpheme.

1.1.4 Typology

Morphological typology divides languages into groups by characterizing the prevalent morphological phenomena in those languages. It can consider various criteria, and during the history of linguistics, different classifications have been proposed [13, 14]. Let us outline the typology that is based on quantitative relations between words, their morphemes, and their features:

Isolating, or **analytic**, languages include no or relatively few words that would comprise more than one morpheme (typical members are Chinese, Vietnamese, and Thai; analytic tendencies are also found in English).

Synthetic languages can combine more morphemes in one word and are further divided into agglutinative and fusional languages.

Agglutinative languages have morphemes associated with only a single function at a time (as in Korean, Japanese, Finnish, and Tamil, etc.).

4. The logical plural of things is formally treated as feminine singular in Arabic.

Fusional languages are defined by their feature-per-morpheme ratio higher than one (as in Arabic, Czech, Latin, Sanskrit, German, etc.).

In accordance with the notions about word formation processes mentioned earlier, we can also discern:

Concatenative languages linking morphs and morphemes one after another.

Nonlinear languages allowing structural components to merge nonsequentially to apply tonal morphemes or change the consonantal or vocalic templates of words.

While some morphological phenomena, such as orthographic collapsing, phonological contraction, or complex inflection and derivation, are more dominant in some languages than in others, in principle, we can find, and should be able to deal with, instances of these phenomena across different language families and typological classes.

1.2 Issues and Challenges

Morphological parsing tries to eliminate or alleviate the variability of word forms to provide higher-level linguistic units whose lexical and morphological properties are explicit and well defined. It attempts to remove unnecessary irregularity and give limits to ambiguity, both of which are present inherently in human language.

By irregularity, we mean existence of such forms and structures that are not described appropriately by a prototypical linguistic model. Some irregularities can be understood by redesigning the model and improving its rules, but other lexically dependent irregularities often cannot be generalized.

Ambiguity is indeterminacy in interpretation of expressions of language. Next to accidental ambiguity and ambiguity due to lexemes having multiple senses, we note the issue of **syncretism**, or systematic ambiguity.

Morphological modeling also faces the problem of productivity and creativity in language, by which unconventional but perfectly meaningful new words or new senses are coined. Usually, though, words that are not licensed in some way by the lexicon of a morphological system will remain completely unparsed. This **unknown word** problem is particularly severe in speech or writing that gets out of the expected domain of the linguistic model, such as when special terms or foreign names are involved in the discourse or when multiple languages or dialects are mixed together.

1.2.1 Irregularity

Morphological parsing is motivated by the quest for generalization and abstraction in the world of words. Immediate descriptions of given linguistic data may not be the ultimate ones, due to either their inadequate accuracy or inappropriate complexity, and better formulations may be needed. The design principles of the morphological model are therefore very important.

In Arabic, the deeper study of the morphological processes that are in effect during inflection and derivation, even for the so-called irregular words, is essential for mastering the

whole morphological and phonological system. With the proper abstractions made, irregular morphology can be seen as merely enforcing some extended rules, the nature of which is phonological, over the underlying or prototypical regular word forms [15, 16].

EXAMPLE 1–7: hl rOyth? lm Orh. lm Or OHdA. هل رأيته؟ لم أره. لم أر أحدا.
hal ra'aytihi? lam 'arahu. lam 'ara 'aḥadan.
whether you-saw+him? not-did I-see+him. not-did I-see anyone.

In Example 1–7, *ra'ayti* is the second-person feminine singular perfective verb in active voice, member of the *ra'ā* 'to see' lexeme of the r ' y root. The prototypical, regularized pattern for this citation form is *fa'al*, as we saw with *qara'* in Example 1–6. Alternatively, we could assume the pattern of *ra'ā* to be *fa'ā*, thereby asserting in a compact way that the final root consonant and its vocalic context are subject to the particular phonological change, resulting in *ra'ā* like *fa'ā* instead of *ra'ay* like *fa'al*. The occurrence of this change in the citation form may have possible implications for the morphological behavior of the whole lexeme.

Table 1–1 illustrates differences between a naive model of word structure in Arabic and the model proposed in Smrž [12] and Smrž and Bielický [17] where morphophonemic merge rules and templates are involved. Morphophonemic templates capture morphological processes by just organizing stem patterns and generic affixes without any context-dependent variation of the affixes or ad hoc modification of the stems. The merge rules, indeed very terse, then ensure that such structured representations can be converted into exactly the surface forms, both orthographic and phonological, used in the natural language. Applying the merge rules is independent of and irrespective of any grammatical parameters or information other than that contained in a template. Most morphological irregularities are thus successfully removed.

Table 1–1: Discovering the regularity of Arabic morphology using morphophonemic templates, where uniform structural operations apply to different kinds of stems. In rows, surface forms S of *qara'* 'to read' and *ra'ā* 'to see' and their inflections are analyzed into immediate I and morphophonemic M templates, in which dashes mark the structural boundaries where merge rules are enforced. The outer columns of the table correspond to P perfective and I imperfective stems declared in the lexicon; the inner columns treat active verb forms of the following morphosyntactic properties: I indicative, S subjunctive, J jussive mood; 1 first, 2 second, 3 third person; M masculine, F feminine gender; S singular, P plural number

P-STEM	P-3MS	P-2FS	P-3MP	II2MS	IS1-S	IJ1-S	I-STEM	
qara'	*qara'a*	*qara'ti*	*qara'ū*	*taqra'u*	*'aqra'a*	*'aqra'*	*qra'*	S
fa'al	*fa'al-a*	*fa'al-ti*	*fa'al-ū*	*ta-f'al-u*	*'a-f'al-a*	*'a-f'al*	*f'al*	I
fa'al	*fa'al-a*	*fa'al-ti*	*fa'al-ū*	*ta-f'al-u*	*'a-f'al-a*	*'a-f'al-*	*f'al*	M
...	...-a	...-ti	...-ū	ta-...-u	'a-...-a	'a-...-	...	
fa'ā	*fa'ā-a*	*fa'ā-ti*	*fa'ā-ū*	*ta-f'ā-u*	*'a-f'ā-a*	*'a-f'ā-*	*f'ā*	M
fa'ā	*fa'ā*	*fa'al-ti*	*fa-aw*	*ta-f'ā*	*'a-f'ā*	*'a-f'a*	*f'ā*	I
ra'ā	*ra'ā*	*ra'ayti*	*ra'aw*	*tarā*	*'arā*	*'ara*	*rā*	S

Table 1–2: Examples of major Korean irregular verb classes compared with regular verbs

Base Form		(-e)		Meaning	Comment
집-	*cip-*	집어	*cip.e*	'pick'	regular
깁-	*kip-*	기워	*ki.we*	'sew'	*p*-irregular
믿-	*mit-*	믿어	*mit.e*	'believe'	regular
싣-	*sit-*	실어	*sil.e*	'load'	*t*-irregular
씻-	*ssis-*	씻어	*ssis.e*	'wash'	regular
잇-	*is-*	이어	*i.e*	'link'	*s*-irregular
낳-	*nah-*	낳아	*nah.a*	'bear'	regular
까맣-	*kka.mah-*	까매	*kka.may*	'be black'	*h*-irregular
치르-	*chi.lu-*	치러	*chi.le*	'pay'	regular u-ellipsis
이르-	*i.lu-*	이르러	*i.lu.le*	'reach'	*le*-irregular
흐르-	*hu.lu-*	흘러	*hul.le*	'flow'	*lu*-irregular

In contrast, some irregularities are bound to particular lexemes or contexts, and cannot be accounted for by general rules. Korean irregular verbs provide examples of such irregularities.

Korean shows exceptional constraints on the selection of grammatical morphemes. It is hard to find irregular inflection in other agglutinative languages: two irregular verbs in Japanese [18], one in Finnish [19]. These languages are abundant with morphological alternations that are formalized by precise phonological rules. Korean additionally features lexically dependent stem alternation. As in many other languages, *i-* 'be' and *ha-* 'do' have unique irregular endings. Other irregular verbs are classified by the stem final phoneme. Table 1–2 compares major irregular verb classes with regular verbs in the same phonological condition.

1.2.2 Ambiguity

Morphological ambiguity is the possibility that word forms be understood in multiple ways out of the context of their discourse. Words forms that look the same but have distinct functions or meaning are called homonyms.

Ambiguity is present in all aspects of morphological processing and language processing at large. Morphological parsing is not concerned with complete disambiguation of words in their context, however; it can effectively restrict the set of valid interpretations of a given word form [20, 21].

In Korean, homonyms are one of the most problematic objects in morphological analysis because they prevail all around frequent lexical items. Table 1–3 arranges homonyms on the basis of their behavior with different endings. Example 1–8 is an example of homonyms through nouns and verbs.

Table 1–3: Systematic homonyms arise as verbs combined with endings in Korean

(-*ko*)		(-e)		(-un)		Meaning
묻고	*mwut.ko*	묻어	*mwut.e*	묻은	*mwut.un*	'bury'
묻고	*mwut.ko*	물어	*mwul.e*	물은	*mwul.un*	'ask'
물고	*mwul.ko*	물어	*mwul.e*	문	*mwun*	'bite'
걷고	*ket.ko*	걷어	*ket.e*	걷은	*ket.un*	'roll up'
걷고	*ket.ko*	걸어	*kel.e*	걸은	*kel.un*	'walk'
걸고	*kel.ko*	걸어	*kel.e*	건	*ken*	'hang'
굽고	*kwup.ko*	굽어	*kwup.e*	굽은	*kwup.un*	'be bent'
굽고	*kwup.ko*	구워	*kwu.we*	구운	*kwu.wun*	'bake'
이르고	*i.lu.ko*	이르러	*i.lu.le*	이른	*i.lun*	'reach'
이르고	*i.lu.ko*	일러	*il.le*	이른	*i.lun*	'say'

EXAMPLE 1–8:
난 'orchid' ← 난 *nan* 'orchid'
난 'I' ← 나 *na* 'I' + -*n* (topic)
난 'which flew' ← 날- *nal-* 'fly' + -*n* (relative, past)
난 'which got out' ← 나- *na-* 'get out' + -*n* (relative, past)

We could also consider ambiguity in the senses of the noun *nan*, according to the Standard Korean Language Dictionary: *nan*[1] 'egg', *nan*[2] 'revolt', *nan*[5] 'section (in newspaper)', *nan*[6] 'orchid', plus several infrequent readings.

Arabic is a language of rich morphology, both derivational and inflectional. Because Arabic script usually does not encode short vowels and omits yet some other diacritical marks that would record the phonological form exactly, the degree of its morphological ambiguity is considerably increased. In addition, Arabic orthography collapses certain word forms together. The problem of morphological disambiguation of Arabic encompasses not only the resolution of the structural components of words and their actual morphosyntactic properties (i.e., morphological tagging [22, 23, 24]) but also tokenization and normalization [25], lemmatization, stemming, and diacritization [26, 27, 28].

When inflected syntactic words are combined in an utterance, additional phonological and orthographic changes can take place, as shown in Figure 1-1. In Sanskrit, one such euphony rule is known as external *sandhi* [29, 30]. Inverting *sandhi* during tokenization is usually nondeterministic in the sense that it can provide multiple solutions. In any language, tokenization decisions may impose constraints on the morphosyntactic properties of the tokens being reconstructed, which then have to be respected in further processing. The tight coupling between morphology and syntax has inspired proposals for disambiguating them jointly rather than sequentially [4].

Czech is a highly inflected fusional language. Unlike agglutinative languages, inflectional morphemes often represent several functions simultaneously, and there is no particular one-to-one correspondence between their forms and functions. Inflectional paradigms

dirāsatī	دراستي	drAsty	→	*dirāsatu ī*	دراسة ي	drAsp y
			→	*dirāsati ī*	دراسة ي	drAsp y
			→	*dirāsata ī*	دراسة ي	drAsp y
muʿallimīya	معلمي	mElmy	→	*muʿallimū ī*	معلمو ي	mElmw y
			→	*muʿallimī ī*	معلمي ي	mElmy y
kataʾtumūhā	كتبتموها	ktbtmwhA	→	*kataʾtum hā*	كتبتم ها	ktbtm hA
yiğrāʾuhu	إجراؤه	IjrAŵh	→	*yiğrāʾu hu*	إجراء ه	IjrA' h
yiğrāʾihi	إجرائه	IjrAʾh	→	*yiğrāʾi hu*	إجراء ه	IjrA' h
yiğrāʾahu	إجراءه	IjrA'h	→	*yiğrāʾa hu*	إجراء ه	IjrA' h
li-ʾl-ʾasafi	للأسف	llAsf	→	*li ʾl-ʾasafi li*	ل الأسف	l AlAsf

Figure 1–1: Complex tokenization and normalization of euphony in Arabic. Three nominal cases are expressed by the same word form with *dirāsatī* 'my study' and *muʿallimīya* 'my teachers', but the original case endings are distinct. In *kataʾtumūhā* 'you-MASC-PL wrote them', the liaison vowel u is dropped when tokenized. Special attention is needed to normalize some orthographic conventions, such as the interaction of *yiğrāʾ* 'carrying out' and the cliticized *hu* 'his' respecting the case ending or the merge of the definite article of *ʾasaf* 'regret' with the preposition *li* 'for'

(i.e., schemes for finding the form of a lexeme associated with the required properties) in Czech are of numerous kinds, yet they tend to include nonunique forms in them.

Table 1–4 lists the paradigms of several common Czech words. Inflectional paradigms for nouns depend on the grammatical gender and the phonological structure of a lexeme. The individual forms in a paradigm vary with grammatical number and case, which are the free parameters imposed only by the context in which a word is used.

Looking at the morphological variation of the word *stavení* 'building', we might wonder why we should distinguish all the cases for it when this lexeme can take only four different forms. Is the detail of the case system appropriate? The answer is yes, because we can find linguistic evidence that leads to this case category abstraction. Just consider other words of the same meaning in place of *stavení* in various contexts. We conclude that there is indeed a case distinction made by the underlying system, but it need not necessarily be expressed clearly and uniquely in the form of words.

The morphological phenomenon that some words or word classes show instances of systematic homonymy is called syncretism. In particular, homonymy can occur due to **neutralization** and **uninflectedness** with respect to some morphosyntactic parameters. These cases of morphological syncretism are distinguished by the ability of the context to demand the morphosyntactic properties in question, as stated by Baerman, Brown, and Corbett [10, p. 32]:

> Whereas *neutralization* is about syntactic irrelevance as reflected in morphology, *uninflectedness* is about morphology being unresponsive to a feature that is syntactically relevant.

For example, it seems fine for syntax in Czech or Arabic to request the personal pronoun of the first-person feminine singular, equivalent to 'I', despite it being homonymous with

Table 1–4: Morphological paradigms of the Czech words *dům* 'house', *budova* 'building', *stavba* 'building', *stavení* 'building'. Despite systematic ambiguities in them, the space of inflectional parameters could not be reduced without losing the ability to capture all distinct forms elsewhere: S singular, P plural number; 1 nominative, 2 genitive, 3 dative, 4 accusative, 5 vocative, 6 locative, 7 instrumental case

	Masculine inanimate	Feminine	Feminine	Neuter
S1	dům	budova	stavba	stavení
S2	domu	budovy	stavby	stavení
S3	domu	budově	stavbě	stavení
S4	dům	budovu	stavbu	stavení
S5	dome	budovo	stavbo	stavení
S6	domu / domě	budově	stavbě	stavení
S7	domem	budovou	stavbou	stavením
P1	domy	budovy	stavby	stavení
P2	domů	budov	staveb	stavení
P3	domům	budovám	stavbám	stavením
P4	domy	budovy	stavby	stavení
P5	domy	budovy	stavby	stavení
P6	domech	budovách	stavbách	stavením
P7	domy	budovami	stavbami	stavením

the first-person masculine singular. The reason is that for some other values of the person category, the forms of masculine and feminine gender are different, and there exist syntactic dependencies that do take gender into account. It is not the case that the first-person singular pronoun would have no gender nor that it would have both. We just observe uninflectedness here. On the other hand, we might claim that in English or Korean, the gender category is syntactically neutralised if it ever was present, and the nuances between *he* and *she*, *him* and *her*, *his* and *hers* are only semantic.

With the notion of paradigms and syncretism in mind, we should ask what is the minimal set of combinations of morphosyntactic inflectional parameters that covers the inflectional variability in a language. Morphological models that would like to define a joint system of underlying morphosyntactic properties for multiple languages would have to generalize the parameter space accordingly and neutralize any systematically void configurations.

1.2.3 Productivity

Is the inventory of words in a language finite, or is it unlimited? This question leads directly to discerning two fundamental approaches to language, summarized in the distinction between *langue* and *parole* by Ferdinand de Saussure, or in the competence versus performance duality by Noam Chomsky.

In one view, language can be seen as simply a collection of utterances (parole) actually pronounced or written (performance). This ideal data set can in practice be approximated by linguistic corpora, which are finite collections of linguistic data that are studied with empirical methods and can be used for comparison when linguistic models are developed.

Yet, if we consider language as a system (langue), we discover in it structural devices like recursion, iteration, or compounding that allow to produce (competence) an infinite set of concrete linguistic utterances. This general potential holds for morphological processes as well and is called morphological productivity [31, 32].

We denote the set of word forms found in a corpus of a language as its vocabulary. The members of this set are word types, whereas every original instance of a word form is a word token.

The distribution of words [33] or other elements of language follows the "80/20 rule," also known as the law of the vital few. It says that most of the word tokens in a given corpus can be identified with just a couple of word types in its vocabulary, and words from the rest of the vocabulary occur much less commonly if not rarely in the corpus. Furthermore, new, unexpected words will always appear as the collection of linguistic data is enlarged.

In Czech, negation is a productive morphological operation. Verbs, nouns, adjectives, and adverbs can be prefixed with *ne-* to define the complementary lexical concept. In Example 1-9, *budeš* 'you will be' is the second-person singular of *být* 'to be', and *nebudu* 'I will not be' is the first-person singular of *nebýt*, the negated *být*. We could easily have *číst* 'to read' and *nečíst* 'not to read', or we could create an adverbial phrase like *noviny nenoviny* that would express 'indifference to newspapers' in general:

EXAMPLE 1-9: Budeš číst ty noviny? Budeš je číst? Nebudu je číst.
you-will read the newspaper? you-will it read? not-I-will it read.

Example 1-9 has the meaning of Example 1-1 and Example 1-6. The word *noviny* 'newspaper' exists only in plural whether it signifies one piece of newspaper or many of them. We can literally translate *noviny* as the plural of *novina* 'news' to see the origins of the word as well as the fortunate analogy with English.

It is conceivable to include all negated lexemes into the lexicon and thereby again achieve a finite number of word forms in the vocabulary. Generally, though, the richness of a morphological system of a language can make this approach highly impractical.

Most languages contain words that allow some of their structural components to repeat freely. Consider the prefix *pra-* related to a notion of 'generation' in Czech and how it can or cannot be iterated, as shown in Example 1-10:

EXAMPLE 1-10:

vnuk 'grandson'	*pravnuk* 'great-grandson'
	prapra...vnuk 'great-great-...grandson'
les 'forest'	*prales* 'jungle', 'virgin forest'
zdroj 'source'	*prazdroj* 'urquell', 'original source'
starý 'old'	*pradávný* 'time-honored', 'dateless'

In creative language, such as in blogs, chats, and emotive informal communication, iteration is often used to accent intensity of expression. Creativity may, of course, go beyond the rules of productivity itself [32].

Let us give an example where creativity, productivity, and the issue of unknown words meet nicely. According to Wikipedia, the word *googol* is a made-up word denoting the number "one followed by one hundred zeros," and the name of the company Google is an inadvertent misspelling thereof. Nonetheless, both of these words successfully entered the lexicon of English where morphological productivity started working, and we now know the verb *to google* and nouns like *googling* or even *googlish* or *googleology* [34].

The original names have been adopted by other languages, too, and their own morphological processes have been triggered. In Czech, one says *googlovat*, *googlit* 'to google' or *vygooglovat*, *vygooglit* 'to google out', *googlování* 'googling', and so on. In Arabic, the names are transcribed as *ǧūgāl* 'googol' and *ǧūgil* 'Google'. The latter one got transformed to the verb *ǧawǧal* 'to google' through internal inflection, as if there were a genuine root *ǧ w ǧ l*, and the corresponding noun *ǧawǧalah* 'googling' exists as well.

1.3 Morphological Models

There are many possible approaches to designing and implementing morphological models. Over time, computational linguistics has witnessed the development of a number of formalisms and frameworks, in particular grammars of different kinds and expressive power, with which to address whole classes of problems in processing natural as well as formal languages.

Various domain-specific programming languages have been created that allow us to implement the theoretical problem using hopefully intuitive and minimal programming effort. These special-purpose languages usually introduce idiosyncratic notations of programs and are interpreted using some restricted model of computation. The motivation for such approaches may partly lie in the fact that, historically, computational resources were too limited compared to the requirements and complexity of the tasks being solved. Other motivations are theoretical given that finding a simple but accurate and yet generalizing model is the point of scientific abstraction.

There are also many approaches that do not resort to domain-specific programming. They, however, have to take care of the runtime performance and efficiency of the computational model themselves. It is up to the choice of the programming methods and the design style whether such models turn out to be pure, intuitive, adequate, complete, reusable, elegant, or not.

Let us now look at the most prominent types of computational approaches to morphology. Needless to say, this typology is not strictly exclusive in the sense that comprehensive morphological models and their applications can combine various distinct implementational aspects, discussed next.

1.3.1 Dictionary Lookup

Morphological parsing is a process by which word forms of a language are associated with corresponding linguistic descriptions. Morphological systems that specify these associations by merely enumerating them case by case do not offer any generalization means. Likewise for systems in which analyzing a word form is reduced to looking it up verbatim in word

lists, dictionaries, or databases, unless they are constructed by and kept in sync with more sophisticated models of the language.

In this context, a dictionary is understood as a data structure that directly enables obtaining some precomputed results, in our case word analyses. The data structure can be optimized for efficient lookup, and the results can be shared. Lookup operations are relatively simple and usually quick. Dictionaries can be implemented, for instance, as lists, binary search trees, tries, hash tables, and so on.

Because the set of associations between word forms and their desired descriptions is declared by plain enumeration, the coverage of the model is finite and the generative potential of the language is not exploited. Developing as well as verifying the association list is tedious, liable to errors, and likely inefficient and inaccurate unless the data are retrieved automatically from large and reliable linguistic resources.

Despite all that, an enumerative model is often sufficient for the given purpose, deals easily with exceptions, and can implement even complex morphology. For instance, dictionary-based approaches to Korean [35] depend on a large dictionary of all possible combinations of allomorphs and morphological alternations. These approaches do not allow development of reusable morphological rules, though [36].

The word list or dictionary-based approach has been used frequently in various ad hoc implementations for many languages. We could assume that with the availability of immense online data, extracting a high-coverage vocabulary of word forms is feasible these days [37]. The question remains how the associated annotations are constructed and how informative and accurate they are. References to the literature on the unsupervised learning and induction of morphology, which are methods resulting in structured and therefore nonenumerative models, are provided later in this chapter.

1.3.2 Finite-State Morphology

By finite-state morphological models, we mean those in which the specifications written by human programmers are directly compiled into finite-state transducers. The two most popular tools supporting this approach, which have been cited in literature and for which example implementations for multiple languages are available online, include XFST (Xerox Finite-State Tool) [9] and LexTools [11].[5]

Finite-state transducers are computational devices extending the power of finite-state automata. They consist of a finite set of nodes connected by directed edges labeled with pairs of input and output symbols. In such a network or graph, nodes are also called states, while edges are called arcs. Traversing the network from the set of initial states to the set of final states along the arcs is equivalent to reading the sequences of encountered input symbols and writing the sequences of corresponding output symbols.

The set of possible sequences accepted by the transducer defines the input language; the set of possible sequences emitted by the transducer defines the output language. For example, a finite-state transducer could translate the infinite regular language consisting of the words *vnuk*, *pravnuk*, *prapravnuk*, ... to the matching words in the infinite regular language defined by *grandson*, *great-grandson*, *great-great-grandson*, ...

5. See http://www.fsmbook.com/ and http://compling.ai.uiuc.edu/catma/ respectively.

The role of finite-state transducers is to capture and compute **regular relations** on sets [38, 9, 11].[6] That is, transducers specify relations between the input and output languages. In fact, it is possible to invert the domain and the range of a relation, that is, exchange the input and the output. In finite-state computational morphology, it is common to refer to the input word forms as **surface strings** and to the output descriptions as **lexical strings**, if the transducer is used for morphological analysis, or vice versa, if it is used for morphological generation.

The linguistic descriptions we would like to give to the word forms and their components can be rather arbitrary and are obviously dependent on the language processed as well as on the morphological theory followed. In English, a finite-state transducer could analyze the surface string `children` into the lexical string `child [+plural]`, for instance, or generate `women` from `woman [+plural]`. For other examples of possible input and output strings, consider Example 1-8 or Figure 1-1.

Relations on languages can also be viewed as functions. Let us have a relation $\mathcal{R}$, and let us denote by $[\Sigma]$ the set of all sequences over some set of symbols Σ, so that the domain and the range of $\mathcal{R}$ are subsets of $[\Sigma]$. We can then consider $\mathcal{R}$ as a function mapping an input string into a set of output strings, formally denoted by this type signature, where $[\Sigma]$ equals *String*:

$$\mathcal{R} :: [\Sigma] \rightarrow \{[\Sigma]\} \qquad \mathcal{R} :: String \rightarrow \{String\} \tag{1.1}$$

Finite-state transducers have been studied extensively for their formal algebraic properties and have proven to be suitable models for miscellaneous problems [9]. Their applications encoding the surface rather than lexical string associations as **rewrite rules** of phonology and morphology have been around since the two-level morphology model [39], further presented in *Computational Approaches to Morphology and Syntax* [11] and *Morphology and Computation* [40].

Morphological operations and processes in human languages can, in the overwhelming number of cases and to a sufficient degree, be expressed in finite-state terms. Beesley and Karttunen [9] stress concatenation of transducers as the method for factoring surface and lexical languages into simpler models and propose a somewhat unsystematic **compile-replace** transducer operation for handling nonconcatenative phenomena in morphology. Roark and Sproat [11], however, argue that building morphological models in general using transducer composition, which is pure, is a more universal approach.

A theoretical limitation of finite-state models of morphology is the problem of capturing **reduplication** of words or their elements (e.g., to express plurality) found in several human languages. A formal language that contains only words of the form λ^{1+k}, where λ is some arbitrary sequence of symbols from an alphabet and $k \in \{1, 2, \ldots\}$ is an arbitrary natural number indicating how many times λ is repeated after itself, is not a regular language, not even a context-free language. General reduplication of strings of unbounded length is thus not a regular-language operation. Coping with this problem in the framework of finite-state transducers is discussed by Roark and Sproat [11].

6. Regular relations and regular languages are restricted in their structure by the limited memory of the device (i.e., the finite set of configurations in which it can occur). Unlike with regular languages, intersection of regular relations can in general yield nonregular results [38].

Finite-state technology can be applied to the morphological modeling of isolating and agglutinative languages in a quite straightforward manner. Korean finite-state models are discussed by Kim et al. [41], Lee and Rim [42], and Han [43], to mention a few. For treatments of nonconcatenative morphology using finite-state frameworks, see especially Kay [44], Beesley [45], Kiraz [46], and Habash, Rambow, and Kiraz [47]. For comparison with finite-state models of the rich morphology of Czech, compare Skoumalová [48] and Šedláček and Smrž [49].

Implementing a refined finite-state morphological model requires careful fine-tuning of its lexicons, rewrite rules, and other components, while extending the code can lead to unexpected interactions in it, as noted by Oazer [50]. Convenient specification languages like those mentioned previously are needed because encoding the finite-state transducers directly would be extremely arduous, error prone, and unintelligible.

Finite-state tools are available in most general-purpose programming languages in the form of support for regular expression matching and substitution. While these may not be the ultimate choice for building full-fledged morphological analyzers or generators of a natural language, they are very suitable for developing tokenizers and morphological guessers capable of suggesting at least some structure for words that are formed correctly but cannot be identified with concrete lexemes during full morphological parsing [9].

1.3.3 Unification-Based Morphology

Unification-based approaches to morphology have been inspired by advances in various formal linguistic frameworks aiming at enabling complete grammatical descriptions of human languages, especially head-driven phrase structure grammar (HPSG) [51], and by development of languages for lexical knowledge representation, especially DATR [52]. The concepts and methods of these formalisms are often closely connected to those of logic programming. In the excellent thesis by Erjavec [53], the scientific context is discussed extensively and profoundly; refer also to the monographs by Carpenter [54] and Shieber [55].

In finite-state morphological models, both surface and lexical forms are by themselves unstructured strings of atomic symbols. In higher-level approaches, linguistic information is expressed by more appropriate data structures that can include complex values or can be recursively nested if needed. Morphological parsing $\mathcal{P}$ thus associates linear forms ϕ with alternatives of structured content ψ, cf. (1.1):

$$\mathcal{P} :: \phi \rightarrow \{\psi\} \qquad \mathcal{P} :: form \rightarrow \{content\} \tag{1.2}$$

Erjavec [53] argues that for morphological modeling, word forms are best captured by regular expressions, while the linguistic content is best described through **typed feature structures**. Feature structures can be viewed as directed acyclic graphs. A node in a feature structure comprises a set of attributes whose values can be feature structures again. Nodes are associated with types, and atomic values are attributeless nodes distinguished by their type. Instead of unique instances of values everywhere, references can be used to establish value instance identity. Feature structures are usually displayed as attribute-value matrices or as nested symbolic expressions.

Unification is the key operation by which feature structures can be merged into a more informative feature structure. Unification of feature structures can also fail, which means

that the information in them is mutually incompatible. Depending on the flavor of the processing logic, unification can be monotonic (i.e., information-preserving), or it can allow inheritance of default values and their overriding. In either case, information in a model can be efficiently shared and reused by means of inheritance hierarchies defined on the feature structure types.

Morphological models of this kind are typically formulated as logic programs, and unification is used to solve the system of constraints imposed by the model. Advantages of this approach include better abstraction possibilities for developing a morphological grammar as well as elimination of redundant information from it.

However, morphological models implemented in DATR can, under certain assumptions, be converted to finite-state machines and are thus formally equivalent to them in the range of morphological phenomena they can describe [11]. Interestingly, one-level phonology [56] formulating phonological constraints as logic expressions can be compiled into finite-state automata, which can then be intersected with morphological transducers to exclude any disturbing phonologically invalid surface strings [cf. 57, 53]

Unification-based models have been implemented for Russian [58], Czech [59], Slovene [53], Persian [60], Hebrew [61], Arabic [62, 63], and other languages. Some rely on DATR; some adopt, adapt, or develop other unification engines.

1.3.4 Functional Morphology

This group of morphological models includes not only the ones following the methodology of functional morphology [64], but even those related to it, such as morphological resource grammars of Grammatical Framework [65]. Functional morphology defines its models using principles of functional programming and type theory. It treats morphological operations and processes as pure mathematical functions and organizes the linguistic as well as abstract elements of a model into distinct types of values and type classes.

Though functional morphology is not limited to modeling particular types of morphologies in human languages, it is especially useful for fusional morphologies. Linguistic notions like paradigms, rules and exceptions, grammatical categories and parameters, lexemes, morphemes, and morphs can be represented intuitively and succinctly in this approach. Designing a morphological system in an accurate and elegant way is encouraged by the computational setting, which supports logical decoupling of subproblems and reinforces the semantic structure of a program by strong type checking.

Functional morphology implementations are intended to be reused as programming libraries capable of handling the complete morphology of a language and to be incorporated into various kinds of applications. Morphological parsing is just one usage of the system, the others being morphological generation, lexicon browsing, and so on. Next to parsing (1.2), we can describe inflection $\mathcal{I}$, derivation $\mathcal{D}$, and lookup $\mathcal{L}$ as functions of these generic types:

$$\mathcal{I} :: lexeme \rightarrow \{parameter\} \rightarrow \{form\} \tag{1.3}$$

$$\mathcal{D} :: lexeme \rightarrow \{parameter\} \rightarrow \{lexeme\} \tag{1.4}$$

$$\mathcal{L} :: content \rightarrow \{lexeme\} \tag{1.5}$$

A functional morphology model can be compiled into finite-state transducers if needed, but can also be used interactively in an interpreted mode, for instance. Computation within a model may exploit lazy evaluation and employ alternative methods of efficient parsing, lookup, and so on [see 66, 12].

Many functional morphology implementations are embedded in a general-purpose programming language, which gives programmers more freedom with advanced programming techniques and allows them to develop full-featured, real-world applications for their models. The Zen toolkit for Sanskrit morphology [67, 68] is written in OCaml. It influenced the functional morphology framework [64] in Haskell, with which morphologies of Latin, Swedish, Spanish, Urdu [69], and other languages have been implemented.

In Haskell, in particular, developers can take advantage of its syntactic flexibility and design their own notation for the functional constructs that model the given problem. The notation then constitutes a so-called domain-specific embedded language, which makes programming even more fun. Figure 1-2 illustrates how the ElixirFM implementation of Arabic morphology [12, 17] captures the structure of words and defines the lexicon. Despite the entries being most informative, their format is simply similar to that found in printed dictionaries. Operators like >|, |<, |<< and labels like verb are just infix functions; patterns and affixes like FaCY, FCI, At are data constructors.

```
|> "d r y" <| [
FaCY                          'verb'  [ "know", "notice" ]
    'imperf'  FCI,
FACY                          'verb'  [ "flatter", "deceive" ],
HaFCY                         'verb'  [ "inform", "let know" ],
IA >| "'a" >>| FCI |<< "iy"           'adj'   [ "agnostic" ],
FiCAL |< aT                   'noun'  [ "knowledge", "knowing" ],
MuFACY |< aT                  'noun'  [ "flattery" ]
    'plural'  MuFACY |< At,
FACI                          'adj'   [ "aware", "knowing" ] ]
```

```
d r y دري
fawā
fi
fāwā
ʔafā
lā-u-fi-iy
frāl-ah
mufāwā-ah
mufāwā-āt
fāʔ
```

know, notice	I (i)	darā دري	knowledge, knowing	dirāyah دراية
flatter, deceive	III	dārā داري	flattery	mudārāh مداراة
inform, let know	IV	ʔadrā أدرى		(mudāreyāt مداريات)
agnostic	lā-ʔadriy لاأدري		aware, knowing	dārin دار

Figure 1-2: Excerpt from the ElixirFM lexicon and a layout generated from it. The source code of entries nested under the *d r y* root is shown in monospace font. Note the custom notation and the economy yet informativeness of the declaration

Even without the options provided by general-purpose programming languages, functional morphology models achieve high levels of abstraction. Morphological grammars in Grammatical Framework [65] can be extended with descriptions of the syntax and semantics of a language. Grammatical Framework itself supports multilinguality, and models of more than a dozen languages are available in it as open-source software [70, 71].

Grammars in the OpenCCG project [72] can be viewed as functional models, too. Their formalism discerns declarations of features, categories, and families that provide type-system-like means for representing structured values and inheritance hierarchies on them. The grammars leverage heavily the functionality to define parametrized macros to minimize redundancy in the model and make required generalizations. Expansion of macros in the source code has effects similar to inlining of functions. The original text of the grammar is reduced to associations between word forms and their morphosyntactic and lexical properties.

1.3.5 Morphology Induction

We have focused on finding the structure of words in diverse languages supposing we know what we are looking for. We have not considered the problem of discovering and inducing word structure without the human insight (i.e., in an unsupervised or semi-supervised manner). The motivation for such approaches lies in the fact that for many languages, linguistic expertise might be unavailable or limited, and implementations adequate to a purpose may not exist at all. Automated acquisition of morphological and lexical information, even if not perfect, can be reused for bootstrapping and improving the classical morphological models, too.

Let us skim over the directions of research in this domain. In the studies by Hammarström [73] and Goldsmith [74], the literature on unsupervised learning of morphology is reviewed in detail. Hammarström divides the numerous approaches into three main groups. Some works compare and cluster words based on their similarity according to miscellaneous metrics [75, 76, 77, 78]; others try to identify the prominent features of word forms distinguishing them from the unrelated ones. Most of the published approaches cast morphology induction as the problem of word boundary and morpheme boundary detection, sometimes acquiring also lexicons and paradigms [79, 80, 81, 82, 83].[7]

There are several challenging issues about deducing word structure just from the forms and their context. They are caused by ambiguity [76] and irregularity [75] in morphology, as well as by orthographic and phonological alternations [85] and nonlinear morphological processes [86, 87].

In order to improve the chances of statistical inference, parallel learning of morphologies for multiple languages is proposed by Snyder and Barzilay [88], resulting in discovery of abstract morphemes. The discriminative log-linear model of Poon, Cherry, and Toutanova [89] enhances its generalization options by employing overlapping contextual features when making segmentation decisions [cf. 90].

7. Compare these with a semisupervised approach to word hyphenation [84].

1.4 Summary

In this chapter, we learned that morphology can be looked at from opposing viewpoints: one that tries to find the structural components from which words are built versus a more syntax-driven perspective wherein the functions of words are the focus of the study. Another distinction can be made between analytic and generative aspects of morphology or can consider man-made morphological frameworks versus systems for unsupervised induction of morphology. Yet other kinds of issues are raised about how well and how easily the morphological models can be implemented.

We described morphological parsing as the formal process recovering structured information from a linear sequence of symbols, where ambiguity is present and where multiple interpretations should be expected.

We explored interesting morphological phenomena in different types of languages and mentioned several hints in respect to multilingual processing and model development.

With Korean as a language where agglutination moderated by phonological rules is the dominant morphological process, we saw that a viable model of word decomposition can work at the morphemes level, regardless of whether they are lexical or grammatical.

In Czech and Arabic as fusional languages with intricate systems of inflectional and derivational parameters and lexically dependent word stem variation, such factorization is not useful. Morphology is better described via paradigms associating the possible forms of lexemes with their corresponding properties.

We discussed various options for implementing either of these models using modern programming techniques.

Acknowledgment

We would like to thank Petr Novák for his valuable comments on an earlier draft of this chapter.

Bibliography

[1] M. Liberman, "Morphology." Linguistics 001, Lecture 7, University of Pennsylvania, 2009. http://www.ling.upenn.edu/courses/Fall_2009/ling001/morphology.html.

[2] M. Haspelmath, "The indeterminacy of word segmentation and the nature of morphology and syntax," *Folia Linguistica*, vol. 45, 2011.

[3] H. Kučera and W. N. Francis, *Computational Analysis of Present-Day American English*. Providence, RI: Brown University Press, 1967.

[4] S. B. Cohen and N. A. Smith, "Joint morphological and syntactic disambiguation," in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pp. 208–217, 2007.

[5] T. Nakagawa, "Chinese and Japanese word segmentation using word-level and character-level information," in *Proceedings of 20th International Conference on Computational Linguistics*, pp. 466–472, 2004.

[6] H. Shin and H. You, "Hybrid n-gram probability estimation in morphologically rich languages," in *Proceedings of the 23rd Pacific Asia Conference on Language, Information and Computation*, 2009.

[7] D. Z. Hakkani-Tür, K. Oflazer, and G. Tür, "Statistical morphological disambiguation for agglutinative languages," in *Proceedings of the 18th Conference on Computational Linguistics*, pp. 285–291, 2000.

[8] G. T. Stump, *Inflectional Morphology: A Theory of Paradigm Structure*. Cambridge Studies in Linguistics, New York: Cambridge University Press, 2001.

[9] K. R. Beesley and L. Karttunen, *Finite State Morphology*. CSLI Studies in Computational Linguistics, Stanford, CA: CSLI Publications, 2003.

[10] M. Baerman, D. Brown, and G. G. Corbett, *The Syntax-Morphology Interface. A Study of Syncretism*. Cambridge Studies in Linguistics, New York: Cambridge University Press, 2006.

[11] B. Roark and R. Sproat, *Computational Approaches to Morphology and Syntax*. Oxford Surveys in Syntax and Morphology, New York: Oxford University Press, 2007.

[12] O. Smrž, "Functional Arabic morphology. Formal system and implementation." PhD thesis, Charles University in Prague, 2007.

[13] H. Eifring and R. Theil, *Linguistics for Students of Asian and African Languages*. Universitetet i Oslo, 2005.

[14] B. Bickel and J. Nichols, "Fusion of selected inflectional formatives & exponence of selected inflectional formatives," in *The World Atlas of Language Structures Online* (M. Haspelmath, M. S. Dryer, D. Gil, and B. Comrie, eds.), ch. 20 & 21, Munich: Max Planck Digital Library, 2008.

[15] W. Fischer, *A Grammar of Classical Arabic*. Trans. Jonathan Rodgers. Yale Language Series, New Haven, CT: Yale University Press, 2002.

[16] K. C. Ryding, *A Reference Grammar of Modern Standard Arabic*. New York: Cambridge University Press, 2005.

[17] O. Smrž and V. Bielický, "ElixirFM." Functional Arabic Morphology, SourceForge.net, 2010. http://sourceforge.net/projects/elixir-fm/.

[18] T. Kamei, R. Kōno, and E. Chino, eds., *The Sanseido Encyclopedia of Linguistics. Volume 6 Terms* (in Japanese). Sanseido, 1996.

[19] F. Karlsson, *Finnish Grammar*. Helsinki: Werner Söderström Osakeyhtiö, 1987.

[20] J. Hajič and B. Hladká, "Tagging inflective languages: Prediction of morphological categories for a rich, structured tagset," in *Proceedings of COLING-ACL 1998*, pp. 483–490, 1998.

[80] H. Johnson and J. Martin, "Unsupervised learning of morphology for English and Inuktitut," in *Companion Volume of the Proceedings of the Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics 2003: Short Papers*, pp. 43–45, 2003.

[81] M. Creutz and K. Lagus, "Induction of a simple morphology for highly-inflecting languages," in *Proceedings of the 7th Meeting of the ACL Special Interest Group in Computational Phonology*, pp. 43–51, 2004.

[82] M. Creutz and K. Lagus, "Unsupervised models for morpheme segmentation and morphology learning," *ACM Transactions on Speech and Language Processing*, vol. 4, no. 1, pp. 1–34, 2007.

[83] C. Monson, J. Carbonell, A. Lavie, and L. Levin, "ParaMor: Minimally supervised induction of paradigm structure and morphological analysis," in *Proceedings of Ninth Meeting of the ACL Special Interest Group in Computational Morphology and Phonology*, pp. 117–125, 2007.

[84] F. M. Liang, "Word Hy-phen-a-tion by Com-put-er," PhD thesis, Stanford University, 1983.

[85] V. Demberg, "A language-independent unsupervised model for morphological segmentation," in *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pp. 920–927, 2007.

[86] A. Clark, "Supervised and unsupervised learning of Arabic morphology," in *Arabic Computational Morphology. Knowledge-based and Empirical Methods* (A. Soudi, A. van den Bosch, and G. Neumann, eds.), vol. 38, pp. 181–200, Berlin: Springer, 2007.

[87] A. Xanthos, *Apprentissage automatique de la morphologie: le cas des structures racine-schème*. Sciences pour la communication, Bern: Peter Lang, 2008.

[88] B. Snyder and R. Barzilay, "Unsupervised multilingual learning for morphological segmentation," in *Proceedings of ACL-08: HLT*, pp. 737–745, 2008.

[89] H. Poon, C. Cherry, and K. Toutanova, "Unsupervised morphological segmentation with log-linear models," in *Proceedings of Human Language Technologies: Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 209–217, 2009.

[90] S. Della Pietra, V. Della Pietra, and J. Lafferty, "Inducing features of random fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 4, pp. 380–393, 1997.

Chapter 2

Finding the Structure of Documents

Dilek Hakkani-Tür, Gokhan Tur, Benoit Favre, and Elizabeth Shriberg

2.1 Introduction

In human language, words and sentences do not appear randomly but usually have a structure. For example, combinations of words form sentences—meaningful grammatical units, such as statements, requests, and commands. Likewise, in written text, sentences form paragraphs—self-contained units of discourse about a particular point or idea. Sentences may also be related to each other by explicit discourse connectives such as *therefore*.

Automatic extraction of structure of documents helps subsequent natural language processing (NLP) tasks; for example, parsing, machine translation, and semantic role labeling use sentences as the basic processing unit [1, 2]. Sentence boundary annotation is also important for aiding human readability of the output of automatic speech recognition (ASR) systems [3]. Furthermore, chunking the input text or speech into topically coherent blocks provides better organization and indexing of the data. For example, portions related to specific topics can be extracted from a long speech. Similarly, articles belonging to the same topic may be categorized and processed further. Given the ever-growing problem of written and spoken information overload, extracting the structure of textual and audio documents is a meaningful and sometimes necessary first step in most speech and language processing applications.

Here, we discuss methods for finding the structure of documents; for simplicity, only the sentence and group of sentences related to a topic are considered as the structure elements.

In this chapter, we call the task of deciding where sentences start and end given a sequence of characters (made of words and typographical cues) **sentence boundary detection**. Similarly, we refer to **topic segmentation** as the task of determining when a topic starts and ends in a sequence of sentences. We present statistical classification approaches that try to infer the presence of sentence and topic boundaries given human-annotated training data, for segmentation.[1] These methods base their predictions on features of the input: local characteristics that give evidence toward the presence or absence of a sentence or topic boundary, such as a punctuation sign, a pause in speech, and a new word in a document. Features are the core of classification approaches and require careful design and selection in order to be successful and prevent overfitting and noise problems.

1. *Segmentation* refers to both tasks.

Note that while most statistical approaches described in this chapter are language independent, every language is a challenge in itself. For example, for processing of Chinese documents, the processor may need to first segment character sequences into words, as the words usually are not separated by a space. Similarly, for morphologically rich languages, the word structure may need to be analyzed to extract additional features. Such processing is usually done in a preprocessing step, where a sequence of tokens is determined. Tokens can be words or subword units, depending on the task and language. These algorithms are then applied on tokens. Segmentation aims to decide whether or not a boundary between two tokens should be marked as a sentence (or a topic) boundary.

Instead of focusing on techniques used for sentence and topic segmentation individually, we first formally define these tasks and present techniques for sentence and topic segmentation in a unified framework. Then, we present the features used for segmenting text or speech.

2.1.1 Sentence Boundary Detection

Sentence boundary detection (also called sentence segmentation) deals with automatically segmenting a sequence of word tokens into sentence units. In written text in English and some other languages, the beginning of a sentence is usually marked with an uppercase letter, and the end of a sentence is explicitly marked with a period (.), a question mark (?), an exclamation mark (!), or another type of punctuation. However, in addition to their role as sentence boundary markers, capitalized initial letters are used to distinguish proper nouns, periods are used in abbreviations, and numbers and other punctuation marks are used inside proper names. For instance, 10% of the periods in the Brown corpus are abbreviations [4] such as "Dr." that can be an abbreviation for the words *doctor* and *drive*. And the period at the end of an abbreviation can mark a sentence boundary at the same time. For example, consider the following two sentences: *I spoke with Dr. Smith.* and *My house is on Mountain Dr.* In the first sentence, the abbreviation *Dr.* does not end a sentence, and in the second it does. This percentage of periods that are used to mark an abbreviation rises to 47% in the Wall Street Journal Corpus [5]. For example, in the following sentence, partly taken from the Wall Street Journal part of the OntoNotes [6] corpus, only the last period ends the sentence:

> *"This year has been difficult for both Hertz and Avis," said Charles Finnie, car-rental industry analyst—yes, there is such a profession—at Alex. Brown & Sons.*

Such sentences containing other sentences are not infrequent. Especially quoted sentences are always problematic, as the speaker may have uttered multiple sentences, and sentence boundaries inside the quotes are also marked with punctuation marks. An automatic method that outputs word boundaries as ending sentences according to the presence of such punctuation marks would result in cutting some sentences incorrectly. Furthermore, if the preceeding sentence is spoken instead of written, prosodic cues usually mark structure.

Ambiguous abbreviations and capitalizations are not the only problem of sentence segmentation in written text. "Spontaneously" written texts, such as short message service (SMS) texts or instant messaging (IM) texts, tend to be nongrammatical and have poorly used or missing punctuation, which makes sentence segmentation even more challenging [7, 8].

Similarly, if the text input to be segmented into sentences comes from an automatic system, such as optical character recognition (OCR) or ASR, that aims to translate images of handwritten, typewritten, or printed text or spoken utterances into machine-editable text, the finding of sentence boundaries must deal with the errors of those systems as well. For example, Taghva et al. [9] observed that an OCR system easily confuses periods and commas and can result in meaningless sentences. ASR transcripts typically lack punctuation marks and are usually monocase; hence, all ASR output word boundaries can be ending or beginning a sentence. Stevenson and Gaizauskas [10] asked human participants to repunctuate monocase texts, and they performed at an F_1-measure of about 80%, which illustrates the difficulty of the task. In such input, sentence segmentation methods usually hypothesize a sentence boundary between every two tokens.

On the other hand, for conversational speech or text or multiparty meetings with ungrammatical sentences and disfluencies, in most cases it is not clear where the boundaries are. The interannotator agreement was quite low [11] during the segmentation of the Linguistic Data Consortium (LDC) distributed ICSI Meeting Corpus [12]. In an example utterance of *okay no problem*, it is not clear whether there is a single sentence or two. The problem may be redefined for the conversational domain as the task of dialog act segmentation, because dialog acts are better defined for conversational speech using a number of markup standards such as Dialog Act Markup in Several Layers (DAMSL) [13] or Meeting Recorder Dialog Act (MRDA) [14]. According to these standards, the example sentence *okay no problem* consists of two sentential units (or dialog act units): *okay* and *no problem*.

In most practical applications relying on automatic sentence segmentation, the task can be redefined according to the need of the following task. For example, the sentence *I think so but you should also ask him* may be a grammatical sentence as a whole, but for DAMSL and MRDA standards, there are two dialog act tags, one affirmation and one suggestion. Such a modification may be needed for conversation analysis, such as speaker role detection or sentiment analysis. This task can be seen as a semantic boundary detection task instead of syntactic.

Code switching—that is, the use of words, phrases, or sentences from multiple languages by multilingual speakers—is another problem that can affect the characteristics of sentences. For example, when switching to a different language, the writer can either keep the punctuation rules from the first language or resort to the code of the second language (e.g., Spanish uses the inverted question mark to precede questions). Code switching also affects technical texts for which the meanings of punctuation signs can be redefined, as in uniform resource locators (URLs), programming languages, and mathematics. We must detect and parse those specific constructs in order to process technical texts adequately.

Conventional rule-based sentence segmentation systems in well-formed texts rely on patterns to identify potential ends of sentences and lists of abbreviations for disambiguating them [5, 15, 16, 17]. For example if the word before the boundary is a known abbreviation, such as "Mr." or "Gov.," the text is not segmented at that position even though some periods are exceptions. Although rules cover most of these cases, they do not address unknown abbreviations, abbreviations at the ends of sentences, or typos in the input text. Furthermore, such rules are not robust to text that is not well formed, such as forums, chats, and blogs, or to spoken input that completely lacks typographic cues. Moreover, each language requires a specific set of rules.

To improve on such a rule-based approach, sentence segmentation is stated as a classification problem. Given training data where all sentence boundaries are marked, we can train a classifier to recognize them, as described in Section 2.2. Sentence segmentation in text usually uses the punctuation marks as delimiters and aims to categorize them as sentence-ending/beginning or not. On the other hand, for speech input, all word boundaries are usually considered as candidate sentence boundaries.

2.1.2 Topic Boundary Detection

Topic segmentation (sometimes called discourse or text segmentation) is the task of automatically dividing a stream of text or speech into topically homogeneous blocks. That is, given a sequence of (written or spoken) words, the aim of topic segmentation is to find the boundaries where topics change. Figure 2-1 gives an example of a topic change boundary from a broadcast news program.

Topic segmentation is an important task for various language-understanding applications, such as information extraction and retrieval and text summarization. For example, in information retrieval, if long documents can be segmented into shorter, topically coherent segments, then only the segment that is about the user's query could be retrieved.

During the late 1990s, the U.S. Defense Advanced Research Projects Agency (DARPA) initiated the Topic Detection and Tracking (TDT) program to further the state of the art in finding and following new topics in a stream of broadcast news stories [18]. One of the tasks in the TDT effort was segmenting a news stream into individual stories. TDT established a common test bed, but most researchers also use simulated environments such as by concatenating news stories from Reuters.

For multiparty meetings, the task of topic segmentation is inspired by discourse analysis. For official and well-structured meetings, the topics are segmented according to the agenda items, whereas for more casual conversational-style meetings, the boundaries are less clear.

Topic segmentation is a nontrivial problem without a very high human agreement because of many natural-language-related issues and hence requires a good definition of topic categories and their granularities. For example, topics are not typically flat but occur in a semantic hierarchy. When a sentence about soccer is followed by a sentence about baseball, one annotator may mark a topic change and the other may not, considering that soccer and baseball both belong to the topic sports. This is also the case for finer-grained distinctions. Even though the annotators are told to segment the text into a predefined number

> Tens of thousands of people are homeless in northern China tonight after a powerful earthquake hit an earthquake registering 6.2 on the Richter scale at least 47 people are dead. Few pictures available from the region but we do know temperatures there will be very cold tonight -7 degrees. <TOPIC_CHANGE> Peace talks expected to resume on Monday in Belfast, Northern Ireland. ...

Figure 2-1: Example of a topic boundary in a news article

of topics, it is hard to define the concept of topic because it varies greatly depending on the semantic content. While high interannotator agreement (with Cohen's kappa values of 0.7–0.9) has been achieved for the TDT corpus [19], which includes broadcast news, documents and hence stories, news, or topics usually had the same boundary. For topic segmentation of multiparty meetings, the agreement is lower [20] (with kappa values of 0.6–0.7). Note that for conversational speech, the topic boundaries may not be absolute. For example, in a multiparty meeting, a few turns after switching the topic, a participant may utter a sentence about the previous topic.

In text, topic boundaries are usually marked with distinct segmentation cues, such as headlines and paragraph breaks. These cues are absent in speech. However, speech provides other cues, such as pause duration and speaker changes. This is analogous to differences between sentence segmentation of text and speech. In Section 2.5 these feature types are analyzed in more detail.

2.2 Methods

Sentence segmentation and topic segmentation have mainly been considered as a boundary classification problem. Given a boundary candidate (between two word tokens for sentence segmentation and between two sentences for topic segmentation), the goal is to predict whether or not the candidate is an actual boundary (sentence or topic boundary). Formally, let $\mathbf{x} \in \mathcal{X}$ be the vector of features (the observation) associated with a candidate and $y \in \mathcal{Y}$ be the label predicted for that candidate. The label y can be b for boundary and $\bar{b}$ for nonboundary. This results in a classification problem: given a set of training examples $(\mathbf{x}, y)_{\text{train}}$, find a function that will assign the most accurate possible label y of unseen examples $\mathbf{x}_{\text{unseen}}$. Alternatively to the binary classification problem, it is possible to model boundary types using finer-grained categories. For example, Gillick [21] suggests that sentence segmentation in text be framed as a three-class problem: sentence boundary with an abbreviation b^a, without an abbreviation $b^{\bar{a}}$, and abbreviation not at a boundary $\bar{b}^a$. Similarly, in spoken language, a three-way classification can be made between nonboundaries $\bar{b}$, statement b^s, and question boundaries b^q.

Features can be the presence of specific word n-grams around the candidate boundary; an indicator of being inside a quotation in text; an indicator of presence of the preceding word tokens in an abbreviation list; or duration of pause, pitch, energy, and other duration-related features in speech. A more detailed discussion of features is presented in Section 2.5.

For sentence or topic segmentation, the problem is defined as finding the most probable sentence or topic boundaries. The natural unit of sentence segmentation is words and of topic segmentation is sentences, as we can assume that topics typically do not change in the middle of a sentence.[2] The words or sentences are then grouped into contiguous stretches belonging to one sentence or topic—that is, the word or sentence boundaries are classified

2. Similarly, it is sometimes assumed for topic-segmentation purposes that topics change only at paragraph boundaries [22].

into sentence or topic boundaries and nonboundaries. The classification can be done at each potential boundary i (local modeling); then, the aim is to estimate the most probable boundary type, $\widehat{y}_i$, for each candidate example, $\mathbf{x}_i$:

$$\widehat{y}_i = \underset{y_i \in \mathcal{Y}}{\operatorname{argmax}} P(y_i|\mathbf{x}_i) \tag{2.1}$$

Here, the $\widehat{\ }$ is used to denote estimated categories, and a variable without a $\widehat{\ }$ is used to show possible categories. In this formulation, a category is assigned to each example in isolation; hence, the decision is made locally. However, the consecutive boundary types can be related to each other. For example, in broadcast news speech, two consecutive sentence boundaries that form a single word sentence are very infrequent. In local modeling, features can be extracted from the surrounding example context of the candidate boundary to model such dependencies. It is also possible to see the candidate boundaries as a sequence and search for the sequence of boundary types, $\widehat{Y} = \widehat{y}_1, \ldots, \widehat{y}_n$, that have the maximum probability given the candidate examples, $X = \mathbf{x}_1, \ldots, \mathbf{x}_n$:

$$\widehat{Y} = \underset{Y}{\operatorname{argmax}} P(Y|X) \tag{2.2}$$

In the following discussion, we categorize the methods into local and sequence classification. Another categorization of methods is done according to the type of the machine learning algorithm: generative versus discriminative. Generative sequence models estimate the joint distribution of the observations, $P(X,Y)$ (e.g., words, punctuation) and the labels (sentence boundary, topic boundary), which requires specific assumptions (such as backoff to account for unseen events) and have good generalization properties. Discriminative sequence models, however, focus on features that characterize the differences between the labeling of the examples.

Such methods (as described in the following sections) can be used for sentence and topic segmentation in both written and spoken language, with one difference: in text, the category of all boundaries that do not include a potential end-of-sentence delimiter (period, question mark, exclamation mark) is preset to nonsentence or nontopic, and a category is estimated for only those word boundaries that include a delimiter, whereas in speech, all boundaries between consecutive tokens are usually considered.

2.2.1 Generative Sequence Classification Methods

The most commonly used generative sequence classification method for topic and sentence segmentation is the hidden Markov model (HMM). The probability in Equation 2.2 is rewritten as the following, using the Bayes rule:

$$\widehat{Y} = \underset{Y}{\operatorname{argmax}} P(Y|X) = \underset{Y}{\operatorname{argmax}} \frac{P(X|Y)P(Y)}{P(X)} = \underset{Y}{\operatorname{argmax}} P(X|Y)P(Y) \tag{2.3}$$

$P(X)$ in the denominator is dropped because it is fixed for different Y and hence does not change the argument of max. $P(X|Y)$, and $P(Y)$ can be estimated as

$$P(X|Y) = \prod_{i=1}^{n} P(\mathbf{x}_i|y_1, \ldots, y_i) \tag{2.4}$$

and

$$P(Y) = \prod_{i=1}^{n} P(y_i|y_1, \ldots, y_{i-1}) \tag{2.5}$$

Simplifying assumptions can be made to make the computation of these probabilities tractable:

$$P(\mathbf{x}_i|y_1, \ldots, y_i) \approx P(\mathbf{x}_i|y_i) \tag{2.6}$$

and a bigram model can be assumed for modeling output categories:

$$P(y_i|y_1, \ldots, y_{i-1}) \approx P(y_i|y_{i-1}) \tag{2.7}$$

The bigram case is modeled by a fully connected m-state Markov model, where m is the number of boundary categories. The states emit words (sentences or paragraphs) for sentence (topic) segmentation, and the state sequence that most likely generated the word (sentence) sequence is estimated. State transition probabilities, $P(y_i|y_{i-1})$, and state observation likelihoods, $P(\mathbf{x}_i|y_i)$, are estimated using the training data. The most probable boundary sequence is obtained by dynamic programming, thanks to the Viterbi algorithm that is used for decoding Markov models [23]. The bigram case can be extended to higher-order n-grams at the cost of an increased complexity.

For example, Figure 2-2 shows the model for the two-class problem, for example *nonboundary* (NB) and *sentence boundary* (SB) for sentence segmentation. Table 2-1 shows an example sequence of words emitted.

For topic segmentation, typically instead of using two states, n states are used, where n is the number of topics. However, obtaining state observation likelihoods without knowing the topic categories is the main challenge. Yamron et al. [24] model topics with unigram language models, and the state observation likelihoods are trained using the k-means clustering algorithm.

Note that this is not different from using an HMM, as is typically done in similar tagging tasks, such as part-of-speech (POS) tagging [25] or named entity extraction [26]. However, it

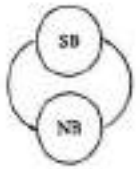


Figure 2–2: Conceptual hidden Markov model for segmentation with two states: one for segment boundaries, one for others

Table 2–1: Sentence segmentation with simple two-state Markov model

Emitted Words	...	people	are	dead	few	pictures	...
State Sequence	...	NB	NB	SB	NB	NB	...

has been shown that the conventional HMM approach has certain weaknesses. For example, it is not possible to use any information beyond words, such as POS tags of the words or prosodic cues, for speech segmentation.

To this end, two simple extensions have been proposed: Shriberg et al. [27] suggested using explicit states to emit the boundary tokens, hence incorporating nonlexical information via combination with other models. This approach is used for sentence segmentation and is inspired by the hidden event language model (HELM), as introduced by Stolcke and Shriberg [28], which was originally designed for speech disfluencies. The approach was to treat such events as extra meta tokens. In this model, one state is reserved for each boundary token, *SB* and *NB*, and the rest of the states are for generating words. To ease the computation, an imaginary token is inserted between all consecutive words in case the word preceding the boundary is not part of a disfluency. Example 2–1 is a conceptual representation of a sequence with boundary tokens:

EXAMPLE 2–1: ... *people NB are NB dead YB few NB pictures* ...

The most probable boundary token sequence is again obtained simply by Viterbi decoding. The conceptual HELM for segmentation is depicted in Figure 2–3.

These extra boundary tokens are then used to capture other meta-information. The most commonly used meta-information is the feedback obtained from other classifiers. Typically, the posterior probability of being in that boundary state is used as a state observation likelihood after being divided by prior probabilities [27]. These other classifiers also may be trained with other feature sets, such as prosodic or syntactic. This hybrid approach is presented in Section 2.2.4.

For topic segmentation, Tur et al. [29] used the same idea and modeled topic-start and topic-final sections explicitly, which helped greatly for broadcast news topic segmentation.

The second extension is inspired from factored language models [30], which capture not only words but also morphological, syntactic, and other information. Guz et al. [31] proposed using factored HELM (fHELM) for sentence segmentation using POS tags in addition to words.

2.2.2 Discriminative Local Classification Methods

Discriminative classifiers aim to model $P(y_i|\mathbf{x}_i)$ of Equation 2.1 directly. The most important distinction is that whereas class densities, $p(\mathbf{x}|y)$, are model assumptions in generative approaches, such as naive Bayes, in discriminative methods, discriminant functions of the feature space define the model. A number of discriminative classification approaches, such as support vector machines, boosting, maximum entropy, and regression, are based on very

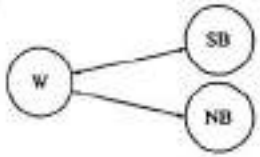


Figure 2–3: Conceptual hidden event language model for segmentation

different machine learning algorithms. While discriminative approaches have been shown to outperform generative methods in many speech and language processing tasks, training typically requires iterative optimization.

In discriminative local classification, each boundary is processed separately with local and contextual features. No global (i.e., sentence or document wide) optimization is performed, unlike in sequence classification models. Instead, features related to a wider context may be incorporated into the feature set. For example, the predicted class of the previous or next boundary can be used in an iterative fashion.

For sentence segmentation, supervised learning methods have primarily been applied to newspaper articles. Stamatatos, Fakotakis, and Kokkinakis [32] used transformation-based learning (TBL) to infer rules for finding sentence boundaries. Many classifiers have been tried for the task: regression trees [33], neural networks [34, 35], a C4.5 classification tree [36], maximum entropy classifiers [37, 38], support vector machines (SVMs), and naive Bayes classifiers [21]. Mikheev treated the sentence segmentation problem as a subtask for POS tagging by assigning a tag to punctuation similar to other tokens [39]. For tagging he employed a combination of HMM and maximum entropy approaches.

The popular TextTiling method of Hearst for topic segmentation [40, 22] uses a lexical cohesion metric in a word vector space as an indicator of topic similarity. TextTiling can be seen as a local classification method with a single feature of similarity. Figure 2–4 depicts a typical graph of similarity with respect to consecutive segmentation units. The document is chopped when the similarity is below some threshold.

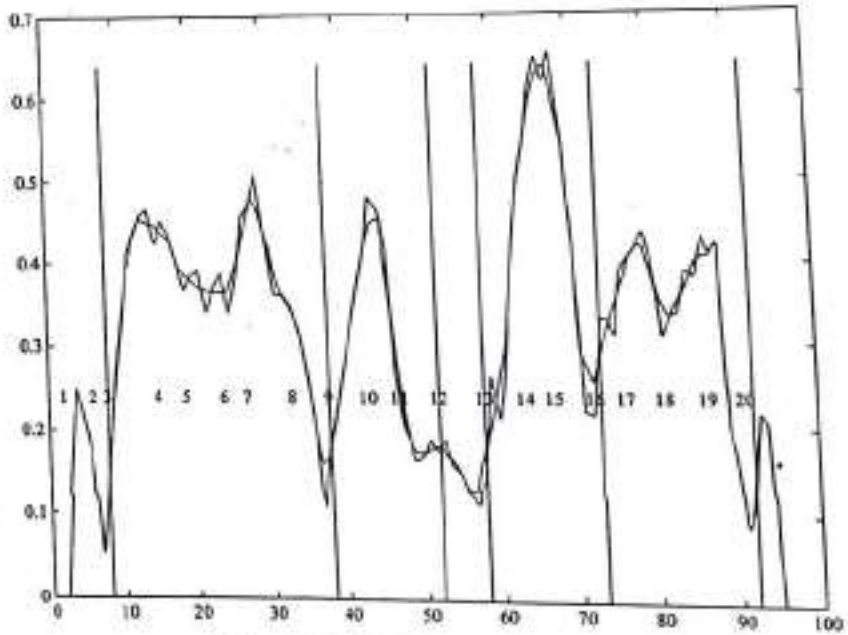


Figure 2–4: Text Tiling example (from [22])

Originally, two methods for computing the similarity scores were proposed: block comparison and vocabulary introduction. The first, block comparison, compares adjacent blocks of text to see how similar they are according to how many words the adjacent blocks have in common. The block size can be variable, not necessarily looking only at the consecutive blocks but instead at a window. Given two blocks, b_1 and b_2, each having k tokens (sentences or paragraphs), the similarity (or topical cohesion) score is computed by the formula:

$$\frac{\sum_t w_{t,b_1} . w_{t,b_2}}{\sqrt{\sum_t w_{t,b_1}^2 . \sum_t w_{t,b_2}^2}}$$

where $w_{t,b}$ is the weight assigned to term t in block b. The weights can be binary or may be computed using other information retrieval–based metrics such as term frequency.

The second, the vocabulary introduction method, assigns a score to a token-sequence gap on the basis of how many new words are seen in the interval in which it is the midpoint. Similar to the block comparison formulation, given two consecutive blocks, b_1 and b_2, of equal number of words, w, the topical cohesion score is computed with the following formula:

$$\frac{NumNewTerms(b_1) + NumNewTerms(b_2)}{2 \times w}$$

where $NumNewTerms(b)$ returns the number of terms in block b, seen for the first time in text.

Brants, Chen, and Tsochantaridis [41] extended this method to exploit latent semantic analysis. Instead of simply looking at all words, they worked on the transformed lexical space, which has led to improved results because this approach also captures semantic similarities implicitly.

Morris and Hirst [42] proposed using lexical chains instead of lexical similarity for estimating cohesion. Later, Kan, Klavans, and McKeown [43] proposed using a simpler interpretation of lexical chains, linking nonfunction words and syntactic phrases to each other only if they occur within n sentences, where n and the weights of the links are tuned on the basis of the syntactic category.

Banerjee and Rudnicky [44] applied the original TextTiling approach to the meetings domain. Galley et al. [45] used a similar approach with chains of repeated terms for meeting segmentation. Hsueh and Moore [20] extended this approach by using decision trees. Purver et al. [46] used a generative topic model with a variant of Latent Dirichlet allocation to learn models of the topics in an unsupervised fashion, simultaneously producing a segmentation of the meetings.

Reynar [47] and Beeferman, Berger, and Lafferty [48] extended TextTiling-based methods using maximum entropy models with a wide range of lexical and discourse features tracking vocabulary shift. Georgescul, Clark, and Armstrong [49] employed SVMs for this task. Rosenberg and Hirschberg [50] employed the Ripper algorithm with lexical chains, cue words, and prosodic features. Levow [51] used decision trees with cosine similarity and prosodic features for broadcast news segmentation.

2.2.3 Discriminative Sequence Classification Methods

In segmentation tasks, the sentence or topic decision for a given example (word, sentence, paragraph) highly depends on the decision for the examples in its vicinity. Discriminative

sequence classification methods are in general extensions of local discriminative models with additional decoding stages that find the best assignment of labels by looking at neighboring decisions to label an example. Conditional random fields (CRFs) [52] are an extension of maximum entropy, SVM struct [53] is an extension of SVM to handle structured outputs, and maximum margin Markov networks (M^3N) are extensions of HMMs [54]. The margin infused relaxed algorithm (MIRA) is an online learning approach that requires loading of one sequence at a time during training [55]. For conciseness, we present only CRFs, which have been successful for many sequence labeling tasks, including sentence segmentation in speech.

CRFs are a class of log-linear models for labeling structures [52]. Contrary to local classifiers that predict sentence or topic boundaries independently, CRFs can oversee the whole sequence of boundary hypotheses to make their decisions. Formally, they model the conditional probability of a sequence of boundary labels ($Y = y_1, \ldots, y_n$) given the sequence of feature sets extracted from the context in which they occur ($X = x_1, \ldots, x_n$).

$$P(Y|X) \sim \frac{1}{Z(X)} \exp\left(\sum_{t=1}^{n}\sum_{i=1}^{m} \lambda_i f_i(y_{t-1}, y_t, y_t)\right) \tag{2.8}$$

$$Z(X) = \sum_{Y} \exp\left(\sum_{t=1}^{n}\sum_{i=1}^{m} \lambda_i f_i(y_{t-1}, y_t, y_t)\right)$$

where $f_i(\cdot)$ are feature functions of the observations and a clique of labels, and λ_i are the corresponding weights. $Z(\cdot)$ is a normalization function dependent only on the observations. CRFs are trained by finding the λ parameters that maximize the likelihood of the training data, usually with a regularization term to avoid overfitting. Gradient, conjugate gradient, or online methods are used for training [56, 57, 58]. Dynamic programming (Viterbi decoding) is used to find the most probable assignment of labels at test time or to compute the $Z(\cdot)$ function.

2.2.4 Hybrid Approaches

Nonsequential discriminative classification algorithms typically ignore the context, which is critical for the segmentation task. While we may add context as a feature or simply use CRFs, which inherently consider context, these approaches are suboptimal when dealing with real-valued features, such as pause duration or pitch range. Most earlier studies simply tackled this problem by binning the feature space either manually or automatically [59].

An alternative is to use a hybrid classification approach, as suggested by Shriberg et al. [27]. The main idea is to use the posterior probabilities, $P_c(y_i|\mathbf{x}_i)$, for each boundary candidate, obtained from the other classifiers, such as boosting or CRF, by simply converting them to state observation likelihoods by dividing to their priors following the well-known Bayes rule:

$$\underset{y_i}{\operatorname{argmax}} \frac{P_c(y_i|\mathbf{x}_i)}{P(y_i)} = \underset{y_i}{\operatorname{argmax}}\, P(\mathbf{x}_i|y_i) \tag{2.9}$$

Applying the Viterbi algorithm to the HMM then returns the most likely segmentation. To handle dynamic ranges of state transition probabilities and observation likelihoods, a

weighting scheme as is usually described in the literature can be applied:

$$\underset{y_i}{\operatorname{argmax}}\, P_c(\mathbf{x}_i|y_i)^\alpha \times P(y_i)^\beta \tag{2.10}$$

where $P(y_i)$ is estimated by the HELM, and α and β are optimized using a held-out set.

Zimmerman et al. compared various discriminative local classification methods, namely boosting, maximum entropy, and decision trees, along with their hybrid versions for sentence segmentation of multilingual speech [60]. They concluded that hybrid approaches are always superior, and Guz et al. [31] concluded that this is also true with CRF, although to a lesser degree.

2.2.5 Extensions for Global Modeling for Sentence Segmentation

So far, most approaches to sentence segmentation have focused on recognizing boundaries rather than sentences in themselves. This has occurred because of the quadratic number of sentence hypotheses that must be assessed in comparison to the number of boundaries. To tackle that problem, Roark et al. [61] segment the input according to likely sentence boundaries established by a local model, and then train a reranker on the n-best lists of segmentations. This approach allows leveraging of sentence-level features such as scores from a syntactic parser or global prosodic features. Favre et al. [62] proposed to extend this concept to a pruned sentence lattice, which allows combining local scores with sentence-level scores in a more efficient manner.

2.3 Complexity of the Approaches

The approaches described here have advantages and disadvantages. In a given context and under a set of observation features, one approach may be better than another. These approaches can be rated in terms of complexity (time and memory) of their training and prediction algorithms and in terms of their performance on real-world datasets. Some may also require specific preprocessing, such as converting or normalizing continuous features to discrete features.

In terms of complexity, training of discriminative approaches is more complex than training of generative ones because they require multiple passes over the training data to adjust for their feature weights. However, generative models such as HELMs can handle multiple orders of magnitude larger training sets and benefit, for instance, from decades of news wire transcripts. On the other hand, they work with only a few features (only words for HELM) and do not cope well with unseen events. Discriminative classifiers allow for a wider variety of features and perform better on smaller training sets. Predicting with discriminative classifiers is also slower, even though the models are relatively simple (linear or log-linear), because it is dominated by the cost of extracting more features.

Compared to local approaches, sequence approaches bring the additional complexity of decoding: finding the best sequence of decisions requires evaluating all possible sequences of decisions. Fortunately, conditional independence assumptions allow the use of dynamic programming to trade time for memory and decode in polynomial time. This complexity

is then exponential in the order of the model (number of boundary candidates processed together) and the number of classes (number of boundary states). Discriminative sequence classifiers, such as CRFs, also need to repeatedly perform inference on the training data, which might become expensive.

2.4 Performances of the Approaches

For sentence segmentation in speech, performance is usually evaluated using the error rate (ratio of number of errors to the number of examples), F_1-measure (the harmonic mean of recall and precision, where recall is defined as the ratio of the number of correctly returned sentence boundaries to the number of sentence boundaries in the reference annotations and precision is the ratio of the number of correctly returned sentence boundaries to the number of all automatically estimated sentence boundaries), and the National Institute of Standards and Technology (NIST) error rate (number of candidates wrongly labeled divided by the number of actual boundaries).

For sentence segmentation in text, researchers have reported error rate results on a subset of the Wall Street Journal Corpus of about 27,000 sentences. For instance, Mikheev [39] reports that his rule-based system performs at an error rate of 1.41%. The addition of an abbreviation list to this system lowers its error rate to 0.45%, and combining it with a supervised classifier using POS tag features leads to an error rate of 0.31%. Without requiring handcrafted rules or an abbreviation list, Gillick's SVM-based system [21] obtains even fewer errors, at 0.25%. Even though the error rates presented seem low, sentence segmentation is one of the first processing steps for any NLP task, and each error impacts subsequent steps, especially if the resulting sentences are presented to the user as for example, in extractive summarization.

For sentence segmentation in speech, Doss et al. [63] report on the Mandarin TDT4 Multilingual Broadcast News Speech Corpus an F_1-measure of 69.1% for a MaxEnt classifier, 72.6% with Adaboost, and 72.7% with SVMs, using the same set of features. A combination of the three classifiers using logistic regression is also proposed. On a Turkish broadcast news corpus, Guz et al. [31] report an F_1-measure of 78.2% with HELM, 86.2% with fHELM with morphology features, 86.9% with Adaboost, and 89.1% with CRFs. In these results, HELMs (and fHELMs) were trained on the same corpus as the other classifiers. They can, however, be trained on a much larger corpus and improve performance when combined with discriminative classifiers. For instance, Zimmerman et al. [64] report that on the English TDT4 broadcast news corpus, Adaboost combined with HELM performs at an F_1-measure of 67.3% compared to 65.5% for Adaboost alone.

2.5 Features

Although most approaches are tightly related to the kinds of features employed, it is beneficial for demonstrative purposes to decouple these. Similarly, although most feature categories, such as lexical or prosodic features, are common in sentence and topic segmentation,

their usage is very different. We refer to "segmentation" when features apply to both sentence and topic segmentation and explicitly state the kind of segmentation otherwise.

In this section, we describe the features of a potential boundary observation as the dimensions of the vector x. A feature f can be either binary (presence of a trigger word denoted by $x_f = 1$ or absence thereof denoted by $x_f = 0$) or can take values with $x_f \in \mathbb{R}$ (e.g., the length of a sentence, the duration of a pause). For binary features, in the following, we replace $x_f = 1$ by x_f and omit $x_f = 0$.

Certain classifiers assume properties for the input features and may require that they are all binary or may prefer that their distribution be standardized. Real-valued features can be converted to binary features by quantification and projection in a space of larger dimension so that the value of the feature being in an interval results in its corresponding dimension in the projected space having a value of 1 while the others yield 0.

2.5.1 Features for Both Text and Speech

Lexical Features

For both text and speech and for both sentence and topic segmentation, lexical features are the key features. Sentence and topic initial and final tokens and phrases can be captured via statistical machine learning methods, as described earlier. Typically, windows of n tokens (or sentences) are analyzed for sentence (or topic) segmentation. Whereas sequence classification methods perform this analysis implicitly, local classification methods can be fed corresponding features, such as the overlap of content words compared to the previous sentence.

For sentence segmentation of text, the lexical cues are tokens in text, and the task is mainly disambiguating sentence final punctuation. For speech, the lexical cues are raw tokens because speech lacks typographic cues.

Note that lexical features have two kinds of usage. The first one is based on the occurrence of lexical features around boundaries, such as cue phrases. For example, in the Broadcast News corpus of TDT, the news elements (i.e., topics) typically end with similar phrases. This first usage is described as "discourse features." The second is similar to TextTiling-based approaches, which typically employ stems of content words that are used while computing the cosine distance. The former usage is dependent on the genre and language, and the second usage is domain independent. These two usages are not alternatives to each other and can be combined in a single classification framework. Reynar's work [47] can be seen as a pioneering study for achieving this framework. In a maximum entropy framework, Reynar used the count of content words and names repeated in the window before and after the boundary.

More formally, let $w_1, w_2, \ldots, w_n$ be the tokens of the input, and let us extract lexical features for the boundary candidate between w_i and w_{i+1}. For sentence segmentation, the most relevant features are generally token n-grams before, after, and across the boundary. For the case of bigrams, this results in extracting the following features: x_{w_{i-1},w_i}, $x_{w_{i+1},w_{i+2}}$ and $x_{w_i,w_{i+1}}$. The cross-boundary features, for example, capture the fact that a sentence boundary is unlikely after *Gov. Smith*, but is likely in *government. The*.

For topic segmentation, boundary candidates occur between sentences. If the sentence before the boundary is denoted s_i, and the sentence after the boundary is denoted s_{i+1}, the presence of cue phrase c in those sentences will be represented as $x_{c \in s_i}$ and $x_{c \in s_{i+1}}$.

A second type of feature is the similarity of the content before and after the boundary, typically expressed as the cosine similarity between the previous and next sentences.

$$x_{\text{cosine}(s_i,s_{i+1})} = \frac{\sum_w \text{tf}(w,s_i)\text{tf}(w,s_{i+1})\text{idf}(w)}{\sqrt{\sum_w (\text{tf}(w,s_i)\text{idf}(w))^2}\sqrt{\sum_w (\text{tf}(w,s_{i+1})\text{idf}(w))^2}}$$

where $\text{tf}(w,s) = \frac{n_{w,s}}{\sum_w n_{w,s}}$ represents the term frequency of token w in sentence s and $\text{idf}(w) = \log \frac{D}{\text{df}(w)}$ is the inverse document frequency of that token, which show how common it is, generally computed on a separate corpus (D is the total number of documents, $\text{df}(w)$ is the number of documents containing w). The content can be compared at different levels: for instance, n sentences before the boundary and n sentences after the boundary.

Lexical chains are another relevant feature for topic segmentation. We usually compute the number of chains that start and stop at a candidate boundary. Let $c \in C$ be a set of words referring to a lexical chain (for example, *leaf*, *rose*, *flower*). For practical reasons, a lexical chain is often reduced to a single token (all occurrences of *leaf*). Then, for a candidate boundary between w_i and w_{i+1}, the broken-lexical-chain feature can be computed as

$$x_{\text{chain}} = \left| \left\{ c \in C : \min_{\substack{k \le i, j > i \\ w_k, w_l \in c}} l - k > d_{\min} \right\} \right|$$

Most automatic topic segmentation work based on text sources has explored topical word usage cues in one form or other. Kozima [65] used mutual similarity of words in a sequence of text as an indicator of text structure. Reynar [66] presented a method that finds topically similar regions in the text by graphically modeling the distribution of word repetitions. Ponte and Croft [67] extracted related word sets for topic segments with the information retrieval technique of local context analysis and then compared the expanded word sets.

Beeferman et al. [48] combined a large set of automatically selected lexical discourse cues in a maximum entropy model. They also incorporated topical word usage into the model by building two statistical language models: one static (topic independent) and one that adapts its word predictions on the basis of past words. They showed that the log likelihood ratio of the two predictors behaves as an indicator of topic boundaries and can thus be used as an additional feature in the exponential model classifier.

Syntactic Features

Syntactic information has been successfully captured by a number of studies. Mikheev [39] implicitly used POS tags for sentence segmentation. Similarly, for global reranking approaches as described in Section 2.2.5, syntactic features in the form of constituency trees or dependency parse trees are also used.

For morphologically rich languages, such as Czech and Turkish, morphological analyses of words are used as additional cues [31, 68].

Formally, let $t_1, \ldots, t_n$ be the sequence of POS or morphologic tags extracted for words $w_1, \ldots, w_n$. The same features can be extracted as for words (n-grams before, after, and across the candidate boundary), for example, x_{t_{i-1},t_i}, $x_{t_i,t_{i+1}}$ and $x_{t_{i+1},t_{i+2}}$. Syntactic features are typically less useful for topic segmentation because topic changes are usually characterized by content shifts.

To assess the grammaticality of a sentence candidate in the global model under a probabilistic context-free grammar (PCFG), we can compute the sum of the probability of all valid parse trees for that sentence:

$$x_{pcfg} = \sum_{t \in s_t} P(t) = \sum_{t \in s_t} \prod_{r \in t} P(r)$$

where t is a parse tree and r is a production rule used in that tree [69].

Discourse Features

Speech or text, discourse features are always important for segmentation. For example, in a broadcast news show, the anchor first gives the headlines, then a commercial follows, and then the stories are presented one by one with optional anchor/reporter interaction and typical topic start and end phrases.

Previous work on both text and speech segmentation has shown that cue phrases or discourse particles (items such as *now* or *by the way*), as well as other lexical cues, can provide valuable indicators of structural units in discourse [e.g., 70,71]. Similarly, for speech, change of speaker may indicate a sentence boundary, and commercials may indicate a topic boundary in broadcast news or conversations. Formally, for all events $e \in \mathcal{E}$ that appear in the vicinity of a boundary, a feature x_e can be generated to represent the occurrence of that event, and if relevant, $x_{\bar{e}}$ will be used to represent the nonoccurrence of that event. Events have to be detected using additional systems not detailed in this book (such as a commercial detector) that may output confidence scores. In this case, the feature will be $x_e = cs$ where cs is the confidence score for that event to be recognized.

Whereas earlier approaches try to capture such predetermined discourse cues, more corpus-based studies rely on the machine learning approaches to automatically learn such patterns using informative feature sets. For example, Tur et al. [29] used explicit HMM states for topic initial and final sentences, which improved performance greatly. Rosenberg and Hirschberg [50] used statistical hypothesis testing for predetermining such phrases.

For meeting or conversation segmentation, discourse features are more complex and rely on argumentation structure. Most studies simply use previous and next turns as discourse features, but higher-level semantic information such as dialog act tags or meeting agenda items can also be used for exploiting discourse information [72].

2.5.2 Features Only for Text

Typographical and Structural Features

For sentence and topic segmentation, typographical and structural cues, such as punctuation and headlines, are very informative. Sentence segmentation systems use words and punctuation before and after the boundary, capitalization and POS tags of those words, their length, and how frequently they are used in nonsentence boundary contexts (e.g., before a lowercase word) compared to at the end/beginning of a sentence. Similarly, gazetteer information containing abbreviations and preprocessing and postprocessing patterns is employed to process text.

Formally, let g be a set of words that appear in a gazetteer. A feature is generated so that $x_{g(w)} = 1$ if $w \in g$. Similarly, the feature that denotes the frequency of the lowercase

form (flc) of a word can be comp... $C_{flc}(w) = \frac{[\text{flc}(w)]}{\ldots}$ ... denotes the lowercase version of w.

In his work on sentence segmentation, ... showed that on a given set of features, the choice of a classifier had a much smaller impact than a mismatch between the training and the test data or a mismatch on the tokenization of the input words. Kiss and Strunk [73] proposed an unsupervised approach for finding sentence boundaries that learns abbreviations using global statistics on an unlabeled corpus. Even though the approach is independent of the language, it is unable to identify abbreviations if they are not used multiple times in the test corpus.

Other structural cues include paragraph boundaries, headlines, and section numbering. Such cues appear only in structured textual sources and may not exist in certain text such as blogs and chatrooms.

2.5.3 Features for Speech

When working with speech recognition output, some words may be incorrect due to recognition errors, degrading the quality of lexical features. Similarly, token start times and their durations may also be wrongly estimated, causing errors in prosodic feature computation. Typically, a large set of prosodic features are extracted for robustness to these errors.

Prosodic Features

When applying segmentation to speech rather than written text, many of the same approaches can be used, but with some important considerations. First, in the case of automatic processing of speech, lexical information comes from the output of a speech recognition, which typically contains errors. Second, spoken language lacks explicit punctuation, capitalization, and formatting information. Rather, this information is conveyed through the language and also through prosody, as explained shortly. Third, although some spoken language, such as news broadcasts, is read from a text, most natural speech is conversational. In natural, spontaneous speech, sentences can be "ungrammatical" (from the perspective of formal syntax) and typically contain significant numbers of normal speech disfluencies, such as filled pauses, repetitions, and repairs.

Spoken language input, on the other hand, provides additional, "beyond words" information through its intonational and rhythmic information, that is, through its **prosody**. Prosody refers to patterns in pitch (fundamental frequency), loudness (energy), and timing (as conveyed through pausing and phonetic durations). Prosodic cues are known to be relevant to discourse structure in spontaneous speech and can therefore be expected to play a role in indicating sentence boundaries and topic transitions. Furthermore, prosodic cues by their nature are in principle independent of word identity. Thus they tend to suffer less than do lexical features from errors in automatic speech recognition.

Figure 2-5 depicts some general prosodic features used for segmenting speech into sentences along with lexical features. Broadly speaking, the prosodic features associated with sentence boundaries are similar to those for topic boundaries because both involve conveying a break that serves to chunk information. Pause length, and pitch and energy resets are generally greater in magnitude for the larger (i.e., topic) breaks, but similar types of prosodic features can be used for both tasks, trained of course for the task at hand.

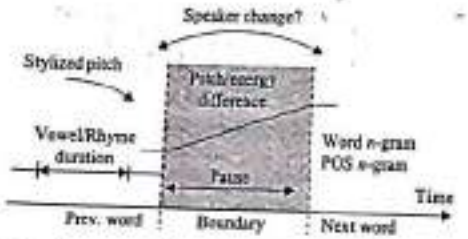


Figure 2–5: Some basic prosodic and lexical features for speech segmentation

Prosodic features for sentence segmentation have been used in a number of studies [74, 75, 27, 76, 77, 78, 51, 11, 79, 60, 80]. The simplest and most often used feature is a pause at the boundary of interest. For automatic processing, pauses are more easily obtained than other prosodic features because, unlike pitch and energy features, pause information can be extracted from automatic speech recognition output. Of course, not all sentence boundaries contain pauses, particularly in spontaneous speech. And conversely, not all pauses correspond to sentence boundaries. For example, many sentence-internal disfluencies also contain pauses. Some methods use simply the presence of a pause; others model the duration of the pause. Pause durations can be quite large in the case of turn-final sentence boundaries in conversation because such regions correspond to time during which another participant is talking. Sentence segmentation for certain dialog acts, such as backchannels (e.g., "uh-huh"), which tend to occur in isolated turns, can thus be achieved fairly successfully using only pause information.

The pause feature is computed as $x_{pause} = start(w_{i+1}) - end(w_i)$ where $start()$ and $end()$ represent the timing in seconds of the beginning and the end of a word in the speech recognition output. Relevant side features are the pause before the word (to know if it is isolated) and the quantized pause $x_{qpause}(w_i) = 1$ iff $x_{pause} > thr_{pause}$, where thr_{pause} is set to, for example, 0.2 second. Pause duration does not follow a normal distribution, by nature, and tends to confuse classifiers that expect such a distribution. However, this single feature is often the most relevant one for segmenting speech.

More detailed prosodic modeling has included pitch, phone duration, and energy information. Pitch is captured by modeling fundamental frequency during voiced regions of speech. Pitch conveys a wide range of types of information, including information about the prominence of a syllable, but for sentence segmentation the goal is usually to capture a reset in pitch. Thus, methods have looked at pitch differences across a word boundary, with a larger negative difference indicating higher probability of a sentence boundary. In addition to modeling the break in pitch across a word boundary, some approaches [27] have also modeled a speaker-specific value to which pitch falls at the ends of utterances, which not only improves performance but also allows for causal modeling because it does not rely on speech after the pause [81].

Pitch is not a continuous function and cannot be computed outside of voiced regions. Therefore, pitch features can be undefined for a given boundary candidate, which might be a problem with certain classifiers. Computing pitch, smoothing and interpolating it properly, is not the matter of this book and should be handled by appropriate software, such as

the widely used Praat toolkit [82]. Typically, features are computed from statistics of pitch values in a window before the end of the word before the candidate boundary and after the beginning of the word after the boundary. For example, the pitch difference feature described in the previous paragraph results in

$$x_{pitch} = \left(\max_{t \in W_s(w_i)} pitch(t)\right) - \left(\min_{t \in W_s(w_{i+1})} pitch(t)\right)$$

where $pitch(t)$ is the pitch value at time t, $W_s(w_i)$ is a temporal window anchored at the end of word w_i, and $W_s(w_{i+1})$ is a similar window at the start of word w_{i+1}. Variants of this feature can be created by changing the window size (i.e., 200 ms, 500 ms), changing the statistics computed on both sides of the boundary (i.e., min, max, mean), and normalizing pitch values according to different factors (i.e., log space projection, standardization of the distribution of pitch values of the current speaker).

Duration features for sentence segmentation aim to capture a phenomenon known as **preboundary lengthening** in which the last region of speech before the end of a unit is stretched out in duration. (Interestingly, this phenomenon is also observed in music and even in bird song [83].) Automatic modeling methods best capture preboundary lengthening when phone durations are normalized by the average duration of those phones in a corpus of similar speaking style. The duration of the rhyme (the vowel and any following consonants) of a prefinal syllable typically shows more lengthening than does the onset of that syllable.

For example, let v be the last vowel in w_i, the word before the boundary candidate. A feature can be computed as the relative duration of that vowel compared to its average duration in a corpus C.

$$x_{vowel} = \frac{\text{end}(v_{w_i}) - \text{start}(v_{w_i})}{\sum_{w \in C} \text{end}(v_w) - \text{start}(v_w)}$$

Energy features have also been employed in sentence boundary modeling, but with less success. From a descriptive point of view, energy behaves somewhat like pitch, falling toward the end of a sentence and often showing a reset for the next sentence. However, energy is affected by a myriad of factors, including the recording itself, and can be difficult to normalize both within and across talkers. Thus it has in general been less successful than pause, pitch, and duration features for automatic segmentation.

A final feature that is sometimes considered in prosodic modeling is voice quality. Descriptive work has shown an association between sentence boundaries and voice quality changes, but because such phenomena are highly speaker dependent and difficult to capture automatically, most automatic segmentation work has relied on the previously mentioned prosodic features.

Descriptive work on topic boundaries has found that major shifts in topic typically show longer pauses, an extra-high F0 onset or reset, a higher maximum accent peak, shifts in speaking rate, and greater range in F0 and intensity [e.g., 84, 85, 86, 87, 27]. Such cues are known to be salient for human listeners; in fact, subjects can perceive major discourse boundaries even if the speech itself is made unintelligible via spectral filtering [88]. In automatic studies of topic shifts, Galley et al. [45] found that features such as changes in speaker activity, amounts of silence and overlapping speech, and the presence of certain cue phrases were all indicative of changes in topic, and adding them to their approach improved their

segmentation accuracy significantly. Georgescul, Clark, and Armstrong [89] found that similar features also gave some improvement with their approach. However, Hsueh, Moore, and Renals [90] found this to be true only for coarse-grained topic shifts (corresponding in many cases to changes in the activity or state of the meeting, such as introductions or closing review) and that detection of finer-grained shifts in subject matter showed no improvement.

2.6 Processing Stages

Usually, the first step in the segmentation tasks is preprocessing to determine tokens and candidate boundaries. In language like English, words are candidate tokens, but special cases like abbreviations and acronyms exist. In languages like Mandarin, with textual sources, a preceding word segmentation step can be employed.

Then a set of features, as described in the previous section, is extracted for each candidate. For speech data, token start times and durations are usually not available in the reference annotations of the spoken utterances, but these are necessary for computing prosodic features. Usually, a forced alignment of decoding step is performed to obtain these features.

Once the features are extracted, each candidate boundary is classified using one of the methods described in the previous sections.

For testing, the automatically estimated token boundaries are compared to the boundaries in reference transcriptions. When speech recognition output is used for training or testing, reference tokens are aligned with speech recognition output words using dynamic programming to minimize alignment error (such as using NIST `sclite` alignment tools), and boundary annotations are transferred to the speech recognition output. Unfortunately, sometimes perfect alignment is not possible. For example, two tokens in reference annotations with a sentence boundary between them may be recognized by the speech recognizer as a single token. In such cases, it is not clear if the sentence boundary should be omitted from the speech recognition annotations or should be included so a heuristic rule is used.

2.7 Discussion

Although sentence segmentation is a useful step for many language processing tasks, careful optimization of the segmentation parameters directly for the following task in comparison to independent optimization for segmentation quality of the predicted sentence boundaries has been empirically shown to be useful. For example, Walker et al. [91] observed that the hardcoded rules for sentence segmentation in a machine translation system resulted in very poor sentence segmentation generalization performance compared to the use of a machine learning approach. Matusov et al. [92] show that optimizing parameters of sentence segmentation in the source language is useful for machine translation of spoken documents. Similarly, Favre et al. [93] and Liu and Xie [94] study the effect of parameter optimization on information extraction and speech summarization, respectively, instead of optimization on the sentence segmentation task itself.

Regarding topic segmentation, automatic transcription of speech uses language models to predict topical information in the language model, and this has been shown to improve ASR, either by selecting a language model trained on a matching topic or by building a general language model wherein the topic is a latent variable estimated during decoding. More generally, topic-driven domain adaptation is used in a wide range of natural language processing tasks. In information retrieval, topic is modeled explicitly [95] by allowing words to contribute differently in function of the topic in which they occur or implicitly [96] using co-occurrence space reduction techniques. In automatic summarization, Tang, Yao, and Chen [97] propose to reconsider the common assumption that a document is made of a single topic and include topic-specific information in their model. Word-sense disambiguation also benefits from topic information, as many words have probably a dominant sense in a given topic [98].

2.8 Summary

We described the tasks of sentence and topic segmentation for text and speech input. We described learning algorithms for these tasks in several categories. Depending on the type of input (i.e., text versus speech), several different types of features may be used for these tasks. For example, in text, typographical cues such as capitalization and punctuation can be benefical, whereas in speech, prosodic features may be useful.

In parallel with the recent advances in speech processing and discriminative machine learning methods, performance of sentence and topic segmentation systems have improved by exploiting very high-dimensional feature sets. However, these systems still make errors requiring the follow-on processing stages, such as machine translation, to be robust to such noise. Further research is required for jointly optimizing the segmentation stage with the follow-on processing systems.

Bibliography

[1] J. Mrozinski, E. W. D. Whittaker, P. Chatain, and S. Furui, "Automatic sentence segmentation of speech for automatic summarization," in *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2005.

[2] J. Makhoul, A. Baron, I. Bulyko, L. Nguyen, L. Ramshaw, D. Stallard, R. Schwartz, and B. Xiang, "The effects of speech recognition and punctuation on information extraction performance," in *Proceedings of International Conference on Spoken Language Processing (Interspeech)*, 2005.

[3] D. Jones, W. Shen, E. Shriberg, A. Stolcke, T. Kamm, and D. Reynolds, "Two experiments comparing reading with listening for human processing of conversational telephone speech," in *Proceedings of EUROSPEECH*, pp. 1145–1148, 2005.

[4] W. Francis, H. Kučera, and A. Mackie, *Frequency Analysis of English Usage: Lexicon and Grammar*. Boston: Houghton Mifflin, 1982.

[88] M. Swerts, R. Geluykens, and J. Terken, "Prosodic correlates of discourse units in spontaneous speech," in Ohala et al. [100] pp. 421–424.

[89] M. Georgescul, A. Clark, and S. Armstrong, "Exploiting structural meeting-specific features for topic segmentation," in *Actes de la 14è me Conférence sur le Traitement Automatique des Langues Naturelles*, Association pour le Traitement Automatique des Langues, June 2007.

[90] P.-Y. Hsueh, J. Moore, and S. Renals, "Automatic segmentation of multiparty dialogue," in *Proceedings of the Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, 2006.

[91] D. Walker, D. Clements, M. Darwin, and J. Amtrup, "Sentence boundary detection: A comparison of paradigms for improving MT quality," in *Proceedings of the MT Summit VIII*, 2001.

[92] E. Matusov, D. Hillard, M. Magimai-Doss, D. Hakkani-Tür, M. Ostendorf, and H. Ney, "Improving speech translation with automatic boundary prediction," in *Proceedings of International Conference on Spoken Language Processing (Interspeech)*, 2007.

[93] B. Favre, R. Grishman, D. Hillard, H. Ji, D. Hakkani-Tür, and M. Ostendorf, "Punctuating speech for information extraction," in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2008.

[94] Y. Liu and S. Xie, "Impact of automatic sentence segmentation on meeting summarization," in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2008.

[95] J. Becker and D. Kuropka, "Topic-based vector space model," in *Proceedings of the 6th International Conference on Business Information Systems*, pp. 7–12, 2003.

[96] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science*, vol. 41, no. 6, pp. 391–407, 1990.

[97] J. Tang, L. Yao, and D. Chen, "Multi-topic based query-oriented summarization," in *Proceedings of SDM*, 2009.

[98] J. Boyd-Graber, D. Blei, and X. Zhu, "A topic model for word sense disambiguation," in *Proceedings of the the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pp. 1024–1033, 2007.

[99] *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics, New Mexico State University, Las Cruces, New Mexico*. Morriston, NJ: ACL, 1994.

[100] J. J. Ohala, T. M. Nearey, B. L. Derwing, M. M. Hodge, and G. E. Wiebe, eds., *Proceedings of the International Conference on Spoken Language Processing*, Edmonton: University of Alberta, 1992.

Chapter 3

Syntax

Anoop Sarkar

Parsing uncovers the hidden structure of linguistic input. In many applications involving natural language, the underlying predicate-argument structure of sentences can be useful. The syntactic analysis of language provides a means to explicitly discover the various predicate-argument dependencies that may exist in a sentence. In natural language processing (NLP), the syntactic analysis of natural language input can vary from being very low-level, such as simply tagging each word in the sentence with a part of speech (POS), or very high level, such as recovering a structural analysis that identifies the dependency between each predicate in the sentence and its explicit and implicit arguments. The major bottleneck in parsing natural language is the fact that ambiguity is so pervasive. In syntactic parsing, ambiguity is a particularly difficult problem because the most plausible analysis has to be chosen from an exponentially large number of alternative analyses. From tagging to full parsing, algorithms that can handle such ambiguity have to be carefully chosen. This chapter explores syntactic analysis methods from tagging to full parsing and the use of supervised machine learning to deal with ambiguity.

3.1 Parsing Natural Language

In a text-to-speech application, input sentences are to be converted to a spoken output that should sound like it was spoken by a native speaker of the language. Consider the following pair of sentences (imagine them spoken rather than written):[1]

1. He wanted to go for a drive in movie.
2. He wanted to go for a drive in the country.

There is a natural pause between the words drive and in in sentence 2 that reflects an underlying hidden structure to the sentence. Parsing can provide a structural description that identifies such a break in the intonation. A simpler case occurs in the following sentence:

3. The cat who lives dangerously had nine lives.

1. When written, drive in would probably be hyphenated in the second utterance.

In this case, a text-to-speech system needs to know that the first instance of the word *lives* is a verb and the second instance is a noun before it can begin to produce the natural intonation for this sentence. This is an instance of the part-of-speech (POS) tagging problem where each word in the sentence is assigned a most likely part of speech. These examples come from the open-source Festival text-to-speech system (www.festvox.org), which uses parsing to disambiguate these cases.

Another motivation for parsing comes from the natural language task of summarization, in which several documents about the same topic should be condensed down to a small digest of information typically limited in size to 100 or 250 words. Such a summary may be in response to a question that is answered (perhaps in different ways) in the set of documents. In this case, a useful subtask is to compress an individual sentence so that only the relevant portions of a sentence is included in the summary [1]. This allows the summary to be concise, informative, and fluent. For example, we may want to compress sentence 4 to a shorter sentence 5.

4. Beyond the basic level, the operations of the three products vary widely.
5. The operations of the products vary.

An elegant way to approach this task is to first parse the sentence to find the various constituents: where we recursively partition the words in the sentence into individual phrases such as a verb phrase or a noun phrase. The output of the parser for the input sentence 4 is shown in Figure 3–1. The parse tree produced by the parser can now be edited using a compression model that is aware of constituents, and a few choice constituent deletions can produce a fluent compressed version of the original sentence.

Another example is the paraphrasing of text [2]. In the sentence fragment 6, the capitalized phrase EUROPEAN COUNTRIES can be replaced with other phrases without changing the essential meaning of the sentence. A few examples of replacement phrases are shown in italics in sentence fragments 7 to 11. This kind of replacement cannot simply rely on substitution of arbitrary words in the sentence because such an approach can lead to

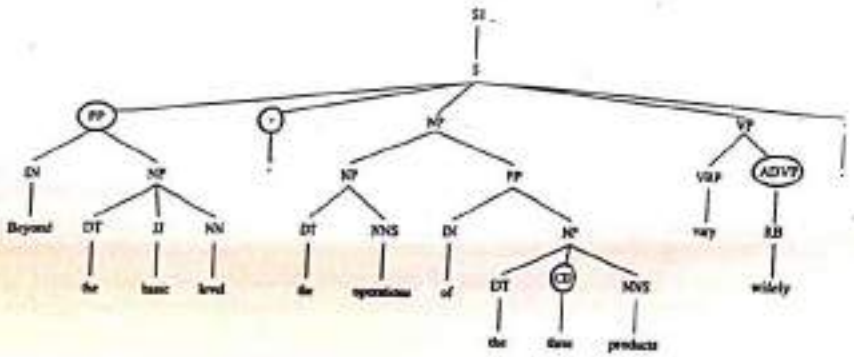


Figure 3–1: Parser output for sentence 4. Deleting the circled constituents PP, ,, CD, and ADVP results in the shorter fluent sentence *The operations of the products vary.* (example from Knight and Marcu [1])

incoherent and disfluent paraphrases. Paraphrasing models build on top of parsers to identify target constituents to replace and also to find appropriate replacement phrases that can substitute for the original phrase. Paraphrases of this type have been shown to be useful in applications such as statistical machine translation.

6. open borders imply increasing racial fragmentation in EUROPEAN COUNTRIES.
7. open borders imply increasing racial fragmentation in *the countries of europe.*
8. open borders imply increasing racial fragmentation in *european states.*
9. open borders imply increasing racial fragmentation in *europe.*
10. open borders imply increasing racial fragmentation in *european nations.*
11. open borders imply increasing racial fragmentation in *the european countries.*

In contemporary NLP, syntactic parsers are routinely used in many applications, including but not limited to statistical machine translation [3], information extraction from text collections [4], language summarization [5], producing entity grids for language generation [6], error correction in text [7], knowledge acquisition from language (e.g., discovering semantic classes or x IS-A y relationships) [8], in speech recognition systems as language models (a language model assigns a probability to a candidate output sentence—syntax is useful in particular for disfluent or error-prone speech input) [7], dialog systems [9], and text-to-speech systems (www.festvox.org). Parsers have been written for a large number of languages around the world and are an essential component in many kinds of multilingual processing tasks.

3.2 Treebanks: A Data-Driven Approach to Syntax

Parsing recovers information that is not explicit in the input sentence. This implies that a parser requires some knowledge in addition to the input sentence about the kind of syntactic analysis that should be produced as output. One method to provide such knowledge to the parser is to write down a grammar of the language—a set of rules of syntactic analysis. For instance, we might write down the rules of syntax as a context-free grammar (CFG). In the rest of this chapter we assume some familiarity with CFGs (please refer to Sipser [10] for a good introduction to the notion of a formal grammar and the formal languages they generate and CFGs in particular).

The following CFG (written in a simple Backus-Naur form) represents a simple grammar of transitive verbs in English, verbs (V) that have a subject and object noun phrase (NP), plus modifiers of verb phrases (VP) in the form of prepositional phrases (PP).

```
S -> NP VP
NP -> 'John' | 'pockets' | D N | NP PP
VP -> V NP | VP PP
V -> 'bought'
D -> 'a'
```

```
N -> 'shirt'
PP -> P NP
P -> 'with'
```

Natural language grammars typically have the words w as terminal symbols in the CFG, and they are generated by rules of type $X \rightarrow w$, where X is the part of speech for the word w. For example, in the preceding CFG, the rule `V -> 'saw'` has the POS symbol V generating the verb *saw*. Such nonterminals are called **part-of-speech tags** or **preterminals**. The preceding CFG can produce a syntax analysis of a sentence like *John bought a shirt with pockets* with S as the start symbol of the grammar. Parsing the sentence with the CFG rules gives us two possible derivations for this sentence. In one parse, pockets are a kind of currency that can be used to buy a shirt, and the other parse, which is the more plausible one, John is purchasing a kind of shirt that has pockets.

```
(S (NP John)
   (VP (VP (V bought)
           (NP (D a)
               (N shirt)))
       (PP (P with)
           (NP pockets))))
```

```
(S (NP John)
   (VP (V bought)
       (NP (NP (D a)
               (N shirt))
           (PP (P with)
               (NP pockets)))))
```

However, writing down a CFG for the syntactic analysis of natural language is problematic. Unlike a programming language, natural language is far too complex to simply list all the syntactic rules in terms of a CFG. A simple list of rules does not consider interactions between different components in the grammar. We could extend this grammar to include other types of verbs and other syntactic constructions, but listing all possible syntactic constructions in a language is a difficult task. In addition, it is difficult to exhaustively list lexical properties of words, for instance, to list all the grammar rules in which a particular word can be a participant. This is a typical knowledge acquisition problem.

Apart from this knowledge acquisition problem, there is a less apparent problem: it turns out that the rules interact with each other in combinatorially explosive ways. Consider a simple CFG that provides a syntactic analysis of noun phrases as a binary branching tree:

```
N -> N N
N -> 'natural' | 'language' | 'processing' | 'book'
```

Recursive rules produce ambiguity: with N as the start symbol, for the input *natural* there is one parse tree `(N natural)`; for the input *natural language* we use the recursive rule once and obtain one parse tree `(N (N natural) (N language))`; for the input *natural language processing* we use the recursive rule twice in each parse, and there are two ambiguous parses:

```
(N (N (N natural)
      (N language))
   (N processing))
```

```
(N (N natural)
   (N (N language)
      (N processing)))
```


Note that the ambiguity in the syntactic analysis reflects a real ambiguity: is it a processing of natural language, or is it a natural way to do language processing? So this issue cannot be resolved by changing the formalism in which the rules are written (e.g., by using finite-state automata, which can be deterministic but cannot simultaneously model both meanings in a single grammar). Any system of writing down syntactic rules should represent this ambiguity. However, by using the recursive rule three times, we get five parses for *natural language processing book* and for longer and longer input noun phrases, using the recursive rule four times, we get 14 parses; using it five times, we get 42 parses; using it six times, we get 132 parses. In fact, for CFGs it can be proved that the number of parses obtained by using the recursive rule n times is the Catalan number of n:

$$\text{Cat}(n) = \frac{1}{n+1}\binom{2n}{n}$$

This occurs not only for coordinate structures such as the noun phrase grammar but also when you have recursive rules to deal with modifiers such as the recursive rule for prepositional phrase modification `VP -> VP PP` in the first CFG in this section. In fact, the ambiguity of PP modification is not independent of the ambiguity of coordination: in a sentence with both types of ambiguity, the total number of parses is the cross product of the parses from each subgrammar. This poses a serious computational problem for parsers. For an input with n words, the number of possible parses is exponential in n.

For most natural language tasks, we do not wish to explore this entire space of ambiguity, even if, as we show later, it is possible to produce a compact representation of the entire exponential number of parses in polynomial time (for CFGs, $O(n^3)$ is the worst-case time complexity) and store it in polynomial space (for CFGs, the space needed is proportional to n^2).

For example, for the input *natural language processing book*, only one out of the five parses obtained using the CFG above is intuitively correct (corresponding to a book about the processing of natural language):

```
(N (N (N (N natural)
         (N language))
      (N processing))
   (N book))
```

This is a second knowledge acquisition problem—not only do we need to know the syntactic rules for a particular language, but we also need to know which analysis is the most plausible for a given input sentence. The construction of a **treebank** is a data-driven approach to syntax analysis that allows us to address both of these knowledge acquisition bottlenecks in one stroke.

A treebank is simply a collection of sentences (also called a corpus of text), where each sentence is provided a complete syntax analysis. The syntactic analysis for each sentence has been judged by a human expert as the most plausible analysis for that sentence. A lot of care is taken during the human annotation process to ensure that a consistent treatment is provided across the treebank for related grammatical phenomena. A style book or set

of annotation guidelines is typically written before the annotation process to ensure a consistent scheme of annotation throughout the treebank.

There is no set of syntactic rules or linguistic grammar explicitly provided by a treebank, and typically there is no list of syntactic constructions provided explicitly in a treebank. In fact, no exhaustive set of rules is even assumed to exist, even though assumptions about syntax are implicit in a treebank. A detailed set of assumptions about syntax is typically used as an annotation guideline to help the human experts produce the single-most plausible syntactic analysis for each sentence in the corpus. The consistency of syntax analysis in a treebank is measured using interannotator agreement by having approximately 10% overlapped material annotated by more than one annotator.

Treebanks provide a solution to the two kinds of knowledge acquisition bottlenecks we discussed. Treebanks provide annotations of syntactic structure for a large sample of sentences. We can use supervised machine learning methods to train a parser to produce a syntactic analysis for input sentences by generalizing appropriately from the training data extracted from the treebank.

Treebanks solve the first knowledge acquisition problem of finding the grammar underlying the syntax analysis because the syntactic analysis is directly given instead of a grammar. In fact, the parser does not necessarily need any explicit grammar rules as long as it can faithfully produce a syntax analysis for an input sentence, although the information used by the trained parser can be said to represent a set of implicit grammar rules. Nivre [11] discusses in further detail this subtle difference between parsing using a grammar and parsing a text using data-driven methods that may or may not be grammar-based.

Treebanks solve the second knowledge acquisition problem as well. Because each sentence in a treebank has been given its most plausible syntactic analysis, supervised machine learning methods can be used to learn a scoring function over all possible syntax analyses. A statistical parser trained on the treebank tries to mimic the human annotation decisions by using indicators from the input and previous decisions made in the parser itself to learn such a scoring function. For a given sentence not seen in the training data, a statistical parser can use this scoring function to return the syntax analysis that has the highest score, which is taken to be the most plausible analysis for that sentence. The scoring function can also be used to produce the k-best syntax analyses for a sentence.

Two main approaches to syntax analysis are used to construct treebanks: dependency graphs and phrase structure trees. These two representations are very closely related to each other, and under some assumptions, one representation can be converted to another. Dependency analysis is typically favored for languages such as Czech and Turkish, that have free(er) word order, where the arguments of a predicate are often seen in different ordering in the sentence, while phrase structure analysis is often used to provide additional information about long-distance dependencies and mostly in languages like English and French, where the word order is less flexible.

In the rest of this chapter, we examine three main components for building a parser: the representation of the syntactic structure, which involves the use of a varying amount of linguistic knowledge to build a treebank (§3.3); the training and decoding algorithms for the model that deal with the potentially exponential search space (§3.4); methods to model ambiguity and provide a way to rank parses so that we can recover the most likely parse (§3.5).

3.3 Representation of Syntactic Structure

3.3.1 Syntax Analysis Using Dependency Graphs

The main philosophy behind dependency graphs is to connect a word—the head of a phrase—with the dependents in that phrase. The notation connects a head with its dependent using a directed (hence asymmetric) connection [12]. Dependency graphs, just like phrase structure trees, is a representation that is consistent with many different linguistic frameworks. The head-dependent relationship could be either semantic (head-modifier) or syntactic (head-specifier). The main difference between dependency graphs and phrase structure trees is that dependency analyses typically make minimal assumptions about syntactic structure and to avoid any annotation of hidden structure such as, for example, using empty elements as placeholders to represent missing or displaced arguments of predicates, or any unnecessary hierarchical structure. The words in the input sentence are treated as the only vertices in the graph, which are linked together by directed arcs representing syntactic dependencies. The CoNLL 2007 shared task on dependency parsing [13] provides the following definition of a **dependency graph**:

> In dependency-based syntactic parsing, the task is to derive a syntactic structure for an input sentence by identifying the syntactic head of each word in the sentence. This defines a *dependency graph*, where the nodes are the words of the input sentence and the arcs are the binary relations from head to dependent. Often, but not always, it is assumed that all words except one have a syntactic head, which means that the graph will be a tree with the single independent node as the root. In *labeled* dependency parsing, we additionally require the parser to assign a specific type (or label) to each dependency relation holding between head word and dependent word.

As in this definition, we will restrict ourselves to dependency tree analyses, where each word depends on exactly one parent, either another word or a dummy root symbol. By convention, in dependency trees the 0 index is used to indicate the root symbol, and the directed arcs are drawn from the head word to the dependent word. For example, Figure 3-2 shows an example of a dependency tree for a Czech sentence taken from the Prague Dependency Treebank, which is a large corpus of Czech text annotated with dependency trees. Each treebank has its own annotation flavor, and the Prague treebank annotates other levels of information as well, such as topic and focus structure of the sentence, but we only show the dependency tree information here.

There are many variants of dependency syntactic analysis, but the basic textual format for a dependency tree can be written in the following form, where each dependent word specifies the head word in the sentence, and exactly one word is dependent to the root of the sentence. The following shows a typical textual representation of a labeled dependency tree:

Index	Word	Part of Speech	Head	Label
1	They	PRP	2	SBJ
2	persuaded	VBD	0	ROOT
3	Mr.	NNP	4	NMOD
4	Trotter	NNP	2	IOBJ
5	to	TO	6	VMOD
6	take	VB	2	OBJ
7	it	PRP	6	OBJ
8	back	RB	6	PRT
9	.	.	2	P

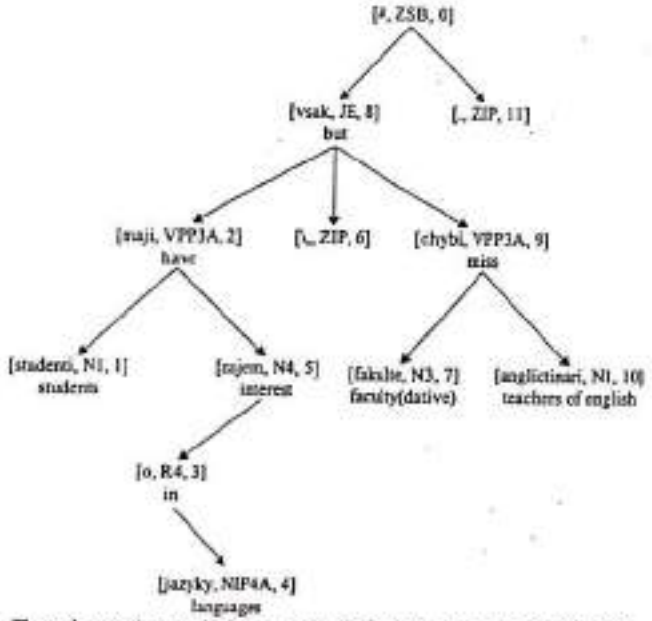


Figure 3–2: An example of a dependency graph syntax analysis for a Czech sentence taken from the Prague Dependency Treebank. Each node in the graph is a word, its part of speech, and the position of the word in the sentence; for example, [fakulte, N3, 7] is the seventh word in the sentence with POS tag N3, which also tells us that the word has dative case. The node [#,ZSB,0] is the root node of the dependency tree. The English equivalent is provided for each node

An important notion in dependency analysis is the notion of **projectivity**, which is a constraint imposed by the linear order of words on the dependencies between words [14]. A **projective dependency tree** is one where if we put the words in a linear order based

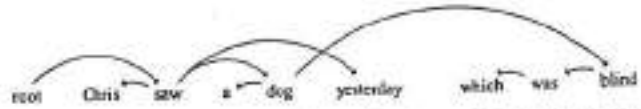


Figure 3–3: An unlabeled nonprojective dependency tree with a crossing dependency

Table 3–1: A multilingual comparison of percentage of crossing dependencies and percentage of sentences with nonprojectivity taken from the CoNLL 2007 shared task data set. Ar = Arabic, Ba = Basque, Ca = Catalan, Ch = Chinese, Cz = Czech, En = English, Gr = Greek, Hu = Hungarian, It = Italian, Tu = Turkish. Note that in some cases the dependency trees were created by conversion via heuristic rules from an original phrase structure tree. From Nivre et al. [13]

	Ar	Ba	Ca	Ch	Cz	En	Gr	Hu	It	Tu
% deps	0.4	2.9	0.1	0.0	1.9	0.3	1.1	2.9	0.5	5.5
% sents	10.1	26.2	2.9	0.0	23.2	6.7	20.3	26.4	7.4	33.3

on the sentence with the root symbol in the first position, the dependency arcs can be drawn above the words without any crossing dependencies. Another way to state projectivity is to say that for each word in the sentence, its descendants form a contiguous substring of the sentence. For example, Figure 3-3 shows a natural analysis of an English sentence that contains an extraposition to the right of a noun phrase modifier phrase, which as a result requires a crossing dependency. However, English has very few cases in a treebank that will need such a nonprojective analysis. In other languages, such Czech and Turkish, the number of nonprojective dependencies can be much higher. As a percentage of the total number of dependencies, crossing dependencies even in those languages are a small percentage. However, a substantial percentage of sentences contain at least one crossing dependency, making it an important issue in some languages. Table 3–1 contains a multilingual comparison of crossing dependencies across the languages that were part of the CoNLL 2007 shared task on dependency parsing.

Dependency graphs in treebanks do not explicitly distinguish between projective and **nonprojective dependency tree** analyses. However, parsing algorithms are sometimes forced to distinguish between projective and nonprojective dependencies. Let us examine this distinction further using CFGs. Note that we can set up dependency links in a CFG. For example, consider the following grammar:

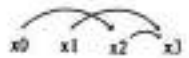
```
X0_2 -> X0_1* X2_1
X0_1 -> x0*
X2_1 -> X1_1 X2_2*
X1_1 -> x1*
X2_2 -> X2_3* X3_1
X2_3 -> x2*
X3_1 -> x3*
```

In this CFG, the terminal symbols are x0, x1, x2, x3, and the asterisk picks out a single symbol in the right-hand side of each rule that specifies the dependency links. We can view the asterisk as either a separate annotation on the nonterminal or simply as a new nonterminal in the probabilistic context-free grammar (PCFG). Abney [15] provides a detailed comparison of the PCFG form for projective dependency graphs and discusses their equivalence in detail. In this example, the dependency tree equivalent to the preceding CFG is as follows:

We can show that if we can convert a dependency tree into an equivalent CFG (using the notation used above), then the dependency tree must be projective. In a CFG converted from a dependency tree, we have only the following three types of rules with one type of rule to introduce the terminal symbols and two rules where Y is dependent on X or vice versa. The head word of X or Y can be traced by following the asterisk symbol.

```
Z -> X* Y
Z -> X Y*
A -> a*
```

Assume that we have a nonprojective dependency tree. For example,



Converting such a dependency tree to a CFG with the asterisk notation gives us two options. Either we can capture that X3 depends on X2 but fail to capture that X1 depends on X3:

```
X2_3 -> X1_1 X2_2*
X1_1 -> x1
X2_2 -> X2_1* X3_1
X2_1 -> x2
X3_1 -> x3
```

or we can capture the fact that X1 depends on X3 but fail to capture that X3 depends on X2:

```
X2_3 -> X1_1 X3_2*
X1_1 -> x1
X3_2 -> X2_1 X3_1*
X2_1 -> x2
X3_1 -> x3
```

In fact, there is no CFG that can capture the nonprojective dependency. Recall that projectivity can be defined as follows: for each word in the sentence, its descendants form a contiguous substring of the sentence. Thus, nonprojectivity can be defined as follows: a nonprojective dependency means that there is a word in the sentence (or equivalently a

nonterminal in the CFG created from the dependency tree) such that its descendants do not form a contiguous substring of the sentence. Put another way, there is a nonterminal Z such that Z derives spans (x_i, x_k) and (x_{k+p}, x_j) for some $p > 0$. This means there must be a rule $Z \rightarrow PQ$ where P derives (x_i, x_k) and Q derives (x_{k+p}, x_j). However, by definition, this is only valid in CFGs if $k = 0$ because P and Q *must* be contiguous substrings. Hence no dependency tree with nonprojective dependencies can be converted into an equivalent (asterisk-marked) CFG.

This gives a useful characterization of projective dependencies in terms of CFG. If we want a dependency parser to only produce projective dependencies, we can implicitly create an equivalent CFG that will ignore all nonprojective dependencies. We explore this topic further when we discuss parsing algorithms.

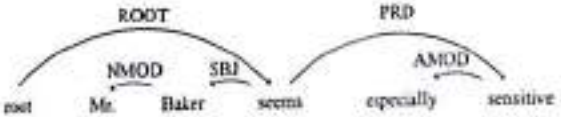
3.3.2 Syntax Analysis Using Phrase Structure Trees

A phrase structure syntax analysis of a sentence derives from the traditional sentence diagrams that partition a sentence into constituents, and larger constituents are formed by merging smaller ones. Phrase structure analysis also typically incorporate ideas from generative grammar (from linguistics) to deal with displaced constituents or apparent long-distance relationships between heads and constituents. A phrase structure tree can be viewed as implicitly having a predicate-argument structure associated with it. For example, the following phrase structure analysis of the sentence *Mr. Baker seems especially sensitive*, taken from the Penn Treebank, shows that the subject of the sentence is marked with the -SBJ marker and the predicate of the sentence is marked with the -PRD marker. The underlying predicate-argument structure is shown below the tree using an informal notation that captures the information implied by the phrase structure tree.

```
(S (NP-SBJ (NNP Mr.)
           (NNP Baker))
   (VP (VBZ seems)
       (ADJP-PRD (RB especially)
                 (JJ sensitive)))))

Predicate-argument structure:
seems((especially(sensitive))(Mr. Baker))
```

The same sentence gets the following dependency tree analysis. Note how some of the information from the bracketing labels from the phrase structure analysis gets mapped onto the labeled arcs of the dependency analysis. Typically, dependency analysis would not link the subject with the predicate directly because it would create an inconvenient crossing dependency with the dependency between *seems* and the root symbol.



To explain some details of phrase structure analysis in treebanks, we use some examples of syntax analysis that show how null elements (constituents with no yield) are used to

localize certain predicate-argument dependencies in the tree structure. The examples are taken from a paper [16] describing the annotation guidelines for the English Penn Treebank, which was a project that annotated 40,000 sentences from the *Wall Street Journal* with phrase structure trees. The POS tags for the words are omitted to simplify the tree.

In the first example, we see that an NP dominates a trace "T", which is a null element, the same as an epsilon symbol in formal language theory, having no yield in the input. This empty trace has an index (here it is 1, but the actual value is not important) and is associated with the WHNP constituent with the same index. This co-indexation allows us to infer the predicate-argument structure shown below the tree.

```
(SBARQ (WHNP-1 What)
       (SQ is (NP-SBJ Tim)
              (VP eating (NP *T*-1)))
       ?)

Predicate-argument structure:
eat(Tim, what)
```

In the second example, the subject of the sentence, *The ball*, is actually not the logical subject of the predicate, which has been displaced due to the passive construction. The logical subject of the sentence, *Chris*, is marked as -LGS, enabling the recovery of the predicate argument structure for this sentence.

```
(S (NP-SBJ-1 The ball)
   (VP was (VP thrown)
           (NP *-1)
           (PP by (NP-LGS Chris))))

Predicate-argument structure:
throw(Chris, the ball)
```

The third example shows that different syntactic phenomena are often combined in the corpus, and both the analyses are combined to provide the predicate-argument structure in such cases.

```
(SBARQ (WHNP-1 Who)
       (SQ was (NP-SBJ-2 *T*-1)
               (VP believed (S (NP-SBJ-3 *-2)
                               (VP to (VP have
                                          (VP been
                                              (VP shot
                                                  (NP *-3))))))))
       ?)

Predicate-argument structure:
believe(*someone*, shoot(*someone*, who))
```

The fourth pair of examples shows how null elements are used to annotate the presence of a subject for a predicate even if it is not explicit in the sentence. In the first case, the

phrase structure annotation in the treebank marks the missing subject for *take back* as the object of the verb *persuaded*.

```
(S (NP-SBJ (PRP They))
   (VP (VP (VBD persuaded)
           (NP-1 (NNP Mr.)
                 (NNP Trotter))
           (S (NP-SBJ (-NONE- *-1))
              (VP (TO to)
                  (VP (VB take)
                      (NP (PRP it))
                      (PRT (RB back))))))))

Predicate argument structure:
persuade(they, Mr. Trotter, take_back(Mr. Trotter, it))
```

In the first case, the phrase structure annotation in the treebank marks the missing subject for *take back* as the subject of the verb *promised*.

```
(S (NP-SBJ-1 (PRP They))
   (VP (VP (VBD promised)
           (NP (NNP Mr.)
               (NNP Trotter))
           (S (NP-SBJ (-NONE- *-1))
              (VP (TO to)
                  (VP (VB take)
                      (NP (PRP it))
                      (PRT (RB back))))))))

Predicate argument structure:
promise(they, Mr. Trotter, take_back(they, it))
```

The dependency analysis for *persuaded* and *promised* do not make such a distinction. The dependency analysis for the two sentences in the preceding pair of examples would be identical, as follows.

```
1 They      PRP 2 SBJ          1 They      PRP 2 SBJ
2 persuaded VBD 0 ROOT         2 promised VBD 0 ROOT
3 Mr.       NNP 4 NMOD         3 Mr.       NNP 4 NMOD
4 Trotter   NNP 2 IOBJ         4 Trotter   NNP 2 IOBJ
5 to        TO  6 VMOD         5 to        TO  6 VMOD
6 take      VB  2 OBJ          6 take      VB  2 OBJ
7 it        PRP 6 OBJ          7 it        PRP 6 OBJ
8 back      RB  6 PRT          8 back      RB  6 PRT
9 .         .   2 P            9 .         .   2 P
```

However, while pointing out these differences in annotation philosophy between dependency and phrase structure treebanks, it is important to note that most statistical parsers that are trained using phrase structure treebanks typically ignore these differences. The rich annotation of logical subjects, null elements, and so on, are all but ignored in modern

statistical parsers. There has been some interest in recovering the null elements that were originally annotated in the Penn Treebank for English and discarded during training a statistical parser. For instance, a postprocessing step described by Johnson [17] recovers empty elements and identifies their antecedents. The evaluation scheme presented by Rimell, Clark, and Steedman [18] shows how to compare different parsers in terms of the recovery of the underlying predicate-argument structure of each sentence of the type shown in the previous examples.

In different treebanks for the same language, or in treebanks for different languages, there might be many differences in the phrase structure annotation. The differences could be in the choice of symbols and what they represent. In the following example from the Chinese treebank, the symbol *IP* is used instead of *S*, which reflects a move from the English Penn Treebank's predominantly transformational grammar–based phrase structure to government binding (GB)–based phrase structures. The differences could also be related to specific syntactic construction. In the following example, the possessive marker 的 is given a particular analysis that results in a fairly complex structural analysis with several null elements for 新 的, with a null WHNP even though Chinese has no relative pronouns. This structure is motivated by the perceived need for a uniform and consistent phrase structure for clauses and clause-like constituents throughout the treebank. Such differences mean that a phrase structure parser developed initially for English parsing and trained on the English treebank may not be easily portable to another language even though a phrase structure treebank exists for that language. Levy and Manning [19] discuss the many challenges in taking a CFG-based parser initially developed for English parsing and adapting it to Chinese parsing by training on the Chinese phrase structure treebank.

```
(IP (NP-SBJ (NP (NN 结算/settlement and sale)
                (NN 制度/system))
            (CC 和/and)
            (NP (CP (WHNP-2 (-NONE- *OP*))
                    (CP (IP (NP-SBJ (-NONE- *T*-2))
                            (VP (VA 新/new)))
                        (DEC 的)))
                (NP (NN 核销/verification and cancellation)
                    (NN 制度/system))))
    (VP (PP-LOC (P 在/in)
                (NP-PN (NR 西藏/Tibet)))
        (ADVP (AD 全面/fully))
        (VP (VV 实施/operating))))

English translation:
A (foreign exchange) settlement and sale system and a verification and
cancellation system that is newly created is fully operational in Tibet.
```

3.4 Parsing Algorithms

Given an input sentence, a parser produces an output analysis of that sentence, which we now assume is the analysis that is consistent with a treebank used to train a parser. Treebank

parsers do not need to have an explicit grammar, but to make the explanation of parsing algorithms simpler we first consider parsing algorithms that assume the existence of a CFG.

Consider the following simple CFG G that can be used to derive strings such as *a and b or c* from the start symbol `N`.

```
N -> N 'and' N
N -> N 'or' N
N -> 'a' | 'b' | 'c'
```

An important concept for parsing is a **derivation**. For the input string *a and b or c*, the following sequence of actions separated by the => symbol represents a sequence of steps called a derivation:

```
N
=> N 'or' N
=> N 'or c'
=> N 'and' N 'or c'
=> N 'and b or c'
=> 'a and b or c'
```

In this derivation, each line is called a **sentential form**. Furthermore, each line of the derivation applies a rule from the CFG to show that the input can, in fact, be derived from the start symbol `N`. In the above derivation, we restricted ourselves to only expand on the rightmost nonterminal in each sentential form. This method is called the **rightmost derivation** of the input using a CFG. An interesting property of a rightmost derivation is revealed if we arrange the derivation in reverse order:

```
'a and b or c'
=> N 'and b or c'       # use rule N -> a
=> N 'and' N 'or c'     # use rule N -> b
=> N 'or c'             # use rule N -> N and N
=> N 'or' N             # use rule N -> c
=> N                    # use rule N -> N or N
```

This derivation sequence exactly corresponds to the construction of the following parse tree from left to right, one symbol at a time.

```
(N (N (N a)
      and
      (N b))
   or
   (N c))
```

However, a unique derivation sequence is not guaranteed. There can be many different derivations, and as we saw before, the number of derivations can be exponential in the input length. For example, there is another rightmost derivation that results in the following parse tree:

```
(N (N a)
   and
   (N (N b)
      or
```

```
  (N c)))
'a and b or c'
=> N 'and b or c'          # use rule N -> a
=> N 'and' N 'or c'        # use rule N -> b
=> N 'and' N 'or' N        # use rule N -> c
=> N 'and' N               # use rule N -> N or N
=> N                       # use rule N -> N and N
```

3.4.1 Shift-Reduce Parsing

To build a parser, we need to create an algorithm that can perform the steps in the preceding rightmost derivation for any grammar and for any input string. Every CFG turns out to have an automaton that is equivalent to it, called a pushdown automaton (just like regular expressions can be converted to finite-state automata). A pushdown automaton is simply a finite-state automaton with some additional memory in the form of a stack (or pushdown). This is a limited amount of memory because only the top of the stack is used by the machine. This provides an algorithm for parsing that is general for any given CFG and input string. The algorithm is called **shift-reduce** parsing, which uses two data structures, a buffer for input symbols and a stack for storing CFG symbols, and is defined as follows:

1. Start with an empty stack and the buffer containing the input string.
2. Exit with success if the top of the stack contains the start symbol of the grammar and if the buffer is empty.
3. Choose between the following two steps (if the choice is ambiguous, choose one based on an oracle):
   - Shift a symbol from the buffer onto the stack.
   - If the top k symbols of the stack are $\alpha_1 \dots \alpha_k$, which corresponds to the right-hand side of a CFG rule $A \rightarrow \alpha_1 \dots \alpha_k$, then replace the top k symbols with the left-hand side nonterminal A.
4. Exit with failure if no action can be taken in previous step.
5. Else, go to step 2.

For the CFG G shown earlier in this section and for the input *a and b or c*, we show the individual steps in the shift-reduce parsing algorithm in Figure 3–4.

The same algorithm can also be applied for dependency parsing, as can be seen by the example in Figure 3-5 of using a shift-reduce parser for dependency parsing. At each step, the parser has a choice: either shift a new token into the stack or combine the top two elements of the stack with a head → dependent link or a dependent ← head link. When using the shift-reduce algorithm in a statistical dependency parser, it helps to combine a shift and reduce action when possible. Other variants that vary the link between parser actions and statistical decisions are discussed by Nivre [20].

Parse Tree	Stack	Input	Action
		a and b or c	Init
a	a	and b or c	shift a
(N a)	N	and b or c	reduce N -> a
(N a) and	N and	b or c	shift and
(N a) and b	N and b	or c	shift b
(N a) and (N b)	N and N	or c	reduce N -> b
(N (N a) and (N b))	N	or c	reduce N -> a
(N (N a) and (N b)) or	N or	c	shift or
(N (N a) and (N b)) or c	N or c		shift c
(N (N a) and (N b)) or (N c)	N or N		reduce N -> c
(N (N (N a) and (N b)) or (N c))	N		reduce N -> N or N
(N (N (N a) and (N b)) or (N c))	N		Accept!

Figure 3–4: The individual steps of the shift-reduce parsing algorithm for the input *a and b or c* for the grammar G defined at the beginning of this section

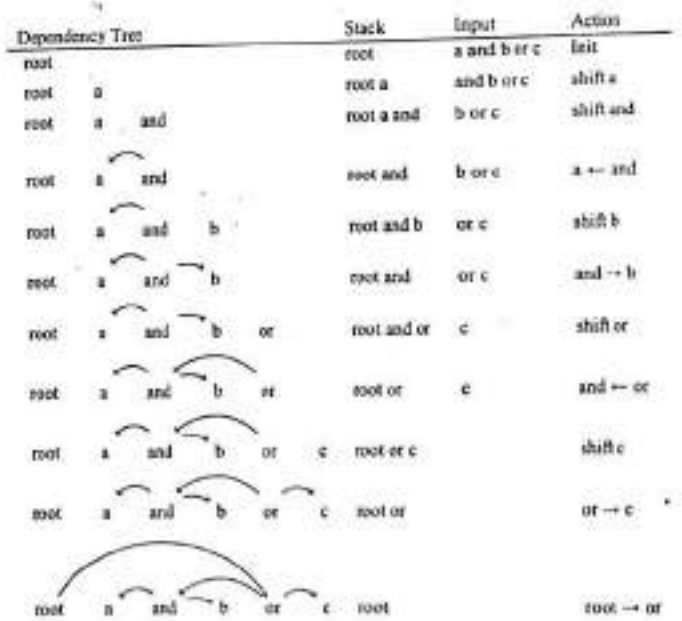


Figure 3–5: Steps of the shift-reduce parsing algorithm for dependency parsing

3.4.2 Hypergraphs and Chart Parsing

Shift-reduce parsing allows a linear time parse but requires access to an oracle. For general CFGs in the worst case, such a parser might have to resort to backtracking, which means reparsing the input, which leads to a time that is exponential in the grammar size in the worst case. On the other hand, CFGs do have a worst-case parsing algorithm that can run in $O(n^3)$ where n is the length of the input. Variants of this algorithm are often used in statistical parsers that attempt to search the space of possible parse trees without the limitation of purely left-to-right parsing.

Our example CFG G:

```
N -> N 'and' N
N -> N 'or' N
N -> 'a' | 'b' | 'c'
```

is rewritten into a new CFG G_ϵ where the right-hand side only contains up to two nonterminals. This is done by introducing two new nonterminals, N^ and Nv:

```
N -> N N^
N^ -> 'and' N
N -> N Nv
Nv -> 'or' N
N -> 'a' | 'b' | 'c'
```

A key insight into this family of parsing algorithms is that we can specialize the above CFG G_ϵ to a particular input string by creating a new CFG that represents a compact encoding of all possible parse trees that are valid in grammar G_ϵ for this particular input sentence. For example, for input string *a and b or c*, this new CFG G_f that represents the forest of parse trees is shown below. Imagine that the input string is broken up into spans *0 a 1 and 2 b 3 or 4 c 5* so that *a* is span 0,1 and the string *b or c* is the span 2,5 in this string. The nonterminals in this forest grammar G_ϵ include the span information. The different parse trees that can be generated using this grammar are the valid parse trees for the input sentence.

```
N[0,5] -> N[0,1] N^[1,5]
N[0,3] -> N[0,1] N^[1,3]
N^[1,3] -> 'and'[1,2] N[2,3]
N^[1,5] -> 'and'[1,2] N[2,5]
N[0,5] -> N[0,3] Nv[3,5]
N[2,5] -> N[2,3] Nv[3,5]
Nv[3,5] -> 'or'[3,4] N[4,5]
N[0,1] -> 'a'[0,1]
N[2,3] -> 'b'[2,3]
N[4,5] -> 'c'[4,5]
```

In this view, a parsing algorithm is defined as taking as input a CFG and an input string and producing a specialized CFG that is a compact representation of all legal parses for the input (see Figure 3-6). A parser has to create all the valid specialized rules or alternatively

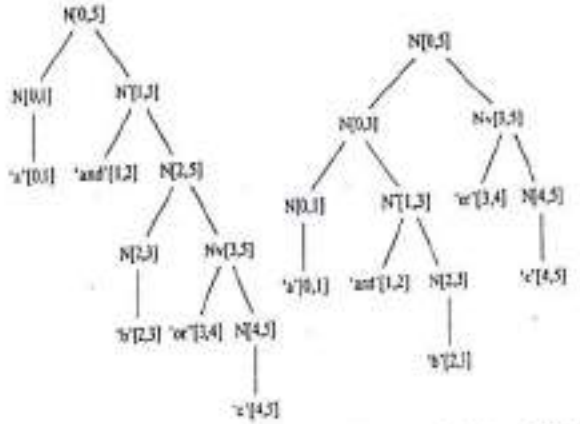


Figure 3-6: Parse trees embedded in the specialized CFG for a particular input string. The nodes with the same label, such as N[0,5], N[0,1], and [1,2], N[2,3], and Nv[3,5], can be merged to form a hypergraph representation of all parses for the input.

create a path from the start symbol nonterminal that spans the entire string to the leaf nodes that are the input tokens.

Let us examine the steps the parser has to take to construct a specialized CFG. First let us consider the rules that generate only lexical items:

```
N[0,1] -> 'a'[0,1]
N[2,3] -> 'b'[2,3]
N[4,5] -> 'c'[4,5]
```

These rules can be constructed by simply checking for the existence of rules of the type $N \rightarrow x$ for any input token x and creating a specialized rule for token x. In pseudocode this step can be written as follows:

```
for i = 0...n do
  if N → x with score s for any x spanning i,i+1 exists then
    add specialized rule N[i,i+1] → x[i,i+1] with score s
    written as N[i,i+1] : s
  end if
end for
```

The next step recursively creates new specialized rules based on previously created specialized rules. If we see that $Y[i,k]$ and $Z[k,j]$ exist as left-hand sides of previously created specialized rules, then if there is a rule in the CFG of the type $X \rightarrow Y\,Z$, we can infer that there should be a new specialized rule $X[i,j] \rightarrow Y[i,k]\,Z[k,j]$. Each nonterminal span is assigned a score s, $X[i,j] : s$. Only the highest scoring span for each nonterminal needs to be retained, so $X[i,j] = \max_s X[i,j] : s$.

```
for j = 2...n do
  for i = j - 1...0 do
    for k = i + 1...j do
      if Y[i,k] : s1 and Z[k,j] : s2 are in the specialized grammar then
        if X → YZ with score s exists in the original grammar then
          add specialized rule X[i,j] → Y[i,k]Z[k,j] with score s + s1 + s2
          keep only the highest scoring rule: X[i,j] → α
        end if
      end if
    end for
  end for
end for
```

This is called the CKY algorithm (named after Cocke, Kasami, and Younger who all discovered it independently). It considers every span of every length and splits up that span in every possible way to see if a CFG rule can be used to derive the span. Eventually we are guaranteed to find a rule that spans the entire input string if such a rule exists. Examining the loop structure of the algorithm shows that it takes n^3 time for an input of size n. However, exhaustively listing all trees from the specialized CFG will take exponential time in the worst case (by the reasoning we already covered about the number of trees possible in the worst case for CFGs). However, picking the most likely tree by using supervised machine learning will take no more than n^3 time.

Notice that for each span i, j and for each nonterminal X, we only keep the highest scoring way to reach $X[i, j]$. Thus, starting from the start symbol that spans the entire string $S[0][n]$ with the highest score, we can create the best-scoring parse tree by expanding the right-hand side of $S[0][n]$ and continue this process recursively to the terminal symbols. This gives us the best-scoring parse tree for a given sentence.

In a probabilistic setting, where the scores can be interpreted as log probabilities, this is called the Viterbi-best parse. Each cell contains the log probability of deriving the string $w[i, j]$ starting with nonterminal X, which can be written as $\Pr(X \Rightarrow^* w[i, j])$. Notice that the utility of a nonterminal X at a particular span i, j depends on reaching the start symbol S, which is captured by the outside probability $\Pr(S \Rightarrow^* w[0, i-1]Xw[j+1, N])$. Ideally we would like to use both the inside and the outside probabilities to compute the utility of each rule starting with $X[i, j]$.

There are ways to speed up the parser by throwing away less likely parts of the search space. For instance, we can compare the score of $X[i, j]$ with the score of the current highest scoring entry $Y[i, j]$ and throw away any rule starting with $X[i, j]$ if it is extremely unlikely compared to $Y[i, j]$. This could lead to a search error (where the highest scoring tree is missed), but it is very unlikely to happen, and we can trade off accuracy for a much faster parsing time. This technique is called beam thresholding. We can additionally augment this by looking at global ways to threshold entries. For instance, a rule starting with $X[i, j]$ cannot be viable if it does not have neighboring rules that can combine with it. Such a technique is called global thresholding. If we have a very complex set of nonterminals, such as the ones shown in Figure 3-7, then we could parse first with

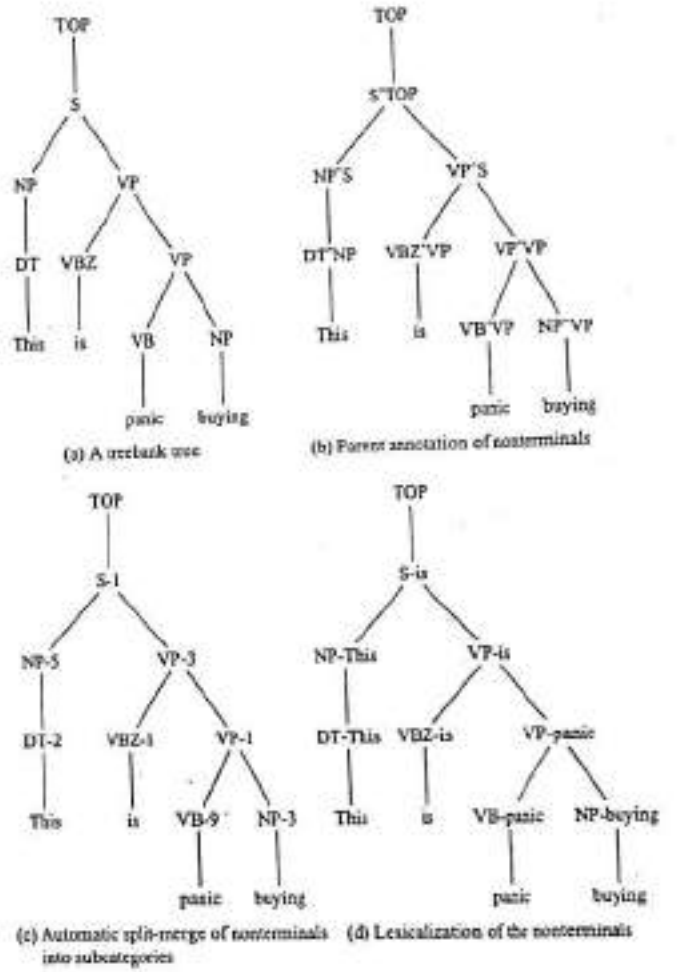


Figure 3-7: A treebank tree that is transformed in order to remove independence assumptions that hurt parsing performance: (a) is the original treebank tree, and a PCFG is easily extracted from that data; (b) grafts the parent node label onto each node label; (c) uses unsupervised learning to create subcategories for each nonterminal; and (d) uses lexical items percolated through the tree via lexicalizing nonterminals

a coarser nonterminal to prune the finer grained nonterminal (e.g., using simply a VP nonterminal instead of a VP^S nonterminal), and then use the scores associated with the coarse $VP[i, j]$ nonterminal to prune the finer grained nonterminals for the same span.

This approach is called coarse to fine parsing and it is particularly useful because the outside probability of the coarse step can be used in addition to the inside probability for more effective pruning. These three techniques, beam thresholding, global thresholding, and coarse to fine pruning, are all covered in a paper by Joshua Goodman on parsing PCFGs [21].

The parser can be sped up even further by using A^* search rather than the exhaustive search used in the algorithm shown previously [22]. A rich choice of heuristics can drive A^* search, which can be used to provide faster observed times for parsing while the asymptotic worst-case complexity remains the same as the CKY algorithm.

For projective dependency parsing, the same algorithm can be used by creating a CFG that produces dependency parses (as we showed in an earlier section). However, for dependency parsing, the above loop takes worst-case n^5 time because each Y and Z is lexicalized and in the worst case there can be n different Y nonterminals and n different Z nonterminals, giving us n^2 different possible combinations in the innermost loop of the CKY algorithm.

However, for dependency parsing, Eisner [23] observed that rather than using nonterminals augmented with words, it would be advantageous to represent compactly the set of different dependency trees for each span of the input string. The idea is to collect the left and the right dependents of each head independently and combine them at a later stage. This leads to the notion of a **split-head** where the head word is split into two: one for the left and the other for the right dependents. In addition to the head word, in each item we store for each span, we store whether the head is gathering left or right dependents and whether the item is complete (a complete item cannot be extended with more dependents). This provides an n^3 worst-case dependency parsing algorithm. This also reduces the number of intermediate states considered by not allowing any interleaving of left and right dependencies, unlike the CKY parser for dependency parsing.

The following pseudocode (from Ryan McDonald's thesis [24]) describes the Eisner algorithm in full detail. The spans are stored in a chart data-structure C, e.g. $C[i][j]$ refers to the dependency analysis of span i, j. Incomplete spans are referred to as C^i, complete spans are C^c. Spans that are being grown toward the left (adding left dependencies only) are referred to as $C_{\leftarrow}$, and spans growing to the right are referred to as $C_{\rightarrow}$. For $C_{\leftarrow}[i][j]$ the head must be j, and for $C_{\rightarrow}[i][j]$ the head must be i.

```
Initialize: for s = 1..n chart C_d^c[s][s] = 0.0 for d ∈ {←, →} and c ∈ {i, c}
for k = 1...n do
  for s = 1...n do
    t = s + k
    break if t > n
    first: create incomplete items
    C_←^i[s][t] = max_{s≤r<t} C_→^c[s][r] + C_←^c[r+1][t] + s(t, s)
    C_→^i[s][t] = max_{s≤r<t} C_→^c[s][r] + C_←^c[r+1][t] + s(s, t)
    second: create complete items
    C_←^c[s][t] = max_{s≤r<t} C_←^c[s][r] + C_←^i[r][t]
    C_→^c[s][t] = max_{s≤r<t} C_→^i[s][r] + C_→^c[r][t]
  end for
end for
```

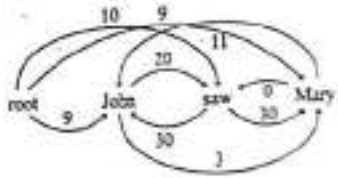

We assume a unique root node as the leftmost token (as before). The score of the best tree for the entire sentence is in $C_{\rightarrow}^c[1][n]$. In addition to running in $O(n^3)$, this algorithm can be extended to provide k-best parses with a complexity of $O(n^3 k \log k)$.

3.4.3 Minimum Spanning Trees and Dependency Parsing

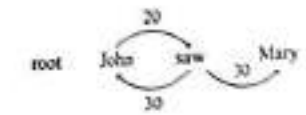
Finding the optimum branching in a directed graph is closely related to the problem of finding a minimum spanning tree in an undirected graph. The directed graph case is of interest because it corresponds to a dependency tree, which is always rooted and cannot have cycles. A prerequisite is that each potential dependency link between words should have a score. In NLP, the tradition is to use the term **minimum spanning tree** (MST) to refer to the optimum branching problem in directed graphs. In the case of parsing with a dependency treebank, we assume we have some model that can be used to provide such a score based on estimates of likelihood of each dependency link in the dependency tree. These scores can be used to find the MST, which is the highest scoring dependency tree. Because the linear order of the words in the input is not taken into account in the MST formulation, crossing or nonprojective dependencies can be recovered by such a parser. This can be an issue in languages that are predominantly projective, like English, but provide a natural way to recover the crossing dependencies in languages like Czech.

Rather than provide pseudocode for the MST algorithm for dependency parsing (which is provided in McDonald [24]), we show how the MST algorithm works using an example dependency parse using this algorithm.

Let us consider the following fully connected graph for the input sentence *John saw Mary*. The edges have weights based on some scoring function on edges (these scores come from various features that are computed on the edge, as explained in the next section).

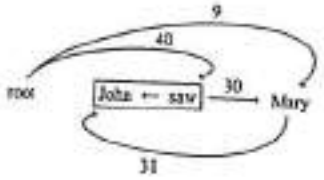


The first step is to find the highest scoring incoming edge. If this step results in a tree, then we report this as the parse because it would have to be an MST. In this example, however, after we pick only the highest scoring incoming edges from the graph, we do have a cycle.

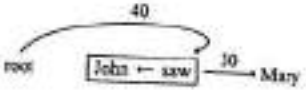


We can contract the cycle into a single node, and we recalculate the edge weights. When we compute the edge weights from each node to that contracted node, we also have to keep

track of which component of the merged node is the one with maximum weight. For example, in the preceding graph, we compare the incoming edge: *root* → *saw* → *John*: wt = 40 with *root* → *John* → *saw*: wt = 29 and compare the incoming edge: *Mary* → *saw* → *John*: wt = 30 with *Mary* → *John* → *saw*: wt = 31 to obtain the ones with maximum weight shown below:



We now run the MST algorithm recursively on this graph, which means finding the graph with the best incoming edges to each word. In this case it means comparing *root* → *Mary* → *John-saw*: wt = 9 + 31 with *root* → *John-saw* → *Mary*: wt = 40 + 30 which results in the following graph:



Unwinding the recursive step provides us with the MST that is the highest scoring dependency parse of the input:

3.5 Models for Ambiguity Resolution in Parsing

In this section we focus on the modeling aspect of parsing: how to design features and ways to resolve ambiguity in parsing. Using these models to parse efficiently is covered in Section 3.4 when we discuss parsing algorithms. The algorithms from that section are used by the models described in this section to find the highest scoring parse tree or dependency analysis and sometimes to train the models as well.

3.5.1 Probabilistic Context-Free Grammars

Consider the ambiguity problem we discussed earlier, where we would like to choose between the following ambiguous parses for the sentence *John bought a shirt with pockets*.


```
(S (NP John)
   (VP (VP (V bought)
           (NP (D a)
               (N shirt)))
       (PP (P with)
           (NP pockets))))
```

```
(S (NP John)
   (VP (V bought)
       (NP (NP (D a)
               (N shirt))
           (PP (P with)
               (NP pockets)))))
```

We want to provide a model that matches the intuition that the second tree above is preferred over the first. The parses can be thought of as ambiguous (leftmost or rightmost) derivations of the following CFG.

```
S -> NP VP
NP -> 'John' | 'pockets' | D N | NP PP
VP -> V NP | VP PP
V -> 'bought'
D -> 'a'
N -> 'shirt'
PP -> P NP
P -> 'with'
```

We can add scores or probabilities to the rules in this CFG in order to provide a score or probability for each derivation. The probability of a derivation is the sum of scores or product of probabilities of all the CFG rules used in that derivation. Because scores can be viewed simply as log probabilities, we use the term probabilistic context-free grammar (PCFG) when scores or probabilities are assigned to CFG rules. To make sure the probability of the set of trees generated by a PCFG is well defined, we assign probabilities to the CFG rules such that for a rule $N \rightarrow \alpha$, the probability is $P(N \rightarrow \alpha \mid N)$; that is, each rule probability is conditioned on the left-hand side of the rule. This means that during the context-free expansion of a nonterminal, the probability is distributed among all the expansions of the nonterminals. In other words,

$$1 = \sum_{\alpha} P(N \rightarrow \alpha)$$

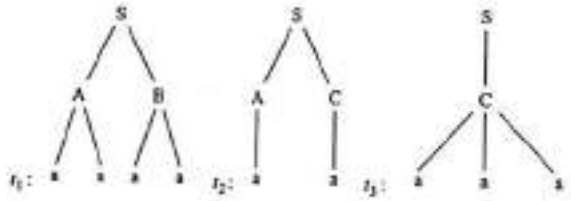
So in our example, we can assign probabilities to rules in the CFG to obtain the result we want—the more plausible parse gets the higher probability.

```
S -> NP VP (1.0)
NP -> 'John' (0.1) | 'pockets' (0.1) | D N (0.3) | NP PP (0.5)
VP -> V NP (0.9) | VP PP (0.1)
V -> 'bought' (1.0)
D -> 'a' (1.0)
N -> 'shirt' (1.0)
PP -> P NP (1.0)
P -> 'with' (1.0)
```

From these rule probabilities, the only deciding factor for choosing between the two parses for *John bought a shirt with pockets* is the two rules `NP -> NP PP` and `VP -> VP PP` because all the other rules in one parse also occur in the other. Because the probability

for NP -> NP PP is set higher in the preceding PCFG, the most plausible analysis gets the higher probability.

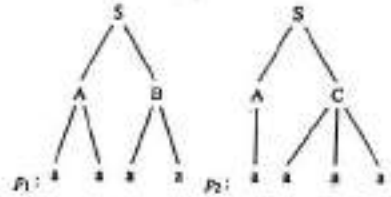
The rule probabilities can be derived from a treebank, as we can observe from the following example. Consider a treebank with three trees t_1, t_2, and t_3:



If we assume that tree t_1 occurred 10 times in the treebank, t_2 occurred 20 times, and t_3 occurred 50 times, then the PCFG we obtain from this treebank would be:

$$\frac{10}{10+20+50} = 0.125 \quad S \rightarrow A\ B$$
$$\frac{20}{10+20+50} = 0.25 \quad S \rightarrow A\ C$$
$$\frac{50}{10+20+50} = 0.625 \quad S \rightarrow C$$
$$\frac{10}{10+20} = 0.334 \quad A \rightarrow a\ a$$
$$\frac{20}{10+20} = 0.667 \quad A \rightarrow a$$
$$\frac{20}{20+50} = 0.285 \quad B \rightarrow a\ a$$
$$\frac{50}{20+50} = 0.714 \quad C \rightarrow a\ a\ a$$

For input a a a a there are two parses using the above PCFG: The probability for



$p_1 = 0.125 \cdot 0.334 \cdot 0.285 = 0.01189$ and for $p_2 = 0.25 \cdot 0.667 \cdot 0.714 = 0.119$. The parse tree p_2 is the most likely tree for that input. The most likely parse tree does not even occur in the treebank! And the reason for this is the context-free nature of PCFGs, where a nonterminal can be expanded by any rule with that nonterminal in the left-hand side. To make appropriate independence assumptions, the usual approach in a statistical parser is to augment the node labels in order to avoid bad independence assumptions.

The Penn Treebank contains trees like those in Figure 3–7(a). The first approach was to remove some independence assumptions by annotating each nonterminal with the label of its parent [25]. The second approach [26] is to automatically learn those nonterminal splits

by using an unsupervised learning algorithm (split-merge over trees using the expectation-maximization (EM) algorithm). The third approach [27] is to lexicalize the nonterminals, which leads to a better model because the words are included in the decision to attach an adjunct.

When each nonterminal is lexicalized, then the standard parsing algorithms have to be modified to work with heavily lexicalized rules. Lexicalized nonterminals have to be treated carefully in the model because sparsity is an issue with lexicalization. For lexicalized nonterminal PCFG, the unfolding history is created head outward: first predicting the head and producing the left siblings of the head and then the right siblings.

An alternative to finding the most likely tree for a given input is to find the most likely set of constituents. The idea is to find the best tree not with highest score but with the highest number of correct constituents. As shown in Goodman [28], the CKY algorithm (which we defined in §3.4.2) can be used to find the tree with the highest number of correct constituents by replacing the scoring function for each $X[i,j]$ to be the product of the inside and outside probability rather than just using the inside probability. This technique is often called max-rule parsing and can produce trees that are not valid PCFG parse trees, similar to the example we covered earlier. Max-rule parsing explicitly maximizes the recall at a per-constituent level and hence often gives higher recall scores in parser evaluations.

3.5.2 Generative Models for Parsing

To find the most plausible parse tree, the parser has to choose between the possible derivations each of which can be represented as a sequence of decisions. Let each derivation $D = d_1, \ldots, d_n$, which is the sequence of decisions used to build the parse tree. Then for input sentence x, the output parse tree y is defined by the sequence of steps in the derivation. We can introduce a probability for each derivation:

$$P(x,y) = P(d_1, \ldots, d_n) = \prod_{i=1}^{n} P(d_i \mid d_1, \ldots, d_{i-1})$$

The conditioning context in the probability $P(d_i \mid d_1, \ldots, d_{i-1})$ is called the history and corresponds to a partially built parse tree (as defined by the derivation sequence). We make a simplifying assumption that keeps the conditioning context to a finite set by grouping the histories into equivalence classes using a function Φ.

$$P(d_1, \ldots, d_n) = \prod_{i=1}^{n} P(d_i \mid \Phi(d_1, \ldots, d_{i-1}))$$

Using Φ, each history $H_i = d_1, \ldots, d_{i-1}$ for all x, y is mapped to some fixed finite set of feature functions of the history $\phi_1(H_i), \ldots, \phi_k(H_i)$. In terms of these k feature functions:

$$P(d_1, \ldots, d_n) = \prod_{i=1}^{n} P(d_i \mid \phi_1(H_i), \ldots, \phi_k(H_i))$$

However, the definition of PCFGs means that various other rule probabilities must be adjusted to obtain the right scoring of parses. Also, the independence assumptions in PCFG,

which are dictated by the underlying CFG, often lead to bad models that cannot use information vital to the decision of rule scores, resulting in high-scoring plausible parses. We would like to model such ambiguities using arbitrary features of the parse tree. Discriminative methods provide us with such a class of models.

3.5.3 Discriminative Models for Parsing

Collins [29] extends the ideas from Freund and Schapire [30] to create a simple notation and framework that describes various discriminative approaches to learning for parsing (and also chunking or tagging). This framework is called a **global linear model** [29]. Let $\mathbf{x}$ be a set of inputs, and $\mathbf{y}$ be a set of possible outputs that can be a sequence of POS tags or a parse tree or a dependency analysis.

- Each $x \in \mathbf{x}$ and $y \in \mathbf{y}$ is mapped to a d-dimensional feature vector $\Phi(x,y)$, with each dimension being a real number, summarizing partial information contained in (x,y).
- A weight parameter vector $\mathbf{w} \in \Re^d$ assigns a weight to each feature in $\Phi(x,y)$, representing the importance of that feature. The value of $\Phi(x,y) \cdot \mathbf{w}$ is the score of (x,y). The higher the score, the more plausible it is that y is the output for x.
- The function $GEN(x)$ generates the set of possible outputs y for a given x.

Having $\Phi(x,y)$, $\mathbf{w}$, and $GEN(x)$ specified, we would like to choose the highest scoring candidate y^* from $GEN(x)$ as the most plausible output. That is,

$$F(x) = \underset{y \in GEN(x)}{\operatorname{argmax}}\, p(y \mid x, \mathbf{w})$$

where $F(x)$ returns the highest scoring output y^* from $GEN(x)$. A conditional random field (CRF) [31] defines the conditional probability as a linear score for each candidate y and a *global* normalization term:

$$\log p(y \mid x, \mathbf{w}) = \Phi(x,y) \cdot \mathbf{w} - \log \sum_{y' \in GEN(x)} \exp(\Phi(x,y') \cdot \mathbf{w})$$

A simpler global linear model that ignores the normalization term is:

$$F(x) = \underset{y \in GEN(x)}{\operatorname{argmax}}\, \Phi(x,y) \cdot \mathbf{w}$$

Many experimental results in parsing have shown that the simpler global linear model that ignores the normalization term (and is thus much faster to train) often provides the same accuracy when compared to the more expensively trained normalized models.

A perceptron [32] was originally introduced as a single-layered neural network. It is trained using online learning, that is, processing examples one at a time, during which it adjusts a weight parameter vector that can then be applied on input data to produce the corresponding output. The weight adjustment process awards features appearing in the truth and penalizes features not contained in the truth. After the update, the perceptron ensures that the current weight parameter vector is able to correctly classify the present training example.

Algorithm 3–1 The Original Perceptron Learning Algorithm

Inputs: Training Data $\langle (x_1, y_1), \ldots, (x_m, y_m) \rangle$; number of iterations T
Initialization: Set $\mathbf{w} = 0$
Algorithm:

```
for t = 1, ..., T do
  for i = 1, ..., m do
    Calculate y_i', where y_i' = argmax_{y ∈ GEN(x_i)} Φ(x_i, y) · w
    if y_i' ≠ y_i then
      w = w + Φ(x_i, y_i) − Φ(x_i, y_i')
    end if
  end for
end forOutput: The updated weight parameter vector w
```

Suppose we have m examples in the training set. The original perceptron learning algorithm [32] is shown in Algorithm 3–1.

The weight parameter vector $\mathbf{w}$ is initialized to 0. Then the algorithm iterates through those m training examples. For each example x, it generates a set of candidates $GEN(x)$, and picks the most plausible candidate, which has the highest score according to the current $\mathbf{w}$. After that, the algorithm compares the selected candidate with the truth, and if they are different from each other, $\mathbf{w}$ is updated by increasing the weight values for features appearing in the truth and by decreasing the weight values for features appearing in this top candidate. If the training data is linearly separable, meaning that it can be discriminated by a function that is a linear combination of features, the learning is proven to converge in a finite number of iterations [30].

This original perceptron learning algorithm is simple to understand and to analyze. However, the incremental weight updating suffers from overfitting, which tends to classify the training data better but at the cost of classifying the unseen data worse. Also, the algorithm is not capable of dealing with training data that is linearly inseparable.

Freund and Schapire [30] proposed a variant of the perceptron learning approach, the voted perceptron algorithm. Instead of storing and updating parameter values inside one weight vector, its learning process keeps track of all intermediate weight vectors, and these intermediate vectors are used in the classification phase to vote for the answer. The intuition is that good prediction vectors tend to survive for a long time and thus have larger weight in the vote. Algorithm 3–2 shows the voted perceptron training and prediction phases from Freund and Schapire, with slightly modified representation.

The voted perceptron keeps a count c_i to record the number of times a particular weight parameter vector $(\mathbf{w}_i, c_i)$ survives in the training. For a training example, if its selected top candidate is different from the truth, a new count c_{i+1}, being initialized to 1, is used, and an updated weight vector $(\mathbf{w}_{i+1}, c_{i+1})$ is produced; meanwhile, the original c_i and weight vector $(\mathbf{w}_i, c_i)$ are stored.

Compared with the original perceptron, the voted perceptron is more stable due to maintaining the list of intermediate weight vectors for voting. Nevertheless, to store those weight vectors is space inefficient. Also, the weight calculation, using all intermediate weight parameter vectors during the prediction phase, is time consuming.

Algorithm 3–2 The Voted Perceptron Algorithm

Training Phase

Input: Training data $((x_1, y_1), \ldots, (x_m, y_m))$, number of iterations T
Initialization: $k = 0$, $w_0 = 0$, $c_1 = 0$
Algorithm:

```
for t = 1, ..., T do
  for i = 1, ..., m do
    Calculate y_i', where y_i' = argmax_{y ∈ GEN(x_i)} Φ(x_i, y) · w_k
    if y_i' = y_i then
      c_k = c_k + 1
    else
      w_{k+1} = w_k + Φ(x_i, y_i) − Φ(x_i, y_i')
      c_{k+1} = 1
      k = k + 1
    end if
  end for
end for
```

Output: A list of weight vectors $((w_1, c_1), \ldots, (w_k, c_k))$

Prediction Phase

Input: The list of weight vectors $((w_1, c_1), \ldots, (w_k, c_k))$, an unsegmented sentence x
Calculate:

$$y^* = \underset{y \in GEN(x)}{\operatorname{argmax}} \left(\sum_{i=1}^{k} c_i \Phi(x, y) \cdot w_i \right)$$

Output: The voted top ranked candidate y^*

The averaged perceptron algorithm [30] is an approximation to the voted perceptron that, on the other hand, maintains the stability of the voted perceptron algorithm but significantly reduces space and time complexities. In an averaged version, rather than using w, the averaged weight parameter vector γ over the m training examples is used for future predictions on unseen data:

$$\gamma = \frac{1}{mT} \sum_{i=1 \ldots m, t=1 \ldots T} w^{i,t}$$

In calculating γ, an accumulating parameter vector σ is maintained and updated using w for each training example. After the last iteration, $\sigma/(mT)$ produces the final parameter vector γ. The entire algorithm is shown in Algorithm 3–3.

When the number of features is large, it is expensive to calculate the total parameter σ for each training example. To further reduce the time complexity, Collins [33] proposed

Algorithm 3–3 The Averaged Perceptron Learning Algorithm

Inputs: Training Data $((x_1, y_1), \ldots, (x_m, y_m))$; number of iterations T
Initialization: Set $w = 0$, $\gamma = 0$, $\sigma = 0$
Algorithm:

```
for t = 1, ..., T do
  for i = 1, ..., m do
    Calculate y_i', where y_i' = argmax_{y ∈ GEN(x_i)} Φ(x_i, y) · w
    if y_i' ≠ y_i then
      w = w + Φ(x_i, y_i) − Φ(x_i, y_i')
    end if
    σ = σ + w
  end for
end for
```

Output: The averaged weight parameter vector $\gamma = \sigma/(mT)$

a lazy update procedure that avoids the expensive update to the entire weight vector in each iteration. After processing each training sentence, not all dimensions of σ are updated. Instead, an update vector τ is used to store the exact location (p,t) where each dimension of the averaged parameter vector was last updated, and only those dimensions corresponding to features appearing in the current sentence are updated. p represents the training example index where this particular feature was last updated, and t represents its corresponding iteration number. For the last example in the final iteration, each dimension of τ is updated, no matter whether the candidate output is correct or not. Algorithm 3–4 shows the averaged perceptron with lazy update procedure.

3.6 Multilingual Issues: What Is a Token?[2]

3.6.1 Tokenization, Case, and Encoding

So far we have assumed that in a grammar or in a treebank, the notion of a word, or more specifically that of a word token, is well defined. However, this definition might be well defined given a treebank or parser but variable across different parsers or treebanks. One example is the possessive marker and copula verb 's (a variant of *be*) in English. In English, a token is typically separated from other tokens by a space character. However, in a parser/treebank for English, a word like *today's* or *There's* is treated as two independent tokens, *today* and *'s*, or *There* and *'s*. As the Penn Treebank annotation guidelines point out,

2. This section describes morphological and tokenization issues as they relate to the task of syntactic parsing. For a thorough review of morphological processing, please see Chapter 1.

Algorithm 3-4 The Averaged Perceptron Learning Algorithm with Lazy Update Procedure

Inputs: Training Data $((x_1, y_1), \ldots, (x_m, y_m))$; number of iterations T
Initialization: Set $\mathbf{w} = 0$, $\gamma = 0$, $\sigma = 0$, $\tau = 0$
Algorithm:

```
for t = 1,...,T do
  for i = 1,...,m do
    Calculate y_i', where y_i' = argmax_{y ∈ GEN(x_i)} Φ(x_i, y) · w
    if t ≠ T or i ≠ m then
      if y_i' ≠ y_i then
        // Update active features in the current sentence
        for each dimension s in (Φ(x_i, y_i) − Φ(x_i, y_i')) do
          if s is a dimension in τ then
            // Include the total weight during the time
            // this feature remains inactive since last update
            σ_s = σ_s + w_s · (t · m + i − t_{τ_s} · m − i_{τ_s})
          end if
          // Also include the weight calculated from comparing y_i' with y_i
          w_s = w_s + Φ(x_i, y_i) − Φ(x_i, y_i')
          σ_s = σ_s + Φ(x_i, y_i) − Φ(x_i, y_i')
          // Record the location where the dimension s is updated
          τ_s = (i, t)
        end for
      end if
    else
      // To deal with the last sentence in the last iteration
      for each dimension s in τ do
        // Include the total weight during the time
        // each feature in τ remains inactive since last update
        σ_s = σ_s + w_s · (T · m + m − t_{τ_s} · m − i_{τ_s})
      end for
      // Update weights for features appearing in this last sentence
      if y_i' ≠ y_i then
        w = w + Φ(x_i, y_i) − Φ(x_i, y_i')
        σ = σ + Φ(x_i, y_i) − Φ(x_i, y_i')
      end if
    end if
  end for
end for
```

Output: The averaged weight parameter vector $\gamma = \sigma/(mT)$

the possessive marker can apply to some previous constituent and not just to the previous token:

```
(SP (NP (NP First)
        (PP of
            (NP America))
        's)
    operating results)
```

Similarly for the copula verb *'s*:

```
(S (NP-SBJ (EX There))
   (VP (VBZ 's)
       (NP-PRD (NP (NN nothing))
               (ADJP (RB very)
                     (JJ hot))))))
```

In some languages there are issues with uppercase and lowercase. It is tempting to lowercase all your treebank data and simply lowercase input texts for the parser. However, case can carry useful information. The token *Boring* if it is previously unseen in the training data might look like a progressive verb like *singing*, but the initial uppercase character makes it more likely to be a proper noun. However, depending on the type and amount of data available for training, some case information, such as selective lowercasing of sentence initial tokens, might have to be done to obtain reasonable estimates from the treebank. Low-count tokens can be replaced with patterns that retain case information; for example, if *Patagonia* appears only twice in the corpus, then it can be replaced with *Xxx* to reflect that new unknown words that match the same pattern can be treated as known under this pattern. The same is true for cases such as dates, times, IP addresses, and URLs.

For language scripts that are not encoded in ASCII, the different encodings need to be managed. In particular, the data the parser will be used on should be converted to the encoding that the treebank uses, or vice versa. There are often issues with the sentence terminator period (.), which in some text corpora might be an ASCII character but in others may be encoded in UTF-8. Some languages, such as Chinese, are encoded in different formats depending on the place where the text originates—for example, GB, BIG5, and UTF-8 are all encodings you might find for Chinese text.

These are trivial issues, algorithmically speaking, compared to writing a parser, but in practice these issues can be quite challenging and time consuming, and while a full discussion of these issues is beyond the scope of this chapter, it does need to be pointed out that thinking about tokenization, case, and encoding are prerequisites for anyone wishing to write a parser or to get one working for a new language.

3.6.2 Word Segmentation

The written form of many languages, including Chinese, lack marks identifying words. Given the Chinese text 北京大学生比赛, a plausible segmentation would be "北京 (Beijing) /大学生 (university students) /比赛 (competition) 'competition among university students in Beijing'. However, if 北京大学 is taken to mean Beijing University, the segmentation

for the character sequence might become 北京大学 (Beijing University) /生 (give birth to) /比赛 (competition) 'Beijing University gives birth to competition', which is less plausible.

Word segmentation is the process of demarcating blocks in a character sequence such that the produced output is composed of separated tokens and is meaningful. Only if we have identified each word in a sentence can POS tags (e.g., NNP or DT) then be assigned and the syntax tree for the whole sentence be built. In systems dealing with English or French, tokens are assumed to be already available because words have always been separated by spaces in these languages, whereas in Chinese, characters are written next to each other without marks identifying words.

Chinese word segmentation has a large community of researchers and has resulted in three SIGHAN bakeoffs [34, 35, 36]. Chapter 1 deals with these issues, so here we simply focus on the impact on parsing.

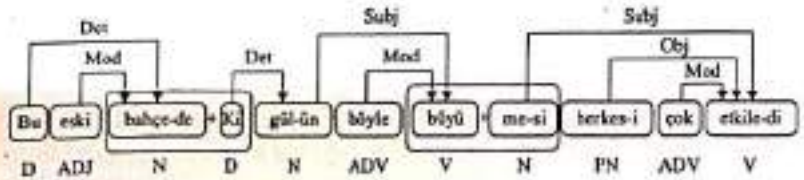
One interesting approach to Chinese parsing [37] is to parse the character sequence directly. The parser itself assigns word boundaries as part of the parsing process, where nonterminals in the tree that span a group of characters can be said to specify the word boundaries. However, this study found that immediate context was the most useful in predicting word boundaries. The global sentence context was not as useful in the discovery of word boundaries even though in some situations the ambiguity in word segmentation could need long-distance dependencies captured by the parse tree.

The use of a single best word segmentation from a word segmentation model creates a pipeline where the parser is unable to choose between different plausible segmentations. Using the result in Bar-Hillel, Perles, and Shamir [38], we know that a parser for CFGs can parse an input word lattice (which represents a finite language as a finite-state automata). The parser uses the states in the automata as indices, which generalizes the notion of indices into the input string. The input word lattice can be used to represent multiple segmentations of the Chinese input, and thus the parser can choose which of the ranked segmentation results lead to the most accurate parses.

3.6.3 Morphology

In many languages the notion of splitting up tokens using spaces is problematic because each word can contain several components, called morphemes, such that the meaning of a word can be thought of as composed of the combination of the meanings of the morphemes. A word must now be thought of as being decomposed into a stem combined with several morphemes.

For example, the following dependency analysis from the Turkish treebank shows that the syntactic dependencies need to be aware of the morphemes within words. In this example, morpheme boundaries within a word are shown using the + symbol. Morphemes, and not words, are used as heads and dependents.

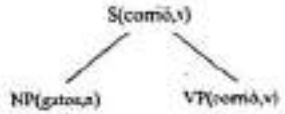


Turkish, Finnish, and other agglutinative languages have this property of entire clauses being combined with morphemes to create very complex words.

Inflectional languages like Czech and Russian are not as extreme but also suffer from the problem that many different morphemes are used to mark grammatical case, gender, and so on, and each type of morpheme can be orthogonal to the others (so they can independently co-occur). For instance, according to Hajic and Hladka [39], most adjectives in Czech can potentially form all four genders, all seven case markers, all three degrees of comparison, and can be either positive or negative in polarity. This results in 336 possible inflected word forms just for adjectives. In addition to a large number of word forms, each inflected word can be ambiguously segmented into morphemes with different analyses. Quite apart from syntactic ambiguity, the parser now also has to deal with morphological ambiguity.

To tackle the disambiguation problem for morphology, the problem of splitting a word into the most likely sequence of morphemes can be reduced to a (very complex) POS tagging task. Each word is to be tagged with a complex part of speech that encodes the various morphemes. For instance, a part of speech of `V—M-3----` can indicate that each word can have morphemes that inflect the word along 10 different dimensions, and in this case, the stem is a verb (V) with masculine gender (M) and in third person (3), and the other types of morphemes are assigned—which indicates that they do not occur in this analysis. The POS tagger has to produce this complex tag—and this is typically done by training separate subclassifiers for each component of the POS tag and combining the output for tagging each word [39]. The word itself is not split into morphemes, but each word is tagged with a POS tag that encodes a lot of information about the morphemes. This enriched tag set can be a rich source of features for a statistical parser for a highly inflected language.

Adding morphological awareness to statistical parsers can lead to improved accuracy. For example, in a language like Spanish [40], if we would like to create the following fragment of a parse tree, if the parser was morphologically aware then it might consider that plural nouns like *gatos* 'cats' are unlikely to modify the singular verb *corrió* 'ran'—even if it had not seen this particular bilexical dependency in the training data.



In discriminative models for statistical parsing (especially for dependency parsing), it has been fairly straightforward to include morphological information. Because discriminative models allow the inclusion of a large number of overlapping features, morphological information associated with the words can be used to build better statistical parsers by simply throwing them into the mix. As shown in the CoNLL 2007 shared task [13] and also in papers that explicitly looked at useful features per language in discriminative dependency parsing (e.g., [41, 42]), it was found that morphological information was helpful in improving the accuracy of statistical parsers especially for morphologically complex languages.

In phrase structure parsing, Cowan and Collins [40] provide a discriminative model for parsing Spanish that takes the k-best output from a generative model and uses morphological features to rerank the output. Various morphological information was included in

the POS tags and in the reranking model, and it was shown that as long as the tagset was not expanded to a very large size, accuracy improved with the addition of morphological information. In Sarkar and Han [43], a generative model for parsing Korean was augmented with morphological information. The probabilities for dependencies were interpolated between full word forms and various morphologically decomposed forms of the words. It was found that, in this particular model, using the stems rather than suffixes helped the parser generalize over morphologically complex word forms and helped improve parsing accuracy.

3.7 Summary

This chapter covered the syntactic analysis of natural language and how parsers can be built that can efficiently and accurately parse natural language and provide syntactic trees. We covered the necessity of using a data-driven approach to parsing of natural language and introduced the notion of a treebank that can provide training data for parsing language. Parsing is also interesting from the perspective of machine learning because it is a complex, structured prediction task where the output labels are not simple class labels but rather decomposed into smaller units, and the number of structured output labels is exponential in the size of the input. We covered the use of phrase structure parsing and dependency parsing as different ways to represent the syntactic analysis of natural language. This chapter covered parsing algorithms that can efficiently parse input sentences, as well as machine learning models for ambiguity resolution in parsing. There are many issues with writing parsers for different languages. We covered a few of the issues that arise when parsing languages that are quite different from English: issues such as tokenization, case, encoding issues, and word segmentation and morphology. In each case, we discussed how solutions to these issues can be incorporated into statistical parsers for these languages.

Acknowledgments

Thanks to the School of Informatics at the University of Edinburgh for hosting me during my sabbatical year away from Simon Fraser University. Most this chapter was written in Edinburgh.

Bibliography

[1] K. Knight and D. Marcu, "Summarization beyond sentence extraction: A probabilistic approach to sentence compression," *Artificial Intelligence*, vol. 139, no. 1, pp. 91–107, 2002.

[2] C. Callison-Burch, "Syntactic constraints on paraphrases extracted from parallel corpora," in *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pp. 196–205, 2008.

[3] M. Galley, M. Hopkins, K. Knight, and D. Marcu, "What's in a translation rule?," in *Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2004): Main Proceedings* (D. M. Susan Dumais and S. Roukos, eds.), pp. 273–280, 2004.

[4] S. Miller, H. Fox, L. Ramshaw, and R. Weischedel, "A novel use of statistical parsing to extract information from text," in *Proceedings of the 1st North American Chapter of the Association for Computational Linguistics Conference (NAACL 2000)*, pp. 226–233, 2000.

[5] R. Barzilay and K. R. McKeown, "Sentence fusion for multidocument news summarization," *Computational Linguistics*, vol. 31, no. 3, 2005.

[6] R. Barzilay, "Probabilistic approaches for modeling text structure and their application to text-to-text generation," in *Empirical Methods in Natural Language Generation* (E. Krahmer and M. Theune, eds.), Lecture Notes in Computer Science (LNAI 5790), Berlin: Springer, 2010.

[7] M. Lease, E. Charniak, and M. Johnson, "SS-11.4: Parsing and its applications for conversational speech," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'05) 5: V-961-V964*, 2005.

[8] P. Pantel and D. Lin, "Concept discovery from text," in *Proceedings of Conference on Computational Linguistics (COLING-02)*, pp. 577–583, 2002.

[9] A. Rudnicky, C. Bennett, A. Black, A. Chotimongkol, K. Lenzo, A. Oh, and R. Singh, "Task and domain specific modeling in the Carnegie-Mellon communicator system," in *Proceedings of the 6th International Conference on Spoken Language Processing (ICSLP)*, 2000. Paper G4-01.

[10] M. Sipser, *Introduction to the Theory of Computation*, 2nd ed., Boston: PWS Publishing Co., 2005.

[11] J. Nivre, "Two notions of parsing," in *A Finnish Computer Linguist: Kimmo Koskenniemi. Festschrift on the 60th Birthday* (A. Arppe, L. Carlson, O. Heinämäki, K. Lindén, M. Miestamo, J. Piitulainen, J. Tupakka, H. Westerlund, and A. Yli-Jyrä, eds.), pp. 111–120, Stanford, CA: CSLI Publications, 2005.

[12] L. Tesnière, *Éléments de syntaxe structurale*. Paris: C. Klincksieck, 1959.

[13] J. Nivre, J. Hall, S. Kübler, R. McDonald, J. Nilsson, S. Riedel, and D. Yuret, eds., *Proceedings of the CoNLL Shared Task Session of Empirical Methods on Natural Language Processing-Conference on Natural Language Learning 2007*, 2007.

[14] H. Gaifman, "Dependency systems and phrase structure systems," Tech. Rep. P-2315, The RAND Corporation, Santa Monica, CA, May 1961.

[15] S. Abney, "Dependency grammars and context-free grammars," manuscript presented at meeting of Linguistic Society of America, Jan. 1995.

[16] M. Marcus, G. Kim, M. A. Marcinkiewicz, R. Macintyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger, "The Penn Treebank: Annotating predicate argument structure," in *Proceedings of the ARPA Human Language Technology Workshop*, pp. 114–119, 1994.

Chapter 4

Semantic Parsing

Sameer Pradhan

Semantics by its dictionary definition is the study of meaning, and **parsing** is the examination of something in a minute way, that is, identifying and relating the pieces of information being parsed. When we put the two of these concepts together, we get **semantic parsing**, which, in the broadest sense of the phrase, is the process of identifying meaning **chunks** contained in an information signal in an attempt to transform it into some data structure that can be manipulated by a computer to perform higher level tasks. In our case, the information signal is human language text. Unfortunately, in the natural language processing community, the term semantic parsing is somewhat ambiguous. Over the years researchers have used it to represent various levels of granularity of meaning representation. Because semantics is such a vague term, it has been used to represent various depths of representations, from something as basic as identifying domain-specific relations between entities, to the more intermediate task of identifying the roles that various entities and artifacts play in an event, to converting a text to a series of specific logical expressions. Within the context of this chapter, we restrict its interpretation to the study of mapping naturally occurring text to some representation that is amenable to manipulation by computers for the purpose of achieving some goals, such as retrieving information, answering a question, populating a database, or taking an action.

4.1 Introduction

The holy grail of research in language understanding is the identification of a meaning representation that is detailed enough to allow reasoning systems to make deductions but, at the same time, is general enough that it can be used across many domains with little to no adaptation. It is not clear whether a final, low-level, detailed semantic representation covering various applications that use some form of language interface can be achieved or whether an ontology can be created that can capture the various granularities and aspects of meanings that are embodied in such a variety of applications—none has yet been created. Therefore, two compromise approaches have emerged in the natural language processing community for language understanding.

In the first approach, a specific, rich meaning representation is created for a limited domain for use by applications that are restricted to that domain, such as air travel reservations, football game simulations, or querying a geographic database. Systems are then

crafted to generate output from text in this rich, domain-specific meaning representation. In the second approach, a related set of intermediate meaning representations is created, going from low-level analysis to a mid-level analysis, and the bigger understanding task is divided into multiple, smaller pieces that are more manageable, such as word sense disambiguation followed by predicate-argument structure recognition. By dividing the problem up this way, each intermediate representation is only responsible for capturing a relatively small component of overall meaning, thereby making the task of defining and modeling each representation easier. Unlike the first approach, each meaning representation, while covering only a small part of the overall meaning, is not tied to a specific domain, and so the data and methods created for it are similarly general purpose.

Unfortunately, we do not yet have the holy grail in the form of a detailed overall representation that would at once be easily learnable and have high coverage across domains. So, in this chapter, we treat the world as though it has exactly two types of meaning representations: a domain-dependent, deeper representation and a set of relatively shallow but general-purpose, low-level, and intermediate representations. The task of producing the output of the first type is often called **deep semantic parsing**, and the task of producing the output of the second type is often called **shallow semantic parsing**. We discuss algorithms for producing both kinds of output.

Both of these approaches are fraught with issues; the first approach is so specific that porting to every new domain can require anywhere from a few modifications to almost reworking the solution from scratch. In other words, the reusability of the representation across domains is very limited. The problem with the latter approach is that it is extremely difficult to construct a general-purpose ontology and create symbols that are shallow enough to be learnable but detailed enough to be useful for all possible applications. Therefore, an application-specific translation layer between the more general representation and the more specific representation becomes necessary. However, this translational component can be relatively small compared to the total effort required to adapt a more specific representation to a new domain. None of this even begins to consider the implications of using such systems across different languages or the role played by the structure of different languages in affecting these meaning representations or their learnability. For these reasons, over the history of language processing, the community has generally moved away from the more detailed, deep, domain-dependent representations to the more shallow ones.

4.2 Semantic Interpretation

Semantic parsing can be considered as part of a larger process, **semantic interpretation**, which involves various components that together let us define a representation of a text that can be fed into a computer to allow further computational manipulations and search, which are prerequisite for any language understanding system or application. The following sections talk about some of the main components of this process.

We begin this discussion with the seminal work by Chomsky, *Syntactic Structures* [1], which introduced the concept of a transformational phrase structure grammar to provide an operational definition for the combinatorial formations of meaningful natural language

sentences by humans. Shortly after Chomsky's 1957 book, Katz and Fodor [2] published the first work treating semantics within the generative grammar paradigm. They found that Chomsky's transformational grammar was not a complete description of language because it did not account for meaning. In their 1963 paper "The Structure of a Semantic Theory," Katz and Fodor put forward what they thought were the properties a semantic theory should possess. A semantic theory should be able to:

1. Explain sentences having ambiguous meanings. For example, it should account for the fact that the word *bill* in the sentence *The bill is large* is ambiguous in the sense that it could represent money or the beak of a bird.
2. Resolve the ambiguities of words in context. For example, if the same sentence is extended to form *The bill is large but need not be paid*, then the theory should be able to disambiguate the monetary meaning of *bill*.
3. Identify meaningless but syntactically well-formed sentences, such as the famous example by Chomsky: *Colorless green ideas sleep furiously*.
4. Identify syntactically or transformationally unrelated paraphrases of a concept having the same semantic content.

In the following subsections we look at some requirements for achieving a semantic representation.

4.2.1 Structural Ambiguity

When we talk of structure, we generally refer to the syntactic structure of sentences. This is a sentence-level phenomenon and essentially means transforming a sentence into its underlying syntactic representation. Because syntax and semantics have such a strong interaction, most theories of semantic interpretation refer to the underlying syntactic representation. Conventionally, syntax has become the first stage of processing followed by various other stages in the process of semantic interpretation (see Chapter 3 for information on syntactic processing).

4.2.2 Word Sense

In any given language, it is almost certainly the case that the same word type, or word lemma, is used in different contexts and with different morphological variants to represent different entities or concepts in the world. For example, we use the word *nail* to represent a part of the human anatomy and also to represent the generally metallic object used to secure other objects. Humans are adept at identifying, through context, which sense of the word is intended by the author or speaker. Let's take the following four examples. The presence of words such as *hammer* and *hardware store* in sentences 1 and 2, and *clipped* and *manicure* in sentences 3 and 4, enable humans to easily disambiguate the sense in which *nail* is used:

1. He nailed the loose arm of the chair with a hammer.
2. He bought a box of nails from the hardware store.

3. He went to the beauty salon to get his *nails* clipped.
4. He went to get a manicure. His *nails* had grown very long.

Resolving the sense of words in a discourse, therefore, constitutes one of the steps in the process of semantic interpretation. We discuss it in greater depth in Section 4.4.

4.2.3 Entity and Event Resolution

Any discourse inevitably consists of a set of entities participating in a series of explicit or implicit events over a period of time. The next important component of semantic interpretation is the identification of various entities that are sprinkled across the discourse using the same or different phrases. Reconciling what type of entity or event is being considered, along with disambiguating various ways in which the same entity is referred to over a discourse, is critical to creating a semantic representation. Two predominant tasks have become popular over the years: **named entity recognition** and **coreference resolution**. These two tasks fall under the umbrella of **information extraction** and are discussed in more detail in Chapter 8.

4.2.4 Predicate-Argument Structure

Once we have the word senses, entities, and events identified, another level of semantic structure comes into play: identifying the participants of the entities in these events. Resolving the argument structure of the predicates in a sentence is where we identify which entities play what part in which event. Generally, this process can be defined as the identification of *who* did *what* to *whom*, *when*, *where*, *why*, and *how*.

Figure 4-1 shows the participants of *say* and *acquire* events.

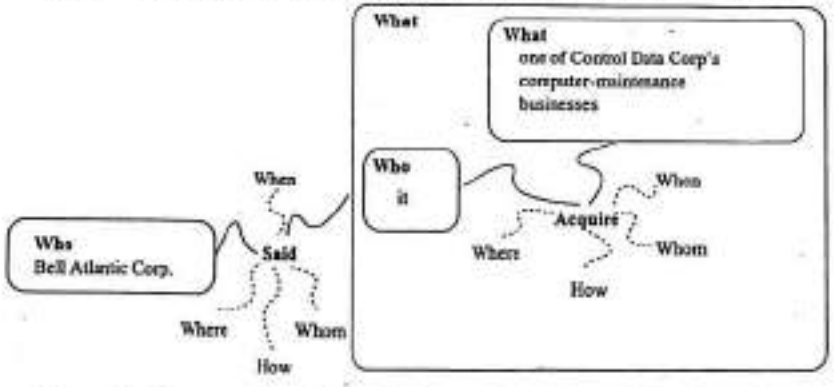


Figure 4-1: A representation of *who* did *what* to *whom*, *when*, *where*, *why*, and *how*

4.2.5 Meaning Representation

The final process of the semantic interpretation is to build a semantic representation or meaning representation that can then be manipulated by algorithms to various application ends. This process is sometimes called the **deep representation**. Unfortunately, as we mentioned earlier, due to the lack of a general-purpose representation that is also deep enough for any given application, most studies in this area have been application dependent, or dependent on the domain of particular applications. The following two examples show sample sentences and their meaning representations for the RoboCup and GeoQuery domains (described in §4.5.1):

(1) If our player 2 has the ball, then position our player 5 in the midfield.
```
((bowner (player our 2)) (do (player our 5) (pos (midfield)))
```

(2) Which river is the longest?
```
answer(x1, longest(x1 river(x1)))
```

This is a domain-specific approach; the remainder of this chapter focuses on domain-independent approaches.

4.3 System Paradigms

The problems discussed in this chapter are familiar to the computational linguistics and linguistics communities. Researchers from these communities have examined meaning representations and methods to recover them at different levels of granularity and generality, exploring the space of numerous languages. For many of the potential experimental conditions, no hand-annotated data is available. Therefore, it is important to get a perspective on the various primary dimensions on which the problem of semantic interpretation has been tackled. It is impossible to cover all these dimensions in this chapter, so while we mention many of the historic approaches, we try to focus on the more prevalent and successful approaches that lend themselves to practical applications. The approaches generally fall into the following three categories.

1. System Architectures
   (a) Knowledge based: As the name suggests, these systems use a predefined set of rules or a knowledge base to obtain a solution to a new problem.
   (b) Unsupervised: These systems tend to require minimal human intervention to be functional by using existing resources that can be bootstrapped for a particular application or problem domain.
   (c) Supervised: These systems involve the manual annotation of some phenomena that appear in a sufficient quantity of data so that machine learning algorithms can be applied. Typically, researchers create feature functions that allow each problem instance to be projected into a space of features. A model is trained to use these features to predict labels, and then it is applied to unseen data.

(d) Semi-Supervised: Manual annotation is usually very expensive and does not yield enough data to completely capture a phenomenon. In such instances, researchers can automatically expand the data set on which their models are trained either by employing machine-generated output directly or by bootstrapping off of an existing model by having humans correct its output. In many cases, a model from one domain is used to quickly adapt to a new domain.

2. Scope

(a) Domain Dependent: These systems are specific to certain domains, such as air travel reservations or simulated football coaching.
(b) Domain Independent: These systems are general enough that the techniques can be applicable to multiple domains without little or no change.

3. Coverage

(a) Shallow: These systems tend to produce an intermediate representation that can then be converted to one that a machine can base its actions on.
(b) Deep: These systems usually create a terminal representation that is directly consumed by a machine or application.

4.4 Word Sense

In a compositional approach to semantics, where the meaning of the whole is composed of the meaning of the parts, the smallest parts under consideration in textual discourse are typically the words themselves: either tokens as they appear in the text or their lemmatized forms. Word sense has been examined and studied for a very long time [3, 4, 5], but its true nature still eludes researchers. It is not clear whether it is possible to identify a finite set of senses that each word in a language exhibits in various contexts. Even if it were possible to do so, it is not clear whether a given word evokes a single discrete sense in a given context or whether the word is associated with a distribution of some subset of all its senses.

Attempts to solve this problem range from rule based and knowledge based to completely unsupervised, supervised, and semi-supervised learning methods. Very early systems were predominantly rule based or knowledge based and used dictionary definitions of senses of words. Unsupervised word sense induction or disambiguation techniques try to induce the senses of a word as it appears in various corpora. These systems perform either a hard or soft clustering of words and tend to allow the tuning of these clusters to suit a particular application. Most recent supervised approaches to word sense disambiguation, on the other hand, primarily assume that a word can evoke only one particular sense in a given context and at a predefined—usually application-independent—level of granularity, although the output of supervised approaches can still be amenable to generating a ranking, or distribution, of membership of senses. In the case of supervised word sense disambiguation, where human annotation is necessary, a delicate balance often exists between making fine-grained distinctions between word senses and maintaining good interannotator agreements given the inventory of senses. The coarser the granularity of senses for a word, the more consistent

the annotation and more learnable they become. However, there is an increased chance that this lower granularity might not identify nuances that are fine enough for the consuming application. An observed win in learning and annotation might not translate directly into the depth of representation of meaning expected by the application. Palmer, Dang, and Fellbaum [6] discussed this issue in great detail.

Although theoretically assumed to be an important aspect of language understanding, the applicability of word sense disambiguation seems to be an issue of much debate. The inherent difficulty of generating huge corpora of manually sense-tagged text, complicated in part by the prevailing agnostic or ambivalent status of applicability of word sense in various applications is also a probable cause of why few computational resources have been generated to support the creation of better automatic systems. Also, the absence of standard criteria has prevented the merging of various resources that have sense information. Some attempts are being made to create mappings between such resources.

One of the principle reasons for this ambivalence, as observed by Resnik and Yarowsky [7], is that in many of the more mature areas of language processing, such as information retrieval and speech recognition, either the sense disambiguation techniques tend to be redundant or cheaper and better alternatives are available. In information retrieval it is a well-accepted fact that the multiple words in a query matching with multiple words in the document context tend to provide an implicit disambiguation that is hard to beat with perfect word sense information [8]. In speech recognition, context classes [9, 10] have always proven to be more applicable than word classes [11]. Specific domains or genres of text tend to invoke a smaller subset, or even just one sense of a given content word. Therefore, in light of the fact that some semantic parsing systems are domain specific and some domain-independent, the disambiguation of sense is more necessary in the latter than in the former. Furthermore, in domain-specific applications, a word usually maps to a unique concept, and thus, finding a good mapping from words to the concepts is an easier problem, further diminishing the necessity for sense disambiguation. Resnik and Yarowsky [7] pointed out several reasons for the lack of progress in word sense disambiguation: a lack of standardized evaluations, the range of resources needed to provide required knowledge as compared to other tasks, and the difficulty of obtaining adequately large sense-tagged data sets. Following that study, the Special Interest Group on LEXicon (SIGLEX) has held several exercises called SENSEVAL 1, 2, and 3 and SEMEVAL 1 and 2. These competitions have been very successful in generating standard datasets and evaluation criteria, as well as identifying related tasks that have advanced the understanding of word sense disambiguation and its applications.

How to measure the performance of automatic word sense disambiguation systems is an important issue. Gale, Church, and Yarowsky [12] discussed it in great detail. Their proposal, which is still commonly followed, is that the lower bound on the performance of a system for disambiguating a word should be the one in which every instance of the lemma is assigned the most frequent sense it exhibits in a sufficiently large corpus. This is commonly known as the most frequent sense, or MFS, baseline. A good property of a gold-standard sense-tagged corpus is that it should be replicable to a high degree. In other words, multiple annotators should be able to annotate the same corpus with a sufficiently high agreement. Let's say this agreement is $x\%$. We would generally view $x\%$ as an upper bound on the performance of any automatic system.

Word sense ambiguities can be of three principal types: (i) homonymy, (ii) polysemy, and (iii) categorial ambiguity [13]. **Homonymy** indicates that the words share the same spelling, but the meanings are quite disparate. Each homonymous partition, however, may contain finer sense nuances that could be assigned to the word depending on the context, and this phenomenon is called **polysemy**. For example, these two senses of the word *bank* are orthogonal: *financial* bank and river bank. Further, *bank* has some other, somewhat finer, and related subsenses that indicate a collection of things: for example, financial bank and bank of clouds. To illustrate **categorial ambiguity**, the word *book* can mean a book such as the one in which this chapter appears or to enter charges against someone in a police register. The former belongs to the grammatical category of noun, and the latter, verb. Distinguishing between these two categories effectively helps disambiguate these two senses. Therefore, categorial ambiguity can be resolved with syntactic information (part of speech) alone, but polysemy and homonymy need more than syntax.

Traditionally, in English, word senses have been annotated for each part of speech separately, whereas in Chinese, the sense annotation has been done per lemma and so ranges across all parts of speech. Part of the reason is that the distinction between a noun or a verb is much more obscure in Chinese.

4.4.1 Resources

As with any language understanding task, the availability of resources is a key factor in the disambiguation of word senses in corpora. Unfortunately, the community has not seen the development of a significant amount of hand-tagged sense data—at least not until very recently. Early work on word sense disambiguation used machine-readable dictionaries or thesauruses as knowledge sources. Two prominent sources were the *Longman Dictionary of Contemporary English* (LDOCE) [14] and *Roget's Thesaurus* [15]. The late 1980s gave birth to a significant lexicographical resource, WordNet [16], which has been very influential. In addition to being a lexical resource with inventories of senses provided for most words in English across multiple parts of speech, it also has a rich taxonomy connecting words across many different relationships, such as hypernymy, homonymy, meronymy, and so on. In addition, to facilitate research in automatic sense disambiguation, a small portion of the Brown Corpus [17] has been annotated with WordNet senses to create a semantic concordance (SEMCOR) corpus [18]. More recently, WordNet has been extended by adding syntactic information on the glosses, disambiguating them with manual and automatic methods and generating logical forms to allow better incorporation in applications such as question answering [19]. Another corpus, the DSO Corpus of Sense-Tagged English, was created by tagging WordNet version 1.5 senses on the Brown and Wall Street Journal (WSJ) corpora for the 121 nouns and 70 verbs that are the most frequent and ambiguous words in English [20]. Further, the SENSEVAL [21] competitions held over the past decade have created many corpora for testing systems on word sense and related problems. The biggest sense annotation effort so far has been the OntoNotes corpus [22, 23, 24] released through the Linguistic Data Consortium (LDC), in which have been tagged a significant number of verb (~2,700) and noun (~2,200) lemmas covering roughly 85% of multiple corpora spanning multiple genres with coarse grained senses and with a very high interannotator agreement. Pradhan et al. [25] based a lexical sample task in SEMEVAL 2007 using this corpus.

Cyc [26] is another good example of a useful resource that creates a formalized representation of common sense knowledge about objects and events in the world to overcome the so-called knowledge bottleneck that is so crucial to word sense disambiguation and many other natural language tasks. Even after a couple of decades of handcrafting this knowledge base, it leaves much to be desired, which underscores the difficulty of such an endeavor.

Fortunately, English seems to have the most highly developed lexicons with various semantic features associated with words and words grouped together to form coherent semantic classes. Efforts are underway to create resources for other languages as well. For example, HowNet [27] is a network of words for Chinese similar to WordNet. The Global WordNet Association (http://www.globalwordnet.org) keeps track of WordNet development across various languages. Researchers are also using semiautomatic methods for expanding coverage of existing languages [28, 29, 30, 31] and for other languages such as Greek [32]. In addition to such corpora annotated with sense information, there are also many resources such as WordNet Domains (http://wndomains.fbk.eu/) that provide structured knowledge to help overcome the knowledge bottleneck in sense disambiguation.

4.4.2 Systems

Now that we have looked at the problem and some resources, we turn to some sense disambiguation systems. As mentioned earlier, researchers have explored various system architectures to address the sense disambiguation problem. We can classify these systems into four main categories: (i) rule based or knowledge based, (ii) supervised, (iii) unsupervised, and (iv) semisupervised.

In the following three sections, we look at each of these systems in order.

Rule Based

The first generation of word sense disambiguation systems was primarily based on dictionary sense definitions and glosses [33, 34]. Most of these techniques were handcrafted and used resources that are not necessarily accessible today. Also, access to the exact rules and systems was very limited, and most information was only available from archived publications and discussions of the specific lexical items and senses that were considered during those experiments. In short, much of this information is historical and cannot readily be translated and made available for building systems today. However, some valuable techniques and algorithms are still accessible, and we look at these in this section. Probably the simplest and oldest dictionary-based sense disambiguation algorithm was introduced by Lesk [35]. The first-generation word sense disambiguation algorithms were mostly based on computerized dictionaries; for example, see Calzolari and Picchi [33].

The first SENSEVAL evaluations [36] used a simplified version of the Lesk algorithm as a baseline for comparing word sense disambiguation performance. The pseudocode for the algorithm is shown in Algorithm 4–1. The core of the algorithm is that the sense of a word in a given context is most likely to be the dictionary sense whose terms most closely overlap with the terms in the context. There have since been further modifications to the algorithm to make it more robust to variation in term usages, context, and definition. Banerjee and Pedersen [37], for example, modified the Lesk algorithm so that synonyms, hypernyms, hyponyms, meronyms, and so on of the words in the context as well as in the dictionary definition are used to get a more accurate overlap statistic. The score

Algorithm 4–1 Pseudocode of the simplified Lesk algorithm
The function COMPUTEOVERLAP returns the number of words common to the two sets
Procedure: SIMPLIFIED_LESK(word, sentence) **returns** best sense of *word*

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

associated with each match is measured as the square of the longest common subsequence between the context and the gloss.[1] Using a context window of five words (two on each side of the target, as well as the target itself), they report a twofold increase in performance from 16% to 32% over the vanilla Lesk algorithm when used on the SENSEVAL-2 lexical sample dataset. This performance improvement is considerable given the simplicity of the algorithm.

Another dictionary-based algorithm was suggested by Yarowsky [38]. This study used *Roget's Thesaurus* categories and classified unseen words into one of these 1,042 categories based on a statistical analysis of 100 word concordances for each member of each category, over a large corpus, in this case the 10-million-word *Grolier's Encyclopedia*. The method performed quite well on a set of 12 words for which there had been some previous quantitative studies. Although the instances and corpora used in this study were not the same as the ones reported previously, it still gives an idea of the success of a relatively simple method. The method consists of three steps, as shown in Figure 4–2. The first step is a collection of contexts. The second step computes weights for each of the salient words. One thing to note is that the amount of context used was 50 words on each side of the target word, which is much higher than the context windows found to be useful for this kind of broad topic classification by Gale et al. [12]. $P(w_i|RCat)$ is the probability of a word w_i occurring in the context of a *Roget's Thesaurus* category *RCat*. Finally, in the third step, the unseen words in the test set are classified into the category that has the maximum weight.

More recently, Navigli and Velardi [39, 40] suggested a knowledge-based algorithm that uses graphical representation of senses of words in context to disambiguate the term under

1. Multiple subsequences in the same gloss are allowable; however, subsequences of only nonintent words such as pronouns, prepositions, articles, and conjunctions are not considered. For example, the subsequence *of the* is not considered in the calculation of a score.

1. Collect contexts for each of the *Roget's Thesaurus* categories.
2. Determine weights for each of the salient words in the context.

$$\frac{P(w_i|RCat)}{P(w_i)}$$

3. Use the weights for predicting the appropriate category of the word in the test corpus.

$$\arg\max_{RCat} \sum_{w} \log \frac{P(w_i|RCat)P(RCat)}{P(w_i)}$$

Figure 4–2: Algorithm for disambiguating words into *Roget's Thesaurus* categories

consideration. This is called the structural semantic interconnections (SSI) algorithm. It uses various sources of information, including WordNet, domain labels [41], and all possible annotated corpora to form structural specifications of concepts, or semantic graphs. The algorithm consists of two steps: an initialization step and an iterative step, in which the algorithm attempts to disambiguate all the words in context iteratively until it cannot disambiguate any further or until all the terms are successfully disambiguated. Its performance is very close to that of supervised learning algorithms. Although it does not technically have a training phase, it surpasses the best unsupervised algorithm in the SENSEVAL-3 all-words task. Figure 4–3 shows the semantic graphs for two senses of the term *bus*. The first one is the *vehicle* sense, and the second one is the *connector* sense.

Notation:

- T (**the lexical context**) is the list of terms in the context of the term t to be disambiguated. $T = [t_1, t_2, \ldots, t_n]$.
- $S_1^t, S_2^t, \ldots, S_n^t$ are structural specifications of the possible concepts (or, senses) of t.
- I (**the semantic context**) is the list of structural specifications of the concepts associated with each of the terms in $T \setminus \{t\}$ (except t). $I = [S^{t_1}, S^{t_2}, \ldots, S^{t_n}]$, that is, the **semantic interpretation** of T.
- G is the grammar defining the various relations between the structural specifications (or **semantic interconnections**) among the graphs.
- Determine how well the structural specifications in I match that of $S_1^t, S_2^t, \ldots, S_n^t$ using G.
- Select the best matching S_i^t.

The algorithm works as follows. A set of pending terms in the context $P = \{t_i | S^{t_i} = null\}$ is maintained, and I is used in each iteration to disambiguate terms in P. The procedure iterates, and each iteration either disambiguates one term in P and removes it from the pending list or stops because no more terms can be disambiguated. The output I is updated with the sense of t. Initially I contains structures for monosemous terms in $T \setminus \{t\}$ and any

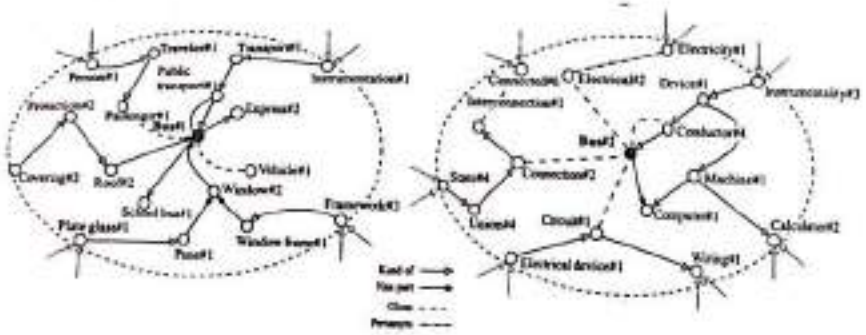


Figure 4–3: The graphs for sense 1 and 2 of the noun bus as generated by the SSI algorithm

possible disambiguated synsets (since we do use sense-tagged data)[2]. If this is a null set, then the algorithm makes an initial guess at what the most likely sense of the least ambiguous term in the context is. During an iteration, the algorithm selects those terms t in P that show semantic interconnections with at least one sense of S of t and one or more senses in I. A function $f_I(S,t)$ determines the likelihood of S being the correct interpretation of t and is defined as:

$$f_I(S,t) = \begin{cases} \rho(\{\varphi(S,S') | S' \in I\}), & \text{if } S \in Senses(t) \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

where $Senses(t)$ are the senses associated with the term t, and

$$\varphi(S,S') = \rho'(\{w(e_1 \cdot e_2 \cdots e_n) | S \xrightarrow{e_1} S_1 \xrightarrow{e_2} \cdots \xrightarrow{e_{n-1}} S_{n-1} \xrightarrow{e_n} S'\}) \tag{4.2}$$

that is, a function (ρ') of the weights (w) of each path connecting S and S', where S and S' are semantic graphs, and edges e_1 to e_n are the edges connecting them. A good choice for ρ and ρ' would be a sum or average sum function.

Finally, a context-free grammar $G = (E, N, SG, PG)$ encodes all the meaningful semantic patterns, where:

$$E = \{e_{kind-of}, e_{has-kind}, e_{part-of}, \dots\}$$

are the edge labels,

$$N = \{S_G, S_s, S_g, S_1, S_2, \dots, E_1, E_2, \dots\}$$

are nonterminal symbols that encode paths between the senses,

$$S_G$$

2. A *synset* is a set of lemmas that all have the same word sense. The term was coined by the creators of WordNet [16].

is the start symbol of the graph G, and

$$P_G = \{S_G \to S_s | S_g, S_s \to S_1 | S_2 | S_3, S_1 \to E_1 S_1 | E_1, E_1 \to e_{kind-of} | e_{part-of}, S_g \to e_{class} S_5 | S_4 | S_1, \dots\}$$

are the productions (roughly 40 in the reported study).

The hierarchical concept information in WordNet has been successfully utilized by many approaches. Refer to Patwardhan, Banerjee, and Pedersen [42] for a comparison of several semantic similarity measures based on WordNet. Recent emergence of unstructured knowledge bases such as Wikipedia has led to a new generation of algorithms that extract the implicit knowledge encoded in them to assist generation of resources that previously relied mostly on WordNet-like resources to generate even wider coverage and multilingual knowledge bases that help further the state of the art in many tasks such as word sense disambiguation. Strube and Ponzetto [43, 44] provided an algorithm called WikiRelate! to estimate the distance between two concepts using the Wikipedia taxonomy. Even more recently, Navigli and Ponzetto [45] introduced a novel method for automatically creating a multilingual lexical knowledge base that establishes a mapping between the large multilingual resource Wikipedia and the English computational lexicon WordNet. It currently includes six languages (German, Spanish, Catalan, Italian, French, and English). The mapping to freely available WordNets in those languages can be easily generated using English WordNet as the interlingua. The continued growth of Wikipedia will enable the generation of resources for many other languages using this methodology. As a starting point, Ponzetto and Navigli [46] have already shown that the English information in BabelNet can be used to create a word sense disambiguation system that rivals previous methods on the task of coarse-grained sense dismabiguation as well as domain-specific sense disambiguation.

Supervised

Ironically, the simpler form of word sense disambiguating systems—the supervised approach, which tends to transfer all the complexity to the machine learning machinery while still requiring hand annotation—tends to be superior to unsupervised methods and performs best when tested on annotated data [21]. The downside to this approach is that the sense inventory has to be predetermined, and any change in the inventory might necessitate a round of expensive reannotation.

These systems typically consist of a machine learning classifier trained on various features extracted for words that have been manually disambiguated in a given corpus and the application of resulting models to disambiguate words in unseen test sets. A good feature of these systems is that the user can incorporate rules and knowledge in the form of features, and possibly semiautomatically generate training data to augment the set that has been manually annotated, in an attempt to achieve the best of all three approaches. Of course, a particular knowledge source and/or classifier combination may have issues that make it less amenable to deriving the most optimal feature representation, and the semiautomatically generated sense-tagged data could be noisy to varying degrees. Nevertheless, state-of-the-art systems usually tend to be a combination of rich features and exploitation of redundancy in language.

We look at some of the typical systems and features in this section. Brown et al. [47] were probably the first to use machine learning for word sense disambiguation using information in parallel corpora. Yarowsky [48] was among the first to use a rich set of features in a machine learning framework—decision lists—to tackle the word sense problem. Several other researchers, such as Ng and Lee [20, 49], have used and refined those features in several variations, including different levels of context and granularities: sentence, paragraph, microcontext, and so on. In this section, we look at some of the more popular methods and features that are relatively easy to obtain.

Classifier Probably the most common and high-performing classifiers are support vector machines (SVMs) and maximum entropy (MaxEnt) classifiers. Many good-quality, freely available distributions of each are available and can be used to train word sense disambiguation models. Typically, because each lemma has a separate sense inventory, it is almost always the case that a separate model is trained for each lemma and POS combination (i.e., if the language, as in the case of English, has separate sense inventories for various parts of speech).

Features We discuss a more commonly found subset of features that have been useful in supervised learning of word sense. These are not exhaustive by any means, but ones that have been time-tested, and provide a very good base that can be used to achieve nearly state-of-the-art performance.

Lexical context—This feature comprises the words and lemmas of words occurring in the entire paragraph or a smaller window of usually five words.

Parts of speech—This feature comprises the POS information for words in the window surrounding the word that is being sense tagged.

Bag of words context—This feature comprises using an unordered set of words in the context window. A threshold is typically tuned to include the most informative words in the larger context.

Local collocations—Local collocations are an ordered sequence of phrases near the target word that provide semantic context for disambiguation. Usually, a very small window of about three tokens on each side of the target word, most often in contiguous pairs or triplets, are added as a list of features. For example, if the target word is w, then $C_{i,j}$ would be a collocation where i and j refer to the start and offsets with respect to the word w. A positive sign indicates words on the right, and a negative sign indicates words on the left of the target.

The following set of 11 features is the union of the collocation features used in Ng and Lee [20, 50]. $C_{-1,-1}, C_{1,1}, C_{-2,-2}, C_{2,2}, C_{-2,-1}, C_{-1,1}, C_{1,2}, C_{-3,-1}, C_{-2,1}, C_{-1,2}, C_{1,3}$. To illustrate a few of these, let's take our earlier example for disambiguating *nail*: *He bought a box of nails from the hardware store.* In this example, the collocation $C_{1,1}$ would be the word *from*, and $C_{1,2}$ would be the string *from_the_hardware*, and so on. Usually, stop-words and punctuations are not removed before creating the collocations. Boundary conditions are treated by adding a null word in a collocation. Researchers could also experiment using root forms of the words and other variations that might help better generalize the context. A guideline on what criteria to consider in choosing the number and context of collocations is discussed by Gale et al. [12].

Syntactic relations—If the parse of the sentence containing the target word is available, then we can use syntactic features. One set of features that was proposed by Lee and Ng [49] is listed in Algorithm 4–2.

Topic features—The broad topic, or domain, of the article that the word belongs to is also a good indicator of what sense of the word might be most frequent.

Chen and Palmer [51] recently proposed some additional rich features for disambiguation:

Voice of the sentence—This ternary feature indicates whether the sentence in which the word occurs is a passive, semipassive,[3] or active sentence.

Presence of subject/object—This binary feature indicates whether the target word has a subject or object. Given a large amount of training data, we could also use the actual lexeme and possibly the semantic roles rather than the syntactic subject/objects.

Sentential complement—This binary feature indicates whether the word has a sentential complement.

Prepositional phrase adjunct—This feature indicates whether the target word has a prepositional phrase, and if so, selects the head of the noun phrase inside the prepositional phrase.

Algorithm 4–2 Rules for selecting syntactic relations as features

```
if w is a noun then
    select parent head word (h)
    select part of speech of h
    select voice of h
    select position of h (left, right)
else if w is a verb then
    select nearest word l to the left of w such that w is the parent head word of l
    select nearest word r to the right of w such that w is the parent head word of r
    select part of speech of l
    select part of speech of r
    select part of speech of w
    select voice of w
else if w is a adjective then
    select parent head word (h)
    select part of speech of h
end if
```

3. Verbs that are past participles and not preceded by *be* or *have* verbs are semipassive.

Named entity—This feature is the named entity of the proper nouns and certain types of common nouns.

WordNet—WordNet synsets of the hypernyms of head nouns of the noun phrase arguments of verbs and prepositions.

More recently, following research in semantic role labeling, Dligach and Palmer [52] proposed the following features for verb sense disambiguation:

Path—This feature is the path from the target verb to the verb's arguments.

Subcategorization—The subcategorization frame is essentially the string formed by joining the verb phrase type with that of its children.

Most likely, developers will have to perform a feature selection per-word to get the best set of features for a particular word.

Unsupervised

Progress in word sense disambiguation is stymied by the dearth of labeled training data to train a classifier for every sense of each word in a given language. There are a few solutions to this problem:

1. Devise a way to cluster instances of a word so that each cluster effectively constrains the examples of the word to a certain sense. This could be considered **sense induction** through clustering.
2. Use some metrics to identify the proximity of a given instance with some sets of known senses of a word and select the closest to be the sense of that instance.
3. Start with seeds of examples of certain senses, then iteratively grow them to form clusters.

We do not discuss in much detail the mostly clustering-based sense induction methods here. We assume that there is already a predefined sense inventory for a word and that the unsupervised methods use very few, if any, hand-annotated examples, and then attempt to classify unseen test instances into one of their predetermined sense categories.

We first look at the category of algorithms that use some form of distance measure to identify senses. Rada et al. [53] introduced a metric for computing the shortest distance between the two pairs of senses in WordNet. This metric assumes that multiple co-occurring words are likely to exhibit senses that would minimize the distance in a semantic network of hierarchical relations, for example, IS-A, from WordNet. Resnik [54] proposed a new measure of semantic similarity: **information content** in an IS-A taxonomy which produces much better results than the edge-counting measure. Agirre and Rigau [55] further refined this measure, calling it **conceptual density**, which not only depends on the number of separating edges but is also sensitive to the depth of the hierarchy and the density of its concepts and is independent of the number of concepts being measured. Conceptual density is defined for each of the subhierarchies in Figure 4–4. The sense that

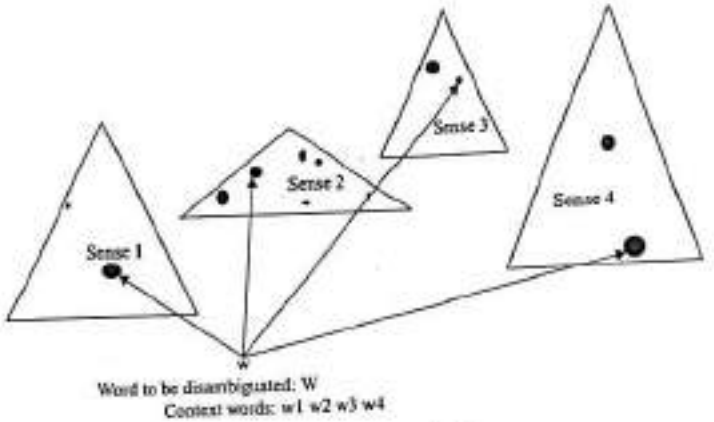


Figure 4–4: Conceptual density

falls in the subhierarchy with the highest conceptual density is chosen to be the correct sense.

$$CD(c,m) = \frac{\sum_{i=0}^{m-1} \text{hyponyms}^{i^{-0.20}}}{\text{descendants}_c} \tag{4.3}$$

In Figure 4-4, Sense 2 is the one with the highest conceptual density and is therefore the chosen sense.

Resnik [56] observed that selectional constraints and word sense are closely related and identified a measure by which to compute the sense of a word on the basis of predicate-argument statistics. Note that this algorithm is primarily limited to the disambiguation of nouns that are arguments of verb predicates.

Let A_R be the selectional association of the predicate p to the concept c with respect to argument R. A_R is defined as:

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

If n is the noun that is in an argument relation R to predicate p, and $\{s_1, s_2, \ldots, s_k\}$ are its possible senses, then, for i from 1 to k, compute:

$$C_i = \{c | c \text{ is an ancestor of } s_i\} \tag{4.4}$$

$$a_i = \max_{c \in C_i} A_R(p,c) \tag{4.5}$$

where a_i is the score for sense s_i. The sense s_i which has the largest value of a_i is sense for the word. Ties are broken by random choice.

Leacock, Miller, and Chodorow [58] provide another algorithm that makes use of corpus statistics and WordNet relations, and show that monosemous relatives can be exploited for disambiguating words.

Algorithms Motivated by Crosslinguistic Evidence

There is a family of unsupervised algorithms based on crosslinguistic information or evidence. Brown et al. [47] were probably the first to make use of this information for purposes of sense disambiguation. They were particularly interested not only in the sense distinctions that were restricted to the ones made by monolingual dictionary resources but also in sense differences that required translating into other languages. They provide a method to use the context information for a given word to identify its most likely translation in the target language. This idea was further explored by Dagan and Itai [59], who use a bilingual lexicon paired with a monolingual corpus to acquire statistics on word senses automatically. They also propose that syntactic relations along with word co-occurrences statistics provide a good source to resolve lexical ambiguity. Further experiments were performed by Diab [60] using machine translated English-to-Arabic translations to extract sense information for training a supervised classifier. These experiments compared favorably with other purely unsupervised methods. The algorithm SALAAM, which requires a word-aligned parallel corpus, is described in Figure 4–5.

Semisupervised

The next category of algorithms we look at are those that start from a small seed of examples and an iterative algorithm that identifies more training examples using a classifier. This additional, automatically labeled data can then be used to augment the training data of the classifier to provide better predictions for the next selection cycle, and so on. The Yarowsky algorithm [61] is the classic case of such an algorithm and was seminal in introducing semisupervised methods to the word sense disambiguation problem. The algorithm is based on the assumption that two strong properties are exhibited by corpora:

1. **One sense per collocation:** Syntactic relationship and the types of words occurring nearby a given word tend to provide a strong indication as to the sense of that word.

1. L1 words that translate into the same L2 word are grouped into clusters.
2. SALAAM (Sense Assignment Leveraging Alignment and Multilinguality) identifies the appropriate senses for the words in those clusters according to the words senses' proximity in WordNet. The word sense proximity is measured in information theoretic terms on the basis of an algorithm by Resnik [57].
3. A sense selection criterion is applied to choose the appropriate sense label or set of sense labels for each word in the cluster.
4. The chosen sense tags for the words in the cluster are propagated back to their respective contexts in the parallel text. Simultaneously, SALAAM projects the propagated sense tags for L1 words onto their L2 corresponding translations.

Figure 4–5: SALAAM algorithm for creating training using parallel English-to-Arabic machine translations

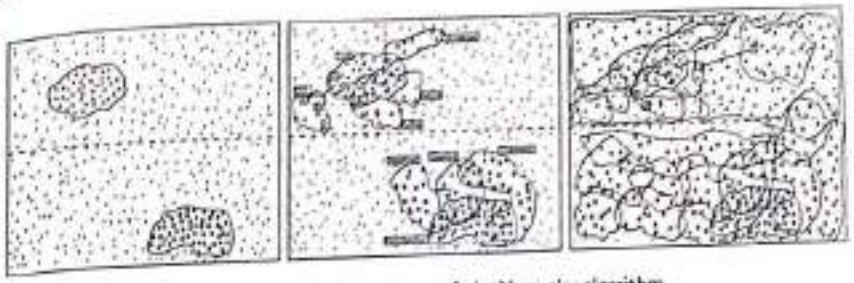


Figure 4–6: The three stages of the Yarowsky algorithm

2. **One sense per discourse:** Usually, in a given discourse, all instances of the same lemma tend to invoke the same sense.

Based on the assumption that these properties exist, the Yarowsky algorithm iteratively disambiguates most of the words in a given discourse.

Figure 4–6 shows the three stages of the algorithm. In the first box, *life* and *manufacturing* are used as collocates to identify the two senses of *plant*. Then, in the next iteration, a new collocate *cell* is identified, and the final block shows the small residual remaining at the end of the algorithm cycle. This algorithm, as described in Figure 4–7, has been shown to perform well on a small number of examples. For it to be successful, it is important to select a good way to identify seed examples and to devise a way to identify potential corruption of the labeled pool by wrong examples. More recently, Galley and McKeown [62] showed that the assumption of one sense per discourse assumption improves performance.

Another variation of semisupervised systems is the use of unsupervised methods for the creation of data combined with supervised methods to learn models for that data. The presumption is that the potential noise of wrong examples selected from a corpus during this process would be low enough so as not to affect learnability. Another presumption is that the overall discriminative ability of the model is superior to purely unsupervised methods or to situations in which not enough hand-annotated data is available to train a purely supervised system. Mihalcea and Moldovan [63] describe one such system in which the algorithm in Figure 4–8 is used to obtain examples from large corpora for particular senses in WordNet.

Mihalcea [64] proposes the following method using Wikipedia for automatic word sense disambiguation.

- **Extract** all the sentences in Wikipedia in which the word under consideration is a link. There are two types of links: a simple link, such as `[[bar]]`, or a piped link, such as `[[musical_notation|bar]]`.
- **Filter** those links that point to a disambiguation page. This means that we need further information to disambiguate the word. If the word does not point to a disambiguation page, then the word itself can be the label. For all piped links, the string before the pipe serves as the label.
- **Collect** all the labels associated with the word, and then map them to possible WordNet senses. Sometimes they might all map to the same sense, essentially making the

Step 1. In a sufficiently large corpus, identify all the instances of a particular polysemous word that needs to be disambiguated, storing its context alongside.
Step 2. Identify a small set of instances that are strongly representative of one of the senses of the word. This can either be done in a completely unsupervised fashion by identifying collocations that give a strong indication of the sense usage for the word under consideration or by manually tagging a small portion of the data. In this example, we assume a polysemous word with only two senses, but this algorithm can be extended to n senses.
Step 3.

Step 3a. Train a supervised classifier on this set of examples.

Step 3b. Using these classifiers, classify the remaining instances of the word in the corpus and select those that are classified above a certain level of confidence.

Step 3c. Filter out the possible misclassifications using one sense per discourse constraint, and identify possible new collocations to be added to the list of seed collocations.

Step 3d. Repeat step 3 iteratively, thereby slowly shrinking the residual.

Step 4. Stop. At some point, a small, stable residual will remain.
Step 5. The trained classifier can now be used to classify new data, and that in turn can be used to annotate the original corpus with sense tags and probabilities.

Figure 4–7: The Yarowsky algorithm

verb monosemous and not useful for this purpose. Often, the categories can be mapped to a significant number of WordNet categories, thereby proving sense-dissmbiguated data for training. The manual mapping is a relatively inexpensive process.

This algorithm provides a cheap way of extracting sense information for many words that display the required properties, and it can alleviate the manually intensive process of sense tagging. Depending on how many words in the entire Wikipedia exhibit this property, it could be very useful for generating sense-tagged data. A rough idea of the coverage of this method can be gleaned from the fact that roughly 30 of 49 nouns that were used for SENSEVAL-2 and SENSEVAL-3 were found to have more than two senses for which data could be extracted from Wikipedia. The average disambiguation accuracy on these senses was in the mid-80% range. The interannotator agreement for mapping the senses to WordNet was around 91%.

4.4.3 Software

Several software programs are made available by the research community for word sense disambiguation, ranging from similarity measure modules to full disambiguation systems. It is not possible to list all of them here, so we list a selected few.

Step 1. *Preprocessing*

- For each sense of a word W, determine the synsets of WordNet in which it appears. For each such synset, determine monosemous words included in that synset. Parse the gloss definition attached to each synset.

Step 2. *Search*

- Form search phrases using the following procedures in order of preference
  1. If they exist, extract monosemous synonyms from the synsets selected in step 1.
  2. Select each of the unambiguous parsed constituents in the gloss as a search phrase.
  3. After parsing the gloss, replace all stop-words with a NEAR operator and create a query from the words in the current synset. For example, if the synset for *produce#6* is *grow, raise, farm, produce*, and the gloss is *cultivate by growing*, then the query will look like: *cultivate* NEAR *growing* AND (*grow* OR *raise* OR *farm* OR *produce*).
  4. Use only the head phrase combined by words in the synset using the AND operator. For example, if the definition for *company#5* is *band of people* and its synset is (*party, company*), then the query becomes: *band of people* AND (*party* OR *company*).
- Search the Internet with the phrases determined in the previous step and gather matching documents
- From these documents, extract the sentences containing these words

Step 3. *Postprocessing*

- Keep only those sentences in which the word under consideration belongs to the same part of speech as the selected sense, and delete the others.

Figure 4–8: Mihalcea and Moldovan [63] algorithm for generating examples for words tagged with particular senses by querying a very large corpus

- **IMS** (It Makes Sense) http://nlp.comp.nus.edu.sg/software
  This is a complete word sense disambiguation system.
- **WordNet-Similarity-2.05** http://search.cpan.org/dist/WordNet-Similarity
  These WordNet Similarity modules for Perl provide a quick way of computing various word similarity measures.
- **WikiRelate!**
  http://www.h-its.org/english/research/nlp/download/wikipediasimilarity.php
  This is a word similarity measure based on the categories in Wikipedia.

4.5 Predicate-Argument Structure

Shallow semantic parsing, or what is popularly known today as **semantic role labeling**, is the process of identifying the various arguments of predicates in a sentence. The linguistic community has been debating for a few decades over what constitutes the set of arguments and what the granularity of such argument labels should be for various predicates, which in turn can be the verbs, nouns, adjectives, and prepositions in a sentence [65, 66].

4.5.1 Resources

The late 1990s saw the emergence of two important corpora that are semantically tagged. One is FrameNet[4] [67, 68, 69, 70] and the other is PropBank[5] [71]. These resources have begun a transition from a long tradition of predominantly rule-based approaches toward more data-oriented approaches. These approaches focus on transforming linguistic insights into features, rather than into rules, and letting a machine learning framework use those features to learn a model that helps automatically tag the semantic information encoded in such resources. FrameNet is based on the theory of **frame semantics**, where a given predicate invokes a **semantic frame**, thus instantiating some or all of the possible semantic roles belonging to that frame [72]. PropBank, on the other hand, is based on Dowty's [73] prototype theory and uses a more linguistically neutral view in which each predicate has a set of core arguments that are predicate dependent, and all predicates share a set of non-core, or adjunctive, arguments. It builds on the syntactic Penn Treebank corpus. We now discuss these approaches in more detail.

FrameNet

FrameNet contains frame-specific semantic annotation of a number of predicates in English. It contains tagged sentences extracted from the British National Corpus (BNC). The process of FrameNet annotation consists of identifying specific semantic frames and creating a set of frame-specific roles called **frame elements**. Then, a set of predicates that instantiate the semantic frame, irrespective of their grammatical category, are identified, and a variety of sentences are labeled for those predicates. The labeling process entails identifying the frame that an instance of the predicate lemma invokes, then identifying semantic arguments for that instance, and tagging them with one of the predetermined set of frame elements for that frame. The combination of the predicate lemma and the frame that its instance invokes is called a lexical unit (LU). This is therefore the pairing of a word with its meaning. Each sense of a polysemous word tends to be associated with a unique frame. For example, the verb *break* can mean *fail to observe (a law, regulation, or agreement)* and can belong to a COMPLIANCE frame along with other word meanings such as *violation, obey, flout*; or it can mean *cause to suddenly separate into pieces in a destructive manner* and can belong to a CAUSE_TO_FRAGMENT frame along with other meanings such as *fracture, fragment, smash*.

4. http://framenet.icsi.berkeley.edu/
5. http://verbs.colorado.edu/~mpalmer/projects/ace.html

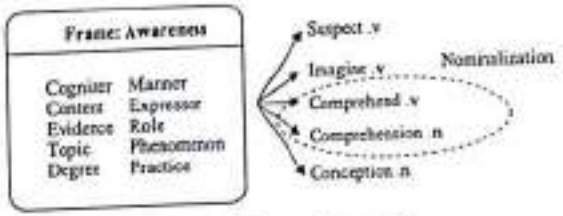


Figure 4–9: FrameNet example

The following example illustrates the general idea. Here, the frame AWARENESS is instantiated by the verb predicate *believe* and the noun predicate *comprehension*. Figure 4–9 shows the AWARENESS frame along with the frame-elements and a sample set of predicates that involve it—including verbs and nominalizations.

1. [$_{Cognizer}$ We] [$_{Predicate:verb}$ believe] [$_{Content}$ it is a fair and generous price]
2. No doubts existed as to [$_{Cognizer}$ our] [$_{Predicate:noun}$ comprehension] [$_{Content}$ of it]

FrameNet encompasses a wide variety of nominal predicates. They include ultra-nominals [74, 73], nominals, and nominalizations. FrameNet also contains some adjective and preposition predicates.

To get an idea of the amount of data we are talking about as of this writing, the latest release of FrameNet, R1.5, contains about 173,000 predicate instances covering about 8,000 frame elements from roughly 1,000 frames over the BNC. Although the number of frame elements seems very large, many share the same meaning across the 11,600 lexical units. For example, the frame element BODY_PART in frame CURE has the same meaning as the same element in the frame GESTURE or in WEARING.

PropBank

PropBank only includes annotations of arguments of verb predicates. All the noncopular verbs in the WSJ part of the Penn Treebank [75] have been labeled with their semantic arguments. PropBank restricts the argument boundaries to that of a syntactic constituent, as defined in the Penn Treebank. It uses a linguistically neutral terminology to label the arguments. The arguments are tagged as either **core arguments**, with labels of the type ARGN, where N takes values from 0 to 5, or **adjunctive arguments** (listed in Table 4–2), with labels of the type ARGM-X, where X can take values such as TMP for temporal, LOC for locative, and so on. Adjunctive arguments share the same meaning across all predicates, whereas the meaning of core arguments has to be interpreted in connection with a predicate. ARG0 is the PROTO-AGENT (usually the subject of a transitive verb), ARG1 is the PROTO-PATIENT (usually its direct object of the transitive verb) [73]. Table 4–1 shows a list of core arguments for the predicates *operate* and *author*. Note that some core arguments, such as ARG2 and ARG3, do not occur with *author*. This is explained by the fact that not all core arguments can be instantiated by all senses of all predicates. A list of core arguments that can occur with a particular sense of the predicate, along with their real-world meaning, is present in a file called the *frames* file. One frames file is associated with each predicate.

Table 4–1: Argument labels associated with the predicate *operate.01* (sense: work), and for *author.01* (sense: to write or construct) in the PropBank corpus

Predicate	Argument	Description
operate.01		
	ARG0	Agent, operator
	ARG1	Thing operated
	ARG2	Explicit patient (thing operated on)
	ARG3	Explicit argument
	ARG4	Explicit instrument
author.01		
	ARG0	Author, agent
	ARG1	Text authored

Table 4–2: List of adjunctive arguments in PropBank—ARGMs

Tag	Description	Examples
ARGM-LOC	Locative	*the museum, in Westborough, Mass.*
ARGM-TMP	Temporal	*now, by next summer*
ARGM-MNR	Manner	*heavily, clearly, at a rapid rate*
ARGM-DIR	Direction	*to market, to Bangkok*
ARGM-CAU	Cause	*In response to the ruling*
ARGM-DIS	Discourse	*for example, in part, Similarly*
ARGM-EXT	Extent	*at $38.375, 50 points*
ARGM-PRP	Purpose	*to pay for the plant*
ARGM-NEG	Negation	*not, n't*
ARGM-MOD	Modal	*can, might, should, will*
ARGM-REC	Reciprocals	*each other*
ARGM-PRD	Secondary Predication	*to become a teacher*
ARGM	Bare ARGM	*with a police escort*
ARGM-ADV	Adverbials	*(none of the above)*

An example extracted from the PropBank corpus along with its syntax tree representation and argument labels is shown in Figure 4–10.

Most Treebank-style trees have **trace nodes** that refer to another node in the tree but have no words associated with them. These can also be marked as arguments. Because traces are not reproduced by a usual syntactic parser, the community has disregarded them from most standard experiments. PropBank also contains arguments that are coreferential. As with any strategy that integrates multiple layers of annotation, there exist some disagreements between the Treebank and PropBank annotation. Sometimes the PropBankers strongly believed an error existed in the tree or the tree structure was not amenable to a

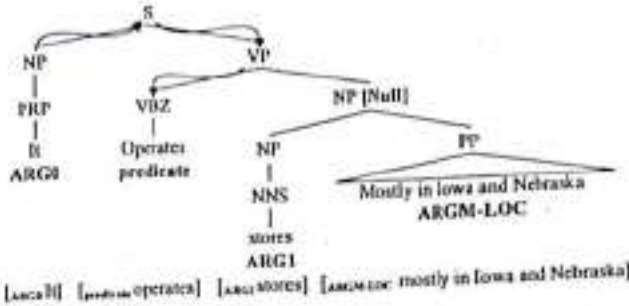


Figure 4–10: Syntax tree for a sentence illustrating the PropBank tags

one-to-one mapping between the argument and a tree node. In such cases, they annotated a sequence of nodes in the tree as the argument. These are called **discontiguous arguments**, and very few (~1%–2%) cases exist in the data. There are around 115,000 predicate instances instantiating about 250,000 instances of the roughly 20 argument types over about 5,000 frames in the WSJ portion of the PropBank release. There are about 18,000 more predicates annotated with arguments from the Brown Corpus. More recently, the OntoNotes project [22, 23, 24] has annotated more corpora from various genres with predicate-argument structures using the PropBank guidelines. This has led to modifications to the Penn Treebank and PropBank guidelines to generate a better, more aligned resource [76]. Most experiments discussed in this chapter use the WSJ annotation from PropBank v1.0.

An important distinction to note between the FrameNet and PropBank corpora is that in FrameNet there exist lexical units, which are words paired with their meanings or the frames that they invoke, whereas in PropBank each lemma has a list of different **framesets** that represent all the senses for which there is a different argument structure. These are akin to word senses but tend to be coarser grained [77].

Other Resources

Other resources have been developed to aid further research in predicate-argument recognition. NomBank [78] was inspired by PropBank. In the process of identifying and tagging the arguments of nouns, the NOMLEX (NOMinalization LEXicon) [79] dictionary was expanded to cover about 6,000 entries. Along with this, the frames from PropBank were used to generate the frame files for NomBank. Another resource that ties PropBank frames with more predicate-independent thematic roles and also provides a richer representation associating the framesets with Levin classes [80] is VerbNet [81]. In fact the PropBank frames have a strong connection with the Levin verb classes, specifically the intersective Levin classes [82]. FrameNet frames are also somewhat related in the sense that FrameNet's generation of verb classes is more data driven than theoretical. Baker and Ruppenhofer [83] present an interesting discussion on how the FrameNet frames relate to Levin classes.

Although FrameNet and PropBank started with annotating predicate-argument structures in English, it was not long before their philosophy propagated to other languages. Because FrameNet was based on frame semantics with coarse-grained semantic frames, and

the nature of semantics is lingua independent, it was apparent that these frames could be reused to annotate data in other languages. The SALSA project [84, 85] was the first to put this into practice. FrameNet tags both literal and metaphorical interpretations of text, but that can lead to ambiguity and lower consistency, so the SALSA project remained close to the literal meaning. It reused all possible, preexisting FrameNet frames, and when linguistic semantic parallelism did not propagate across languages, they created more frames. Subsequently, there exist FrameNets in Japanese [86, 87], Spanish [88], and Swedish [89]. As of this writing, there are FrameNet projects underway in more than 10 languages [90].

PropBank has inspired the creation of similar resources in Chinese [91], Arabic [92, 93], Korean [94], Spanish, Catalan [95], and most recently, Hindi [96]. Many of these involve the same core researchers. Unlike FrameNet, every new PropBank requires the creation of a new set of frame files.

Although the FrameNet and PropBank philosophies have inspired similar projects in other languages, they are not the only styles used in practice. For example, the Prague Dependency Treebank [97] takes a different approach, tagging the predicate-argument structure in its tectogrammatic layer on top of the dependency structure. It also makes a distinction similar to that of core and adjunctive arguments called **inner participants** and **free modifications**. The NAIST Text Corpus [98] is strongly influenced by the traditions in Japanese linguistics.

4.5.2 Systems

Unlike word sense disambiguation, little research has gone into learning predicate-argument structures from unannotated corpora, perhaps because it is closer to the actual applications and has been more or less absorbed in the area of information extraction. Most early systems, such as KL-ONE [99] and others [100, 101], were based primarily on heuristics on syntax trees, which were in turn rule based until the Penn Treebank was available as a training resource for supervised syntactic parsers. Most of these systems dealt with (predicate-independent) thematic roles. There has been a great deal of linguistic inquiry into the nature of argument structure, but most of it did not directly apply to domain-independent understanding systems. Until corpora were available, the main resources were rules based on syntactic trees. One significantly useful resource, mentioned earlier in context of PropBank, was the classification of verbs and their alternations by Levin [80]. The Absity parser [102, 13] is one of the first rule-based semantic parsers. Also notable was the parser used in the PUNDIT understanding system [103, 104]. Efforts were later made to use a hybrid method for thematic role tagging [105, 106] using WordNet as a resource for domain-specific interpretation [107]. Other notable efforts are the corpus-based studies by Manning [108] and Briscoe and Carroll [109], which seek to derive the subcategorization information from large corpora, and by Pustejovsky [110], which tries to acquire lexical semantic knowledge from corpora.

A major leap forward in semantic role labeling research happened after the introduction of FrameNet and PropBank. One big problem with the FrameNet philosophy, and also partly with that of PropBank, is that significant work goes into creating the frames, that is, in classifying verbs into the framesets in preparation for manual annotation. Therefore, providing coverage for all possible verbs in one or more languages requires significant manual effort upfront. Green, Dorr, and Resnik [111] propose a way to learn the frame structures

automatically, but the result is not accurate enough to replace the manual frame creation. Swier and Stevenson [112] represent one of the more recent approaches to handling this problem in an unsupervised fashion.

Let us now review the more recent and common approaches since the advent of these corpora. The process of semantic role labeling can be defined as identifying a set of word sequences, each of which represents a semantic argument of a given predicate. For example, for the sentence in Figure 4–10, for the predicate *operates*, the word *I* fills the role ARG0, the word *stores* fills the role ARG1, and the sequence of words *mostly in Iowa and Nebraska* fills the role ARGM-LOC. Recall that PropBank is largely agnostic to any generalizations made across predicates; an ARGN for one predicate need not have similar semantics compared to another predicate.[6]

FrameNet was the first project that used hand-tagged arguments of predicates in data, and Gildea and Jurafsky [113] were the first to formulate semantic role labeling as a supervised classification problem that assumes the arguments of a predicate and the predicate itself can be mapped to a node in the syntax tree for that sentence. They introduced three tasks that could be used to evaluate the system and that have become standard since:

Argument identification—This is the task of identifying all and only the parse constituents that represent valid semantic arguments of a predicate.

Argument classification—Given constituents known to represent arguments of a predicate, assign the appropriate argument labels to them.

Argument identification and classification—This task is a combination of the previous two tasks, where the constituents that represent arguments of a predicate are identified, and the appropriate argument label is assigned to them.

Once the sentence has been parsed using a syntactic parser, each node in the parse tree can be classified as either one that represents a semantic argument (i.e., non-null node) or one that does not represent any semantic argument (i.e., a null node). The non-null nodes can then be further classified into the set of argument labels.

For example, in the tree of Figure 4–10, the noun phrase that encompasses *stores mostly in Iowa and Nebraska* is a null node because it does not correspond to a semantic argument. The node NP that encompasses *stores* is a non-null node because it does correspond to a semantic argument: ARG1.

The pseudocode for a generic semantic role labeling (SRL) algorithm is shown in Algorithm 4-3.

Syntactic Representations

As we have seen, PropBank was created as a layer of annotation on top of Penn Treebank-style phrase structure trees, and in some of the early work in recovering PropBank annotations, Gildea and Jurafsky [113] added argument labels to parses obtained from a parser trained on Penn Treebank. In subsequent years, researchers have used various other types of

6. The PropBank project did not haphazardly assign argument role numbers, either. For instance, in practice, ARG0 tends to have the role *Agent* and ARG1 tends to have the role *Patient*, to borrow the terminology from θ-roles.

Algorithm 4–3 The semantic role labeling (SRL) algorithm

Procedure: SRL(sentence) **returns** best *semantic role labeling*

Input: *syntactic parse* of the *sentence*

1: *generate* a full syntactic parse of the *sentence*
2: *identify* all the *predicates*
3: **for all** *predicate* ∈ *sentence* **do**
4: *extract* a set of features for each node in the tree relative to the *predicate*
5: *classify* each feature vector using the *model* created in training
6: *select* the class of highest scoring classifier
7: **return** best *semantic role labeling*
8: **end for**

sentence representations, either directly or as an independent source of evidence, to tackle the semantic role labeling problem. We look at each of these sentence representations in turn and at the features that were used to tag text with PropBank arguments.

Phrase Structure Grammar (PSG) FrameNet marks word spans in sentences to represent arguments, whereas PropBank tags nodes in a treebank tree with arguments. Because several high-quality statistical parsers existed that could produce phrase structure trees, and because the phrase structure representation is amenable to tagging, Gildea and Jurafsky [113] used it. They introduced the following features, some of which were extracted from the parse tree of the sentence:

Path—This feature is the syntactic path through the parse tree from the parse constituent to the predicate being classified. For example, in Figure 4–10, the path from ARG0 *It* to the predicate *operates* is represented by the string NP↑S↓VP↓VBZ. ↑ and ↓ represent upward and downward movement in the tree respectively.

Predicate—The identity of the predicate lemma is used as a feature.

Phrase type—This feature is the syntactic category (NP, PP, S, etc.) of the constituent to be labeled.

Position—This feature is a binary feature identifying whether the phrase is before or after the predicate.

Voice—This feature indicates whether the predicate is realized as an active or passive construction. A set of handwritten `tgrep2`[7] expressions on the syntax tree are used to identify the passive-voiced predicates.

Head word—This feature is the syntactic head of the phrase. It is calculated using a head word table described by Magerman [114] and modified by Collins [115].

7. See http://tedlab.mit.edu/~dr/Tgrep2/.

Subcategorization—This feature is the phrase structure rule expanding the predicate's parent node in the parse tree. For example, in Figure 4–10, the subcategorization for the predicate *operates* is VP→VBZ-NP.

Verb clustering—The predicate is one of the most salient features in predicting the argument class. Given various syntactic/semantic constructions that a predicate can appear in, any amount of hand-tagged training data would be relatively limited for estimating the parameters of a model, and any real-world test set will contain predicate sense/frames that have not been seen in training. In these cases, researchers can benefit from some information about the predicate by creating clusters or classes and using them as features. Gildea and Jurafsky [113] used a distance function for clustering that is based on the intuition that verbs with similar semantics will tend to have similar direct objects. For example, verbs such as *eat*, *devour*, and *savor* will tend to all occur with direct objects describing food. The clustering algorithm uses a database of verb-direct object relations extracted by Lin [116]. The verbs were clustered into 64 classes using the probabilistic co-occurrence model of Hofmann and Puzicha [117].

Surdeanu et al. [118] suggested the following additional features:

Content word—Because the head word feature of some constituents, such as PP and SBAR, is not very informative, they defined a set of heuristics for some constituent type. A different set of rules was used to identify a so-called **content** word instead of using the usual head word–finding rules. This was used as an additional feature. The rules that they used are shown in Figure 4–11.

Part of speech of the head word and content word—Adding part of speech of the head word and content word of a constituent as a feature to help generalize in the

```
H1: if phrase type is PP then select the rightmost child
    Example: phrase = "in Texas," content word = "Texas"
H2: if phrase type is SBAR then select the leftmost sentence (S*) clause
    Example: phrase = "that occurred yesterday," content word = "occurred"
H3: if phrase type is VP then
      if there is a VP child then
        select the leftmost VP child
      else
        select the head word
    Example: phrase = "had placed," content word = "placed"
H4: if phrase type is ADVP then select the rightmost child, not IN or TO
    Example: phrase = "more than," content word = "more"
H5: if phrase type is ADJP then select the rightmost adjective, verb, noun, or ADJP
    Example: phrase = "61 years old," content word = "61"
H6: for all other phrase types select the head word
    Example: phrase = "red house," content word = "red"
```

Figure 4–11: List of content word heuristics

task of argument identification gave a significant performance boost to their decision tree-based system.

Named entity of the content word—Certain roles, such as ARGM-TMP and ARGM-LOC, tend to contain TIME or PLACE named entities. This information was added as a set of binary-valued features.

Boolean named entity flags—Surdeanu et al. also added named entity information as a feature. They created indicator functions for each of the seven named entity types: PERSON, PLACE, TIME, DATE, MONEY, PERCENT, ORGANIZATION.

Phrasal verb collocations—This feature comprises frequency statistics related to the verb and the immediately following preposition.

Fleischman, Kwon, and Hovy [119] added the following features to their system:

Logical function—This is a feature that takes three values—external argument, object argument, and other argument—and is computed using some heuristics on the syntax tree.

Order of frame elements—This feature represents the position of a frame element relative to other frame elements in a sentence.

Syntactic pattern—This feature is also generated using heuristics on the phrase type and the logical function of the constituent.

Previous role—This is a set of features indicating the n^{th} previous role that had been observed/assigned by the system for the current predicate.

Pradhan et al. [120] suggested using the following additional feature variations:

Named entities in constituents—Surdeanu et al. [118] reported a performance improvement on classifying the semantic role of the constituent by using the presence of a named entity in the constituent. Some of these entities, such as location and time, are particularly important for the adjunctive arguments ARGM-LOC and ARGM-TMP. Entity tags should also help in cases where the head words are not common or for a closed set of locative or temporal cues, such as in *Mexico*, or in *2003*. They also tagged seven named entities in the corpus using IdentiFinder [121] and added them via seven binary features. Each of these features is true if its respective type of named entity is contained in the constituent.

Verb sense information—The arguments that a predicate can take depend on the sense of the predicate. Each predicate tagged in the PropBank corpus is assigned a separate set of arguments depending on the sense in which it is used. This is also known as the **frameset ID**. Table 4–3 illustrates the argument sets for a word. Depending on the sense of the predicate *talk*, either ARG1 or ARG2 can identify the *hearer*. Absence of this information can be potentially confusing to the learning mechanism.

Verb sense information extracted from PropBank is added by treating each sense of a predicate as a distinct predicate, which helps performance. The disambiguation of PropBank framesets can be performed at a very high accuracy [122].

Table 4–3: Argument labels associated with the two senses of predicate *talk* in PropBank corpus

talk.01		talk.02	
Tag	Description	Tag	Description
ARG0	Talker	ARG0	Talker
ARG1	Subject	ARG1	Talked to
ARG2	Hearer	ARG2	Secondary action

Noun head of prepositional phrases—Many adjunctive arguments, such as temporals and locatives, occur as prepositional phrases in a sentence, and the head words of those phrases, which are always prepositions, often are not very discriminative. For instance, *in the city* and *in a few minutes* both share the same head word *in*, and neither contains a named entity, but the former is ARGM-LOC, whereas the latter is ARGM-TMP. Therefore, Pradhan et al. [120] replaced the head word of a prepositional phrase with that of the first noun phrase inside the prepositional phrase. The preposition information was retained by appending it to the phrase type; for example, the head word of the prepositional phrase *for about 20 minutes* was originally the preposition *for*. After the transformation, the head word was changed to *minutes*, and the phrase PP was changed to PP-FOR. The head word was used in its surface form as well as a lemmatized form. The lemmatization was performed automatically using the XTAG morphology database [123].[8]

First and last word/POS in constituent—Some arguments tend to contain discriminative first and last words, so these were used along with their part of speech as four new features.

Ordinal constituent position—This feature avoids false positives where constituents far away from the predicate are spuriously identified as arguments. It is a concatenation of the constituent type and its ordinal position from the predicate.

Constituent tree distance—This is a finer way of specifying the already present position feature, where the distance of the constituent from the predicate is measured in terms of the number of nodes that need to be traversed through the syntax tree to go from one to the other.

Constituent relative features—These are nine features representing the constituent type, head word and head word part of speech of the parent, and left and right siblings of the constituent in focus. These were added on the intuition that encoding the tree context this way might add robustness and improve generalization.

Temporal cue words—Several temporal cue words are not captured by the named entity tagger and were therefore added as binary features indicating their presence.

8. ftp://ftp.cis.upenn.edu/pub/xtag/morph-1.5/morph-1.5.tar.gz

The BOW (Bag of words) toolkit[9] was used to identify words and bigrams that had highest average mutual information with the ARGM-TMP argument class.

Dynamic class context—In the task of argument classification, these are dynamic features that represent the hypotheses of, at most, the previous two non-null nodes belonging to the same tree as the node being classified.

Path generalizations—As will be seen in Section 4.5.2, for the argument identification task, the path is one of the most salient features. However, it is also the most data-sparse feature. To overcome this problem, the path was generalized in several different ways:

Clause-based path variations—Position of the clause node (S, SBAR) seems to be an important feature in argument identification [124]. Accordingly, Pradhan et al. [120] experimented with four clause-based path feature variations.

- Replacing all the nodes in a path other than clause nodes with an (*). For example, the path NP↑S↑VP↑SBAR↑NP↑VP↓VBD becomes NP↑S↑*S↑*↑*↓VBD. SBAR is replaced with S.
- Retaining only the clause nodes in the path, which for the above example would produce NP↑S↑S↓VBD.
- Adding a binary feature that indicates whether the constituent is in the same clause as the predicate,
- Collapsing the nodes between S nodes, which gives NP↑S↑NP↑VP↓VBD.

Path n-grams—This feature decomposes a path into a series of trigrams. For example, the path NP↑S↑VP↑SBAR↑NP↑VP↓VBD becomes NP↑S↑VP, S↑VP↑SBAR, VP↑SBAR↑NP, SBAR↑NP↑VP, and so on. Shorter paths were padded with nulls.

Single-character phrase tags—Each phrase category is clustered to a category defined by the first character of the phrase label.

Path compression—Compressing sequences of identical labels into following the intuition that successive embedding of the same phrase in the tree might not add additional information.

Directionless path—Removing the direction in the path, thus making insignificant the point at which it changes direction in the tree.

Partial path—Using only that part of the path from the constituent to the lowest common ancestor of the predicate and the constituent. For example, the partial path for the path illustrated in Figure 4-10 is NP↑S.

Another work that deals with paths in an orthogonal fashion is that of Vickrey and Koller [125]. They perform a rule-based sentence simplification in an attempt to automatically accrue path generalizations.

9. http://www.cs.cmu.edu/~mccallum/bow/

Predicate context—This feature captures predicate sense variations. Two words before and two words after were added as features. The part of speech of the words were also added as features.

Punctuations—For some adjunctive arguments, punctuation plays an important role. This set of features captures whether punctuation appears immediately before and after the constituent.

Feature context—Features of constituents that are parent or siblings of the constituent being classified were found useful. Traditionally, each constituent is classified independently; however, in reality, there is a complex interaction between the types and number of arguments that a constituent can have. In other words, the classification of each argument is dependent on the classifications of other nodes. As we will see later, the method of Pradhan et al. performs a postprocessing step using the argument sequence information, but that does not cover all possible constraints. One way of trying to capture those best in the current architecture would be to take into consideration the feature vector compositions of all the non-null constituents for the sentence. This is exactly what this feature does. It uses all the other feature vector values of the constituents that are likely to be non-null, as an added context.

Combinatory Categorial Grammar (CCG) As we learned, although the path feature is very important for the argument identification task, it is one of the most sparse features and may be difficult to train or generalize [126, 127]. A dependency grammar should generate shorter paths from the predicate to dependent words in the sentence and could be a more robust complement to the phrase structure grammar paths extracted from the PSG parse tree. Gildea and Hockenmaier [128] report that using features extracted from a CCG representation improves semantic role labeling performance on core arguments (ARG0-5). Because CCG trees are binary trees and the constituents have poor alignment with the semantic arguments of a predicate, the researchers performed experiments using head words rather than the entire span. Later, Pradhan et al. (2005) [129] used these features to augment the original phrase structure tree-based algorithm to accrue further benefits.

Figure 4-12 shows the CCG parse of the sentence *London denied plans on Monday*. Gildea and Hockenmaier [128] introduced three features:

Phrase type—This is the category of the maximal projection between the two words, the predicate and the dependent word.

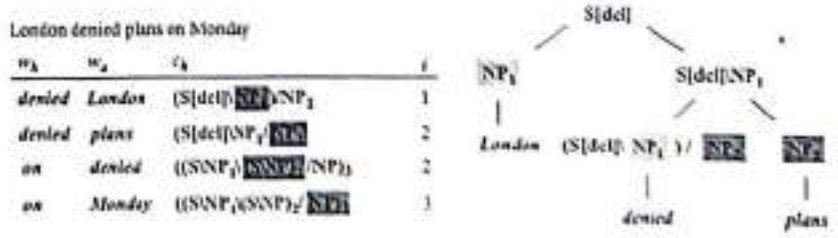


Figure 4-12: Combinatory categorial grammar parse

Categorial path—This is a feature formed by concatenating the following three values: (i) category to which the dependent word belongs, (ii) the direction of dependence, and (iii) the slot in the category filled by the dependent word. For example, for the tree in Figure 4–12, the path between *denied* and *plans* would be (S[dcl]\NP)/NP.2.←.

Tree Path—This is the categorial analogue of the path feature in the Charniak parse-based system, which traces the path from the dependent word to the predicate through the binary CCG tree.

Tree-Adjoining Grammar (TAG) Chen and Rambow [130] report results using two different sets of features: (i) surface syntactic features much like the Gildea and Palmer [131] system and (ii) additional features that result from the extraction of a TAG from the Penn Treebank. They chose a TAG because of its ability to address long-distance dependencies in text. The additional features they introduced are:

Supertag path—This feature is the same as the path feature seen earlier except that in this case it is derived from a TAG rather than from a PSG.

Supertag—This feature is the tree frame corresponding to the predicate or the argument.

Surface syntactic role—This feature is the surface syntactic role of the argument.

Surface subcategorization—This feature is the subcategorization frame.

Deep syntactic role—This feature is the deep syntactic role of an argument, whose values include *subject* and *direct object*.

Deep subcategorization—This is the deep syntactic subcategorization frame. For example, for a transitive verb, it would be NP0_NP1. When available, the preposition modifying the NP is used to lexicalize the feature. So, in case of the predicate *load*, a possible frame would be NP0_NP1_NP2(into).

Semantic subcategorization—Gildea and Palmer also used a semantic subcategorization frame where, in addition to the syntactic categories, the feature includes semantic role information.

While many researchers were constructing various features manually, Moschitti, Pighin, and Basili [132] tried a different approach. They used a tree kernel that identified and selected subtree patterns from a large number of automatically generated patterns to capture the tree context. Unfortunately, for this application, the performance was somewhat inferior to the manually selected features. It is possible that for some other machine learning problem where handcrafting of features is not as intuitive, this technique could prove valuable.

Dependency Trees One issue in the formulations so far is that the performance of a system depends on the exact span of the arguments annotated according to the constituents in the Penn Treebank. Labels are scored as correct only if they match the PropBank annotation exactly; both the bracketing and the label must match. Because PropBank and most syntactic parsers are developed on the Penn Treebank corpus and are therefore based on the same syntactic structures, they would be expected to match the PropBank labeling

better than the other representations. But does a better score here imply that the output is more usable for any applications built on the role labels? It may often be the case that the specific bracketing is not really important; rather, the critical information is the relation of the argument head word to the predicate. Scoring the output of the algorithm using this strategy gives a much higher performance with an F-score of about 85 (vs. 79).

Hacioglu [133] formulated the problem of semantic role labeling on a dependency tree by converting the Penn Treebank trees to a dependency representation using a script by Hwa, Lopez, and Diab [134] and creating a dependency structure labeled with PropBank arguments. The performance on this system seemed to be about 5 F-score points better than the one trained on the phrase structure trees. One possible shortcoming of this and other approaches is that all the parsers are trained on the same Penn Treebank, and when evaluated on sources other than WSJ, seem to degrade in performance. Pradhan et al. [129] experimented to find how well a rule-based dependency parser might fare. Minipar [135, 136] is the rule-based dependency parser that was used. It outputs dependencies between a word called **head** and another called **modifier**. Each word can modify at most one word. The dependency relationships form a dependency tree. The set of words under each node in Minipar's dependency tree form a contiguous segment in the original sentence and correspond to the constituent in a constituent tree. Figure 4-13 shows how the arguments of the predicate *kick* map to the nodes in a phrase structure grammar tree as well as the nodes in a Minipar parse tree. The nodes that represent head words of constituents are the targets of classification. They used the same features as Hacioglu [133] (see Table 4-4).

Minipar performance on the PropBank corpus is substantially worse than the Charniak-based system (47.2 if computed using the strict span criteria). This is expected, as Minipar is not designed to produce constituents that exactly match the constituent segmentation

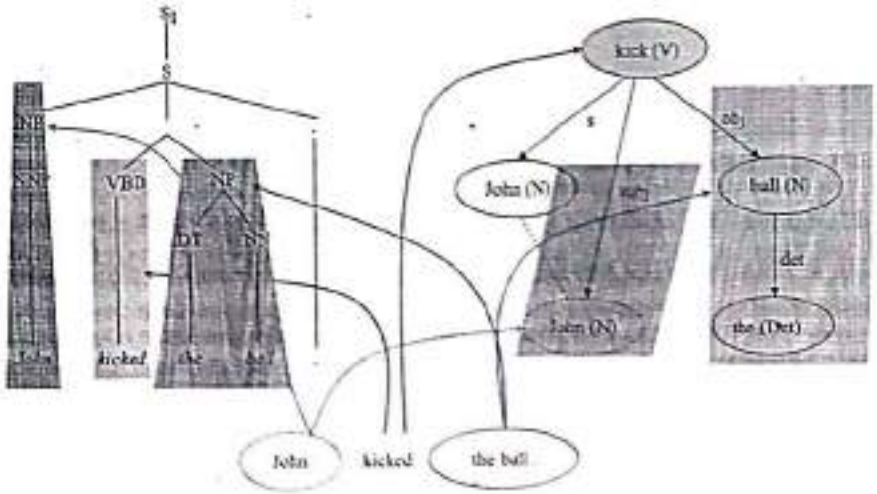


Figure 4-13: New architecture

Table 4–4: Features used in the baseline system using Minipar parses

Head word	The word representing the node in the dependency tree.
Head word POS	Part of speech of the head word.
POS path	The path from the predicate to the head word through the dependency tree connecting the part of speech of each node in the tree.
Dependency path	Each word that is connected to the head word has a dependency relationship to the word. These are represented as labels on the arc between the words. This feature comprises the dependencies along the path that connects two words.
Voice	Voice of the predicate.
Position	Whether the node is before or after the predicate.

used in the Penn Treebank. In experiments reported by Hacioglu [133], a mismatch of about 8% was introduced in the transformation from Treebank trees to dependency trees. Using an errorful automatically generated tree, a still higher mismatch would be expected. In the case of the CCG parses, as reported by Gildea and Hockenmaier [128], the mismatch was about 23%. A more realistic way to score the performance is to score tags assigned to head words of constituents, rather than considering the exact boundaries of the constituents. This score turns out to be an F-score of about 61.7, which is much better and provides orthogonal benefits. These results are a compelling argument for the integration of dependency trees with phrase structure predicate-argument structure.

Since then, there was significant work done on dependency parsing, which led to a series of experiments in this vein. Two Computational Natural Language Learning (CoNLL) shared tasks [137, 138] were held to further research ways to combine dependency parsing and semantic role labeling. These included the use of a richer syntactic dependency representation by Johansson and Nugues [139], which considers tree gaps and traces, and mapping PropBank predicates and arguments to that representation.

Base Phrase Chunks A common question is, How much does the full syntactic representation help the task of semantic role labeling? In other words, how important is it to create a full syntactic tree before classifying the arguments of a predicate? A chunk representation can be faster and more robust to phrase reordering as happens in speech data. Gildea and Palmer [131] explored this question using a chunk-based approach and concluded that syntactic parsing helps fill a big gap. Hacioglu [124] further experimented with a chunk-based semantic labeling method and reached a somewhat more optimistic conclusion. Punyakanok, Roth, and Yih [140] also reported experiments in chunking. Generally, chunk-based systems classify each base phrase as the B(eginning) of a semantic role, I(nside) a semantic role, or O(utside) any semantic role (i.e., null). This is referred to as an IOB representation [141]. This system uses SVM classifiers to first chunk input text into flat chunks or base phrases, each labeled with a syntactic tag. A second SVM is trained to assign semantic labels to the chunks. Figure 4-14 shows a schematic of the chunking process. Table 4-5 lists the features used by the semantic chunker.

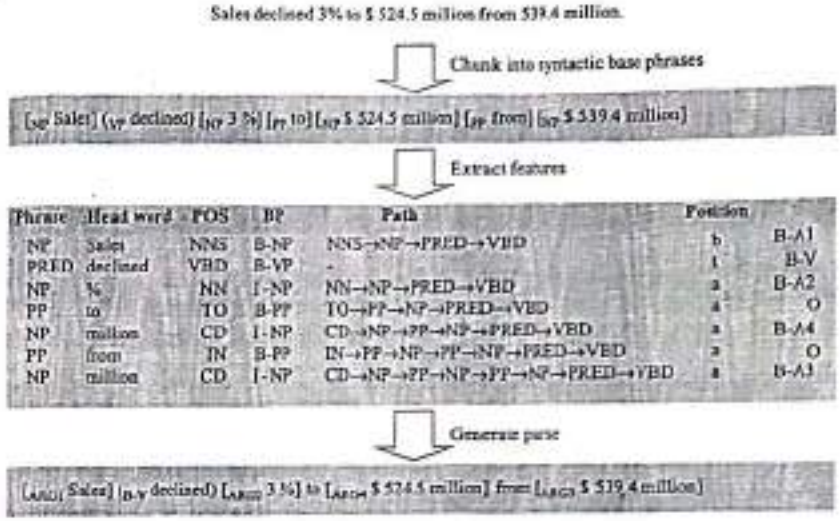


Figure 4–14: Semantic chunker

For each token (base phrase) to be tagged, a set of features is created from a fixed-size context that surrounds each token. In addition to the above features, the chunker uses previous semantic tags that have already been assigned to the tokens contained in the linguistic context. A five-token sliding window is used for the context. The performance on the task of identification and classification was an F-score of about 70.

Classification Paradigms

In the previous section, we looked at sentence-level structural representations that have been used to tackle the problem of semantic role labeling and the features in each of those representations that were identified to capture those characteristics that would help train a model to identify them automatically. In this section, we focus on the ways in which machine learning has been brought to bear on the problem. The approaches range widely in their complexity. The simplest approaches are those that view semantic role labeling as a pure classification problem, where each argument of a predicate may be classified independently of the others. Other researchers have adopted this basic paradigm but added a simple postprocessor that removes implausible analysis, such as when two arguments overlap. A few more complicated approaches augment the postprocessing step to use argument-specific language models or frame-element group statistics. These postprocessors tend to take care of a significant portion of the problems introduced by the original independence assumption.

There have also been more sophisticated approaches to performing joint decoding of all the arguments, trying to capture the arguments' interdependence. Unfortunately, these

Table 4–5: Features used by chunk-based classifier

Words	Words in the chunk.
Predicate lemma	The predicate lemma.
POS tags	Part of speech of the words in the chunk.
BP positions	The position of a token in a base phrase (BP) using the IOB2 representation (e.g., B-NP, I-NP, O).
Clause tags	The tags that mark token positions in a sentence with respect to clauses.
Named entities	The IOB tags of named entities.
Token position	The position of the phrase with respect to the predicate. It has three values: "before," "after," and "-" (for the predicate).
Path	Defines a flat path between the token and the predicate.
Clause bracket patterns	
Clause position	A binary feature that identifies whether the token is inside or outside the clause containing the predicate.
Headword suffixes	Suffixes of head words of length 2, 3, and 4.
Distance	Distance of the token from the predicate as a number of base phrases and the distance as the number of VP chunks.
Length	The number of words in a token.
Predicate POS tag	The part of speech category of the predicate.
Predicate frequency	Frequent or rare using a threshold of 3.
Predicate BP context	The chain of BP's centered at the predicate within a window of size $-2/+2$.
Predicate POS context	POS tags of words immediately preceding and following the predicate.
Predicate-argument frames	Left and right core argument patterns around the predicate.
Number of predicates	This is the number of predicates in the sentence.

sophisticated approaches have so far yielded only slight gains, partly because the performance with a pure classifier followed by a simple postprocessor is already quite high. In this section, rather than provide detailed descriptions of all previous approaches, we concentrate on a current high-performing approach that effectively utilizes multiple knowledge sources and uses a combined architecture that degrades gracefully when run on out-of-genre text.

To begin with, Gildea and Jurafsky's [113] variation of the SRL algorithm involves two steps. In the first step, the system calculates maximum likelihood probabilities that the constituent is an argument, based on those two features, P(argument|Path, Predicate) and P(argument|Head, Predicate), and interpolates them to generate the probability that the constituent under consideration represents an argument. In the second step, it assigns each constituent that has a nonzero probability of being an argument a normalized probability calculated by interpolating distributions conditioned on various sets of features, and selects

Table 4–6: Distributions used for semantic argument classification, calculated from the features extracted from a Charniak parse

Distributions
P(argument|Predicate)
P(argument|Phrase Type, Predicate)
P(argument|Phrase Type, Position, Voice)
P(argument|Phrase Type, Position, Voice, Predicate)
P(argument|Phrase Type, Path, Predicate)
P(argument|Phrase Type, Path, Predicate, Subcategorization)
P(argument|Head Word)
P(argument|Head Word, Predicate)
P(argument|Head Word, Phrase Type, Predicate)

Table 4–7: Argument classification using same features but different classifiers

Classifier	Accuracy (%)
SVM (Pradhan et al.) [120]	88
Decision Tree (Surdeanu et al.) [118]	79
Gildea and Palmer [131]	77

the most probable argument sequence. Some of the distributions they used are shown in Table 4–6.

Surdeanu et al. [118] used a decision tree classifier algorithm, C5 [142, 143] on the same features as Gildea and Jurafsky [113]. The built-in boosting capabilities of this classifier gave a slight improvement in performance. Chen and Rambow [130] also report results using a decision tree classifier, C4.5 [142]. Fleischman and Hovy [144] report results on the FrameNet corpus using a maximum entropy framework. Pradhan et al. [120] used SVM for the same and got even better performance on the PropBank corpus. Nonetheless, the difference between a maximum entropy classifier and SVM turned out to be small.

Pradhan et al. [120] report results using the same set of features on the same data and compare the classifiers with each other. Gildea and Palmer's [131] system estimates the posterior probabilities using several different feature sets and interpolates the estimates—exactly like the Gildea and Jurafsky [113] system—whereas Surdeanu et al. [118] use a decision tree classifier. Table 4–7 shows compares the three systems for the task of argument classification.

For our discussion, we use TinySVM[10] along with YamCha[11] as the SVM training and test software [145, 146]. The SVM parameters, such as the type of kernel, and the values of various parameters was empirically determined using the development set. A polynomial

10. http://chasen.org/~taku/software/TinySVM/
11. http://chasen.org/~taku/software/yamcha/

kernel of degree 2 was selected with the cost per unit violation of the margin $C = 1$ and tolerance of the termination criterion $\epsilon = 0.001$.

SVMs perform well on text classification tasks where data are represented in a high-dimensional space using sparse feature vectors [147, 148]. Inspired by the success of using SVMs for tagging syntactic chunks [145], Pradhan et al. [149, 125, 120] formulated the semantic role labeling problem as a multiclass classification problem using SVMs.

SVMs are inherently binary classifiers, but multiclass problems can be reduced to a number of binary-class problems using either the pairwise approach or the one versus all (OVA) approach [150]. For an N class problem, in the pairwise approach, a binary classifier is trained for each pair of the possible $\frac{N(N-1)}{2}$ class pairs, whereas, in the OVA approach, N binary classifiers are trained to discriminate each class from a metaclass created by combining the rest of the classes. Comparing these two approaches, there is a trade-off between the number of classifiers to be trained and the data used to train each classifier. Although some experiments have reported that the pairwise approach outperforms the OVA approach [151], Pradhan et al.'s [120] initial experiments show better performance for OVA. Therefore, they chose the OVA approach.

SVM outputs the distance of a feature vector from the maximum margin hyperplane. To facilitate probabilistic thresholding and generate an n-best hypotheses lattice, they convert the distances to probabilities by fitting a sigmoid to the scores, as described in Platt [152].

The system can be viewed as comprising two stages: the training stage and the testing stage. We first discuss how the SVM is trained for this task. Because the training time taken by SVMs scales exponentially with the number of examples, and about 90% of the nodes in a syntactic tree have null argument labels, it is efficient to divide the training process into two stages:

1. Filter out the nodes that have a very high probability of being null. A binary null/non-null classifier is trained on the entire dataset. Fit a sigmoid function to the raw scores to convert the scores to probabilities, as described by Platt [152]. All the training examples are run through this classifier, and the respective scores for null and non-null assignments are converted to probabilities using the sigmoid function. Nodes that are most likely null (probability >0.90) are pruned from the training set. This reduces the number of null nodes by about 90% and the total number of nodes by about 80%. This is accompanied by a negligible (about 1%) pruning of nodes that are non-null.

2. The remaining training data are used to train OVA classifiers for all the classes along with a null class.

With this strategy, only one classifier (null or non-null) has to be trained on all of the data. The remaining OVA classifiers are trained on the nodes passed by the filter (approximately 20% of the total), resulting in a considerable savings in training time.

In the testing stage, all the nodes are classified directly as null or one of the arguments using the classifier trained in step 2. They observe a slight decrease in recall if we filter the test examples using a null/non-null OVA classifier in a first pass, as we do in the training process. This small performance gain is obtained at little or no cost of computation because SVMs are very fast in the testing phase. Pseudocode for the testing algorithm is shown earlier in Figure 4-3. A variation in this strategy would be to filter all the examples that

are null in the first pruning stage instead of just pruning out the high-probability ones. This strategy, however, has a statistically significant performance degradation associated with it.

On gold-standard Treebank parses, the performance of such a system on the combined task of argument identification and classification is in the low 90s, whereas on automatically generated parses, the performance tends to be in the high 70s.

Overcoming the Independence Assumption

As mentioned earlier, various postprocessing stages have been proposed to overcome the limitations of treating semantic role labeling as a series of independent argument classification steps. We now look at some of these strategies.

Disallowing Overlaps Since each constituent is classified independently of the other, it is possible that two constituents that overlap get assigned an argument type. Because we are dealing with parse tree, nodes overlapping in words always have an ancestor-descendant relationship, and therefore the overlaps are restricted to subsumptions only as shown in Example 4-1.

EXAMPLE 4-1: But [ARG0 nobody] [predicate knows] [ARG1 at what level [ARG1 the futures and stocks will open today]]

This is a problem because overlapping arguments were not allowed in PropBank (or, more specifically, any two arguments of a verb predicate even in FrameNet cannot overlap). One way to deal with this issue is to choose among overlapping constituents by retaining the one for which the SVM has the highest confidence based on the classification probabilities and labeling the others Null. The probabilities obtained by applying the sigmoid function to the raw SVM scores are used as the measure of confidence.

Argument Sequence Information Another way makes use of the fact that a predicate is likely to instantiate a certain set of argument types as done by Gildea and Jurafsky (2002) [113] to improve the performance of their statistical argument tagger. A similar but more principled approach involves imposing additional constraints in which argument ordering information is retained and the predicate is considered as an argument and is part of the sequence. This can be achieved by training a trigram language model on the argument sequences by first converting the raw SVM scores to probabilities, as described earlier. Then, for each sentence being parsed, an argument lattice is generated using the n-best hypotheses for each node in the syntax tree. Then a Viterbi search is performed through the lattice using the probabilities assigned by the sigmoid as the observation probabilities along with the language model probabilities, to find the maximum likelihood path through the lattice such that each node is assigned a value belonging to the PropBank arguments or null.

The search is constrained in such a way that no two non-null nodes overlap. To simplify the search, Pradhan et al. [120] allowed only null assignments to nodes having a null likelihood above a threshold. While training the language model, we can use the actual predicate to estimate the transition probabilities in and out of the predicate, or we can perform a joint estimation over all the predicates. Pradhan et al. found that there was an improvement in the core arguments accuracy on the combined task of identifying and assigning semantic

arguments, whereas the accuracy of the adjunctive arguments slightly deteriorated. This seems logical considering that the adjunctive arguments have looser constraints on their ordering and even their quantity. It is therefore beneficial to use this strategy only for the core arguments. Some other strategies have been used to incorporate argument context information. Toutanova, Haghighi, and Manning [153] report performance on using a more global model to predict semantic roles for a given predicate using log-linear models, whereas Punyakanok et al. [154] use an integer linear programming-based inference framework to improve the performance of semantic role labeling. Both show small gains over the Viterbi approach.

Feature Performance

Not all features are equally useful in each task. Some features add more noise than information in one context than in another, and features can vary in efficacy depending on the classification paradigm in which they are used. Table 4–8 shows the effect each feature has on the argument classification and argument identification tasks when added individually to the baseline. Addition of named entities to the null/non-null classifier degraded its performance in this particular configuration of classifier and features. This effect can be attributed to a combination of two things: (i) a significant number of constituents contain named entities but are not arguments of a predicate (the parent of an argument node also contains the same named entity), resulting in a noisy feature for null/non-null classification; and (ii) SVMs don't seem to handle irrelevant features very well [155]. When this feature was solely used in the task of classifying constituents known to represent arguments, using features extracted from Treebank parses, the overall classification accuracy increased from 87.9% to 88.1%, whereas adding head word POS as a feature significantly improves both the argument classification and the argument identification tasks.

Feature Salience

In analyzing the performance of the system, it is useful to estimate the relative contribution of the various feature sets used. Table 4–9 shows the argument classification accuracies for combinations of features on the training and test set for all PropBank arguments, using Treebank parses.

In the upper part of Table 4–9 we see the degradation in performance by leaving out one feature at a time. The features are arranged in the order of increasing salience. Removing all head word-related information has the most detrimental effect on performance. The lower part of the table shows the performance of some feature combinations by themselves. Table 4–10 shows the feature salience on the task of argument identification. As opposed to the argument classification task, where removing the path has the least effect on performance, on the task of argument identification, removing the path causes the convergence in SVM training to be very slow and has the most detrimental effect on performance.

Feature Selection

The fact that adding the named entity features to the null/non-null classifier had a detrimental effect on the performance of the argument identification task, while the same feature set showed significant improvement to the argument classification task, indicates that a feature

Table 4–8: Effect of each feature on the argument classification task and argument identification task when added to the baseline system. An asterisk indicates that the improvement is statistically significant

FEATURES	ARGUMENT CLASSIFICATION	ARGUMENT IDENTIFICATION		
	A	P	R	F_1
Baseline [120]	87.9	93.7	88.9	91.3
+ Named entities	88.1	93.3	88.9	91.0
+ Head POS	* 88.6	94.4	90.1	* 92.2
+ Verb cluster	88.1	94.1	89.0	91.5
+ Partial path	88.2	93.3	88.9	91.1
+ Verb sense	88.1	93.7	89.5	91.5
+ Noun head PP (only POS)	* 88.6	94.4	90.0	* 92.2
+ Noun head PP (only head)	* 89.8	94.0	89.4	91.7
+ Noun head PP (both)	* 89.9	94.7	90.5	* 92.6
+ First word in constituent	* 89.0	94.4	91.1	* 92.7
+ Last word in constituent	* 89.4	93.8	89.4	91.6
+ First POS in constituent	88.4	94.4	90.6	* 92.5
+ Last POS in constituent	88.3	93.6	89.1	91.3
+ Ordinal const. pos. concat.	87.7	93.7	89.2	91.4
+ Const. tree distance	88.0	93.7	89.5	91.5
+ Parent constituent	87.9	94.2	90.2	* 92.2
+ Parent head	85.8	94.2	90.5	* 92.3
+ Parent head POS	* 88.5	94.3	90.3	* 92.3
+ Right sibling constituent	87.9	94.0	89.9	91.9
+ Right sibling head	87.9	94.4	89.9	* 92.1
+ Right sibling head POS	88.1	94.1	89.9	92.0
+ Left sibling constituent	* 88.6	93.6	89.6	91.6
+ Left sibling head	86.9	93.9	86.1	89.9
+ Left sibling head POS	* 88.8	93.5	89.3	91.4
+ Temporal cue words	* 88.6	-	-	-
+ Dynamic class context	88.4	-	-	-

selection strategy would be beneficial. One strategy would be to perform a leave-one-out experiment whereby one feature is left out of the full set of features at a time, and depending on the degradation in performance, it is either kept or pruned out. This is a very naïve feature-selection strategy that assumes the features are independent of each other. We could use a more complicated feature-selection strategy instead. One downside of feature selection based on each argument type in the SVM paradigm is that SVMs output distances, not probabilities. These distances may not be comparable across classifiers, especially if different features are used to train each binary classifier. A solution is to use the algorithm described by Platt [152] to convert the SVM scores into probabilities by fitting to a sigmoid. Foster and Stine [156] show that the pool-adjacent-violators (PAV) algorithm [157] provides a better

Table 4–9: Performance of various feature combinations on the task of argument classification

FEATURES	ACCURACY
All features [120]	91.0
All except Path	90.8
All except Phrase Type	90.8
All except HW and HW-POS	90.7
All except All Phrases	*83.6
All except Predicate	*82.4
All except HW and FW and LW info.	*75.1
Only Path and Predicate	74.4
Only Path and Phrase Type	47.2
Only Head Word	37.7
Only Path	28.0

Table 4–10: Performance of various feature combinations on the task of argument identification

FEATURES	P	R	F_1
All features [120]	96.2	92.5	93.8
All except HW	96.1	92.3	93.7
All except Predicate	94.5	91.9	93.2
All except HW and FW and LW info.	91.8	88.5	*90.1
All except Path and Partial Path	88.4	88.9	*88.6
Only Path and HW	88.5	84.3	86.3
Only Path and Predicate	89.3	81.2	85.1

method for converting raw classifier scores to probabilities when Platt's algorithm fails. The probabilities resulting from either conversion may not be properly calibrated, in which case the probabilities can be binned, and a warping function can be trained to calibrate them.

Size of Training Data

One important concern in any supervised learning method is the amount of training examples required for decent performance of a classifier. To check the behavior of this learning problem, Pradhan et al. [129] trained the classifiers on varying amounts of training data. The resulting plots are shown in Figure 4–15. The first curve from the top indicates the change in F_1 score on the task of argument identification alone. The third curve indicates the F_1 score on the combined task of argument identification and classification. It can be seen that after about 10,000 examples, the performance starts to plateau, which indicates that simply tagging more data might not be a good strategy. A better strategy is to tag only appropriate new data. Also, the fact that the first and third curves—the first being the

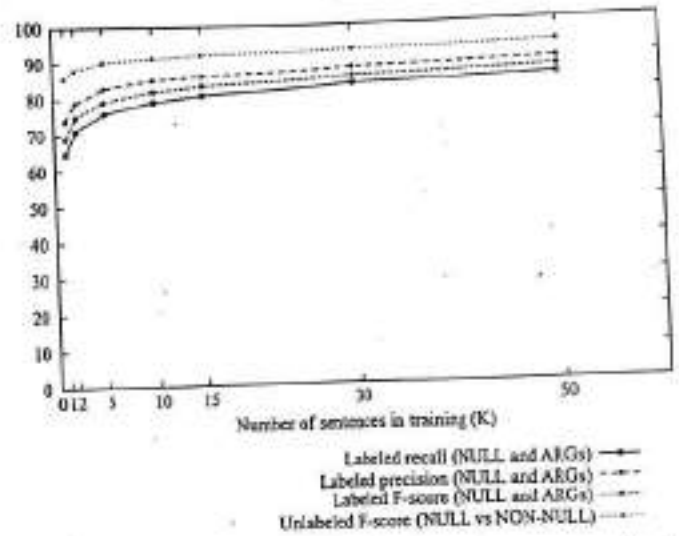


Figure 4–15: Learning curve for the tasks of identifying and classifying arguments using Treebank parses

F-score on the task of argument identification task and the third being the F-score on the combined task of identification and classification—run almost parallel to each other tells us that a constant loss occurs due to classification errors throughout the data range. One way to bridge this gap could be to identify better features.

Overcoming Parsing Errors

After performing a detailed error analysis, Pradhan et al. [129] found that the identification problem poses a significant bottleneck to improving overall system performance. The baseline system's accuracy on the task of labeling nodes known to represent semantic arguments is 90%. On the other hand, the system's performance on the identification task is quite a bit lower, achieving only 80% recall with 86% precision. The two sources of these identification errors are failures by the system to identify all and only those constituents that correspond to semantic roles, *when those constituents are present in the syntactic analysis*, and failures by the syntactic analyzer to provide the constituents that align with correct arguments. Classification performance using Charniak parses is about 3 F-score points worse than when using treebank parses. On the other hand, argument identification performance using Charniak parses is about 12.7 F-score points worse. Half of these errors—about 7 points—are due to missing constituents, and the other half—about 6 points—are due to misclassifications.

This severe degradation in argument identification performance for automatic parses was the motivation for examining two techniques for improving argument identification: combining parses from different syntactic representations and using n-best parses or a parse forest in the same representation.

Multiple Views Pradhan et al. [129] report on experiments that address the problem of arguments missing from a given syntactic analysis. They investigate ways to combine hypotheses generated from semantic role taggers trained using different syntactic views: one trained using the Charniak parser [158]; another on a rule-based dependency parser, Minipar [135]; and a third based on a flat, shallow syntactic chunk representation [159]. They showed that these three views complement each other to improve performance.

Some of the systems we have discussed use features based on syntactic constituents produced by a syntactic parser [149, 126], and others use only a flat syntactic representation produced by a syntactic chunker [160, 159, 124]. The latter approach lacks the information provided by the hierarchical syntactic structure, and the former imposes the limitation that the possible candidate roles should be one of the nodes already present in the syntax tree. Although the chunk-based systems are very efficient and robust, the systems that use features based on full syntactic parses are generally more accurate. Analysis of the source of errors for the parse constituent-based systems showed that incorrect parses were a major source of error. The syntactic parser often did not produce any constituent that corresponded to the correct segmentation for the semantic argument. Pradhan et al. [129] report on a first attempt to overcome this problem by combining semantic role labels produced from different syntactic parses. The hypothesis is that the syntactic parsers will make different errors, and combining their outputs will be an improvement over any one system. This initial attempt used features from the Charniak parser, the Minipar parser, and a chunk-based parser. It did show some improvement from the combination, but the method for combining the information was heuristic and suboptimal. The researchers proposed an improved framework for combining information from different syntactic views. The goal was to preserve the robustness and flexibility of the segmentation of the phrase-based chunker but to take advantage of features from full syntactic parses. They also wanted to combine features from different syntactic parses to gain additional robustness. To this end, they used features generated from the Charniak parser and the Collins parser. The main contribution of combining both the Minipar-based and the Charniak-based semantic role labeler was significantly improved performance on ARG1 in addition to slight improvements to some other arguments. See Figure 4–16.

The semantic parses were combined as follows. Scores for arguments were converted to calibrated probabilities, and arguments with scores below a threshold value were deleted. Separate thresholds were used for each semantic role labeler. For the remaining arguments, if any set of arguments overlapped, the least probable among them were removed until no overlaps remained. In the chunk-based system, an argument could consist of a sequence of chunks. The probability assigned to the BEGIN tag of an argument was used as the probability of the sequence of chunks forming an argument.

The general framework is to train separate semantic role labeling systems for each of the parse tree views, and then to use the role arguments output by these systems as additional features in a semantic role classifier using a flat syntactic view. The constituent-based classifiers walk a syntactic parse tree and classify each node as null (no role) or as one of the set of semantic roles. As we saw in Section 4.5.2, chunk-based systems classify each base phrase using the IOB representation. The constituent level roles are mapped to the IOB representation used by the chunker. The IOB tags are then used as features for a separate base-phrase semantic role labeler (chunker), in addition to the standard set of features used

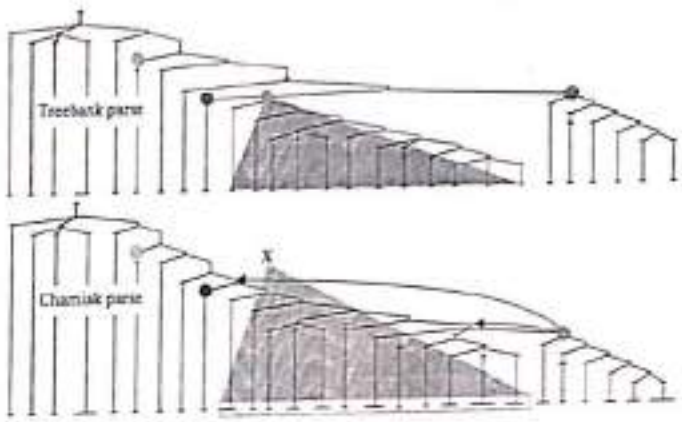


Figure 4–16: Argument deletions owing to parse error

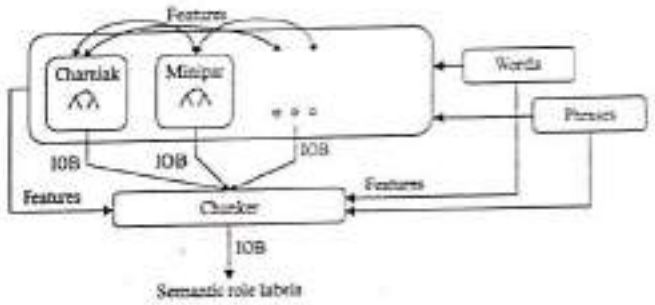


Figure 4–17: New architecture

by the chunker. An n-fold cross-validation paradigm is used to train the constituent-based role classifiers and the chunk-based classifier. See Figure 4–17.

The chunking system for combining all features is trained using a fourfold paradigm. In each fold, separate SVM classifiers are trained using 75% of the training data. The other 25% of the training data are then labeled by each of the systems. Iterating this process four times creates the training set for the chunker. After the chunker is trained, the PSG- and Minipar-based semantic labelers are retrained using all of the training data. Once the retraining is complete, SVMs are trained for begin (B) and inside (I) classes of all arguments and an outside (O) class. One particular advantage of this architecture, as depicted in Figure 4–18, is that the final segmentation does not necessarily have to adhere to one of the input segmentations. Depending on the information provided in terms of features, the classifier can generate a new, better segmentation.

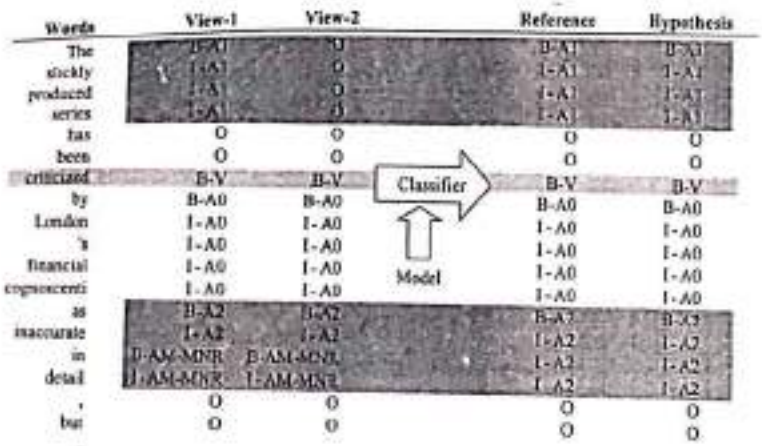


Figure 4–18: Example classification using the new architecture

This is just one way of combining various views. Another combination strategy by Surdeanu et al. [161] also shows improvement over using a single view.

Broader Search Another approach is to broaden the search by selecting constituents in n-best parses [153, 154] or using a packed forest representation [162], which more efficiently represents variations over much larger n. Using a parse forest shows an absolute improvement of 1.2 points over single best parses and 0.5 points over n-best parses.

Noun Arguments

Until now we have only dealt with the task of identification and classification of arguments of verb predicates in a sentence. To generate a sentence-level semantic representation, it is necessary to identify arguments of other possible predicates in a sentence, such as nominal predicates, adjectival predicates, and prepositional predicates. This chapter discusses the task of semantic role labeling as applied to a class of nominal predicates, or nominalizations. A simple linguistic definition of nominalization is "the process of converting a verb into an abstract noun." Take, for example, the two sentence pairs in Figure 4–19, where the second sentence in each pair is a nominalized version of the first. One important thing to note is that the verbs in the nominalized sentences are *make* and *took* respectively. A semantic analyzer that parses these sentences for the arguments of these verbs will miss the events—*complaining* and *walking*—that are central to understanding the meaning of the sentences and that are represented by the nominal predicates *complain* and *walk* respectively.

Of the vast literature on automatic semantic role labeling, very little deals with assigning the semantic role properties of nouns. Of what does, most deal with nominalizations. However, because of the lack of a corpus annotated with nominal predicates and their arguments, there have been no investigations into applying statistical algorithms that can automatically

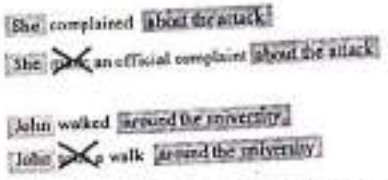


Figure 4–19: Example of nominalization

identify and label the arguments of nouns. To our knowledge, the only works that come close are the rule-based system described by Hull and Gomez [163] and the work of Lapata [164] on interpreting the relationship between the head of a nominalized compound and its modifier noun. With the availability of hand-labeled argument information for nominal predicates from the FrameNet project, an investigation into the feasibility of automatically identifying nominal predicates using this data can be performed.

In this section, we look at the adaptation to nouns of features that were originally derived for verbs; most of these adaptations are quite straightforward. We also investigate how well these transformed features identify the semantic properties of noun arguments. In other words, how good are these features at identifying and classifying nominal arguments? Furthermore, can any new sets of features specific to nominalizations be added to good effect?

Following are some new features that were identified by Pradhan et al. [165] along with a justification. Some of these features don't exist for some constituents. In those cases, the respective feature values are set to UNK. Almost all the new features that we used for the verb predicates, except the CCG features, were added to the baseline system.

Intervening verb features—Support verbs play an important role in realizing the arguments of nominal predicates. Three classes of intervening verbs were used: (i) verbs of being, (ii) light verbs (a small set of verbs such as *make*, *take*, *have*), and (iii) other verbs with part of speech starting with the string VB. Three features were added for each: (i) a binary feature indicating the presence of the verb between the predicate and the constituent, (ii) the actual word as a feature, and (iii) the path through the tree from the constituent to the verb. The following example illustrates the intuition behind these intervening verb features:

[$_{Speaker}$ Leapor] *makes* general [$_{Predicate}$ assertions] [$_{Topic}$ about marriage]

Predicate NP expansion rule—This is the noun equivalent of the verb subcategorization feature used by Gildea and Jurafsky [113]. It represents the expansion rule instantiated by the syntactic parser, for the lowermost NP in the tree, encompassing the predicate. This feature tends to cluster noun phrases with a similar internal structure and thus helps find argument modifiers.

Is predicate plural—This binary feature indicates whether the predicate is singular or plural, as these tend to have different argument selection properties.

Genitives in constituent—This is a binary feature that is true if there is a genitive word (one with the part of speech POS, PRP, PRP$, or WP$) in the constituent, as

these tend to be subject/object markers for nominal arguments. The following example helps clarify this notion:

[Speaker Burma 's] [Phenomenon oil] [Predicate search] hits virgin forests

Verb dominating predicate—The head word of the first VP ancestor of the predicate.

More recently, Jiang and Ng [166] use additional features in a maximum entropy framework to perform argument tagging on the NomBank corpus. Also, NomBank arguments have been added to an integrated syntax-semantic dependency representation in the recent CoNLL evaluations [137, 138].

Multilingual Issues

Because early research on semantic role labeling was performed on English corpora, various core features and learning mechanisms were explored specifically for English. Apparently, most of the core features for English translate well for other languages [167, 168]. Some special cases of language-specific features turned out to be important for a specific language but also improved performance for English systems; for example, the predicate frame feature that was introduced by Xue and Palmer [127] for Chinese also shows some features improvement in English. Some features are so language specific that they have no parallels in English, and those usually are unique to a particular language; for example, Chinese requires a much more complex word segmentation process—something that can be performed quite accurately in English using very simple rules. Therefore, special word segmentation models have to be trained in the case of Chinese before parsing or semantic role labeling can begin.

On the other hand, fortunately, the morphology-poor nature of Chinese blurs the difference between verbs, nouns, and adjectives, forming a closer connection between these predicates and their arguments. This allows the training of a unified model across all types of predicates. Yet another property of Chinese that impacts automatic semantic role labeling is that Chinese has many more verb types than English—at least a factor of four—and so in a similarly sized corpus, the number of instances per verb is much smaller for Chinese, which exacerbates an already critical issue of data sparsity. At the same time, this means that there is less polysemy to deal with, which is a preferred property from a performance perspective. The creation of a specific clustering feature helps overcome this problem to some degree. So, in some sense, although a similar set of features are useful across languages, the specific instantiation of some can differ greatly, and the relative benefit of each varies with language as well. These new features were introduced by Xue [167] to improve the performance of the Chinese semantic role labeling system. Another thing to note about Chinese is that the syntactic parsing performance is inferior to that of English, and so systems based on shallow syntactic chunking [169] achieved quite competitive results with those based on a full syntactic parse.

In contrast to Chinese, a particular characteristic of Arabic is its morphological richness. What this means in parsing is that there are many more syntactic POS categories for Arabic than there are for English or Chinese—almost an order of magnitude. So far, results reported in the literature on Arabic semantic role labeling systems have not exploited its specific morphologically rich nature [168].

Another notable difference from English is that both Arabic and Chinese have many more implicit (or dropped) subjects. The Penn Treebank marks them with a trace, and they are tagged with an argument in PropBank. Unlike English, Chinese and Arabic require the training of special models to identify dropped subjects before the predicate-argument structure can be fully realized [170].

Robustness across Genre

One possible shortcoming of this and other approaches is that all the parsers are trained on the same Penn Treebank, which, when evaluated on sources other than WSJ, seems to degrade in performance. Carreras and Màrquez [171] show that the drop in performance on test data from the Brown corpus was 10 F-score points lower than the test data from WSJ. When trained and tested on WSJ propositions the syntactic parser's performance is the main source of error, and the classification performance is quite good. However, Pradhan, Ward, and Martin [172] report that when we train the system on WSJ data and test on the Brown propositions, the classification performance and the identification performance are affected to the same degree. This provides some evidence that more lexical semantic features are needed to bridge the performance gap across genres. Zapirain [173] shows that incorporating features based on selectional preferences provides one way of effecting more lexico-semantic generalization.

4.5.3 Software

Following is a list of software packages available for semantic role labeling.

- **ASSERT** (Automatic Statistical SEmantic Role Tagger)
  [http://www.cemantix.org/assert.html]
  A semantic role labeler trained on the English PropBank data.
- **C-ASSERT** [http://hlt030.cse.ust.hk/research/Pc-assert/]
  An extension of ASSERT for the Chinese language.
- **SwiRL** [http://www.surdeanu.name/mihai/swirl/]
  Another semantic role labeler trained on PropBank data.
- **Shalmaneser** (A *Shallow* Semantic *Parser*)
  [http://www.coli.uni-saarland.de/projects/salsa/shal/]
  A toolchain for shallow semantic parsing based on the FrameNet data.

4.6 Meaning Representation

We now turn to the third, deeper level of semantic interpretation whose objective is to take natural language input and transform it into an unambiguous representation that a machine can act on. This is the form that would be more likely to be as incomprehensible to humans as it would be comprehensible to machines. We can think of a parallel between programming languages that are much closer to the way humans manipulate information and the low-level machine code that the computer executes. Although compilers and interpreters

impose various specific syntactic and semantic restrictions on a program written in a high-level programming language, no such restrictions are imposed on the form that natural language can take; whereas precision in artificial languages is necessary to define scope and eliminate ambiguity, natural language relies on the recipient to disambiguate it using context and general world knowledge. Researchers have spent decades figuring out how to interpret and/or encode context and use world knowledge so that we can make machines understand what humans seem to understand so effortlessly. However, there is still a lot of progress to be made, and so far the techniques that have been developed only work within specific domains and problems instead of being scalable to arbitrary domains. This is often termed **deep semantic parsing**, as opposed to shallow semantic parsing that comprises word sense disambiguation and semantic role labeling.

4.6.1 Resources

A number of projects have created representations and resources that have promoted experimentation in this area. Let us look at a few of those resources.

ATIS

Though not quite focused on formal knowledge representation, the Air Travel Information System (ATIS) project [174] is considered one of the first concerted efforts to build systems to transform natural language into a representation that could be used by an end application to make decisions. The task involved a machine to transform a user query in spontaneous speech, using a restricted vocabulary, about flight information. It then formed a representation that was compiled into a SQL query, which was used to extract answers from a flight database. A hierarchical frame representation was used to encode the intermediate semantic information. Figure 4-20 shows a sample user query and the corresponding frame representation. The training corpus includes over 774 scenarios completed by 137 subjects, yielding a total of over 7,300 utterances. All utterances are transcribed, and 2,900 of them have been categorized and annotated with canonical reference answers.[12] Roughly 600 of these have also been treebanked.[13]

Communicator

The Communicator program was the follow-on to ATIS. While ATIS was more focused on **user-initiated dialog** (i.e., the user asked questions to the machine, which provided answers), Communicator involved a **mixed-initiative dialog**, whereby the human and machine had a dialog with each other with the computer presenting users with real-time travel information and helping them negotiate a preferred itinerary. Over the period of the program, many thousands of dialogs were collected and are available through the Linguistic Data Consortium. Carnegie-Mellon University collected more data, a portion of which is available for research.[14] Roughly a million words and ~1,600 dialogs have been annotated with dialog acts.[15]

12. http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC93S26
13. http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC99T42
14. http://www.speech.cs.cmu.edu/Communicator/Corpus/
15. http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2002S56

FRAME Representation

```
SHOW:
 FLIGHTS:
  TIME:
   PART-OF-DAY:
  ORIGIN:
   CITY: Boston
  DEST:
   CITY: San Francisco
  DATE:
   DAY-OF-WEEK: Tuesday
```

Natural language representation

Please show me morning flights from Boston to San Francisco on Tuesday

Figure 4-20: A sample user query and its *frame* representation in the ATIS program

GeoQuery

In the domain of U.S. geography, there is a natural language interface (NLI) to a geographic database called Geobase [175], which has about 800 Prolog facts stored in a relational database with geographic information such as population, neighboring states, major rivers, and major cities. Some sample queries and their representations are as follows:

(1) What is the capital of the state with the largest population?
```
answer(C, (capital(S, C), largest(P, (state(S), population(S, P)))))
```

(2) What are the major cities in Kansas?
```
answer(C, (major(C), city(C), loc(C, S), equal(S, stateid(kansas))))
```

This is the GeoQuery corpus, which has also been translated into Japanese, Spanish, and Turkish.

Robocup: CLang

RoboCup (www.robocup.org) is an international initiative by the artificial intelligence community that uses robotic soccer as its domain. There is a special formal language, CLang, which is used to encode the advice from the team coach, and the behaviors are expressed as if-then rules. Following is an example representation in this domain:

(1) If the ball is in our penalty area, all our players except player 4 should stay in our half.
```
((bpos (penalty-area our)) (do (player-except our 4) (pos (half our))))
```

4.6.2 Systems

As we can see in these examples, depending on the consuming application, the meaning representation can be a SQL query, a Prolog query, or a domain-specific query representation.

Now we look at the various ways the problem of mapping the natural language to such meaning representation has been tackled.

Rule Based

Some of the semantic parsing systems that performed very well for both the ATIS and Communicator projects were rule-based systems in the sense that they used an interpreter whose semantic grammar was handcrafted to be robust to speech recognition errors. The underlying philosophy was that the traditional syntactic explanation of a sentence is much more complex than the underlying semantic information, so parsing the meaning units in the sentence into semantics proved to be a better approach. Furthermore, especially in dealing with spontaneous speech, the system has to account for ungrammatical instructions, stutters, filled pauses, and so on. Word order therefore becomes less important, which leads to meaning units scattered in the sentences/utterances and not necessarily in the order that would make sense to a syntactic parser. Ward's [176, 177, 178] system, Phoenix, uses recursive transition networks (RTNs) [179] and a handcrafted grammar to extract a hierarchical frame structure, and reevaluates and adjusts the values of these frames with each new piece of information obtained. This system had an error rate of 13.2% for spontaneous speech input with a speech recognition word-error rate of 4.4%, and a 9.3% error for transcript input.

Supervised

Although rule-based techniques are relatively easy to craft in the beginning and serve a good purpose to formulate solutions to various tasks, they have several downsides: (i) they need some effort upfront to create the rules, (ii) the time and specificity required to write rules usually restricts the development to systems that operate in limited domains, (iii) they are hard to maintain and scale up as the problem becomes more complex and more domain independent, and (iv) they tend to be brittle. The alternative is to use statistical models derived from hand-annotated data. However, unless some hand-annotated data is available, statistical models cannot be used to deal with unknown phenomena. During the ATIS evaluations, some data was hand-tagged for semantic information. Schwartz et al. [180] used this as an opportunity to create what was probably the first end-to-end supervised statistical learning system for the ATIS domain. They had four components in their system: (i) semantic parse, (ii) semantic frame, (iii) discourse, and (iv) backend. This system used a supervised learning approach combined with quick training augmentation through a human-in-the-loop corrective approach to generate slightly lower quality but more data for improved supervision. Miller et al. [181] described the algorithm in more detail. Their system achieved an error rate of 14.5% on the entire test set and 9.5% on the subset of sentences that were context independent. Since then, various improvements have been made, such as by He and Young [182].

Continuing on the line of research that is today commonly known as natural language interface for databases (NLIDB), Zelle and Mooney [183] tackled the task of retrieving answers from a Prolog database by converting natural language questions into Prolog queries

in the domain of GeoQuery. They introduced a system called CHILL (Constructive Heuristics Induction for Language Learning), based on the relational learning techniques of inductive logic programming. It uses a shift-reduce parser to map the input sentence into parses expressed as a Prolog program. They preferred a representation closer to formal logic rather than SQL, because once achieved, it can easily be translated into other equivalent representations. They tested the system performance with the rule-based system Geobase that comes with the GeoQuery dataset over a varying number of queries as inputs. It took CHILL roughly 175 training queries to match the performance of Geobase. Additional queries made it surpass Geobase, achieving an accuracy of 84% on novel queries, at times inducing 1,100 lines of Prolog code.

Since then, advances have been made in machine learning and syntactic parsing, and researchers have identified new approaches and refined existing approaches. The SCISSOR (Semantic Composition that Integrates Syntax and Semantics to get Optimal Representation) system, for example, uses a statistical syntactic parser to create a semantically augmented parse tree (SAPT) [184, 185]. Training for SCISSOR consists of a (natural language, SAPT, meaning representation) triplet. It uses a standard syntactic parser augmented with semantic tags, then a recursive procedure is used to compositionally construct the meaning representation for each node in the tree given the representations of its children. SCISSOR shows significant performance improvement over earlier approaches. KRISP (Kernel-based Robust Interpretation for Semantic Parsing) [186] uses string kernels and SVMs to improve the underlying learning techniques. WASP (Word Alignment-based Semantic Parsing) [187] takes a radical approach to semantic parsing by using state-of-the-art machine translation techniques to learn a semantic parser. Wong and Mooney treat the meaning representation language as an alternative form of natural language and use GIZA++ to produce an alignment between the natural language and a variation of the meaning representation language. Complete meaning representations are then formed by combining these aligned strings using a synchronous CFG (SCFG) framework. SCISSOR is somewhat more accurate than WASP and KRISP, which can themselves benefit from the information in SAPTs [188]. KRISP, CHILL, and WASP have also learned semantic parsers for Spanish, Turkish, and Japanese with similar accuracies. Yet another approach comes from Zettlemoyer and Collins [189], who trained a structured classifier for natural language interfaces by learning probabilistic combinatory categorial grammar (PCCG) along with a log-linear model that represents the distribution over the syntactic and semantic analysis conditioned on the natural language input.

4.6.3 Software

Not a lot of software programs are available for the older, more rule-based systems, but the following are available for download.

- **WASP** [http://www.cs.utexas.edu/~ml/wasp/]
- **KRISPER** [http://www.cs.utexas.edu/~ml/krisp/]
- **CHILL** [http://www.cs.utexas.edu/~ml/chill.html]

4.7 Summary

In this chapter we looked at the problem of semantic parsing through various lenses. There is no silver bullet to the problem of meaning representation and language understanding, so over the years, researchers have come up with tasks that either solve parts of the bigger problem in a more domain-independent fashion or solve the complete problem but for a very restricted domain. The first case, shallow semantic interpretation, deals separately with the four main aspects of language: structural ambiguity (which is syntactic in nature and is the subject of a separate chapter), word sense, entity and event recognition, and predicate-argument structure recognition. The latter three are components of what has widely come to be known as shallow semantic parsing. As we have seen, syntax plays a very important role in this process, and cannot be considered completely divorced from semantics. The second, deep parsing, or semantic parsing, comprises taking natural language input and transforming it into a meaning representation, which tends to be task specific and something that the end application can unambiguously execute.

We learned that developments on various fronts have been made in all these methods. In the early era of the field, few hand-labeled corpora and few well-developed learning techniques were available. Even now, in the case of resource-poor languages, there is not much data to train sophisticated learning algorithms. In these cases, researchers resort to encoding the domain information in a rule-based system, which is usually domain specific. For languages for which there is enough human-annotated data available, more statistical approaches became predominant. Given the sparseness of data even when there is sufficient annotation (any amount of realistic human annotation would never be enough to learn all the various nuances of language), researchers have resorted to semisupervised or unsupervised methods, the latter being usually much less accurate than supervised or rule-based methods.

4.7.1 Word Sense Disambiguation

Word sense disambiguation is an integral part of language understanding. In information retrieval, speech understanding, and applications restricted to certain domains, it has not been very important, because of limited sense usability or implicit disambiguation. However, for applications that deal with a deeper understanding of text, sense disambiguation may be critical. Research in this area started with senses that were defined in dictionaries, because they were the primary resource in the beginning. Lesk's algorithm is generally recognized as the first dictionary-based word sense algorithm, where sense disambiguation is performed using the overlap between the context in which a word appears in the discourse and its dictionary gloss. The creation of *Roget's Thesaurus* led to more English-specific algorithms to classify words into the categories defined in it. The notion of one sense per discourse led to an important semisupervised algorithm: the Yarowsky algorithm. With the advent of a much richer lexicon like WordNet and corpora annotated with senses defined in it (SEMCOR)—interestingly, in parallel with the advances in machine learning—most of the research community shifted to using them as the standard until later studies showed that the granularity of senses in WordNet might be too fine. If humans cannot agree on sense distinctions to a certain degree, then machines should not be expected to either. This led to the folding of word senses in WordNet into coarser units that are more amenable to

human agreement and at the same time provide a better means of achieving high-accuracy automatic disambiguation. WordNet has continued to be an important resource that has significantly improved the field, and it is still used in state-of-the-art disambiguation systems.

In a separate vein, with the growth of the Internet, the availability of resources such as Wikipedia, which served as a surrogate annotation resource, exploiting Internet resources became one of the mainstream pursuits. An increasing number of areas in language understanding are making use of this resource in novel ways. Active learning is another direction that is still probably more an art than a science but has been quite useful for amassing annotations for words that are either rare (low-frequency), high sense perplexity (many senses), or do not have enough annotation for some reason or other, including, but not limited to, low-resource languages [190]. In languages where there was no hand-annotated data available, various unsupervised approaches were developed, some of which exploited the differing sense granularities and instantiations across parallel corpora.

4.7.2 Predicate-Argument Structure

Unlike word sense disambiguation, there were far fewer rule-based systems that tagged thematic roles in text. With the advent of corpora such as FrameNet and PropBank labeled with a predicate-argument structure, there was a giant wave of research focused on building systems to tag these structures in text, primarily for verb and noun predicates. Many new features were introduced in various syntactic frameworks, some of them not even conducting a full syntactic analysis and resorting only to the base phrase chunks. It turns out that, at least for the genres that have treebanks, the contribution of a syntactic parser is invaluable. When lexicalization is the only way in which a syntactic representation is informed by semantics, a semantic role labeler tends to make errors that could be avoided using a more bottom-up approach. Also, using richer features in the first pass can be prohibitive, so a combination of producing n-best hypotheses and reranking them based on a more global feature set is the approach that generally performs better. Furthermore, a combination of a top-down and bottom-up approach by combining the information from various syntactic and nonsyntactic views improves performance as well. One big bottleneck at present is that performance on genres of text that exhibit even somewhat different syntactic styles, word usage, or entity and event structures tends to be much worse than if you have matched training and test corpora. We are at a point where structural information has been utilized to a significant degree to the benefit of semantic analysis, but the lexical- and sense-level generalization is significantly lacking, thereby making the existing approaches much less robust across genres or domains of text. We also saw that the fundamental techniques developed for English—which happens to be the language for which the hand-tagged corpora were first created—translate very well to other languages. Of course, each new language has its own idiosyncrasies that lead to the identification of new features. These new features may in turn improve the original English system. Many annotation efforts are underway across the world, and we have much more to learn.

4.7.3 Meaning Representation

Finally, we looked at meaning representation. This is a much less researched topic and especially so across languages. Meaning representation is the process of converting natural

language input into a format that is unambiguous and easily understandable by a machine or end application, which can take actions based on the input. So far, there is no one universally accepted representation, so these systems and their representations tend to be domain specific.

New research programs are stretching the possibilities of existing techniques and creating novel ones that will eventually let us bring these pieces together into a richer, deeper representation that would also be independent of domain.

Bibliography

[1] N. Chomsky, *Syntactic Structures*. The Hague: Mouton, 1957.

[2] J. Katz and J. Fodor, "The structure of a semantic theory," *Language*, vol. 39, pp. 170–210, 1963.

[3] V. H. Yngve, "Syntax and the problem of multiple meaning," in *Machine Translation of Languages* (W. N. Locke and A. D. Booth, eds.), pp. 208–226, Cambridge, MA: MIT Press, 1955.

[4] N. Ide and J. Véronis, "Introduction to the special issue on word sense disambiguation: The state of the art," *Computational Linguistics*, vol. 24, no. 1, pp. 2–40, 1998.

[5] E. Agirre and P. Edmonds, eds., *Word Sense Disambiguation: Algorithms and Applications*. Dordrecht: Springer, 2006.

[6] M. Palmer, H. Dang, and C. Fellbaum, "Making coarse-grained and fine-grained sense distinctions, both manually and automatically," *Natural Language Engineering Journal*, vol. 13, no. 2, pp. 137–163, 2007.

[7] P. Resnik and D. Yarowsky, "Distinguishing systems and distinguishing senses: New evaluation methods for word sense disambiguation," *Journal of Natural Language Engineering*, vol. 5, no. 2, pp. 113–133, 1999.

[8] R. Krovetz and W. B. Croft, "Lexical ambiguity and information retrieval," *ACM Transactions on Information Systems*, vol. 10, no. 2, pp. 115–141, 1992.

[9] S. Katz, "Estimation of probabilities from sparse data for the language model component of a speech recognizer," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 35, no. 3, pp. 400–401, 1987.

[10] L. Bahl, F. Jelinek, and R. Mercer, "A maximum likelihood approach to continuous speech recognition," *PAMI—IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 5, no. 2, pp. 179–190, 1983.

[11] P. F. Brown, V. J. Della Pietra, P. V. deSouza, J. C. Lai, and R. L. Mercer, "Class-based n-gram models of natural language," *Computational Linguistics*, vol. 18, no. 4, pp. 467–479, 1992.

[12] W. Gale, K. W. Church, and D. Yarowsky, "Estimating upper and lower bounds on the performance of word-sense disambiguation programs," in *Proceedings of the 30th annual meeting on Association for Computational Linguistics*, pp. 249–256, 1992.

[13] G. Hirst, *Semantic Interpretation and the Resolution of Ambiguity*. Cambridge: Cambridge University Press, 1987.

[14] P. Procter, *Longman Dictionary of Contemporary English (LDOCE)*. Harlow: Longman Group, 1978.

[15] R. Chapman, *Roget's International Thesaurus*. New York: Harper and Row, 1977.

[16] G. A. Miller, "WordNet: An on-line lexical database," *International Journal of Lexicography*, vol. 3, no. 4, pp. 235–312, 1990.

[17] H. Kučera and W. N. Francis, *Computational Analysis of Present-Day American English*. Providence, RI: Brown University Press, 1967.

[18] G. A. Miller, C. Leacock, R. Tengi, and R. T. Bunker, "A semantic concordance," in *Proceedings of the Workshop on Human Language Technology*, pp. 303–308, 1993.

[19] D. I. Moldovan and V. Rus, "Logic form transformation of WordNet and its applicability to question answering," in *Proceedings of the Association for Computational Linguistics*, pp. 394–401, 2001.

[20] H. T. Ng and H. B. Lee, "Integrating multiple knowledge sources to disambiguate word sense: An exemplar-based approach," in *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pp. 40–47, 1996.

[21] SIGLEX, "SENSEVAL: Evaluation Exercises for the Semantic Analysis of Text," 2011. http://www.senseval.org.

[22] R. Weischedel, E. Hovy, M. Palmer, M. Marcus, R. Belvin, S. Pradhan, L. Ramshaw, and N. Xue, "OntoNotes: A large training corpus for enhanced processing," in *Handbook of Natural Language Processing and Machine Translation* (J. Olive, C. Christianson, and J. McCary, eds.) New York: Springer, 2011.

[23] S. Pradhan, E. Hovy, M. Marcus, M. Palmer, L. Ramshaw, and R. Weischedel, "OntoNotes: A unified relational semantic representation," *International Journal of Semantic Computing*, vol. 1, no. 4, pp. 405–419, 2007.

[24] E. Hovy, M. Marcus, M. Palmer, L. Ramshaw, and R. Weischedel, "OntoNotes: The 90% solution," in *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 57–60, 2006.

[25] S. Pradhan, E. Loper, D. Dligach, and M. Palmer, "Semeval-2007 task-17: English lexical sample, srl and all words," in *Proceedings of the 4th International Workshop on Semantic Evaluations (SemEval-2007)*, pp. 87–92, 2007.

[26] D. Lenat, "Cyc: A large-scale investment in knowledge infrastructure," *Communications of the ACM*, vol. 38, no. 11, pp. 33–35, 1995.

[27] Z. Dong and Q. Dong, *HowNet and the Computation of Meaning*. Hackensack, NJ: World Scientific, 2006.

Chapter 5

Language Modeling

Katrin Kirchhoff

5.1 Introduction

Many applications in humaa language technology involve the use of a statistical language model—a model that specifies the a priori probability of a particular word sequence in the language of interest. Given an alphabet or inventory of units Σ and a sequence $W = w_1 w_2 \ldots w_i \in \Sigma^*$, a language model can be used to compute the probability of W based on parameters previously estimated from a training set. Most commonly, the inventory Σ (also called **vocabulary**) is the list of unique words encountered in the training data; however, as we will see in this chapter, selecting the units over which a language model should be defined can be a rather difficult problem, particularly in languages other than English.

A language model is usually combined with some other model or models that hypothesize possible word sequences. In speech recognition, a speech recognizer combines acoustic model scores (and possibly other scores, such as pronunciation model scores) with language model scores to decode spoken word sequences from an acoustic signal. In machine translation, a language model is used to score translation hypotheses generated by a translation model. Language models have also become a standard tool in information retrieval [1], authorship identification [2], and document classification [3]. In several related fields, language models are used that are defined not over words but over acoustic units or isolated text characters. One of the core approaches to language identification, for example, relies on language models over phones or phonemes [4]; in optical character recognition, language models predicting character sequences are used [5, 6]. In this chapter, however, the focus is on language models over natural language words or wordlike units, which we define for now as units delimited by whitespace. We first present the fundamental n-gram modeling approach to statistical language modeling, as well as a range of more advanced modeling techniques, before discussing problems arising from the characteristics of particular languages, such as morphologically rich languages or languages without explicit word segmentation. The chapter concludes with a presentation of multilingual and crosslingual language modeling approaches.

5.2 *n*-Gram Models

The probability of a word sequence W of nontrivial length cannot be computed directly because unrestricted natural language permits an infinite number of word sequences of variable lengths. The probability $P(W)$ can be decomposed into a product of component probabilities according to the chain rule of probability:

$$P(W) = P(w_1 \dots w_t) = P(w_1) \prod_{i=1}^{t} P(w_i | w_{i-1} w_{i-2} \dots w_2 w_1) \tag{5.1}$$

Because the individual terms in this product are still too difficult to be computed directly, statistical language models make use of the **n-gram approximation**, which is why they are also called *n*-gram models. The assumption is that all previous words except for the $n-1$ words directly preceding the current word are irrelevant for predicting the current word, or, alternatively, that they are equivalent. Given this assumption of "history equivalence classes," the *n*-gram model is defined as:

$$P(W) \approx \prod_{i=1}^{t} P(w_i | w_{i-1}, \dots w_{i-n+1}) \tag{5.2}$$

Depending on the length of n, we can distinguish between unigrams ($n = 1$), bigrams ($n = 2$), trigrams ($n = 3$), or 4-grams, 5-grams, and so on. An *n*-gram model is also often called an $(n-1)$-th order Markov model, because the approximation in Equation 5.2 embodies the Markov assumption of the independence of the current word given all other words except the $n-1$ preceding words.

5.3 Language Model Evaluation

Before describing parameter estimation methods and further refinements of the basic *n*-gram modeling approach, let us consider the problem of judging the performance of a language model. According to the basic definition given previously, a language model computes the probability of a word sequence W. How can we tell whether a language model is successful at estimating word sequence probabilities? Typically, two criteria are used: **coverage rate** and **perplexity** on a held-out test set that does not form part of the training data. The coverage rate measures the percentage of *n*-grams in the test set that are represented in the language model. A special case of this is the out-of-vocabulary rate (or OOV rate), which is 100 minus the unigram coverage rate, or, in other words, the percentage of unique word types *not* covered by the language model. The second criterion, perplexity, is an information-theoretic measure. Given a model p of a discrete probability distribution, perplexity can be defined as 2 raised to the entropy of p:

$$PPL(p) = 2^{H(p)} = 2^{-\sum_x p(x) \log_2 p(x)} \tag{5.3}$$

In language modeling, we are often more interested in the performance of a language model q on a test set of a fixed size, say t words ($w_1 w_2 \dots w_t$). Then, language model perplexity can be computed as

$$PPL(p, q) = 2^{H(p,q)} = 2^{-\sum_{i=1}^{t} p(w_i) \log_2 q(w_i)} \tag{5.4}$$

or simply

$$2^{-\frac{1}{t}\sum_{i=1}^{t} \log_2 q(w_i)} \tag{5.5}$$

where $q(w_i)$ computes the probability of the i'th word. If $q(w_i)$ is an *n*-gram probability, the equation becomes

$$2^{-\frac{1}{t}\sum_{i=1}^{t} \log_2 p(w_i | w_{i-1}, \dots w_{i-n+1})} \tag{5.6}$$

When comparing different language models, especially models based on different ways of decomposing a text into language modeling units (e.g., words vs. morphemes), care must be taken to normalize their perplexities with respect to the same number of units in order to obtain a meaningful comparison.

Perplexity can be thought of as the average number of equally likely successor words when transitioning from one position in the word string to the next. If the model has no predictive power at all, perplexity is equal to the vocabulary size. A model achieving perfect prediction, by contrast, has a perplexity of one. The goal in language model development is most often to minimize the perplexity on a held-out data set representative of the domain of interest.

However, it should be noted that sometimes the goal of language modeling is not to predict word sequence probabilities but to distinguish between "good" and "bad" word sequence hypotheses generated by some frontend such as a machine translation system or a speech recognizer. In this case, the language model should assign maximally distinct scores to word sequences that are erroneous, ungrammatical, or otherwise unacceptable, as opposed to those that are correct. Optimizing for minimum perplexity does not necessarily achieve this goal; we return to this point in Section 5.6.3.

5.4 Parameter Estimation

5.4.1 Maximum-Likelihood Estimation and Smoothing

The standard procedure in training *n*-gram models is to estimate *n*-gram probabilities using the maximum-likelihood criterion in combination with parameter smoothing. The maximum-likelihood estimate is obtained by simply computing relative frequencies:

$$P(w_i | w_{i-1}, w_{i-2}) = \frac{c(w_i, w_{i-1}, w_{i-2})}{c(w_{i-1}, w_{i-2})} \tag{5.7}$$

where $c(w_i, w_{i-1}, w_{i-2})$ is the count of the trigram $w_{i-2}w_{i-1}w_i$ in the training data. It is obvious that this method fails to assign nonzero probabilities to word sequences that have not been observed in the training data; on the other hand, the probability of sequences that were observed might be overestimated. The process of redistributing probability mass such that peaks in the n-gram probability distribution are flattened and zero estimates are floored to some small nonzero value is called **smoothing**. The most common smoothing technique is **backoff**. Backoff involves splitting n-grams into those whose counts in the training data fall below a predetermined threshold τ and those whose counts exceed the threshold. In the former case, the maximum-likelihood estimate of the n-gram probability is replaced with an estimate derived from the probability of the lower-order $(n-1)$-gram and a backoff weight. In the latter case, n-grams retain their maximum-likelihood estimates, discounted by a factor that redistributes probability mass to the lower-order distribution. Thus, the backed-off probability P_{BO} for w_i given w_{i-1}, w_{i-2} is computed as follows:

$$P_{BO}(w_i|w_{i-1}, w_{i-2}) = \begin{cases} d_c P(w_i|w_{i-1}, w_{i-2}) \text{ if } c > \tau \\ \alpha(w_{i-1}, w_{i-2}) P_{BO}(w_i|w_{i-1}) \text{ otherwise} \end{cases} \tag{5.8}$$

where c is the count of (w_i, w_{i-1}, w_{i-2}), and d_c is a discounting factor that is applied to the higher-order distribution. The normalization factor $\alpha(w_{i-1}, w_{i-2})$ ensures that the entire distribution sums to one and is computed as

$$\alpha(w_{i-1}, w_{i-2}) = \frac{1 - \sum_{w_i : c(w_i, w_{i-1}, w_{i-2}) > \tau} d_c P(w_i|w_{i-1}, w_{i-2})}{\sum_{w_i : c(w_i, w_{i-1}, w_{i-2}) \leq \tau} P_{BO}(w_i|w_{i-1})} \tag{5.9}$$

The way in which the discounting factor is competed determines the precise smoothing technique. Well-known techniques include Good-Turing, Witten-Bell, Kneser-Ney, and others; see the study by Chen and Goodman [7] for an in-depth description and comparison of various smoothing techniques. For example, in Kneser-Ney smoothing, a fixed discounting parameter D is applied to the raw n-gram counts before computing the probability estimates:

$$P_{KN}(w_i|w_{i-1}, w_{i-2}) = \begin{cases} \frac{\max\{c(w_i, w_{i-1}, w_{i-2}) - D, 0\}}{\sum_{w_i} c(w_i, w_{i-1}, w_{i-2})} \text{ if } c > \tau \\ \alpha(w_{i-1}, w_{i-2}) P_{KN}(w_i|w_{i-1}) \text{ otherwise} \end{cases} \tag{5.10}$$

In modified Kneser-Ney smoothing, which is one of the most widely used techniques, different discounting factors D_1, D_2, D_{3+} are used for n-grams with exactly one, two, or three or more counts:

$$Y = \frac{n_1}{n_1 + 2 * n_2} \tag{5.11}$$

$$D_1 = 1 - 2Y\frac{n_2}{n_1} \tag{5.12}$$

$$D_2 = 2 - 3Y\frac{n_3}{n_2} \tag{5.13}$$

$$D_{3+} = 3 - 4Y\frac{n_4}{n_3} \tag{5.14}$$

where $n_1, n_2, \ldots$ are the counts of n-grams with one, two, ..., counts.

Another common way of smoothing language model estimates is linear model interpolation [8]. In linear interpolation, M models are combined by

$$P(w_i|w_{i-1}, w_{i-2}) = \sum_{m=1}^{M} \lambda_m P(w_i|h_m) \tag{5.15}$$

where λ is a model-specific weight. The component models may use different conditioning variables, such as histories of different lengths, or they may have been estimated from different data sets, such as a large set of general data or a smaller set of domain-specific data [see Section 5.5]. The following constraints hold for the model weights: $0 \leq \lambda \leq 1$, and $\sum_m \lambda_m = 1$. Weights are estimated by maximizing the log-likelihood (minimizing the perplexity) on a held-out data set that is different from the training set for the component models (and also different from the final evaluation or test set). This is typically done using the expectation-maximization (EM) procedure [9].

5.4.2 Bayesian Parameter Estimation

Bayesian probability estimation is an alternative parameter estimation method whereby the set of parameters of a model is itself viewed as a random variable governed by a prior statistical distribution. Given a training sample S and a set of parameters θ, $P(\theta)$ denotes a prior distribution over different possible values of θ, and $P(\theta|S)$ is the posterior distribution, which can be expressed using Bayes's rule as

$$P(\theta|S) = \frac{P(S|\theta)P(\theta)}{P(S)} \tag{5.16}$$

In language modeling, the set of parameters is the vector of word probabilities, that is, $\theta = \langle P(w_1), \ldots, P(w_K)\rangle$ (where K is the vocabulary size) for a unigram model, or, more generally, $\theta = \{P(w_1|h_1), \ldots, P(w_K|h_K)\}$ for an n-gram model with K n-grams and history h of a specified length. The training sample S is a sequence of words, $w_1 \ldots w_l$. We require a point estimate of θ given the constraints expressed by the prior distribution and the training sample. This can be done using either the maximum a posteriori (MAP) criterion or the Bayesian criterion. The former finds the value that maximizes the posterior probability given in Equation 5.16:

$$\theta^{MAP} = \underset{\theta \in \Theta}{\operatorname{argmax}}\, P(\theta|S) = \underset{\theta \in \Theta}{\operatorname{argmax}}\, P(S|\theta)P(\theta) \tag{5.17}$$

where Θ is the space of all possible assignments for θ. The Bayesian criterion finds the expected value of θ given the sample S:

$$\theta^{B} = E[\theta|S] = \int_{\Theta} \theta P(\theta|S) d\theta \tag{5.18}$$

$$= \frac{\int_{\Theta} \theta P(S|\theta)P(\theta) d\theta}{\int_{\Theta} P(S|\theta)P(\theta) d\theta} \tag{5.19}$$

Under the assumption that the prior distribution is a uniform distribution, the MAP estimate of the probability for a given word w is equivalent to the maximum-likelihood

estimate, whereas the Bayesian estimate is equivalent to the maximum-likelihood estimate with Laplace smoothing:

$$\theta_w^B = \frac{c(w)+1}{\sum_w c(w)+K} \tag{5.20}$$

Different choices for the prior distribution lead to different estimation functions. The most commonly used prior distribution in language model is the Dirichlet distribution. The Dirichlet distribution is the conjugate prior to the multinomial distribution (i.e., prior and posterior distribution have the same functional form). It is defined as

$$p(\theta) = D(\alpha_1,\ldots,\alpha_K) = \frac{\Gamma(\sum_{k=1}^K \alpha_k)}{\prod_{k=1}^K \Gamma(\alpha_k)} \prod_{k=1}^K \theta_k^{\alpha_k-1} \tag{5.21}$$

where Γ is the gamma function and $\alpha_1,\ldots,\alpha_K$ are the parameters of the Dirichlet distribution (or hyperparameters), which can also be thought of as counts derived from an a priori training sample. The MAP estimate under the Dirichlet prior is

$$\theta^{MAP} = \underset{\theta\in\Theta}{\operatorname{argmax}} \frac{\Gamma(\sum_{k=1}^K \alpha_k)}{\prod_{k=1}^K \Gamma(\alpha_k)} \prod_{k=1}^K \theta_k^{n_k+\alpha_k-1} \tag{5.22}$$

where n_k is the number of times word k occurs in the training sample. The result is another Dirichlet distribution, parameterized by $n_k+\alpha$. The MAP estimate of $P(\theta|W,\alpha)$ thus is equivalent to the maximum-likelihood estimate with add-m smoothing, where $m_k=\alpha_k-1$; that is, pseudocounts of size α_k-1 are added to each word (or n-gram) count. Hyperparameters offer a convenient way of integrating different information sources into the language model estimation process. This approach has most successfully been used for language model adaptation (e.g., [10]), where the prior is computed from a large out-of-domain data set and the observed counts are computed from a small in-domain data set; see Section 5.5 for further details on Bayesian language model adaptation. Early approaches to building language models entirely based on Bayesian estimation [11] did not perform as well as standard n-gram models estimated with the techniques described in Section 5.4.1. However, with recent progress in Bayesian statistics, alternative models have been developed that yield results comparable to those of Kneser-Ney smoothed n-gram models. In particular, this includes models that assume a latent topic structure of documents and use Bayesian estimation techniques to model this structure. These models are discussed more fully in Section 5.6.8.

5.4.3 Large-Scale Language Models

Recently, there has been much interest in scaling language models to very large data sets. The amount of available monolingual data increases daily, and for many languages, models can now be built from sets as large as several billions or trillions of words. Scaling language models to data sets of this size requires modifications to the ways in which language models are trained, stored, and integrated into real-world systems (e.g., speech recognition decoders). It also affects parameter estimation in that exact probability computations may no longer be feasible.

Several sites [12, 13] have proposed distributed approaches to large-scale language modeling. Their common feature is that the entire language model training data is subdivided into several partitions, and counts or probabilities derived from each partition are stored in separate physical locations (i.e., they are distributed over a cluster of independent compute nodes in a client-server architecture). During runtime, clients can request statistics from a set of data partitions through a language model server, thus producing (possibly interpolated) probability estimates on demand. The advantage of distributed language modeling is that it scales to very large amounts of data and large vocabulary sizes and allows new data to be added dynamically without having to recompute static model parameters. Desired parameters, such as the n-gram model order or the specific mixture of different data partitions, can be chosen and requested at runtime, which allows dynamic decoding approaches to be used. The drawback of distributed approaches, however, is the slow speed of networked queries.

In Brants et al. [13], a nonnormalized form of backoff is introduced that differs from standard backoff (Equation 5.8) in that it uses the raw relative frequency estimate instead of a discounted probability if the n-gram count exceeds the minimum threshold (in this case 0):

$$S(w_i|w_{i-1},w_{i-2}) = \begin{cases} P(w_i|w_{i-1},w_{i-2}) \text{ if } c>0 \\ \alpha S(w_i|w_{i-1}) \text{ otherwise} \end{cases} \tag{5.23}$$

The α parameter is fixed for all contexts rather than being dependent on the lower-order n-gram, as in Equation 5.8. The result is no longer a normalized probability distribution but a set of unnormalized scores (denoted by S rather than P for probabilities) that are used in the same manner as standard probabilities. The advantage of this scheme is that unnormalized scores are easier to compute in a distributed framework because summing over all n-gram contexts (which are stored in different physical locations and are therefore expensive to query) is no longer required. Interestingly, the authors found that the model performs almost as well as a model trained with standard Kneser-Ney smoothing on large amounts of data.

An alternative possibility is to use large-scale distributed language models at a second-pass rescoring stage only, after first-pass hypotheses have been generated using a smaller language model [14, 15]. Yet another approach is to store large-scale language models in the working memory of a single machine but to use efficient though possibly lossy data structures. Talbot and Osborne [16] investigate the use of Bloom filters for this purpose. Under this approach, corpus statistics (n-gram frequency counts, context counts, etc.) are represented in a highly memory-efficient, randomized data structure (a Bloom filter) in a quantized fashion. If $c(w_1\ldots w_n)$ is the count of n-gram $w_1\ldots w_n$, the quantized count $q(w_1\ldots w_n)$ is defined as

$$q(w_1\ldots w_n) = 1 + \lfloor \log_b c(w_1\ldots w_n) \rfloor \tag{5.24}$$

At test time, the necessary statistics are queried from the filter, and the true frequency is approximated by the expected count given the quantized count:

$$E[c(w_1 \dots w_n)|q(w_1 \dots w_n) = j] = \frac{b^{j-1} + b^j - 1}{2} \tag{5.25}$$

In this framework, frequencies will never be underestimated but may possibly be overestimated, although the probability of overestimation decreases exponentially with the size of the estimation error. The advantage of this method is that, although the raw frequency counts may be inaccurate, querying the data structure is fast, thus enabling the model to compute smoothed probabilities on the fly. In practice, it was found that the performance of a Bloom filter language model on a machine translation task was close to that of a language model based on exact parameter estimation while providing memory savings of a factor of 4 to 6 [16].

The overall trend in large-scale language modeling is to abandon exact parameter estimation of the type described in the previous section in favor of approximate techniques. This development is likely to continue, leading to more powerful and refined approximation techniques as the number and size of text data collections continue to increase.

5.5 Language Model Adaptation

It is often the case that the amount of language model training data is insufficient, particularly when porting a speech or language processing system to a new domain, topic, or language. For this reason, much effort has been invested in **language model adaptation**, that is, designing and tuning a language model such that it performs well on a new test set for which little equivalent training data is available.

The most commonly used adaptation method is that of mixture language models, or model interpolation. Usually, an in-domain language model is trained using a small amount of in-domain data, and a larger background or generic model is trained using a large amount of out-of-domain data. These models are then interpolated according to Equation 5.15, optimizing the interpolation weights on a small development set. Naturally, this approach generalizes to more than two models, and several variations of basic model interpolation have been developed.

One popular method is topic-dependent language model adaptation. Seymour and Rosenfeld [17] show how documents can first be clustered into a large number of different topics, and individual language models can be built for each topic cluster. The desired final model is then fine-tuned by choosing and interpolating a smaller number of topic-specific language models.

A form of dynamic self-adaptation of a language model is provided by **trigger models**. The idea is that, in accordance with the underlying topic of the text, certain word combinations are more likely than others to co-occur; some words are said to "trigger" others—for example, the words *stock* and *market* in a financial news text. For clustering words according to topic and using them as trigger pairs, latent semantic analysis (LSA) [18] and probabilistic latent semantic analysis (PLSA) [19] have also been been used [20, 21, 22]. LSA, originally

formulated for information retrieval, represents a collection of texts as a document-word co-occurrence matrix, where rows correspond to words, columns correspond to documents, and individual cells represent the (possibly weighted) frequency of the word in the document. Singular-value decomposition applied to this matrix maps words to a low-rank continuous vector space. The semantic similarity within this space can then be computed using, for example, the cosine similarity of the corresponding word vectors. A language model can be adapted dynamically as follows:

$$P(w_i|h_i, \bar{h}_i) = \frac{P(w_i|h_i)\rho(w_i, \bar{h}_i)}{Z(h_i, \bar{h}_i)} \tag{5.26}$$

Here, $\bar{h}_i$ represents the global document history in LSA space up to word i, and ρ represents the similarity function measuring the compatibility between the current word and the semantic history. The idea is that the probabilities of semantically similar words get boosted by a factor proportional to their similarity with the global document history. Trigger relationships can also be incorporated into a language model in the form of constraints within constraint-based modeling frameworks (such as the maximum entropy [MaxEnt] model discussed in Section 5.6.5 [23] or the discriminative language model discussed in Section 5.6.3 [24]).

PLSA extends the basic, nonprobabilistic LSA approach by assuming a more sophisticated latent class model to decompose the word-document co-occurrence matrix instead of using simple singular-value decomposition. Given a latent class c, the probability of each word-document co-occurrence (w, d) can be expressed as:

$$P(w, d) = \sum_c P(c)P(w|c)P(d|c) = P(d)\sum_c P(c|d)P(w|c) \tag{5.27}$$

However, a potential concern with PLSA is its tendency to overfit to the training data. A more recent form of topic-based clustering is latent Dirichlet allocation (LDA) [25], which can be interpreted as a regularized version of PLSA. Topic models based on LDA and its extensions are described in Section 5.6.8.

A further variant of the standard adaptation framework is **unsupervised adaptation**, which is primarily relevant for speech recognition applications. Rather than using written text or perfectly transcribed speech as adaptation data, it is possible to directly use the output of a speech recognizer instead [26]. Several studies (e.g., [27, 28]) have shown that this method achieves roughly half of the improvement obtained by using perfect transcriptions for adaptation.

Recently, it has become common to utilize the Internet as a resource for additional language model data. If available text data for a given topic, domain, or language is insufficient, the web can be queried and additional domain-related data can be retrieved. After preprocessing and possibly filtering the data, it is added to the pool of existing data, or a separate model is trained on the web data and is subsequently interpolated with an existing baseline language model. Several rapid adaptation methods based on this general procedure have been described [29, 30, 31, 32].

Finally, alternative probability estimation schemes for language model adaptation have also been investigated. One of these is **maximum a posteriori (MAP) adaptation** [10].

Here, the counts collected separately from the generic out-of-domain (OD) and the in-domain adaptation (ID) data are combined as follows:

$$P(w|h) = \frac{c_{OD}(w,h) \cdot \varepsilon c_{ID}(w,h)}{c_{OD}(h) \cdot \varepsilon c_{ID}(h)} \tag{5.28}$$

where h and w are the history and predicted word respectively, c_{OD} is the count obtained from the out-of-domain data, and c_{ID} is the count from the in-domain data. The ε parameter ranges between 0 and 1 and represents the weight assigned to the adaptation data: because the amount of out-of-domain data is usually going to outweigh the amount of available adaptation data, the contributions from both sets can be balanced by appropriately setting the ε parameter, which is done empirically. A comparison of MAP and mixture models [33] has shown that mixture models are less robust than MAP adaptation toward variability in the adaptation data.

Although much of the work on language model adaptation has been performed in the context of speech recognition, several studies have also been done on adapting language models for machine translation. In Eck, Vogel, and Waibel [34] and Zhao, Eck, and Vogel [35], queries constructed from first-pass translation hypotheses are used to select additional sentences from a large corpus of target language data, which are then used as additional training data. Models built from this data are interpolated with the baseline language model and are used to retranslate the source language input text. Additional techniques for adapting language models using crosslingual data are described in Section 5.8.2.

5.6 Types of Language Models

Although n-gram models are still the most widely used type of statistical language model by far, a range of alternative models have been developed that have shown additional benefits in practical applications, often when used in combination with n-gram models.

5.6.1 Class-Based Language Models

Class-based language models [36] are a simple way of addressing data sparsity in language modeling. Words are first clustered into classes, either by automatic means [37] or based on linguistic criteria, for example, using part-of-speech (POS) classes. The statistical model makes the assumption that words are conditionally independent of other words given the current word class. If c_i is the class of word w_i, a class-based bigram model can be defined as follows:

$$P(w_i|w_{i-1}) = \sum_{c_i,c_{i-1}} P(w_i|c_i)p(c_i|c_{i-1},w_{i-1})P(c_{i-1}|w_{i-1}) \tag{5.29}$$

$$= \sum_{c_i,c_{i-1}} P(w_i|c_i)P(c_i|c_{i-1})P(c_{i-1}|w_{i-1}) \tag{5.30}$$

under the assumption that c_i is independent of w_{i-1} given c_{i-1}. Usually, a class will contain more than one word, such that the model simplifies to

$$P(w_i|w_{i-1}) = P(w_i|c_i)P(c_i|c_{i-1}) \tag{5.31}$$

Goodman [38] compared this decomposition to the following model:

$$P(w_i|w_{i-1}) \approx P(w_i|c_i,c_{i-1})P(c_i|c_{i-1}) \tag{5.32}$$

where the current word is conditioned not only on the current word class but also on the preceding word classes. Experiments on the North American Business News Corpus (with the training size ranging between 100,000 and 284 million words), using 20,000 test sentences and a vocabulary of 58,000 words showed that the model in Equation 5.32 worked better unless the training data size was at the lower end. Class-based models have been successful in reducing perplexity as well as practical performance in a wide range of language processing systems; however, they typically need to be interpolated with a word-based language model.

5.6.2 Variable-Length Language Models

In standard language modeling, vocabulary units are defined by simple criteria, such as whitespace delimiters, and the prediction of the probability of the next word is based on an invariable fixed-length history (save for backoff). Several modifications to this basic approach have been developed that aim at redefining vocabulary units in a data-driven way, resulting in merged units composed out of a variable number of basic units. These approaches are termed **variable-length** n-gram models. The challenge in these models is to find the best segmentation of the word sequence $w_1w_2 \ldots w_i$ into language modeling units in addition to estimating the language model probabilities. Deligne and Bimbot [39] model the segmentation as a hidden variable and use an ME procedure to find the best segmentation. A variable-length model of order 7 yielded slight improvements in perplexity compared to a standard word-based bigram model, but no application results were reported.

A simpler approach is to start with the initial whitespace-based segmentation suggested by the standard orthography of the language and to merge selected units, rather than attempting to rederive the entire vocabulary segmentation from scratch. A limited number of merged units corresponding to frequently observed short phrases can thus be added to the language model vocabulary. A common criterion for identifying potential phrasal candidate units is the mutual information between adjacent words (e.g., [40]). The actual selection of phrasal units is then made using a greedy iterative algorithm: in each iteration, those candidates are selected that reduce the perplexity of a development corpus the most [41, 42]. In Zitouni, Smaïli, and Haton [42], word class information is used to identify candidate phrasal units in that mutual information is computed over pairs of classes rather than pairs of words, which was shown to reduce perplexity by roughly 10% compared to word-based selection of candidate pairs. This model also achieved an 18% relative reduction in word-error rate on a medium-scale French automatic speech recognition (ASR) task.

5.6.3 Discriminative Language Models

Standard n-gram models embody a generative model for assigning a probability to a given word sequence W. However, in practical applications like machine translation or speech recognition, the task of a language model is often to separate good sentence hypotheses from bad sentence hypotheses. For this reason, it would be desirable to train language model parameters discriminatively, such that word strings of widely differing quality receive maximally distinct probability estimates. Recent attempts at such **discriminative**

language modeling include Roark et al. [43], Collins, Saraçlar, and Roark [44], Shafran and Hall [45], and Arisoy et al. [46]. Here, the language model is applied to an existing set of competing sentence hypotheses $\mathcal{Y}$ generated for an input x (e.g., an acoustic sequence in the case of speech recognition, or the source language string in the case of machine translation) by some generation function $\text{GEN}(x)$. Arbitrary feature functions $\phi(x, y)$ can be defined jointly over the input and each output $y \in \mathcal{Y}$. These are then used in a global linear model that selects the best hypothesis:

$$F(x) = \underset{y \in \text{GEN}(x)}{\text{argmax}}\ \phi(x, y)\alpha \tag{5.33}$$

where α is a weight vector. In the most basic case, the feature functions are the raw n-gram counts obtained from the training data. However, additional feature functions representing statistics over word classes or subword units may be integrated (see §5.7.1). The parameter vector α can be trained by the perceptron algorithm [47] or a conditional log-linear model [43]. The perceptron algorithm, for example, iterates through all training samples (for several epochs) and selects the currently best-scoring hypothesis for each sample. If it differs from the true reference hypothesis, it updates the current weights by adding the counts of the features in the correct hypothesis and subtracting their counts in the chosen hypothesis. Such a training procedure directly minimizes the desired objective function, such as word-error rate in a speech recognition system. Thus, the weights with which the counts of different n-grams contribute to the final model are trained to optimize system performance rather than minimize the perplexity criterion described in Section 5.3. Roark, Saraçlar, and Collins [48] reported a 1.8% absolute improvement in word-error rate (from 39.2% to 37.4%) on a large-vocabulary speech recognition task for a one-pass system and a 0.9% reduction in word-error rate for a multipass recognizer. Recently, discriminative language modeling has also been applied to statistical machine translation [49], where it yielded an improvement of 1 to 2 BLEU points over a state-of-the-art baseline system. As we shall see, a discriminative language model also offers a convenient way of integrating additional linguistic information, such as morphological features.

5.6.4 Syntax-Based Language Models

A well-known drawback of n-gram language models is that they cannot take into account relevant words in the history that fall outside the limited window of the directly preceding $n-1$ words. However, natural language exhibits many types of **long-distance dependencies**, where the choice of the current word is dependent on words that are relatively far removed in terms of sentence position. In the following example, the plural noun *Investors* triggers the plural verb *were* but is not taken into account as a conditioning variable by an n-gram model, where n is usually no larger than 4 or 5.

Investors, *who still showed confidence in financial markets last week*, **were** *responsible for today's downturn.*

To address this problem, several approaches to syntax-based language modeling have been developed, whose goal is to explicitly model such syntactic relationships and use them to

estimate better probabilities. Most of these approaches use a statistical parser to construct the syntactic representation S of the sentence and define a probability model that incorporates S. Chelba and Jelinek's **structured language model** [50] computes the joint probability of a word sequence and its parse S, $P(W, S)$ and decomposes it into a product of component probabilities involving various elements of the the word sequence proper, the head words from the parse structure, and the POS tags in the parse structure. Results reported in [50] indicate that a structured language model interpolated with a trigram model achieved a relative reduction in perplexity of 8% on the Wall Street Journal Continuous Speech Recognition (CSR) and Switchboard corpora. When used for lattice rescoring in a speech recognition system, it yielded a 6% relative reduction on the Wall Street Journal corpus and a 0.5% absolute reduction (41.1% to 40.6%) on Switchboard.

Another syntax-based model is the "almost-parsing" language model by Wang and Harper [51] (also called SuperARV model), which is based on a constraint-dependency grammar. Here, sentences are annotated with so-called SuperARVs, which are rich tags combining lexical features and syntactic information pertaining to a word (entry in the lexicon). A joint language model (the SuperARV language model) is defined over the word sequence and the sequence of tags as follows:

$$P(w_1, \ldots, w_N, t_1, \ldots, t_N) = \prod_{i=1}^{N} P(w_i t_i | w_1 \ldots, w_{i-1}, t_1 \ldots t_{i-1}) \tag{5.34}$$

$$= \prod_{i=1}^{N} P(t_i | w_1 \ldots, w_{i-1}) P(w_i | w_1 \ldots w_{i-1}, t_1 \ldots t_{i-1}) \tag{5.35}$$

$$\approx \prod_{i=1}^{N} P(t_i | w_{i-2}, w_{i-1}, t_{i-1}, t_{i-1}) P(w_i | w_{i-2}, w_{i-1}, t_{i-2}, t_{i-1}) \tag{5.36}$$

The model is smoothed by recursive linear interpolation of higher-order and lower-order models. The SuperARV model was evaluated on the Wall Street Journal Penn Treebank and CSR tasks and was compared to other parser-based language models, including the structured language model described previously, as well as to standard trigram and POS-based models. It was found that the SuperARV achieved the lowest perplexity scores out of all. When used for lattice rescoring on the CSR task, the SuperARV model yielded relative word-error rate reductions between 3.1% and 13.5% and again outperformed all other models.

5.6.5 MaxEnt Language Models

One shortcoming of maximum-likelihood-based probability estimation for language models is that the constraints imposed by estimates solely derived from the training data are too strong. MaxEnt models represent an alternative where these constraints are relaxed. Rather than setting the probability of a given n-gram to its relative frequency in the training data (modulo smoothing), the requirement of MaxEnt modeling is that the model agree,

on average, with the observed counts of events in the training data. The MaxEnt model is formulated as follows:

$$P(y|x) = \frac{1}{Z(x)} \exp\left(\sum_k \lambda_k f_k(x,y)\right) \tag{5.37}$$

where $f(x,y)$ is a feature function defined both on the input and the predicted variables, λ is a feature function specific weight, and $Z(x)$ is a normalization factor, computed as

$$Z(x) = \sum_{y \in Y} \exp\left(\sum_k \lambda_k f_k(x,y)\right) \tag{5.38}$$

Once appropriate feature functions have been defined, the expected value of f_k is

$$E(f_k) = \sum_{x \in X, y \in Y} \tilde{p}(x) p(y|x) f_k(x,y) \tag{5.39}$$

where $\tilde{p}(x)$ is the empirical distribution of x in the training data. The empirical expectation of f_k (derived from the training data) is

$$\tilde{E}(f_k) = \sum_{x \in X, y \in Y} \tilde{p}(x,y) f_k(x,y) \tag{5.40}$$

The model is then trained such that expected values match empirical expected values

$$E(f_k) = \tilde{E}(f_k) \qquad \forall k \tag{5.41}$$

while simultaneously maximizing the entropy of the distribution $p(y|x)$. This is equivalent to maximizing the conditional log-likelihood of the training data.

The MaxEnt framework was first applied to language modeling by Rosenfeld [52]. In the context of language modeling, y represents the predicted word and x the history or, more generally, the conditioning variables used for prediction. Note that in this case, a much larger context than the $n-1$ directly preceding words may be included: feature functions may be defined over the entire sentence [53] or over an even larger domain. Most commonly, however, feature functions are simply defined over n-grams; for example, for a given word w_i and history h_i, a bigram feature function would be defined as follows:

$$f_{w_1,w_2}(h_i, w_i) = \begin{cases} 1 & \text{if } h_i \text{ ends in } w_1 \text{ and } w_i = w_2 \\ 0 & \text{otherwise} \end{cases} \tag{5.42}$$

The model can be trained using iterative methods, such as generalized iterative scaling [54] or improved iterative scaling [55], or by the faster quasi-Newton approaches (see [56]). Nevertheless, training of MaxEnt language models can be computationally demanding: in principle, the normalization factor in Equation 5.38 needs to be computed for all different values of x, and computing the feature expectations requires summation over all (x,y) pairs for which the feature is defined. Wu and Khudanpur [57] presented efficient training methods that result in considerable speedups during training. First the vocabulary is partitioned according to whether words are subject to marginal or conditional constraints, and

the summation required for the normalization factor is computed separately for both sets. Second, a hierarchical normalization computation procedure is proposed where partial sums (e.g., for histories ending in the same suffix) are reused. Together, these modifications led to speedups ranging between 15x and 30x.

Another potential problem of MaxEnt models is that they are prone to overfitting, especially when a large number of feature functions is used in relation to the number of samples. Possible solutions to this problem are feature selection [58], regularization [59], or incorporating priors on the feature functions. Using a Gaussian prior was proposed by Chen and Rosenfeld [59], for example. Rather than simply maximizing the conditional log-likelihood of the training data,

$$\underset{\Lambda}{\operatorname{argmax}} \sum_{i=1}^{M} log P_\Lambda(y_i|x_i) \tag{5.43}$$

this approach maximizes the conditional log-likelihood modulo a penalty term composed of a product of zero-mean Gaussians for all feature functions:

$$\underset{\Lambda}{\operatorname{argmax}} \sum_{i=1}^{M} \log P_\Lambda(y_i|x_i) \times \prod_{k=1}^{K} \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\left(\frac{\lambda^2}{2\sigma_k^2}\right) \tag{5.44}$$

where σ_k^2 is the variance of the k^{th} Gaussian. Goodman [60] suggests exponential priors as an alternative, which can sometimes yield better performance.

Wu and Khudanpur [61] report on a MaxEnt model for speech recognition on the Switchboard task that incorporates topic constraints. The model achieved a 7% perplexity reduction and an absolute word-error rate reduction of 0.7% (38.5% to 37.8%). Integration of syntactic constraints independently yielded a 7% perplexity reduction and a 0.8% reduction in word-error rate. The combination of both types of constraints was shown to be additive, yielding a 12% relative perplexity reduction and an absolute word-error rate improvement of 1.3%.

5.6.6 Factored Language Models

The factored language model (FLM) approach [62, 63] builds on the observations that the prediction of words is dependent on the surface form of the preceding words and that better generalizations can be made when additional information, such as the word's POS or morphological class, is taken into account. In particular, word n-gram counts might be insufficient to robustly estimate the probability of word w_i given word w_{i-1}, but if we know that word w_{i-1} is of a particular class, say a determiner, we might be able to obtain a good probability estimate for $P(w_i|\text{determiner})$. This is reminiscent of the class-based models described previously; however, in an FLM, many such class-based estimates are combined and structured hierarchically through the use of a **generalized backoff** strategy. FLMs assume a factored word representation, where words are considered feature vectors rather than individual surface forms; that is, $W = f_{1:K}$. An example is shown here:

WORD:	*Stock*	*prices*	*are*	*rising*
STEM:	Stock	price	be	rise
TAG:	Nsg	N3pl	V3pl	Vpart

The word surface form itself can be one of the features. A statistical model over this representation can be defined as follows (using a trigram approximation):

$$p\left(f_1^{1:K}, f_2^{1:K}, \ldots, f_t^{1:K}\right) \approx \prod_{i=3}^{t} p\left(f_i^{1:K} | f_{i-1}^{1:K}, \ldots, f_{i-2}^{1:K}\right) \tag{5.45}$$

Thus, each word is dependent not only on a single stream of temporally ordered word variables but also on additional parallel (i.e., simultaneously occurring) feature variables.

In the definition of standard backoff (Equation 5.8), the model backs off from the higher-order to the next lower-order distribution. In an FLM, however, it is not immediately obvious how backoff should proceed because conditioning variables are not only temporally ordered but also occur in parallel. Thus, a decision must be made as to which subset of features the model should back off to in which order. In principle, there are several different ways of choosing among different backoff "paths":

1. Choose a fixed, predetermined backoff path based on linguistic knowledge (e.g., always drop syntactic before morphological variables).
2. Choose the path at runtime based on statistical criteria.
3. Choose multiple paths and combine their probability estimates.

The last option, called **parallel backoff**, is implemented via a new, generalized backoff function (here shown for a trigram):

$$P_{GBO}(f|f_1, f_2) = \begin{cases} d_c P_{ML}(f|f_1, f_2) \text{ if } c > \tau \\ \alpha(f_1, f_2) g(f, f_1, f_2) \text{ otherwise} \end{cases} \tag{5.46}$$

where, similar to Equation 5.8, c is the count of (f, f_1, f_2), $P_{ML}(f|f_1, f_2)$ is the maximum likelihood estimate, τ is the count threshold, and $\alpha(f_1, f_2)$ is the normalization factor that ensures that the resulting scores form a probability distribution. The function $g(f, f_1, f_2)$ determines the backoff strategy. In a typical backoff procedure, $g(f, f_1, f_2)$ equals $P_{BO}(f|f_1)$. In generalized parallel backoff, however, g can be any nonnegative function of f, f_1, f_2 and can be instantiated to, for example, the mean, weighted mean, product, or maximum functions. For example, the mean function would take the average of the individual estimates:

$$g_{\text{mean}}(f, f_1, f2) = 0.5 P_{BO}(f|f_1) + 0.5 P_{BO}(f|f_2) \tag{5.47}$$

In addition to different choices for g, different discounting parameters can be chosen at different levels in the backoff graph.

It is not a priori obvious which backoff strategy works best; the optimal strategy is highly dependent on the particular language modeling task. Because the space of possible factored language model structures and backoff parameters is very large, it is advisable to use an automatic, data-driven procedure to find the best settings. An automatic FLM optimization procedure based on genetic algorithms was proposed by Duh and Kirchhoff [64].

FLMs have been implemented as an add-on to the widely used SRILM (Stanford Research Institute Language Modeling) toolkit [65] and have been used successfully for morpheme-based language modeling [62], multispeaker language modeling [66], dialog act tagging [67], and speech recognition [68, 63], especially in sparse data scenarios such as highly inflecting languages (see also §5.7.2).

5.6.7 Other Tree-Based Language Models

Several other language modeling approaches make use of tree structures; one, for example, is the hierarchical class-based backoff model proposed by Zitouni [69]. Here, backoff proceeds along a hierarchy of word classes arranged in a tree structure, with more general classes at the top and more specific classes at the bottom. Backoff proceeds along the class hierarchy from bottom to top; that is, more specific backoff classes are utilized before more general ones. The main differences from FLMs are that the backoff path is fixed and predefined in advance, whereas FLMs permit the combination of probability estimates from different paths in the backoff graph, as well as the dynamic choice of a path at runtime. Zitouni found that a hierarchical class-based language model yields gains primarily when the test set contains a large number of unseen events: on a speech recognition language modeling task with a 5,000-word vocabulary, the unseen word perplexity was reduced by 10%, whereas on a task with a 20,000-word vocabulary, it was reduced by 26% and the word-error rate decreased by 12%. Some extensions to this hierarchical class n-gram language model were proposed by Wang and Vergyri [70]. Specifically, POS information was integrated into the word clustering process, and separate hierarchical class trees were defined for different POS categories. On an Egyptian Colloquial Arabic speech recognition task (a test set of 18,000 words), the model achieved 8% relative improvement in perplexity over a standard n-gram model and a 3% relative improvement over the model described by Zitouni [69].

Random forest language models (RFLMs), proposed by Xu and Jelinek [71], model all word histories seen in the training data as a collection of randomly grown decision trees (a random forest). Nodes in the decision trees are associated with sets of histories, with the root node containing all histories. Trees are grown by dividing the set of histories into two subsets according to the word identity at a particular position in the history. Out of a number of possible splits, the split that maximizes the log-likelihood of the training data is chosen. Two steps are taken to introduce randomness into this process: first, the set of histories from a parent node are initially assigned randomly to the two subnodes; second, the set of splits chosen to undergo the log-likelihood test is selected randomly as well. After the growing process has stopped, each leaf node in a decision tree can be viewed as a cluster of similar word histories forming an equivalence class. Rather than defining equivalence classes deterministically by the most recent $n-1$ words (as done by conventional n-gram models), equivalence classes are now defined based on the set membership of the words in the history.

The decision tree–growing procedure is run multiple times, and the decision trees resulting from each run are added to the random forest. Supposing that M decision trees have been obtained, the RFLM probabilities are computed as averages of the individual decision tree probabilities:

$$P_{RF}(w_i|w_{i-n+1}, \ldots w_{i-1}) = \frac{1}{M} \sum_{j=1}^{M} P_{DT_j}(w_i|\phi_{DT_j}(w_{i-n+1}, \ldots, w_{i-1})) \tag{5.48}$$

Here, ϕ_{DT_j} is the $j'th$ is a function mapping the history $w_{i-1+n+1}, \ldots, w_{i-1}$ to a leaf node in the j'th decision tree. The number of decision trees M usually ranges in the dozens or hundreds. Perplexity and word-error rate tests on the Penn Treebank portion of the Wall Street Journal corpus showed that the perplexity of a random forest trigram was reduced by 10.6% relative to a trigram with interpolated Kneser-Ney smoothing. Interpolation of

the RFLM with a Kneser-Ney model did not improve perplexity any further. N-best list rescoring with RFLMs yielded a relative word-error rate improvement of 11% on the Wall Street Journal DARPA'93 HUB1 benchmark task. Since their inception, RFLMs have been applied to structured language modeling [71] and prosodic modeling [72]. In the context of multilingual language modeling, they have also been applied to morphologically rich languages (see §5.7.1).

5.6.8 Bayesian Topic-Based Language Models

A significant recent trend in statistical language modeling is Bayesian modeling of latent topic structure in documents. One of the first models of this type was the latent Dirichlet allocation model proposed by Blei, Ng, and Jordan [25]. The LDA model assumes that a document is composed of K topics, denoted as $z_1, \ldots, z_K$. Each topic generates words according to a topic-specific distribution over individual words (i.e., a topic is modeled as a bag of words; n-grams are not taken into account). The probability vector over words is denoted by ϕ_k for each topic $k = 1, \ldots, K$. Each topic has a prior probability, denoted by θ_k. The topic priors $\theta_1, \theta_2, \ldots, \theta_K$ are themselves distributed according to the Dirichlet distribution with hyperparameters $\alpha_1, \ldots, \alpha_K$ (see §5.4.2 for an explanation of the Dirichlet distribution):

$$P(\theta_1, \ldots, \theta_k) = \frac{\Gamma(\sum_k \alpha_k)}{\prod_k \Gamma(\alpha_k)} \prod_{k=1}^{K} \theta_k^{\alpha_k - 1} \tag{5.49}$$

The generative model underlying this approach is that a set of priors $\theta_1, \ldots, \theta_K$ is sampled from the Dirichlet distribution. A given topic z_k is chosen with probability θ_k, then a word w is generated from the topic with probability $\phi_k(w)$. The probability of an entire document consisting of a sequence W of t words is modeled as

$$p(W|\alpha, \phi) = \int p(\theta|\alpha) \left(\prod_{i=1}^{t} \sum_{z_i} p(z_i|\theta) p(w_i|z_i, \phi) \right) d\theta \tag{5.50}$$

The challenging aspect of LDA is computing the posterior distribution of the latent variables θ and $\mathbf{z}$, $p(\theta, \mathbf{z}|W, \alpha, \phi)$, which is not amenable to exact inference. Sampling techniques such as Markov chain Monte Carlo (e.g., [73]) or variational inference [25] need to be used.

Because the LDA model is a unigram model, it also needs to be combined with an n-gram model for practical purposes. Wang et al. [74] combined LDA with a trigram and a probabilistic context-free grammar, yielding perplexity reductions of 9% to 23% on the Wall Street Journal corpus relative to a Kneser-Ney smoothed trigram model. Hsu and Glass [75] used LDA combined with a hidden Markov model for a spoken lecture recognition task. A combination of language models trained with the topic labels provided by this model yielded a 16.1% perplexity reduction and a 2.4% word-error rate reduction over an already adapted trigram model.

The LDA model has been extended in various ways: first, LDA can be generalized to utilize Dirichlet processes [76], a prior for nonparametric models that can handle an infinite number of topics. Thus, rather than assuming a fixed set of K topics, the number can be adjusted on the basis of training data properties. Second, the latent topic variables can

be structured hierarchically: each topic can have a number of subtopics, and individual topics can be shared between different data clusters. This is modeled by a hierarchical Dirichlet process (HDP) [77]. Huang and Renals exploited HDPs for integrating topic and participant role into language models for meeting-style conversational speech recognition. HDP-adapted language models yielded slight word-error-rate reductions (0.3% absolute) compared to standard adaptation methods, for baseline systems ranging around 39% word-error rate. Teh [78] reports on a Bayesian language model based on a Pitman-Yor process that by itself achieves perplexities comparable to those of a Kneser-Ney smoothed trigram model (without requiring interpolation with the baseline model).

5.6.9 Neural Network Language Models

With the exception of LSA-based language models, all of the language modeling approaches described previously estimate probabilities for events in a discrete space. Neural network language models (NNLMs) [79] take a different approach: discrete word sequences are first mapped into a continuous representation, and n-gram probabilities are then estimated within this continuous space. The assumption is that words having similar distributional properties will receive similar continuous representations, which will in turn result in smoother probability estimates.

The neural network is typically a multilayer perceptron with an input layer, a projection layer, a hidden layer, and an output layer of nodes. A graphical representation of the architecture of an NNLM is shown in Figure 5-1. Adjacent layers are fully interconnected by weights. For a vocabulary of V words, the input consists of a concatenation of $n-1$ V-dimensional binary feature vectors representing the history of $n-1$ words (e.g., the first two words in a trigram). The projection layer i of a given fixed dimensionality d encodes the shared continuous representation of the words, which is learned during training. The hidden layer h also has a fixed number of J nodes, each of which computes a thresholded, nonlinear combination of incoming activations, for example, using the tangent function:

$$h_j = \tanh\left(\sum_{k=1}^{d} w_{jk}^{h} i_k + b_j^{h} \right) \qquad \forall j, i = 1, \ldots, J \tag{5.51}$$

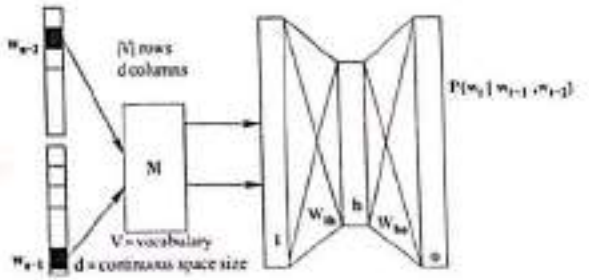


Figure 5-1: Neural network language model

where w^h denotes the weights connecting the projection, and the hidden layers and b^h are the biases on the hidden layer nodes. Finally, the output layer o computes a posterior probability distribution over V by

$$o_j = \frac{a_j}{\sum_{k=1}^{V} a_k} \qquad \forall j, j = 1, \ldots, V \tag{5.52}$$

and

$$a_j = \sum_{i=1}^{J} w_{ji}^o h_k + b_j^o \tag{5.53}$$

During training, binary target labels are associated with the output layer, 1 for the predicted word and 0 for all other words. The network is trained (e.g., using backpropagation) to maximize the log-likelihood of the training data, possibly enriched with a regularization term R to constrain the parameter values θ:

$$L = \frac{1}{T} \sum_{t=1}^{T} log(P(w_t|h_t); \theta) - R(\theta) \tag{5.54}$$

The regularization term can take various forms; a common approach is to use the sum of the squared weights: $\sum_w w^2$. This limits the complexity of the network and reduces the chance of overfitting by preventing the weights from becoming too large. Thus, a particular word in the history is first projected into a continuous representation shared among all words, which is then used as the basis for estimating the probability of the predicted word given its history.

NNLMs have been successfully used for speech recognition by Schwenk and Gauvain [80], Schwenk [81], and Schwenk, Déchelotte, and Gauvain [82], for example, and for machine translation by Alexandrescu and Kirchhoff [83]. Although they typically use a truncated vocabulary consisting of the m most frequent words only, they have yielded significant improvements when used in combination with standard n-gram models. Schwenk [81] reported an 8% relative reduction in perplexity and a 0.5% absolute improvement in word-error rate on a French broadcast news recognition task. Emami and Mangu [82] obtained a 0.8% absolute (3.8% relative) improvement in word-error rate on an Arabic speech recognition task.

Emami and Jelinek [84] used neural network–based probability estimation in combination with a structured language model; Alexandrescu and Kirchhoff [85] combined NNLMs with a factored word representation for Arabic to be able to exploit word similarities derived not only from distributional properties but from morphological and POS classes.

5.7 Language-Specific Modeling Problems

The majority of language modeling research has focused on the English language. However, speech and language processing technology has been ported to a range of other languages, some of which have highlighted problems with the standard n-gram modeling approach and

have necessitated modifications to the traditional language modeling framework. In this section, we look at three types of language-specific problems: morphological complexity, lack of word segmentation, and spoken versus written languages.

5.7.1 Language Modeling for Morphologically Rich Languages

A morphologically rich language is characterized by a large number of different unique word forms (types) in relation to the number of word tokens in a text that is due to productive morphological (word-formation) processes in the language. A morpheme is the smallest meaning-bearing unit in a language. Morphemes can be either free (i.e., they can occur on their own), or they are bound (i.e., they must be combined with some other morpheme). Morphological processes include compounding (forming a new word out of two independently existing free morphemes), derivation (combination of a free morpheme and a bound morpheme to form a new word), and inflection (combination of a free and a bound morpheme to signal a particular grammatical feature).

Germanic languages, for example, are notorious for their high degree of compounding, especially for nominals. Turkish, an agglutinative language, combines several morphemes into a single word; thus, the same material that would be expressed as a syntactic phrase in English can be found as a single whitespace-delimited unit in a Turkish sentence, such as: *görülmemeliydik* = 'we should not have been seen'.

As a result, Turkish has a huge number of possible words. Many languages have rich inflectional paradigms. In languages like Finnish and Arabic, a root (base form) may have thousands of different morphological realizations. Table 5–1 shows two Modern Standard Arabic (MSA) inflectional paradigms, one for present tense verbal inflections for the root *skn* (basic meaning: 'live'), one for pronominal possessive inflections for the root *ktb* (basic meaning 'book').

Morphological complexity causes problems for language modeling due to the high ratio of types to tokens, which results in a lack of training data: many n-grams in the test data are not seen at all in the training data or are not observed frequently enough, leading to unreliable probability estimates. Another effect is a high OOV rate. To some extent, this effect can be mitigated by simply collecting more training data; however, depending on the

Table 5–1: MSA inflectional paradigms for present tense verb forms and possessive pronouns (affixes are separated from the word stem by hyphens)

Word	Meaning	Word	Meaning
'a-skun(u)	I live	kitaab-iy	my book
ta-skun(u)	you (MASC) live	kitaabu-ka	your (MASC) book
ta-skun-iyna	you (FEM) live	kitaabu-ki	your (FEM) book
ya-skun(u)	he lives	kitaabu-hu	his book
ta-skun(u)	she lives	kitaabu-haa	her book
na-skun(u)	we live	kitaabu-na	our book
ta-skun-uwna	you (MASC-PL) live	kitaabu-kum	your book
ya-skun-uwna	they live	kitaabu-hum	their book

Table 5–2: Number of word tokens, word types, and OOV rates for different languages in non-decomposed form

Language	Style	# Tokens	# Types	OOV Rate at (N) Words	Source
English	news text	19M	105k	1% (60k)	[86]
Arabic	news text	19M	690k	11% (60k)	[86]
Czech	news text	16M	415k	8% (60k)	[87]
Korean	news text	15.5M	1.5M	25% (100k)	[88]
Turkish	mixed text	9M	460k	12% (460k)	[89]
Finnish	news text, books	150M	4M	1.5% (4M)	[90]

morphological complexity of the language, the vocabulary growth does not slow down as sharply as for morphologically poor languages as more and more running text is taken into account. Table 5–2 shows examples of the relationship between word types and word tokens for different languages, as well as typical OOV rates for different languages on held-out test sets.

When processing morphologically rich languages, it must be determined whether the vocabulary needed for a given application can be expressed as a list of full word forms (e.g., the most frequent forms occurring in the training data) or whether smaller subword units need to be chosen as basic language modeling units. The choice depends on the constraints imposed by available computing resources, such as the efficiency of the decoder in a speech recognition application, memory, and desired speed, as well as on the amount of training data. The advantage of decomposing words into smaller units lies in the reduction of the vocabulary size, which in turn reduces the number of distinct n-grams. In addition to improving speed and reducing memory consumption, subword units occur in multiple words; therefore, training data can be shared across words, and the number of training tokens per unit increases, which leads to more robust probability estimation. Finally, modeling based on subword units might also allow the language model to assign probabilities to words that were not seen in the training data. On the other hand, if a word is decomposed linearly, a fixed n-gram context provides only information about the relationship between different parts of a word but not about interword dependencies. Thus, the predictive power of the language model is diminished. Moreover, when the language modeling vocabulary is identical to the vocabulary used in a speech recognizer, care must be taken to define units that do not become too short and hence acoustically confusable.

Arabic, generally recognized as a morphologically rich language, is an interesting example highlighting different situations where decomposition may or may not be required. Several studies [68, 63, 91] have shown that integrating morphological information into the language model is helpful for modeling dialectal Arabic. Although the dialects of Arabic exhibit morphological simplifications compared to MSA (the written standard), they have very sparse training data because they are essentially spoken languages and need to be transcribed manually to obtain language modeling data. Large amounts of data are available for MSA, and significant improvements through morphological decomposition in the language model have not been observed for this variety when large training sets were used [91]. Moreover, the vocabulary size needed for large-scale applications such as speech recognition of MSA

is around 600,000 to 800,000 words, which is within the range that current decoders can accommodate [92].

It is obvious, however, that for languages with particularly high type to token ratios (e.g., Finnish or Turkish), some form of decomposition is required: for large tasks, the size of the vocabulary needed to achieve sufficient coverage of the test data is likely to exceed current decoder capabilities, and the corresponding language model would not be able to be estimated reliably. In the following section, we discuss several recent approaches to the problem of word decomposition.

5.7.2 Selection of Subword Units

The identification of subword units can be performed in a data-driven, unsupervised way; it can be based on linguistic information (e.g., a morphological analyzer); or it can be a combination of both. Linguistically based methods typically involve a handcrafted morphological analysis tool, such as the Buckwalter Morphological Analyzer developed for Arabic [93], which analyzes each word form into its morphological components. In the case where several possible analyses are provided for each word form, statistical disambiguation needs to be performed in a subsequent step (e.g., [94]). Data-driven approaches may incorporate varying degrees of information about the specific language in question, and they may vary with respect to the optimization criterion. Some approaches are intended to discover units corresponding to linguistically defined morphemes, whereas others are aimed at selecting an inventory of units that is best suited to the task or application at hand.

Automatic algorithms for identifying linguistic morphemes are as old as Zellig Harris's 1955 approach of estimating the perplexity of different letters following each letter in a word [95]. If the perplexity at a certain transition is high (i.e., the following letters are not easily predictable) a morpheme boundary can be hypothesized at that point. Adda-Decker and Lamel [96] show how a modified version of this approach can be used to decompose German compounds, thus reducing the OOV rate by a relative amount of 23% to 50%, in a German corpus of 300 million words and a fixed vocabulary ranging from 65,000 to 100,000.

In general, simple frequency-based approaches are prone to overfitting to the training data and hypothesizing more morphemes than desired, because the fit to the data is not balanced against the total size of the inventory. This shortcoming is addressed by models that include explicit penalty terms for the size of the morpheme inventory. The more recently developed Morfessor package [97] is one such example. It attempts to derive a morpheme inventory M from a corpus C by maximizing its posterior probability:

$$M = \operatorname*{argmax}_M P(M|C) = P(C|M)P(M) \tag{5.55}$$

which is equivalent to a minimum description length approach. The search over possible morphemes proceeds by way of a greedy algorithm that recursively splits each word into two subparts, trying out all possible positions. Those splits that improve the probability $P(M|C)$ (reduce the code length) are chosen. Later versions of this approach [98, 99] include a stochastic morphological category model and different probability estimation techniques. Morfessor models outperformed most other automatic segmentation algorithms in a benchmark evaluations task involving Finnish, Turkish, and English [100]. Language models using Morfessor to decompose words have been applied to Finnish, Estonian, Turkish, and Arabic

speech recognition and achieved good results on the first three languages, which are highly agglutinative [90].

An alternative to trying to match a predefined linguistic inventory is to derive a unit inventory that directly optimizes a criterion related to language model performance, such as perplexity or OOV rate. This may be preferable in cases where languages are not strictly agglutinative but contain some amount of fusion, such as nontransparent changes in the word form produced by the combination of two or more morphemes. This approach has been pursued by Whittaker and Woodland [101], for example, for Russian language modeling. Here, a particle-based model was developed, defined as

$$P(w_i|h) = \frac{1}{Z(h)} P\left(u^{w_i}_{L(w_i)} | u^{w_i}_{L(w_i)-1}\right) P\left(u^{w_i}_{L(w_i)-1} | u^{w_i}_{L(w_i)-2}\right), \ldots, P(u^{w_i}_1) \left| u^{w_{i-1}}_{L(w_{i-1})}\right. \tag{5.56}$$

where word w_i is decomposed by some decomposition function L into $L(w_i)$ particles $u_1, \ldots, u_{L(w_i)}$. The particle language model then computes the probability of a particle given its history consisting of all particles up to the last particle in the preceding word. Two data-driven methods for deriving particles were compared: a greedy search over possible units of a fixed length, retaining those particles that maximizing the likelihood of the data, and a particle-growing technique whereby particles are initialized to all single-character units and are successively extended by adding surrounding characters such that the resulting units minimize perplexity.

Kiecza, Schultz, and Waibel [88] proposed an approach to Korean language modeling whereby elementary syllable units are combined into units larger than syllables but smaller than Korean words (called *eojols*), which are of a complexity similar to Turkish words. Syllable combination was done by minimizing OOV rate. In both these approaches, significant improvements in perplexity/OOV rate, but only limited gains if any in the final system evaluation (speech recognition word-error rate), were reported.

More recently, the choice of subword units has been optimized with respect to the final system performance, usually by trying different segmentation schemes and evaluating each with respect to its effect on the performance metric. An example is presented by Arisoy, Sak, and Saraçlar [102], where words, statistically derived units, linguistically defined morphemes, and stems plus endings were compared for the purpose of Turkish language modeling for speech recognition, with the result that stems plus endings yielded the best word-error rate out of all four choices.

5.7.3 Modeling with Morphological Categories

Most of the work on language modeling with subword units has focused on linear decomposition of words, primarily for agglutinative languages. As a consequence, the resulting subword units are most frequently used in a standard n-gram model. As mentioned earlier, one problem is that the n-gram context needs to be increased in order to be able to model interword dependencies in addition to dependencies between subword units. This in turn places demands on the amount of training data needed.

However, several alternative approaches have been developed wherein the word remains the basic modeling unit but the probability assignment takes into account statistics over subword components or morphological classes. Arisoy et al. [46] proposed a discriminative language model (see §5.6.3) for Turkish that incorporates feature functions defined over

morphemes, such as root n-gram counts or inflectional group counts. The language model assigns probabilities to sequences of entire words (undecomposed n-best hypotheses), but does so by taking into account the constraints provided by morpheme-based feature functions. This model was shown to provide slightly better results (0.3% absolute word-error rate reduction on a Broadcast News recognition task) than a discriminative language model using only word-based features. The same approach was taken by Shafran and Hall [45] for Czech, with similar results.

Kirchhoff et al. [63] and Vergyri et al. [68] used both morphological classes (stem, root, etc.) and words as conditioning variables in a factored language model for Arabic. Although the model predicts probabilities for entire word forms, the probabilistic backoff procedure makes use of morphological constituents. On a dialectal Arabic speech recognition task with a limited amount of training data, FLMs yielded slight reductions in word error rate (0.5%–1.5% absolute). FLMs have also been used successfully for speech recognition of other highly inflected languages, such as Estonian [103], and for machine translation [104].

Morphological features were also used as additional input features (beyond words) to a neural language model (see §5.6.9) for both Arabic and Turkish [85], thus creating a factored neural language model. Perplexity was shown to improve substantially over a word-based neural language model (up to 10% for Arabic and 40% for Turkish), but no application results were reported.

Sarikaya and Deng [105] proposed a joint morphological-lexical language model for Arabic. Here, a sentence is annotated with a rich parse tree representing morphological, syntactic, semantic, and other attribute information. The language model predicts the probability of the word string and the associated tree jointly, using MaxEnt probability estimation (see §5.6.5). The model was evaluated on an English-to-Arabic translation task and improved the BLEU score by 0.3 points (absolute) over a word trigram model and 0.6 points over a morpheme trigram model.

RFLMs (see §5.6.7) were applied to morphological language modeling by Oparin et al. [106]. The difference from standard word-based RFLMs is that the decision trees used in the random forest model can ask questions not only about the set membership of words but also about morphological features (inflections or morphological tags), stems, lemmas, and parts of speech. Morphological RFLMs were evaluated on a Czech spoken lecture recognition task with a 240,000-word vocabulary. Whereas word-based RFLMs did not show any substantial gain over Kneser-Ney-style trigram language models, morphological RFLMs yielded a relative gain in perplexity of up to 10.4% and a relative improvement in word accuracy of up to 3.4%. Unlike previous studies' results, it was also found that interpolation of RFLMs and standard n-gram models yielded an improvement (of up to 15.6% in perplexity). In addition to producing different word history clusters (induced by morphological features rather than words), a morphological RFLM offers more possibilities for randomization because the set of potential splits at each decision tree node is greatly enlarged, which may contribute to the superior performance of morphological RFLMs in this case.

5.7.4 Languages without Word Segmentation

Although agglutination produces a large number of long and complex word forms in many languages, other languages do not possess any explicit segmentation of character strings into words at all. In languages such as Chinese and Japanese, sentences are written as sequences of characters delimited by punctuation signs but without any intervening whitespaces to

indicate word boundaries. Readers with knowledge of the language can immediately decompose character sequences into the word segmentation that corresponds to the most plausible interpretation of the sentence. Although statistical language models for such languages can be constructed over characters, it is preferable to first segment sentences into words automatically and then train the language model over the resulting words. Similar to the situation in which words are decomposed into subword units (§5.7.1), a language model using characters as the basic modeling units may fail to express important interword relationships. Moreover, the word segmentation may determine how characters are pronounced, which is important if the same modeling units are intended to be used in the language model and the acoustic model of a speech recognition system. Finally, it has been shown experimentally that a language model built on automatically segmented text outperforms a character-based language model in terms of perplexity for both Japanese [107] and Chinese [108].

Automatic segmentation algorithms typically use a combination of dictionary information, statistical search, and additional features such as nonnative letters, character co-occurrence counts, and character positions. Generating the most likely segmentation follows a statistical decoding framework that uses, for example, Viterbi search. Other modeling approaches that have been explored recently include conditional random fields [109, 110, 111], MaxEnt modeling [112, 113], and discriminative modeling using perceptrons [114]. Much of the work on Chinese word segmentation has been performed in the context of the SIGHAN Chinese word segmentation competitions that have been taking place since 2003, organized by the Association for Computational Linguistics. This competition serves as a benchmark comparing different word segmentation systems. Automatic word segmentations are compared against a linguistically segmented gold standard and are evaluated in terms of precision (P), that is, the percentage of correct cases out of all hypothesized boundaries, recall R (the percentage of identified boundaries out of all possible boundaries), and their combination in the form of F-measure, $F = 2PR/(P+R)$. These measures can be calculated separately on the OOV words versus in-vocabulary words. The best-performing systems in the most recent evaluation achieved an F-measure of 0.96; however, performance on OOV words was much lower with F-measures around 0.76 [115].

Rather than trying to match linguistically defined words, it is also possible to optimize segmentation directly for language model performance. Sproat et al. [108] showed that the dictionary used for Chinese word segmentation significantly influences the perplexity of a bigram language model trained on the segmented text and that it is possible to iteratively optimize the dictionary by merging frequent word co-occurrences, such that the perplexity is reduced at each iteration. Note that such approaches are similar to the data-driven algorithms for deriving morpheme-like subword units described earlier in Section 5.7.1. Another example of a data-driven approach, in this case for Japanese, uses chunks of characters for language modeling [107]. Chunks are derived from the training data by selecting the highest-frequency n-grams and additional patterns with close similarity to them. Thus, the basic modeling units are neither characters nor words but intermediate units.

5.7.5 Spoken versus Written Languages

Statistical language modeling crucially relies on large amounts of written text data, and a significant trend within language modeling research is the development of methods for scaling current language modeling techniques to ever-larger databases. However, many of the world's 6,900 languages are spoken languages, that is, languages without a writing system. They are either indigenous languages without a literary tradition, or they are linguistic varieties such as regional dialects that are used in everyday spoken communication but are rarely put into writing. This is the case, for example, with the many dialects of Arabic, which are used in everyday conversation but are almost never found in written form. Other languages may be spoken as well as written but may not have a standardized orthography.

Both cases represent difficulties for language modeling. In the first case, the only way of obtaining language model training data is to manually transcribe the language or dialect. This is a costly and time-consuming process because it involves (i) the development of a writing standard, (ii) training native speakers to use the writing system consistently and accurately, and (iii) the actual transcription effort. In the second case, those text resources that can be obtained for the language in question (e.g., from the web) will need to be normalized, which can also be a laborious process. As a consequence, very little work has been carried out on language model development for such underresourced languages. Most studies have concentrated on how to rapidly collect corpora for underresourced languages using web resources. Some of the challenges inherent in this process are described by Le et al. [116] and Ghani, Jones, and Mladenic [117]. Possible methods for rapid language model bootstrapping for spoken languages and languages characterized by a lack of standardization might include grammar-based or class-based approaches in combination with a limited amount of transcribed material. For a constrained application, such as the development of a dialog system, the structure of possible utterances could be predefined by a task grammar or a class-based language model, whereas more fine-grained word sequence probabilities, or probabilities of words given their classes, are modeled by a language model trained from a small amount of data. An interesting research direction is the possible use of data from language that is closely related to the language in question or from a language that is unrelated but resource rich. The following section describes some of these approaches.

5.8 Multilingual and Crosslingual Language Modeling

5.8.1 Multilingual Language Modeling

Up to this point we have discussed problems that arise when tailoring statistical language models to a particular language or language type, such as agglutinative languages or languages without word segmentation. The tacit assumption has been that the resulting language model is used in an application where only the language of interest is encountered. However, in many situations, a system can be presented with multiple languages sequentially (e.g., different users speaking different languages, without advance indication of which language will be encountered next), or simultaneously, as happens in the case of **code switching**. Here, speakers may use several languages or dialects side by side, often within the same utterance. The phenomenon of code switching exists in a variety of bilingual and multilingual communities or where diglossia exists, such as where a formal standard language is used in addition to colloquial or dialectal varieties. The use of "Spanglish" (mixed Spanish/English)

in the United States is one example of code switching, as demonstrated by the following example from Franco and Solorio [118]:

I need to tell her que no voy a poder ir.
'I need to tell her that I won't be able to make it.'

To handle multilingual input where language can be switched dynamically between utterances, separate language models can be constructed from monolingual corpora, and a system using these models (e.g., a speech-based information kiosk or telephone-based dialog system) can access them dynamically based on the output from a first-step language identification module or based on which language model (possibly combined with an acoustic model in the case of speech recognition) yields the highest scores after the first processing steps.

Füger et al. showed how several such monolingual models can be combined into a single multilingual language model by means of a context-free grammar whose nonterminal states can encode language information and whose terminal states correspond to monolingual n-gram models. Explicit grammar rules can be leveraged to expand current states with n-grams from the matching language only, thus preventing language switching at inappropriate times. Alternative ways of constructing a single multilingual language model are to combine monolingual corpora and train a single model on the pooled data or to interpolate several monolingual language models. The first technique has been shown to degrade performance, especially when the sizes of the corpora are unbalanced [120, 121]. The second technique may fare slightly better yet was shown to be inferior to the grammar-based combination method described earlier [119].

Building a language model for the second case (intrasentential language switching) is more difficult because little or no relevant training material is available. Weng et al. [122] built a four-lingual language model by introducing a common backoff node in the form of a pause unit; that is, language switches are allowed with some probability after a pause has occurred.

5.8.2 Crosslingual Language Modeling

Another question that might be asked is whether data in one language can be leveraged to improve a language model for a different language, provided that the style or domain are closely related. If insufficient data is available for the language of interest, inaccurate probability estimation might be improved if sufficient information can be extracted from a large amount of foreign-language text.

The most straightforward way of applying this idea is to automatically translate the foreign-language text into the desired language and to use the translated text (though errorful) as additional language training data. This approach was chosen, for example, by Khudanpur and Kim [123] and Jensson et al. [124].

In the former study, training data for a Mandarin news text language model intended for speech recognition was enriched with training data obtained by automatically translating English text from the same domain. A unigram extracted from the translated text was interpolated with a trigram baseline language model trained on the available Mandarin data. The selection of English text for translation, and the determination of the interpolation parameter λ were specific to each news story, thus providing at the same time an implicit

form of topic adaptation. The resulting model improved character perplexity by about 10% relative and the word-error rate of the speech recognizer by 0.5% absolute (for various different systems ranging around 25% baseline character-error rate). The authors also noted that the English text was more recent than the Chinese text, which may have contributed to the improved performance. This is an important consideration when investigating potential out-of-language data resources.

Jensson et al. [124] developed a language model for weather reports in Icelandic using a small amount of in-language data and a limited amount of parallel English–Icelandic data for training a machine translation system. A language model trained on a larger set of automatically translated data was then interpolated with the baseline language model, with positive effects on perplexity (9.20% improvement) and word-error rate (1.9–9.5% relative improvement) of an Icelandic speech recognition system.

The use of machine translation technology for processing out-of-language data is likely to fail when the desired language does not have sufficient data to train a machine translation system in the first place, although the abovementioned experiments on Icelandic show that a limited amount of parallel data may still be sufficient if the domain is very constrained. An alternative to using a fully fledged machine translation system is to rely on a high-quality word-based translation dictionary only. Kim and Khudanpur [125] showed that a good-quality translation dictionary can be extracted by simply computing mutual information statistics on word pairs from document-aligned parallel corpora, eliminating the need for sentence-aligned parallel text. In their experiments on Mandarin broadcast news recognition, they found that a unigram model built from the resulting dictionary-based translations and interpolated with a baseline language model achieved a performance similar to that of a crosslingual language model. Another possibility for constructing a translation dictionary from document-aligned data is to use crosslingual latent semantic analysis [126]. Under this approach, words in both languages are projected into a common semantic space, and a measure of similarity between words from different languages in this space is used to construct word translation probabilities.

A drawback of the previous approaches is that the quality of the resulting model is heavily dependent on the translation accuracy. Tam et al. [127] recently presented an alternative model in which bilingual latent semantic analysis (bLSA) is used for adaptation before translation. The approach uses one LSA model each for the source and the target languages; it thus requires a parallel training corpus. The LSA models incorporate Dirichlet-style prior distributions over topics (see §5.6.8). Topic mixture weights are determined from the source LSA model and are projected onto the target LSA model, where they can be used to compute target-language marginal distributions. Assuming that the source language is Chinese (Ch) and the target language is English (En), the marginal word probability distribution in English is

$$P_{\mathrm{En}}(w) = \sum_k \phi_k^{\mathrm{En}}(w)\theta_k^{\mathrm{Ch}} \tag{5.57}$$

where θ_k is the prior for the k^{th} topic and $\phi_k(w)$ is the probability assigned to word w by the k^{th} latent topic. As we can see, topic priors are determined by the source language, whereas the topic-dependent word probability distribution is determined by the target language. The

target-language marginals are then incorporated into the target-language model as follows:

$$P_{\text{target}}(w|h) \propto \left(\frac{P_{\text{bLSA}}(w)}{P_{\text{base}}(w)}\right)^{\beta} P_{\text{base}}(w|h) \tag{5.58}$$

where P_{bLSA} is the adapted probability and P_{base} is the baseline probability. This approach enforces a one-to-one topic correspondence across languages. Evaluation on a Chinese-English statistical machine translation task in the news domain showed that the bLSA-adapted language model reduced perplexity by 9% to 13% and improved the BLEU score by up to 0.3 points.

5.9 Summary

Statistical language modeling has undergone drastic developments in recent years. Although the classical n-gram model in combination with smoothed maximum-likelihood estimation is still the predominant approach, many novel models, ranging from neural network models to discriminative language models, are now being used side by side with standard n-gram models.

Many language modeling techniques, such as language model adaptation, have proven applicable to languages of vastly different types, and the core techniques can be said to be language independent. Crucial differences exist in those cases where languages have a rich morphology, in particular in highly agglutinative languages that can produce a very large number of different word forms per lexeme. In these cases, word decomposition prior to n-gram language modeling or the integration of statistics over subword components into discriminative, factored, or neural language models has been helpful.

A relatively recent trend is the use of very large-scale, distributed language models, which do not follow the traditional probability estimation scheme but use approximate scores or counts. Given that the amount of language modeling data is increasing daily, this trend will certainly become very influential in the near future. It also has significance for languages with large vocabularies (e.g., morphologically rich languages) because it facilitates the use of large language models in practical systems.

Comparatively little research has been carried out on languages that can be characterized as resource poor, that is, languages that do not have a large amount of text data. This includes most of the spoken languages and dialects in the world. At this point, the standard way of obtaining data for these linguistic varieties is to transcribe them manually, which will yield only a limited amount of data. The use of speech recognition technology in combination with a bootstrapping technique could be used to automatically transcribe more data in an incremental fashion. However, there is a chicken-and-egg problem in that the development of sufficiently accurate speech recognition systems requires a sufficient amount of both text and acoustic data to train initial models. The further development of crosslingual adaptation techniques for language models is an important future research direction, which would make a significant contribution toward applying human language technology to resource-poor languages.