EXERCISES 24

NATURAL LANGUAGE PROCESSING

24.1 Language Models

Exercise 24.TXUN

Read the following text once for understanding, and remember as much of it as you can. There will be a test later.

> The procedure is actually quite simple. First you arrange things into different groups. Of course, one pile may be sufficient depending on how much there is to do. If you have to go somewhere else due to lack of facilities that is the next step, otherwise you are pretty well set. It is important not to overdo things. That is, it is better to do too few things at once than too many. In the short run this may not seem important but complications can easily arise. A mistake is expensive as well. At first the whole procedure will seem complicated. Soon, however, it will become just another facet of life. It is difficult to foresee any end to the necessity for this task in the immediate future, but then one can never tell. After the procedure is completed one arranges the material into different groups again. Then they can be put into their appropriate places. Eventually they will be used once more and the whole cycle will have to be repeated. However, this is part of life.

No answer required; just read the passage.

Exercise 24.NGRM

This exercise explores the quality of the n-gram model of language. Find or create a monolingual corpus of 100,000 words or more. Segment it into words, and compute the frequency of each word. How many distinct words are there? Also count frequencies of bigrams (two consecutive words) and trigrams (three consecutive words). Now use those frequencies to generate language: from the unigram, bigram, and trigram models, in turn, generate a 100-word text by making random choices according to the frequency counts. Compare the three generated texts with actual language. Finally, calculate the perplexity of each model.

Code not shown. The distribution of words should fall along a Zipfian distribution: a straight line on a log-log scale. The generated language should be similar to the examples in the chapter.

Exercise 24.SEGM

Write a program to do **segmentation** of words without spaces. Given a string, such as the URL "thelongestlistofthelongeststuffatthelongestdomainnameatlonglast.com," return a list of component words: ["the," "longest," "list," ...]. This task is useful for parsing URLs, for spelling correction when words runtogether, and for languages such as Chinese that do not have spaces between words. It can be solved with a unigram or bigram word model and a dynamic programming algorithm similar to the Viterbi algorithm.

Using a unigram language model, the probability of a segmentation of a string $s_{1:N}$ into k nonempty words $s = w_1 \dots w_k$ is $\prod_{i=1}^{k} P_{lm}(w_i)$ where P_{lm} is the unigram language model. This is not normalized without a distribution over the number of words k, but let's ignore this for now.

To see that we can find the most probable segmentation of a string by dynamic programming, let $p(i)$ be the maximum probability of any segmentation of $s_{i:N}$ into words. Then $p(N+1) = 1$ and

$$p(i) = \max_{j=i,\dots,N} P_{lm}(s_{i:j})\, p(j+1)$$

because any segmentation of $s_{i:N}$ starts with a single word spanning $s_{i:j}$ and a segmentation of the rest of the string $s_{j+1:N}$. Because we are using a unigram model, the optimal segmentation of $s_{j+1:N}$ does not depend on the earlier parts of the string.

Using the techniques of this chapter to form a unigram model accessed by the function `prob_word(word)`, the following Python code solves the above dynamic program to output an optimal segmentation:

```
def segment(text):
    length = len(text)
    max_prob = [0] * (length+1)
    max_prob[length] = 1
    split_idx = [-1] * (length+1)
    for start in range(length,-1,-1):
        for split in range(start+1,length+1):
            p = max_prob[split] * prob_word(text[start:split])
            if p > max_prob[start]:
                max_prob[start] = p
                split_idx[start] = split
    i = 0
    words = []
    while i < length:
        words.append(text[i:split_idx[i]])
        i = split_idx[i]
        if i == -1:
            return None  # for text with zero probability
    return words
```

One difficulty is dealing with unknown words. Should a word have probability zero just because it was not seen in the training corpus? If not, what probability should it have? Simple Laplace smoothing would not work well for this application, because the word consisting of the entire input would get the same probability as all other unknown words. The language

model should assign probabilities to unknown words based on their length. One natural option is to fit an exponential distribution to the words lengths of a corpus. Alternatively, one could learn a distribution over the number of words in a string based on its length, add a $P(k)$ term to the probability of a segmentation, and modify the dynamic program to handle this (i.e., to compute $p(i, k)$ the maximum probability of segmenting $s_{i:N}$ into k words).

Another implementation is discussed at `https://norvig.com/ngrams/ch14.pdf`.

Exercise 24.LPCS

Consider a text corpus consisting of N tokens of d distinct words and the number of times each distinct word w appears is given by x_w. We want to apply a version of Laplace smoothing that estimates a word's probability as:

$$\frac{x_w + \alpha}{N + \alpha d}$$

for some constant α (Laplace recommended $\alpha = 1$, but other values are possible.) In the following problems, assume N is 100,000, d is 10,000 and α is 2.

a. Give both the unsmoothed maximum likelihood probability estimate and the Laplace smoothed estimate of a word that appears 1,000 times in the corpus.

b. Do the same for a word that does not appear at all.

c. You are running a Naive Bayes text classifier with Laplace Smoothing, and you suspect that you are overfitting the data. How would you increase or decrease the parameter α?

d. Could increasing α increase or decrease training set error? Increase or decrease validation set error?

a. Unsmoothed: 1,000/100,000 = 0.01; Smoothed:

b. Unsmoothed: 0; Smoothed: 2/(100,000 + 20,000)

c. Increase α, so that the presence of a word that appears very few times in the corpus has less effect.

d. Increasing α tends to increase training error, because it trusts the training data less. It may or may not increase validation set error; you need to find the value of α that balances overfitting and underfitting, and the given information isn't enough to tell.

Exercise 24.ZIPF

Zipf's law of word distribution states the following: Take a large corpus of text, count the frequency of every word in the corpus, and then rank these frequencies in decreasing order. Let f_I be the Ith largest frequency in this list; that is, f_1 is the frequency of the most common word (usually "the"), f_2 is the frequency of the second most common word, and so on. Zipf's law states that f_I is approximately equal to α/I for some constant α. The law tends to be highly accurate except for very small and very large values of I.

Choose a corpus of at least 20,000 words of online text, and verify Zipf's law experimentally. Define an error measure and find the value of α where Zipf's law best matches your experimental data. Create a log–log graph plotting f_I vs. I and α/I vs. I. (On a log–log graph, the function α/I is a straight line.) In carrying out the experiment, be sure to eliminate any formatting tokens (e.g., HTML tags) and normalize upper and lower case.

Using a corpus of 123,612 words extracted from wikipedia with unique words removed, we selected α to optimize two different measures:

a. Least-squares: $\sum_{I=1}^{N}(\log(\alpha/I) - \log f_I)^2$

b. Weighted least-squares: $\sum_{I=1}^{N}(1/I)(\log(\alpha/I) - \log f_I)^2$

The former measure is the obvious fit of a line to the curve in the log-log domain, resulting in $\alpha = 0.0162$ illustrated in Figure S24.1. This line is distorted to handle the numerous large values of I which don't fit the law. The latter measure assigns lower weight to large values of I to correct for this, resulting in $\alpha = 0.0600$ illustrated in Figure S24.2

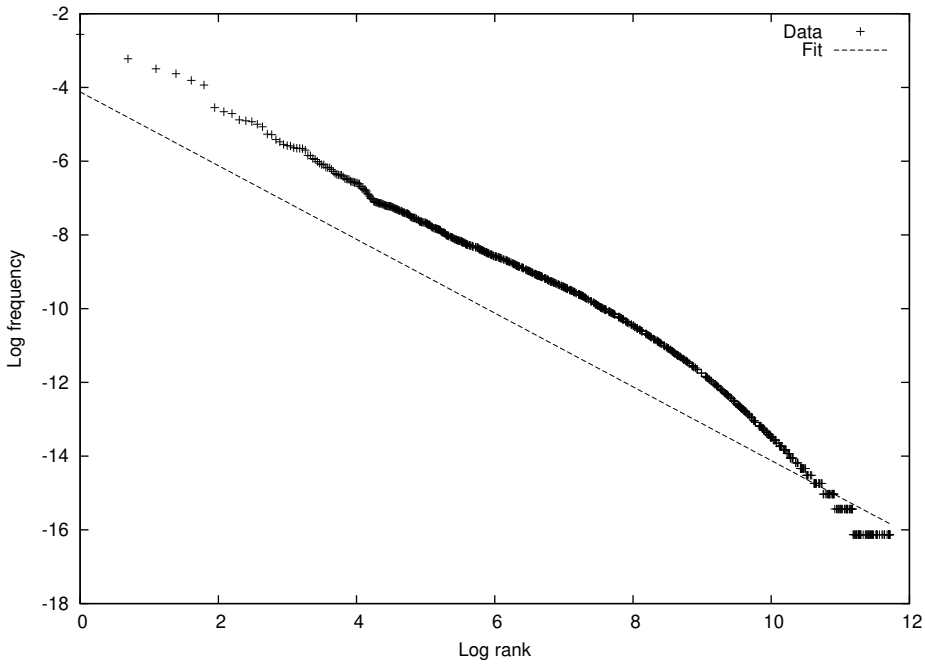


Figure S24.1 Data and least squares fit of α.

Exercise 24.AUTH

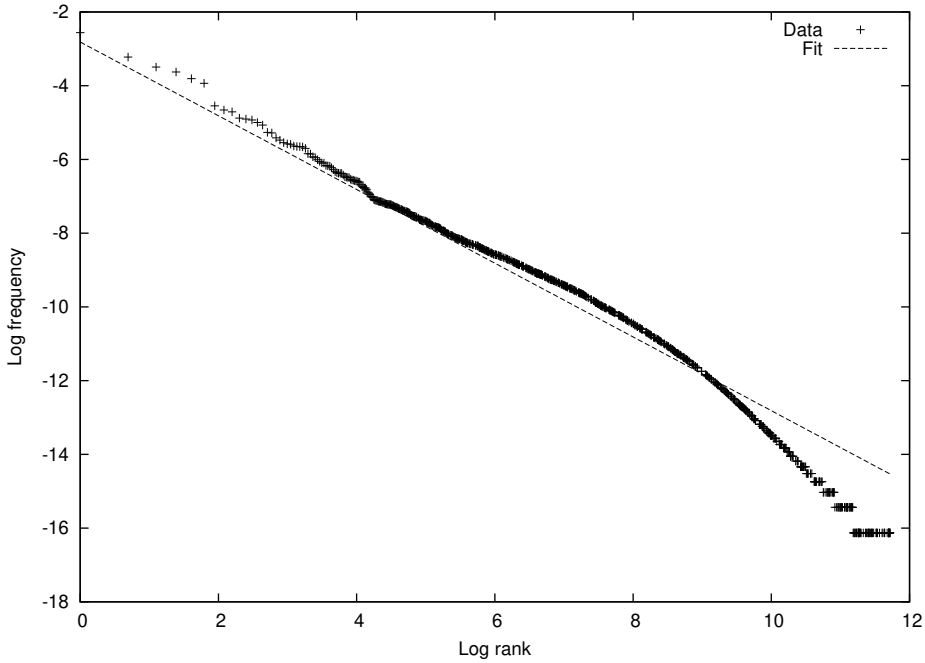


Figure S24.2 Data and weighted least squares fit of α.

In this exercise you will develop a classifier for authorship: given a text, the classifier predicts which of two candidate authors wrote the text. Obtain samples of text from two different authors. Separate them into training and test sets. Now train a language model on the training set. You can choose what features to use; n-grams of words or letters are the easiest, but you can add additional features that you think may help. Then compute the probability of the text under each language model and chose the most probable model. Assess the accuracy of this technique. How does accuracy change as you alter the set of features? This subfield of linguistics is called **stylometry**; its successes include the identification of the author of the disputed *Federalist Papers* and some disputed works of Shakespeare. (Adapted from Jurafsky and Martin (2000).)

Code not shown. The approach suggested here will work in some cases, for authors with distinct vocabularies. For more similar authors, other features such as bigrams, average word and sentence length, parts of speech, and punctuation might help. Accuracy will also depend on how many authors are being distinguished. One interesting way to make the task easier is to group authors into male and female, and try to distinguish the sex of an author not previously seen. This was suggested by the work of Shlomo Argamon.

Exercise 24.SPAM

This exercise concerns the classification of spam e mail. Create a corpus of spam email and one of non-spam mail. Examine each corpus and decide what features appear to be useful for classification: unigram w ords? b igrams? message length, sender, time of a rrival? Then train a classification algorithm (decision tree, naive Bayes, SVM, logistic regression, or some other algorithm of your choosing) on a training set and report its accuracy on a test set.

Code not shown. There are now several open-source projects to do Bayesian spam filtering, so beware if you assign this exercise.

Exercise 24.POSX

Some linguists have argued as follows:

> Children learning a language hear only *positive examples* of the language and no *negative examples*. Therefore, the hypothesis that "every possible sentence is in the language" is consistent with all the observed examples. Moreover, this is the simplest consistent hypothesis. Furthermore, all grammars for languages that are supersets of the true language are also consistent with the observed data. Yet children do induce (more or less) the right grammar. It follows that they begin with very strong innate grammatical constraints that rule out all of these more general hypotheses *a priori*.

Comment briefly on the weak point(s) in this argument, given what you know about statistical learning.

If we consider the simplest kind of maximum likelihood learning, for grammars viewed as generative models, the argument fails. The grammar that accepts every sentence must put a very small probability on each, so the likelihood for such a grammar will be much lower than for the correct grammar. Another way to say this is that if the grammar that accepts every sentence were the true grammar, we would see many more sentences than we actually do!

24.2 Grammar

Exercise 24.GMSL

This exercise concerns grammars for very simple languages.

a. Write a context-free grammar for the language $a^n b^n$.

b. Write a context-free grammar for the palindrome language: the set of all strings whose second half is the reverse of the first half.

c. Write a context-sensitive grammar for the duplicate language: the set of all strings whose second half is the same as the first half.

The purpose of this exercise is to get some experience with simple grammars, and to see how context-sensitive grammars are more complicated than context-free. One approach to writing grammars is to write down the strings of the language in an orderly fashion, and then see how a progression from one string to the next could be created by recursive application of rules. For example:

a. The language $a^n b^n$: The strings are ϵ, ab, $aabb$, ...(where ϵ indicates the null string). Each member of this sequence can be derived from the previous by wrapping an a at the start and a b at the end. Therefore a grammar is:

$$S \rightarrow \epsilon$$
$$S \rightarrow a\ S\ b$$

b. The palindrome language: Let's assume the alphabet is just a, b and c. (In general, the size of the grammar will be proportional to the size of the alphabet. There is no way to write a context-free grammar without specifying the alphabet/lexicon.) The strings of the language include ϵ, *a, b, c, aa, bb, cc, aaa, aba, aca, bab, bbb, bcb, ...*. In general, a string can be formed by bracketing any previous string with two copies of any member of the alphabet. So a grammar is:

$$S \rightarrow \epsilon \mid\ a \mid\ b \mid\ c \mid\ a\ S\ a \mid\ b\ S\ b \mid\ c\ S\ c$$

c. The duplicate language: For the moment, assume that the alphabet is just ab. (It is straightforward to extend to a larger alphabet.) The duplicate language consists of the strings: ϵ, aa, bb, $aaaa$, $abab$, $bbbb$, $baba$, ...Note that all strings are of even length.

One strategy for creating strings in this language is this:

- Start with markers for the front and middle of the string: we can use the non-terminal F for the front and M for the middle. So at this point we have the string FM.
- Generate items at the front of the string: generate an a followed by an A, or a b followed by a B. Eventually we get, say, $FaAaAbBM$. Then we no longer need the F marker and can delete it, leaving $aAaAbBM$.
- Move the non-terminals A and B down the line until just before the M. We end up with $aabAABM$.
- Hop the As and Bs over the M, converting each to a terminal (a or b) as we go. Then we delete the M, and are left with the end result: $aabaab$.

Here is a grammar to implement this strategy:

$S \rightarrow F\ M$	(starting markers)
$F \rightarrow F\ a\ A$	(introduce symbols)
$F \rightarrow F\ b\ B$	
$F \rightarrow \epsilon$	(delete the F marker)
$A\ a \rightarrow a\ A$	(move non-terminals down to the M)
$A\ b \rightarrow b\ A$	
$B\ a \rightarrow a\ B$	
$B\ b \rightarrow b\ B$	
$A\ M \rightarrow M\ a$	(hop over M and convert to terminal)
$B\ M \rightarrow M\ b$	
$M \rightarrow \epsilon$	(delete the M marker)

Here is a trace of the grammar deriving $aabaab$:

$\underline{S}$
$\underline{F}M$
$\underline{F}bBM$
$\underline{F}aAbBM$
$\underline{F}aAaAbBM$
$a\underline{Aa}AbBM$
$aaA\underline{Ab}BM$
$aa\underline{Ab}ABM$
$aabAA\underline{BM}$
$aabA\underline{AM}b$
$aab\underline{AM}ab$
$aab\underline{M}aab$
$aabaab$

Exercise 24.CNFX

In this exercise you will transform $\mathcal{E}_0$ into Chomsky Normal Form (CNF). There are five steps: (a) Add a new start symbol, (b) Eliminate ϵ rules, (c) Eliminate multiple words on right-hand sides, (d) Eliminate rules of the form ($X \rightarrow Y$), (e) Convert long right-hand sides into binary rules.

a. The start symbol, S, can occur only on the left-hand side in CNF. Add a new rule of the form $S' \rightarrow S$, using a new symbol S'.

b. The empty string, ϵ cannot appear on the right-hand side in CNF. $\mathcal{E}_0$ does not have any rules with ϵ, so this is not an issue.

c. A word can appear on the right-hand side in a rule only of the form ($X \rightarrow$ *word*). Replace each rule of the form ($X \rightarrow \ldots$ *word* $\ldots$) with ($X \rightarrow \ldots W' \ldots$) and ($W' \rightarrow$ *word*), using a new symbol W'.

d. A rule ($X \rightarrow Y$) is not allowed in CNF; it must be ($X \rightarrow Y\ Z$) or ($X \rightarrow$ *word*). Replace each rule of the form ($X \rightarrow Y$) with a set of rules of the form ($X \rightarrow \ldots$), one for each rule ($Y \rightarrow \ldots$), where ($\ldots$) indicates one or more symbols.

e. Replace each rule of the form ($X \rightarrow Y\ Z \ldots$) with two rules, ($X \rightarrow Y\ Z'$) and ($Z' \rightarrow Z \ldots$), where Z' is a new symbol.

Show each step of the process and the final set of rules.

The final grammar is shown below. In step **d**, students may be tempted to drop the rules ($Y \rightarrow \ldots$), which fails immediately.

$$
\begin{array}{rcl}
S & \rightarrow & NP\ VP \\
 & | & S'\ Conj\ S' \\
S' & \rightarrow & NP\ VP \\
 & | & SConj\ S' \\
SConj & \rightarrow & S'\ Conj \\
 & & \\
NP & \rightarrow & \textbf{me} \mid \textbf{you} \mid \textbf{I} \mid \textbf{it} \mid \ldots \\
 & | & \textbf{John} \mid \textbf{Mary} \mid \textbf{Boston} \mid \ldots \\
 & | & \textbf{stench} \mid \textbf{breeze} \mid \textbf{wumpus} \mid \textbf{pits} \mid \ldots \\
 & | & Article\ Noun \\
 & | & ArticleAdjs\ Noun \\
 & | & Digit\ Digit \\
 & | & NP\ PP \\
 & | & NP\ RelClause \\
ArticleAdjs & \rightarrow & Article\ Adjs \\
 & & \\
VP & \rightarrow & \textbf{is} \mid \textbf{feel} \mid \textbf{smells} \mid \textbf{stinks} \mid \ldots \\
 & | & VP\ NP \\
 & | & VP\ Adjective \\
 & | & VP\ PP \\
 & | & VP\ Adverb \\
 & & \\
Adjs & \rightarrow & \textbf{right} \mid \textbf{dead} \mid \textbf{smelly} \mid \textbf{breezy} \ldots \\
 & | & Adjective\ Adjs \\
PP & \rightarrow & Prep\ NP \\
RelClause & \rightarrow & RelPro\ VP
\end{array}
$$

Exercise 24.HMMG

An *HMM grammar* is essentially a standard HMM whose state variable is N (nonterminal, with values such as Det, $Adjective$, $Noun$ and so on) and whose evidence variable is

W (word, with values such as is, $duck$, and so on). The HMM model includes a prior $\mathbf{P}(N_0)$, a transition model $\mathbf{P}(N_{t+1}|N_t)$, and a sensor model $\mathbf{P}(W_t|N_t)$. Show that every HMM grammar can be written as a PCFG. [Hint: start by thinking about how the HMM prior can be represented by PCFG rules for the sentence symbol. You may find it helpful to illustrate for the particular HMM with values A, B for N and values x, y for W.]

The prior is represented by rules such as

$$P(N_0 = A) : \qquad S \rightarrow A\ S_A$$

where S_A means "rest of sentence after an A." Transitions are represented as, for example,

$$P(N_{t+1} = B \mid N_t = A) : \qquad S_A \rightarrow B\ S_B$$

and the sensor model is just the lexical rules such as

$$P(W_t = \text{is} \mid N_t = A) : \qquad A \rightarrow \text{is}\ .$$

Exercise 24.VPGR

Consider the following PCFG for simple verb phrases:

$0.1 : VP \rightarrow Verb$
$0.2 : VP \rightarrow Copula\ Adjective$
$0.5 : VP \rightarrow Verb\ the\ Noun$
$0.2 : VP \rightarrow VP\ Adverb$
$0.5 : Verb \rightarrow is$
$0.5 : Verb \rightarrow shoots$
$0.8 : Copula \rightarrow is$
$0.2 : Copula \rightarrow seems$
$0.5 : Adjective \rightarrow$ **unwell**
$0.5 : Adjective \rightarrow$ **well**
$0.5 : Adverb \rightarrow$ **well**
$0.5 : Adverb \rightarrow$ **badly**
$0.6 : Noun \rightarrow$ **duck**
$0.4 : Noun \rightarrow$ **well**

a. Which of the following have a nonzero probability as a VP? (i) shoots the duck well well well (ii) seems the well well (iii) shoots the unwell well badly

b. What is the probability of generating "is well well"?

c. What types of ambiguity are exhibited by the phrase in (b)?

d. Given any PCFG, is it possible to calculate the probability that the PCFG generates a

string of exactly 10 words?

a. (i).

b. This has two parses. The first uses $VP \rightarrow VP\ Adverb$, $VP \rightarrow Copula\ Adjective$, $Copula \rightarrow is$, $Adjective \rightarrow$ **well**, $Adverb \rightarrow$ **well**. Its probability is

$$0.2 \times 0.2 \times 0.8 \times 0.5 \times 0.5 = 0.008\ .$$

The second uses $VP \rightarrow VP\ Adverb$ twice, $VP \rightarrow Verb$, $Verb \rightarrow is$, and $Adverb \rightarrow$ **well** twice. Its probability is

$$0.2 \times 0.2 \times 0.1 \times 0.5 \times 0.5 \times 0.5 = 0.0005\ .$$

The total probability is 0.0085.

c. It exhibits both lexical and syntactic ambiguity.

d. True. There can only be finitely many ways to generate the finitely many strings of 10 words.

Exercise 24.NPGR

Consider the following simple PCFG for noun phrases:

$0.6 : NP \rightarrow Det\ AdjString\ Noun$
$0.4 : NP \rightarrow Det\ NounNounCompound$
$0.5 : AdjString \rightarrow Adj\ AdjString$
$0.5 : AdjString \rightarrow \Lambda$
$1.0 : NounNounCompound \rightarrow Noun\ Noun$
$0.8 : Det \rightarrow$ **the**
$0.2 : Det \rightarrow$ **a**
$0.5 : Adj \rightarrow$ **small**
$0.5 : Adj \rightarrow$ **green**
$0.6 : Noun \rightarrow$ **village**
$0.4 : Noun \rightarrow$ **green**

where Λ denotes the empty string.

a. What is the longest NP that can be generated by this grammar? (i) three words (ii) four words (iii) infinitely many words

b. Which of the following have a nonzero probability of being generated as complete NPs? (i) a small green village (ii) a green green green (iii) a small village green

c. What is the probability of generating "the green green"?

d. What types of ambiguity are exhibited by the phrase in (c)?

e. Given any PCFG and any finite word sequence, is it possible to calculate the probability that the sequence was generated by the PCFG?

a. (iii): we can have infinitely long $AdjStrings$, although the probability decreases exponentially with length.

b. (i) and (ii).

c. "The green green" can be generated two ways, so we add the probabilities. If the first green is an adjective, the probability is $.6 \times .5 \times .5 \times .5 \times .5 \times .4 = 0.015$. If the first green is a noun, the probability is $.4 \times .5 \times 1.0 \times .5 \times .5 = 0.05$. So the total probability is 0.065.

d. The phrase "the green green" exhibits lexical ambiguity ("green" has two meanings) and syntactic ambiguity (there are two distinct parses).

e. This simply requires summing over all possible parses, and there are finitely many of them. CFGs are considerably less powerful than Turing machines.

24.3 Parsing

Exercise 24.SOWS

Consider the sentence "Someone walked slowly to the supermarket" and a lexicon consisting of the following words:

$Pronoun \rightarrow$ **someone**	$Verb \rightarrow$ **walked**
$Adv \rightarrow$ **slowly**	$Prep \rightarrow$ **to**
$Article \rightarrow$ **the**	$Noun \rightarrow$ **supermarket**

Which of the following three grammars, combined with the lexicon, generates the given sentence? Show the corresponding parse tree(s).

(A):

$S \rightarrow NP\ \ VP$
$NP \rightarrow Pronoun$
$NP \rightarrow Article\ \ Noun$
$VP \rightarrow VP\ \ PP$
$VP \rightarrow VP\ \ Adv\ \ Adv$
$VP \rightarrow Verb$
$PP \rightarrow Prep\ \ NP$
$NP \rightarrow Noun$

(B):

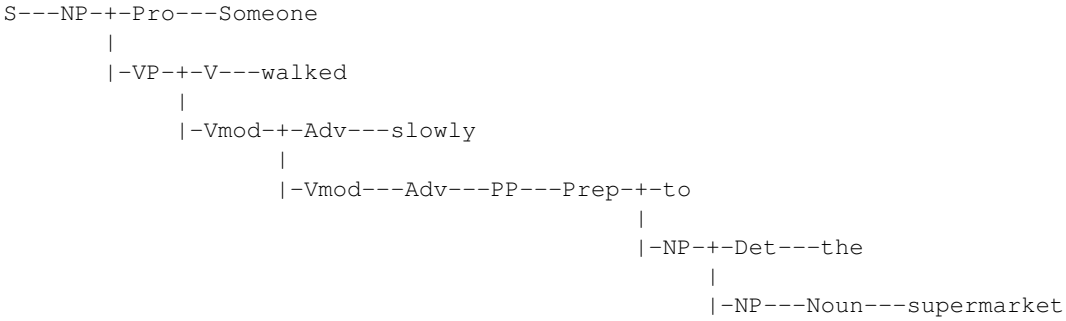
$S \rightarrow NP\ \ VP$
$NP \rightarrow Pronoun$
$NP \rightarrow Noun$
$NP \rightarrow Article\ \ NP$
$VP \rightarrow Verb\ \ Vmod$
$Vmod \rightarrow Adv\ \ Vmod$
$Vmod \rightarrow Adv$
$Adv \rightarrow PP$
$PP \rightarrow Prep\ \ NP$

(C):

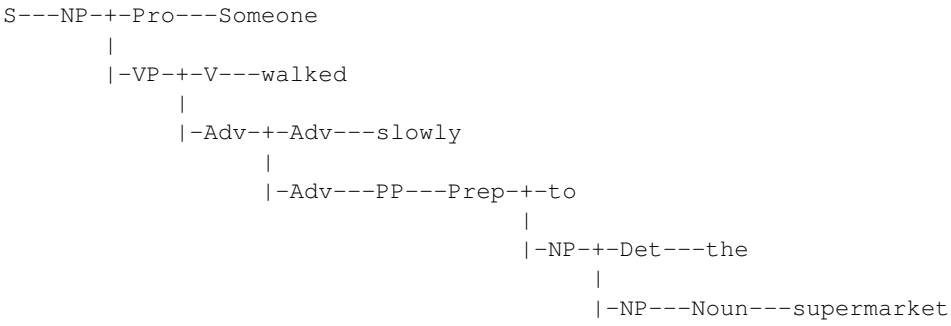
$S \rightarrow NP\ \ VP$
$NP \rightarrow Pronoun$
$NP \rightarrow Article\ \ NP$
$VP \rightarrow Verb\ \ Adv$
$Adv \rightarrow Adv\ \ Adv$
$Adv \rightarrow PP$
$PP \rightarrow Prep\ \ NP$
$NP \rightarrow Noun$

For each of the preceding three grammars, write down three sentences of English and three sentences of non-English generated by the grammar. Each sentence should be significantly different, should be at least six words long, and should include some new lexical entries (which you should define). Suggest ways to improve each grammar to avoid generating the non-English sentences.

Grammar (A) does not work, because there is no way for the verb "walked" followed by the adverb "slowly" and the prepositional phrase "to the supermarket" to be parsed as a verb phrase. A verb phrase in (A) must have either two adverbs or be just a verb. Here is the parse under grammar (B):



Here is the parse under grammar (C):



Exercise 24.TOYG

Consider the following toy grammar:

$S \rightarrow NP\ VP$
$NP \rightarrow Noun$
$NP \rightarrow NP$ **and** NP
$NP \rightarrow NP\ PP$
$VP \rightarrow Verb$
$VP \rightarrow VP$ **and** VP
$VP \rightarrow VP\ PP$
$PP \rightarrow Prep\ NP$

$Noun \rightarrow$ **Sally** | **pools** | **streams** | **swims**
$Prep \rightarrow$ **in**
$Verb \rightarrow$ **pools** | **streams** | **swims**

a. Show all the parse trees in this grammar for the sentence "Sally swims in streams and pools."

b. Show all the table entries that would be made by a (non-probabalistic) CYK parser on this sentence.

a. There is a unique parse. Using the grammar in part two of the answer to have at most binary branching, this is:

```
S---NP-+-Noun---Sally
       |
       |-VP-+-VP---V---swims
            |
            |-PP-+---Prep---in
                 |
                 |-NP-+---NP---Noun---streams
                      |
                      |-ANP-+---A---and
                            |
                            |---NP---Noun---pools
```

b. To run CYK on this sentence we first need to convert the grammar into Chomsky Normal Form. For simplicity we'll allow unary rules $X \rightarrow Y$ which CYK can be adapted to handle. This gives us the grammar:

$S \rightarrow NP\ VP$
$NP \rightarrow N$
$NP \rightarrow NP$ **ANP**
$ANP \rightarrow$ **A** NP
$NP \rightarrow NP\ PP$
$VP \rightarrow V$
$VP \rightarrow VP$ **AVP**
$AVP \rightarrow A\ VP$
$VP \rightarrow VP\ PP$
$PP \rightarrow P\ NP$

$N \rightarrow$ **Sally** | **pools** | **streams** | **swims**
$P \rightarrow$ **in**
$V \rightarrow$ **pools** | **streams** | **swims**
$A \rightarrow$ **and**

Running the algorithm results in the table in Figure S24.3.

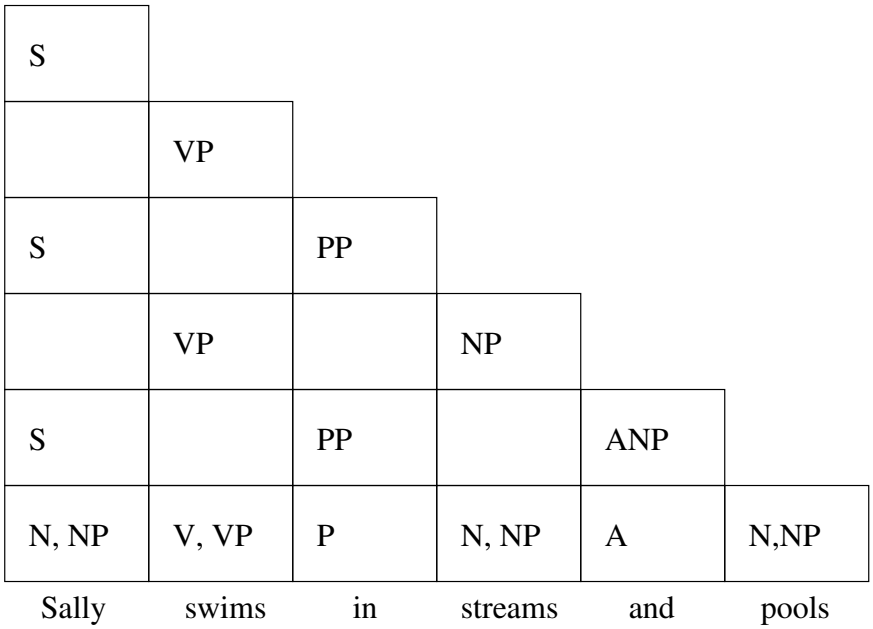


Figure S24.3 CYK parse table.

24.4 Augmented Grammars

Exercise 24.ACFG

An augmented context-free grammar can represent languages that a regular context-free grammar cannot. Show an augmented context-free grammar for the language $a^n b^n c^n$. The allowable values for augmentation variables are 1 and SUCCESSOR(n), where n is a value. The rule for a sentence in this language is

$$S(n) \rightarrow A(n)\ B(n)\ C(n)\ .$$

Show the rule(s) for each of A, B, and C.

The rule for A is

$$A(n') \rightarrow aA(n)\ \{n' = \text{SUCCESSOR}(n)\}$$
$$A(1) \rightarrow a$$

The rules for B and C are similar.

Exercise 24.ENGO

Augment the $\mathcal{E}_1$ grammar so that it handles article–noun agreement. That is, make sure

that "agents" and "an agent" are NPs, but "agent" and "an agents" are not.

$$
\begin{aligned}
&NP(case, number, Third) \rightarrow Name(number)\\
&NP(case, Plural, Third) \rightarrow Noun(Plural)\\
&NP(case, number, Third) \rightarrow Article(number)\, Noun(number)\\
&Article(Singular) \rightarrow \textbf{a} \mid \textbf{an} \mid \textbf{the}\\
&Article(Plural) \rightarrow \textbf{the} \mid \textbf{some} \mid \textbf{many}
\end{aligned}
$$

Exercise 24.DCGN

Using DCG notation, write a grammar for a language that is just like $\mathcal{E}_1$, except that it enforces agreement between the subject and verb of a sentence and thus does not generate ungrammatical sentences such as "I smells the wumpus."

Here is a partial DCG. We include both person and number annotation although English really only differentiates the third person singular for verb agreement (except for the verb *be*).

$$
\begin{aligned}
&S \rightarrow NP(Subjective, number, person)\ VP(number, person) \mid \ldots\\
&NP(case, number, person) \rightarrow Pronoun(case, number, person)\\
&NP(case, number, Third) \rightarrow Name(number) \mid Noun(number) \mid \ldots\\
&VP(number, person) \rightarrow VP(number, person)\ NP(Objective, \_, \_) \mid \ldots\\
&PP \rightarrow Preposition\ NP(Objective, \_, \_)\\
&Pronoun(Subjective, Singular, First) \rightarrow \textbf{I}\\
&Pronoun(Subjective, Singular, Second) \rightarrow \textbf{you}\\
&Pronoun(Subjective, Singular, Third) \rightarrow \textbf{he} \mid \textbf{she} \mid \textbf{it}\\
&Pronoun(Subjective, Plural, First) \rightarrow \textbf{we}\\
&Pronoun(Subjective, Plural, Second) \rightarrow \textbf{you}\\
&Pronoun(Subjective, Plural, Third) \rightarrow \textbf{they}\\
&Pronoun(Objective, Singular, First) \rightarrow \textbf{me}\\
&Pronoun(Objective, Singular, Second) \rightarrow \textbf{you}\\
&Pronoun(Objective, Singular, Third) \rightarrow \textbf{him} \mid \textbf{her} \mid \textbf{it}\\
&Pronoun(Objective, Plural, First) \rightarrow \textbf{us}\\
&Pronoun(Objective, Plural, Second) \rightarrow \textbf{you}\\
&Pronoun(Objective, Plural, Third) \rightarrow \textbf{them}\\
&Verb(Singular, First) \rightarrow \textbf{smell}\\
&Verb(Singular, Second) \rightarrow \textbf{smell}\\
&Verb(Singular, Third) \rightarrow \textbf{smells}\\
&Verb(Plural, \_) \rightarrow \textbf{smell}
\end{aligned}
$$

Exercise 24.PSFG

Consider the following PCFG:

$S \rightarrow NP\ VP$ [1.0]
$NP \rightarrow Noun$ [0.6] | $Pronoun$ [0.4]
$VP \rightarrow Verb\ NP$ [0.8] | $Modal\ Verb$ [0.2]

$Noun \rightarrow$ **can** [0.1] | **fish** [0.3] | ...
$Pronoun \rightarrow$ **I** [0.4] | ...
$Verb \rightarrow$ **can** [0.01] | **fish** [0.1] | ...
$Modal \rightarrow$ **can** [0.3] | ...

The sentence "I can fish" has two parse trees with this grammar. Explain the semantic difference between the two. Show the two trees, their prior probabilities, and their conditional probabilities, given the sentence.

One parse captures the meaning "I am able to fish" and the other "I put fish in cans." Both have the left branch $NP \rightarrow Pronoun \rightarrow$ **I**, which has probability 0.16.

- The first has the right branch $VP \rightarrow Modal\,Verb$ (0.2) with $Modal \rightarrow$ **can** (0.3) and $Verb \rightarrow$ **fish** (0.1), so its prior probability is

$$0.16 \times 0.2 \times 0.3 \times 0.1 = 0.00096\,.$$

- The second has the right branch $VP \rightarrow Verb\,NP$ (0.8) with $Verb \rightarrow$ **can** (0.1) and $NP \rightarrow Noun \rightarrow$ **fish** (0.6×0.3), so its prior probability is

$$0.16 \times 0.8 \times 0.1 \times 0.6 \times 0.3 = 0.002304\,.$$

As these are the only two parses, and the conditional probability of the string given the parse is 1, their conditional probabilities given the string are proportional to their priors and sum to 1: 0.294 and 0.706.

24.5 Complications of Real Natural Language

Exercise 24.TMGM

Collect some examples of time expressions, such as "two o'clock," "midnight," and "12:46." Also think up some examples that are ungrammatical, such as "thirteen o'clock" or "half past two fifteen." Write a grammar for the time language.

Here is a start of a grammar:

```
Time => DigitHour ":" DigitMinute
    | "midnight" | "noon" | "12 midnight" | "12 noon''
    | ClockHour "o'clock"
    | Difference BeforeAfter ExtendedHour
```

```
DigitHour => 0 | 1 | ... | 23
DigitMinute => 1 | 2 | ... | 60
HalfDigitMinute => 1 | 2 | ... | 29
ClockHour => ClockDigitHour | ClockWordHour
ClockDigitHour => 1 | 2 | ... | 12
ClockWordHour => "one" | ... | "twelve"
BeforeAfter =>  "to" | "past" | "before" | "after"
Difference => HalfDigitMinute "minutes" | ShortDifference
ShortDifference => "five" | "ten" | "twenty" | "twenty-five" | "quarter" | "half"
ExtendedHour => ClockHour | "midnight" | "noon"
```

The grammar is not perfect; for example, it allows "ten before six" and "quarter past noon," which are a little odd-sounding, and "half before six," which is not really OK.

Exercise 24.JAVA

Outline the major differences between a programming language such as Java and a natural language such as English, commenting on the "understanding" problem in each case. Think about such things as grammar, syntax, semantics, pragmatics, compositionality, context-dependence, lexical ambiguity, syntactic ambiguity, reference finding (including pronouns), background knowledge, and what it means to "understand" in the first place.

The purpose of this exercise is to get the student thinking about the properties of natural language. There is a wide variety of acceptable answers. Here are ours:

- **Grammar and Syntax** Java: formally defined in a reference book. Grammaticality is crucial; ungrammatical programs are not accepted. English: unknown, never formally defined, constantly changing. Most communication is made with "ungrammatical" utterances. There is a notion of graded acceptability: some utterances are judged slightly ungrammatical or a little odd, while others are clearly right or wrong.
- **Semantics** Java: the semantics of a program is formally defined by the language specification. More pragmatically, one can say that the meaning of a particular program is the JVM code emitted by the compiler. English: no formal semantics, meaning is context dependent.
- **Pragmatics and Context-Dependence** Java: some small parts of a program are left undefined in the language specification, and are dependent on the computer on which the program is run. English: almost everything about an utterance is dependent on the situation of use.
- **Compositionality** Java: almost all compositional. The meaning of "A + B" is clearly derived from the meaning of "A" and the meaning of "B" in isolation. English: some compositional parts, but many non-compositional dependencies.
- **Lexical Ambiguity** Java: a symbol such as "Avg" can be locally ambiguous as it might refer to a variable, a class, or a function. The ambiguity can be resolved simply by checking the declaration; declarations therefore fulfill in a very exact way the role played by background knowledge and grammatical context in English. English: much lexical ambiguity.
- **Syntactic Ambiguity** Java: the syntax of the language resolves ambiguity. For example, in "if (X) if (Y) A; else B;" one might think it is ambiguous whether the "else"

belongs to the first or second "if," but the language is specified so that it always belongs to the second. English: much syntactic ambiguity.

- **Reference** Java: there is a pronoun "this" to refer to the object on which a method was invoked. Other than that, there are no pronouns or other means of indexical reference; no "it," no "that." (Compare this to stack-based languages such as Forth, where the stack pointer operates as a sort of implicit "it.") There is reference by name, however. Note that ambiguities are determined by scope—if there are two or more declarations of the variable "X", then a use of X refers to the one in the innermost scope surrounding the use. English: many techniques for reference.
- **Background Knowledge** Java: none needed to interpret a program, although a local "context" is built up as declarations are processed. English: much needed to do disambiguation.
- **Understanding** Java: understanding a program means translating it to JVM byte code. English: understanding an utterance means (among other things) responding to it appropriately; participating in a dialog (or choosing not to participate, but having the potential ability to do so).

As a follow-up question, you might want to compare different languages, for example: English, Java, Morse code, the SQL database query language, the Postscript document description language, mathematics, etc.

Exercise 24.NYTS

Consider the following sentence (from *The New York Times,* July 28, 2008):

> Banks struggling to recover from multibillion-dollar loans on real estate are curtailing loans to American businesses, depriving even healthy companies of money for expansion and hiring.

a. Which of the words in this sentence are lexically ambiguous?

b. Find two cases of syntactic ambiguity in this sentence (there are more than two.)

c. Give an instance of metaphor in this sentence.

d. Can you find semantic ambiguity?

a. Webster's New Collegiate Dictionary (9th edn.) lists multiple meaning for all these words except "multibillion" and "curtailing".

b. The attachment of all the propositional phrases is ambiguous, e.g. does "from . . . loans" attach to "struggling" or "recover"? Does "of money" attach to "depriving" or "companies"? The coordination of "and hiring" is also ambiguous; is it coordinated with "expansion" or with "curtailing" and "depriving" (using British punctuation).

c. The most clear-cut case is "healthy companies" as an example of HEALTH for IN A GOOD FINANCIAL STATE. Other possible metaphors include "Banks . . . recover" (same metaphor as "healthy"), "banks struggling" (PHYSICAL EFFORT for WORK), and "expansion" (SPATIAL VOLUME for AMOUNT OF ACTIVITY); in these cases, the line between metaphor and polysemy is vague.

24.6 Natural Language Tasks

Exercise 24.SEEN

Create a test set of ten queries, and pose them to two different Web search engines. Evaluate each one for precision at the top 1, 3, and 10 documents. Can you explain the differences between engines?

Doing the evaluation is easy, if a bit tedious (requiring 200 evaluations for the complete 10 documents $\times$ 2 engines $\times$ 10 queries). Explaining the differences is more difficult. Some things to check are whether the good results in one engine are even in the other engines at all (by searching for unique phrases on the page); check whether the results are commercially sponsored, are produced by human editors, or are algorithmically determined by a search ranking algorithm; check whether each engine implements features auch as spelling correction and synonyms.

Exercise 24.WBCO

Estimate how much storage space is necessary for the index to a 100 billion-page corpus of Web pages. Show the assumptions you made.

Computations like this are given in the book *Managing Gigabytes* (Witten *et al.*, 1999). Here's one way of doing the computation: Assume an average page is about 10KB (giving us a 10TB corpus), and that index size is linear in the size of the corpus. Bahle *et al.* (2002) show an index size of about 2GB for a 22GB corpus; so our billion page corpus would have an index of about 1TB.

Exercise 24.REGX

Write a regular expression or a short program to extract company names. Test it on a corpus of business news articles. Report your recall and precision.

Code not shown. The simplest approach is to look for a string of capitalized words, followed by "Inc" or "Co." or "Ltd." or similar markers. A more complex approach is to get a list of company names (e.g. from an online stock service), look for those names as exact matches, and also extract patterns from them. Reporting recall and precision requires a clearly-defined corpus.

Exercise 24.WESE

Consider the problem of trying to evaluate the quality of an Information Retrieval system that returns a ranked list of answers (like most Web search engines). The appropriate measure of quality depends on the presumed model of what the searcher is trying to achieve, and what

strategy she employs. For each of the following models, propose a corresponding numeric measure.

a. The searcher will look at the first twenty answers returned, with the objective of getting as much relevant information as possible.

b. The searcher needs only one relevant document, and will go down the list until she finds the first one.

c. The searcher has a fairly narrow query and is able to examine all the answers retrieved. She wants to be sure that she has seen everything in the document collection that is relevant to her query. (E.g., a lawyer wants to be sure that she has found *all* relevant precedents, and is willing to spend considerable resources on that.)

d. The searcher needs just one document relevant to the query, and can afford to pay a research assistant for an hour's work looking through the results. The assistant can look through 100 retrieved documents in an hour. The assistant will charge the searcher for the full hour regardless of whether he finds it immediately or at the end of the hour.

e. The searcher will look through all the answers. Examining a document has cost $\$A$; finding a relevant document has value $\$B$; failing to find a relevant document has cost $\$C$ for each relevant document not found.

f. The searcher wants to collect as many relevant documents as possible, but needs steady encouragement. She looks through the documents in order. If the documents she has looked at so far are mostly good, she will continue; otherwise, she will stop.

A. Use the precision on the first 20 documents returned.

B. Use the reciprocal rank of the first relevant document. Or just the rank, considered as a cost function (large is bad).

C. Use the recall.

D. Score this as 1 if the first 100 documents retrieved contain at least one relevant to the query and 0 otherwise.

E. Score this as $\$(A(R+I)+BR-NC)$ where R is the number of relevant documents retrieved, I is the number of irrelevant documents retrieved, and C is the number of relevant documents not retrieved.

F. One model would be a probabilistic one, in which, if the user has seen R relevant documents and I irrelevant ones, she will continue searching with probability $p(R, I)$ for some function p, to be specified. The measure of quality is then the expected number of relevant documents examined.

Exercise 24.TRAN

Select five sentences and submit them to an online translation s ervice. Translate them from English to another language and back to English. Rate the resulting sentences for grammaticality and preservation of meaning. Repeat the process; does the second round of iteration give worse results or the same results? Does the choice of intermediate language make a

difference to the quality of the results? If you know a foreign language, look at the translation of one paragraph into that language. Count and describe the errors made, and conjecture why these errors were made.

The main point of this exercise is to show that current translation software is useful but not perfect. The mistakes made are often amusing for students.

Exercise 24.TXUT

Without looking back at Exercise 24.TXUN, answer the following questions:

a. What are the four steps that are mentioned?
b. What step is left out?
c. What is "the material" that is mentioned in the text?
d. What kind of mistake would be expensive?
e. Is it better to do too few things or too many? Why?

This is a very difficult exercise—most readers have no idea how to answer the questions (except perhaps to remember that "too few" is better than "too many"). This is the whole point of the exercise, as we will see in Exercise 24.TXUZ.

Exercise 24.TXUZ

We forgot to mention that the text in Exercise 24.TXUN is entitled "Washing Clothes." Reread the text and answer the questions again:

a. What are the four steps that are mentioned?
b. What step is left out?
c. What is "the material" that is mentioned in the text?
d. What kind of mistake would be expensive?
e. Is it better to do too few things or too many? Why?

Did you do better this time? Bransford and Johnson (1973) used this text in a controlled experiment and found that the title helped significantly. What does this tell you about how language and memory works?

Now we can answer the difficult questions of 22.7:

- The steps are sorting the clothes into piles (e.g., white vs. colored); going to the washing machine (optional); taking the clothes out and sorting into piles (e.g., socks versus shirts); putting the piles away in the closet or bureau.
- The actual running of the washing machine is never explicitly mentioned, so that is one possible answer. One could also say that drying the clothes is a missing step.
- The material is clothes and perhaps other washables.

- Putting too many clothes together can cause some colors to run onto other clothes and ruin them.
- It is better to do too few: so they won't run; so they get thoroughly cleaned; so they don't cause the machine to become unbalanced.