POLIMI

DATA SCIENTISTS

Natural Language Processing

Course Notes

Edited by:

Mattia Mancassola

Natural Language Processing

Mattia Mancassola

2019

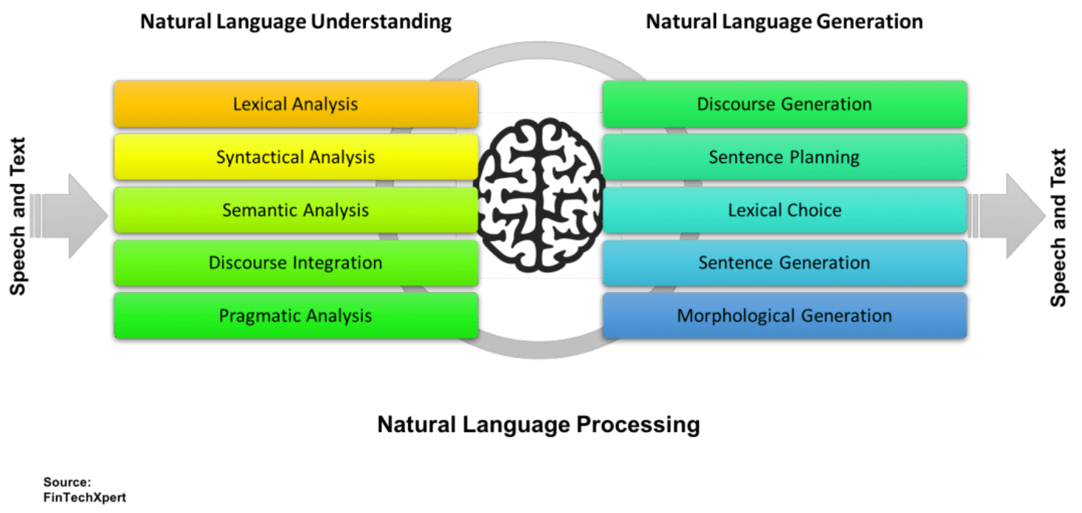
Contents

Disclaimer

This document was written following the slides provided by the professor Roberto Tedesco and especially the book 'Speech and Language Processing'. I mainly used the 3rd edition of the book, which is freely available here: `https://web.stanford.edu/~jurafsky/slp3/`, but since as I'm writing some chapters are not available yet, also the 2nd edition was consulted.

1 Introduction

Natural language processing (NLP) is a subfield of linguistics, computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural language data.
Challenges in natural language processing frequently involve speech recognition, natural language understanding, and natural language generation.



1.1 Some terminology

Before we talk about processing words, we need to decide what counts as a word. Let's start by looking at one particular **corpus (plural corpora)**, a computer-readable collection of text or speech. For example the Brown corpus is a million-word collection of samples from 500 written English texts from different genres (newspaper, fiction, non-fiction, academic, etc.), assembled at Brown University in 1963–64.

How many words are in the following Brown sentence?

They picnicked by the pool, then lay back on the grass and looked at the stars.

This sentence has 16 words if we don't count punctuation marks as words, 18 if we count punctuation. Whether we treat period ("."), comma (","), and so on as words depends on the task.
More precisely, we talk about **Token** if we consider every word, no matter if it is repeated in

the sentence, while we talk about **Types** if we consider unique words. In the previous sentence, for example, we have 18 tokens and 16 types.

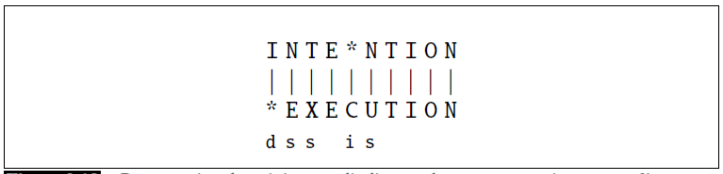
Another interesting distinction is the following: how about inflected forms like cats versus cat? These two words have the same lemma cat but are different wordforms. A **lemma** is a set of lexical forms having the same stem, the same major part-of-speech, and the same word sense. The **wordform** is the full inflected or derived form of the word. For morphologically complex languages like Arabic, we often need to deal with lemmatization. For many tasks in English, however, wordforms are sufficient.

2 Error Correction

Spelling correction is often considered from two perspectives. Non-word spelling correction is the detection and correction of spelling errors that result in non-words (like graffe for giraffe). By contrast, real word spelling correction is the task of detecting and correcting spelling errors even if they accidentally result in an actual word of English (real-word errors).
Non-word errors are detected by looking for any word not found in a dictionary. For example, the misspelling graffe above would not occur in a dictionary. The larger the dictionary the better; modern systems often use enormous dictionaries derived from the web. To correct non-word spelling errors we first generate candidates: real words that have a similar letter sequence to the error. Candidate corrections from the spelling error graffe might include giraffe, graf, gaffe, grail, or craft. We then rank the candidates using a distance metric between the source and the surface error.

2.1 Edit Distance

Edit distance gives us a way to quantify both of these intuitions about string similarity. More formally, the minimum edit distance between two strings is defined distance as the minimum number of editing operations (operations like insertion, deletion, substitution) needed to transform one string into another. The gap between intention and execution, for example, is 5.



```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

Figure 2.13 Representing the minimum edit distance between two strings as an **alignment**. The final row gives the operation list for converting the top string into the bottom string: d for deletion, s for substitution, i for insertion.

We can also assign a particular cost or weight to each of these operations. The Levenshtein distance between two sequences is the simplest weighting factor in which each of the three operations has a cost of 1.

To solve the Minimum Edit Distance problem we can use Dynamic Programming, in particular the minimum edit distance algorithm, named by Wagner and Fischer (1974).

Real word spelling error detection is a much more difficult task, since any word in the input text could be an error. Still, it is possible to use the noisy channel to find candidates for each word w typed by the user, and rank the correction that is most likely to have been the users original intention.

2.2 The Noisy Channel Model

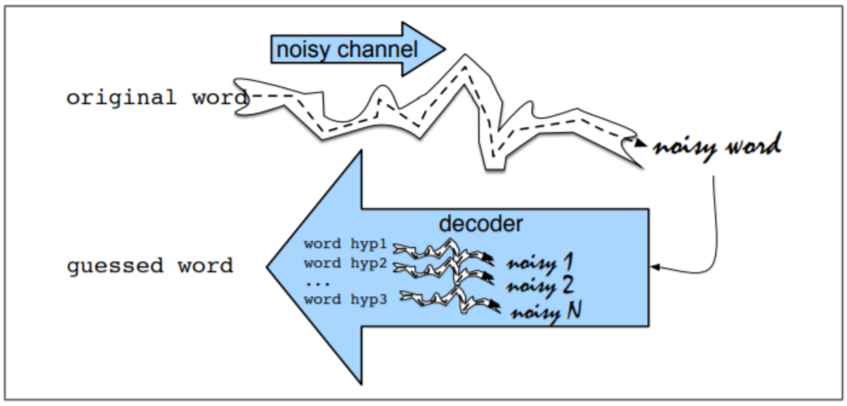


Figure B.1 In the noisy channel model, we imagine that the surface form we see is actually a "distorted" form of an original word passed through a noisy channel. The decoder passes each hypothesis through a model of this channel and picks the word that best matches the surface noisy word.

This noisy channel model is a kind of **Bayesian inference**: we see an observation x (a misspelled word) and our job is to find the word w that generated this misspelled word. Out of all possible words in the vocabulary V we want to find the word w such that $P(w|x)$ is highest:

$$\hat{w} = \arg\max_{w \in V} P(w|x)$$

That thanks to the Bayes' rule, it can be written as:

$$\hat{w} = \arg\max_{w \in V} \frac{P(x|w)P(w)}{P(x)}$$

Since we will compute the above formula for each word and $P(x)$ does not change, we can discard it, obtaining the simpler:

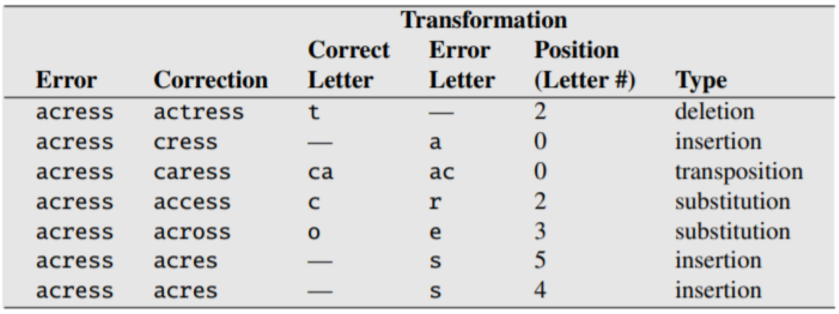
$$\hat{w} = \arg\max_{w \in V} P(x|w)P(w)$$

In order to understand how these quantities are computed, let's consider an example, applying the algorithm to the misspelled *acress*.

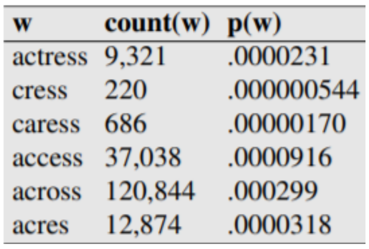
1st - Find the candidates
We will use the minimum distance edit algorithm introduced before, but extended with a new type of edit, i.e. transpositions, in which two letters are swapped. This version is called **Damerau-Levenshtein** edit distance.

2nd - Compute the prior and the likelihood
The prior probability $P(w)$ is the language model probability of the word w in context, which can be computed using any language model, from unigram to trigram or 4-gram. For this example let's assume a **unigram language model**.

Computing the likelihood $P(x|w)$, also called the channel model, is much more difficult. A perfect model of the probability that a word will be mistyped would condition on all sorts of



		Transformation			
Error	Correction	Correct Letter	Error Letter	Position (Letter #)	Type
acress	actress	t	—	2	deletion
acress	cress	—	a	0	insertion
acress	caress	ca	ac	0	transposition
acress	access	c	r	2	substitution
acress	across	o	e	3	substitution
acress	acres	—	s	5	insertion
acress	acres	—	s	4	insertion



w	count(w)	p(w)
actress	9,321	.0000231
cress	220	.000000544
caress	686	.00000170
access	37,038	.0000916
across	120,844	.000299
acres	12,874	.0000318

factors: who the typist was, whether the typist was left-handed or right-handed, and so on. Luckily, we can get a pretty reasonable estimate of $P(x|w)$ just by looking at local context: the identity of the correct letter itself, the misspelling, and the surrounding letters. For example, the letters m and n are often substituted for each other.
A simple model might estimate $p(acress|across)$ just using the number of times that the letter e was substituted for the letter o in some large corpus of errors. What we need in order to apply this approach is the so called **Confusion Matrix**.

Following Kernighan et al. (1990), we'll use four confusion matrices.

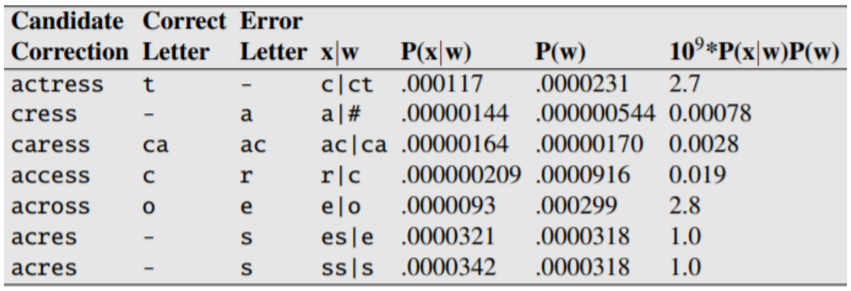
del[x,y]: count(xy typed as x)
ins[x,y]: count(x typed as xy)
sub[x,y]: count(x typed as y)
trans[x,y]: count(xy typed as yx)

The confusion matrices can be downloaded online or constructed by iteratively applying this very spelling error correction algorithm itself. The iterative algorithm first initializes the matrices with equal values; thus, any character is equally likely to be deleted, equally likely to be substituted for any other character, etc. Next, the spelling error correction algorithm is run on a set of spelling errors. Given the set of typos paired with their predicted corrections, the confusion matrices can now be recomputed, the spelling algorithm run again, and so on. This iterative algorithm is an instance of the important **EM algorithm**.

Once we have the confusion matrix, we can estimate $P(x|w)$ as follows:

$$P(x|w) = \begin{cases} \frac{del[x_{i-1},w_i]}{count[x_{i-1}w_i]} \text{if deletion} \\ \frac{ins[x_{i-1},w_i]}{count[w_{i-1}]} \text{if insertion} \\ \frac{sub[x_i,w_i]}{count[w_i]} \text{if substitution} \\ \frac{trans[w_i,w_{i+1}]}{count[w_i w_{i+1}]} \text{if transposition} \end{cases}$$

The results say that the noisy channel model chooses **across** as the best correction, and **actress** as the second most likely word.



Candidate Correction	Correct Letter	Error Letter	x|w	P(x|w)	P(w)	10^9*P(x|w)P(w)
actress	t	-	c|ct	.000117	.0000231	2.7
cress	-	a	a|#	.00000144	.000000544	0.00078
caress	ca	ac	ac|ca	.00000164	.00000170	0.0028
access	c	r	r|c	.000000209	.0000916	0.019
across	o	e	e|o	.0000093	.000299	2.8
acres	-	s	es|e	.0000321	.0000318	1.0
acres	-	s	ss|s	.0000342	.0000318	1.0

Unfortunately, the algorithm was wrong here; the writer's intention becomes clear from the context: *...was called a "stellar and versatile* ***acress*** *whose combination of sass and glamour has defined her..."*. The surrounding words make it clear that **actress** and not **across** was the indended word.

This is why it is important to use larger language models than unigrams.

3 N-gram Language Models

In this section we will introduce models that assign a probability to each possible next word. The same models will also serve to assign a probability to an entire sentence.

Such a model, for example, could predict that the following sequence has a much higher probability of appearing in a text:

all of a sudden I notice three guys standing on the sidewalk

than does this same set of words in a different order:

on guys all I of notice sidewalk three a sudden standing the

Models that assign probabilities to sequences of words are called language models or LMs. In this chapter we introduce the simplest model that assigns probabilities to sentences and sequences of words, the n-gram. An n-gram is a sequence of N words: a 2-gram (or bigram) is a two-word sequence of words like "please turn", "turn your", or "your homework", and a 3-gram (or trigram) is a three-word sequence of words like "please turn your", or "turn your homework".

Let's begin with the task of computing $P(w|h)$, the probability of a word w given some history h. Suppose the history h is "its water is so transparent that" and we want to know the probability that the next word is *the*:

$$P(the|its\ water\ is\ so\ transparent\ that)$$

One very simple way to estimate this probability is from relative frequency counts:

$$P(the|its\ water\ is\ so\ transparent\ that) = \frac{C(its\ water\ is\ so\ transparent\ that\ the)}{C(its\ water\ is\ so\ transparent\ that)}$$

According to Google those counts are 5/9. As you can imagine, these numbers are too small to represent a good estimate. This is because language is creative; new sentences are created all the time, and we won't always be able to count entire sentences. Even simple extensions of the example sentence may have counts of zero on the web.

3.1 The Chain Rule

$$P(A|B) = \frac{P(A,B)}{P(B)}$$

$$P(A,B) = P(A|B)P(B)$$

For sequences:

$$P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)$$

This, considering the shorthand $P(w_1, w_2, ..., w_n) = P(w_1^n)$:

$$P(w_1^n) = P(w_1)P(w_2|w_1)P(w_3|w_1^2)...P(w_n|w_1^{n-1}) = \prod_{k=1}^{n} P(w_k|w_1^{k-1})$$

The chain rule shows the link between computing the joint probability of a sequence and computing the conditional probability of a word given previous words.

The problem with this formula is that, as said before, we don't know any way to compute the exact probability of a word given a long sequence of preceding words.
The intuition of the n-gram model is that instead of computing the probability of a word given its entire history, we can approximate the history by just the last few words.

The **bigram** model, for example, approximates the probability of a word given all the previous words by using only the conditional probability of the preceding word, thus making the following approximation:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-1})$$

The assumption that the probability of a word depends only on the previous word is called a **Markov assumption**. Markov models are the class of probabilistic models that assume we can predict the probability of some future unit without looking too far into the past.

The general equation for a n-gram approximation is:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1})$$

How do we estimate these bigram or n-gram probabilities? An intuitive way to estimate probabilities is called **Maximum Likelihood Estimation (MLE)**.

For the bigram case:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{\sum_w C(w_{n-1}w)}$$

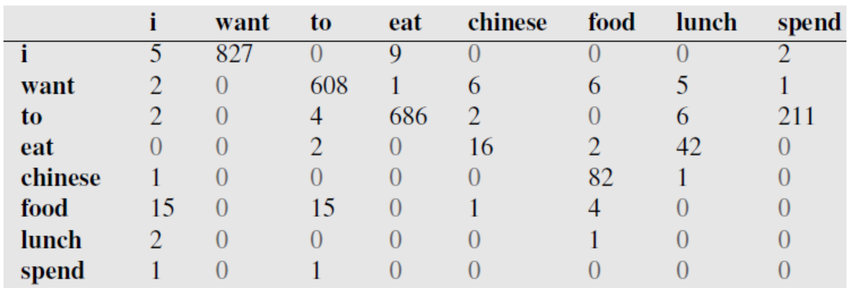
We can simplify this equation, since the sum of all bigram counts that start with a given word w_{n-1} must be equal to the unigram count for that word w_{n-1}.

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

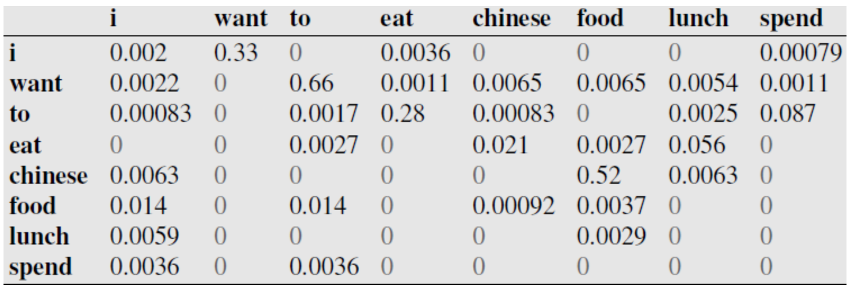
Leading to the generalization:

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}w_n)}{C(w_{n-N+1}^{n-1})}$$

Let's now consider the data coming from the Berkeley Restaurant Project (bigram counts and probabilities respectively):



	i	**want**	**to**	**eat**	**chinese**	**food**	**lunch**	**spend**
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0



	i	**want**	**to**	**eat**	**chinese**	**food**	**lunch**	**spend**
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

It is interesting to notice that as crude as they are, N-gram probabilities capture a range of interesting facts about language. In fact, some of the bigram probabilities above encode some facts that we think of as strictly syntactic in nature, like the fact that what comes after eat is usually a noun or an adjective, or that what comes after to is usually a verb. Others might be a fact about the personal assistant task, like the high probability of sentences beginning with the words I. And some might even be cultural rather than linguistic, like the higher probability that people are looking for Chinese versus English food.

3.2 Evaluating Language Models

Extrinsic evaluation
1) Put model A into an application
2) Evaluate the performance of the application with model A
3) Put model B into the application and evaluate
4) Compare performance of the application with the two models

Unfortunately, running big NLP systems end-to-end is often very expensive. Instead, it would be nice to have a metric that can be used to quickly evaluate potential improvements in a language model.

Intrinsic evaluation
1) Train parameters of our model on a training set
2) Look at the models performance on some new data
3) So use a test set. A dataset which is different than our training set, but is drawn from the same source
4) Then we need an evaluation metric to tell us how well our model is doing on the test set

One evaluation metric is the *perplexity*.

3.3 Perplexity

The perplexity (sometimes called PP for short) of a language model on a test set is the inverse probability of the test set, normalized by the number of words. For a test set $W = w_1 w_2 ... w_N$:

$$PP(W) = P(w_1 w_2 ... w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1 w_2 ... w_N)}}$$

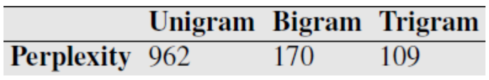
Expanding with the chain-rule:

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_1 ... w_{i-1})}}$$

So on a bigram language model:

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i | w_{i-1})}}$$

Notice that minimizing perplexity is equivalent to maximizing the test set probability according to the language model.



	Unigram	**Bigram**	**Trigram**
Perplexity	962	170	109

The perplexity of two language models is only comparable if they use identical vocabularies.

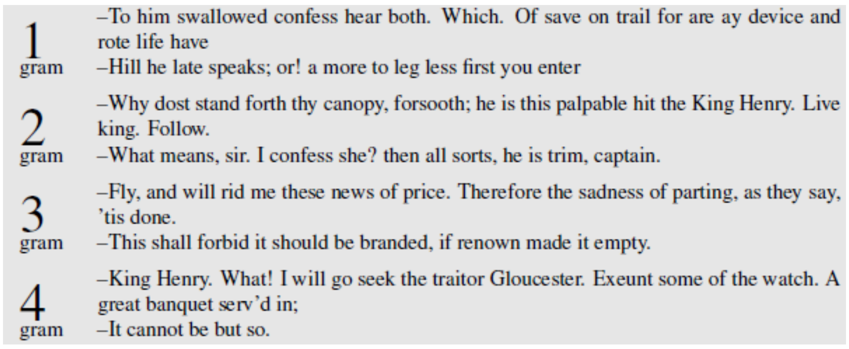
An (intrinsic) improvement in perplexity does not guarantee an (extrinsic) improvement in the performance of a language processing task like speech recognition or machine translation. Nonetheless, because perplexity often correlates with such improvements, it is commonly used as a quick check on an algorithm. But a model's improvement in perplexity should always be confirmed by an end-to-end evaluation of a real task before concluding the evaluation of the model.

3.4 Generalization and Zeros

The n-gram model, like many statistical models, is dependent on the training corpus. One implication of this is that the probabilities often encode specific facts about a given training corpus. Another implication is that n-grams do a better and better job of modeling the training corpus as we increase the value of N.

This can be easily visualized by using the **Shannon's method**, that consists of generating random sentences from different n-gram models:

1) Sample a random bigram $(< s >, w)$ according to its probability $P(w| < s >)$
2) Now sample a random bigram (w, x) according to its probability $P(x|w)$
3) And so on until we randomly choose a $(y, < /s >)$
4) String the words together



1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; –It cannot be but so.

The longer the context on which we train the model, the more coherent the sentences. However, notice that the text generated by quadrigrams looks like Shakespeare because it is Shakespeare! The words *It cannot be but so*, indeed, are directly from King John. This is due to the fact that the Shakespeare oeuvre is not very large as corpora go; we have $N = 884,647$ tokens and $V = 29,066$ types and our n-gram probability matrices are ridiculously sparse (out of $V^2 = 844$ *million* possible bigrams types Shakespeare actually produced 300,000 bigram types).

So, two important things here:

1) There is a dependence of the grammar on its training set, thus according to what you want to accomplish, you have to select a proper corpus.
2) In the above example 99.96% of the possible bigrams were never seen (have zero entries in the table). This is the biggest problem in language modeling.

3.5 Smoothing

What do we do with words that are in our vocabulary (they are not unknown words) but appear in a test set in an unseen context (for example they appear after a word they never appeared after in training)? To keep a language model from assigning zero probability to these unseen events, we'll have to shave off a bit of probability mass from some more frequent events and give it to the events we've never seen. This modification is called **smoothing** or **discounting**.

3.5.1 Laplace Smoothing

The simplest way to do smoothing is to add one to all the bigram counts, before we normalize them into probabilities. All the counts that used to be zero will now have a count of 1, the counts of 1 will be 2, and so on. This algorithm is called **Laplace smoothing**. Laplace smoothing does not perform well enough to be used in modern n-gram models, but it usefully introduces many of the concepts that we see in other smoothing algorithms, gives a useful baseline, and is also a practical smoothing algorithm for other tasks like text classification.

Recalling that

$$P(w_i) = \frac{c_i}{N}$$

Since there are V words in the vocabulary and each one was incremented, we also need to adjust the denominator to take into account the extra V observations:

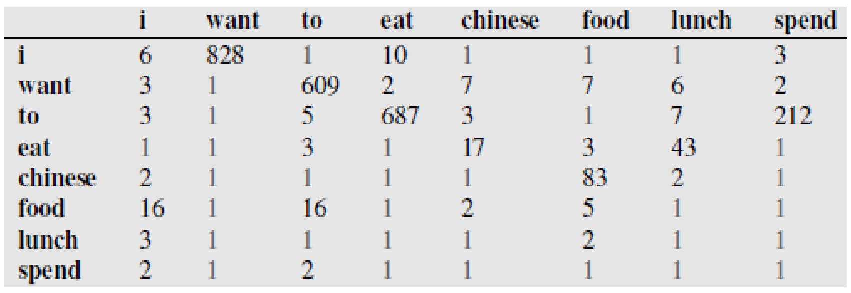
$$P^*(w_i) = \frac{c_i + 1}{N + V}$$

Instead of changing both the numerator and denominator, it is convenient to describe how a smoothing algorithm affects the numerator, by defining an adjusted count c^*. This adjusted count is easier to compare directly with the MLE counts and can be turned into a probability like an MLE count by normalizing by N.

$$c_i^* = (c_i + 1)\frac{N}{N + V}$$

$$P^*(w_i) = \frac{c_i^*}{N}$$

So, starting from



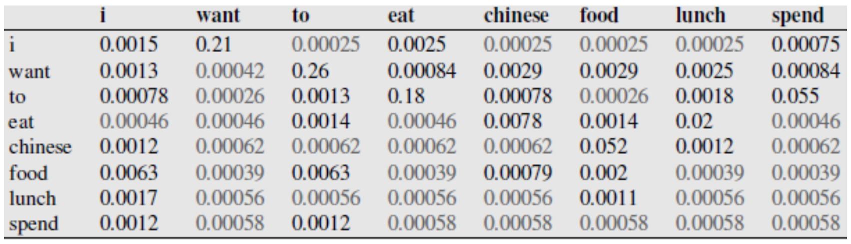
	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Knowing that

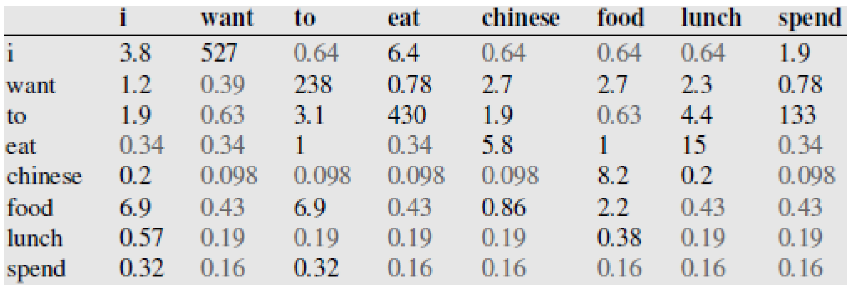
$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + 1}{C(w_{n-1}) + V}$$

We can reconstruct the counting matrix in the following way

$$c^*(w_{n-1}w_n) = \frac{[C(w_{n-1}w_n) + 1] \times C(w_{n-1})}{C(w_{n-1}) + V}$$



	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058



	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Note that add-one smoothing has made a very big change to the counts. C(want to) changed from 608 to 238! We can see this in probability space as well: P(to—want) decreases from .66 to .26. The sharp change in counts and probabilities occurs because too much probability mass is moved to all the zeros.

3.5.2 Add-k smoothing

One alternative to add-one smoothing is to move a bit less of the probability mass from the seen to the unseen events. Instead of adding 1 to each count, we add a fractional count k (.5? .05? .01?).

$$P^*_{Add-k}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n) + k}{C(w_{n-1}) + kV}$$

The value of k can be chosen by cross-validation.

3.5.3 Good-Turing smoothing

The basic insight of Good-Turing smoothing is to re-estimate the amount of probability mass to assign to N-grams with zero or low counts by looking at the number of N-grams with higher counts. In other words, we estimate the probability of words occurring c times with adjusted MLE estimation of words occurring $c+1$ times:

$$c^* = (c+1)\frac{N_{c+1}}{N_c}$$

For example, the revised count for the bigrams that never occurred (c_0) is estimated by dividing the number of bigrams that occurred once by the number of bigrams that never occurred.

Observations

1) Note that the Good-Turing estimate relies on the assumption that we know N_0, the number

of bigrams we haven't seen. We know this because given a vocabulary of size V, the total number of bigrams is V^2, hence N_0 is V^2 minus all the bigrams we have seen.

2) In practice, this discounted estimate c^* is not used for all counts c. Large counts (where $c > k$ for some threshold k) are assumed to be reliable. Katz (1987) suggested setting k at 5. Introducing k makes the correct equation for c^* more complicated:

$$c^* = \frac{(c+1)\frac{N_{c+1}}{N_c} - c\frac{(k+1)N_{k+1}}{N_1}}{1 - \frac{(k+1)N_{k+1}}{N_1}} \quad for\ 1 \leq c \leq k$$

3) Good-Turing is usually used in combination with *backoff* and *interpolation.*

3.5.4 Backoff and Interpolation

If we are trying to compute $P(w_n|w_{n-2}w_{n-1})$ but we have no examples of a particular trigram $w_{n-2}w_{n-1}w_n$, we can instead estimate its probability by using the bigram probability $P(w_n|w_{n-1})$. Similarly, if we don't have counts to compute $P(w_n|w_{n-1})$, we can look to the unigram $P(w_n)$.
In backoff, we use the trigram if the evidence is sufficient, otherwise we use the bigram, otherwise the unigram. In other words, we only "back off" to a lower-order n-gram if we have zero evidence for a higher-order n-gram.

By contrast, in interpolation we always mix the probability estimates from all the n-gram estimators, weighting and combining the trigram, bigram, and unigram counts.

Simple linear interpolation

$$\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1 P(w_n|w_{n-2}w_{n-1}) + \lambda_2 P(w_n|w_{n-1}) + \lambda_3 P(w_n)$$

such that $\sum_i \lambda_i = 1$

Lambdas conditional on context

$$\hat{P}(w_n|w_{n-2}w_{n-1}) = \lambda_1(w_{n-2}^{n-1}) P(w_n|w_{n-2}w_{n-1}) + \lambda_2(w_{n-2}^{n-1}) P(w_n|w_{n-1}) + \lambda_3(w_{n-2}^{n-1}) P(w_n)$$

such that $\sum_i \lambda_i(w_{n-2}^{n-1}) = 1$

How are these λ values set?

Both the simple interpolation and conditional interpolation λs are learned from a held-out corpus. A held-out corpus is an additional training corpus that we use to set hyperparameters like these λ values, by choosing the λ values that maximize the likelihood of the held-out corpus. One way is to use the **EM** algorithm, an iterative algorithm that converges to locally optimal λs.

3.5.5 Katz Backoff

Katz backoff combines both backoff and discounting approaches in the following way: we rely on a discounted probability P^* if we've seen this n-gram before (i.e., if we have non-zero counts). Otherwise, we recursively back off to the Katz probability for the shorter-history (N-1)-gram.

$$P_{katz}(w_n|w_{n-N+1}^{n-1}) = \begin{cases} P^*(w_n|w_{n-N+1}^{n-1}) \quad if\ C(w_{n-N+1}^n) > 0 \\ \alpha(w_{n-N+1}^{n-1})P_{katz}(w_n|w_{n-N+2}^{n-1}) \quad \text{otherwise} \end{cases}$$

The probability $P^*(w_n|w_{n-N+1}^{n-1})$ will be slightly less than the MLE estimate $\frac{c(w_{n-N+1}^n)}{c(w_{n-N+1}^{n-1})}$, i.e. on average the c^* will be less than c. This will leave some probability mass for the lower order n-grams. Now we need to build the α weighting we'll need for passing this mass to the lower-order n-grams. Let's represent the total amount of left-over probability mass by the function β, a function of the n-1-gram context.

$$\beta(w_{n-N+1}^{n-1}) = 1 - \sum_{w_n:C(w_{n-N+1}^n)>0} P^*(w_n|w_{n-N+1}^{n-1})$$

This gives us the total probability mass the we are ready to distribute to all n-1-gram (e.g. bigrams if our original model was a trigram). Each individual n-1-gram will only get a fraction of this mass, so we need to normalize β by the total probability of all the n-1-grams that begin some n-gram.

$$\alpha(w_{n-N+1}^{n-1}) = \frac{\beta(w_{n-N+1}^{n-1})}{\sum_{w_n:C(w_{n-N+1}^n)=0} P_{katz}(w_n|w_{n-N+2}^{n-1})}$$

3.6 Unknown Words

Usually a language model is learned starting from:

- A collection of documents;
- A vocabulary (i.e. a lexicon) containing all the words we want to recognize (generated from the document collection).

Then, the document collection is split into two parts:

- A training set, used for learning the language model;
- A test set, used for testing the language model (e.g. perplexity).

At training time, the words not found into the training set are managed using smoothing/interpolation/backoff.
At test time, using the test set, no new words can appear.
At run-time, we can adopt two different approaches:

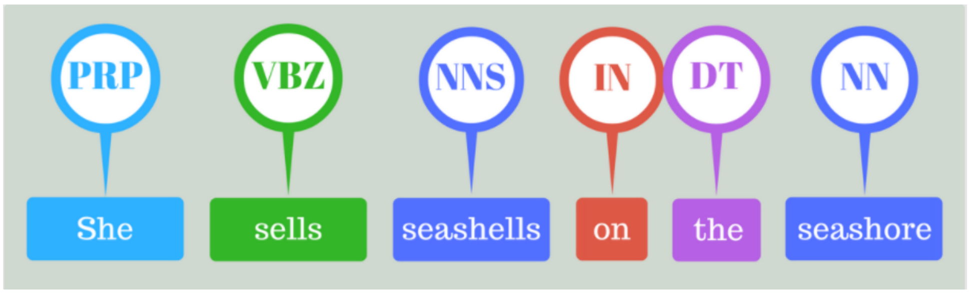
- Closed vocabulary: we assume that new words cannot appear. This is a reasonable assumption in some domains, such as speech recognition or machine translation.
- Open vocabulary: new words may appear, called **unknown** words or **out of vocabulary (OOV)** words. We don't use smoothing/interpolation/backoff for these, but we create an unknown word token $< UNK >$. There are two common ways to train the probabilities of the unknown word model $< UNK >$. The first one is to turn the problem back into a closed vocabulary one by choosing a fixed vocabulary in advance:
  1) Choose a vocabulary (word list) that is fixed in advance;
  2) Convert in the training set any word that is not in this set (any OOV word) to the unknown word token $< UNK >$ in a text normalization step;

3) Estimate the probabilities for $< UNK >$ from its counts just like any other regular word in the training set.
The second alternative, in situations where we don't have a prior vocabulary in advance, is to create such a vocabulary implicitly, replacing words in the training data by $< UNK >$ based on their frequency. For example, we can replace all words that occur fewer than n times in the training set, where n is some small number, or equivalently select a vocabulary size V in advance (say 50,000) and choose the top V words by frequency and replace the rest by UNK. In either case, we then proceed to train the language model as before, treating $< UNK >$ like a regular word.

The exact choice of $< UNK >$ model does have an effect on metrics like perplexity. A language model can achieve low perplexity by choosing a small vocabulary and assigning the unknown word a high probability. *For this reason, perplexities should only be compared across language models with the same vocabularies.*

4 Part-Of-Speech Tagging

Parts-of-speech (also known as POS, word classes, or syntactic categories) are useful because they reveal a lot about a word and its neighbors. Knowing whether a word is a noun or a verb tells us about likely neighboring words (nouns are preceded by determiners and adjectives, verbs by nouns) and syntactic structure word (nouns are generally part of noun phrases), making part-of-speech tagging a key aspect of parsing.
In this section we will introduce POS and two algorithms for POS tagging, the task of assigning parts-of-speech to words. One is generative - **Hidden Markov Model (HMM)** - and one is discriminative - **Maximum Entropy Markov Model (MEMM)**. Also Recurrent Neural Network (RNN) can be used to approach this problem. These three models have roughly equal performance but different tradeoffs.

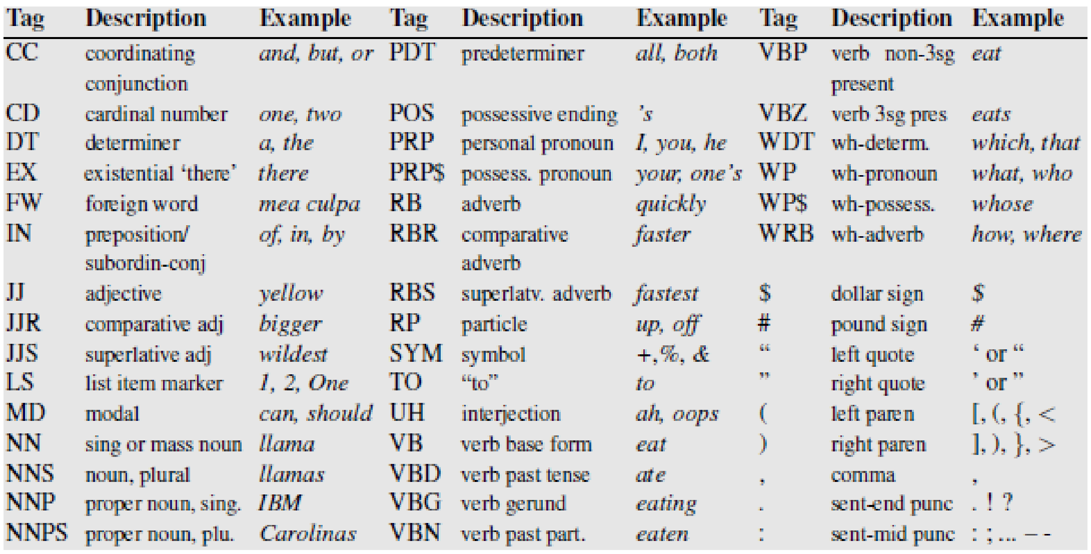


4.1 Open e Closed Classes

Parts-of-speech can be divided into two broad supercategories: closed class types and open class types. Closed classes are those with relatively fixed membership, such as prepositions—new prepositions are rarely coined. By contrast, nouns and verbs are open classes—new nouns and verbs like iPhone or to fax are continually being created or borrowed.

4.2 The Tagset

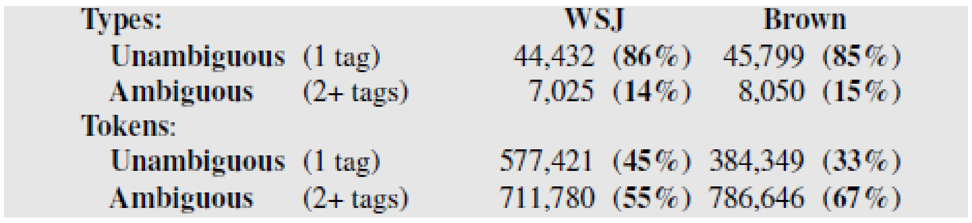
To do POS tagging, we need to choose a standard set of tags to work with. An important tagset for English is the 45-tag Penn Treebank tagset.



Tag	Description	Example	Tag	Description	Example	Tag	Description	Example
CC	coordinating conjunction	*and, but, or*	PDT	predeterminer	*all, both*	VBP	verb non-3sg present	*eat*
CD	cardinal number	*one, two*	POS	possessive ending	*'s*	VBZ	verb 3sg pres	*eats*
DT	determiner	*a, the*	PRP	personal pronoun	*I, you, he*	WDT	wh-determ.	*which, that*
EX	existential 'there'	*there*	PRP$	possess. pronoun	*your, one's*	WP	wh-pronoun	*what, who*
FW	foreign word	*mea culpa*	RB	adverb	*quickly*	WP$	wh-possess.	*whose*
IN	preposition/ subordin-conj	*of, in, by*	RBR	comparative adverb	*faster*	WRB	wh-adverb	*how, where*
JJ	adjective	*yellow*	RBS	superlatv. adverb	*fastest*	$	dollar sign	*$*
JJR	comparative adj	*bigger*	RP	particle	*up, off*	#	pound sign	*#*
JJS	superlative adj	*wildest*	SYM	symbol	+,%, &	“	left quote	‘ or “
LS	list item marker	*1, 2, One*	TO	“to”	*to*	”	right quote	’ or ”
MD	modal	*can, should*	UH	interjection	*ah, oops*	(	left paren	[, (, {, <
NN	sing or mass noun	*llama*	VB	verb base form	*eat*	)	right paren	],), }, >
NNS	noun, plural	*llamas*	VBD	verb past tense	*ate*	,	comma	,
NNP	proper noun, sing.	*IBM*	VBG	verb gerund	*eating*	.	sent-end punc	. ! ?
NNPS	proper noun, plu.	*Carolinas*	VBN	verb past part.	*eaten*	:	sent-mid punc	: ; ... – -

4.3 Measuring Ambiguity

Tagging is a disambiguation task; words are ambiguous—have more than one possible part-of-speech—and the goal is to find the correct tag for the situation. For example, book can be a verb (*book that flight*) or a noun (*hand me that book*). The goal of POS-tagging is to resolve these ambiguities, choosing the proper tag for the context.
But how common is tag ambiguity?



		WSJ	Brown
Types:			
Unambiguous	(1 tag)	44,432 (**86%**)	45,799 (**85%**)
Ambiguous	(2+ tags)	7,025 (**14%**)	8,050 (**15%**)
Tokens:			
Unambiguous	(1 tag)	577,421 (**45%**)	384,349 (**33%**)
Ambiguous	(2+ tags)	711,780 (**55%**)	786,646 (**67%**)

Nonetheless, many words are easy to disambiguate, because their different tags aren't equally likely. For example, *a* can be a determiner or the letter *a*, but the determiner sense is much more likely. This idea suggests a simplistic baseline algorithm for part-of-speech tagging: given an ambiguous word, choose the tag which is most frequent in the training corpus.

How good is this baseline? A standard way to measure the performance of part-of-speech taggers is accuracy: the percentage of tags correctly labeled. If we train on the WSJ training corpus and test on sections 22-24 of the same corpus the most-frequent-tag baseline achieves an accuracy of 92.34%. By contrast, the state of the art in part-of-speech tagging on this dataset is around 97% tag accuracy, a performance that is achievable by most algorithms (HMMs, MEMMs, neural networks, rule-based algorithms).

4.4 Hidden Markov Model Tagging

Disclaimer: *in this introduction part I will skip some concepts since they were already treated in the Machine Learning course. If you want to better understand these models, read the relative section in the book.*

The HMM is a sequence model. A sequence model or sequence classifier is a model whose job is to assign a label or class to each unit in a sequence, thus mapping a sequence of observations to a sequence of labels. An HMM is a probabilistic sequence model: given a sequence of units (words, letters, morphemes, sentences, whatever), it computes a probability distribution over possible sequences of labels and chooses the best label sequence.

4.4.1 Markov Chains

The HMM is based on augmenting the **Markov chain**. A Markov chain is a model that tells us something about the probabilities of sequences of random variables, states, each of which can take on values from some set. These sets can be words, or tags, or symbols representing anything, for example the weather.

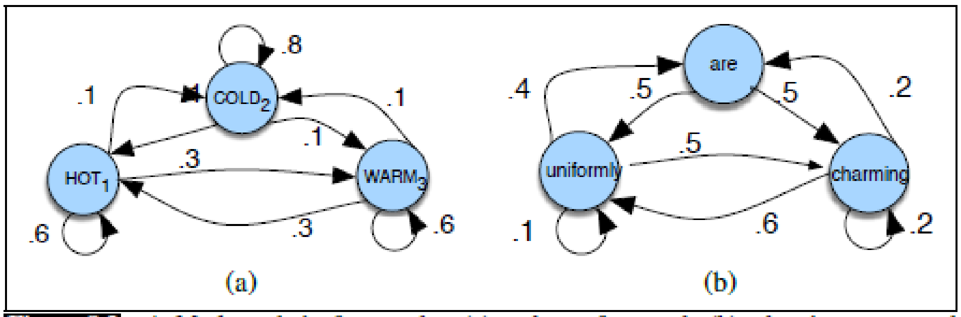


Figure 8.3 A Markov chain for weather (a) and one for words (b), showing states and transitions. A start distribution π is required; setting $\pi = [0.1, 0.7, 0.2]$ for (a) would mean a probability 0.7 of starting in state 2 (cold), probability 0.1 of starting in state 1 (hot), etc.

A Markov chain makes a very strong assumption that if we want to predict the future in the sequence, all that matters is the current state.

$$P(q_i = a|q_1...q_{i-1}) = P(q_i = a|q_{i-1})$$

A Markov chain is useful when we need to compute a probability for a sequence of **observable** events. In many cases, however, the events we are interested in are **hidden**: we don't observe them directly. For example we don't normally observe part-of-speech tags in a text. Rather, we see words, and must infer the tags from the word sequence.
That's why we need a **hidden Markov model**, which is specified by the following components:

$$< Q, A, O, B, \pi >$$

where Q is a set of N states, A a transition probability matrix, O a sequence of T observations, B a sequence of observation likelihoods, also called emission probabilities, each expressing the probability of an observation o_t being generated from a state i, and π an initial probability distribution over states.

A first-order hidden Markov model instantiates two simplifying assumptions. First, the Markov assumption already discussed above. Second, the probability of an output observation o_i depends only on the state that produced that observation q_i and not on any other states or any other observations.

4.4.2 HMM tagging as decoding

An HMM has two components, the A and B probabilities:

- The A matrix contains the tag transition probabilities $P(t_i|t_{i-1})$, which represent the probability of a tag occurring given the previous tag;
- The B emission probabilities $P(w_i|t_i)$ represent the probability, given a tag, that it will be associated with a given word.

For any model, such as an HMM, that contains hidden variables, the task of determining the hidden variables sequence corresponding to the sequence of observations decoding is called decoding.

Decoding: Given as input a HMM $\lambda = (A, B)$ and a sequence of observations $O = o_1, o_2, ..., o_T$, find the most probable sequence of states $Q = q_1 q_2 q_3 ... q_T$.

In our case the goal of HMM decoding is to choose the tag sequence t_1^n that is most probable given the observation sequence of n words w_1^n:

$$\hat{t_1^n} = \arg\max_{t_1^n} P(t_1^n|w_1^n)$$

How do we compute this value? We'll use Bayes' rule:

$$\hat{t_1^n} = \arg\max_{t_1^n} \frac{P(w_1^n|t_1^n)P(t_1^n)}{P(w_1^n)}$$

By dropping the denominator:

$$\hat{t_1^n} = \arg\max_{t_1^n} P(w_1^n|t_1^n)P(t_1^n)$$

HMM taggers make two further simplifying assumptions. The first is that the probability of a word appearing depends only on its own tag and is independent of neighboring words and tags:

$$P(w_1^n|t_1^n) \approx \prod_{i=1}^{n} P(w_i|t_i)$$

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

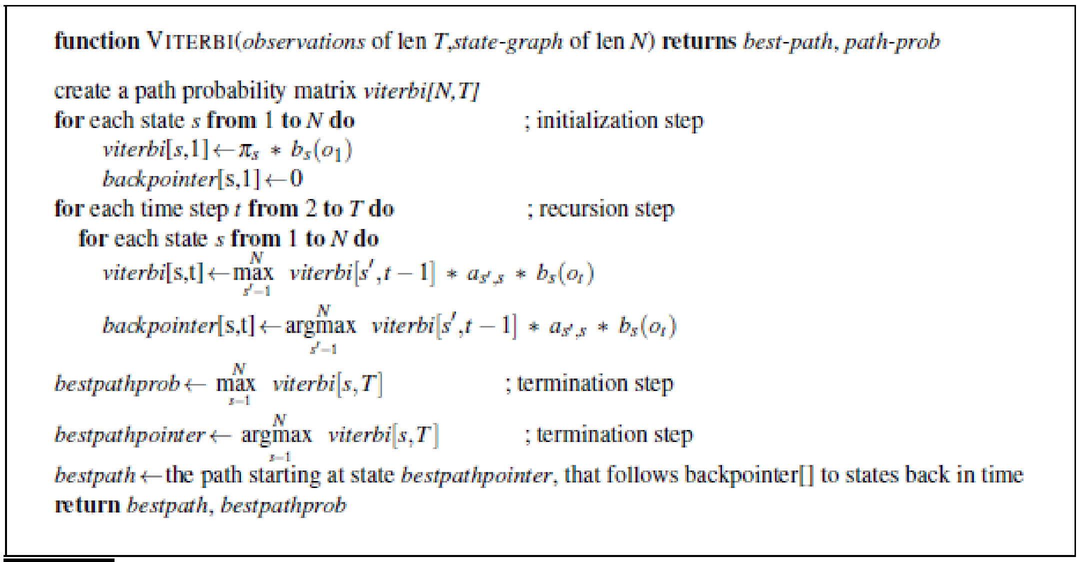
$$P(t_1^n) \approx \prod_{i=1}^{n} P(t_i|t_{i-1})$$

Plugging the simplifying assumptions into our equation for $\hat{t_1^n}$ we obtain:

$$\hat{t_1^n} = \arg\max_{t_1^n} P(w_1^n|t_1^n)P(t_1^n) = \arg\max_{t_1^n} \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1})$$

Note that the two parts correspond neatly to the **B emission probability** and **A transition probability** that we just defined above!

4.4.3 The Viterbi Algorithm



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

create a path probability matrix viterbi[N,T]
for each state s from 1 to N do                          ; initialization step
      viterbi[s,1] ← $\pi_s * b_s(o_1)$
      backpointer[s,1] ← 0
for each time step t from 2 to T do                      ; recursion step
   for each state s from 1 to N do
      viterbi[s,t] ← $\max_{s'=1}^{N}$ viterbi[s',t-1] $* a_{s',s} * b_s(o_t)$
      backpointer[s,t] ← $\operatorname{argmax}_{s'=1}^{N}$ viterbi[s',t-1] $* a_{s',s} * b_s(o_t)$
bestpathprob ← $\max_{s=1}^{N}$ viterbi[s,T]               ; termination step
bestpathpointer ← $\operatorname{argmax}_{s=1}^{N}$ viterbi[s,T]   ; termination step
bestpath ← the path starting at state bestpathpointer, that follows backpointer[] to states back in time
return bestpath, bestpathprob
```

Figure 8.5 Viterbi algorithm for finding the optimal sequence of tags. Given an observation sequence and an HMM $\lambda = (A, B)$, the algorithm returns the state path through the HMM that assigns maximum likelihood to the observation sequence.

The Viterbi algorithm first sets up a probability matrix or lattice, with one column for each observation o_t and one row for each state in the state graph.
Each cell of the trellis, $v_t(j)$, represents the probability that the HMM is in state j after seeing the first t observations and passing through the most probable state sequence $q_1, ..., q_{t-1}$, given the HMM λ.

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

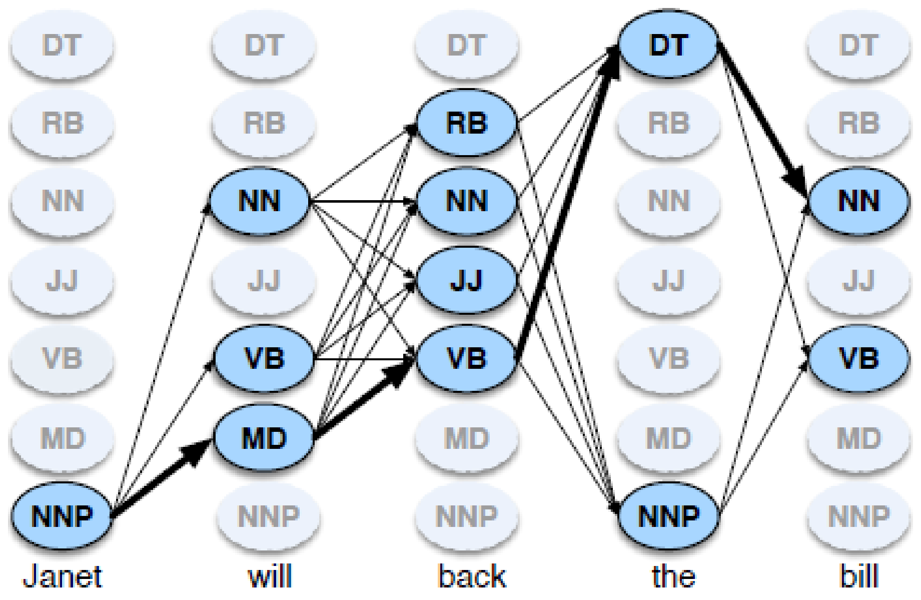
Given that we had already computed the probability of being in every state at time $t-1$ we compute the Viterbi probability by taking the most probable of the extensions of the paths that lead to the current cell:

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

where

- $v_{t-1}(i)$ is the previous Viterbi path probability from the previous time step;
- a_{ij} is the transition probability from previous state q_i to current state q_j;

- $b_j(o_t)$ is the state observation likelihood of the observation symbol o_t given the current state j.



Practical HMM taggers have a number of extensions of this simple model. One important missing feature is a wider tag context. Extending the algorithm from bigram to trigram taggers gives a small (perhaps a half point) increase in performance, but conditioning on two previous tags instead of one requires a significant change to the Viterbi algorithm.
Moreover, when the number of states grows very large, the vanilla Viterbi algorithm can be slow. The complexity of the algorithm is $O(N^2T)$, where N (the number of states) can be large for trigram taggers, which have to consider every previous pair of the 45 tags, resulting in $45^3 = 91,125$ computations per column. One common solution to the complexity problem is the use of **beam search** decoding. We will not discuss this approach, but just to give an idea in beam search, instead of keeping the entire column of states at each time point t, we just keep the best few hypothesis at that point.

5 Formal Grammars

The word **syntax** comes from the Greek, it means "setting out together or arrangement", and refers to the way words are arranged together. We will focus on the **context-free grammars**, since they are the backbone of many formal models of the syntax of natural language (and, for that matter, of computer languages). As such, they are integral to many computational applications, including grammar checking, semantic interpretation, dialogue understanding, and machine translation. They are powerful enough to express sophisticated relations among the words in a sentence, yet computationally tractable enough that efficient algorithms exist for parsing sentences with them.

5.1 Constituency

The fundamental idea of consistency is that groups of words may behave as a single unit or phrase, called a **constituent**. A significant part of developing a grammar involves discovering the inventory of constituents present in the language.

How do words group together in English? Consider the noun phrase, a sequence of words surrounding at least one noun, for example:

Harry the Horse
the Broadway coppers
they

What evidence do we have that these words group together (or "form constituents")? One piece of evidence is that they can all appear in similar syntactic environments, for example, before a verb. Note however that while the whole noun phrase can occur before a verb, this is not true of each of the individual words that make up a noun phrase. There is nothing easy or obvious about how we come up with the right set of constituents and the rules that govern how they combine. That's why there are so many different theories of grammar and competing analysis of the same data. The approach that we will use here will be very generic and it will not correspond to any modern linguistic theory of grammar.

5.2 Context-Free Grammars

Disclaimer: *in this section I will skip some concepts since they were already treated in the Formal Languages and Compilers course. As always, if you want to go deeper, read the relative sections on the book.*

The most widely used formal system for modeling constituent structure in English and other natural languages is the **Context-Free Grammar**, or **CFG**. Context-free grammars are also called **Phrase-Structure Grammars**, and the formalism is equivalent to **Backus-Naur Form**, or **BNF**.

A context-free grammar consists of a set of rules or productions, each of which expresses the ways that symbols of the language can be grouped and ordered together, and a lexicon of words and symbols.

- Terminals ($\sum$): for us these will be **words**
- Non-terminals (N):
  - The constituents (like noun-phrase, verb-phrase...)
  - The preterminals (e.g. POS tags)
  - A special symbol S to start from
- Rules (R): $<$ single non-terminal $>$ $\rightarrow$ $<$ any number of terminals and non-terminals $>$

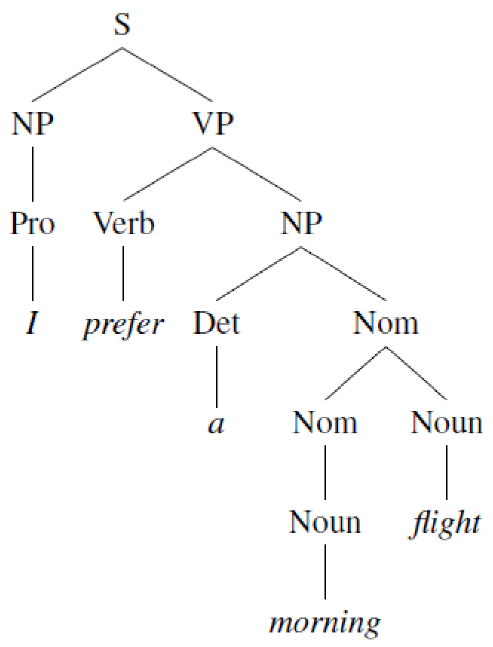
Here an example:

NP $\rightarrow$ Det Nominal
NP $\rightarrow$ ProperNoun
Nominal $\rightarrow$ Noun — Nominal Noun

Note that in the third rule we have an explicit disjunction (OR) and a recursive definition, i.e. the same non-terminal on the right and left-side of the rule.

A CFG can be thought of in two ways: as a device for generating sentences and as a device for assigning a structure to a given sentence. Viewing a CFG as a generator, we can read the $\rightarrow$

arrow as “rewrite the symbol on the left with the string of symbols on the right”. The sequence of rule expansions is called **derivation**. It is common to represent a derivation by using a **parse tree**, like the one below:



generated according to the following grammar:

Grammar Rules			Examples
S	→	*NP VP*	I + want a morning flight
NP	→	*Pronoun*	I
	|	*Proper-Noun*	Los Angeles
	|	*Det Nominal*	a + flight
Nominal	→	*Nominal Noun*	morning + flight
	|	*Noun*	flights
VP	→	*Verb*	do
	|	*Verb NP*	want + a flight
	|	*Verb NP PP*	leave + Boston + in the morning
	|	*Verb PP*	leaving + on Thursday
PP	→	*Preposition NP*	from + Los Angeles

The problem of mapping from a string of words to its parse tree is called **syntactic parsing**.

5.3 Noun Phrases

Let’s now consider the rule *NP* → *Det Nominal* more in detail, since most of the complexity of English noun phrases is hidden in this rule. These noun phrases consist of a **head**, the central noun in the noun phrase, along with various modifiers that can occur before or after the head noun.

5.3.1 Agreement

By **agreement** we have in mind constraints that hold among various constituents that take part in a rule or set of rules. For example, in English, determiners and the head nouns in NPs have to agree in their number:

This flight and not *This flights*
Those flights and not *Those flight*

The problem is that our earlier NP rule does not capture this constraint: it accepts and assigns correct structures to grammatical examples (this flight), but it is also happy with incorrect examples. Such a rule is said to **overgenerate**.

5.4 Verb Phrases

English VPs consist of a head verb along with 0 or more following constituents which we will call **arguments**.

VP → Verb *disappear*
VP → Verb NP *prefer a morning flight*
VP → Verb NP PP *leave Boston in the morning*
VP → Verb PP *leaving on Thursday*

5.4.1 Subcategorization

Verb phrases can be significantly more complicated than this. Many other kinds of constituents, such as an entire embedded sentence, can follow the verb. Similarly, another potential constituent of the VP is another VP. This is often the case for verbs like *want, would like, try, intend, need.*
While a verb phrase can have many possible kinds of constituents, not every verb is compatible with every verb phrase. For example, the verb *want* can be used either with an NP complement (*I want a flight...*) or with an infinitive VP complement (*I want to fly to...*). By contrast, a verb like *find* cannot take this sort of VP complement (**I found to fly to Dallas*).
This idea that verbs are compatible with different kinds of complements is a very old one; traditional grammar distinguishes between transitive verbs like *find*, which take a direct object NP (*I found a flight*), and intransitive verbs like *disappear*, which do not (**I disappeared a flight*). Where traditional grammars **subcategorize** verbs into these two categories (transitive and intransitive), modern grammars distinguish as many as 100 subcategories.

5.5 Solution for Agreement and Subcategorization

So, as with agreement phenomena, we need a way to formally express the constraints. A possible solution consists of creating ad-hoc sub-classes. It works and stays within the power of CFGs, however it increases the size of the grammar in a dramatic way. Other solutions exist, like the so called **feature-based grammars**, which however do not solve completely the problem.

5.6 Treebanks

Sufficiently robust grammars consisting of context-free grammar rules can be used to assign a parse tree to any sentence. This means that it is possible to build a corpus where every sentence in the collection is paired with a corresponding parse tree. Such a syntactically annotated corpus is called a **treebank**.
A wide variety of treebanks have been created, generally through the use of parsers (of the sort described in the next sections) to automatically parse each sentence, followed by the use of humans (linguists) to hand-correct the parses. Penn TreeBank is a widely used treebank.

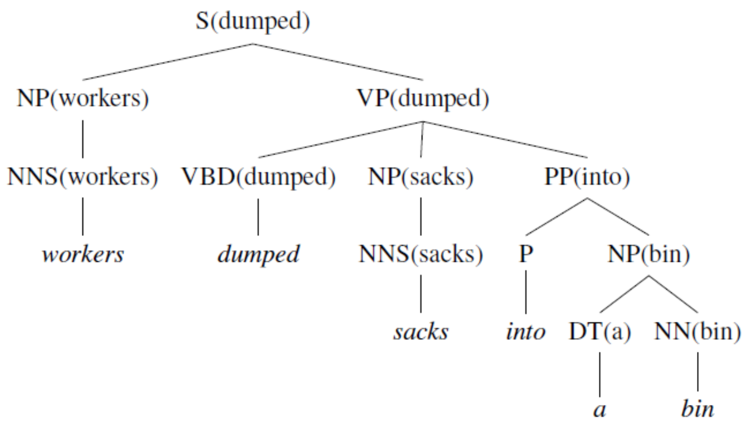
Note that treebanks implicitly define a grammar for the language covered in the treebank: we can simply take the local rules that make up the sub-trees in all the trees in the collection and we obtain a grammar. Of course, a not complete one, but if you have decent size corpus, you'll have a grammar with decent coverage.
However, such grammars tend to be very flat due to the fact that they tend to avoid recursion. For example, among the approximately 4,500 different rules for expanding VPs there are separate rules for PP sequences of any length and every possible arrangement of verb arguments:

VP $\rightarrow$ VBD PP
VP $\rightarrow$ VBD PP PP
VP $\rightarrow$ VBD PP PP PP
VP $\rightarrow$ VBD PP PP PP PP

This fact (and others) about the treebank grammars pose problems for probabilistic parsing algorithms. For this reason, it is common to make various modifications to a grammar extracted from a treebank.

5.6.1 Heads and Head Finding

Regarding probabilistic parsing, of which we will talk later, one very important and common task is that of finding heads in treebank trees. In one simple model of lexical heads, each context-free rule is associated with a head. The head is the word in the phrase that is grammatically the most important. Heads are passed up the parse tree; thus, each non-terminal in a parse tree is annotated with a single word, which is its **lexical head**.



For the generation of such a tree, each CFG rule must be augmented to identify one right-side constituent to be the head daughter. The headword for a node is then set to the headword of its head daughter. Choosing these head daughters is simple for textbook examples (NN is the head of NP) but is complicated and indeed controversial for most phrases. Modern linguistic theories of syntax generally include a component that defines heads.

An alternative approach to find a head is used in most practical computational systems. Instead of specifying head rules in the grammar itself, heads are identified dynamically in the context of trees for specific sentences. In other words, once a sentence is parsed, the resulting tree is walked to decorate each node with the appropriate head. Most current systems rely on a simple set of hand-written rules:

If the last word is tagged POS, return last-word.
Else search from right to left for the first child which is an NN, NNP, NNPS, NX, POS, or JJR.
Else search from left to right for the first child which is an NP.
Else search from right to left for the first child which is a $, ADJP, or PRN.
Else search from right to left for the first child which is a CD.
Else search from right to left for the first child which is a JJ, JJS, RB or QP.
Else return the last word.

6 Syntactic Parsing

Syntactic parsing is the task of recognizing a sentence and assigning a syntactic structure to it. Since they are based on a purely declarative formalism, context-free grammars don't specify how the parse tree for a given sentence should be computed. We therefore need to specify algorithms that employ these grammars to efficiently produce correct trees.

6.1 Ambiguity

Ambiguity is perhaps the most serious problem faced by syntactic parsers. Structural ambiguity occurs when the grammar can assign more than one parse to a sentence.

Let's consider for example the following sentence:

I shot an elephant in my pajamas.

Structural ambiguity, appropriately enough, comes in many forms. Two common kinds of ambiguity are **attachment ambiguity** and **coordination ambiguity**.
A sentence has an attachment ambiguity if a particular constituent can be attached to the parse tree at more than one place, e.g.

We saw the Eiffel Tower flying to Paris.

where the gerundive-VP flying to Paris can be part of a gerundive sentence whose subject is *the Eiffel Tower* or it can be an adjunct modifying the VP headed by *saw.*

In coordination ambiguity different sets of phrases can be conjoined by a conjunction like *and.* For example, the phrase *old men and women* can be bracketed as *[old [men and women]]*, referring to *old men* and *old women*, or as *[old men] and [women]*, in which case it is only the men who are old.

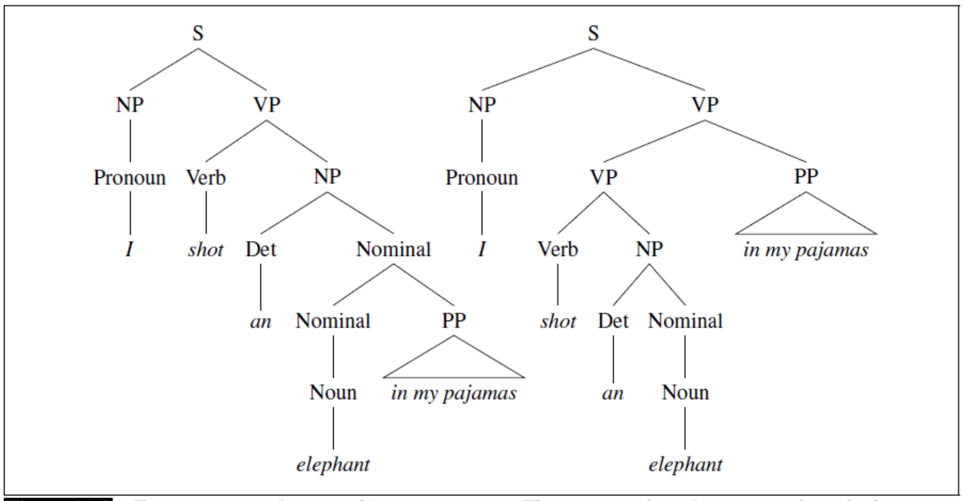


Figure 11.2 Two parse trees for an ambiguous sentence. The parse on the left corresponds to the humorous reading in which the elephant is in the pajamas, the parse on the right corresponds to the reading in which Captain Spaulding did the shooting in his pajamas.

The fact that there are many grammatically correct but semantically unreasonable parses for naturally occurring sentences is an irksome problem that affects all parsers. Effective disambiguation algorithms require statistical, semantic, and contextual knowledge sources that vary in how well they can be integrated into parsing algorithms.

6.1.1 Top-down and Bottom-up approaches

Before talking about more complex algorithms, let's introduce a baseline approach, which consists of seeing parsing as a search. More precisely, the parser can be viewed as searching through the space of all possible parse trees to find the correct parse tree for the sentence. Just as the search space of possible paths is defined by the structure of a FSA, so the search space of possible parse trees is defined by the grammar.
There exist two different ways with which we can perform this task:

- Top-Down Search: a top-down parser searches for a parse tree by trying to build from the root node S down to the leaves.
- Bottom-Up Search: in bottom-up parsing the parser starts with the words of the input and tries to build trees from the words up, by applying rules from the grammar one at a time. The parse is successful if the parser succeeds in building a tree rooted in the start symbol S that covers all of the input.

Each of these two architectures has its own advantages and disadvantages. The top-down approach only searches for trees that can be answers, but also suggests trees that are not consistent with any of the words, while the bottom-up approach only forms trees consistent with the words, but suggests trees that make no sense globally.

With that said, till now we left out how to keep track of the search space and how to make choices, like how to choose the node to expand next and which grammar rule to use to expand a node. One approach is called **backtracking**:

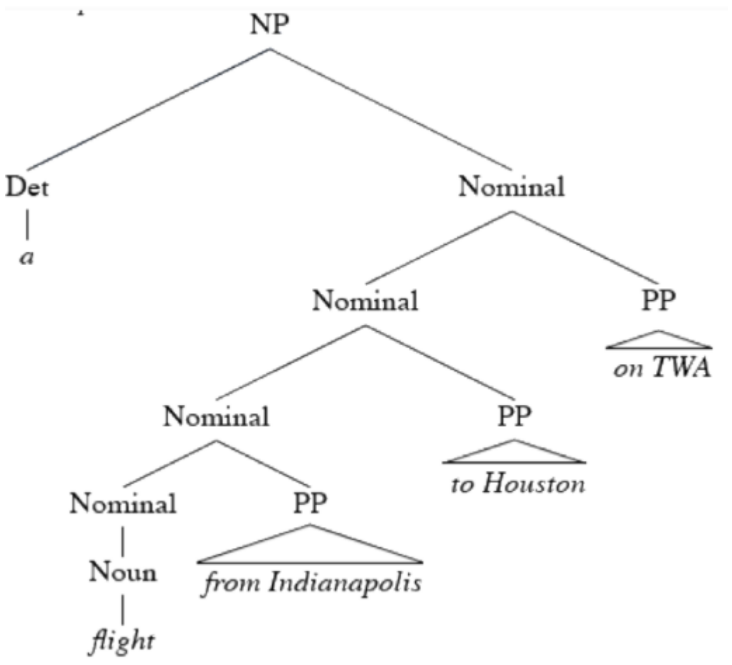
- Make a choice, if it works out then fine

- If not then back up and make a different choice

This method is very simple, indeed it also has some problems. Backtracking methods are doomed because of two inter-related problems:

- **Structural ambiguity**: the grammar assigns more than one possible parse to a phrase (we have already discussed about this)
- **Repeated parsing** of subtrees

The structural ambiguity is exactly the problem that for example occurs in the sentence *I shot an elephant in my pajamas.* The repeated parsing problem, instead, occur when we essentially redo something we have already done, leading to duplicated work. Consider for example the sentence *"A flight from Indianapolis to Houston on TWA"*. The correct parse is the following:



If we now assume that we have a top-down parser making choices between the two rules

Nominal → Noun
Nominal → Nominal PP

Statically choosing the rules in this order leads to a very bad result, since the algorithm will **fail three times** before finding the correct parse tree.

6.1.2 Dynamic Programming approaches

Dynamic Programming provides a powerful framework for addressing these problems, just as it did with the Minimum Edit Distance and Viterbi. Recall that dynamic programming approaches systematically fill in tables of solutions to sub-problems. When complete, the tables contain the solution to all the sub-problems needed to solve the problem as a whole. In the case of syntactic parsing, these sub-problems represent parse trees for all the constituents detected in the input. The dynamic programming advantage arises from the context-free nature of our grammar rules — once a constituent has been discovered in a segment of the input we can record its presence and make it available for use in any subsequent derivation that might require it. This provides both time and storage efficiencies since subtrees can be looked up in a table, not reanalyzed.

Examples of algorithms that adopt this approach are the **CKY** algorithm and the **Earley** algorithm. We will not analyze these algorithms (check the book if you are interested), but for us it is enough to say that even if the CKY parsing algorithm can represent the ambiguities we discussed in an efficient way, it is not equipped to resolve them. We therefore need a different approach, a **probabilistic approach**. Before going on in that direction, however, let's add a small note: many language processing tasks do not require complex, complete parse trees for all inputs. For these tasks, a partial parse, or shallow parse, of input sentences may be sufficient. There are many different approaches to partial parsing. One of these is called **chunking**. Chunking is the process of identifying and classifying the flat, non-overlapping segments of a sentence that constitute the basic non-recursive phrases corresponding to the major content-word parts-of-speech: noun phrases, verb phrases, adjective phrases, and prepositional phrases.

$[_{NP}$ *The morning flight]* $[_{PP}$ *from]* $[_{NP}$ *Denver]* $[_{VP}$ *has arrived.]*

This bracketing notation makes clear the two fundamental tasks that are involved in chunking: segmenting (finding the non-overlapping extents of the chunks) and labeling (assigning the correct tag to the discovered chunks).

State-of-the-art approaches to chunking use supervised machine learning to train a chunker by using annotated data as a training set and training any sequence labeler. Given a training set, any sequence model can be used. For example, we can use a simple feature-based model, using features like the words and parts-of-speech within a 2 word window, and the chunk tags of the preceding inputs in the window. In training, each training vector would consist of the values of 13 features; the two words to the left of the decision point, their parts-of-speech and chunk tags, the word to be tagged along with its part-of-speech, the two words that follow along with their parts-of speech, and the correct chunk tag. During classification, the classifier is given the same vector without the answer and assigns the most appropriate tag from its tagset. Viterbi decoding is commonly used.

7 Statistical Parsing

7.1 Probabilistic Context Free Grammars (PCFG)

The simplest augmentation of the context-free grammar is the Probabilistic Context-Free Grammar (PCFG), also known as the Stochastic Context-Free Grammar (SCFG), first proposed by Booth (1969).

That is, a PCFG differs from a standard CFG by augmenting each rule in R with a conditional probability:

$$A \rightarrow \beta \; [p]$$

Here p expresses the probability that the given non-terminal A will be expanded to the sequence β. That is, p is the conditional probability of a given expansion β given the left-hand-side (LHS) non-terminal A. We can represent this probability as:

$$P(A \to \beta)$$

$$P(A \to \beta | A)$$

$$P(RHS | LHS)$$

Thus, if we consider all the possible expansions of a non-terminal, the sum of their probabilities must be 1:

$$\sum_{\beta} P(A \to \beta) = 1$$

How are PCFGs used? A PCFG can be used to estimate a number of useful probabilities concerning a sentence and its parse tree(s), including the probability of a particular parse tree (**useful in disambiguation**) and the probability of a sentence or a piece of a sentence (**useful in language modeling**).

7.2 PCFGs for Disambiguation

A PCFG assigns a probability to each parse tree T (i.e., each derivation) of a sentence S. The probability of a particular parse T is defined as the product of the probabilities of all the n rules used to expand each of the n non-terminal nodes in the parse tree T, where each rule i can be expressed as $LHS_i \to RHS_i$:

$$P(T, S) = \prod_{i=1}^{n} P(RHS_i | LHS_i)$$

The resulting probability $P(T, S)$ is both the joint probability of the parse and the sentence and also the probability of the parse $P(T)$. How can this be true? First, by the definition of joint probability:

$$P(T, S) = P(T)P(S|T)$$

But since a parse tree includes all the words of the sentence, $P(S|T)$ is 1. Thus,

$$P(T, S) = P(T)P(S|T) = P(T)$$

7.3 PCFGs for Language Modeling

A second attribute of a PCFG is that it assigns a probability to the string of words constituting a sentence.
The probability of an unambiguous sentence is $P(T, S) = P(T)$.

The probability of an ambiguous sentence is the sum of the probabilities of all the parse trees for the sentence.

An additional feature of PCFGs that is useful for language modeling is their ability to assign a probability to substrings of a sentence. For example, suppose we want to know the probability of the next word w_i in a sentence given all the words we've seen so far $w_1, ...w_{i-1}$. We saw that a simple approximation of this probability can be obtained by using N-grams, conditioning on only the last word or two instead of the entire context; but the fact that the N-gram model can only make use of a couple words of context means it is ignoring potentially useful prediction cues. PCFGs allow us to condition on the entire previous context.

7.4 Ways to Learn PCFG Rule Probabilities

Where do PCFG rule probabilities come from? The simplest way is to use a treebank, a corpus of already parsed sentences. Given a treebank, we can compute the probability of each expansion of a non-terminal by counting the number of times that expansion occurs and then normalizing:

$$P(\alpha \to \beta|\alpha) = \frac{Count(\alpha \to \beta)}{\sum_\gamma Count(\alpha \to \gamma)}$$

7.5 Problems with PCFGs

While probabilistic context-free grammars are a natural extension to context-free grammars, they have two main problems as probability estimators:

- **Poor independence assumptions:** CFG rules impose an independence assumption on probabilities, resulting in poor modeling of structural dependencies across the parse tree.
- **Lack of lexical conditioning:** CFG rules don't model syntactic facts about specific words, leading to problems with subcategorization ambiguities, preposition attachment, and coordinate structure ambiguities.

Let's analyze these problems in more detail.

7.5.1 Structural Dependencies

Recall that in a CFG the expansion of a non-terminal is independent of the context, that is, of the other nearby non-terminals in the parse tree. Similarly, in a PCFG, the probability of a particular rule like $NP \to DetN$ is also independent of the rest of the tree. By definition, the probability of a group of independent events is the product of their probabilities.
Unfortunately, this CFG independence assumption results in poor probability estimates. This is because in English the choice of how a node expands can after all depend on the location of the node in the parse tree. For example, in English it turns out that NPs that are syntactic subjects are far more likely to be pronouns, and NPs that are syntactic objects are far more likely to be non-pronominal:

Unfortunately, there is no way to represent this contextual difference in the probabilities in a PCFG; there is no way to capture the fact that in subject position, the probability for $NP \to PRP$ should go up to .91, while in object position, the probability for $NP \to DT\ NN$

	Pronoun	Non-Pronoun
Subject	91%	9%
Object	34%	66%

should go up to .66. These dependencies could be captured if the probability of expanding an NP as a pronoun (e.g., $NP \rightarrow PRP$) versus a lexical NP (e.g., $NP \rightarrow DT\ NN$) were conditioned on whether the NP was a subject or an object. We will see the **parent annotation** technique for adding this kind of conditioning.

7.5.2 Lexical Dependencies

A second class of problems with PCFGs is their lack of sensitivity to the words in the parse tree. Since prepositional phrases in English can modify a noun phrase or a verb phrase, when a parser finds a prepositional phrase, it must decide where to attach it into the tree.

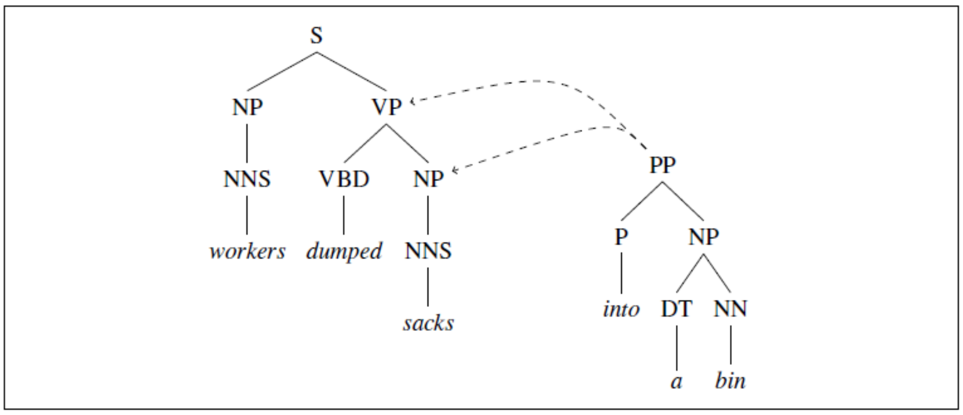


Figure 12.6 Another view of the preposition attachment problem. Should the *PP* on the right attach to the *VP* or *NP* nodes of the partial parse tree on the left?

Why doesn't a PCFG already deal with PP attachment ambiguities? The two parse trees have almost exactly the same rules; they differ only since one has this rule:

$VP \rightarrow VBD\ NP\ PP$

while the other one has:

$VP \rightarrow VBD\ NP$
$NP \rightarrow NP\ PP$

Depending on how these probabilities are set, a PCFG will always either prefer NP attachment or VP attachment. As it happens, NP attachment is slightly more common in English, so if we trained these rule probabilities on a corpus, we might always prefer NP attachment, causing us to misparse this sentence.

What information in the input sentence lets us know that the above example requires VP attachment? It should be clear that these preferences come from the identities of the verbs, nouns, and prepositions. It seems that the affinity between the verb *dumped* and the preposition *into* is greater than the affinity between the noun *sacks* and the preposition *into*, thus leading to VP attachment.

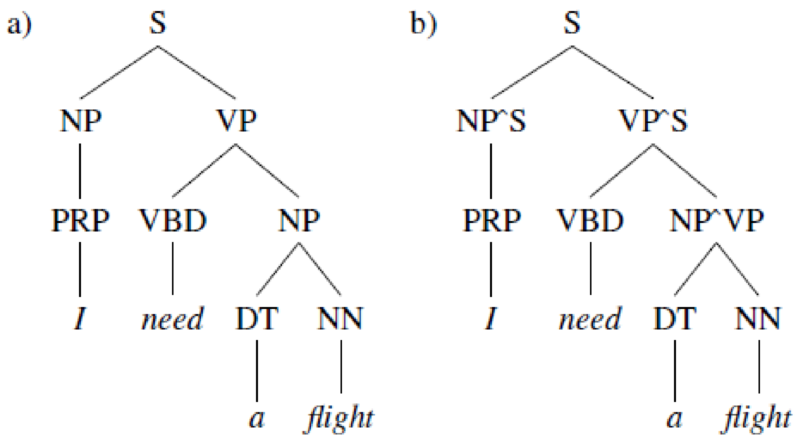
Coordination ambiguities are another case in which lexical dependencies are the key to choosing the proper parse. If we consider the phrase *dogs in houses and cats.*, because *dogs* is semantically a better conjunct for *cats* than *houses* (and because most dogs can't fit inside cats), the parse *[dogs in [NP houses and cats]]* is intuitively unnatural and should be dispreferred. The two parses, however, have exactly the same PCFG rules, and thus a PCFG will assign them the same probability.

7.6 Improving PCFGs by Parent Annotation

Let's start with the first of the two problems with PCFGs mentioned above: their inability to model structural dependencies.

One idea would be to split the NP non-terminal into two versions: one for subjects, one for objects. Having two nodes (e.g., $NP_{subject}$ and NP_{object}) would allow us to correctly model their different distributional properties, since we would have different probabilities for the rule $NP_{subject} \rightarrow PRP$ and the rule $NP_{object} \rightarrow PRP$.

One way to implement this intuition of splits is to do **parent annotation (Johnson, 1998)**, in which we annotate each node with its parent in the parse tree. Thus, an NP node that is the subject of the sentence and hence has parent S would be annotated NPˆS, while a direct object NP whose parent is VP would be annotated NPˆVP.



To deal with cases in which parent annotation is insufficient, we can also handwrite rules that specify a particular node split based on other features of the tree. For example, to distinguish between complementizer IN and subordinating conjunction IN, both of which can have the same parent, we could write rules conditioned on other aspects of the tree such as the lexical identity (the lexeme that is likely to be a complementizer, as a subordinating conjunction).
Node-splitting is not without problems; it increases the size of the grammar and hence reduces the amount of training data available for each grammar rule, leading to overfitting. Thus, it is important to split to just the correct level of granularity for a particular training set.

While early models employed hand-written rules to try to find an optimal number of non-terminals, modern models automatically search for the optimal splits. One example is the **split and merge algorithm** of Petrov et. al (2006), whose performance is the best of any parsing algorithm on the Penn Treebank.

7.7 Probabilistic Lexicalized CFGs

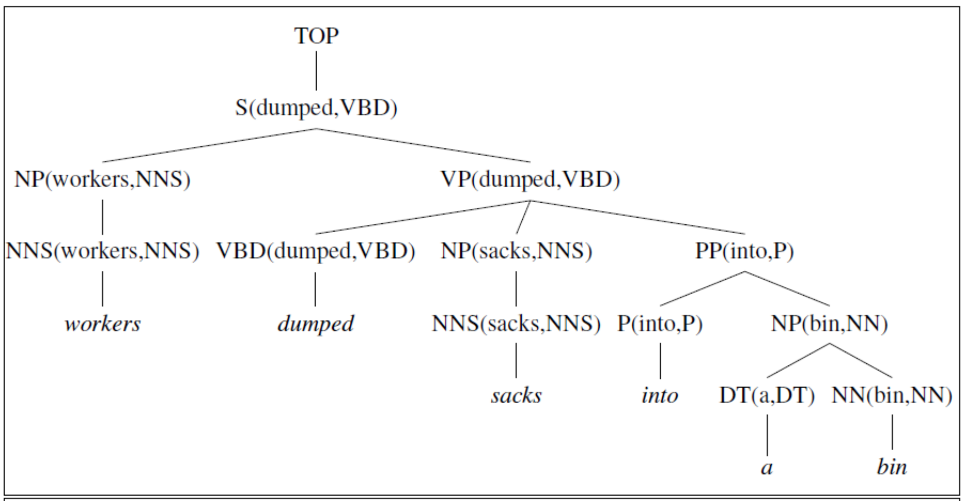
In this section, we discuss an alternative family of models in which instead of modifying the grammar rules, we modify the probabilistic model of the parser to allow for lexicalized rules. The resulting family of lexicalized parsers includes the well-known **Collins parser (Collins, 1999)** and **Charniak parser (Charniak, 1997)**.

We saw in section 5.6.1 that syntactic constituents could be associated with a lexical head, and we defined a lexicalized grammar in which each non-terminal in the tree is annotated with its lexical head, where a rule like $VP \rightarrow VBD\ NP\ PP$ would be extended as

$$VP(dumped) \rightarrow VBD(dumped)\ NP(sacks)\ PP(into)$$

In the standard type of lexicalized grammar, we actually make a further extension, which is to associate the **head tag**, the part-of-speech tags of the headwords, with the non-terminal symbols as well.

$$VP(dumped, VBD) \rightarrow VBD(dumped, VBD)\ NP(sacks, NNS)\ PP(into, P)$$



Internal Rules					Lexical Rules		
TOP	→	S(dumped,VBD)			NNS(workers,NNS)	→	workers
S(dumped,VBD)	→	NP(workers,NNS)	VP(dumped,VBD)		VBD(dumped,VBD)	→	dumped
NP(workers,NNS)	→	NNS(workers,NNS)			NNS(sacks,NNS)	→	sacks
VP(dumped,VBD)	→	VBD(dumped, VBD)	NP(sacks,NNS)	PP(into,P)	P(into,P)	→	into
PP(into,P)	→	P(into,P)	NP(bin,NN)		DT(a,DT)	→	a
NP(bin,NN)	→	DT(a,DT)	NN(bin,NN)		NN(bin,NN)	→	bin

A natural way to think of a lexicalized grammar is as a parent annotation, that is, as a simple context-free grammar with many copies of each rule, one copy for each possible headword/head tag for each constituent.
Note that in the image above **Internal rules** are distinguished from **Lexical rules**, since they are associated with very different kinds of probabilities. The lexical rules are deterministic, that is, they have probability 1.0 since a lexicalized pre-terminal like $NN(bin, NN)$ can only expand to the word *bin*. But for the internal rules, we need to estimate probabilities.

Now, how can we compute the MLE estimate for the probability for the rule $VP(dumped, VBD) \rightarrow VBD(dumped, VBD)\ NP(sacks, NNS)\ PP(into, P)$?

$$\frac{C(VP(dumped, VDB) \rightarrow VDB(dumped, VDB)\ NP(sacks, NNS)\ PP(into, P))}{C(VP(dumped, VDB))}$$

But there's no way we can get good estimates of counts, because they are so specific: we're unlikely to see many (or even any) instances of a sentence with a verb phrase headed by *dumped* that has one NP argument headed by *sacks* and a PP argument headed by *into*. In other words, counts of fully lexicalized PCFG rules like this will be far too sparse, and most rule probabilities will come out 0.

The idea of lexicalized parsing is to make some further independence assumptions to break down each rule so that we would estimate the probability

$$P(VP(dumped, VBD) \rightarrow VBD(dumped, VBD)\ NP(sacks, NNS)\ PP(into, P))$$

as the product of smaller independent probability estimates for which we could acquire reasonable counts. One such method is the **Collins parsing method**.

7.7.1 The Collins parser

Modern statistical parsers differ in exactly which independence assumptions they make. In this section we describe a simplified version of Collins's worth knowing about.

The first intuition of the Collins parser is to think of the right-hand side of every (internal) CFG rule as consisting of a head non-terminal, together with the nonterminals to the left of the head and the non-terminals to the right of the head:

$$LHS \rightarrow L_n L_{n-1} ... L_1 H R_1 ... R_{n-1} R_n$$

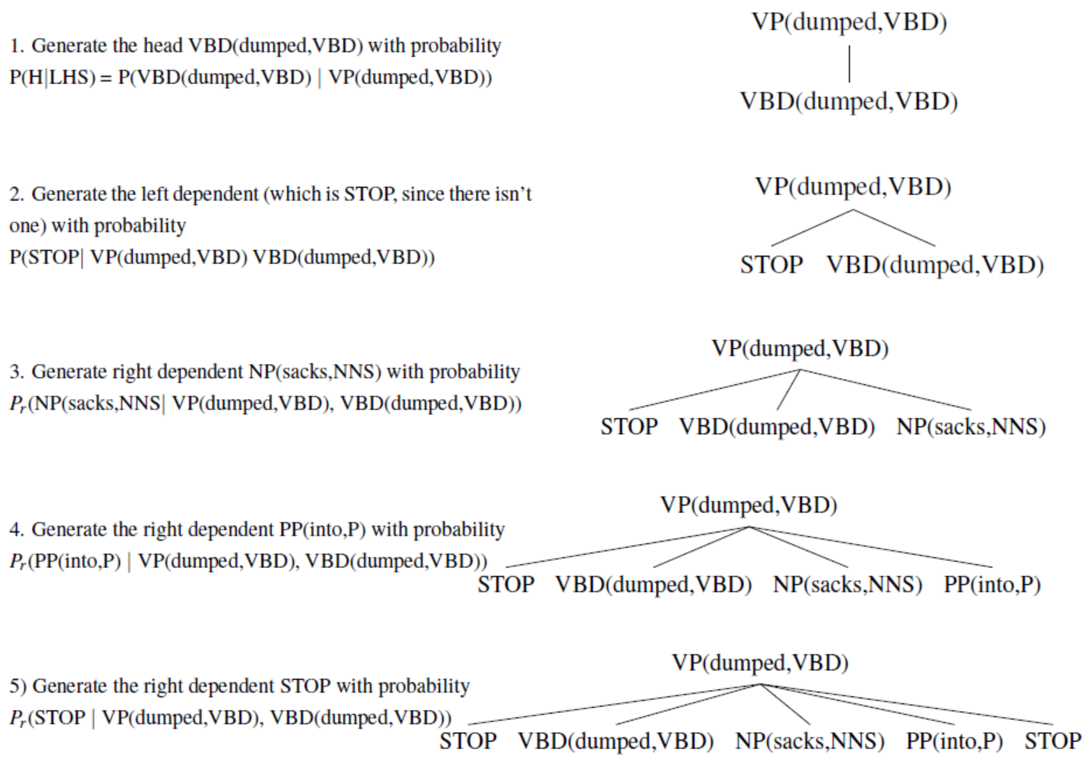
Now, instead of computing a single MLE probability for this rule, we are going to break down this rule via a neat generative story, a slight simplification of what is called **Collins Model 1**. This new generative story is that given the left-hand side, we first generate the head of the rule and then generate the dependents of the head, one by one, from the inside out. Each of these generation steps will have its own probability. We also add a special STOP non-terminal at the left and right edges of the rule; this non-terminal allows the model to know when to stop generating dependents on a given side. We generate dependents on the left side of the head until we've generated STOP on the left side of the head, at which point we move to the right side of the head and start generating dependents there until we generate STOP.

$$P(VP(dumped, VBD) \rightarrow STOP\ VBD(dumped, VBD)\ NP(sacks, NNS)\ PP(into, P)\ STOP)$$

Below you can see the generative story for this augmented rule. We make use of three kinds of probabilities: P_H for generating heads, P_L for generating dependents on the left, and P_R for generating dependents on the right.

In summary, the probability of the rule

$$P(VP(dumped, VBD) \rightarrow VBD(dumped, VBD)\ NP(sacks, NNS)\ PP(into, P))$$


1. Generate the head VBD(dumped,VBD) with probability
P(H|LHS) = P(VBD(dumped,VBD) | VP(dumped,VBD))

2. Generate the left dependent (which is STOP, since there isn't one) with probability
P(STOP| VP(dumped,VBD) VBD(dumped,VBD))

3. Generate right dependent NP(sacks,NNS) with probability
P_r(NP(sacks,NNS| VP(dumped,VBD), VBD(dumped,VBD))

4. Generate the right dependent PP(into,P) with probability
P_r(PP(into,P) | VP(dumped,VBD), VBD(dumped,VBD))

5) Generate the right dependent STOP with probability
P_r(STOP | VP(dumped,VBD), VBD(dumped,VBD))

is estimated as

$$\begin{aligned} P_H(VBD|VP, dumped) \times P_L(STOP|VP, VBD, dumped) \\ \times P_R(NP(sacks, NNS)|VP, VBD, dumped) \\ \times P_R(PP(into, P)|VP, VBD, dumped) \\ \times P_R(STOP|VP, VBD, dumped) \end{aligned}$$

Each of these probabilities can be estimated from much smaller amounts of data, for example the MLE for the component probability $P_R(NP(sacks, NNS)|VP, VBD, dumped)$ is

$$\frac{Count(VP(dumped, VBD) \text{ with } NNS(sacks) \text{ as a daughter somewhere on the right})}{Count(VP(dumped, VBD)}$$

So, generally speaking, if H is a head with head word hw and head tag ht, lw/lt and rw/rt are the word/tag on the left and right respectively, and P is the parent, then the probability of an entire rule can be expressed as follows:

- 1. Generate the head of the phrase $H(hw, ht)$ with probability $P_H(H(hw, ht)|P, hw, ht)$
- 2. Generate modifiers to the left of the head with total probability

  $$\prod_{i=1}^{n+1} P_L(L_i(lw_i, lt_i)|P, H, hw, ht)$$

  such that $L_{n+1}(lw_{n+1}, lt_{n+1}) = STOP$, and we stop generating once we've generated a STOP token.

- 3. Generate modifiers to the right of the head with total probability

$$\prod_{i=1}^{n+1} P_P(R_i(rw_i, rt_i)|P, H, hw, ht)$$

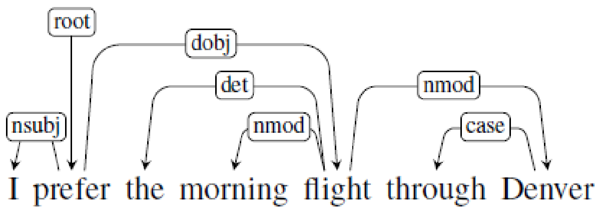
  such that $R_{n+1}(rw_{n+1}, rt_{n+1}) = STOP$, and we stop generating once we've generated a STOP token.

The actual Collins parser models are more complex than the one we described here. If you want to know more about this, check the section 12.6.2 of the book (3 ed.).

8 Dependency Parsing

The focus of the three previous chapters has been on context-free grammars and their use in automatically generating constituent-based representations. Here we present another family of grammar formalisms called **dependency grammars**.

In these formalisms, phrasal constituents and phrase-structure rules do not play a direct role. Instead, the syntactic structure of a sentence is described solely in terms of the words (or lemmas) in a sentence and an associated set of directed binary grammatical relations that hold among the words.



Relations among the words are illustrated above the sentence with directed, labeled arcs from heads to dependents. These relationships directly encode important information that is often buried in the more complex phrase-structure parses. For example, the arguments to the verb *prefer* are directly linked to it in the dependency structure, while their connection to the main verb is more distant in the phrase-structure tree.

A major advantage of dependency grammars is their ability to deal with languages that are morphologically rich and have a relatively **free word order**. Indeed, a dependency grammar approach abstracts away from word-order information, representing only the information that is necessary for the parse.

8.1 Dependency Treebanks

As with constituent-based methods, treebanks play a critical role in the development and evaluation of dependency parsers. Dependency treebanks have been created using similar approaches to those already discussed, i.e. having human annotators directly generate dependency structures for a given corpus, or using automatic parsers to provide an initial parse and then having annotators hand correct those parsers.

8.2 Transition-Based Dependency Parsing

Disclaimer: *this section was not covered during the course. I'm inserting it since I think it is interesting to see at least one approach to dependency parsing and especially because it is a good reference to the 'Formal Languages and Compilers' course.*

Our first approach to dependency parsing is motivated by a stack-based approach called **shift-reduce parsing** originally developed for analyzing programming languages (Aho and Ullman, 1972). This classic approach is simple and elegant, employing a context-free grammar, a stack, and a list of tokens to be parsed. Input tokens are successively shifted onto the stack and the top two elements of the stack are matched against the right-hand side of the rules in the grammar; when a match is found the matched elements are replaced on the stack (reduced) by the non-terminal from the left-hand side of the rule being matched. We will adapt this approach to our goals, altering the reduce operation so that instead of adding a non-terminal to a parse tree, it introduces a dependency relation between a word and its head.

The idea is simple: given what is called a **configuration**, the parsing process consists of a sequence of transitions through the space of possible configurations, in order to find a final configuration where all the words have been accounted for and an appropriate dependency tree has been synthesized.

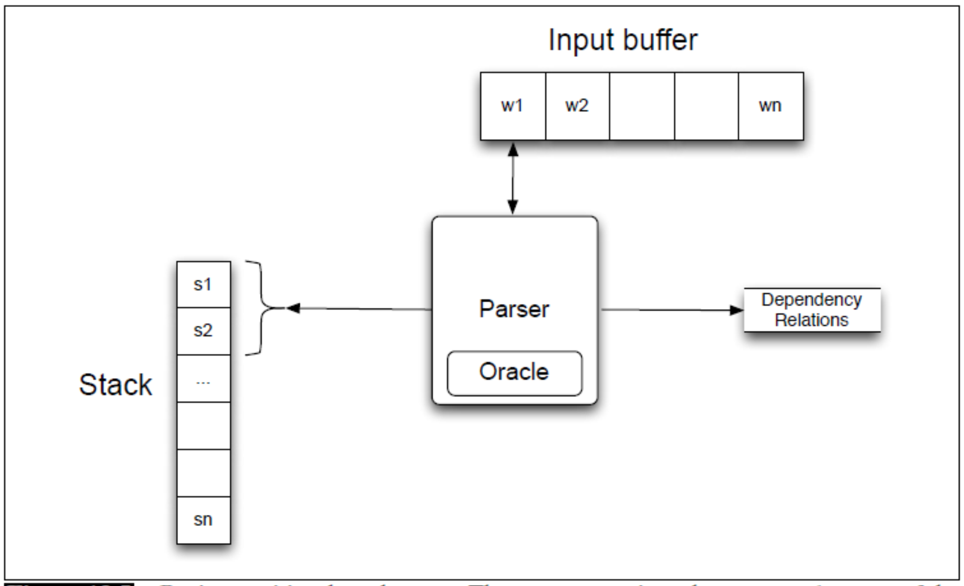
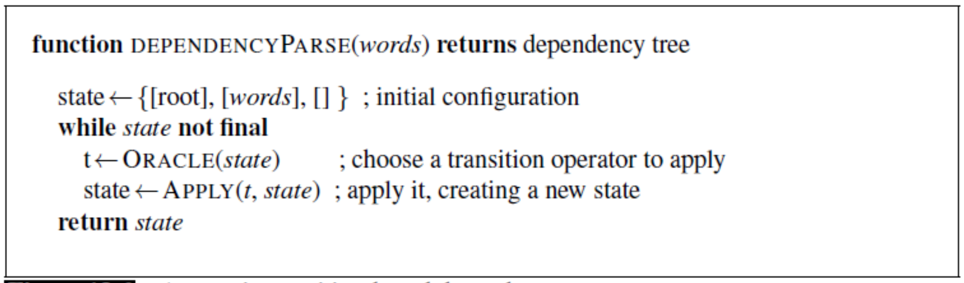


Figure 13.5 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.

In the standard approach to transition-based parsing, the operators used to produce new configurations are surprisingly simple and correspond to the intuitive actions one might take in creating a dependency tree by examining the words in a single pass over the input from left to right.

- LEFTARC: Assert a head-dependent relation between the word at the top of stack and the word directly beneath it; remove the lower word from the stack.
- RIGHTARC: Assert a head-dependent relation between the second word on the stack and the word at the top; remove the word at the top of the stack;

- SHIFT: Remove the word from the front of the input buffer and push it onto the stack.

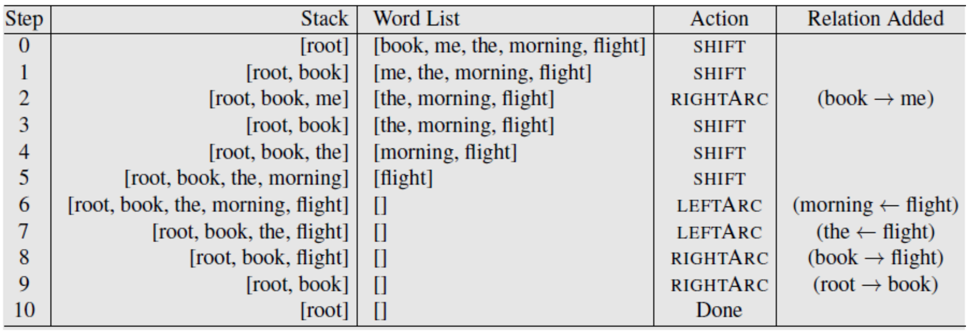


```
function DEPENDENCYPARSE(words) returns dependency tree

  state ← {[root], [words], [] }  ; initial configuration
  while state not final
    t ← ORACLE(state)        ; choose a transition operator to apply
    state ← APPLY(t, state)  ; apply it, creating a new state
  return state
```

Figure 13.6 A generic transition-based dependency parser

The efficiency of transition-based parsers should be apparent from the algorithm. The complexity is **linear in the length of the sentence** since it is based on a single left to right pass through the words in the sentence. More specifically, each word must first be shifted onto the stack and then later reduced.
Note that unlike the dynamic programming and search-based approaches previously discussed, this approach is a straightforward **greedy algorithm** — the oracle provides a single choice at each step and the parser proceeds with that choice, no other options are explored, no back-tracking is employed, and a single parse is returned in the end.



Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book $\rightarrow$ me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning $\leftarrow$ flight)
7	[root, book, the, flight]	[]	LEFTARC	(the $\leftarrow$ flight)
8	[root, book, flight]	[]	RIGHTARC	(book $\rightarrow$ flight)
9	[root, book]	[]	RIGHTARC	(root $\rightarrow$ book)
10	[root]	[]	Done	

8.2.1 Observations

1) The sequence given is not the only one that might lead to a reasonable parse. In general, there may be more than one path that leads to the same result, and due to ambiguity, there may be other transition sequences that lead to different equally valid parses.

2) We are assuming that the oracle always provides the correct operator at each point in the parse — an assumption that is unlikely to be true in practice. As a result, given the greedy nature of this algorithm, incorrect choices will lead to incorrect parses since the parser has no opportunity to go back and pursue alternative choices. There are several techniques that allow to explore the search space more fully.

3) For simplicity, we have illustrated this example without the labels on the dependency relations. To produce labeled trees, we can parameterize the LEFTARC and RIGHTARC operators with dependency labels. This, of course, makes the job of the oracle more difficult since it now has a much larger set of operators from which to choose.

8.2.2 Creating an Oracle

State-of-the-art transition-based systems use **supervised machine learning** methods to train classifiers that play the role of the oracle. Given appropriate training data, these methods learn a function that maps from configurations to transition operators.
Over the years, the dominant approaches to training transition-based dependency parsers have been **multinomial logistic regression** and **support vector machines**, both of which can make effective use of large numbers of sparse features of the kind described in the last section. More recently, **neural network**, or **deep learning**, approaches have been applied successfully to transition-based parsing.

8.3 Graph-Based Dependency Parsing

Graph-based approaches to dependency parsing search through the space of possible trees for a given sentence for a tree (or trees) that maximize some score. These methods encode the search space as directed graphs and employ methods drawn from graph theory to search the space for optimal solutions (e.g. **Maximum Spanning Tree**).

Disclaimer: *As the previous section, also this one was not covered during the course. If you want to go deeper, go to the chapter 13.5 of the book (3 ed.).*

9 The Representation of Sentence Meaning

The approach to semantics introduced here, and elaborated on in the next two chapters, is based on the idea that the meaning of linguistic expressions can be captured in formal structures called **meaning representations**.
We need representations that bridge the gap from linguistic inputs to the knowledge of the world needed to perform tasks. Consider the following ordinary language tasks that require some form of semantic processing of natural language:

- Deciding what to order at a restaurant by reading a menu
- Learning to use a new piece of software by reading the manual
- Answering essay questions on an exam
- Realizing that you've been insulted
- Following recipes

Grammatical representations aren't sufficient to accomplish these tasks.

9.1 Desiderata for Representations

Let's see some of the basic requirements that a meaning representation must fulfill and some of the complications that inevitably arise in the process of designing such meaning representations.

9.1.1 Verifiability

Consider the following question:

Does Maharani serve vegetarian food?

This example illustrates the most basic requirement for a meaning representation: it must be possible to use the representation to determine the relationship between the meaning of a sentence and the state of the world as we know it. In other words, we need to be able to determine the truth of our representations.
For now we can gloss this representation as *Serves(Maharani, VegetarianFood).* This representation of the input can be matched against our knowledge base of facts about a set of restaurants. If the system finds a representation matching this proposition in its knowledge base, it can return an affirmative answer. Otherwise, it must either say No if its knowledge of local restaurants is complete, or say that it doesn't know if there is reason to believe that its knowledge is incomplete.

9.1.2 Unambiguous Representations

Semantics, like all the other domains we have studied, is subject to ambiguity. Specifically, individual linguistic expressions can have different meaning representations assigned to them based on the circumstances in which they occur.

I wanna eat someplace that's close to Polimi.

Given the allowable argument structures for the verb eat, this sentence can either mean that the speaker wants to eat at some nearby location, or under a Godzilla-asspeaker interpretation, the speaker may want to devour some nearby location. The answer generated by the system for this request will depend on which interpretation is chosen as the correct one.

A concept closely related to ambiguity is **vagueness**. Like ambiguity, vagueness can make it difficult to determine what to do with a particular input on the basis of its meaning representation. Vagueness, however, does not give rise to multiple representations.

I want to eat Italian food.

A vague representation of the meaning of this phrase may be appropriate for some purposes, while a more specific representation may be needed for other purposes.

9.1.3 Canonical Form

The notion that single sentences can be assigned multiple meanings leads to the related phenomenon of distinct inputs that should be assigned the same meaning representation.

- Does Maharani have vegetarian dishes?
- Do they have vegetarian food at Maharani?
- Are vegetarian dishes served at Maharani?
- Does Maharani serve vegetarian fare?

9.1.4 Inference and Variables

Consider the sentence

Can vegetarians eat at Maharani?

Here, it would be a mistake to invoke canonical form to force our system to assign the same representation to this request as for the previous examples. That this request results in the same answer as the others arises, not because they mean the same thing, but because there is a common-sense connection between what vegetarians eat and what vegetarian restaurants serve. This is a fact about the world and not a fact about any particular kind of linguistic regularity. This implies that no approach based on canonical form and simple matching will give us an appropriate answer to this request. What is needed is a systematic way to connect the meaning representation of this request with the facts about the world as they are represented in a knowledge base.

We use the term **inference** to refer generically to a system's ability to draw valid conclusions based on the meaning representation of inputs and its store of background knowledge.

Now consider the following somewhat more complex request:

I'd like to find a restaurant where I can get vegetarian food.

Unlike our previous examples, this request does not make reference to any particular restaurant. The user is expressing a desire for information about an unknown and unnamed entity that is a restaurant that serves vegetarian food. Since this request does not mention any particular restaurant, the kind of simple matching-based approach we have been advocating is not going to work. Rather, answering this request requires a more complex kind of matching that involves the use of variables.

Serves(x, VegetarianFood)

Matching such a proposition succeeds only if the variable x can be replaced by some known object in the knowledge base in such a way that the entire proposition will then match.

9.1.5 Expressiveness

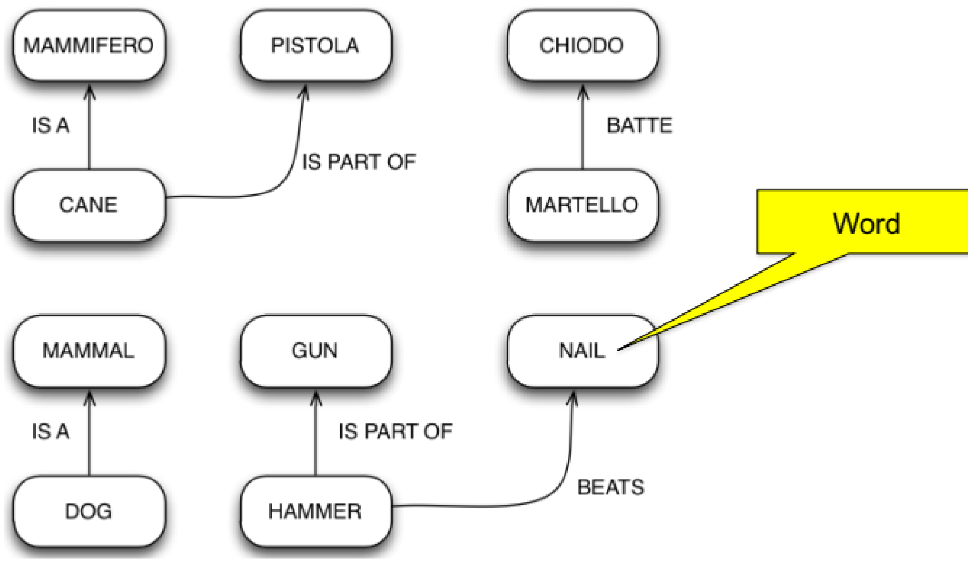
Finally, to be useful, a meaning representation scheme must be expressive enough to handle a wide range of subject matter. The ideal situation would be to have a single meaning representation language that could adequately represent the meaning of any sensible natural language utterance. Although this is probably too much to expect from any single representational system, **First-Order Logic** is expressive enough to handle quite a lot of what needs to be represented.

9.2 Semantic Networks

Semantic networks consist in a graphic notation introduced to model the organization of human semantic memory, or memory for word concepts.

- node $\rightarrow$ word concept
- arc $\rightarrow$ relationship between word concepts

In this approach, concepts coincide with words, thus making the knowledge modeling **language-dependent**, like one can notice from the following examples:



9.3 First-Order Logic

First-Order Logic (FOL) is a flexible, well-understood, and computationally tractable meaning representation language that satisfies many of the desiderata given before.

An additional attractive feature of FOL is that it makes very few specific commitments as to how things ought to be represented. And, the specific commitments it does make are ones that are fairly easy to live with and that are shared by many of the schemes mentioned earlier; the represented world consists of objects, properties of objects, and relations among objects.

Disclaimer: *the book gives a complete introduction to FOL, which I will not cover here. The only thing I will discuss is the lambda notation.*

9.3.1 Lambda notation

This notation provides a way to abstract from fully specified FOL formula in a way that will be particularly useful for semantic analysis.

$$\lambda x.P(x)$$

The usefulness of these l-expressions is based on the ability to apply them to logical terms to yield new FOL expressions where the formal parameter variables are bound to the specified terms. This process is known as λ-reduction and consists of a simple textual replacement of the λ variables with the specified FOL terms, accompanied by the subsequent removal of the λ.

$$\lambda x.\lambda y.Near(x, y)(Bacaro)$$
$$\lambda y.Near(Bacaro, y)$$

9.3.2 Inference - Modus Ponens

This section briefly discusses modus ponens, the most widely implemented inference method provided by FOL. Modus ponens is a familiar form of inference that corresponds to what is informally known as if-then reasoning.

$$\begin{array}{l} \alpha \\ \alpha \implies \beta \\ - - - - \\ \beta \end{array}$$

A schema like this indicates that the formula below the line can be inferred from the formulas above the line by some form of inference. Modus ponens simply states that if the left-hand side of an implication rule is true, then the right-hand side of the rule can be inferred.

Modus ponens can be put to practical use in one of two ways: **forward chaining** and **backward chaining**.

Forward chaining
In forward chaining systems, modus ponens is used in precisely the manner just described. In this kind of arrangement, as soon as a new fact is added to the knowledge base, all applicable implication rules are found and applied, each resulting in the addition of new facts to the knowledge base. These new propositions in turn can be used to fire implication rules applicable to them. The process continues until no further facts can be deduced. The forward chaining approach has the advantage that facts will be present in the knowledge base when needed, because, in a sense all inference is performed in advance. This can substantially reduce the time needed to answer subsequent queries since they should all amount to simple lookups. The disadvantage of this approach is that facts that will never be needed may be inferred and stored.

Backward chaining
In backward chaining, modus ponens is run in reverse to prove specific propsitions called queries. The first step is to see if the query formula is true by determining if it is present in the knowledge base. If it is not, then the next step is to search for applicable implication rules present in the knowledge base. An applicable rule is one whereby the consequent of the rule matches the query formula. If there are any such rules, then the query can be proved if the antecedent of any one them can be shown to be true. Not surprisingly, this can be performed recursively by backward chaining on the antecedent as a new query. The Prolog programming language is a backward chaining system that implements this strategy.

While forward and backward reasoning are sound, neither is **complete**. This means that there are valid inferences that cannot be found by systems using these methods alone. Fortunately, there is an alternative inference technique called **resolution** that is sound and complete. Unfortunately, inference systems based on resolution are far more computationally expensive than forward or backward chaining systems.

9.4 Description Logics

Description logics are an effort to better specify the semantics of these earlier structured network representations and to provide a conceptual framework that is especially well suited to certain kinds of domain modeling. Formally, the term Description Logics refers to a family of logical approaches that correspond to varying subsets of FOL.
When using Description Logics to model an application domain, the emphasis is on the representation of knowledge about categories, individuals that belong to those categories, and the relationships that can hold among these individuals.

Once we've specified the categories of interest in a particular domain, the next step is to arrange them into a hierarchical structure. One way to do so is by asserting **subsumption** relations between the appropriate concepts in a terminology. The subsumption relation is conventionally written as $C \sqsubseteq D$ and is read as C is subsumed by D; that is, all members of the category C are also members of the category D.

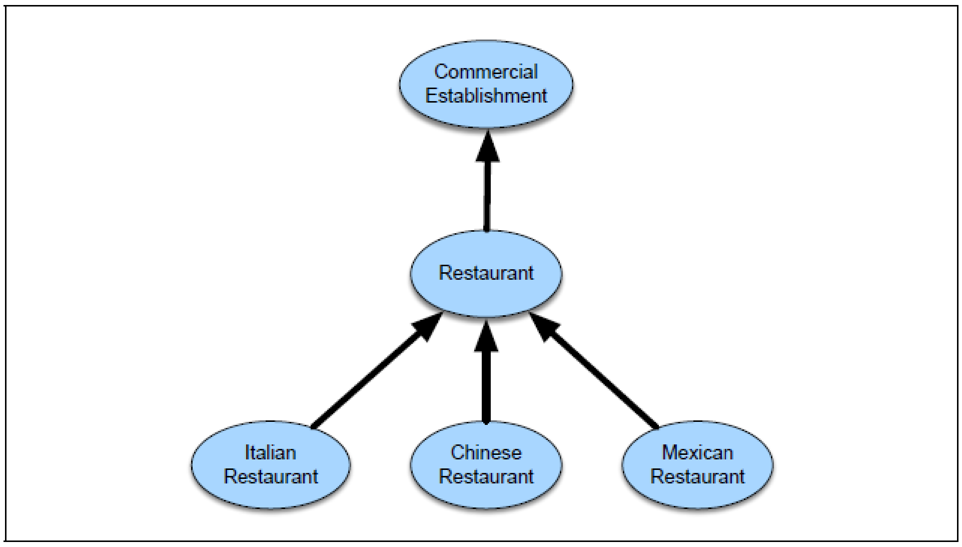


Figure 14.6 A graphical network representation of a set of subsumption relations in the restaurant domain.

Having a hierarchy such as the one above tells us next to nothing about the concepts in it. We certainly don't know anything about what makes a restaurant a restaurant, much less Italian, Chinese, or expensive. What is needed are additional assertions about what it means to be a member of any of these categories. In Description Logics such statements come in the form of relations between the concepts being described and other concepts in the domain.

$$ItalianRestaurant \sqsubseteq Restaurant \sqcap \exists hasCuisine.ItalianCuisine$$

An equivalent statement in FOL would be

$$\forall x ItalianRestaurant(x) \implies Restaurant(x) \wedge (\exists y Serves(x, y) \wedge ItalianCuisine(y))$$

9.5 Frame-based Representation

In frame-based systems objects are represented as feature-structures:

- Features (slots)
- Values (fillers)
- Values can, in turn, be a frame

For example, the sentence *I believe Mary ate British food.* can be represented as:

$$\begin{bmatrix} \textsc{Believing} & \\ \textsc{Believer} & \textsc{Speaker} \\ \textsc{Believed} & \begin{bmatrix} \textsc{Eating} & \\ \textsc{eater} & \textsc{Mary} \\ \textsc{eaten} & \textsc{BritishFood} \end{bmatrix} \end{bmatrix}$$

9.6 Ontologies

Ontologies and Natural Language Processing (NLP) can often be seen as two sides of the same coin.

An Ontology Model is:

- the classification of entities
- modeling the relationships between those entities.

The purpose of NLP is:

- the identification of entities
- understanding the relationship between those entities.

Ontologies **do not depend** on the language:

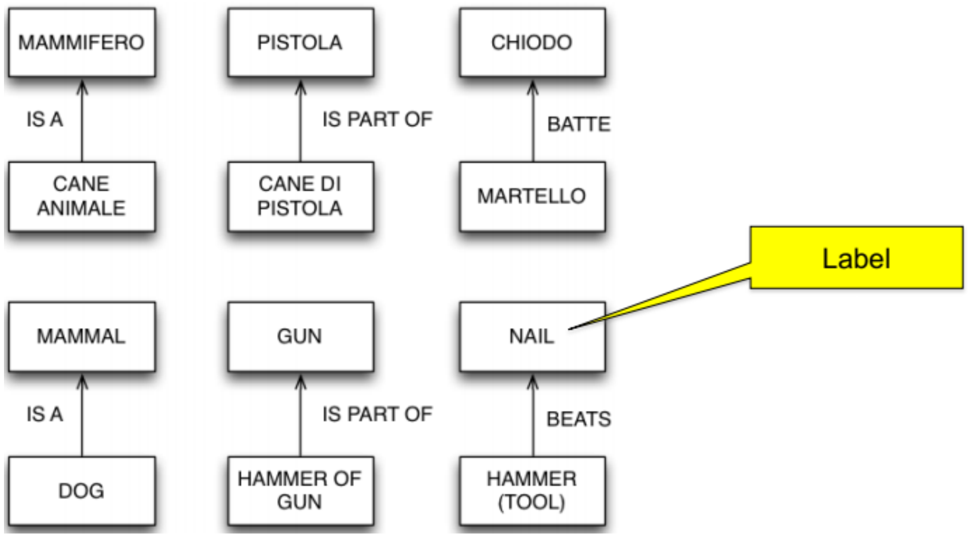
Most ontologies are composed of:

- Classes (e.g. Wine, Winery)
- Individuals (e.g. champagne)
- Attributes (e.g. price)
- Relationships (e.g. Winery *produces* Wine)

and they can be used to represent both KBs and sentences.

9.6.1 The OWL Language

- OWL Lite: taxonomies and simple constraints
- OWL DL: permits to represent a Description Logic (decidable subset of FOL)



- OWL Full: higher order logics (not decidable)

More details can be found at `https://www.w3.org/OWL/`.

9.7 Open and Closed World

- The closed world assumption (e.g. SQL): any statement that is not known to be true is false. The system is assumed to have complete knowledge.
- The open world assumption (e.g. FOL, OWL): any statement that is not known to be true is...unknown. The system does not have enough information to decide. This limits the kinds of inferences an agent can make. On the other hand, however, it represents the notion that no single agent has complete knowledge.

10 Semantic Analysis

This section presents a number of computational approaches to the problem of **semantic analysis**, the process whereby meaning representations of the kind discussed in the previous section are composed and assigned to linguistic inputs.

10.1 Syntax-driven Semantic Analysis

This is the first approach that we'll see. It assigns meaning representations to inputs based solely on static knowledge from the lexicon and the grammar. In this approach, when we refer to an input's meaning, or meaning representation, we have in mind an impoverished representation that is both context-independent and inference-free. Nevertheless, we are interested in this method since there are some limited application domains where such representations are sufficient to produce useful results. Moreover, these impoverished representations can serve as inputs to subsequent processes that can produce richer, more useful, meaning representations.

Syntax-driven semantic analysis is based on the **principle of compositionality**. The key idea is that the meaning of a sentence can be composed from the meanings of its parts. Of course,

when interpreted superficially, this principle is somewhat less than useful. Indeed, the meaning of a sentence is not based solely on the words that make it up, it is based on the ordering, grouping, and relations among the words in the sentence.
Note that this is simply another way of saying that the meaning of a sentence is partially based on its syntactic structure.

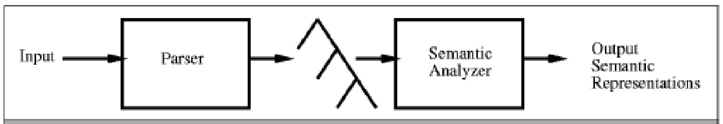


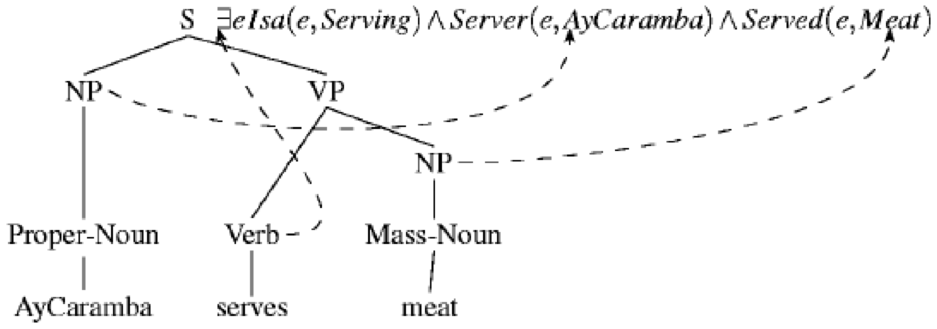
Figure 15.1 A simple pipeline approach to semantic analysis.

As we can see from the figure above, the syntactic analysis of an input sentence will form the input to a semantic analyzer. Note that although the diagram shows a parse tree as input, other syntactic representations such as feature structures, or lexical dependency diagrams, can be used.

Assumption:
In the syntax driven approach presented here, ambiguities arising from the syntax and the lexicon will lead to the creation of multiple ambiguous meaning representations. It is not the job of the semantic analyzer to resolve these ambiguities. Instead, it is the job of subsequent interpretation processes with access to domain specific knowledge, and knowledge of context to select among competing representations.

Let's consider an example:

AyCaramba serves meat.


S $\exists e Isa(e, Serving) \wedge Server(e, AyCaramba) \wedge Served(e, Meat)$
NP
VP
NP
Proper-Noun
Verb
Mass-Noun
AyCaramba
serves
meat

The above figure represents a simplified parse tree. As suggested by the dashed arrows, a semantic analyzer, given this tree as input, might proceed by first retrieving a meaning representation from the subtree corresponding to the verb *serves*. The analyzer might next retrieve meaning representations corresponding to the two noun phrases in the sentence. Then, using the representation acquired from the verb as a template, the noun phrase meaning representations can be used to bind the appropriate variables in the verb representation, thus producing the meaning representation for the sentence as a whole.

What's the problem here? Well, the function used to interpret the tree must know, among other things, that it is the verb that carries the template upon which the final representation is based, where this verb occurs in the tree, where its corresponding arguments are, and which argument fills which role in the verb's meaning representation. In other words, it requires a good deal of specific knowledge about **this particular example and its parse tree**.

10.1.1 Semantic Augmentation to CFG Rules

We will begin by augmenting context-free-grammar rules with **semantic attachments**. These attachments can be thought of as instructions that specify how to compute the meaning representation of a construction from the meanings of its constituents parts.

$$A \rightarrow \alpha_1 ... \alpha_n \quad \{f(\alpha_j.sem, ..., \alpha_k.sem)\}$$

This notation states that the meaning representation assigned to the construction A, which we will denote as $A.sem$, can be computed by running the function f on some subset of the semantic attachments of A's constituents. Let's see this in practice:

$$ProperNoun \rightarrow AyCaramba \quad \{AyCaramba\}$$
$$MassNoun \rightarrow meat \quad \{Meat\}$$

Note, however, that the subtrees corresponding to these rules do not directly contribute to the final meaning representation. Rather, it is the NPs higher in the tree that contribute them to the final representation. We can deal with this indirect contribution by stipulating that the upper NPs obtain their meaning representations from the meanings of their children:

$$NP \rightarrow ProperNoun \quad \{ProperNoun.sem\}$$
$$NP \rightarrow MassNoun \quad \{MassNoun.sem\}$$

These rules state that the meaning representation of the noun phrases are the same as the meaning representations of their individual components.

What about *Verb* $\rightarrow$ *serves*?

$$Verb \rightarrow serves \quad \{\exists e, x, y\ Isa(e, Serving) \land Server(e, x) \land Served(e, y)\}$$

Moving up to the parse tree, the next constituent to be considered is the VP that dominates both *serves* and *meat*. Unlike the NPs, we can not simply copy the meaning of these children up to the parent VP. Rather, we need to **incorporate** the meaning of the NP into the meaning of the *Verb* and assign the resulting representation to the *VP.sem*. Fortunately, there is the **lambda notation** we discussed, that provides exactly the kind of formal parameter functionality that we need.

$$Verb \rightarrow serves \quad \{\lambda x \lambda y \exists e\ Isa(e, Serving) \land Server(e, y) \land Served(e, x)\}$$
$$VP \rightarrow Verb\ NP \quad \{Verb.sem(NP.sem)\}$$

10.1.2 Idioms

There are many cases where the meaning of a constituent is not based on the meaning of its parts, at least not in the straightforward compositional sense.

Coupons are just the tip of the iceberg.

The phrase *the tip of the iceberg* does not have much to do with tips or icebergs, instead it means something like *the beginning.* The most straightforward way to handle idiomatic constructions like this is to introduce new grammar rules specifically designed to handle them:

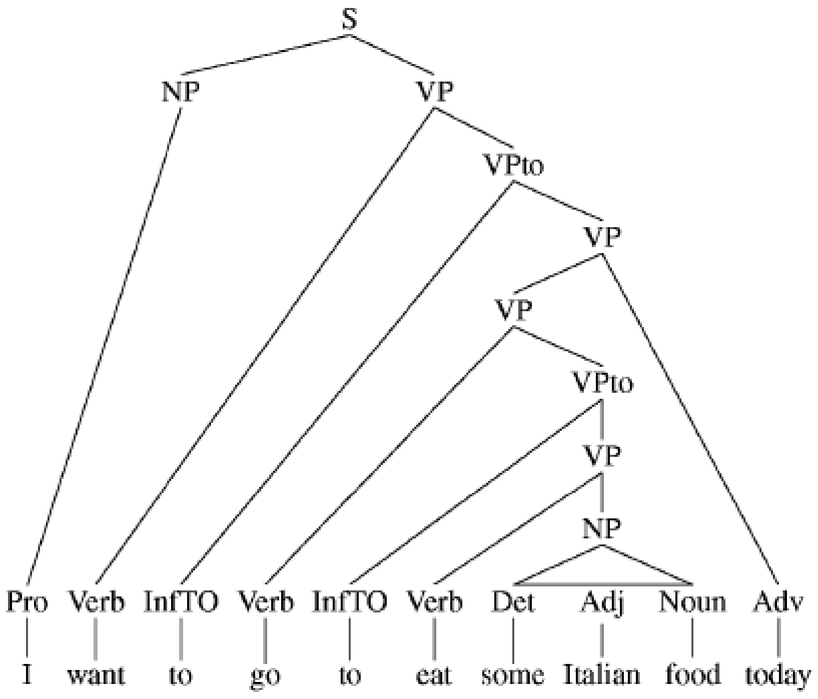
NP $\rightarrow$ *the tip of the iceberg* $\{$*Beginning*$\}$

Of course, this solution is not very efficient as it is not general enough to handle a lot of cases. Idioms are far more frequent and far more productive than is generally recognized and pose serious difficulties for many applications.

10.2 Semantic Grammars

Syntactic grammars are not well-suited for the task of semantic analysis. This mismatch typically manifests itself in the following three ways:

- Key semantic elements are often widely distributed across parse trees, thus complicating the composition of the required meaning representation.
- Parse trees often contain many syntactically motivated constituents that play essentially no role in semantic processing.
- The general nature of many syntactic constituents results in semantic attachments that create nearly vacuous meaning representations.


S
NP
VP
VPto
VP
VP
VPto
VP
NP
Pro Verb InfTO Verb InfTO Verb Det Adj Noun Adv
I want to go to eat some Italian food today

The branching structure of this tree distributes the key components of the meaning representations widely throughout the tree. At the same time, most of the nodes in the tree contribute almost nothing to the meaning of this sentence. This structure requires three lambdas-expressions and a complex term to bring the few contentful elements together at the top of the tree.

Semantic grammars can overcome these problems. In this approach, the rules and constituents of the grammar are designed to correspond directly to entities and relations from the domain being discussed. More specifically, such grammars are constructed so that key semantic components can occur together within single rules, and rules are made no more general than is needed to achieve sensible semantic analyses.

InfoRequest → *User* want **to go to** eat *FoodType TimeExpr FoodType* → *Nationality FoodType*

One of the key motivations for the use of semantic grammars in these domains was the need to deal with various kinds of anaphor and ellipsis. Semantic grammars can help with these phenomena since by their nature they enable certain amount of prediction. For example:

When does flight 573 arrive in Atlanta? When does it arrive in Dallas?

Sentences like these can be analyzed with a rule like the following, which makes use of the domain specific non-terminals *Flight* and *City*:

InfoRequest → *when does* ***Flight*** *arrive in* ***City***

A rule such as this gives far more information about the likely referent of the *it* than a purely syntactic rule that would simply restrict it to anything expressible as a noun phrase.

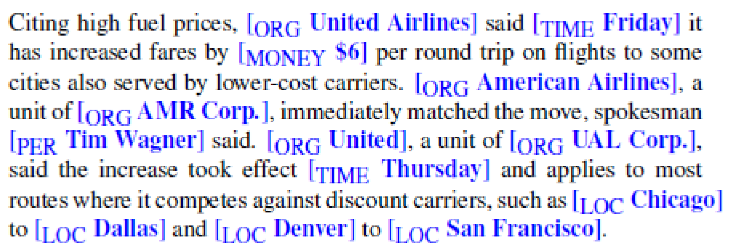
Of course, as one can imagine, the main drawback of semantic grammars is the almost complete **lack of reuse**, due to the fact that as we said it is domain-specific.

10.3 Information Extraction

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

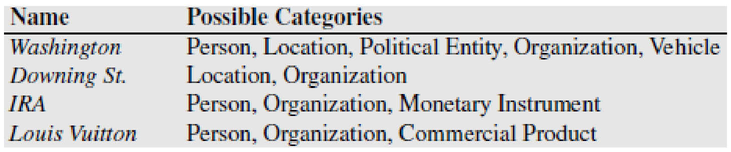
10.3.1 Named Entity Recognition (NER)

The first step in information extraction is to detect the entities in the text. A named entity is, roughly speaking, anything that can be referred to with a proper name: a person, a location, an organization. The term is commonly extended to include things that aren't entities per se, including dates, times, and other kinds of temporal expressions, and even numerical expressions like prices.



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

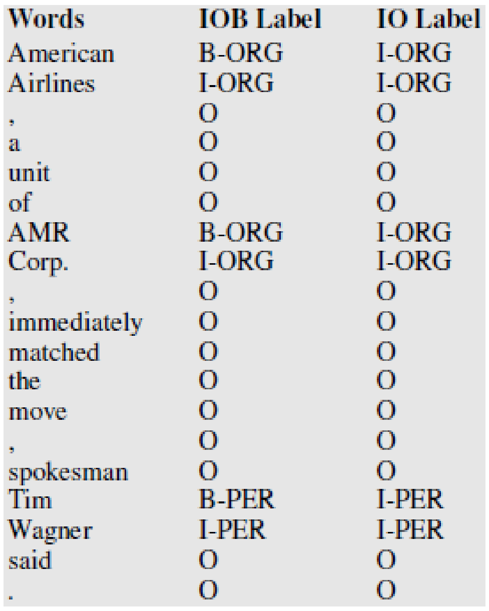
Recognition is difficult partly because of the ambiguity of segmentation; we need to decide what's an entity and what isn't, and where the boundaries are. Another difficulty is caused by type ambiguity.



Name	Possible Categories
Washington	Person, Location, Political Entity, Organization, Vehicle
Downing St.	Location, Organization
IRA	Person, Organization, Monetary Instrument
Louis Vuitton	Person, Organization, Commercial Product

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

In **IOB tagging** we introduce a tag for the beginning (B) and inside (I) of each entity type, and one for tokens outside (O) any entity. The number of tags is thus $2n + 1$ tags, where n is the number of entity types. IOB tagging can represent exactly the same information as the bracketed notation.



Words	IOB Label	IO Label
American	B-ORG	I-ORG
Airlines	I-ORG	I-ORG
,	O	O
a	O	O
unit	O	O
of	O	O
AMR	B-ORG	I-ORG
Corp.	I-ORG	I-ORG
,	O	O
immediately	O	O
matched	O	O
the	O	O
move	O	O
,	O	O
spokesman	O	O
Tim	B-PER	I-PER
Wagner	I-PER	I-PER
said	O	O
.	O	O

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

10.3.2 Relation Extraction

Next on our list of tasks is to discern the relationships that exist among the detected entities. Considering the same example as before, the text tells us, for example, that *Tim Wagner* is a spokesman for *American Airlines*, that *United* is a unit of *UAL Corp.*, and that *American* is a unit of *AMR*. These binary relations are instances of more generic relations such as **part-of** or **employs**.

Supervised Machine Learning

One common approach to solve this task is **supervised machine learning**, whose scheme should be familiar by now. A fixed set of relations and entities is chosen, a training corpus is hand-annotated with the relations and entities, and the annotated texts are then used to train classifiers to annotate an unseen test set.

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

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

Semisupervised Relation Extraction via Bootstrapping

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

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

- Budget airline Ryanair, which uses Charleroi as a hub, scrapped all weekend flights out of the airport.
- All flights in and out of Ryanair's Belgian hub at Charleroi airport were grounded on Friday...
- A spokesman at Charleroi, a main hub for Ryanair, estimated that 8000 passengers had already been affected.

From these results, we can use the context of words between the entity mentions, the words before mention one, the word after mention two, and the named entity types of the two mentions, and perhaps other features, to extract general patterns, which can then be used to search for additional tuples.

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

Distant Supervision for Relation Extraction

The distant distant supervision method of **Mintz et al. (2009)** combines the advantages of bootstrapping supervision with supervised learning. Instead of just a handful of seeds, distant supervision uses a large database to acquire a huge number of seed examples, creates lots of noisy pattern features from all these examples and then combines them in a supervised classifier. For example suppose we are trying to learn the *place-of-birth* relationship between people and their birth cities. In the seed-based approach, we might have only 5 examples to start with. But Wikipedia-based databases like DBPedia or Freebase have tens of thousands of examples of many relations. The next step is to run named entity taggers on large amounts of text (Mintz et al. (2009) used 800,000 articles from Wikipedia) and extract all sentences that have two named entities that match the tuple, like the following:

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

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

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

and so on. We can then apply feature-based or neural classification.

Unsupervised Relation Extraction

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

For example, the **ReVerb system (Fader et al., 2011)** extracts a relation from a sentence s in 4 steps:

- Run a part-of-speech tagger and entity chunker over s.
- For each verb in s, find the longest sequence of words w that start with a verb and satisfy syntactic and lexical constraints, merging adjacent matches.
- For each phrase w, find the nearest noun phrase x to the left which is not a relative pronoun, wh-word or existential "there". Find the nearest noun phrase y to the right.
- Assign confidence c to the relation $r = (x, w, y)$ using a confidence classifier and return it.

More details can be found in the book.

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

10.3.3 Temporal Expression Extraction

Times and dates are a particularly important kind of named entity that play a role in question answering, in calendar and personal assistant applications. In order to reason about times and dates, after we extract these **temporal expressions** they must be **normalized**, i.e. converted to a standard format so we can reason about them.

The temporal expression recognition task consists of finding the start and end of all of the text spans that correspond to such temporal expressions.

Rule-based approaches to temporal expression recognition use cascades of automata to recognize patterns at increasing levels of complexity. Tokens are first part-of-speech tagged, and then larger and larger chunks are recognized from the results from previous stages, based on patterns containing trigger words (e.g., February) or classes (e.g., MONTH).

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

10.3.4 Extracting Events

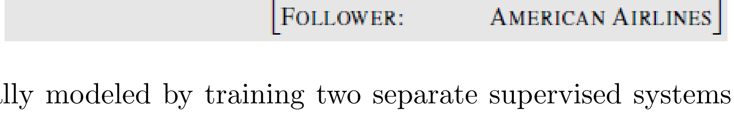
The task of **event extraction** is to identify mentions of events in texts. For the purposes of this task, an event mention is any expression denoting an event or state that can be assigned to a particular point, or interval, in time.

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

10.3.5 Template Filling

Many texts contain reports of events, and possibly sequences of events, that often correspond to fairly common, stereotypical situations in the world. These abstract situations can be characterized as **scripts**. In their simplest form, such scripts can be represented as templates consisting of fixed sets of slots that take as values slot-fillers belonging to particular classes. The task of **template filling** is to find documents that invoke particular scripts and then fill the slots in the associated templates with fillers extracted from the text. These slot-fillers may consist of text segments extracted directly from the text, or they may consist of concepts that have been inferred from text elements through some additional processing.

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



FARE-RAISE ATTEMPT:	LEAD AIRLINE:	UNITED AIRLINES
	AMOUNT:	$6
	EFFECTIVE DATE:	2006-10-26
	FOLLOWER:	AMERICAN AIRLINES

The task is generally modeled by training two separate supervised systems. The first system decides whether the template is present in a particular sentence. This template task is called **template recognition**. The second system has the job of **role-filler extraction**. A separate classifier is trained to detect each role (LEAD-AIRLINE, AMOUNT, and so on). This can be a binary classifier that is run on every noun-phrase in the parsed input sentence, or a sequence model run over sequences of words.

10.4 Lexical Semantics

In the previous sections we made minimal use of the notion of the *meaning of a word*, due to a general approach motivated by the view that while words may contribute content to the meanings of sentences, they do not themselves have meanings, i.e. they do not refer to the world, can not be judged to be true or false etc. Now, instead, we will see that the lexicon has a highly systematic structure that governs what words can mean and how they can be used. The study of this systematic, meaning related, structure is called **Lexical Semantics**.

We will focus on the notion of **lexeme**, the smallest unit with orthographic form, phonological form and meaning.

Relations among lexemes:

- Homonymy: different lexemes with the same form but with unrelated meanings;
  - Homographs: lexemes with the same orthographic form
  - Homophones: lexemes with the same phonological form
  - Perfect homonym: homograph + homophone
- Polysemy: a single lexeme with multiple related meanings;
  - Metaphor: constructs an analogy between two things or ideas, the analogy is conveyed by the use of a metaphorical word in place of some other word
  - Metonymy: a concept is denoted by naming some other concept closely related to it
- Synonymy: different lexemes with the same meaning; two lexemes are considered synonyms if they can be substituted for one another in sentences without changing the meaning of the sentence (**substitutability**). Perfect synonyms are rare.
- Antonymy: different lexemes with opposite (but related) sense;
- Hypernymy: an hypernym lexeme denotes a superclass of another lexeme;
- Hyponymy: an hyponym lexeme denotes a subclass of another lexeme;
- Meronymy: a meronym lexeme denotes a constituent part of, or a member of another lexeme;
- Holonymy: an holonym lexeme denotes the whole of a lexeme that denotes a part of it.

Note that these relations are important, since they affect most of the NLP tasks. For example, Text-To-Speech is affected by homographs with different phonological form; Information Retrieval is affected by homographs; Spelling Correction is affected by homophones; Speech Recognition is affected by homophones and perfect homonyms.

10.4.1 Lexical Databases

The widespread use of lexical relations in linguistic, psycho-linguistic and computational research has led to a number of efforts to create large electronic databases of such relations. These efforts have, in general, followed one of two basic approaches: mining information from existing dictionaries and thesauri, and handcrafting a database from scratch. Despite the obvious advantage of reusing existing resources, **WordNet**, one of the most well-developed lexical database for English, was developed using the latter approach.

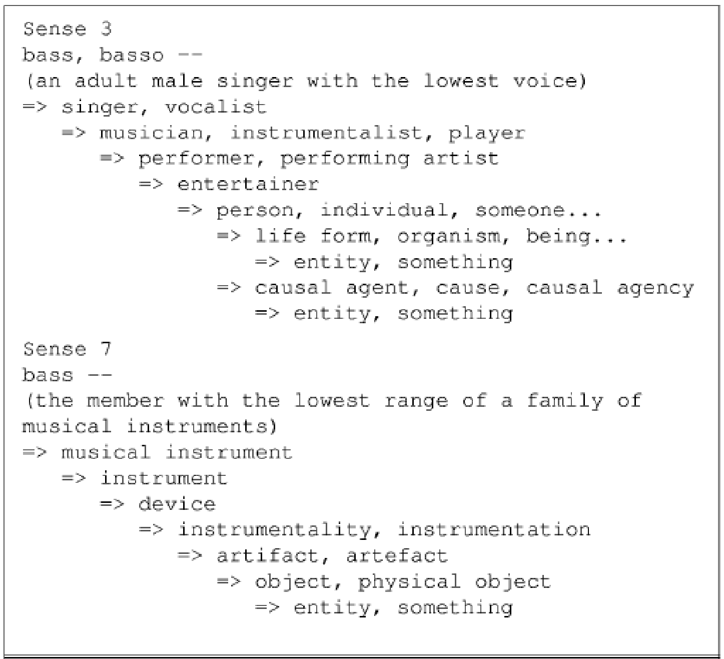
Link to the WordNet website: *`https://wordnet.princeton.edu/`*

WordNet consists of three separate databases, one each for nouns and verbs, and a third for adjectives and adverbs. Of course, a simple listing of lexical entries would not be much more useful that an ordinary dictionary. The power of WordNet lies in its set of domain-independent lexical relations. These relations can hold among WordNet entries, senses, or sets of synonyms.

Synonyms are indeed one of the most well-developed features of WordNet. They are organized in **synsets**:

- A synset contains synonym lexemes
- A synset carries a specific sense, a meaning
- A synset has a gloss, explaining the carried meaning
- A lexeme can appear in several synset

Moreover, each synset is related to its immediately more general and more specific synsets via direct hypernym and hyponym relations.



```
Sense 3
bass, basso --
(an adult male singer with the lowest voice)
=> singer, vocalist
   => musician, instrumentalist, player
      => performer, performing artist
         => entertainer
            => person, individual, someone...
               => life form, organism, being...
                  => entity, something
               => causal agent, cause, causal agency
                  => entity, something

Sense 7
bass --
(the member with the lowest range of a family of
musical instruments)
=> musical instrument
   => instrument
      => device
         => instrumentality, instrumentation
            => artifact, artefact
               => object, physical object
                  => entity, something
```

Figure 16.8 Hyponymy chains for two separate senses of the lexeme *bass*. Note that the chains are completely distinct, only converging at *entity*.

10.4.2 The Internal Structure of Words

Thematic Roles

Thematic roles are a set of categories which provide a shallow semantic language for characterizing certain arguments of verbs.

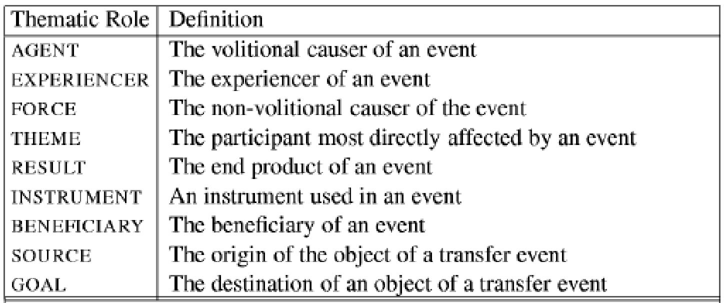
Houston's Billy Hatcher broke a bat.
He opened a drawer.

In the predicate calculus event representation seen previously, part of the representation of these two sentences would be the following:

$$\exists e,x,y\ Isa(e,Breaking) \wedge Breaker(e,BillyHatcher) \wedge BrokenThing(e,y) \wedge Isa(y,BaseballBat)$$

$$\exists e,x,y\ Isa(e,Opening) \wedge Opener(e,he) \wedge OpenedThing(e,y) \wedge Isa(y,Door)$$

In this representation, the roles of the subjects of the verbs *break* and *open* are *Breaker* and *Opener* respectively. These deep roles are specific to each possible kind of event. Breaking events have Breakers, Opening events have Openers etc. But *Breakers* and *Openers* have something in common. They are both volitional actors, often animate, and they have direct causal responsibility for their events. A **thematic role** is a way of expressing this commonality. We say that the subjects of both these verbs are AGENTS. Similar, the direct objects of both these verbs, the *BrokenThing* and *OpenedThing*, are THEME.



Thematic Role	Definition
AGENT	The volitional causer of an event
EXPERIENCER	The experiencer of an event
FORCE	The non-volitional causer of the event
THEME	The participant most directly affected by an event
RESULT	The end product of an event
INSTRUMENT	An instrument used in an event
BENEFICIARY	The beneficiary of an event
SOURCE	The origin of the object of a transfer event
GOAL	The destination of an object of a transfer event

Figure 16.9 Some commonly-used thematic roles with their definitions.

Linking Theory

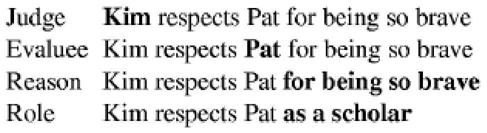
One common use of thematic roles in computational systems is as a shallow semantic language. For example, they are sometimes used in machine translation systems as part of a useful intermediate language. Another use of thematic roles was as an intermediary between semantic roles in conceptual structure or common-sense knowledge and their more language-specific surface grammatical realization as subject or object:

$$AGENT \rightarrow INSTRUMENT \rightarrow THEME$$

Thus if the thematic description of a verb includes an AGENT, an INSTRUMENT and a THEME, it is the AGENT which will be realized as the subject.

FrameNet

A FrameNet entry for a word lists every set of arguments it can take, including the possible sets of thematic roles, syntactic phrases and their grammatical function. The thematic roles used in FrameNet are much more specific than the ones we've seen till now. Each FrameNet thematic role is defined as part of a **frame**, and each frame as part of a **domain**. For example, one domain is the **Cognition** domain. All the cognition frames define the thematic role COGNIZER. In the **judgement** frame, the COGNIZER is referred to as the JUDGE; the frame also includes an EVALUEE, a REASON, and a ROLE.



Judge	**Kim** respects Pat for being so brave
Evaluee	Kim respects **Pat** for being so brave
Reason	Kim respects Pat **for being so brave**
Role	Kim respects Pat **as a scholar**

The problem with a scheme like FrameNet is the extensive human effort it requires in defining thematic roles for each domain and each frame.

Selectional Restrictions

The notion of a **selectional restriction** can be used to augment thematic roles by allowing lexemes to place certain semantic restrictions on the lexemes and phrases that can accompany them in a sentence.

I wanna eat someplace that's close to Politecnico.

There are two possible parses for this sentence corresponding to the intransitive and transitive versions of the verb *eat.* These two parses lead, in turn, to two distinct semantic analyses. The sentence is semantically ill-formed, since the THEME in the sentence, corresponding to *someplace* cannot easily be interpreted as edible. Thus we have a **selection restriction violation**, i.e. a situation where the semantics of the filler of a thematic role is not consistent with a constraint imposed on the role by the predicate.

In order to represent selection restrictions we can extend the event-oriented meaning representations often used.

$$\exists e, x, y \; Eating(e) \wedge Agent(e, x) \wedge Patient(e, y)$$

becomes

$$\exists e, x, y \; Eating(e) \wedge Eater(e, x) \wedge Patient(e, y) \wedge Isa(y, EdibleThing)$$

While this approach adequately captures the semantics of selectional restrictions, there are two practical problem with its direct use. First, using the full power of FOL to perform the simple task of enforcing selection restrictions is overkill. There are far simpler formalism that can do the job with far less computational cost. The second problem is that it presupposes a large logical knowledge-base of facts about the concepts that make up selection restrictions.

A far more practical approach is to exploit the hyponymy relations present in the WordNet database. A given meaning representation can be judged to be well-formed if the lexeme that fills a thematic role has as one of its hypernyms the synset specified by the predicate for that thematic role.

10.4.3 Word Sense Disambiguation (WSD)

The task of selecting the correct sense for a word is called **word sense disambiguation**, or WSD. WSD algorithms take as input a word in context and a fixed inventory of potential word senses and outputs the correct word sense in context. The input and the senses depends on the task. For machine translation from English to Spanish, the sense tag inventory for an English word might be the set of different Spanish translations. For automatic indexing of medical articles, the sense-tag inventory might be the set of MeSH (Medical Subject Headings) thesaurus entries.

Supervised Word Sense Disambiguation is commonly used whenever we have sufficient data that has been hand-labeled with correct word senses.
Supervised WSD algorithms can use any standard classification algorithm. Features generally include the word identity, part-of-speech tags, and embeddings of surrounding words, usually computed in two ways: **collocation** features are words or n-grams at a particular location, (i.e., exactly one word to the right, or the two words starting 3 words to the left, and so on). **bag of word** features are represented as a vector with the dimensionality of the vocabulary (minus stop words), with a 1 if that word occurs in the neighborhood of the target word.

An electric guitar and ***bass*** *player stand off to one side.*

If we use a small 2-word window, a standard feature vector might include a bag of words, parts-of-speech, unigram and bigram collocation features, and embeddings, that is:

$$[w_{i-2}, POS_{i-2}, w_{i-1}, POS_{i-1}, w_{i+1}, POS_{i+1}, w_{i+2}, POS_{i+2},\\ w_{i-2}^{i-1}, w_{i+1}^{i+2}, E(w_{i-2}, w_{i-1}, w_{i+1}, w_{i+2}), bag()]$$

would yield the following vector:

$$[guitar, NN, and, CC, player, NN, stand, VB, andguitar, playerstand,\\ E(guitar, and, player, stand), bag(guitar, player, stand)]$$

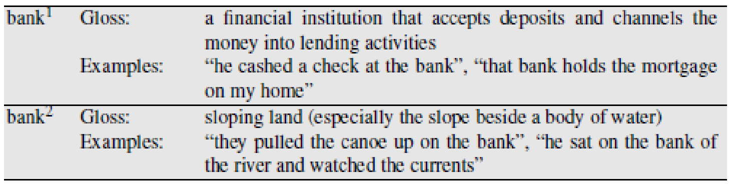
Supervised algorithms based on sense-labeled corpora are the best-performing algorithms for sense disambiguation. However, such labeled training data is expensive and limited. One alternative is to get indirect supervision from dictionaries and thesauruses, and so this method is also called **knowledge-based** WSD. Methods like this that do not use texts that have been hand-labeled with senses are also called weakly supervised.

The most well-studied dictionary-based algorithm for sense disambiguation is the **Lesk algorithm**, really a family of algorithms that choose the sense whose dictionary gloss or definition shares the most words with the target word's neighborhood.

Example:

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

given the following two WordNet senses:



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

Sense *bank*^{1} has two non-stopwords overlapping with the context in our sentence: *deposits* and *mortgage*, while sense *bank*^{2} has zero words, so sense *bank*^{1} is chosen.

There are many obvious extensions to Simplified Lesk. The best solution, if any sense-tagged corpus data like SemCor is available, is to add all the words in the labeled corpus sentences for a word sense into the signature for that sense. This version of the algorithm, the **Corpus Lesk** algorithm, is the best-performing of all the Lesk variants.
Instead of just counting up the overlapping words, the Corpus Lesk algorithm also applies a weight to each overlapping word. The weight is the **inverse document frequency** or **IDF**. IDF measures how many different "documents" (in this case, glosses and examples) a word occurs in and is thus a way of discounting function words. Since function words like *the, of*, etc., occur in many documents, their IDF is very low, while the IDF of content words is high. Corpus Lesk thus uses IDF instead of a stop list.

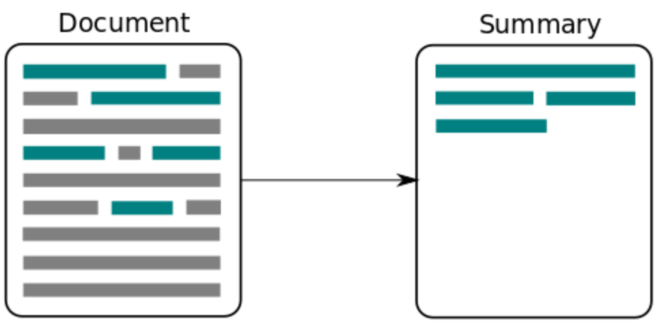
Formally, the IDF for a word i can be defined as

$$idf_i = log\left(\frac{N_{doc}}{nd_i}\right)$$

where N_{doc} is the total number of "documents" (glosses and examples) and nd_i the number of these documents containing word i.

Also in this case, as seen in 10.3.2, we can use **bootstrapping** and **unsupervised** methods to solve the WSD task.

11 Summarization



Summarization is the process of distilling the most important information from a text to produce an abridged version for a particular task and user.

There exist different kinds of summaries:

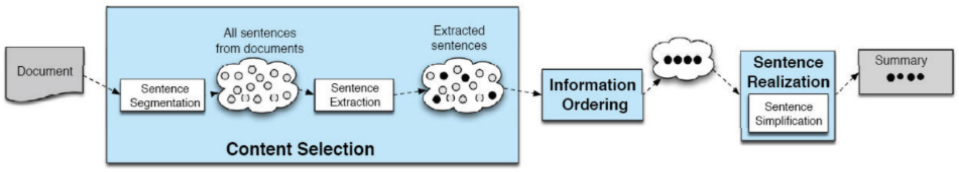
- Outlines

- Abstract
- Headlines
- Snippets (summarizing a web page on a search engine's result page)
- Action Items or other summaries (of spoken business meetings)
- Summaries of emails
- Compressed Sentences
- Answers to complex questions

More in general, we can identify the following classes:

- Single-document vs Multiple-document summarization
- Generic vs Query-focused summarization
- Abstractive vs Extractive summarization

11.1 Single document (generic, extractive)



1) Content Selection: choose pieces of text to extract from the document (granularity, usually sentences or clauses).

2) Information Ordering: choose the order of the extracted units (usually easily solved by keeping the appearance order).

3) Sentence Realization: clean up the extracted units so that they are fluent in their new context.

11.1.1 Content selection

It is a classification task, since we want to put each sentence into the *important* or *unimportant* classes. There are many methods with which one can solve this task:

- Unsupervised Content Selection
- Unsupervised Summarization based on Rhetorical Parsing
- Supervised Content Selection

Unsupervised Content Selection

The content selection task is treated as a clusterization task. The salience (i.e. informativeness) of words is computed and only the sentences with the highest number of salient words are selected.

Before going on, let's make a digression. When we're dealing with this kind of problems, documents are usually represented as **vectors** composed of **weights**.

$$\vec{d_j} = (w_{1,j}, w_{2,j}, w_{3,j}, ..., w_{M,j})$$
$$\vec{d_k} = (w_{1,k}, w_{2,k}, w_{3,k}, ..., w_{M,k})$$

Thus a documents collection can be represented as a $M \times N$ matrix $A = [w_{i,j}]$, where N is the number of documents in the collection and M the number of unique terms (word types or word forms).

With that said, one way to compute the salience of words is the **TF-IDF model**. TF-IDF, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. The TF-IDF value increases proportionally to the number of times a word appears in the document and it is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general. TF-IDF is one of the most popular term-weighting schemes today; 83% of text-based recommender systems in digital libraries use TF-IDF.

The TF-IDF is the product of two statistics, *term frequency* and *inverse document frequency*:

Term frequency:

$tf_{i,j}$ = (# occurrences of the term t_i in doc. d_j) / (# terms in doc. d_j)

Inverse document frequency:

$idf_i = log(N/($# docs containing $t_i))$

Weights:

$w_{i,j} = tf_{i,j} \cdot idf_i$

If a term (a word type or a word form) t_i is *dense* in a given document d_j, but rare in the collection, it is highly relevant for d_j.

At the end, the weight of a sentence s_k in the document j is the average weight of its non-stop words:

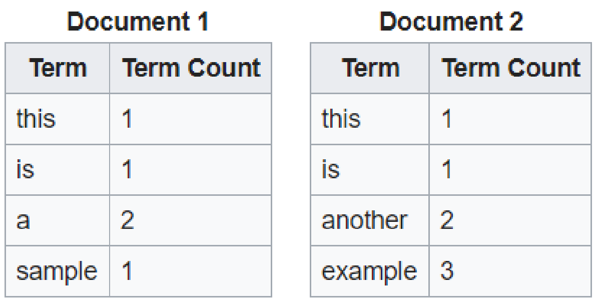
$$weight_j(s_k) = \sum_{w_i \in nonstop(s_k)} \frac{weight(w_i)}{|nonstop(s_k)|}$$

Example:

Suppose that we have term count tables of a corpus consisting of only two documents, as listed below.

$$tf(\text{"}this\text{"}, d_1) = \frac{1}{5} = 0.2$$
$$tf(\text{"}this\text{"}, d_2) = \frac{1}{7} \approx 0.14$$

An *idf* is constant per corpus, and accounts for the ratio of documents that include the word "this". In this case, we have a corpus of two documents and all of them include the word "this".



Document 1

Term	Term Count
this	1
is	1
a	2
sample	1

Document 2

Term	Term Count
this	1
is	1
another	2
example	3

$$idf(\text{“}this\text{”}, D) = log(\frac{2}{2}) = 0$$

So tf–idf is zero for the word “this”, which implies that the word is not very informative as it appears in all documents.

Let’s now analyze another approach, which is **centroid based**. The method is the **log likelihood ratio**.

Disclaimer: this is the reference from which I took the part related to the log likelihood ratio, since not completely explained in the slides: `https://www.cs.bgu.ac.il/~elhadad/nlp16/nenkova-mckeown.pdf`

Its goal is to find the signature words types, i.e. the words that “characterize” the document. The decision is made based on a test for statistical significance, hence information about the frequency of occurrence of words in a large background corpus is necessary to compute the statistic on the basis of which topic signature words are determined. The likelihood of the input I and the background corpus is computed under two assumptions:
(H1) that the probability of a word in the input is the same as in the background B or (H2) that the word has a different, higher probability, in the input than in the background.

$$H1 : P(w|I) = P(w|B) = p \quad \text{w is not descriptive}$$
$$H2 : P(w|I) = p_I \; and \; P(w|B) = p_B \; and \; p_I > p_B \quad \text{w is descriptive}$$

The likelihood of a text with respect to a given word of interest, w, is computed via the binomial distribution formula. The input and the background corpus are treated as a sequence of words $w_i : w_1 w_2 ... w_N$. The occurrence of each word is a Bernoulli trial with probability p of success, which occurs when $w_i = w$.

The overall probability of observing the word w appearing k times in the N trials is given by the binomial distribution

$$b(k, N, p) = \binom{N}{k} p^k (1-p)^{N-k}$$

For H1, the probability p is computed from the input and the background collection taken together. For H2, p_1 is computed from the input, p_2 from the background, and the likelihood of the entire data is equal to the product of the binomial for the input and that for the background. More specifically, the likelihood ratio is defined as

$$\lambda = \frac{b(k, N, p)}{b(k_I, N_I, p_I) \cdot b(k_B, N_B, p_B)}$$

The statistic equal to $-2log\lambda$ has a known statistical distribution (χ^2), which can be used to determine which words are topic signatures.

Topic signature words are those that have a likelihood statistic greater than what one would expect by chance. The probability of obtaining a given value of the statistic purely by chance can be looked up in a χ^2 distribution table; for instance a value of 10.83 can be obtained by chance with probability of 0.001.

The importance of a sentence is computed as the number of topic signatures it contains or as the proportion of topic signatures in the sentence.

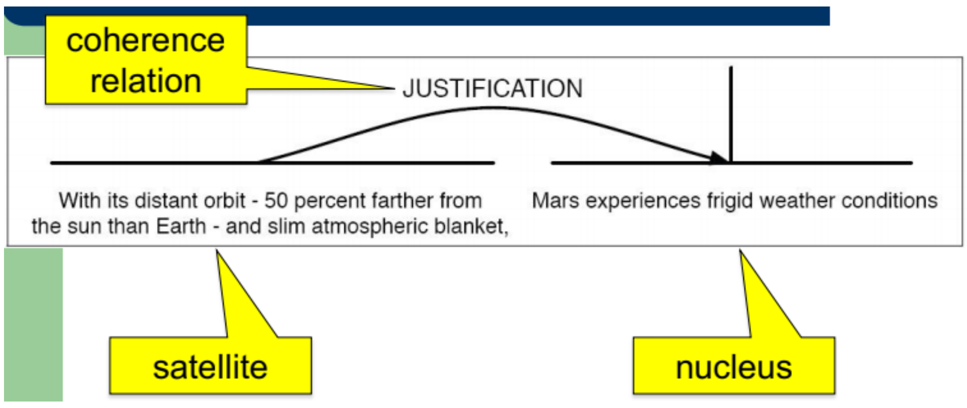
The last approach we will mention is the **centrality based approach**. Centrality-based methods compute distances between each candidate sentence and each other sentence, of the current document. In order to compute centrality, sentences are represented as a bag-of-words vector and each sentence s of the document is then assigned a centrality score:

$$centrality(s) = \frac{1}{K} \sum_y tf\text{-}idf\text{-}cosine(s, y)$$

Unsupervised Summarization based on Rethorical Parsing

"Discourse structure theories involve understanding the part-whole nature of textual documents. The task of rhetorical parsing, for example, involves understanding how two text spans are related to each other in the context. The theoretical foundation is rhetorical structure theory (RST) Mann and Thompson (1988), which a comprehensive theory of discourse organization. RST investigates how clauses, sentences and even larger text spans connect together into a whole. RST assumes that discourse is not merely a collection of random utterances but the discourse units connect to each other as a whole in a logical and topological way." (reference here).

We will see more on this in the section about **Discourse**.


coherence relation
JUSTIFICATION
With its distant orbit - 50 percent farther from the sun than Earth - and slim atmospheric blanket,
Mars experiences frigid weather conditions
satellite
nucleus

Supervised Content Selection

Weighting words is only a single cue for finding extract-worthy sentences. Many other cues exist:

- position of the sentence: sentences at the very beginning or end of the document tend to be more important
- the length of each sentence
- and so on...

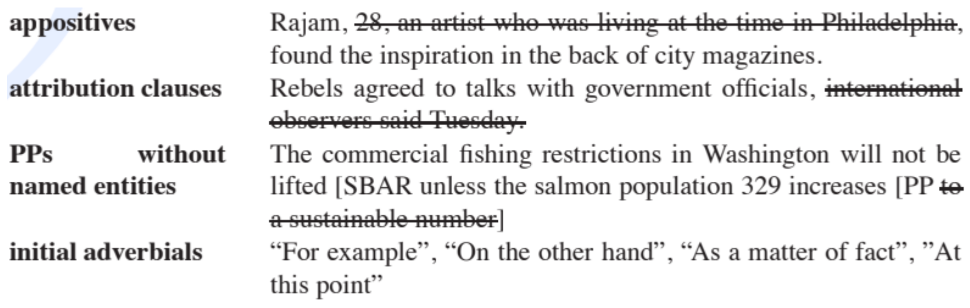
We'd like a model able to weigh and also to take into consideration also these clues. Thus we need a **supervised model**, whose goal is to classify each sample as extract-worthy or not, and a **training set** of documents paired with human-created summary extracts. A set of features is computed for the sentence to classify, and then models like Naive Bayes or MaxEnt can classify the sentence.

When we write summaries, however, we do not want to use only sentences extracted from the document. We would like to combine these sentences, change some words or even write completely new abstractive sentences. That's why, in order to generalize the corpus, let us ensure that it does not hold that each sentence in the summary is taken from the document. Although, now we need to align each document's sentence with its summary sentence(s). There are different possible algorithms to do that:

- align document and abstract sentences with the longest common subsequences of non-stopwords
- edit distance
- WordNet-based distance

11.1.2 Sentence Realization: sentence simplification

In the last step of our pipeline, we perform **Sentence Realization**, i.e. we apply some rules to select parts of the sentence to prune or keep. This is often done by running a parser/chunker over the sentences.

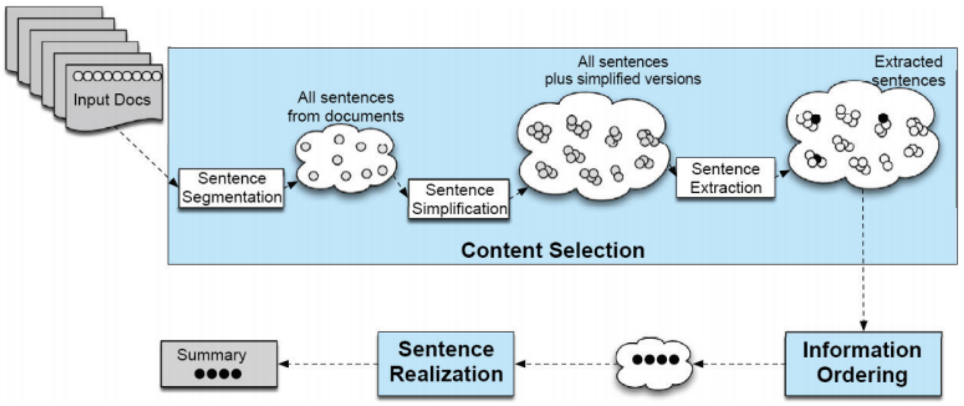


appositives	Rajam, ~~28, an artist who was living at the time in Philadelphia~~, found the inspiration in the back of city magazines.
attribution clauses	Rebels agreed to talks with government officials, ~~international observers said Tuesday.~~
PPs without named entities	The commercial fishing restrictions in Washington will not be lifted [SBAR unless the salmon population 329 increases [PP ~~to a sustainable number~~]]
initial adverbials	"For example", "On the other hand", "As a matter of fact", "At this point"

11.2 Multiple document (generic, extractive)

As we can see from the pipeline, it is quite similar to the single-document we have seen so far. One important difference is that in the multi-document summarization there is a great amount of redundancy. Since the summary should not contain redundancy:

- Algorithms for multi-document summarization focus on ways to avoid redundancy
- When adding a new sentence to the summary, be sure that it does not overlap too much with sentences already in the summary



- Redundancy factor: similarity between a candidate sentence and the sentences into the summary. A sentence is penalized if it is too similar.

So how do we select a sentence?

Maximal Marginal Relevance (MMR)

MMR is a measure for quantifying the extend of dissimilarity between the item being considered and those already selected. Higher MMR means the considered item is both relevant to the query and contains minimal similarity to previous selected items.

$$MMR = \underset{D_i \in R \setminus S}{\arg\max}[\lambda(Sim_1(D_i, Q) - (1-\lambda) \cdot max_{D_j \in S} Sim_2(D_i, D_j))]$$

where C is a document collection; Q is a query; $R = IR(C, Q, \theta)$, i.e. the ranked list of documents retrieved by an IR system, given C, Q and a relevance threshold θ; S is the subset of documents in R already selected.

Given the above definition, MMR computes incrementally the standard relevance-ranked list when the parameter $\lambda = 1$ and computes a maximal diversity ranking among the documents in R when $\lambda = 0$. For intermediate values of λ in the interval $[0, 1]$ a linear combination of both criteria is optimized.

Click on the link below to see the paper:
The Use of MMR, Diversity-Based Reranking for Reordering Documents and Producing Summaries

Clustering

This is a the standard clustering technique, applied to all the sentences in the documents to be summarized, producing a number of clusters of related sentences. We then select a single (centroid) sentence from each cluster into the summary.

11.2.1 Information Ordering in Multi-Document Summarization

To decide how to concatenate the extracted sentences into a coherent order there exist many methods:

- Chronological ordering

- Coherence
- Centering

Chronological ordering

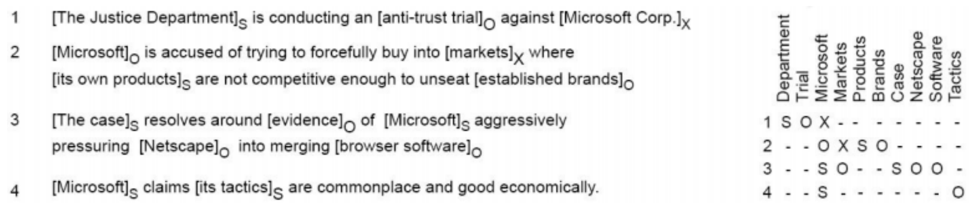
This approach is used for sentences extracted from news stories, so that the associated dates can be used. However, it turns out that pure chronological ordering can produce summaries which lack cohesion (this problem can be addressed by ordering slightly larger chunks of sentences.

Coherence

In this approach we try to obtain coherence relations between the sentences. A coherence discourse is one in which entities are mentioned in coherent patterns.
Lexical cohesion can be used as an ordering heuristic: we use the standard TF-IDF cosine distance between each pair of sentences and then we choose the overall ordering that minimizes the average distance between neighbouring sentences.

Centering

This approach is based on the fact that each discourse segment has a salient entity: the *focus*. A discourse is said to be coherent if the focus appears as certain *syntactic realizations* (i.e. as subject or object) and if the focus appears in certain *transitions* between these realizations (e.g. if the same entity is the subject of adjacent sentences). We prefer orderings in which the transition between entity mentions is a preferred one.



1 [The Justice Department]$_S$ is conducting an [anti-trust trial]$_O$ against [Microsoft Corp.]$_X$

2 [Microsoft]$_O$ is accused of trying to forcefully buy into [markets]$_X$ where [its own products]$_S$ are not competitive enough to unseat [established brands]$_O$

3 [The case]$_S$ resolves around [evidence]$_O$ of [Microsoft]$_S$ aggressively pressuring [Netscape]$_O$ into merging [browser software]$_O$

4 [Microsoft]$_S$ claims [its tactics]$_S$ are commonplace and good economically.

	Department	Trial	Microsoft	Markets	Products	Brands	Case	Netscape	Software	Tactics
1	S	O	X	-	-	-	-	-	-	-
2	-	-	O	X	S	O	-	-	-	-
3	-	-	S	O	-	-	S	O	O	-
4	-	-	S	-	-	-	-	-	-	O

For example, the transitions X,O,S,S for the entity Microsoft say that "Microsoft" in a discourse is introduced first in oblique or object position and then only later appears in subject position.

11.2.2 Selection and Ordering together: HMM

So far we have treated information extraction and ordering as separated processes. With HMMs we can instead model the problem in such a way that these two tasks are performed together.

- clusterize sentences
- each cluster c is an (unnamed) topic: hidden state
- observations correspond to sentences s
- HMM transition probability distribution:
  $P(c_j|c_i) = C_{doc}(sent(c_i) \rightarrow sent(c_j))/C_{doc}(c_i)$
- HMM emission probability distribution: $P(s|c)$

The HMM $P(c_j|c_i)$ implicitly represents information-ordering facts:

- $P(c_j|c_i)$ is high if often sentences in c_i precedes sentences in c_j; in other words: c_i is "before" c_j

- select the ordering, among all the candidates, that the HMM assigns the highest probability to

The ordering chosen for the extracted sentences may not respect coherence rules. For this reason a coreference resolution algorithm must be applied to the output, extracting names and applying cleanup rewrite rules, e.g.:

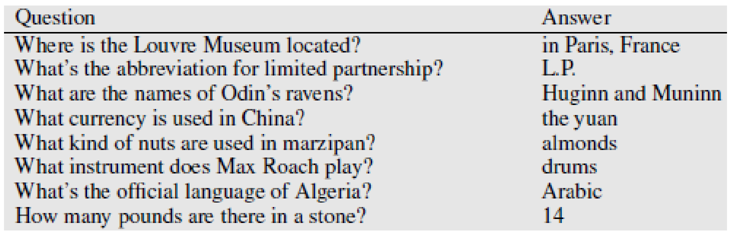
- use the full name at the first mention, and just the last name at subsequent mentions (*U.S. President George W. Bush* and then *Bush*)
- use a modified form for the first mention, but remove appositives or premodifiers from any subsequent mentions
- many more...

11.3 Focused Summarization

Summarization techniques are often used to build answers to complex questions. Two methods that can be used in this task are the following:

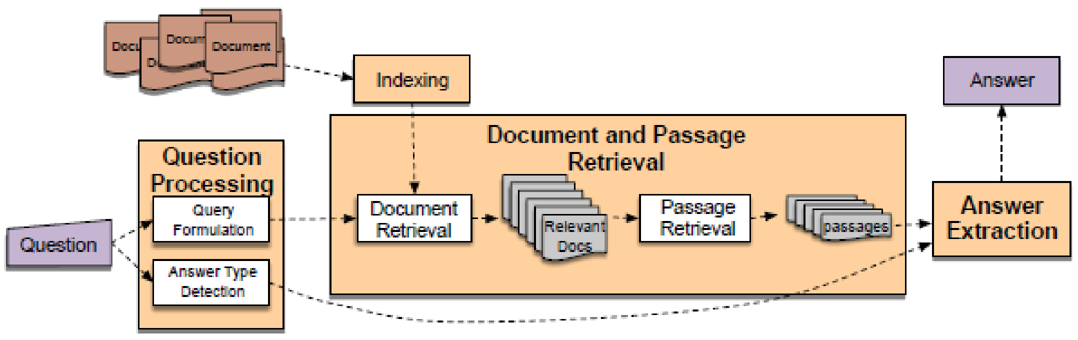
- Slightly modify the algorithms for multiple-document summarization to make use of the query, i.e. extracting sentences containing at least one word overlapping with the query
- Use Information Extraction methods

11.3.1 IR-based Factoid Question Answering



Question	Answer
Where is the Louvre Museum located?	in Paris, France
What's the abbreviation for limited partnership?	L.P.
What are the names of Odin's ravens?	Huginn and Muninn
What currency is used in China?	the yuan
What kind of nuts are used in marzipan?	almonds
What instrument does Max Roach play?	drums
What's the official language of Algeria?	Arabic
How many pounds are there in a stone?	14

Information-retrieval or **IR-based question answering** relies on the vast quantities of textual information on the web or in collections like PubMed. Given a user question, information retrieval techniques first find relevant documents and passages. Then systems (feature-based, neural, or both) use reading comprehension algorithms to read these retrieved documents or passages and draw an answer directly from spans of text.



Question Processing

The main goal of the question-processing phase is to extract the query: the keywords passed to the IR system to match potential documents. For example, for the question *Which US state capital has the largest population?* the query processing might produce:

query: "US state capital has the largest population"
answer type: city
focus: state capital

Query formulation is the task of creating a query—a list of tokens— to send to an information retrieval system to retrieve documents that might contain answer strings. For question answering from the web, we can simply pass the entire question to the web search engine, at most perhaps leaving out the question word (where, when, etc.). For question answering from smaller sets of documents like corporate information pages or Wikipedia, we still use an IR engine to index and search our documents, generally using standard tf-idf cosine matching, but we might need to do more processing. For example, for searching Wikipedia, it helps to compute tf-idf over bigrams rather than unigrams in the query and document (Chen et al., 2017). Or we might need to do query expansion, since while on the web the answer to a question might appear in many different forms, one of which will probably match the question, in smaller document sets an answer might appear only once. Query expansion methods can add query terms in hopes of matching the particular form of the answer as it appears, like adding morphological variants of the content words in the question, or synonyms from a thesaurus.

Some systems make use of question classification, the task of finding the **answer type**, the named-entity categorizing the answer. A question like *"Who founded Virgin Airlines?"* expects an answer of type PERSON. If we know that the answer type for a question is a person, we can avoid examining every sentence in the document collection, instead focusing on sentences mentioning people.
We can also use a larger hierarchical set of answer types called an **answer type taxonomy**. Such taxonomies can be built automatically, from resources like WordNet, or they can be designed by hand. Most question classifiers, however, are based on supervised learning, trained on databases of questions that have been hand-labeled with an answer type.

Document and Passage Retrieval

The IR query produced from the question processing stage is sent to an IR engine, resulting in a set of documents ranked by their relevance to the query. Because most answer-extraction methods are designed to apply to smaller regions such as passages paragraphs, QA systems next divide the top n documents into smaller passages such as sections, paragraphs, or sentences. These might be already segmented in the source document or we might need to run a paragraph segmentation algorithm. The simplest form of passage retrieval is then to simply pass along every passages to the answer extraction stage. A more sophisticated variant is to filter the passages by running a named entity or answer type classification on the retrieved passages. Passages that don't contain the answer type that was assigned to the question are discarded.

Answer Extraction

The final stage of question answering is to extract a specific answer from the passage, for example responding *29,029 feet* to a question like *"How tall is Mt. Everest?"*. This task is commonly modeled by **span labeling**: given a passage, identifying the span of text which constitutes an answer.

A simple baseline algorithm for answer extraction is to run a named entity tagger on the candidate passage and return whatever span in the passage is the correct answer type. Unfortunately, the answers to many questions, such as DEFINITION questions, don't tend to be of a particular named entity type. For this reason modern work on answer extraction uses more sophisticated algorithms, generally based on supervised learning.

I will not cover such methods. Check them on the book if you are curious.

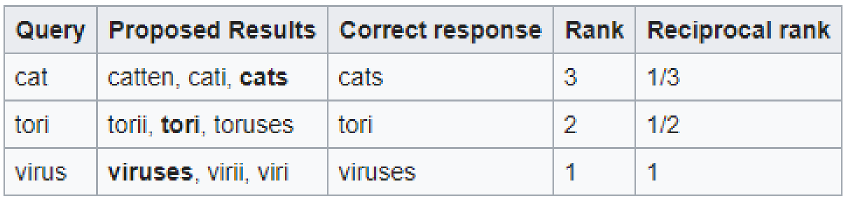
11.3.2 Evaluation of Factoid Answers

A common evaluation metric for factoid question answering, introduced in the TREC Q/A track in 1999, is mean reciprocal rank, or MRR. MRR assumes a test set of questions that have been human-labeled with correct answers. MRR also assumes that systems are returning a short ranked list of answers or passages containing answers. Each question is then scored according to the reciprocal of the rank of the first correct answer.

$$MRR = \frac{1}{N} \sum_{i=1 \ s.t. \ rank_i \neq 0} \frac{1}{rank_i}$$

where N is the number of questions of the test set.

For example, suppose we have the following three sample queries for a system that tries to translate English words to their plurals. In each case, the system makes three guesses, with the first one being the one it thinks is most likely correct:



Query	Proposed Results	Correct response	Rank	Reciprocal rank
cat	catten, cati, **cats**	cats	3	1/3
tori	torii, **tori**, toruses	tori	2	1/2
virus	**viruses**, virii, viri	viruses	1	1

Given those three samples, we could calculate the mean reciprocal rank as $(1/3 + 1/2 + 1)/3 = 11/18$ or about 0.61.

If none of the proposed results are correct, reciprocal rank is 0. Please note that only the rank of the first relevant answer is considered, possible further relevant answers are ignored. If users are interested also in further relevant items, mean average precision is a potential alternative metric.

11.3.3 Evaluation of Automatic Summarization

ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation. It is essentially of a set of metrics for evaluating automatic summarization of texts as well as machine translation. It works by comparing an automatically produced summary or translation against a set of reference summaries (typically human-produced).

System Summary (what the machine produced):

the cat was found under the bed

Reference Summary (gold standard – usually by humans):

the cat was under the bed

ROUGE-2 refers to the overlap of bigrams between the system and reference summaries, hence:

$$ROUGE2_{recall} = \frac{4}{5} = 0.8$$

$$ROUGE2_{precision} = \frac{4}{6} = 0.67$$

Reference:
What is ROUGE and how it works for evaluation of summarization tasks?

Another method of evaluation is the **Pyramid Method**.

"The idea behind the Pyramid method is that the relevance of a unit of information can be determined by how many reference summaries include it. The unit of information used by the Pyramid method is the Summary Content Unit (SCU). An SCU is a semantically atomic unit representing a single fact, but is not tied to its lexical realization; two paraphrases of the same fact represent the same SCU despite being expressed differently. An SCU is assigned a score proportional to the number of reference summaries that contain it. A Pyramid Score for a summary is calculated by taking a normalized mean of the scores of the contained SCUs. One advantage of Pyramid scores is that it directly assesses the identification of relevant facts, while ignoring their lexical realization." (`http://www.cs.columbia.edu/~smaskey/papers/pm_pyramid.pdf`)

12 Coreference Resolution

Reference in a text to an entity that has been previously introduced into the discourse is called **anaphora**.
Coreference resolution is the task of determining whether two mentions corefer, by which we mean they refer to the same entity in the discourse model (the same discourse entity).

Coreference is an important component of natural language understanding. A dialogue system that has just told the user *"There is a 2pm flight on United and a 4pm one on Cathay Pacific"* must know which flight the user means by *"I'll take the Cathay Pacific flight"*. A question answering system that uses Wikipedia to answer a question about where Marie Curie was born must know who *she* was in the sentence *"She was born in Warsaw"*. And a machine translation system translating from a language like Spanish, in which pronouns can be dropped, must use coreference from the previous sentence to decide whether the Spanish sentence *'"Me incanta el conocimiento", dice.'* should be translated as *'"I love knowledge", he said'*, or *'"I love knowledge", she said'*. Indeed, this example comes from an actual news article about a female professor and was mistranslated as *"he"* by Google Translate because of inaccurate coreference resolution **(Schiebinger, 2019)**.

12.1 Types of Referring Expressions

- Indefinite Noun Phrases: the most common form of indefinite reference in English is marked with the determiner *a* (or *an*), but it can also be marked by a quantifier such as *some* or even the determiner *this*. Indefinite reference generally introduces into the discourse context entities that are **new to the hearer**.

- Definite Noun Phrases: definite reference, such as via NPs that use the English article *the*, refers to an entity that is **identifiable to the hearer**, because already mentioned, or because identifiable from beliefs, or because inherently unique.
- Pronouns: *Emma smiled and chatted as cheerfully as* ***she*** *could.*
- Demonstrative Pronouns: demonstrative pronouns *this* and *that* can appear either alone or as determiners. Example: *I just bought a copy of Thoreau's Walden. I had bought one five years ago. That one had been very tattered; this one was in much better condition.*
- Zero Anaphora: instead of using a pronoun, in some languages (including Chinese, Japanese, and Italian) it is possible to have an anaphor that has no lexical zero anaphor realization at all:
  $[John]_i$ *went to visit some friends. On the way* $[he]_i$ *bought some wine.*
  $[Giovanni]_i$ *ando a far visita a degli amici. Per via* ϕ_i *compro del vino.*
- Inferrable: *I almost bought an Acura Integra today, but* ***the engine*** *seemed noisy.*

There are other types of referring expressions, thus we can understand that we are dealing with a very complex problem. Moreover, there are cases in which noun phrases or other nominals are not referring expressions, although they may bear a confusing superficial resemblance, thus complicating this task even more.

Let's focus on the **Pronominal Reference resolution**. Given a pronoun, we want to find the reference. In order to address this problem it is useful to introduce some constraints, which may help us both in designing novel features and performing error analyses.

12.2 Linguistic Properties of the Coreference Relation

12.2.1 Hard Constraints

- Number agreement: referring expressions and their referents must generally agree in number.
- Person agreement: English distinguishes between first, second, and third person, and a pronoun's. antecedent must agree with the pronoun in person. Thus a third person pronoun must have a third person antecedent. However, phenomena like quotation can cause exceptions.
- Gender agreement: in many languages, all nouns have grammatical gender or noun class and pronouns generally agree with the grammatical gender of their antecedent.
- Binding Theory Constraints: The binding theory is a name for syntactic constraints on the relations between a mention and an antecedent in the same sentence. Example:
  Janet bought ***herself*** *a bottle of fish sauce. [herself=Janet]*
  Janet bought ***her*** *a bottle of fish sauce. [herself* $\neq$ *Janet]*

12.2.2 Soft Constraints

- Recency: entities introduced in recent utterances tend to be more salient than those introduced from utterances further back.
- Grammatical Role: entities mentioned in subject position are more salient than those in object position, which are in turn more salient than those mentioned in oblique positions.
- Verb Semantics: Some verbs semantically emphasize one of their arguments, biasing the interpretation of subsequent pronouns. Example:
  John telephoned Bill. He lost the laptop.
  John criticized Bill. He lost the laptop.
- Selectional Restrictions: many other kinds of semantic knowledge can play a role in referent preference. For example, the selectional restrictions that a verb places on its arguments can help eliminate referents: *I ate the soup in my new bowl after cooking it for hours.* There are two possible referents for *it*, the *soup* and the *bowl.* The verb *eat*, however, requires that its direct object denote something edible, and this constraint can rule out *bowl* as a possible referent.

12.3 Algorithms for Anaphora Resolution

- Knowledge-rich approach:
  - Syntactic-based: Hobbs' algorithm
  - Discourse-based: Centering Theory
  - Hybrid approaches: Lappin and Leas
  - Corpus-based: Charniak, Hale and Ge
- Knowledge-poor approach:
  - Machine Learning

Modern systems for coreference are based on supervised neural machine learning, supervised from hand-labeled datasets like OntoNotes. However, we will describe the Lappin and Leas algorithm. **If you want to see the modern approaches, they are well defined on the book (3.ed), section 22.4.**

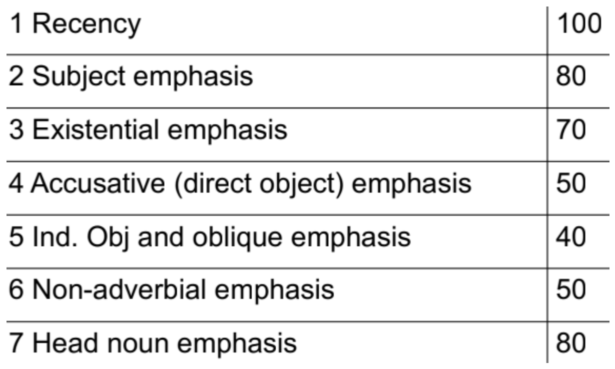
12.3.1 Lappin and Leas Algorithm (1994)

Original paper: *https://www.cis.upenn.edu/~elenimi/lappinleass1994.pdf*

The problem can be defined in a pretty simple way: given he/she/it, assign antecedent. The Lappin and Leass algorithm implements only recency and syntactic preferences and (oversimplifying) it can be splitted in two steps:

1) Discourse model update:
when a new noun phrase is encountered, add a representation to discourse model with a salience value; then modify saliences.

2) Pronoun resolution:
choose the most salient antecedent.



1 Recency	100
2 Subject emphasis	80
3 Existential emphasis	70
4 Accusative (direct object) emphasis	50
5 Ind. Obj and oblique emphasis	40
6 Non-adverbial emphasis	50
7 Head noun emphasis	80

Salience weighting is accomplished using salience factors. A given salience factor is associated with one or more discourse referents. These discourse referents are said to be in the factor's scope. A weight is associated with each factor, reflecting its relative contribution to the total salience of individual discourse referents. Initial weights are degraded in the course of processing. Degradation of salience factors occurs as the first step in processing a new sentence in the text. All salience factors that have been assigned prior to the appearance of this sentence have their weights degraded by a factor of two. When the weight of a given salience factor reaches zero, the factor is removed.

Head noun emphasis: this factor increases the salience value of an NP that is not embedded within another NP (as its complement or adjunct). Examples of NPs not receiving head noun emphasis are:

1) "the configuration information copied by *Backup configuration*."
2) "the assembly in *bay C*."
3) "the owner's manual for *an Acura Integra* is on John's desk."

Non-adverbial emphasis: any NP not contained in an adverbial PP demarcated by a separator. Like head noun emphasis, this factor penalizes NPs in certain embedded constructions. Examples of NPs not receiving non-adverbial emphasis are:

1) "Throughout *the first section* of *this guide*, these symbols are also used ... "
2) "In *the Panel definition panel*, select the Specify option from the action bar."

The algorithm performs the following operations:

- Collect the potential referents (up to 4 sentences back)
- Remove potential referents that do not agree in number or gender with the pronoun
- Remove potential references that do not pass syntactic coreference constraints
- Compute total salience value of referent from all factors, including, if applicable:
  - role parallelism (+35)
  - cataphora (-175)
- Select referent with highest salience value. In case of tie, select closest

Let's make an example:

John saw a beautiful Acura Integra today at the dealership. He showed it to Bob. He bought it.

Sentence 1 (*John saw a beautiful Acura Integra today at the dealership*):

Referent	1 Recency	2 Subject	3 Exist	4 Object	5 Ind-object	6 Non-adv	7 Head N	Total
John	100	80				50	80	310
Integra	100			50		50	80	280
dealership	100					50	80	230

Now we cut all values in half, obtaining $\{155, 140, 115\}$ respectively.

Sentence 2 (*He showed it to Bob.*):

He specifies male gender, hence Step 2 reduces set of referents to only *John*. Now update the discourse model: *He* in current sentence (recency=100), subject position (=80), not adverbial (=50), not embedded (=80), so add 310:

Referent	Phrases	Value
John	{John, he$_1$}	155+310
Integra	{a beautiful Acura Integra}	140
dealership	{the dealership}	115

Remaining on the sentence 2, we have to consider also *it*, which can be "Integra" or "dealership". Since *it* and *Integra* are objects (*dealership* is not), there is parallelism, hence +35 for "Integra", which grows to 175, while "dealership" remains at 115. Thus we pick "Integra" and the referent for *it* is found. Update discourse model: *it* is object, it gets $100 + 50 + 50 + 80 = 280$:

Referent	Phrases	Value
John	{John, he$_1$}	465
Integra	{a beautiful Acura Integra, it$_1$}	140+280
dealership	{the dealership}	115

There is also *Bob* to consider: it is an oblique argument, its weight is $100 + 40 + 50 + 80 = 270$:

Referent	Phrases	Value
John	{John, he$_1$}	465
Integra	{a beautiful Acura Integra, it$_1$}	420
Bob	{Bob}	270
dealership	{the dealership}	115

Again, we cut all values in half, obtaining $\{232.5, 210, 135, 57.5\}$ respectively.

Sentence 3 (*He bought it.*):

He_2 will be resolved to *John*, and it_2 to *Integra*.

13 Discourse Coherence

As you know, language does not normally consist of isolated, unrelated sentences, but instead of collocated, structured, coherent groups of sentences. We refer to such a coherent structured group of sentences as a **discourse**, and we use the word **coherence** to refer to the relationship between sentences that makes real discourses different than just random assemblages of sentences.

What makes a discourse coherent?

1) First, sentences or clauses in real discourses are related to nearby sentences in systematic ways. For example, the sequence *John took a train from Paris to Istanbul. He likes spinach.* is incoherent, because it is unclear to a read why the second sentence follows the first. By contrast, in the example *Jane took a train from Paris to Istanbul. She had to attend a conference.* the second sentence gives a REASON for Jane's action in the first sentence. Structured relationships like REASON that hold between text units are called **coherence relations**.

2) A second way a discourse can be locally coherent is by virtue of being "about" someone or something. In a coherent discourse some entities are **salient**, and the discourse focuses on them and doesn't go back and forth between multiple entities. This is called **entity-based coherence**.

3) Finally, discourses can be locally coherent by being **topically coherent**: nearby topically coherent sentences are generally about the same topic and use the same or similar vocabulary to discuss these topics.

In addition to the local coherence between adjacent or nearby sentences, discourses also exhibit **global coherence**. Many genres of text are associated with particular conventional discourse structures. Academic articles might have sections describing the Methodology or Results. Stories might follow conventional plotlines or motifs.

13.1 Hobbs 1979 Coherence Relations

Result:
Infer that the state or event asserted by S0 causes or could cause the state or event asserted by S1.

Explanation:
Infer that the state or event asserted by S1 causes or could cause the state or event asserted by S0.

Parallel:
Infer proposition $P(a_1, a_2, ...)$ from the assertion of S0 and $P(b_1, b_2, ...)$ from the assertion of S1, where a_i and b_i are similar, for all i.

Example:
(S0) *John bought an Acura.* $\rightarrow$ Possession(Person, Car) (S1) *Bill leased a BMW.* $\rightarrow$ Possession(Person, Car)

Elaboration:
Infer the same proposition P from the assertions of S0 and S1.

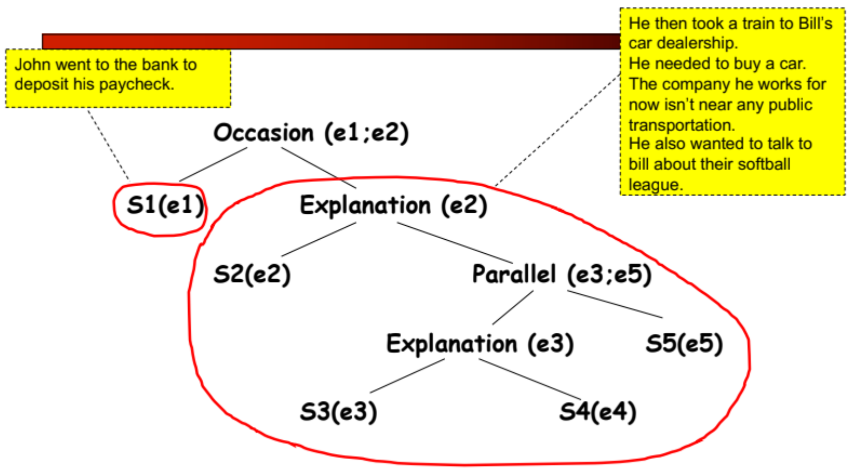
Occasion:
A change of state can be inferred from the assertion of S0, whose final state can be inferred from S1, or vice versa.

Example:
(S0) *Dorothy picked up the oil-can.*
and because of this, at the end
(S1) *She oiled the Tin Woodman's joints.*

Let's now do a complete example, taking into consideration the following text:

"John went to the bank to deposit his paycheck. (S1) He then took a train to Bill's car dealership. (S2) He needed to buy a car. (S3) The company he works for now isn't near any public transportation. (S4) He also wanted to talk to bill about their softball league. (S5)"

The discourse structure can be represented as:


John went to the bank to deposit his paycheck.
He then took a train to Bill's car dealership. He needed to buy a car. The company he works for now isn't near any public transportation. He also wanted to talk to bill about their softball league.
Occasion (e1;e2)
S1(e1)
Explanation (e2)
S2(e2)
Parallel (e3;e5)
Explanation (e3)
S5(e5)
S3(e3)
S4(e4)

13.2 Rhetorical Structure Theory

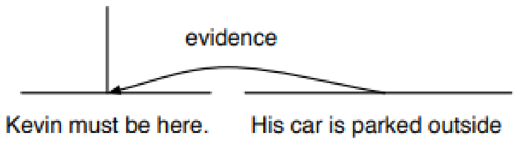
The most commonly used model of discourse organization is **Rhetorical Structure Theory (RST)** (Mann and Thompson, 1987). In RST relations are defined between two spans of text, generally a **nucleus** and a **satellite**. The nucleus is the unit that is more central to the writer's purpose and that is interpretable independently; the satellite is less central and generally is only interpretable with respect to the nucleus. Some symmetric relations, however, hold between two nuclei.

The following are some rhetorical relations:

- Reason: The nucleus is an action carried out by an animate agent and the satellite is the reason for the nucleus.
- Elaboration: The satellite gives additional information or detail about the situation presented in the nucleus.
- Evidence: The satellite gives additional information or detail about the situation presented in the nucleus. The information is presented with the goal of convince the reader to accept the information presented in the nucleus.

- Attribution: The satellite gives the source of attribution for an instance of reported speech in the nucleus.
- List: In this multinuclear relation, a series of nuclei is given, without contrast or explicit comparison.

RST relations are traditionally represented graphically; the asymmetric NucleusSatellite relation is represented with an arrow from the satellite to the nucleus:



We can also talk about the coherence of a larger text by considering the hierarchical structure between coherence relations.

"With its distant orbit–50 percent farther from the sun than Earth–and slim atmospheric blanket, Mars experiences frigid weather conditions. Surface temperatures typically average about -60 degrees Celsius (-76 degrees Fahrenheit) at the equator and can dip to -123 degrees C near the poles. Only the midday sun at tropical latitudes is warm enough to thaw ice on occasion, but any liquid water formed in this way would evaporate almost instantly because of the low atmospheric pressure."

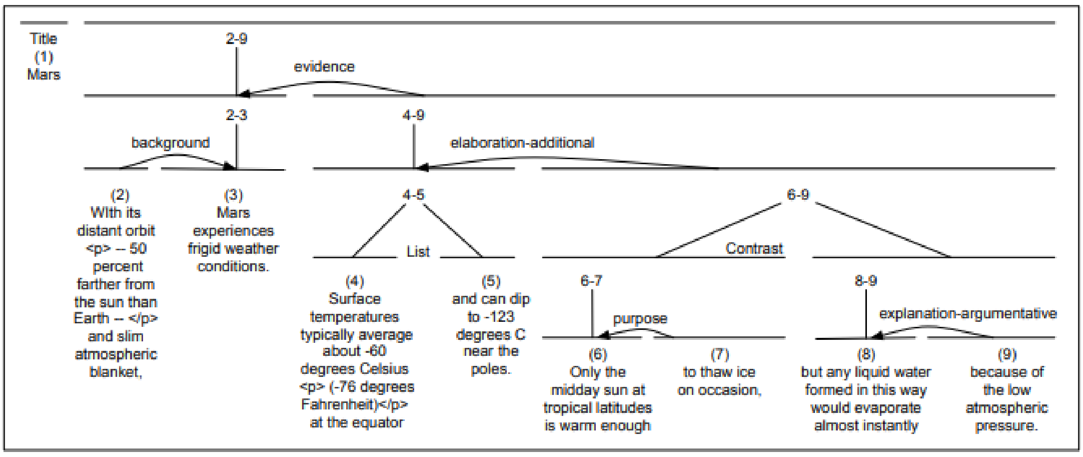

Title (1) Mars
2-9
evidence
2-3
background
4-9
elaboration-additional
(2) With its distant orbit <p> -- 50 percent farther from the sun than Earth -- </p> and slim atmospheric blanket,
(3) Mars experiences frigid weather conditions.
4-5
List
(4) Surface temperatures typically average about -60 degrees Celsius <p> (-76 degrees Fahrenheit)</p> at the equator
(5) and can dip to -123 degrees C near the poles.
6-9
Contrast
6-7
purpose
(6) Only the midday sun at tropical latitudes is warm enough
(7) to thaw ice on occasion,
8-9
explanation-argumentative
(8) but any liquid water formed in this way would evaporate almost instantly
(9) because of the low atmospheric pressure.

Figure 23.1 A discourse tree for the *Scientific American* text in (23.12), from Marcu (2000a). Note that asymmetric relations are represented with a curved arrow from the satellite to the nucleus.

13.2.1 Some Problems with RST

- How many rhetorical relations are there?
- How can we use RST in dialogue as well as monologue?
- RST does not model overall structure of the discourse.
- Difficult to get annotators to agree on labeling the same texts.

14 Dialogue Systems

Language is the mark of humanity and sentience, and conversation or dialog is the most fundamental and specially privileged arena of language. This section introduces the fundamental algorithms of conversational agents, or dialog systems. These programs communicate with users in natural language (text, speech, or even both), and generally fall into two classes:

- **Task-oriented dialog agents:** designed for a particular task and set up to have short conversations to get information from the user to help complete the task.
- **Chatbots:** systems designed for extended conversations, set up to mimic the unstructured conversational or 'chats' characteristic of human-human interaction, rather than focused on a particular task like booking plane flights.

14.1 What Makes Dialogue Different?

Dialogues exhibit anaphora and discourse structure and coherence, although with some slight changes from monologue. For example, when resolving an anaphor in dialogue it's important to look at what the other speaker said, like in the following example:

A: *There's three non-stops today.*
B: *What are they?*

where in order to realize that the pronoun *they* refers to *non-stop flights* it is required to look at the previous utterance.

Dialogue does differ from written monologue in deeper ways, however. We will now highlight some of these differences.

14.1.1 Turn-taking

One difference is the **turn-taking**. Speaker A says something, then speaker B, then speaker A, and so on. How do speakers know when is the proper time to contribute their turn? A natural conversation must be set up in a way that (most of the time) people can quickly figure out **who** should talk next, and exactly **when** they should talk. This kind of turn-taking behaviour is generally studied in the field of **Conversation Analysis (CA)**. In a key conversation-analytic paper, Sacks et. al. (1974) argued that turn-taking behaviour, at least in American English, is governed by a set of turn-taking rules. Here is a simplified version of these rules:

Turn-taking Rule.

- a. If during this turn the current speaker has selected A as the next speaker then A must speak next.
- b. If the current speaker does not select the next speaker, any other speaker may take the next turn.
- c. If no one else takes the next turn, the current speaker may take the next turn.

These apparently simple rules have many implications, one of which is the interpretation of silence: while silence can occur after any turn, silence which follows the first part of an adjacency pair-part is **significant silence**.

A: *Is there something bothering you or not?*
(1.0)
A: *Yes or no?*
(1.5)
A: *Eh?*
B: *No.*

In the above example, the silence is interpreted as a refusal to respond, or perhaps a dispreferred response.

14.1.2 Grounding

Another important characteristic of dialogue that distinguishes it from monologue is that it is a collective act performed by the speaker and the hearer. One implication of this collectiveness is that the speaker and the hearer must constantly establish **common ground** (Stalnaker, 1978), the set of things that are mutually believed by both speakers.

A: *...returning on US flight one one one eight.*
C: *Mm hmm*

The word *mm-hmm* here is a **continuer**, also often called a **backchannel** or an **acknowledgement token**. A continuer is a short utterance which acknowledges the previous utterance in some way, often cueing the other speaker to continue talking.
Continuers are just one of the ways that the hearer can indicate that he/she believes he/she understands what the speaker meant. Clark and Schaefer (1989) discuss five main types of methods, ordered from weakest to strongest:

- Continued attention: B shows she is continuing to attend and therefore remains satisfied with A's presentation.
- Relevant next contribution: B starts in on the next relevant contribution.
- Acknowledgement: B nods or says a continuer like *uh-huh, yeah*, or the like, or an assessment like *that's great.*
- Demonstration: B demonstrates all or part of what she has understood A to mean, for example by paraphrasing or reformulating A's utterance, or by collaboratively completing A's utterance.
- Display: B displays verbatim all or part of A's presentation.

14.1.3 Conversational Implicature

The final important property of conversation is the way the interpretation of an utterance relies on more than just the literal meaning of the sentences.

A: *And, whay day in May did you want to travel?*
C: *Ok uh I need to be there for a meeting that's from the 12th to the 15th.*

Notice that the client does not in fact answer the question. The speaker seems to expect the hearer to draw certain inferences; These kind of examples were pointed out by Grice (1975,1978) as part of his theory of **conversational implicature**. Implicature means a particular class of licensed inferences. Grice proposed that what enables hearers to draw these inferences is that conversation is guided by a set of **maxims**, general heuristics which play a guiding role in the interpretation of conversational utterances. He proposed the following four maxims:

- Maxim of Quantity: be exactly as informative as is required.
- Maxim of Quality: try to make your contribution one that is true.
- Maxim of Relevance: be relevant.
- Maxim of Manner: be perspicuous.

14.1.4 Dialogue Acts

An important insight about conversation, due to Austin (1962), is that an utterance in a dialogue is a kind of **action** being performed by the speaker.

I name this ship the Titanic.
I second that motion.

Verbs like *name, second...* are called **performative verbs**, and Austin called these kinds of actions **speech acts.** What makes Austin's work so far-reaching is that speech acts are not confined to this small class of performative verbs. Austin's claim is that the utterance of any sentence is a real speech situation constitutes three kinds of acts:

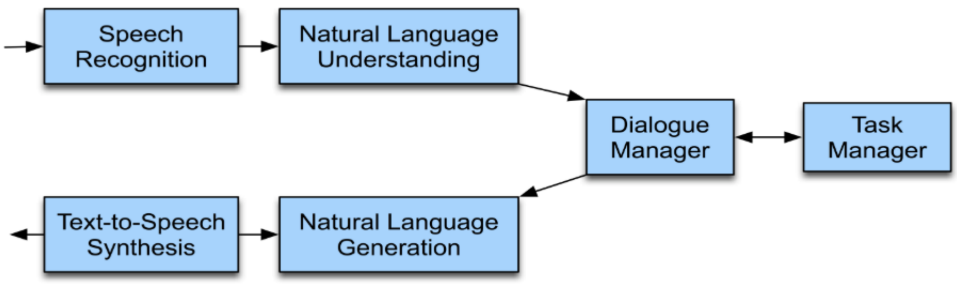
- locutionary act: the utterance of a sentence with a particular meaning.
- illocutionary act: the act of asking, answering, promising etc., in uttering a sentence.
- perlocutionary act: the (often intentional) production of certain effects upon the feelings, thoughts, or actions of the addressee in uttering a sentence.

Searle (1975), in modifying a taxonomy of Austin's, suggests that all speech acts can be classified into one of 5 major classes:

- Assertives: committing the speaker to something's being the case (*suggesting, putting forward, swearing, boasting, concluding*).
- Directives: attempts by the speaker to get the addressee to do something (*asking, ordering, requesting, inviting, advising, begging*).
- Commissives: committing the speaker to some future course of action (*promising, planning, vowing, betting, opposing*).
- Expressives: expressing the psychological state of the speaker about a state of affairs (*thanking, apologizing, welcoming, deploring*).
- Declarations: bringing about a different state of the world via the utterance (including many of the performative examples above, *I resign, You're fired*).

14.2 Dialogue System Architecture

The two blocks "Speech Recognition" and "Text-to-Speech Synthesis" will be analyzed in future sections.

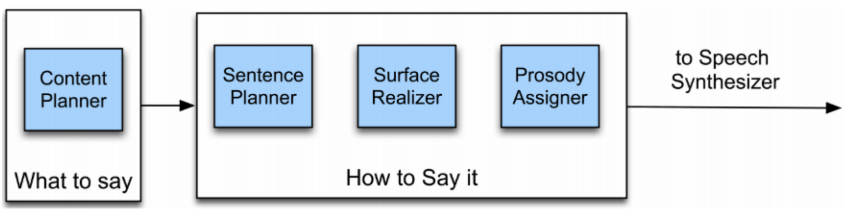


"Natural Language Understanding": there are many ways to represent the meaning of sentences. For speech dialogue systems, the most common one is the **Frame and slot semantics**:

```
SHOW:
   FLIGHTS:
       ORIGIN:
              CITY: Boston
              DATE:  Tuesday
              TIME:  morning
       DEST:
              CITY: San Francisco
```

The simplest way to generate these semantics is by using CFG in which the LHS of rules is a semantic category.

The "Natural Language Generation" block, instead, chooses syntactic structures and words to express meaning. Generators have three components: a *sentence planner*, a *surface realizer* and a *prosody assigner*:



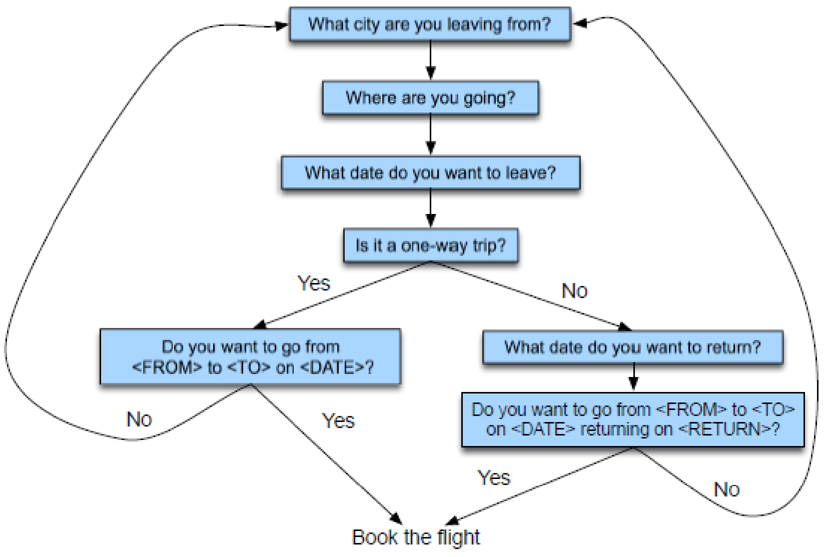
The "Task Manager" represents the behaviour of the Agent. It depends on the goal of the Agent: specific (e.g. flight booking), less specific (e.g. synthetic psychologist), general (open conversation, much more difficult). It can be modeled with Logics or with a Machine Learning approach (Reinforcement Learning is quite popular).

Let's now focus on the **Dialogue Manager**.

14.3 Frame Based Dialog Agents

Modern task-based dialog systems are based on a domain ontology, a knowledge structure representing the kinds of intentions the system can extract from user sentences. The ontology defines one or more frames, each a collection of slots, and defines the values that each slot can take. This frame-based architecture was first introduced in 1977 in the influential GUS system for travel planning.

The control architecture of frame-based dialog systems is designed around the frame. The goal is to fill the slots in the frame with the fillers the user intends, and then perform the relevant action for the user (answering a question, or booking a flight). Most frame-based dialog systems are based on finite-state automata that are hand-designed for the task by a dialog designer.


What city are you leaving from?
Where are you going?
What date do you want to leave?
Is it a one-way trip?
Yes
No
Do you want to go from <FROM> to <TO> on <DATE>?
What date do you want to return?
Do you want to go from <FROM> to <TO> on <DATE> returning on <RETURN>?
No
Yes
Yes
No
Book the flight

This system completely controls the conversation with the user. It asks the user a series of questions, ignoring (or misinterpreting) anything that is not a direct answer to the question and then going on to the next question. The speaker in initiative control of any conversation is said to have the initiative in the conversation. Systems that completely control the conversation in this way are thus called **system-initiative**.

The single-initiative finite-state dialog architecture has the advantage that the system always knows what question the user is answering. This means the system can prepare the speech recognizer with a language model tuned to answers for this question, and also makes natural language understanding easier.
Most finite-state systems also allow **universal commands** that can be said anywhere in the dialog, like *help*, to give a help message, and *start over* (or *main menu*), which returns the user to some specified main start state.
Nonetheless such a simplistic finite-state architecture is generally applied only to simple tasks such as entering a credit card number, or a name and password. For most applications, users need a bit more flexibility.

The standard GUS architecture for frame-based dialog systems, used in various forms in modern systems like Apple's Siri, Amazon's Alexa, and the Google Assistant, therefore follows the frame in a more flexible way. The system asks questions of the user, filling any slot that the user specifies, even if a user's response fills multiple slots or doesn't answer the question asked. The system simply skips questions associated with slots that are already filled. Slots may thus be filled out of sequence. The GUS architecture is thus a kind of **mixed initiative**, since the user can take at least a bit of conversational initiative in choosing what to talk about.

14.3.1 Open vs. Directive Prompts

Open prompt:

- system gives user very few constraints
- users can respond how they please
- *How may I help you?*

Directive prompt:

- explicit instructs user how to respond
- *Say yes if you accept the call; otherwise, say no.*

14.3.2 Restrictive vs. Non-restrictive grammars

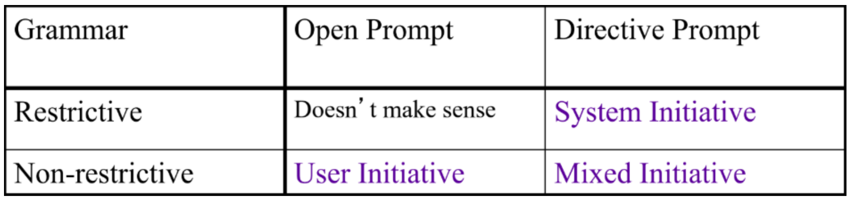
Restrictive grammar:

- language model which strongly constraints the ASR system, based on dialogue state.
- i.e. the Agent ASR is only able to understand specific words at specific conversation points.

Non-restrictive grammar:

- open language model which is not restricted to a particular dialogue state.

Summarizing:



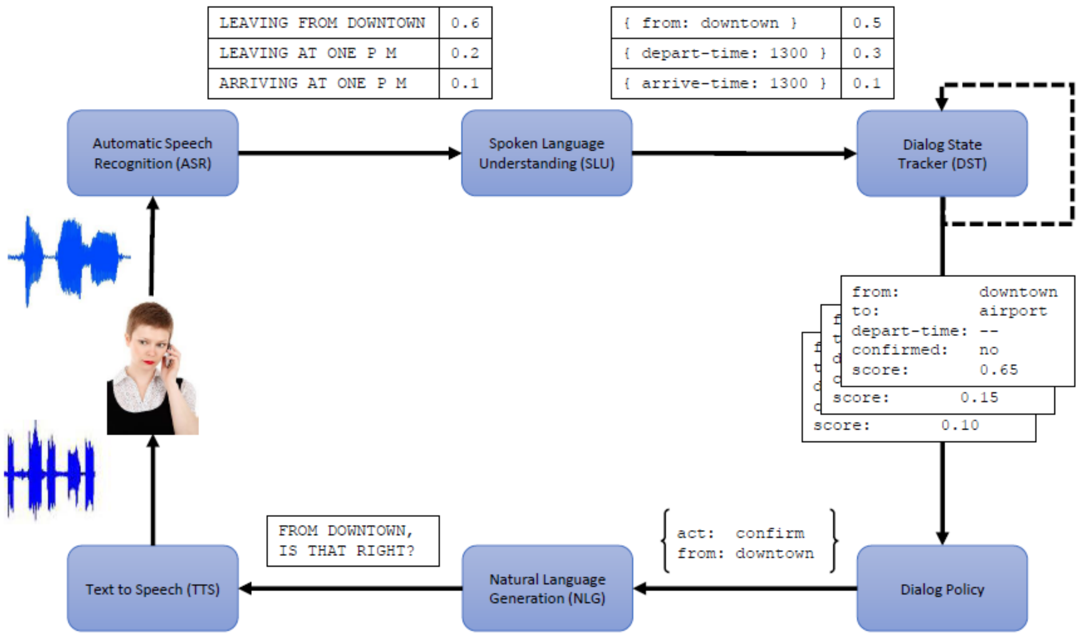
Grammar	Open Prompt	Directive Prompt
Restrictive	Doesn't make sense	System Initiative
Non-restrictive	User Initiative	Mixed Initiative

15 Advanced Dialogue Systems

Understanding and participating in dialog requires knowing whether the person you are talking to is making a statement or asking a question. Asking questions, giving orders, or making informational statements are things that people do in conversation, yet dealing with these kind of actions in dialog (what we will call **dialog acts**) is something that the GUS-style frame-based dialog systems are completely incapable of.

In this section we will introduce the **dialogue-state** architecture. Like the GUS systems, the dialog-state architecture is based on filling in the slots of frames, and so dialog-state systems have an NLU component to determine the specific slots and fillers expressed in a user's sentence. Systems must additionally determine what dialog act the user was making, also taking into account the dialog context.

Furthermore, the dialog-state architecture has a different way of deciding what to say next than the GUS systems. Simple frame-based systems often just continuously ask questions corresponding to unfilled slots and then report back the results of some database query. But in natural dialog users sometimes take the initiative, such as asking questions of the system; alternatively, the system may not understand what the user said, and may need to ask clarification questions. The system needs a **dialog policy** to decide what to say.


LEAVING FROM DOWNTOWN 0.6
LEAVING AT ONE P M 0.2
ARRIVING AT ONE P M 0.1
{ from: downtown } 0.5
{ depart-time: 1300 } 0.3
{ arrive-time: 1300 } 0.1
Automatic Speech Recognition (ASR)
Spoken Language Understanding (SLU)
Dialog State Tracker (DST)
from: downtown
to: airport
depart-time: --
confirmed: no
score: 0.65
score: 0.15
score: 0.10
act: confirm
from: downtown
FROM DOWNTOWN, IS THAT RIGHT?
Text to Speech (TTS)
Natural Language Generation (NLG)
Dialog Policy

As we can see from the image, the parts that are different than the simple GUS system are the **dialog state tracker** which maintains the current state of the dialog (which include the user's most recent dialog act, plus the entire set of slot-filler constraints the user has expressed so far) and the **dialog policy**, which decides what the system should do or say next.

In the previous section, we have spoken about the idea of speech acts and grounding. Now we will combine these two aspects into a single kind of action called **dialogue act**. Different types of dialog systems require labeling different kinds of acts, and so the tagset (defining what a dialog act is exactly) tends to be designed for particular tasks.

Dialog acts don't just appear discretely and independently; conversations have structure, and dialog acts reflect some of that structure. One aspect of this structure comes from the field of conversational analysis or CA, which defines **adjacency pairs** as a pairing of two dialog acts, like QUESTIONS and ANSWERS, PROPOSAL and ACCEPTANCE (or REJECTION), COMPLIMENTS and DOWNPLAYERS, GREETING and GREETING. This can help dialog-state models decide what actions to take. Note however that dialog acts aren't always followed immediately by their second pair part, thus making the task more complex.

15.1 Dialogue State: Interpreting Dialogue Acts

How can we interpret a dialog act, deciding whether a given input is a QUESTION, a STATEMENT, or a SUGGEST (directive)?
One could think that we can just use the surface syntactic forms as a useful cue, since yes-no questions in English have aux-inversion (the auxiliary verb precedes the subject), statements have declarative syntax (no aux-inversion), and commands have no syntactic subject. However, **the surface form can be different from the speech act type**, like in a sentence of the type "*Are you capable of giving me a list of...?*" Indeed, this would look like a INFO-REQUEST, so the answer is YES. Of course, as we know this is not the case (it's like if you ask your friend "Do you know what time is it?" and he answers "Yes."...Not so funny.

15.1.1 Dialogue Act Interpretation as Statistical Classification

There are a lot of clues in each sentence that can tell us which DA it is:

- Words and Collocations:
  - *Please* or *would you*: good cue for REQUEST
  - *Are you*: good cue for INFO-REQUEST
- Prosody:
  - Rising pitch is a good cue for INFO-REQUEST
  - Loudness/stress can help distinguish *yeah*/AGREEMENT from *yeah*/BACKCHANNEL
- Conversational Structure: *Yeah* following a proposal is probably AGREEMENT; *yeah* following an INFORM probably a BACKCHANNEL

Since our goal is to decide for each sentence what dialogue act it is, we are dealing with a classification problem with N classes, where N is the number of dialog acts. Following the order of the previous list, we have:

- Words and Syntax: probability of a sequence of words given a dialogue act:
  $P(\text{“do you”} \mid Question)$
- Prosody: probability of prosodic features given a dialogue act:
  $P(\text{[pitch rises at the end of sentence]} \mid Question)$
- Conversational Structure: probability of one dialogue act following another:
  $P(Answer \mid Question)$

As one can notice, it seems an HMM, where the Conversational Structure probability is the transition probability, while the other two are emission probabilities.

15.1.2 A special case: Detecting Correction Acts

Some dialog acts are important because of their implications for dialog control. If a dialog system misrecognizes or misunderstands an utterance, the user will generally correct the error by repeating or reformulating the utterance. Detecting these **user correction acts** is therefore quite important. Ironically, it turns out that corrections acts are actually harder to recognize than normal sentences! One reason for this is that speakers sometimes use a specific prosodic style for corrections called **hyperarticulation**, in which the utterance contains some exaggerated energy, duration, or F0 contours. Even when they are not hyperarticulating, users who are frustrated seem to speak in a way that is harder for speech recognizers.

Machine Learning algorithms can be used to detect these corrections, by using some standard features like *lexical information*, *prosodic features*, *lenght*, *ASR confidence*, *Language Model probability*, *various dialogue features (repetition)*.

15.2 Dialogue Policy: Generating Dialogue Acts (Confirmation and Rejection)

Modern dialog systems often make mistakes. It is therefore important for dialog systems to make sure that they have achieved the correct interpretation of the user's input. This is generally done by two methods: **confirming** understandings with the user and **rejecting** utterances that the system is likely to have misunderstood.

Explicit confirmation:

S: *Which city do you want to leave from?*
U: *Baltimore*
S: *Do you want to leave from Baltimore?*
U: *Yes*

Implicit confirmation:

U: *I'd like to travel to Berlin*
S: *When do you want to travel to Berlin?*

Explicit and implicit confirmation have complementary strengths. Explicit confirmation makes it easier for users to correct the system's misrecognitions since a user can just answer "no" to the confirmation question. But explicit confirmation is awkward and increases the length of the conversation. The explicit confirmation dialog fragments above sound non-natural and definitely non-human; implicit confirmation is much more conversationally natural.

Modern systems are adaptive, i.e. they decide what approach to use according to some parameters: ASR can give a **confidence metric**, which expresses how convinced system is of its transcription of the speech. If the level of confidence is high, then use implicit confirmation, otherwise use explicit confirmation.

Rejection, instead, happens when the system gives the user a prompt like *I'm sorry, I didn't understand that.*
We reject when the ASR confidence is too low or if the best interpretation is semantically ill-formed. Thus, now we can have four level of confidence:

- Below confidence threshold $\Longrightarrow$ reject
- Above threshold $\Longrightarrow$ explicit confirmation
- If even higher $\Longrightarrow$ implicit confirmation
- Even higher $\Longrightarrow$ no confirmation at all

When an utterance is rejected, systems often follow a strategy of **progressive prompting** or **escalating detail**.

S: *When would you like to leave?*
U: *Well, um, I need to be in New York in time for the first World Series game.*
S: *Sorry, I didn't get that. Please say the month and day you'd like to leave.*
U: *I wanna go on October fifteenth.*

16 Lexicons for Sentiment, Affect, and Connotation

How should affective meaning be defined? One influential typology of affective states comes from Scherer (2000), who defines each class of affective states by factors like its cognition realization and time course:

- Emotion: Relatively brief episode of response to the evaluation of an external or internal event as being of major significance. (angry, sad, joyful, fearful, ashamed, proud, elated, desperate)
- Mood: Diffuse affect state, most pronounced as change in subjective feeling, of low intensity but relatively long duration, often without apparent cause. (cheerful, gloomy, irritable, listless, depressed, buoyant)
- Interpersonal stance: affective stance taken toward another person in a specific interaction, colouring the interpersonal exchange in that situation. (distant, cold, warm, supportive, contemptuous, friendly)
- Attitude: Relatively enduring, affectively colored beliefs, preferences, and predispositions towards objects or persons. (liking, loving, hating, valuing, desiring)
- Personality traits: Emotionally laden, stable personality dispositions and behavior tendencies, typical for a person. (nervous, anxious, reckless, morose, hostile, jealous)

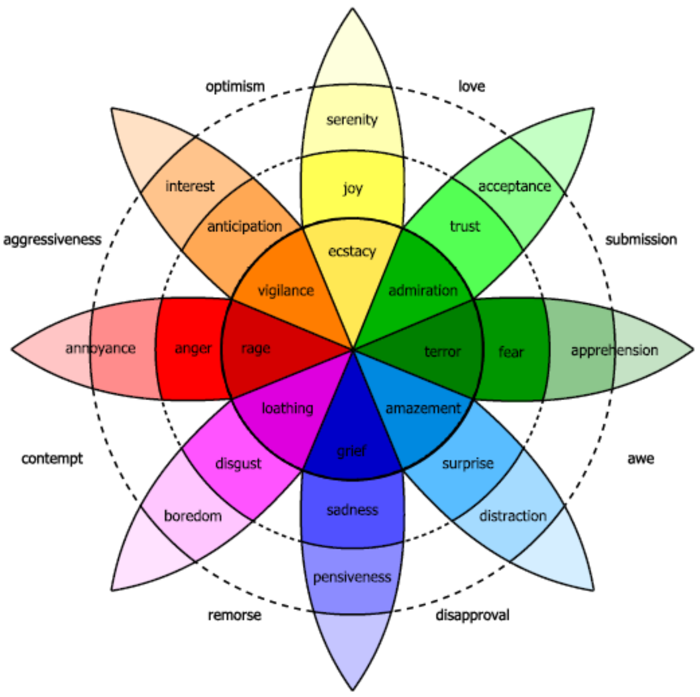
Various classifiers have been successfully applied to many of the task related to sentiment analysis etc., using all the words in the training set as input to a classifier which then determines the affect status of the text. In this case, however, we will use an alternative model, in which instead of using every word as a feature, we focus only on certain words, ones that carry particularly strong cues to affect or sentiment. We call these lists of words **affective lexicons** or **sentiment lexicons**. These lexicons presuppose a fact about semantics: that words have affective meanings or **connotations**.

16.1 Emotion

Detecting emotion has the potential to improve a number of language processing tasks. Automatically detecting emotions in reviews or customer responses (anger, dissatisfaction, trust) could help businesses recognize specific problem areas or ones that are going well. Emotion recognition could help dialog systems like tutoring systems detect that a student was unhappy, bored, hesitant, confident, and so on. Emotion can play a role in medical informatics tasks like detecting depression or suicidal intent. Detecting emotions expressed toward characters in novels might play a role in understanding how different social groups were viewed by society at different times.
There are two widely-held families of theories of emotion. In one family, emotions are viewed as fixed atomic units, limited in number, and from which others are generated, often called basic emotions (Tomkins 1962, Plutchik 1962).

The second class of emotion theories views emotion as a space in 2 or 3 dimensions (Russell, 1980). Most models include the two dimensions **valence** and **arousal**, and many add a third, **dominance**. These can be defined as:

- valence: the pleasantness of the stimulus
- arousal: the intensity of emotion provoked by the stimulus



- dominance: the degree of control exerted by the stimulus

16.2 Semi-supervised Induction of Affect Lexicons

16.2.1 Semantic axis methods

One of the most well-known lexicon induction methods, the Turney and Littman (2003) algorithm, is given seed words like good or bad, and then for each word w to be labeled, measures both how similar it is to good and how different it is from bad. Here we describe a slight extension of the algorithm due to An et al. (2018), which is based on computing a semantic axis.
In the first step, we choose seed words by hand. Because the sentiment or affect of a word is different in different contexts, it's common to choose different seed words for different genres, and most algorithms are quite sensitive to the choice of seeds.
In the second step, we compute embeddings for each of the pole words. These embeddings can be off-the-shelf word2vec embeddings, or can be computed directly on a specific corpus (for example using a financial corpus if a finance lexicon is the goal), or we can fine-tune off-the-shelf embeddings to a corpus. Fine-tuning is especially important if we have a very specific genre of text but don't have enough data to train good embeddings. In fine-tuning, we begin with off-the-shelf embeddings like word2vec, and continue training them on the small target corpus. Once we have embeddings for each pole word, we we create an embedding that represents each pole by taking the centroid of the embeddings of each of the seed words; recall that the centroid is the multidimensional version of the mean.

Given a set of embeddings for the positive seed words

$$S^+ = \{E(w_1^+), E(w_2^+), ..., E(w_n^+)\}$$

and embeddings for the negative seed words

$$S^- = \{E(w_1^-), E(w_2^-), ..., E(w_m^-)\}$$

the pole centroids are:

$$\boldsymbol{V}^+ = \frac{1}{n}\sum_1^n E(w_i^+)$$

$$\boldsymbol{V}^- = \frac{1}{m}\sum_1^m E(w_i^-)$$

The semantic axis defined by the poles is computed just by subtracting the two vectors:

$$\boldsymbol{V_{axis}} = \boldsymbol{V}^+ - \boldsymbol{V}^-$$

$\boldsymbol{V}_{axis}$ is a vector in the direction of sentiment. Finally, we compute how close each word w is to this sentiment axis, by taking the cosine between w's embedding and the axis vector. A higher cosine means that w is more aligned with S^+ than S^-.

$$score(w) = cos(E(w), V_{axis}) = \frac{E(w)\cdot \boldsymbol{V}_{axis}}{||E(w)||\cdot||\boldsymbol{V}_{axis}||}$$

16.2.2 Label propagation

An alternative family of methods defines lexicons by propagating sentiment labels on graphs, an idea suggested in early work by Hatzivassiloglou and McKeown (1997). We'll describe the simple SentProp (Sentiment Propagation) algorithm of Hamilton et al. (2016a), which has four steps:

1. **Define a graph:** given word embeddings, build a weighted lexical graph by connecting each word with its k nearest neighbors (according to cosine similarity). The weights of the edge between words w_i and w_j are set as:

$$\boldsymbol{E_{i,j}} = arccos\Big(-\frac{\boldsymbol{w_i^T w_j}}{||\boldsymbol{w_i}||\;||\boldsymbol{w_j}||}\Big)$$

2. **Define a seed set:** by hand, choose positive and negative seed words.

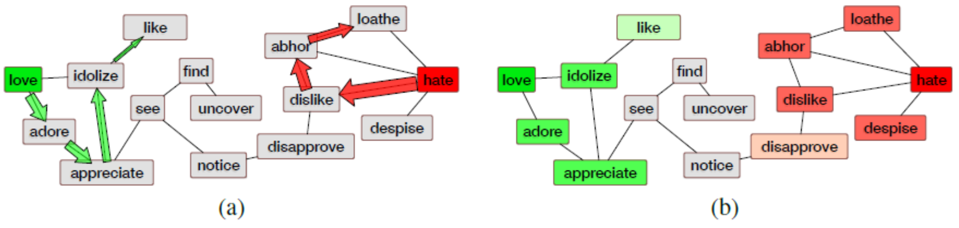
3. **Propagate polarities from the seed set:** now we perform a random walk on this graph, starting at the seed set. In a random walk, we start at a node and then choose a node to move to with probability proportional to the edge probability. A word's polarity score for a seed set is proportional to the probability of a random walk from the seed set landing on that word.

4. **Create word scores:** we walk from both positive and negative seed sets, resulting in positive ($score^+(w_i)$) and negative ($score^-(w_i)$) label scores. We then combine these values into a positive-polarity score as:

$$score^{+}(w_i) = \frac{score^{+}(w_i)}{score^{+}(w_i) + score^{-}(w_i)}$$

It's often helpful to standardize the scores to have zero mean and unit variance within a corpus.

5. **Assign confidence to each score:** because sentiment scores are influenced by the seed set, we'd like to know how much the score of a word would change if a different seed set is used. We can use bootstrap-sampling to get confidence regions, by computing the propagation B times over random subsets of the positive and negative seed sets (for example using $B = 50$ and choosing 7 of the 10 seed words each time). The standard deviation of the bootstrap-sampled polarity scores gives a confidence measure.



16.2.3 Using WordNet to Learn Polarity

A word's synonyms presumably share its polarity while a word's antonyms probably have the opposite polarity. After a seed lexicon is built, each lexicon is updated as follows, possibly iterated.

Lex^{+}: Add synonyms of positive words (*well*) and antonyms (like *fine*) of negative words
Lex^{-}: Add synonyms of negative words (*awful*) and antonyms (like *evil*) of positive words

An extension of this algorithm assigns polarity to WordNet senses, called **SentiWordNet**, where polarity is assigned to entire synsets rather than words.

In summary, semisupervised algorithms use a human-defined set of seed words for the two poles of a dimension, and use similarity metrics like embedding cosine, coordination, morphology, or thesaurus structure to score words by how similar they are to the positive seeds and how dissimilar to the negative seeds.

16.3 How to Measure Polarity of a Phrase?

Positive phrases co-occur more with "excellent".
Negative phrases co-occur more with "poor".

But how can we measure co-occurence?

16.3.1 Mutual Information

The mutual information (MI) of two random variables is a measure of the mutual dependence between the two variables. More specifically, it quantifies the "amount of information" obtained about one random variable through observing the other random variable.

$$I(X,Y) = \sum_{x \in X} \sum_{y \in Y} P(x,y)\ log_2 \frac{P(x,y)}{P(x)P(y)}$$

Pointwise Mutual Information:

In contrast to mutual information, PMI refers to single events, whereas MI refers to the average of all possible events.

$$PMI(x,y) = log_2 \frac{P(x,y)}{P(x)P(y)}$$

So how much more do two words co-occur than if they were independent?

$$PMI(word_1, word_2) = log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}$$

In computational linguistics, PMI has been used for finding collocations and associations between words. For instance, countings of occurrences and co-occurrences of words in a text corpus can be used to approximate the probabilities $p(x)$ and $p(x,y)$ respectively.

For example, let's compute the PMI value of the words *puerto* and *rico* using the first 50 millions words of Wikipedia as text collection:

$word_1$: *puerto*
$countword_1$: 1938

$word_2$: *rico*
$countword_2$: 1311

We know that the count of co-occurences is 1159, hence:

$$PMI(puerto, rico) = ln\Big(50.000.000 \cdot \frac{1159}{1938 \cdot 1311}\Big) = 10.034$$

16.4 Supervised Learning of Word Sentiment

The web contains an enormous number of online reviews for restaurants, movies, books, or other products, each of which have the text of the review along with an associated review score: a value that may range from 1 star to 5 stars, or scoring 1 to 10.
We can use this review score as supervision: positive words are more likely to appear in 5-star reviews; negative words in 1-star reviews. And instead of just a binary polarity, this kind of supervision allows us to assign a word a more complex representation of its polarity: its distribution over stars (or other scores).

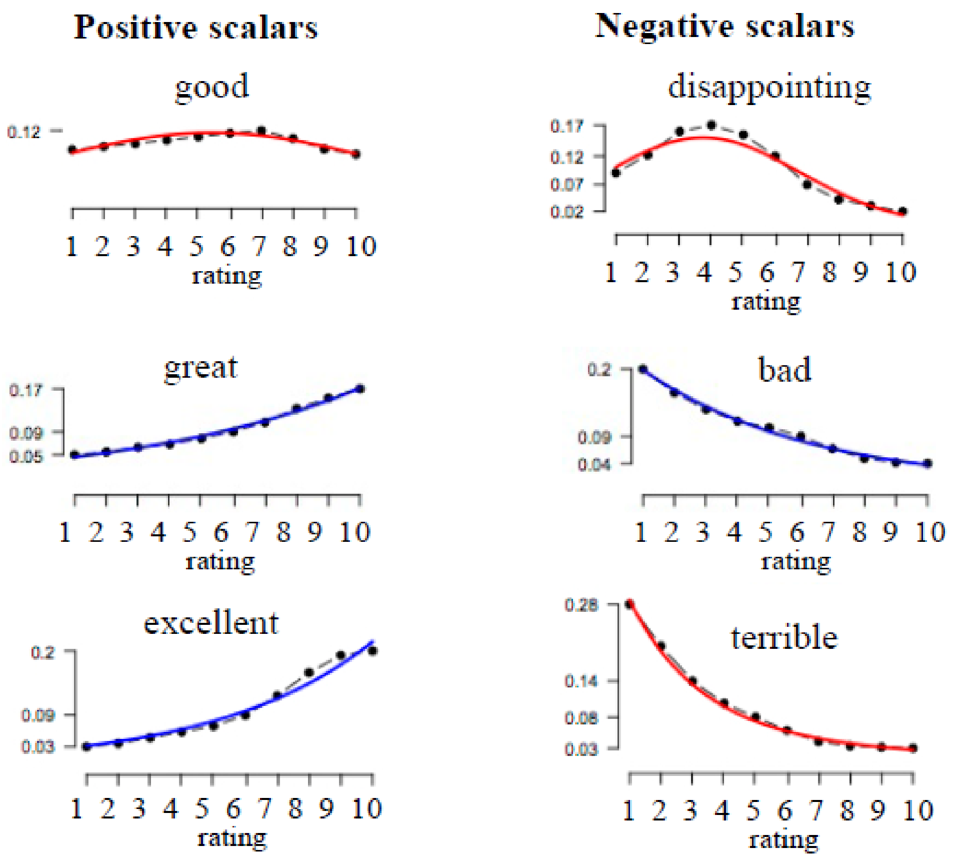
For example, we could compute the IMDB likelihood of a word like *disappoint(ed/ing)* occurring in a 1 star review by dividing the number of times *disappoint(ed/ing)* occurs in 1-star reviews in the IMDB dataset (8,557) by the total number of words occurring in 1-star reviews (25,395,214), so the IMDB estimate of $P(disappointing \mid 1)$ is .0003.

A slight modification of this weighting, the normalized likelihood, can be used as an illuminating visualization (Potts, 2011):

$$P(w|c) = \frac{count(w,c)}{\sum_{w \in C} count(w,c)}$$

$$PottsScore(w) = \frac{P(w|c)}{\sum_c P(w|c)}$$

Dividing the IMDB estimate $P(disappointing \mid 1)$ of .0003 by the sum of the likelihood $P(w|c)$ over all categories gives a Potts score of 0.10. The word disappointing thus is associated with the vector $[.10, .12, .14, .14, .13, .11, .08, .06, .06, .05]$.



16.5 Affect Recognition

The most common algorithms involve supervised classification: a training set is labeled for the affective meaning to be detected, and a classifier is built using features extracted from the training set. As with sentiment analysis, if the training set is large enough, and the test set is sufficiently similar to the training set, simply using all the words or all the bigrams as features in a powerful classifier like SVM or logistic regression is an excellent algorithm whose performance is hard to beat. Thus we can treat affective meaning classification of a text sample as simple document classification.

Some modifications are nonetheless often necessary for very large datasets. For example, the Schwartz et al. (2013) study of personality, gender, and age using 700 million words of Facebook posts used only a subset of the n-grams of lengths 1-3. Only words and phrases used by at least 1% of the subjects were included as features, and 2-grams and 3-grams were only kept if they had sufficiently high PMI (PMI greater than $2 \cdot$ length, where length is the number of words):

$$pmi(phrase) = log \frac{p(phrase)}{\prod_{w \in phrase} p(w)}$$

Various weights can be used for the features, including the raw count in the training set, or some normalized probability or log probability. Schwartz et al. (2013), for example, turn feature counts into phrase likelihoods by normalizing them by each subject's total word use:

$$p(phrase|subject) = \frac{freq(phrase, subject)}{\sum_{phrase' \in vocab(subject)} freq(phrase', subject)}$$

16.6 Personality Detection

Many theories of human personality are based around a small number of dimensions, such as various versions of the "Big Five" dimensions (Digman, 1990):

- Extroversion vs. Introversion: sociable, assertive, playful vs. aloof, reserved, shy
- Emotional stability vs. Neuroticism: calm, unemotional vs. insecure, anxious
- Agreeableness vs. Disagreeableness: friendly, cooperative vs. antagonistic, faultfinding
- Conscientiousness vs. Unconscientiousness: self-disciplined, organized vs. inefficient, careless
- Openness to experience: intellectual, insightful vs. shallow, unimaginative

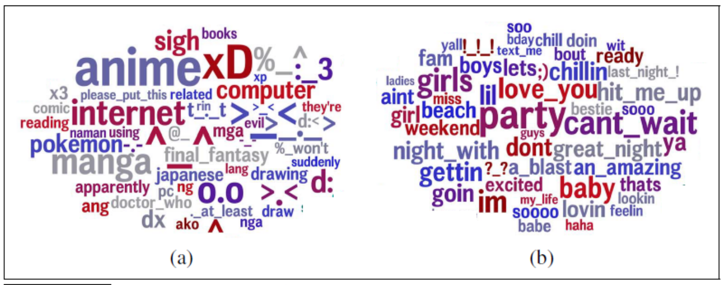


Figure 19.12 Word clouds from Schwartz et al. (2013), showing words highly associated with introversion (left) or extroversion (right). The size of the word represents the association strength (the regression coefficient), while the color (ranging from cold to hot) represents the relative frequency of the word/phrase (from low to high).

17 Computational Phonology (Speech Recognition)

The core task of **automatic speech recognition** is take an acoustic waveform as input and produce as output a string of words. The core task of **text-to-speech** synthesis is to take a sequence of text words and produce as output an acoustic waveform.
Phonology is the area of linguistics that describes the systematic way that sounds are differently realized in different environments, and how this system of sounds is related to the rest of the grammar. **Phonetics** is the study of the speech sounds used in the languages of the world. We will be modeling the pronunciation of a word as a string of symbols which represent **phones** or **segments**. A phone is a speech sound. Phones are divided into two main classes: **consonants** and **vowels**. Consonants are made by restricting or blocking the airflow in some way, and

may be voiced or unvoiced. Vowels have less obstruction, are usually voiced, and are generally louder and longer-lasting than consonants.
Consonants and vowels combine to make a **syllable**. There is no completely agreed-upon definition of a syllable; roughly speaking a syllable is a vowel-like sound together with some of the surrounding consonants that are most closely associated with it.

17.1 The Phoneme and Phonological Rules

All [t]s are not created equally. That is, phones are often produced differently in different contexts. For example, consider the different pronunciation of [t] in the words *tunafish* and *starfish*. The [t] of *tunafish* is **aspirated**. Aspiration is a period of voicelessness after a stop closure and before the onset of voicing of the following vowel. Since the vocal cords are not vibrating, aspiration sounds like a puff of air after the [t] and before the vowel. By contrast, a [t] following an initial [s] is **unaspirated**; thus the [t] in *starfish* has no period of voicelessness after the [t] closure. This variation in the realization of [t] is predictable: whenever a [t] begins a word or unreduced syllable in English, it is aspirated. The same variation occurs for [k], e.g. in Jimi Hendrix's lyrics:

"*Scuse me, while I kiss the sky*"

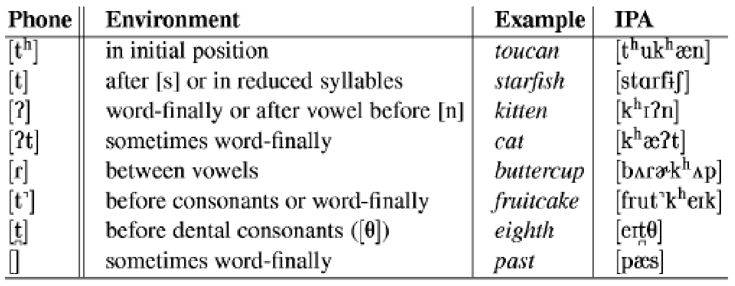
is often understood as

"*Scuse me, while I kiss this guy*"

How do we represent this relation between a [t] and its different realizations in different contexts? We generally capture this kind of pronunciation variation by positing an abstract class called the **phoneme**, which is realized as different **allophones** in different contexts.
The relationship between a phoneme and its allophones is often captured by writing a **phonological rule**:

$/\mathrm{t}/ \rightarrow [\mathrm{t}]\ /\_\_\theta$

In this notation, the surface allophone appears to the right of the arrow, and the phonetic environment is indicated by the symbols surrounding the underbar.



Phone	Environment	Example	IPA
[tʰ]	in initial position	*toucan*	[tʰukʰæn]
[t]	after [s] or in reduced syllables	*starfish*	[stɑrfɨʃ]
[ʔ]	word-finally or after vowel before [n]	*kitten*	[kʰɪʔn]
[ʔt]	sometimes word-finally	*cat*	[kʰæʔt]
[ɾ]	between vowels	*buttercup*	[bʌɾɚkʰʌp]
[t̚]	before consonants or word-finally	*fruitcake*	[frut̚kʰeɪk]
[t̪]	before dental consonants ([θ])	*eighth*	[eɪt̪θ]
[]	sometimes word-finally	*past*	[pæs]

17.2 Acoustic Processing of Speech

The input to a speech recognizer, like the input to the human ear, is a complex series of changes in air pressure. We represent sound waves by plotting the change in air pressure over time. Two important characteristics of a wave are its **frequency** and **amplitude**.
Two important perceptual properties are related to frequency and amplitude. The **pitch** of a sound is the perceptual correlate of frequency; in general if a sound has a higher-frequency

we perceive it as having a higher pitch, although the relationship is not linear, since human hearing has different acuities for different frequencies. Similarly, the **loudness** of a sound is the perceptual correlate of the **power**, which is related to the square of the amplitude. So sounds with higher amplitudes are perceived as louder, but again the relationship is not linear.

While some broad phonetic features (presence of voicing, stop closures, fricatives) can be interpreted from a waveform, more detailed classification (which vowel? which fricative?) requires a different representation of the input in terms of **spectral** features. Spectral features are based on the insight of Fourier that every complex wave can be represented as a sum of many simple waves of different frequencies.
A **spectrum** is a representation of these different frequency components of a wave and it can be computed by a **Fourier transform**.

Why is spectrum useful? It turns out that these spectral peaks that are easily visible in a spectrum are very characteristic of different sounds; phones have characteristic spectral "signatures". For example, different chemical elements give off different wavelengths of light when they burn, allowing us to detect elements in stars light-years away by looking at the spectrum of the light. Similarly, by looking at the spectrum of a waveform, we can detect the characteristic signature of the different phones that are present.
While a spectrum shows the frequency components of a wave at one point in time, a **spectrogram** is a way of envisioning how the different frequencies which make up a waveform change over time. The dark horizontal bars on a spectrogram, representing spectral peaks, usually of vowels, are called **formants**.

17.2.1 Feature Extraction

Disclaimer: *this part is extensively discussed in the book, while here, following what we did during the course, I will provide a brief overview of the main steps and concepts.*

What we want to do is this: *sound wave → feature vector*

In just two words, the idea is the following: the input soundwave is first **digitalized**. This process has two steps:

- Sampling: a signal is sampled by measuring its amplitude at a particular time; the sampling rate is the number of samples taken per second.
- Quantization: since even a 8000 Hz sampling rate requires 8000 amplitude measurements for each second of speech, it is important to store the amplitude measurements efficiently. Thus they are usually stored as integers, either 8-bit or 16-bit. This process of representing a real-valued number as a integer is called quantization.

Once a waveform has been digitalized, it is converted to some set of spectral features. One popular feature set is **cepstral coefficients**.
Taking into consideration a source-filter model of the human phonation, we can say that the glottis produces pulses (source), while the vocal tract shapes such pulses (filter). Source and filter are not equally useful. The source, indeed, is not important for ASR; what we are interested in is the filter, which carries the important information. The **cepstrum** can be used to separate source and filter and it can be derived as follows:

- Take a digitized waveform $x[n]$
- Compute a discrete Fourier transform $X[k]$

- Consider the discrete Fourier transform as a new digitized waveform
- Compute the discrete Fourier transform of the log of such a new waveform
- The cepstrum coefficients are the amplitudes of such new spectrum

What is actually used is the **mel-cepstrum**, obtained by applying the cepstrum to the mel-spectrum. The difference between the cepstrum and the mel-frequency cepstrum is that in the MFC, the frequency bands are equally spaced on the mel scale, which approximates the human auditory system's response more closely than the linearly-spaced frequency bands used in the normal cepstrum.

Thus, the MFCC coefficients that we have obtained are:

- First mel-cepstrum coefficients $c_m[n] \quad 0 \leq n \leq 11$
- Variations
  - Deltas:
$$d(c_m[n]) = \frac{c_m[n+1] - c_m[n-1]}{2} \quad 0 \leq n \leq 11$$
  - Double-deltas:
$$dd(c_m[n]) = \frac{d_m[n+1] - d_m[n-1]}{2} \quad 0 \leq n \leq 11$$
  - Energy:
    * Energy of the samples: $E_m = \sum_{n=0}^{L-1} x^2[n]$
    * Energy of deltas: $E_d = \sum_{n=0}^{L-1} d^2(c_m[n])$
    * Energy of double-deltas: $E_{dd} = \sum_{n=0}^{L-1} dd^2(c_m[n])$

Thus 39 coefficients (tend to be uncorrelated) for each chunk ($\sim$ 10 ms).

17.3 Speech Recognition Architecture

The problem that we want to solve can be expressed as follows:

given a vocal signal, composed of M MFCC vectors, compute the right sequence of M subphones s:

$$\hat{s}[0:M-1] = \underset{s[0:M-1]}{\arg\max}(P(s[0:M-1]|MFCC[0:M-1]))$$

Thus, we can adopt the usual Bayesian approach:

$$\begin{aligned} P(s[0:M-1]|MFCC[0:M-1]) &= \frac{P(MFCC[0:M-1]|s[0:M-1]) \cdot P(s[0:M-1])}{P(MFCC[0:M-1])} \\ &= \frac{\prod_{m=0}^{M-1} P(MFCC[m]|s[m]) \cdot P(s[0:M_s-1])}{P(MFCC[0:M-1])} \\ &= \frac{\prod_{m=0}^{M-1} P(MFCC[m]|s[m]) \cdot P(s[m]|s[m-1])}{P(MFCC[0:M-1])} \end{aligned}$$

where we assumed that the MFCC are independent, that MFCC[m] only depends on s[m] and in the last step we adopted the Markov assumption.

Thus...

$$\begin{aligned}\hat{s}[0:M-1] &= \underset{s[0:M-1]}{\arg\max}\left(\frac{\prod_{m=0}^{M-1} P(MFCC[m]|s[m]) \cdot P(s[m]|s[m-1])}{P(MFCC[0:M-1]}\right) \\ &= \underset{s[0:M-1]}{\arg\max}\prod_{m=0}^{M-1} P(MFCC[m]|s[m]) \cdot P(s[m]|s[m-1])\end{aligned}$$

The model, as we can see, is a HMM. More formally:

- $Q = \{s_1, s_2, ..., s_{N_s}\}$: set of subphones
- O: set of observations (MFCC vectors)
- $A = \{a_{i,j}\}$: transition probability matrix
  $a_{i,j} = P(s[m] = s_j|s[m-1] = s_i)$
- $B = \{b_j(o_t)\}$: emission probability matrix
  $b_j(o_t) = P(MFCC[m] = o_t|s[m] = s_j)$

Now, how are the features vectors turned into probabilities? There are two different approaches:

- Discretize MFCC $\rightarrow$ discrete HMM: B stored as an actual matrix
- Continue MFCC $\rightarrow$ use Gaussian Mixture Model (GMM), where $b_j(o_t)$ is actually a function (no matrix B exists) and we store the *parameters* of $b_j(o_t)$

17.3.1 Computing Acoustic Probabilities

One way to compute probabilities on feature vectors is to first cluster them into discrete symbols that we can count; we can then compute the probability of a given cluster just by counting the number of times it occurs in some training set. This method is usually called **vector quantization** and it was quite common in early speech recognition algorithms. It has been replaced by a more direct but compute-intensive approach: computing observation probabilities on a real-valued (continuous) input vector. This method computes a **probability density function** or **pdf** over a continuous space.
There are two popular version of the continuous approach. The most widespread of the two is the use of **Gaussian** pdfs, in the simplest version of which each state has a single Gaussian function which maps the observation vector o_t to a probability. An alternative approach is the use of **neural networks**.

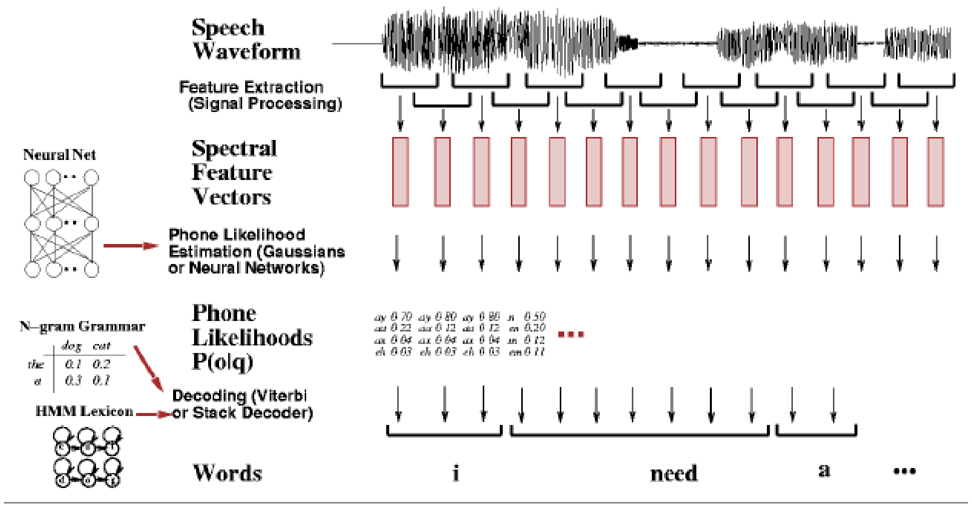
In the simplest use of Gaussians, we assume that the possible values of the observation feature vector o_t are normally distributed, and so we represent the observation probability function $b_j(o_t)$ as a Gaussian curve with mean vector μ_j and covariance matrix Σ_j.

$$b_j(o_t) = \frac{1}{\sqrt{2\pi\left|\Sigma j\right|}} e^{[(o_t-\mu_j)^T \Sigma_j^{-1}(o_t-\mu_j)]}$$

Usually we make the simplifying assumption that the covariance matrix Σ_j is diagonal, i.e. that it contains the simple variance of cepstral feature 1, the simple variance of cepstral feature 2, and so on, without worrying about the effect of cepstral feature 1 on the variance of cepstral feature 2. This means that in practice we are keeping only a single separate mean and variance for each feature in the feature vector.
Most recognizers do something even more complicated; they keep multiple Gaussians for each state, so that the probability of each feature of the observation vector is computed by adding together a variety of Gaussian curves. This technique is called **Gaussian mixtures**:

$$b_j(o_t) = \sum_{n=0}^{G-1} c_n^{(j)} \cdot g(o_t|\mu_n^{(j)}, \Sigma_n^{(j)})$$

$$g(o_t|\mu_n^{(j)}, \Sigma_n^{(j)}) = \frac{1}{\sqrt{2\pi\left|\Sigma_n^{(j)}\right|}} e^{[(o_t-\mu_n^{(j)})^T(\Sigma_n^{(j)})^{-1}(o_t-\mu_n^{(j)})]}$$

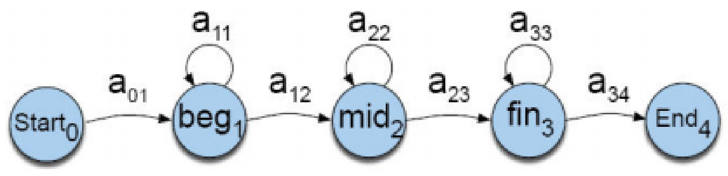

Speech Waveform
Feature Extraction (Signal Processing)
Neural Net
Spectral Feature Vectors
Phone Likelihood Estimation (Gaussians or Neural Networks)
N-gram Grammar
Phone Likelihoods P(o|q)
Decoding (Viterbi or Stack Decoder)
HMM Lexicon
Words
i
need
a
...

Now, as we can see in the above image, the only thing the we have to do is the **decoding phase**. Before proceeding in that direction, however, let's represent our HMM.

17.3.2 Overview of HMM

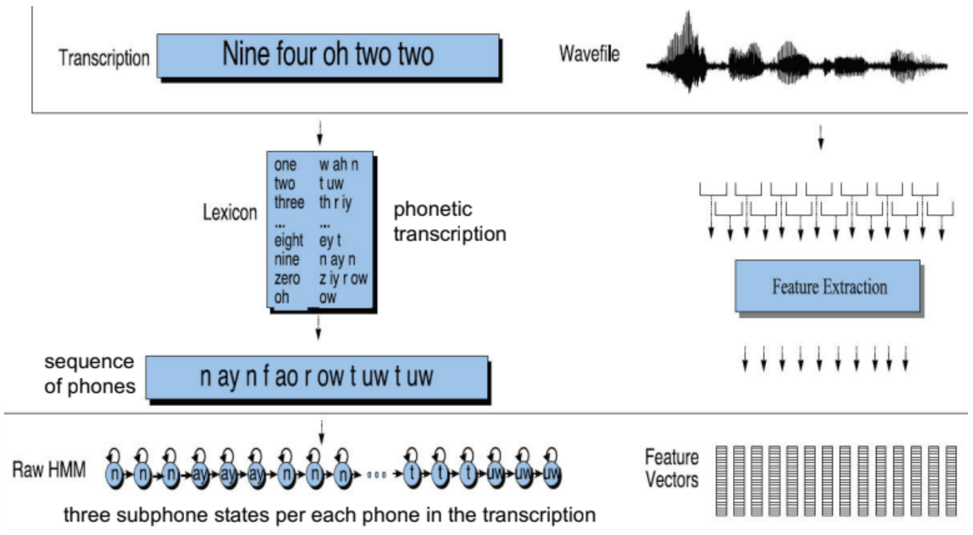
Each phone is represented with 5 states:

- 3 emitting (beginning, middle, final) subphone states
- 2 non-emitting (start, end) states



Dividing up a phone in this way captures the intuition that the significant changes in the acoustic input happen at a finer granularity than the phone; for example the closure and release of a stop consonant. Furthermore, many systems use a separate instance of each of these subphones for each **triphone** context. Thus instead of 60 phone units, there could be as many as 60^3 context-dependent triphones. In practice, many possible sequences of phones never occur or are very rare, so systems create a much smaller number of triphones models by clustering the possible triphones.

17.3.3 Embedded Training



- Given a phoneset, a pronunciation lexicon and the transcribed wavefiles
- Build a "whole sentence" HMM for each sentence
- Initialize A probabilities to 0.5 (for loop-backs or for the correct next subphones) or to zero (for all other transitions)
- Initialize B probabilities: randomly in case of discrete B or by setting the mean and variance for each Gaussian to the global mean and variance for the entire training set in case of GMM
- Run multiple iterations of the **Baum-Welch algorithm**

17.3.4 Decoding the HMM

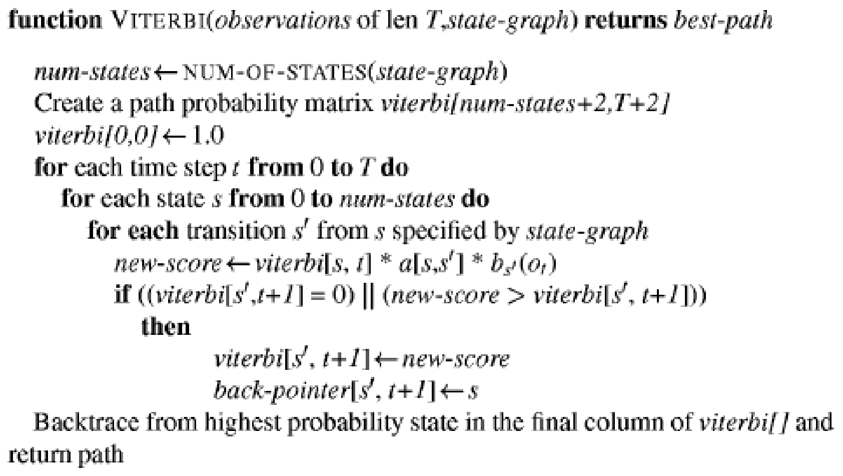
In this final stage, we take a dictionary of word pronunciations and a language model (probabilistic grammar) and use a **Viterbi** or A^* **decoder** to find the sequence of words which has the highest probability given the acoustic events.

Recall the the goal of the Viterbi algorithm is to find the best state sequence $q = (q_1q_2q_3...q_t)$ given the set of observed phones $o = (o_1o_2o_3...o_t)$. More formally, we are searching for the best state sequence $q^* = (q_1q_2...q_T)$, given an observation sequence $o = (o_1, o_2...o_T)$ and a model (a weighted automaton or state graph) λ. Each cell *viterbi[i,t]* of the matrix contains the probability of the best path which accounts for the first t observations and ends in state i of the HMM. This is the most-probable path out of all possible sequences of states of length $t-1$:

$$viterbi[t,i] = max_{q_1,q_2,...,q_{t-1}} P(q_1 q_2 ... q_{t-1}, q_t = i, o_1, o_2 ... o_t | \lambda)$$

In order to compute *viterbi[t,i]*, the Viterbi algorithm assumes the **dynamic programming invariant**. This is the simplifying (but incorrect) assumption that if the ultimate best path for the entire observation sequence happens to go through a state q_i, this best path must include the best path up to and including state q_i. This doesn't mean that the best path at any time t is the best path for the whole sentence. A path can look bad at the beginning but turn out to be the best path. The reason for making the Viterbi assumption is that it allows us to break down the computation of the optimal path probability in a simple way; each of the best paths at time t is the best extension of each of the paths ending at time $t-1$. In other words, the recurrence relation for the best path at time t ending in state j, *viterbi[t,j]*, is the maximum of the possible extensions of every possible previous path from time $t-1$ to time t:

$$viterbi[t,j] = max_i(viterbi[t-1,i]a_{ij})b_j(o_t)$$



```
function VITERBI(observations of len T,state-graph) returns best-path
  num-states ← NUM-OF-STATES(state-graph)
  Create a path probability matrix viterbi[num-states+2,T+2]
  viterbi[0,0] ← 1.0
  for each time step t from 0 to T do
    for each state s from 0 to num-states do
      for each transition s' from s specified by state-graph
        new-score ← viterbi[s, t] * a[s,s'] * $b_{s'}(o_t)$
        if ((viterbi[s',t+1] = 0) || (new-score > viterbi[s', t+1]))
          then
                viterbi[s', t+1] ← new-score
                back-pointer[s', t+1] ← s
  Backtrace from highest probability state in the final column of viterbi[] and
return path
```

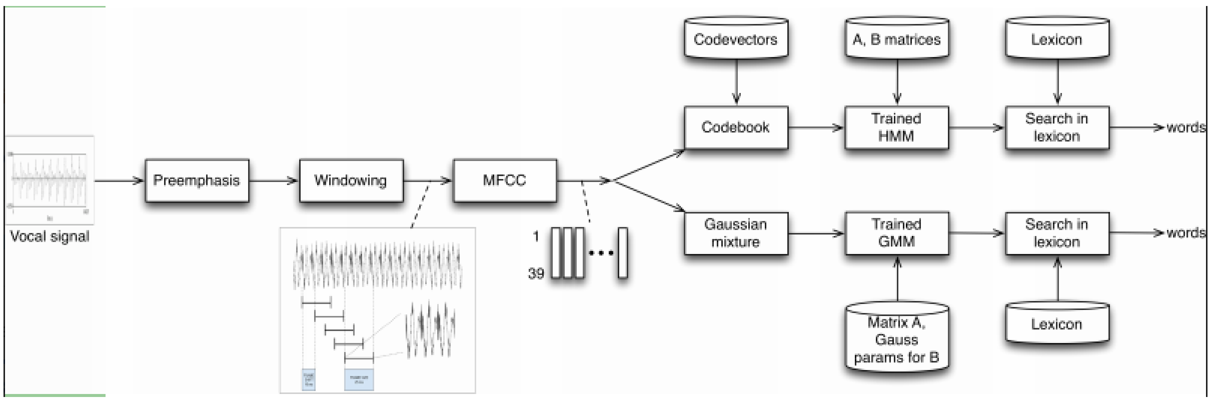
Disclaimer: *much more on the Viterbi algorithm (and also A* decoder) can be found in the book.*

What is important to say before concluding this part is that actual implementations of Viterbi decoding are more complex in two key ways. First, in an actual HMM for speech recognition, the input would not be phones, but the **feature vector** of spectral and acoustic features we talked about. Thus, as we have seen, the **observation likelihood probabilities** $b_i(t)$ of an observation o_t given a state i will not simply take on the values 0 or 1, but will be more fine-grained probability estimates, computed via mixture of Gaussian probability estimators (or neural nets).
Second, in practice in large-vocabulary recognition it is too expensive to consider all possible words when the algorithm is extending paths from one state-column to the next. Instead, low-probability paths are pruned at each time step and not extended to the next state column. This is usually implemented via **beam search**: for each state column (time step), the algorithm maintains a short list of high-probability words whose path probabilities are within some percentage (beam width) of the most probable word path. Only transitions from these words are extended when moving to the next time step.

17.3.5 Advanced Methods for Decoding

There are two main limitations of the Viterbi decoder. First, the Viterbi decoder does not actually compute the sequence of words which is most probable given the input acoustics. Instead, it computes an approximation to this: the sequence of states (i.e. phones or subphones) which is most probable given the input. This difference may not always be important; the most probable sequence of phones may very well correspond exactly to the most probable sequence of words. But sometimes this does not happen. Consider for example a speech recognition system whose lexicon has multiple pronunciation for each word. Suppose the correct word sequence includes a word with very many pronunciations. Since the probabilities leaving the start arc of each word must sum to 1.0, each of these pronunciation-paths through this multiple-pronunciation HMM word model will have a smaller probability than the path through a word with only a single pronunciation path.
A second problem with the Viterbi decoder is that it cannot be used with all possible language models. In fact, the Viterbi algorithm as we have defined cannot take complete advantage of any language model more complex than a bigram grammar. This is because of the fact that a trigram grammar, for example, violates the **dynamic programming invariant** that makes dynamic programming algorithms possible.

There are two classes of solutions to these problems. One class involves modifying the Viterbi decoder to return multiple potential utterances and then using other high-level language model or pronunciation-modeling algorithms to re-rank these multiple outputs (**multiple-pass decoding**).
The second solution to the problems with Viterbi decoding is to employ a completely different decoding algorithm. The most common alternative is the **stack decoder**, also called the A^* **decoder**, that we've already mentioned.



Other advanced techniques that can be used to improve the ASR's results are:

- Spectral subtraction: to deal with **additive noise** (external sound source that is relatively constant)
- Cepstral mean normalization: to deal with **convolutional noise** (introduced by channel characteristics like different microphones)
- Vocal tract length normalization: warping the frequency axis of the speech power spectrum to account for the fact that the precise locations of vocal-tract resonances vary roughly monotonically with the physical size of the speaker
- Pitch normalization: MFCC does not make use of the pitch, but it can be distorted by

the pitch, hence normalization could improve accuracy for children (not easy)

- Short non-verbal sounds (coughs, loud breathing...) $\rightarrow$ filled pauses
- Environmental sounds
- Modify the acoustic model: there are several methods, for example MLLR (Maximum Likelihood Linear Regression) for GMM

17.4 Evaluation

The standard evaluation metric for speech recognition systems is the **word error rate**. The word error rate is based on how much the word string returned by the recognizer differs from a correct or reference transcription. Given such a correct transcription, the first step in computing word errors is to compute the **minimum edit distance** in words between the hypothesized and correct strings. The word error rate is then defined as follows (note that because the equation includes insertions, the error rate can be great than 100%):

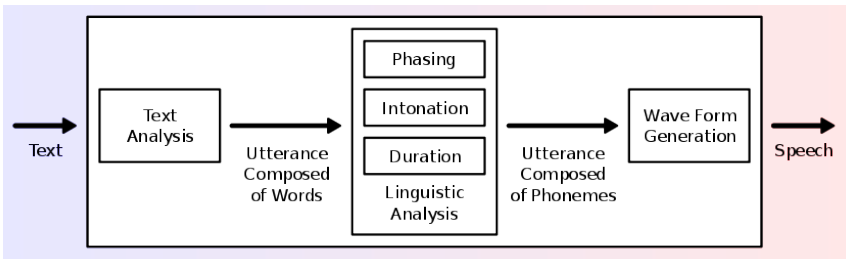
$$WER = 100 \cdot \frac{Insertions + Substitutions + Deletions}{\text{Total Words in Correct Transcripts}}$$

There is also the **Sentence Error Rate**:

$$SER = 100 \cdot \frac{\text{Number of sentences with at least one error}}{\text{Total Number of Sentences}}$$

18 Text-To-Speech

A text-to-speech (TTS) system converts normal language text into speech:



There are different approaches in which we can perform waveform synthesis:

- Formant analysis: rules/filters are used to create speech using additive synthesis and an acoustic model (physical modelling synthesis). The drawback of this approach is a resulting “robotic” voice.
- Articulatory synthesis: modelling and simulating movements of articulators and acoustics of vocal tract. Very complex approach.
- Concatenative synthesis: this is the most used approach in modern TTS.

18.1 Concatenative TTS

Concatenative synthesis is based on a database of speech that has been recorded by a single speaker. This database is then segmented into a number of short units, which can be phones, diphones, syllables, words or other units. By selecting units appropriately, we can generate a series of units which match the phone sequence in the input. By using signal processing to smooth joins at the unit edges, we can simply concatenate the waveforms for each of these units to form a single synthetic speech waveform.

However, before performing this unit selection, another important process is done: **text analysis**. Text analysis consists of going from text to phonemes and it can be described with the following steps:

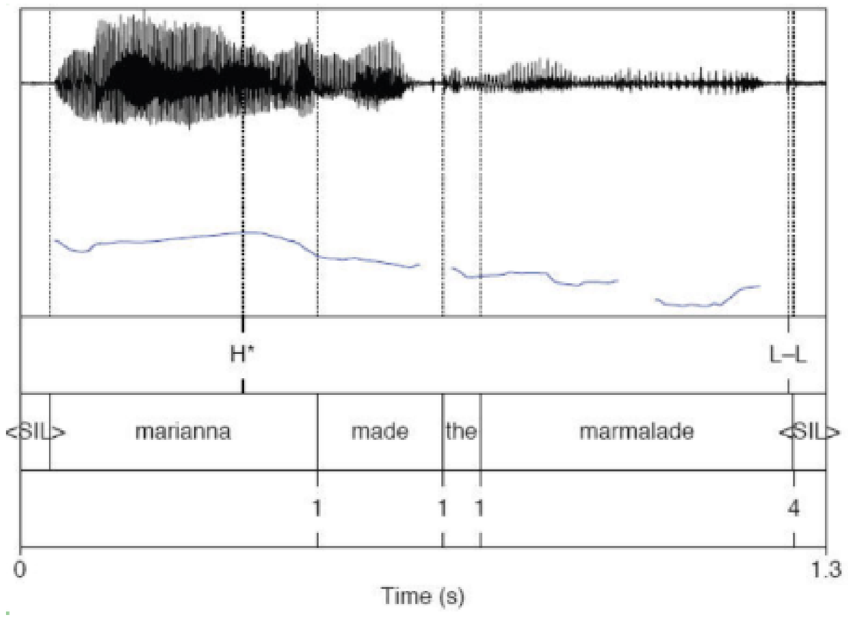
- Text Normalization: in this phase sentence tokenization is performed (often relying on machine learning), non-standard words are handled (not an easy task, since NSW can be ambiguous, e.g. how to pronunce 1970? Is it a date, a price or a string?) and homograph disambiguation is done (e.g. *bass* as fish or as instrument).
- Phonetic Analysis: it transforms a word into a list of phonemes. There are two general approaches:
  - Dictionary based: collections of pronunciations, for each word
  - Grapheme-to-phoneme conversion: for words non present in the dictionary (mostly names)

  For transparent languages, like Italian, pronunciation rules are used.
- Prosodic Analysis: [described below]

Prosodic Analysis

Prosody operates on longer linguistic units than phones, and hence is sometimes called the study of **suprasegmental** phenomena. There are three main phonological aspects to prosody: prominence, structure and tune. Prominence is a broad term used to cover **stress** and **accent**. Sentences have prosodic structure in the sense that some words seem to group naturally together and some words seem to have a noticeable break or disjuncture between them. Often prosodic structure is described in terms of **prosodic phrasing**, meaning that an utterance has a prosodic phrase structure in a similar way to it having a syntactic phrase structure. For example, in the sentence *I wanted to go to London, but could only get tickets for France* there seems to be two main prosodic phrases, their boundary occurring at the comma. These larger prosodic units are commonly called **intonation phrases**, while phrases like *I wanted — to go — to London* are called **intermediate phrases**.
Two utterances with the same prominence and phrasing patterns can still differ prosodically by having different **tunes**. Tune refers to the intonational melody of an utterance. Intonational tunes can be broken into component parts, the most important of which is the **pitch accent**. Pitch accents occur on stressed syllables and form a characteristic pattern in the F0 contour. A popular model of pitch accent classification is the Pierrehumbert or ToBI model, which says there are 5 pitch accents in English, which are made from combining two simple tones (high H, and low L) in various ways. A H+L patter forms a fall, while a L+H pattern forms a rise. An asterisk is also used to indicate which tone falls on the stressed syllable.

So, a major task for a TTS system is to generate appropriate linguistic representations of prosody, and from them generate appropriate acoustics patterns which will be manifested in



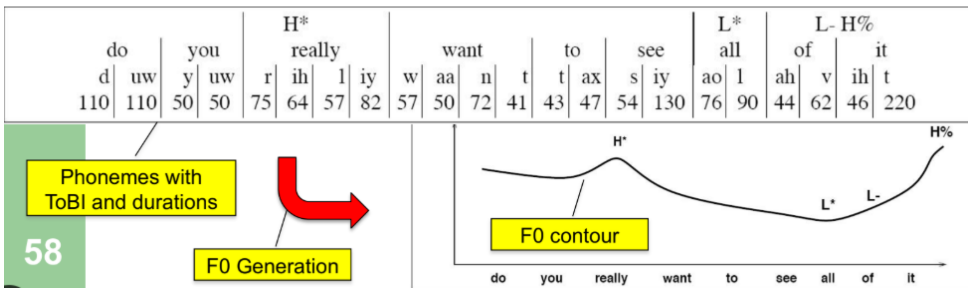
the output speech waveform. The output of a TTS system with such a prosodic component is a sequence of phones, each of which has a duration and an F0 (pitch) value. The duration of each phone is dependent on the phonetic context.

Given the set of N factor weights f_i, the Klatt formula for the duration d of a phone is:

$$d = d_{min} + \prod_{i=1}^{N} f_i \times (\bar{d} - d_{min})$$

The F0 value is influences by the factors discussed above, including the lexical stress, the accented or focused element in the sentence, and the intonational tune of the utterance. From the annotation, F0 is generated in two steps:

- Defining "key" F0 values for pitch accents and breaks
- Generate the F0 contour interpolating such F0 values

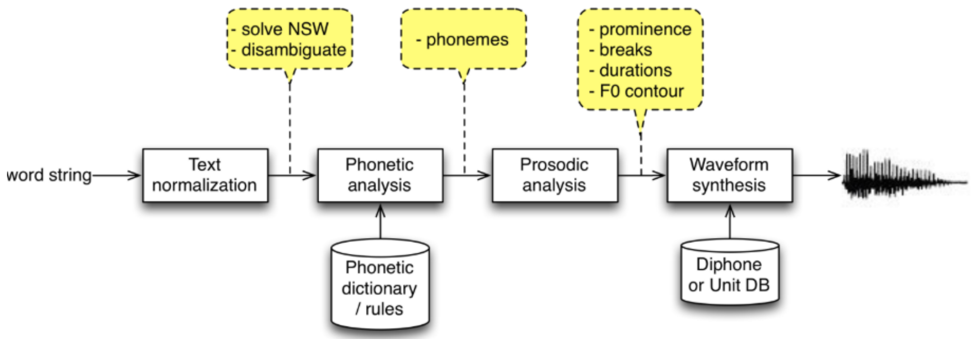


				H*												L*		L- H%			
do		you		really				want				to		see		all		of		it	
d	uw	y	uw	r	ih	l	iy	w	aa	n	t	t	ax	s	iy	ao	l	ah	v	ih	t
110	110	50	50	75	64	57	82	57	50	72	41	43	47	54	130	76	90	44	62	46	220

So, to summarize:

18.2 Evaluation

The evaluation process is usually done by human testers, following these indexes:



- Intelligibility: the ability of the tester to correctly interpret the words and the meaning of the utterance.
- Quality: measure of naturalness, fluency, clarity of the speech.

19 Machine Translation

Machine Translation is a sub-field of computational linguistics that investigates the use of software to translate text or speech from one language to another.

Even if some aspects of the human language seem to be **universal** (every language, for example, seems to have words for referring to people, for talking about women, men, and children, eating and drinking, for being polite or not), many other are not, and this makes MT one of the most difficult task to perform in NLP.
Morphologically, languages are often characterized along two dimensions of variation. The first is the number of morphemes per word. The second dimension is the degree to which morphemes are segmentable.
Synctatically, languages are perhaps most saliently different in the basic word order of verbs, subjects and objects in simple declarative clauses. German, French, English and Mandaring, for example, are all **SVO** languages, meaning that the verb tends to come between the subject and object. Hindi and Japanese, by constrast, are **SOV** languages, while Irish and Classical Arabic are **VSO** languages.
Another important syntactico-morphological distinction is between **head-marking** languages and **dependent-marking** languages. Head-marking languages tend to mark the relation between the head and its dependents on the head, while dependent-marking languages tend to mark the relation on the non-head.
Another distinction can be done by considering if the direction of motion and manner of motion are marked on the verb or on the satellites, e.g. in the following sentence, in English the direction is marked on the particle *out*:

The bottle floated out.

but in Spanish the direction is marked on the verb:

La botella salió flotando.

In addition to such properties that systematically vary across large classes of languages, there are many specific characteristics, more or less unique to single languages. To give an idea of how trivial, yet crucial, these differences can be, think of dates. Dates not only appear in various formats, the calendars themselves may differ, for example dates in Japanese often are relative to the start of the current Emperor's reign rather than to the start of the Christian Era.

Talking about lexical divergences, instead, we can have problems like:

- Word to phrases:
  English: *computer science*
  French: *informatique*

- Part of Speech divergences:
  English: *She likes to sing*
  German: *Sie singt gerne* [She sings likefully]

- Grammatical specificity:
  Italian: plural pronouns have gender (essi/esse)
  English: plural pronouns have no gender (they)

- Semantic specificity:
  English: *brother*
  Mandarin: *gege* (older brother), *didi* (younger brother)

Further, one language may have a **lexical gap**, where no word or phrase, short of an explanatory footnote, can express the meaning of a word in the other language.
Moreover, dependencies on cultural context, as manifest in the background and expectations of the readers of the original and translation, further complicate matters. A number of translation theorists refer to a clever story by Jorge Luis Borges showing that even two linguistic texts with the same words and grammar may have different meanings because of their different cultural contexts. These last points suggest a more general question about cultural differences and the possibility (or impossibility) of translation. A theoretical position sometimes known as **Sapir-Whorf hypothesis** suggests that language may constrain thought, i.e. that the language you speak may affect the way you think. To the extent that this hypothesis is true, there can be no perfect translation, since speakers of the source and target languages necessarily have different conceptual systems. In any case it is clear that the differences between languages run deep, and that the process of translation is not going to be simple.

19.1 Direct Translation

Direct MT systems are built according the idea that MT systems should do as little word as possible. Typically they are built with only one language pair in mind, and the only processing done is that needed to get from one specific source language to one specific target language. A direct MT system is typically composed of several stages, each focused on one type of problem. For example, these are six stages to rewrite a Japanese sentence as an English one:

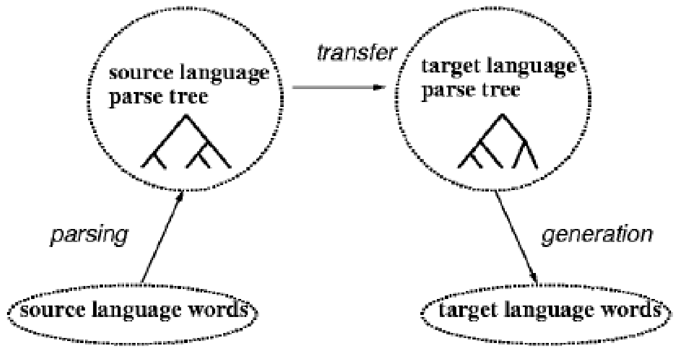
1. morphological analysis: segments the input string into words and does morphological analysis.
2. lexical transfer of content words: chooses translation equivalents for the content words, e.g. using a bilingual dictionary.
3. various work relating to prepositions
4. SVO rearrangements
5. miscellany: handles things like moving case markers before nouns and inserting articles.

6. morphological generation: inflects the verbs.

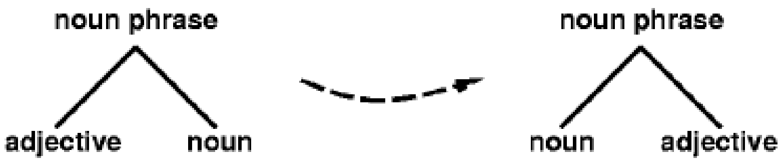
19.2 The Transfer Model

Another strategy is to translate by a process of overcoming these differences, altering the structure of the input to make it conform to the rules of the target language. This can be done by applying **constrastive knowledge**, that is, knowledge about the differences between the two languages. Systems that use this strategy are sometimes said to be based on the **transfer model**.
More precisely, this model involves three phases:

- Analysis: syntactically parse source language
- Transfer: rules to turn this parse into parse for target language
- Generation: generate target sentence from parse tree



Let's consider an example we mentioned before: in English the unmarked order in a noun-phrase had adjectives precede nouns, while in French adjectives follow nouns (even if there are exceptions to this generalization). The following scheme shows how a MT system based on the transfer approach can use **syntactic transformation** to represent this difference:



19.3 The Interlingua Idea: Using Meaning

One problem with the transfer model is that it requires a distinct set of transfer rules for each pair of languages. An alternative approach is to treat translation as a process of extracting the meaning of the input and then expressing that meaning in the target language. If this can be done, a MT system can do without contrastive knowledge, merely relying on the same syntactic and semantic rules used by a standard interpreter and generator for the language.
This scheme presupposes the existence of a meaning representation, or **interlingua**, in a language-independent canonical form. The idea is for the interlingua to represent all sentences that mean the same thing in the same way, regardless of the language they happen to be in.

Translation in this model proceeds by performing a semantic analysis on the input from language X into the interlingual representation and generating from the interlingua to language Y. A frequently used element in interlingual representations is the notion of a small fixed set of thematic roles (we already discussed about this). When used in an interlingua, these thematic roles are taken to be language universals.

19.4 Statistical Techniques

There is another way to look at the problem of machine translation, which consists on focusing on the result, not the process. Taking this perspective, let's consider what it means for a sentence to be a translation of some other sentence. This is an issue to which philosophers of translation have given a lot of thought. The consensus seems to be, sadly, that it is impossible for a sentence in one language to be a translation of a sentence in other, strictly speaking. For example, one cannot really translate Hebrew *adonai roi* ("the Lord is my shepherd") into the language of a culture that has no sheep. We have two options:

- Something **fluent** and understandable, but not faithful:
  The Lord will look after me
- Something **faithful**, but not fluent or natural:
  The Lord is for me like somebody who looks after animals with cotton-like hair

Thus, we have to compromise and this is exactly what translators do in practice: they produce translations that do tolerably well on both criteria. This gives us an intuition: we can model the goal of translation as the production of an output that maximizes some value function that represents the importance of both faithfulness and fluency:

$$\hat{T} = \arg\max_T fluency(T)\ faithfulness(T, S)$$

where T is the target-language sentence and S the source-language sentence. As we did at the beginning of this file, we can apply the noisy channel model so that we can think of the input we must translate as a corrupted version of some target language sentence, and our task is to discover that target language sentence:

$$\hat{T} = \arg\max_T P(T)P(S|T)$$

To implement this, we need to quantify fluency $P(T)$, quantify faithfulness $P(S|T)$ and create an algorithm to find the sequence that maximizes the product of these two things.
Here comes an important difference w.r.t. the previous three approaches. In those models the process is fixed, in that there is no flexibility to trade-off a modicum of faithfulness for a smidgeon of naturalness, or conversely, based on the specific input sentence at hand. This new model, sometimes called **statistical model of translation** allows exactly that.

19.4.1 Quantifying Fluency

Fortunately, we already have some useful metrics for how likely a sentence is to be a real English sentence: the language models we saw in previous sections, the N-gram models.

Fluency models can be arbitrarily sophisticated; any technique that can assign a better probability to a target language string is appropriate, including the more sophisticated probabilistic grammars.

19.4.2 Quantifying Faithfulness

Although it is hard to quantify this property, one basic factor often used in metrics for fidelity is the degree to which the words in one sentence are plausible translations of the words of the other. Thus we can approximate the probability of a sentence being a good translation as the product of the probabilities that each target language word is an appropriate translation of some source language word. For this we need to know, for every source language word, the probability of it mapping to each possible target language word.
Where do we get these probabilities? They can be computed from bilingual corpora, however this is not trivial, since bilingual corpora do not come with annotations specifying which word maps to which. Solving this problem requires first solving the problem of **sentence alignment** in a bilingual corpus. The second problem, **word alignment**, that is, determining which word(s) of the target correspond to each source language word or phrase, is rather more difficult, and is often addressed with EM methods.

19.4.3 IBM Alignment Models

IBM alignment models are a sequence of increasingly complex models used in statistical machine translation to train a translation model and an alignment model, starting with lexical translation probabilities and moving to reordering and word duplication.

- Model 1: lexical translation
- Model 2: additional absolute alignment model
- Model 3: extra fertility model
- Model 4: added relative alignment model
- Model 5: fixed deficiency problem.
- Model 6: Model 4 combined with a HMM alignment model in a log linear way

IBM Model 1 is weak in terms of conducting reordering or adding and dropping words. In most cases, words that follow each other in one language would have a different order after translation, but IBM Model 1 treats all kinds of reordering as equally possible.
Another problem while aligning is the fertility (the notion that input words would produce a specific number of output words after translation). In most cases one input word will be translated into one single word, but some words will produce multiple words or even get dropped (produce no words at all). The fertility of word models addresses this aspect of translation. While adding additional components increases the complexity of models, the main principles of IBM Model 1 are constant.

IBM Model 1

Reference: `http://mt-class.org/jhu/slides/lecture-ibm-model1.pdf`

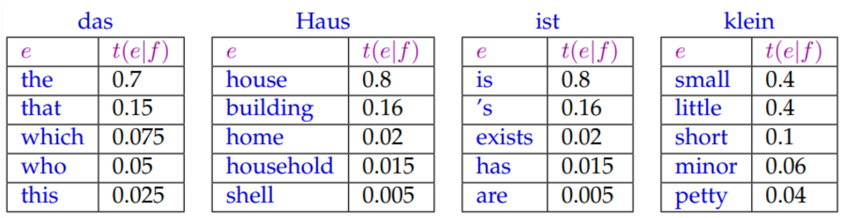
Translation probability:

- for a foreign sentence $\boldsymbol{f} = (f_1, ..., f_{l_f})$ of length l_f
- to an English sentence $\boldsymbol{e} = (e_1, ..., e_{l_e}$ of length l_e
- with an alignment of each English word e_j to a foreign word f_i according to the alignment function $a : j \to i$

$$p(\boldsymbol{e}, a|\boldsymbol{f}) = \frac{\epsilon}{(l_f+1)^{l_e}} \prod_{j=1}^{l_e} t(e_j|f_{a(j)})$$

- parameter ϵ is a normalization constant

For example:



das	
e	$t(e\|f)$
the	0.7
that	0.15
which	0.075
who	0.05
this	0.025

Haus	
e	$t(e\|f)$
house	0.8
building	0.16
home	0.02
household	0.015
shell	0.005

ist	
e	$t(e\|f)$
is	0.8
’s	0.16
exists	0.02
has	0.015
are	0.005

klein	
e	$t(e\|f)$
small	0.4
little	0.4
short	0.1
minor	0.06
petty	0.04

$$\begin{aligned} p(e,a|f) &= \frac{\epsilon}{4^3} \times t(the|das) \times t(house|Haus) \times t(is|ist) \times t(small|klein) \\ &= \frac{\epsilon}{4^3} \times 0.7 \times 0.8 \times 0.8 \times 0.4 \\ &= 0.0028\epsilon \end{aligned}$$

IBM Model 1 and EM

EM algorithm consists of two steps:

1. Expectation step: apply model to the data
   - parts of the model are hidden (here: alignments)
   - using the model, assign probabilities to possible values
2. Maximization step: estimate model from data
   - take assign values as fact
   - collect counts (weighted by probabilities)
   - estimate model from counts

Iterate these steps until convergence.

Thus, we need to be able to compute:

- Expectation step: probability of assignments
- Maximization step: count collection

Applying the chain rule:

$$p(a|\boldsymbol{e},\boldsymbol{f}) = \frac{p(\boldsymbol{e},a|\boldsymbol{f})}{p(\boldsymbol{e}|\boldsymbol{f})}$$

We already have the formula for $p(\boldsymbol{e},a|\boldsymbol{f})$. We need to compute $p(\boldsymbol{e}|\boldsymbol{f})$:

$$\begin{aligned}
p(\boldsymbol{e}|\boldsymbol{f}) &= \sum_a p(\boldsymbol{e},a|\boldsymbol{f}) \\
&= \sum_{a(1)=0}^{l_f} \cdots \sum_{a(l_e)=0}^{l_f} p(\boldsymbol{e},a|\boldsymbol{f}) \\
&= \sum_{a(1)=0}^{l_f} \cdots \sum_{a(l_e)=0}^{l_f} \frac{\epsilon}{(l_f+1)^{l_e}} \prod_{j=1}^{l_e} t(e_j|f_{a(j)}) \\
&= \frac{\epsilon}{(l_f+1)^{l_e}} \sum_{a(1)=0}^{l_f} \cdots \sum_{a(l_e)=0}^{l_f} \prod_{j=1}^{l_e} t(e_j|f_{a(j)}) \\
&= \frac{\epsilon}{(l_f+1)^{l_e}} \prod_{j=1}^{l_e} \sum_{i=0}^{l_f} t(e_j|f_i)
\end{aligned}$$

Note the trick in the last line, which removes the need for an exponential number of products, making the IBM Model 1 estimation tractable.

Combining what we have:

$$\begin{aligned}
p(a|\boldsymbol{e},\boldsymbol{f}) &= \frac{p(\boldsymbol{e},a|\boldsymbol{f})}{p(\boldsymbol{e}|\boldsymbol{f})} \\
&= \frac{\frac{\epsilon}{(l_f+1)^{l_e}} \prod_{j=1}^{l_e} t(e_j|f_{a(j)})}{\frac{\epsilon}{(l_f+1)^{l_e}} \prod_{j=1}^{l_e} \sum_{i=0}^{l_f} t(e_j|f_i)} \\
&= \prod_{j=1}^{l_e} \frac{t(e_j|f_{a(j)})}{\sum_{i=0}^{l_f} t(e_j|f_i)}
\end{aligned}$$

Now we have to collect counts. Evidence from a sentence pair $\boldsymbol{e},\boldsymbol{f}$ that word e is a translation of word f:

$$\begin{aligned}
c(e|f;\boldsymbol{e},\boldsymbol{f}) &= \sum_a p(a|\boldsymbol{e},\boldsymbol{f}) \sum_{j=1}^{l_e} \delta(e,e_j)\delta(f,f_{a(j)}) \\
&= \frac{t(e|f)}{\sum_{i=0}^{l_f} t(e|f_i)} \sum_{j=1}^{l_e} \delta(e,e_j) \sum_{i=0}^{l_f} \delta(f,f_i)
\end{aligned}$$

After collecting these counts over a corpus, we can estimate the model:

$$t(e|f;\boldsymbol{e},\boldsymbol{f}) = \frac{\sum_{(\boldsymbol{e},\boldsymbol{f})} c(e|f;\boldsymbol{e},\boldsymbol{f})}{\sum_e \sum_{(\boldsymbol{e},\boldsymbol{f})} c(e|f;\boldsymbol{e},\boldsymbol{f})}$$

```
Input: set of sentence pairs (e, f)
Output: translation prob. $t(e|f)$
 1: initialize $t(e|f)$ uniformly
 2: while not converged do
 3:   // initialize
 4:   count($e|f$) = 0 for all $e, f$
 5:   total($f$) = 0 for all $f$
 6:   for all sentence pairs (e,f) do
 7:     // compute normalization
 8:     for all words $e$ in e do
 9:       s-total($e$) = 0
10:       for all words $f$ in f do
11:         s-total($e$) += $t(e|f)$
12:       end for
13:     end for
14:     // collect counts
15:     for all words $e$ in e do
16:       for all words $f$ in f do
17:         count($e|f$) += $\frac{t(e|f)}{\text{s-total}(e)}$
18:         total($f$) += $\frac{t(e|f)}{\text{s-total}(e)}$
19:       end for
20:     end for
21:   end for
22:   // estimate probabilities
23:   for all foreign words $f$ do
24:     for all English words $e$ do
25:       $t(e|f) = \frac{\text{count}(e|f)}{\text{total}(f)}$
26:     end for
27:   end for
28: end while
```

e	f	initial	1st it.	2nd it.	3rd it.	...	final
the	das	0.25	0.5	0.6364	0.7479	...	1
book	das	0.25	0.25	0.1818	0.1208	...	0
house	das	0.25	0.25	0.1818	0.1313	...	0
the	buch	0.25	0.25	0.1818	0.1208	...	0
book	buch	0.25	0.5	0.6364	0.7479	...	1
a	buch	0.25	0.25	0.1818	0.1313	...	0
book	ein	0.25	0.5	0.4286	0.3466	...	0
a	ein	0.25	0.5	0.5714	0.6534	...	1
the	haus	0.25	0.5	0.4286	0.3466	...	0
house	haus	0.25	0.5	0.5714	0.6534	...	1

19.5 Evaluating MT

Various evaluation metrics are used to predict acceptability. Evaluation metrics for MT intended to be used raw include the percentage of sentences translated correctly, or nearly correctly, where correctness depends on both fidelity and fluency. The typical evaluation metric for MT output to be post-edited is **edit cost**, either relative to some standard translation via some automatic measure of edit-distance, or measured directly as the amount of time required to correct the output to an acceptable level.

BLEU (bilingual evaluation understudy) is an algorithm for evaluating the quality of text which has been machine-translated from one natural language to another.
Scores are calculated for individual translated segments—generally sentences—by comparing them with a set of good quality reference translations. Those scores are then averaged over the whole corpus to reach an estimate of the translation's overall quality. Intelligibility or grammatical correctness are not taken into account. BLEU's output is always a number between

0 and 1. This value indicates how similar the candidate text is to the reference texts, with values closer to 1 representing more similar texts.

20 Conclusions

The last week of the course was dedicated to a general overview of the models that are mainly used nowadays in NLP, i.e. Neural Networks. I'm not going to write anything here about that, since it was given only a quick overview and the professor said that during the exam only some generic questions will be asked on this part. By the way, in the 3rd edition of the book there is a specific chapter on these topics, which I highly recommend you.

This is the end. I hope this file could be helpful to anyone interested in NLP. Please, if you notice something wrong, contact me via email: 29mett@gmail.com