

Computational Linguistics

Fall 2008

John Goldsmith

October 13, 2008

Contents

1	Introduction	2
1.1	Basics of information theory	2
1.1.1	Some basic notations	2
1.1.2	Some comments on probability and how to think about it	3
1.1.3	Conditional probability	4
1.1.4	Bayes' rule	5
1.1.5	Sequences of events	6
1.1.6	Stationary models	6
1.1.7	Review of logarithms	7
1.1.8	Cross entropy	7
1.1.9	Mutual Information	9
1.2	Improvement in probability if we start chunking a corpus	9
1.2.1	Improvement by chunking (the case of th)	9
1.2.2	Prose	10
1.3	Encoding	11
1.3.1	Bits	12
1.3.2	Best encoding	12
1.3.3	Prefix codes	13
1.3.4	Shannon's coding discovery	14
1.4	Conditional probability and probabilities of strings.	16
1.5	Bayesian analysis	17
2	Letters: Transitional phone (or letter) models	17
2.1	Language identification	23
2.2	Cross-entropy	23
2.3	Categories: V, C etc.; HMMs	24
2.3.1	Hidden Markov models: finding classes of letters	24
2.3.2	Context vectors	25
2.4	Syllables	25
2.5	Vowel harmony	25
3	Words	25
3.1	Word frequencies; Zipf's Law	25
3.2	Maximize (word-probability/phoneme frequency) over the corpus	26

4	Finding words	26
4.1	Non-probabilistically: <i>Sequitur</i>	28
4.2	Minimum Description Length	29
4.2.1	Brent, de Marcken	29
4.2.2	MDL approaches	29
4.3	Problems with this approach to word discovery	30
5	Morphology: Making a lexicon	31
5.1	General remarks on morphology	31
5.2	Putting phonology into the lexicon	33
5.3	Putting segmentation structure in the lexicon: morphology 1	33
5.4	String Edit Distance	33
5.5	Rich morphologies : morphology 2	33
6	Proto-syntax	33
6.1	Word-based transition models	33
6.2	Syntactic categories, for smoother transitional probabilities	33
6.3	Brill POS tagging	33
7	Prior distributions over grammars: grammar complexity and Universal Turing machines	33
8	Semantics and meaning	33
8.1	Latent semantic indexing	33
8.2	Word sense disambiguation	33

1 Introduction

1.1 Basics of information theory

1.1.1 Some basic notations

Language data and its models.¹ Alphabet: Σ . Includes $\# = \text{'\#'}$. All strings: Σ^* ; Σ^+ . These sets always countable. A subset of Σ^* is a lexicon or a vocabulary.

A corpus or text is a single string in Σ^* that begins and ends with $\#$ and contains no internal $\#\#$. If it contains at least one internal $\#$, it is ‘broken’ (a good thing, not bad). It has word-breaks, boundaries.

A broken corpus gives us a lexicon in a natural way.

Hierarchical lexicons.

Notation: $s \in \Sigma^*$; ‘dog’ $S[1] = \text{'d'}$; ‘#dog#’ $t[1] = \text{'\#'}$.

Counts, frequencies, probabilities. (Virtually) all our models are discrete for now (later we will have probabilities over models with continuous real-valued parameters). $Count[s] = Count_C[s] = [s]_C = [s]$.

We define a probability on a set, either Σ or a subset of Σ^* . E.g., $pr_1 : V \subset \Sigma^* \rightarrow [0, 1]$. $\sum_{r \in V} pr(r) = 1.0$. Two different Σ ’s!

We can use frequencies as our probabilities. If we assume a (natural) ordering of the sets we care about (lexicographic), then we have 1-to-1 mapping between $s \in S$ and $x \in [0, 1]$. Later on, we will look at a different mapping between s

¹Class 2 started here.

and points in $[0,1]$, but this mapping will *assume* knowledge of a distribution: we will consider using the lexicographically shortest number in an interval to serve as the label of that interval. In general, the shorter the interval, the longer will be the shortest number in that interval.

Prob on Σ^* : 0 order, 1st order (freq). Unigram. Bigram frequencies. Stationary model.

Conditional probability: $pr(S[i]|S[i-1])$ is defined as $\frac{pr(S[i,i+1])}{pr(S[i])}$

Probability of string.

1.1.2 Some comments on probability and how to think about it

. For better or worse, there are at least two quite different ways that we have to be able to think about probabilities. One is static, and I'd say more mathematical, and from that perspective, strings just *exist*; they're not generated in any sense. The other is dynamic, and involves imagining a string (such as the strings that we care about: strings of letters, of words, etc.) as being generated by a process that we can define explicitly. Just about anything that can be understood in one way can be understood in the other, and sometimes the ability to think about them in both ways is important.

A word about the phrase *random variable*. In this course, most of the variables we look at (and all of the variables that we are looking at now) are discrete, by which we mean that they are countable, and can be enumerated one by one. (This case greatly simplifies things, compared to the niceties required for talking about cases that are continuous, i.e., not discrete).

Now the vocabulary gets a little confusing (at least, I find it confusing, and I want to avoid as much confusion as possible). A random variable is a function that maps onto a set of outcomes, which can be real numbers, vectors, an alphabet, a vocabulary, etc. Random variables usually have capital letters as their names, like 'P'. Each of their outcomes is associated with a probability (a number in $[0,1]$), and we call that function the *probability mass function* (or *mass function*, simply).

One of the things that's confusing is that people always try to explain this in terms of the other metaphor, the event or choice metaphor. Charniak writes,

Let X be the uncertain outcome of some event. We assume that this event has a finite number of possible outcomes...and denote them by $V(X)$. X is called a *random variable*.... The probability of any particular outcome of an event is the fraction of the trials that this outcome is, in fact, the result. Formally, if x is a possible outcome of X (i.e., $x \in V(X)$) we denote the probability as: $P(X = x)$.

The way I read this, it confuses the notion of probability (which is a mathematical notion, within a model) with frequencies that we read off of reality, and it doesn't really help me.

If we want to talk about sequences of symbols, as we do, like the string s , then we need to describe strings by means of sequences of random variables. Probability in general is interested in sets or groups of random variables, and we're focusing on sets that form a sequence. There is no guarantee that the probability assigned by the family (here, sequence) of random variables to a particular string is assigned to each symbol independently of all of the others; that's only a very special case (it includes our 'unigram' model).

We will do this by naming our successive random variables U_i (where ‘ U ’ stands for ‘utterance’). Such a sequence of random variables assigns a probability (of course, it might be 0) to any string in our alphabet. So the string ‘dog#’ is modeled by: $U_1 = d$ and $U_2 = o$ and $U_3 = g$ and $U_4 = \#$. There are many ways that a probability can be assigned to that set of outcomes. If we select a function pr in which the probabilities are computed independently, so that $pr(s) = \prod pr(U_i)$, then we say that the different random variables are *independent*. That’s true in the unigram model, but not the bigram model.

The alternative way to think about this is as a *process*, where each random variable represents the outcome of a choice, which can be made dependent on the value of other variables. $pr(U_2 = h)$ might be quite different if $U_1 = t$ than if $U_1 = r$. This is what we deal with in most cases, and especially in the bigram case. If we think of random variables as expressing the outcome of an event, then we can associate the choice of U_2 as being made in slightly different ways, depending on our knowledge of U_1 , and this is often and easily represented by means of a finite state automaton diagram.

This is very often the easiest way to think of things (but not always).

My opinion is that the most useful overall mental image to have, as we look at the topics in this course, is that there is a flow from left to right, which marks the advance of time; at each discrete moment in time, the flow spreads apart into more small flowlets, so that at any given moment, there is a small flow whose cross-sectional area corresponds to its probability; all of those cross-sections at a given moment k sum to 1.0; and each corresponds to a string of length k .

We have a *bigram model* (what most current literature calls a 1st order Markov model) when we assign probability to pairs of successive outcomes, e.g., $pr(U_3 = t \& U_4 = h)$ and (in a consistent way) with individual outcomes (e.g., $pr(U_4 = h)$), and then the conditional probability of h , given t , is $\frac{pr(U_3=t \& U_4=h)}{pr(U_3=t)}$.²

1.1.3 Conditional probability

Let’s suppose³ we take the days of a leap year (such as 2008) as the universe about which we want to speak, and we assign a uniform distribution over those 366 elements. Just to remind you, the starting and ending days this year are:

²I will try to stay away from the terminology of 0-order and 1st-order Markov models, because the meaning of those terms has changed over time. Nowadays, a 1st order Markov model is one in which two states, the current state and the previous state, condition the transition and/or emission probabilities, while earlier on, the term was used to describe Markov models sensitive only to frequencies—i.e., what today is called a 0-order model. Go figure.

³Class 3 started here.

	Sun	M	Tu	Wed	Th	Fri	Sat
Jan			1		31		
Feb						1, 29	
March		31					1
April			1	30			
May					1		31
June	1	30					
July			1		31		
August	31					1	
Sept		1	30				
Oct				1		31	
Nov	30						1
Dec		1		31			

As you can see, there were 52 weeks plus 2 days in 2008, and the two extra days were a Tuesday and a Wednesday. So the probability of November 22 was $\frac{1}{366}$, the probability of a Monday was $\frac{52}{366}$, the probability of a Tuesday was $\frac{53}{366}$.

So what is the probability that a day of the year 2008 is the 1st of the month, given that it is a Monday? In this particular year, the answer is $\frac{2}{52}$, because we care only about the subpart of the year that are Mondays, and there are exactly two of them, in September and December. The conditional probability, which we have just calculated, is defined in this way: the probability of A, given B, is the probability of A and B, divided by the probability of B. The probability of A and B in this case is the probability that a day is the first and a Monday, so the answer is the ratio of the number of days that are both the 1st and a Monday, divided by the number of Mondays in the year (which is 52). What is the probability that it is the 1st of the month, given that it is a Tuesday? $\frac{2}{53}$.

What is the probability that it is Tuesday, given that it is the first of the month? $\frac{3}{12}$.

1.1.4 Bayes' rule

Simple manipulations of the definition of conditional probability. By definition,

$$pr(A|B) = \frac{pr(A \ \& \ B)}{pr(B)}$$

so

$$pr(A \ \& \ B) = pr(A|B)pr(B).$$

and for the very same reason

$$pr(A \ \& \ B) = pr(B|A)pr(A).$$

Hence

$$pr(A|B)pr(B) = pr(B|A)pr(A)$$

or

$$pr(A|B) = \frac{pr(B|A)pr(A)}{pr(B)}.$$

Things start to get tricky when we think of one of the “events” as a *hypothesis*, because we have to ask what we mean by saying that a hypothesis has a particular probability. That is the heart of bayesian reasoning: a willingness to go *there*.

1.1.5 Sequences of events

A *word* is a string that starts and ends with # and has no internal #'s. Here's how we think about a word w of length 5 (let's say): its probability $pr(w)$ is: $pr(U[1]=\# \text{ and } U[2]=w[2] \text{ and } U[3]=w[3] \text{ and } U[4]=w[4] \text{ and } U[5]=\#)$. That's just the definition of what a word is. By the definition of conditional probability, this is equal to:

$$\frac{pr(U[1] = \# \text{ and } U[2] = w[2] \text{ and } U[3] = w[3] \text{ and } U[4] = w[4] \text{ and } U[5] = \#)}{pr(U[1] = \# \text{ and } U[2] = w[2] \text{ and } U[3] = w[3] \text{ and } U[4] = w[4])} \\ \times pr(U[1] = \# \text{ and } U[2] = w[2] \text{ and } U[3] = w[3] \text{ and } U[4] = w[4])$$

which is to say,

$$pr(w|w[1234]) \times pr(w[1234])$$

That's just a matter of definition, nothing more, although you should be aware that while this notation may seem cleaner, it's actually an abuse of notation, and the more accurate formulation is the longer one above it.

And we can continue this (abusing the notation, though), to get:

$$pr(w|w[1234]) \times pr(w[1234]|w[123]) \times pr(w[123]|w[12]) \times pr(w[12]|w[1]) \times pr(U[1] = \#)$$

The last factor above, the probability that $U[1]$ is #, is 1.

I repeat: so far, all of this is a matter of definition. But we can make assumptions now, such as the unigram model assumption (all probabilities of letters are independent of what precedes them) or the bigram model assumption: the probability of a letter is dependent only on the immediately preceding letter. The latter would be written:

$$pr(U[1] = \#) \times \\ pr(U[2] = w[2] | U[1] = w[1]) \times \\ pr(U[3] = w[3] | U[2] = w[2]) \times \\ pr(U[4] = w[4] | U[3] = w[3]) \times \\ pr(U[5] = \# | U[4] = w[4]) \tag{1}$$

1.1.6 Stationary models

We are thinking of the generation of strings as the concatenation of symbols in the domain of each random variable U_i . Typically, we consider this model to be *stationary*, which means that the values of the parameters do not differ among the various random variables. This is not the same thing as being a unigram model (or anything like that). Consider a simple example of a unigram model which is not stationary. We have a model in which words are generated with alternations of consonants and vowels, but with no dependence between any choices. The odd numbered positions generate only consonants, and the even numbered positions generate only vowels. This is a unigram model (since there are no dependencies between one symbol chosen and another), but the model is not stationary, since the distribution associated with an even-numbered position is different from that of an odd-numbered position.

1.1.7 Review of logarithms

Log probability of string.⁴ $-1 \times \log \text{pr}(x) = \log\left(\frac{1}{\text{pr}(x)}\right) \equiv$ positive log prob (s) \equiv inverse log prob (s) \equiv plog (s). The smaller the plog, the higher the probability.

A review of base 2 logs. Natural logs, base 10 logs. $\ln(x)$ is the natural log (base e) of x . $y = e^{\ln y}$ by definition. What's y 's base 2 log? Since $e = 2^{\log_2 e}$ it follows that $y = (2^{\log_2 e})^{\ln(y)}$, which is $2^{\log_2(e)\ln(y)}$; so $\log_2(y) = \log_2(e)\ln(y)$. This illustrates the more general fact that changing the base of a logarithm consistently just changes our numbers by a constant multiplicative factor. There's only one place where we really care about a special property of natural logs that \log_2 do not have, and that's the fact that $\ln(x) \leq x - 1$, which depends on the derivative of $\ln(x)$ being 1 at $x = 1$ (something that isn't true for logarithms to other bases).

Probability of a string:

$$\prod_{i=1}^{|S|} \text{pr}(S[i]) = \prod_{l \in \Sigma} \text{pr}(l)^{\text{count}_S(l)} = \prod_{l \in \Sigma} \text{pr}(l)^{[l]_S}. \quad (2)$$

So

$$\log \text{pr}(S) = \sum_{l \in \text{lexicon}} [l]_S \log \text{pr}(l). \quad (3)$$

If this looks odd at first, it's just because we're using two systems of accounting: on the left, we're counting down the string, so to speak, whereas on the right, we're counting through the alphabet in question.

Now, if we divide through by the length of our string and multiply by -1, we get the average which is known as Shannon's entropy:

$$\text{entropy}(S) = -\frac{\log \text{pr}(S)}{|S|} = -\sum_{l \in \text{lexicon}} \frac{[l]_S}{|S|} \log(l) = \sum_{l \in \text{lexicon}} \text{freq}(l) \text{plog}(l).$$

Do note that we've dropped the -1 in the last expression because we have switched from logs to plogs.

1.1.8 Cross entropy

[Covered Week 2, class 1): where we keep the empirical frequencies, but vary the distribution whose plog we use to compute the entropy. This is the "cross-entropy" of one distribution to the other (but not symmetrical!). Entropy, or self-entropy, is always smaller than cross-entropy. The cross-entropy of p and q is often written $H(p, q)$ and is defined for the discrete case, i.e., when p and q are defined over the same discrete domain X , as

$$-\sum_{x \in X} p(x) \log q(x)$$

⁴Class 4 started here.

The Kullback-Leibler (KL) divergence between distributions p and q (note that this is not symmetrical!)⁵ is the difference between the cross-entropy and the self-entropy:

$$KL(p||q) = \sum_{x \in \Sigma} p(x) \log \frac{p(x)}{q(x)} \quad (4)$$

You also sometimes see $D_{KL}(p||q)$ used to represent this.

Let's consider -1 times the value of this expression for a moment:

$$\sum_x p(x) \ln \frac{q(x)}{p(x)} \leq \sum_x p(x) \left(\frac{q(x)}{p(x)} - 1 \right) \quad (5)$$

Why? Look at the plot of $\ln(x)$, and compute its first and second derivatives, and its value at (1,0): $\ln(x) \leq x - 1$.

$$= \sum_x p(x) \frac{q(x)}{p(x)} - \sum_x p(x) = 1 - 1 = 0. \quad (6)$$

So $\sum_x p(x) \ln \left(\frac{q(x)}{p(x)} \right) \leq 0$, which is to say, the cross-entropy always exceeds the entropy that isn't cross, when we use natural logs as our base. But we can maintain the inequality when we switch to base 2 logs (which is what we use with plogs), since it just amounts to multiplying both sides by a constant. First we get:

$$\sum_x p(x) \ln q(x) \leq \sum_x p(x) \ln p(x) \quad (7)$$

and then we multiply by -1:

$$\sum_x p(x) \text{plog} p(x) \leq \sum_x p(x) \text{plog} q(x) \quad (8)$$

Instead of changing the direction of the inequality, I left the inequality the same and flipped what was to the left and to the right.

The Kullback-Leibler divergence $D_{KL}(p, q)$ is defined as

$$\sum_x p(x) \ln \frac{p(x)}{q(x)} \quad (9)$$

You see that it's the difference between the cross-entropy and the self-entropy—pay careful attention to the *absence* of a minus before the sum.⁶

As we will see more clearly next class, given a distribution $q(x)$ over an alphabet Σ , we can always construct an encoding of Σ (which is map into $\{0, 1\}$ with the prefix property) in which each symbol is encoded by a string whose length is no longer than the plog of that symbol's property (well, we may have to round up to get an integral number of bits, but eventually we can even get away from that restriction, hard as it may be to believe.)

⁵Actually, if I'm not mistaken—and I'll track down the citation here — KL originally defined KL-divergence as the symmetrized version of this, that is, the average of the (in the modern sense) KL-divergence of p and q and the KL-divergence of q and p .

⁶End of Week 2, Class 1.

An encoding has the prefix property iff there are no two $x, y \in \Sigma$ such that the encoding of x is the encoding of y followed by something else. Examples of encoding that does not have the prefix property: $a \rightarrow 1$; $b \rightarrow 11$; $c \rightarrow 01$. Given an encoding 111, we don't know whether it is an encoding of aaa , ab , or ba . We only want encoding systems with the prefix property, because they parse themselves (so to speak) as we scan them from left to right, and if we already know the encoding system, of course!

1.1.9 Mutual Information

$MI(a, b) = \log \frac{pr(a \& b)}{pr(a)pr(b)}$. We will often care about this when we are considering two successive positions in a string. So it's the difference between the log conditional probability of b , given a (that is, the conditional probability that a position yields letter b , given that the preceding letter is a) minus the log probability of b .

Or, it's $plog(a) + plog(b) - plog(ab)$. The weighted mutual information of a pair is the MI of the pair, multiplied by its count in the corpus.

One of the reasons that MI is important is that it helps us to think about probabilities – even in bigram models – as something like an *extensive* quantity (as the term is used in physics). [Covered Week 2, Class 3.]

1.2 Improvement in probability if we start chunking a corpus

State 1: we compute a unigram probability. State 2: we take all occurrences of th to form an elementary unit.

1.2.1 Improvement by chunking (the case of th)

[Begun Week 2, Class 3; carried over to Week 3, Class 1.]

N_1 is the length of string S : $N_1 = |S|$. $N_2 = N_1 - [th]$.

$$\begin{aligned}
 pr_1(S) &= \prod_l \left(\frac{[l]}{N_1} \right)^{[l]} \\
 pr_2(S) &= \prod_{l \in \Sigma, l \neq t, h} \left(\frac{[l]}{N_2} \right)^{[l]} \left(\frac{[t] - [th]}{N_2} \right)^{[t] - [th]} \left(\frac{[h] - [th]}{N_2} \right)^{[h] - [th]} \left(\frac{[th]}{N_2} \right)^{[th]} \\
 \frac{pr_2(S)}{pr_1(S)} &= \prod_{l \in \Sigma, l \neq t, h} \frac{N^{[l]}}{(N_2)^{[l]}} \left(\frac{fr_2(t)}{fr_1(t)} \right)^{[t] - [th]} \left(\frac{fr_2(h)}{fr_1(h)} \right)^{[h] - [th]} (fr_2(th))^{[th]} \\
 &= \left(\frac{N_1}{N_2} \right)^{N - [t] - [h]} \left(\frac{fr_2(t)}{fr_1(t)} \right)^{[t]} \left(\frac{1}{fr_2(t)} \right)^{[th]} \left(\frac{fr_2(h)}{fr_1(h)} \right)^{[h]} \left(\frac{1}{fr_2(h)} \right)^{[th]} (fr_2(th))^{[th]} \\
 &= \left(\frac{N_1}{N_2} \right)^{N - [t] - [h]} \left(\frac{fr_2(t)}{fr_1(t)} \right)^{[t]} \left(\frac{fr_2(h)}{fr_1(h)} \right)^{[h]} \left(\frac{fr_2(th)}{fr_2(t)fr_2(h)} \right)^{[th]}
 \end{aligned} \tag{10}$$

Taking logs, and using $\Delta F = \frac{E_2}{F_1}$:

$$\Delta S = -(|S| - [t] - [h])\Delta N + [t]\Delta fr(t) + [h]\Delta fr(h) + [th]\log \frac{fr_2(th)}{fr_2(t)fr_2(h)} \quad (11)$$

You should understand what each of these terms *means*, and be able to express in words how each term contributes to the measurement of how the inclusion of [th] in our encoding scheme improves our modeling of the data.

It might be more interesting to compare the counts of t, h, th in the two models rather than their frequency. We could go there directly from the last line above, or start after the second line above:

$$\begin{aligned} \frac{pr_2(S)}{pr_1(S)} &= \frac{N_1^{N_1}}{N_2^{N_2}} \frac{([t] - [th])^{[t]-[th]}}{[t]^{[t]}} \frac{([h] - [th])^{[h]-[th]}}{[h]^{[h]}} [th]^{[th]} \\ &= \frac{N_1^{N_2} N_1^{[th]}}{N_2^{N_2}} \left(\frac{[t] - [th]}{[t]} \right)^{[t]-[th]} \frac{1}{[t]^{[th]}} \left(\frac{[h] - [th]}{[h]} \right)^{[t]-[th]} \frac{1}{[h]^{[th]}} [th]^{[th]} \\ &= \left(\frac{N_1}{N_2} \right)^{N_2} N_1^{[th]} \left(\frac{[t] - [th]}{[t]} \right)^{[t]-[th]} \left(\frac{[h] - [th]}{[h]} \right)^{[t]-[th]} \frac{1}{[t]^{[th]}} \frac{1}{[h]^{[th]}} [th]^{[th]} \\ &= \left(\frac{N_1}{N_2} \right)^{N_2} \left(\frac{[t] - [th]}{[t]} \right)^{[t]-[th]} \left(\frac{[h] - [th]}{[h]} \right)^{[t]-[th]} \frac{N_1^{[th]} [th]^{[th]}}{[t]^{[th]} [h]^{[th]}} \end{aligned} \quad (12)$$

Taking logs, we get

$$-N_2 \log \Delta N + ([t] - [th]) \Delta [t] + ([h] - [th]) \Delta [h] + [th] \left(\log N + \log \frac{[th]}{[t][h]} \right)$$

or, with some obvious notation (for $[t]_2, [h]_2$):

$$-N_2 \log \Delta N + [t]_2 \Delta [t] + [h]_2 \Delta [h] + [th] \left(\log N + \log \frac{[th]}{[t]_1 [h]_1} \right)$$

Questions, etc.:

1. You probably remember that $\ln(1+x) \approx x$ when x is small (and the next term in the approximation is $-\frac{x^2}{2}$). How can we use that to approximate the values in this expression?

2. If we were calculating the conditional probability of the string using a bigram model (i.e., including mutual information between segments, but in this case, just between t and h), how would the log probability differ by comparison to the simple unigram probability? That is, what additional terms come into play with this “chunking” analysis of the data?

1.2.2 Prose

We will assume throughout our discussion that there is an alphabet Σ in which all raw data is expressed; we could take it to be some version of Unicode, for concreteness's sake. By the term *corpus* we mean a subset of Σ^* ; we may refer to it as *data* as well. There is a distinguished symbol in Σ , which we call “space,”

and represent it either as “ ” or as “#”. Some corpora contain “#” while others do not; we say that the first kind indicate word-boundaries, while the second do not. If we obtain S_2 from S_1 by removing all instances of # in S_1 , then we say that S_2 has been obtained by stripping # from S_1 . We can also speak of the natural lexicon of any corpus that indicates word boundaries in the natural way: after affixing a “#” to the beginning and end of each string in the corpus, we define the lexicon as the set of strings consisting of the maximal substrings of the corpus that do not contain “#”.

Information theory is closely related to probability theory and to the theory of encoding. The theory of encoding describes properties of mappings from some universe of formal representations \mathcal{L} (that might be, for example, the set of sentences of a particular language) to a set \mathcal{E} of strings with very restricted properties: \mathcal{E} might be $\{0,1\}^*$, for example.⁷

Most of the time, we will want to restrict our attention to cases where \mathcal{E} has the *prefix property*. We say that a set of strings has the prefix property iff there are no pairs of strings S, T in the set such that S is a proper prefix of T . (A string S is a *prefix* of T if $T = S + X$, where “+” is the concatenation operator.)

The reason for this is that it is especially easy to assign probability distributions over such sets.

It is also easy to see (or it *will* be easy to see) that sets of strings S with the prefix property can be associated with a tree, where each $s \in S$ is associated with a terminal element of T .

We will define the information content of an $s \in S$, where $pr(s) > 0$, as $-\log pr(s) = \log \frac{1}{pr(s)}$.

Terminology: please bear in mind the difference between *counts*, *frequency*, and *probability*. Counts are numbers (initially, integers) that count the number of occurrences of something. Frequencies are counts which have been normalized, so that the sum of frequencies from an appropriate set will sum to 1.0. Probabilities are parameters of a model. A human being creates a model, and has the privilege if she chooses, to set the parameters however she likes. She may set them to be the same as frequencies, or related in some other fashion to frequencies, but that is a choice.

1.3 Encoding

We are interested in the properties of binary encoding systems. A binary encoding system is a subset E of $\{0,1\}^*$ and a mapping from E onto a lexicon \mathcal{L} . The reason for our interest is that there are, surprisingly enough, *interesting* relationships between the very simple structure of an element of $\{0,1\}^*$ and things we care more about, like probability.

Why are we interested in encoding, though? Because we are trying to understand *the structure of data*. Data can be short or long, data can be generated by a random process or it can be highly structured. But in real life, we have to work hard to discover the structure that inheres in the data, and we normally think of that structure as coming from the structure of the device that generated the data in the first place.

⁷If you are familiar with probability theory, you might want to know what our measurable sets are. We will restrict our attention to enumerable sets, so all subsets are measurable.

Our goal is to “squeeze” the structure out of the data, and to display it separate from the left over non-structure. This left over non-structure is, as a first approximation, what is called *information*. You may not like that terminology, but at least understand why it has arisen.

Compression (in particular, what is called *lossless* compression) is the heart of this operation. Compression is an operation from one string of symbols to another; the operation must have an inverse; and the expected ratio of the length of the compressed string to the length of the original string is less than 1.

We must be completely explicit with regard to the nature of the compression algorithm. The first reason for this is that we do not wish to fool ourselves: if the compression method is itself longer than the difference in length of the original data and the compressed form of the data, then we have not really compressed anything.

Thus the first question is: can we find a way to compare the “size” of the compression method, on the one hand, with the success we have achieved in extracting structure from the data?

The answer is: Yes. We will measure the size of the compression method in bits, by measuring how complex our compression method is in completely general terms (or as completely general as we can make them). And we will measure the size of the data after we have squeezed the structure out of it; we measure it in bits also—what we measure will be the data’s inverse log probability.

1.3.1 Bits

The *bit* (which etymologically came from *binary digit*) is the basic unit of information. It is the amount of information provided by an answer to an unbiased yes/no question (“unbiased” means we have no basis for preferring one outcome or the other).

If we have 2^n answers to a question Q, and there is no bias to any of them (i.e., the probability of the answers is described by a uniform distribution: they all have the same probability), then an answer to Q contains n bits of information. You can transmit the answer to Q with n bits, and over the long run, no other encoding will be better. (Here you can think of the 20 yes/no questions game.)

What, now, if the probabilities to the answers are not uniformly distributed? Consider the simple case, where the probabilities are all of the form 2^{-ni} : for example, 0.5, 0.25, 0.25; or (see Figure 1) 0.125, 0.125, 0.25, 0.25, 0.25. In such cases, an answer with probability 2^{-l} provides l bits of information.

There is a simple way to make an encoding with the right number of bits in such cases (as we’ll see just below). It is based on the “canonical mapping” from binary strings of length l to certain intervals $[i,j)$ where $0 \leq i < j \leq 1$, and each such string will be mapped to an interval of size 2^{-l} (so $j - i = 2^{-l}$).

1.3.2 Best encoding

We can transform something we already know into something that is—almost—about encodings. We can say this: if we had an encoding in which each symbol l were encoding by a string of bits whose length was exactly equal to its plog (i.e., $-\log_2 pr(l)$), then that encoding would be the best possible encoding, in

the sense that the encoding would be *shorter* than any other. *If* we had an encoding that were closely related to a distribution, we would know something about that encoding. That's the first thing we know.

1.3.3 Prefix codes

Let's switch gears for a moment. *Prefix-encodings* are encoding systems in which there are no two encodings e_1 and e_2 such that e_1 is a prefix of e_2 , i.e., such that $e_2 = e_1x$, for any x .⁸ It is not hard to see that we can organize all the strings in an encoding as a set of points in a binary-branching tree: each string s can be interpreted as a set of instructions for starting from the root, and then moving to the left or to the right at the i^{th} branch point, depending on whether on whether $s[i]$ is 0 or 1. See Figure 1. If all of the elements of our encodings are terminal nodes of such a tree, then (and only then) the encoding has the prefix property. Is that clear?

Let's refer to the strings that begin with a binary point (like a decimal point, but in binary), followed by a string from $\{0, 1\}^*$ as \mathcal{S} . We will not forget that there are many strings in there that have the same value as binary fractions: both .1 and .10 are in there, and they are both names of the number one-half (as are 0.100, 0.1000, ...). There is a canonical mapping Γ from such a string s to an interval in $[0, 1)$, whereby s maps to the interval $[p, p+2^{-|s|})$, where p is the number that s names (s is a string, after all, not a number). This is illustrated in Figure 2. You should be able to see that any prefix code can be sketched in a figure much like this one, and that if we sum up the lengths of the intervals associated with each terminal node, we get 1.0; hence (since an encoding might fail to actually *use* one of the terminal nodes), we can conclude that a prefix-code will always have the property that $\sum_i 2^{-|e_i|} \leq 1.0$. This is known as the Kraft inequality.⁹ There is a natural, and inverse, connection between the length of the encoding and the length of the interval it encodes.

The converse of this statement also holds: suppose you select a set of encoding lengths before you actually determine the encoding. You might do this to see if you can get a short encoding, encoding your messages with very short encodings (which is what encodings are about, after all: we're compressing text, for example). If you don't make too many of them too short, you will be able to come up with a prefix encoding with the lengths you are looking for. The condition for the lengths being long enough is the Kraft inequality at the end of the preceding paragraph: for the encoding of each symbol s , calculate $2^{-\text{length}(\text{encoding}(s))}$. If the sum (over all symbols) is less than 1, then you can find an encoding with those lengths. Thus the Kraft inequality is a necessary and sufficient condition for encoding lengths to be available as encodings with the prefix property.

If you want to know how to do this: sort the lengths of the encodings into increasing (well, non-decreasing) order, so the short encodings come at the beginning (these will correspond to larger probabilities); we call this list of lengths l_1, l_2, \dots, l_n . We will keep track of how much of the unit interval we have used

⁸It seems pretty illogical to call those prefix encodings, or encodings with the prefix property, but that's how people talk. You would think it would be more natural to say that these encodings were *prefix-free*, which is what Li and Vitányi call them.

⁹This discussion is drawn largely from Li and Vitányi, p. 74. Do read this book when you can.

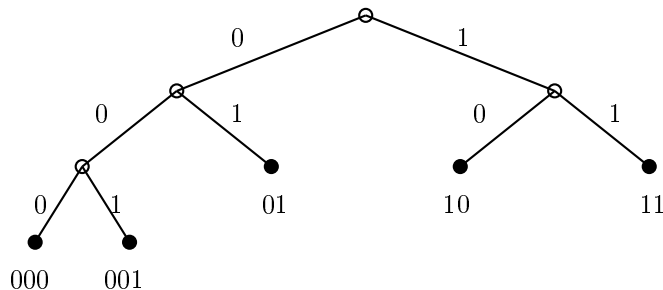


Figure 1: binary tree of encodings with prefix property

up so far; that amount is Q , and it starts at 0. The first length l_1 is associated with the interval $[0, 2^{-l_1})$, so Q is now 2^{-l_1} . And we lay out each of our intervals on the row corresponding to the length of its encoding: so the interval corresponding to a symbol encoding of length 2 is on the second row, an interval corresponding to a symbol encoding of length 4 is on the fourth row, and so on. After the n^{th} interval, we have used up $\sum_{i=1}^n 2^{-l_i}$ of the unit interval. Now we go to the next length on the list of encoding lengths, and that length will be shorter, or at worst of the same length, as the last one; hence each such element of the encoding corresponds to a lower (hence shorter) interval on the grid. You can show yourself that you can find the string s that canonically corresponds to each of these intervals. See Figure 3.

1.3.4 Shannon's coding discovery

Shannon discovered the relationship between the shortest encoding that we can find for messages from a system (under certain conditions) and the entropy of the system (which is the average \log of the symbols, remember): they are the same. The best encoding you can find for a message system takes approximately x bits per symbol, if the x is the entropy of the system.

And yet...the entropy of a message system is a statement about the probabilities of messages, and those probabilities are accessible to us only through the models that we have created to describe the system. It does follow that the better we understand a system, the better will be the compression we will achieve of the messages emitted by the system.

There is another way of looking at this which has led to an approach called arithmetic encoding. Suppose we had simply any partition at all of the unit interval $[0, 1)$, and an association of each of the intervals with an element of the lexical \mathcal{L} . If you think about it for a moment, you realize that in general, the smaller the interval you consider, the less likely it is that you can find a short binary fraction within it, and the longer the interval, the more likely it is that you can. Let's take a further step, and use the canonical association Γ that we mentioned above to associate any string s with an interval $[p, p + 2^{-|s|})$. Then it

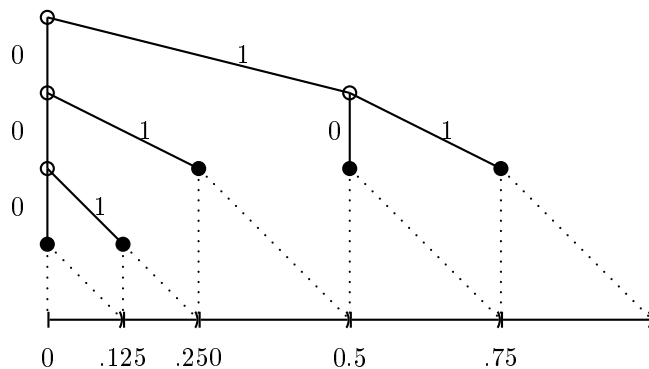


Figure 2: The canonical encoding

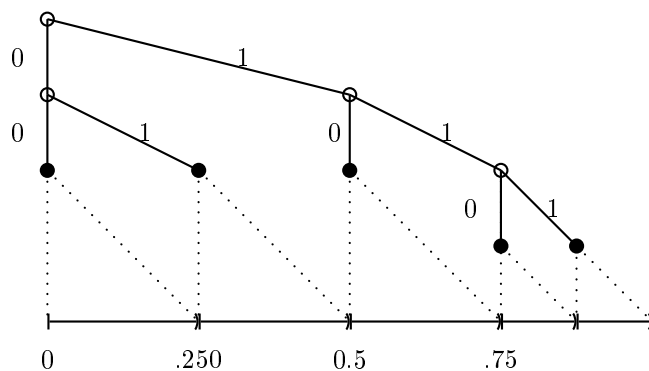


Figure 3: Second half of Kraft inequality

stands to reason that we can employ s as the encoding for a particular interval I if and only if $\Gamma(s)$ lies entirely within I .

You might think that this procedure would waste an extra bit, if you look at Figure 3. But the reason that this approach is so interesting is that with arithmetic encoding, what we calculate in order to perform the encoding is a smaller and smaller interval that corresponds to the encoding of the entire message. So in the end, there's only one extra bit to be paid at the end of the whole message.

This material is well discussed in Li and Vitányi, *An Introduction to Kolmogorov Complexity and its Applications* – the bible of this area. Highly recommended.

What we have seen so far is a close connection between: (1) probability distributions over alphabets; (2) ways of partitioning $[0,1]$; and (3) identifying intervals I in $[0,1]$ with an encoding from B^* whose length is approximately

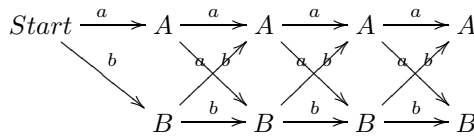
$-\log_2(|I|)$.

1.4 Conditional probability and probabilities of strings.

We care about assigning distributions to Σ^* . We have already observed that if we assign to a string s a value $Uni(s) = \prod pr(s[i])$, then the sum of those values will be 1 if we sum over all the strings of length $|s|$. So we can use this as a probability measure over all strings of a fixed or given length, but certainly not over all of Σ^* !

We have two ways to go. First, we could pick an arbitrary distribution over length λ , and then assign a probability to each string s as $\lambda(|s|) \times Uni(s)$. That would give us a well-formed distribution over all of Σ^* . Second, we could insist that we care not about Σ^* as such, but only about those finite strings in it that end in ‘#’ and have no internal ‘#’s. Let’s call $w = 1 - pr(\#)$; it’s the probability of emitting a real letter. The probability of emitting the null string followed by # is $1 - w$; then the sum of the probabilities of all strings of the form $x\#$ will be $w(1 - w)$. The sum of the probabilities of all strings of the form $xy\#$ will be $w^2(1 - w)$, sum of the probabilities of all strings of the form $xyz\#$ will be $w^3(1 - w)$, and so on. Those numbers sum to 1, so we’re fine: we have a way to assign a distribution, using ‘Uni’, to strings that end in # and have no internal #s.

Conditional probability: we have a sequence of random variables $U_i(i)$, but they typically are not independent. For our purposes, we may think of a variable that is independent of what precedes it as being specified by a single distribution labeled with the relevant alphabet Σ , and one that is *not* independent as one that has several such labeled distributions, and the one that is employed is determined by the outcome of a preceding variable—and in the case that we want to consider (the *bigram* model), it is determined by the outcome of the immediately preceding variable.



Each *state* is associated with a labeled distribution which is illustrated by its arcs leaving to the right; each random variable has as many outcomes as there are states.

$$pr(U(t) = A|U(t - 1) = B) = \frac{pr(U(t) = A \text{ and } U(t - 1) = B)}{pr(U(t - 1) = B)}$$

This makes perfect sense if we think about using frequencies for our parameters. The probability of h , given that we have just seen a t , is then defined as the probability of a th , divided by the probability of a t .

Last thing: mutual information. We say $MI(a,b)$ when we really mean something like $MI(U(i) = a, U(i + 1) = b)$, for example. In such a case, this is defined as $\log \frac{pr(U(i)=a \& U(i+1)=b)}{pr(U(i)=a) \times pr(U(i+1)=b)}$.

You can see that in such a case, $MI(a,b)$ is $\log \frac{pr(a|b)}{pr(a)} = \log \frac{pr(a\&b)}{pr(b) \times pr(a)}$

The upshot of that is simply this: the bigram conditional plog of b , when immediately following a , is equal to b ’s unigram plog less $MI(a,b)$:

$$plog(U(t) = b | U(t-1) = a) = plog(U(t) = b) - MI(U(t-1) = a \& U(t) = b)$$

- Our goal is to find a sequence of increasingly complex grammars, each of which decreases the plog (=information content) of the data.
- A complex system is one whose entropy, given a zero-order model, is very high, and whose entropy continues to steadily decline over a long sequence of increasingly complex grammars. In a complex system, the complexity does not drop very fast, like a stone – it continues to drop gradually as we strip away more and more regularities within the data.

1.5 Bayesian analysis

What is Bayesian analysis? It would not be too far off to say that it is probabilistic analysis that uses Bayes' theorem or rule (which we have seen is just a simple algebraic manipulation of the definition of conditional probability). But in fact there are two related but distinct ideas of what this means for real work. We will be focusing on the second. They both involve the idea of assigning a probability to a hypothesis h , given some evidence.

The first style of bayesian analysis is one in which the probability of a hypothesis is continually updated, giving a growing mass of evidence $\{E_i\}_i$. [to be filled in.]

Bayesian analysis may be defined as the probabilistic analysis of a set of data D (that is, an anaysis which assigns a probability to D) by virtue of selecting a probabilistic model M from a set of possible models \mathcal{M} over which a probability distribution has been defined. That is, we have two entirely different probability distributions at work: we have a distribution over models; we select a specific model m in \mathcal{M} , and ask what probability m assigns to our data D .

(Actually, that is a special case, a simple case, of what most real bayesians would expect from a bayesian analysis. They would expect us to consider not a single model m , but rather a distribution d over models. We'll come back to this. For now, we will stick with the special case.)

2 Letters: Transitional phone (or letter) models

NB: I will use the terms “letter” and “phone” interchangeably here. Let's explore transitional letter models by seeing if we can write an algorithm that will identify the language in which a text is written, based on letter frequencies.

We are given a text T . We will assume that all of our texts are encoded in (some) standard Unicode, and we call that alphabet Σ , so $T \in \Sigma^*$. We are also given a set of texts from several known languages, which we hope are reasonably representative of their respective languages. We will consider three ways to assign probabilities to strings in Σ^* , and ways to draw the respective parameters from those sample texts.

In the terms used originally by Shannon, these would be 0-order, 1st order, and 2nd order models. Unfortunately, terminology has changed over time!

By a 0-order model, Shannon meant a model that assigns a uniform distribution over all of the symbols that are in the alphabet of the model. The usual

way of interpreting that in this context is to say that we infer the alphabet used by a language from our sample: it is the smallest set of symbols Σ such that $S \in \Sigma^*$; and then we assign a uniform distribution over this alphabet. This choice will assign a probability of zero to any string containing one or more symbols not in the original sample.

By a 1st order model, Shannon meant a model in which the probability of a symbol pr_U (where ‘U’ stands for ‘unigram’) is taken to be its frequency in the sample, and the probability of a string is equal to the product of the probabilities of the symbols. (Note that we have to assume a distribution over string length as well, which is typically done by assuming the existence of a special symbol that only appears at the end of strings. We will return to this. For now, we will assume that there is a function $pr_l()$ which is a distribution over the positive integers, and which assigns a probability that a string will be of a given length.)¹⁰

$$pr(s) = pr_l(|s|) \prod_{i=1}^{|s|} pr_U(s[i]) \quad (13)$$

But instead of calling this a 1st order model, we will call this a *unigram* model, since many writers today use the term “1st order model” to mean something else.

By a 2nd order model, Shannon meant a model much like the unigram model, but in which the probability of a symbol was conditioned by the preceding symbol. We will call this a *bigram* model. (Many writers today call this a 1st order Markov model.)

Shannon, in “The mathematical theory of communication,” gave three approximations of English:

Zero-order approximation: XFOML R XKHRJFFJ UJ ALPW XFWJXYJ FF-
JEYVJCQSGHYD QPAAMKBZAACIBZLKJQD

First-order approximation: OCRO HLO RGWR NMIELWIS EU LL NBNE-
SEBYA TH EEI ALHENHTTPA OOBTTVA NAH BRL

Second-order approximation: ON IE ANTSOUTINYS ARE T INCTORE
ST BE S DEAMY ACHIN D ILONASIVE TUCCOOWE AT TEASONARE

¹⁰This kind of model was explored by early cryptographers, notably al-Kindi, who lived from 801-873, and who worked under the Abbasid caliphs in Baghdad, as well as al-Khwarizmi, c. 780-850. Al-Kindi wrote (I have taken this from <http://www.muslimheritage.com/topics/default.cfm?articleID=372>, who quotes Singh, *The Code Book*):

One way to solve an encrypted message, if we know its language, is to find a different plaintext of the same language long enough to fill one sheet or so, and then we count the occurrences of each letter. We call the most frequently occurring letter the ‘first’, the next most occurring letter the ‘second’, the following most occurring the ‘third’, and so on, until we account for all the different letters in the plaintext sample....

Then we look at the cipher text we want to solve and we also classify its symbols. We find the most occurring symbol and change it to the form of the ‘first’ letter of the plaintext sample, the next most common symbol is changed to the form of the ‘second’ letter, and so on, until we account for all symbols of the cryptogram we want to solve.

rank	orthography	phonemes	\log_1	$averageplog_1$
1	a	ə	6.23	3.11
2	an	ə n	10.33	3.44
3	to	t ə	10.40	3.47
4	and	ə n d	15.18	3.80
5	eh	é	6.23	3.88
6	the	ð ə	11.63	3.88
7	can	k ə n	15.60	3.90
8	an	æ n	11.72	3.91
9	Ann	æ n	11.72	3.91
10	in	í n	11.72	3.91
63195	bourgeois	b ʌ r ž w á	50.44	7.21
63196	Ceausescu	č š č é s k ů	64.86	7.21
63197	Peugeot	p y ů ž ó	43.34	7.22
63198	Giraud	ž aÿ r ó	36.19	7.24
63199	Godoy	g á d oÿ	36.35	7.27
63200	geoid	ǰ í oÿ d	37.00	7.40
63201	Cesare	č ě z á r ě	51.80	7.40
63202	Thurgood	θ ʒ g ʌ d	44.86	7.47
63203	Chenoweth	č é n ō w ě θ	52.46	7.49
63204	Qureshey	k ə r é š ě	52.77	7.54

Table 1: Top and bottom of English word list, based solely on unigram frequencies.

FUSO TIZIN ANDY TOBE SEACE CTISBE

Third-order approximation: IN NO IST LAT WHEY CRATICT FROURE
BIRS GROCID PONDENOME OF DEMONSTURES OF THE REPTAGIN
IS REGOACTIONA OF CRE

We'll get to word-based models shortly, but I'll share with you Shannon's approximations of English using a word-based model:

First-order word approximation: REPRESENTING AND SPEED-
ILY IS AN GOOD APT OR COME CAN DIFFERENT NATURAL
HERE HE THE A IN CAME THE TO OF TO EXPERT GRAY
COME TO FURNISHES THE LINE MESSAGE HAD BE THESE

Second-order word approximation THE HEAD AND IN FRONTAL
ATTACK ON AN ENGLISH WRITER THAT THE CHARACTER
OF THIS POINT IS THEREFORE ANOTHER METHOD FOR
THE LETTERS THAT THE TIME OF WHO EVER TOLD THE
PROBLEM FOR AN UNEXPECTED

[Some of this material, especially the tables, is from a paper in progress written by Jason Riggle and John Goldsmith, Information theoretical approaches to phonological structure: the case of vowel harmony.]

rank	phoneme	frequency	plog
1	#	0.20	2.30
2	ə	0.066	3.92
3	n	0.058	4.10
4	t	0.056	4.17
5	s	0.041	4.61
6	r	0.040	4.76
7	d	0.037	4.85
8	l	0.035	4.94
9	k	0.026	5.27
10	æ	0.025	5.31
45	ɔ̃y	0.000 78	10.32
46	ǣ	0.000 69	10.50
47	ž	0.000 54	10.84
48	ǎy	0.000 38	11.36
49	ǎ	0.000 36	11.42
50	ǝ	0.000 28	11.79
51	ě	0.000 14	12.76
52	ǎ	0.000 05	14.30
53	aǎw	0.000 05	14.35
54	ɔ̃y	0.000 02	15.91

Table 2:

rank	orthography	phonemic representation	average plog
1	the	ð ə	1.93
2	hand	h æ n d	2.15
3	and	æ n d	2.20
12640	plumbing	p l ɹ m ɪ ŋ	3.71
12641	aerobatics	é r ə b æ t ɪ k s	3.71
12642	Friday	f r aɪ d i	3.71
25281	tolls	t ó l z	4.01
25282	recorder	r ɪ k ó r d ə	4.01
25283	fives	f aɪ v z	4.01
37922	overburdened	ó v ə b ə d ə n d	4.32
37923	Australians	ɔ̃ s t r é y l y ə n z	4.32
37924	seeps	s i y p s	4.32
50563	retire	r ɪ t aɪ r	4.75
50564	poorer	p ú r ə	4.75
50565	vanished	v æ n ɪ š t	4.75
63,200	eh	é	9.07
63,201	Oahu	ó á h ũ	9.21
63,202	Zhao	ž aǎw	9.25

Table 3: Examples from English word list, ranked by average plog, bigram model

predicted rank	average reported rank	standard deviation	word
1	6.5	0.96	stations
2	4.17	3.02	hounding
3	4.17	2.97	wasting
4	10.2	5.37	dispensing
5	5.3	3.72	gardens
6	5.3	2.62	fumbling
7	15.5	1.88	telescences
8	9.8	3.58	disapproves
9	1.8	0.69	tinker
10	12.7	4.35	observant
11	10.1	4.52	outfitted
12	18.7	2.29	diphtheria
13	11	3.27	voyager
14	13.8	4.63	Schafer
15	11.8	3.71	engage
16	16.2	3.71	Louisa
17	19.2	3.76	sauté
18	13.2	5.55	zigzagged
19	12.5	4.64	Gilmour
20	15.7	5.50	aha
21	16.5	4.11	Ely
22	23	0.58	Zhivkov
23	22.2	1.07	kukje

Table 4: predicted (bigram model) and average reported rank for 23 words of English

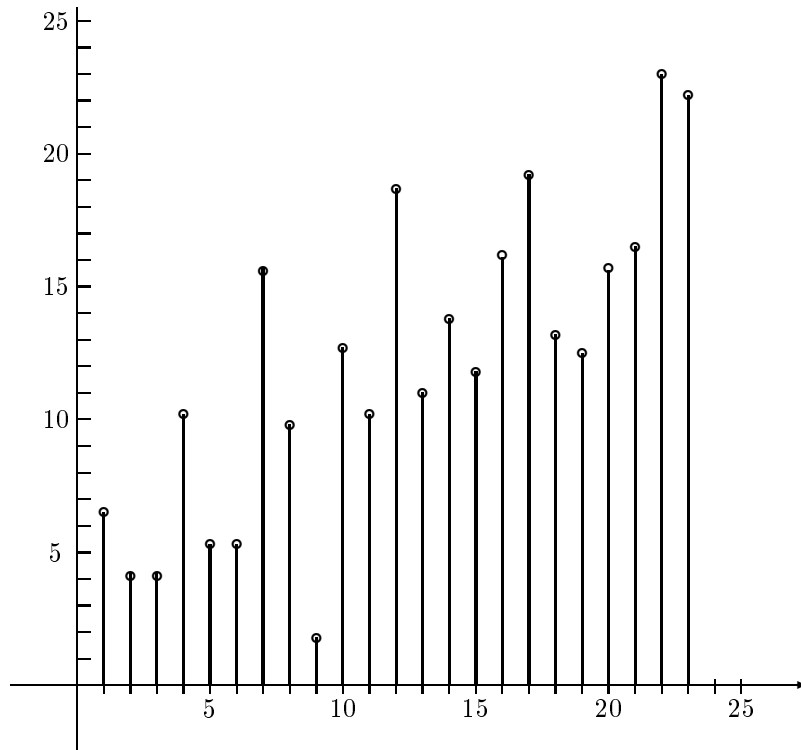


Figure 4: Average reported rank of words in Figure 2

2.1 Language identification

If we know that a sample S was produced by one of the languages that constitute our sample of languages, then we want to know what the probability is that language l_i generated it.

Why? Because our goal is to understand the world by finding the model that maximizes the probability of the perceived world. We want a method that will provide us with the way(s) that maximize the probability of the observations. (It's less important to actually compute the probability of the observations; what matters is comparing the models and the probabilities of the data generated by the models. Sometimes you can more easily calculate the relationship between the values $f(x)$ and $g(x)$ (e.g., $\frac{f(x)}{g(x)}$) than it is to compute either $f(x)$ or $g(x)$.)

If we choose a model, we can calculate $pr_{l_i}(S) = pr(S|l_i)$. But the probability that language l_i generated S is $pr(S|l_i)$, which by Bayes' rule is $\frac{pr(S|l_i)pr(l_i)}{pr(S)}$.

2.2 Cross-entropy

We have a corpus C of length N whose letter frequencies we know (because we can count them); the frequency of a typical letter l is $\frac{[l]}{N}$. We can consider various probability distributions π_i over the alphabet Σ . The probability of the corpus, using a unigram probability model and distribution π , is

$$\prod_{l \in \Sigma} (\pi(l))^{[l]}.$$

The logarithm of this quantity is

$$\sum_{l \in \Sigma} [l] \log \pi(l),$$

and the -1 times the average of this quantity is called the entropy:

$$-\sum_{l \in \Sigma} \frac{[l]}{N} \log \pi(l) = -\sum_{l \in \Sigma} freq(l) \log \pi(l)$$

We can visualize the entropy as the inner product of two vectors in $R^{|\Sigma|}$. One of the vectors describes a frequency, and hence is on the simplex consisting of points with non-negative coordinates, whose coordinates sum to 1; the other vector is a surface consisting of the points whose coordinates describe the -1 times the logarithms of a distribution (i.e., the surface of all points p such that $\sum 2^{-p_i} = 1$). Each such point describes a probability distribution for our alphabet.

We can vary these two points independently. If we vary the first but keep the second fixed, we may be looking at the entropy of different corpora, assuming the same distribution. If we vary the second but keep the first fixed, we may be looking at the entropy of a given corpus under different assumptions of the distribution that generated it. In that case, we call the quantity that we have calculated the *cross-entropy*. Explain.

It is an amazing fact that if we keep the first vector fixed and vary the second (considering different distributions), the cross-entropy is always greater....

2.3 Categories: V, C etc.; HMMs

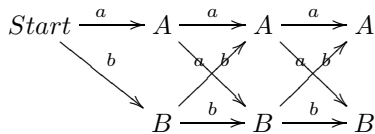
Suppose we want to divide the symbols of a language into two sets. There are many ways to do this. The natural way to understand this goal is to say, what is the simplest model we can think of that assigns a probability to Σ^* and which uses a partitioning of Σ into two sets in order to assign a higher probability to observed sets of data?

Already that critically important word “simplest” has crept in! What do we mean when we say one model is simpler than another? Let’s assume for now that we calculate the complexity of a model by the number of parameters that are used by the model. For example, a unigram with V symbols in its alphabet Σ has $V + 1$ parameters. One of them is the alphabet-size parameter (whose value is $V!$), and the others are $pr(l_i)$, for each letter l in Σ .

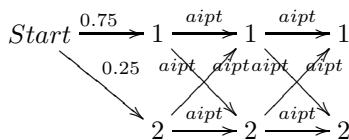
2.3.1 Hidden Markov models: finding classes of letters

Brief overview of HMMs, to which we will return in more detail. HMMs present us with the first case in which we talk about non-integral counts (which we can also call *expected counts*). This involves the case where we understand observed data (which normally we would count with integral counts) as containing only a partial specification of the “reality” we’re interested in: reality contains further parameters whose values we can’t directly observe. So we use a model to make statements about how often we expect the system to be in various states, given the observations that we in fact make.

Here is a non-hidden markov model: given the outputs, you know the path it takes through the graph.

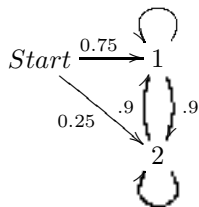


Here you don't:



Let us suppose that in State 1, the probability of generating p is $1/3$, t $1/3$, a $1/6$, i $1/6$, and in State 2, the probability of generating p is $1/6$, t $1/6$, a $1/3$, and i $1/3$.

p.33,t.33,a.16,i.16



p.33,t.33,a.16,i.16

Then we want to be able to answer questions like (and we *can* answer them): What is the probability of emitting the string #p (i.e., the string starting ‘p’)? The answer is: it’s the sum of going through the paths Start-1-2 and emitting p,

Start-1-1 and emitting p, Start-2-1 and emitting p, and Start-2-2 and emitting p. Which is: $.75(1/3) + .25(1/6) = 7/24 \approx 0.29$. (You can see I summed together the probabilities of the paths State-1-1 and Start-1-2, that is, $.75(1/3)(0.1) + .75(1/3)(0.9)$). Or we can ask: what is the probability of being in state 1 after emitting #p? The answer to that is $.75(1/3)(0.1) + 0.25(1/6)(0.9) = .025 + .0375 = 0.0625$.

And now we can turn that into soft counts. That is, if we know the probability of being in State 2 after emitting #p just like we know the probability of being in State 1 after emitting #p, then we distribute the count of 1 over those two paths, in proportion to those probabilities.

Prob(being in state 2 after emitting #p) = $0.25(1/6)(.1) + 3/4(1/3)(.9) = 0.225 + 0.004167 = 0.229$.

So the sum of the probabilities of being in states 1,2 after emitting #p is $0.0625 + 0.229 = 0.2915$.

So now we can assign softcounts. The softcount of generating #p and being in State 1 is $\frac{.0625}{.2915} = 0.2144$, while the softcount of generating #p and being in State 2 is $\frac{.229}{.2915} = 0.7855$.

2.3.2 Context vectors

2.4 Syllables

2.5 Vowel harmony

3 Words

3.1 Word frequencies; Zipf's Law

The earliest work on word frequencies is known as Zipf's Law, named after George Zipf. Assume a text composed of words, in the everyday sense. The set of words is the vocabulary V . Some words occur often; others rarely. Let us count the frequency of each word in the text, and then rank the words by their count. Each word w occurs $Count(w)$ times; we will also write this $[w]$ for brevity's sake. Each word w has a rank r_w in the list; if w_1 is more frequent than w_2 , then its rank is lower ($r_{w_1} < r_{w_2}$).

The rate at which the frequencies drop off is rather regular, and Zipf's law describes this. A simple version is:

$$freq(w) \times r_w \approx Z_{language}$$

where $Z_{language}$ is fixed for a given language (though will vary over different languages) and w is a word in the sample from the language. Unless it's important, I will not write the subscript on Z .

This approximation can be rewritten for a particular corpus C :

$$freq(w) = \frac{Count(w)}{|C|} \approx \frac{Z}{r_w}$$

and so we would expect

$$1 = \sum_{w \in V} \text{freq}(w) \approx \sum_{w \in V} \frac{Z}{r_w} = Z \sum_{i=1}^{i=|V|} \frac{1}{i}$$

But we know¹¹ that this sum does not converge as $i \rightarrow \infty$, which has made a number of people uncomfortable with this formulation.

To put the point another way, this formula works badly as the size of V gets large. But it also works poorly, from an empirical point of view, when we look at the most frequently words—the most frequent 4 or 5 words. Back in the 1950s, Benoît Mandelbrot proposed a relaxed version of Zipf’s law with two additional parameters. One way it which it can be expressed is this, where we clarify things by separating out the normalizing factor:

$$f(k; N, q, s) = \frac{1}{H_{N,q,s}} \frac{1}{(k + q)^s}$$

where

$$H_{N,q,s} = \sum_{i=1}^N \frac{1}{(k + q)^s}$$

The reality behind this formula is simpler than it looks at first glance. You can see that if $q = 0$ and $s = 1$ then we are back with the old Zipf’s law. So q and s can be looked at as parameters we adjust in order to deal with the problem of the two ends (low rank, high rank) of the curve. Do you see which parameter deals with which end?

<http://www.nslj-genetics.org/wli/zipf/>

<http://www.hpl.hp.com/research/idl/papers/ranking/ranking.html>

3.2 Maximize (word-probability/phoneme frequency) over the corpus

What is the relationship between word probability and phoneme probability in a natural language? How can we ask that question in a coherent or meaningful way?

Let’s assume the simple unigram model for the phonemes of a language, and the unigram model for the words as well (but over words, not phonemes, of course), and let’s ask how the ratio of these two *kinds* of information content

$$Q = \prod_{n=1}^{|Corpus|} \frac{pr_{syntax}(w_n)}{\prod_{i=1}^{|w_n|} pr_{phono}(w_n[i])} \quad (14)$$

4 Finding words

There are two broad families of ways in which we analyze the structure of strings, as we find in data: probabilistic (markov) models, which tell us about the probabilities of selection of elements from Σ in the future, given the past; and segmentation models, whose purpose is to allow for the restructuring of a string of elements from a fine-grained alphabet (such as Σ) to a coarser alphabet

¹¹and have known since Nicole Oresme, one of the greatest minds of the 14th century, and perhaps of all time, proved it.

\mathcal{L} which is typically called a *lexicon*; each element $w \in \mathcal{L}$ is associated with an element of Σ^* , its “spell-out”—“associated with” rather than “is,” because w may be decorated with other information, including meaning, syntactic category, and so on; but to keep things simple, we may assume that no two elements in a lexicon are associated with the same spell-out. We will always assume that each member of Σ is also a member of \mathcal{L} (roughly speaking, each member of the alphabet is a word). If there is an element w of \mathcal{L} associated with the string *the*, we will write w as **(the)**, and indicate its associated string as $h(\mathbf{(the)})$ or, when it will not cause confusion, simply as **the**. In short, **(the)** is a member of the lexicon, and it is spelled out as $h(\mathbf{(the)})$, or **the**.

Thus any string s of words formed from our lexicon \mathcal{L} is naturally associated with a string in Σ^* in one of two ways: it is associated in a natural way with a string containing word-boundaries (we call that association $h_{\#}$); and it is also associated with a string that does not contain word-boundaries (by h). For example, if our lexicon contains the words that we write as **(the)** and **(dog)**, then $h_{\#}((the)(dog)) = \mathbf{the\#dog}$, while $h((the)(dog)) = \mathbf{thedog}$. We have defined things in this way so that we can be sure that $h_{\#}$ has a well-defined and unique inverse: any string that indicates word-boundaries is associated with a unique string of words. On the other hand, a string that does not indicate word-boundaries will typically be associated with several different strings of words. For example, **the** is associated with **(t)(he)**, **(the)**, and **(t)(h)(e)**, under usual assumptions regarding the lexicon of English.

The *first* problem of word-segmentation, then, is to find a method to undo the stripping of $\#$, the following sense. Given *any* corpus C containing $\#$ s, we construct its natural lexicon L and C 's stripped version C' . We wish to find a completely general algorithm $S_1(L, C')$ that can reconstruct C , given the natural lexicon L , and possibly some statistical information available in the original corpus, such as word-frequency and word-sequence information. Needless to say, perhaps, there is no guarantee that such an algorithm exists or that if it exists, it can be found algorithmically. In general, we may wish to develop an algorithm that assigns a probability distribution over possible analyses, allowing for ranking of analyses: given a string *anicecream*, we may develop an algorithm that prefers *an ice cream* to a *nice cream*.

The *second* problem of word-segmentation assumes that the first problem has been solved; the second problem is to find a general algorithm $S_2(C')$ which takes as input a corpus C' , which is created by stripping boundaries from a corpus C , and which gives as output a lexicon L which will satisfy the conditions for L established for S_1 in the preceding paragraph. Since there are an astronomical number of different boundary-marked corpora, most with distinct natural lexicons, it goes without saying that if we can solve this problem for naturally occurring corpora, we do not expect it to be extendable to any randomly generable corpus: to put it another way, to the extent that we can solve this problem, it will be by inferring something about the nature of the device that generated the data in the first place.

$$\begin{array}{ccc} \text{strippedcorpus, lexicon} & \longrightarrow & \boxed{\text{device}} \longrightarrow \text{originalcorpus} \\ \text{strippedcorpus} & \longrightarrow & \boxed{\text{device}} \longrightarrow \text{lexicon} \end{array}$$

The problem of word breaking, or word segmentation, may seem artificial from the point of view of someone familiar with reading Western languages: it is the problem of locating the breaks between words in a corpus. In written

English, as in many other written languages, the problem is trivial: we mark those breaks with white space. But the problem is not at all trivial in the case of a number of Asian languages, including Chinese and Japanese, where the white space convention is not followed, and the problem is not at all trivial from the point of view continuous speech recognition, or that of the scientific problem of understanding how infants, still incapable of reading, are able to infer the existence of words in the speech they hear around them.

Another computational perspective from which the problem of word breaking is interesting is this: to what extent do methods of analysis that have worked well in non-linguistic domains work well to solve this particular problem? This question is of general interest to the computer scientist, who is interested in a general way regarding the range of problems for which an approach is suitable, and of considerable interest to the linguist, for the following reason. The most important contribution to linguistics of the work of Noam Chomsky since the mid 1950s has been his insistence that some aspects of the structure of natural language are unlearnable, or at the very least unlearned, and that therefore the specification of a human’s knowledge of language *prior* to any exposure to linguistic data is a valid and an important task for linguistics. But knowledge of the lexicon of a given language, or the analysis of the words of the lexicon into morphemes, is a most unlikely candidate for any kind of innate knowledge. Few would seriously entertain the idea that our knowledge of the words of this paper, or any other, are matters of innate knowledge or linguistic theory; at best—and this is plausible—the linguist must attempt to shed light on the *process* by which the language learner infers the lexicon, given sufficient data. To say that the *ability* to derive the lexicon from the data is something that few if any would disagree with, and to the extent that a careful study of what it takes to infer a lexicon or a morphology from data provides evidence of an effective statistically-based method of language learning, such work sheds important light on quite general questions of linguistic theory.

The idea of segmentation of a string $S \in \Sigma^*$ into words is based on a simple intuition: that between two extreme analyses, there must be a happy medium that is optimal. The two extremes here are the two “trivial” ways to slice S into pieces: the first is to not slice it at all, and to leave it as exactly one piece, identical to the original S , while the second is to slice it into many, many pieces, each of which is one symbol in length. The first is too coarse, and the second is too fine, for most strings that are symbolic in any sense at all. The intuition is that there is an intermediate level of “chunking” at which interesting structure emerges, and at which the average length of the chunks is greater than 1, but not enormously greater than 1. The goal is to find the right intermediate level—and to understand what “right” means in such a context.

4.1 Non-probabilistically: *Sequitur*

Craig Nevill-Manning, along with Ian Witten (see [?, ?]) developed an intriguing non-probabilistic approach to the discovery of hierarchical structure, dubbed *Sequitur*. They propose a style of analysis for a string S , employing context-free phrase-structure rules $\{R_i\}$ that are subject to two very strong restrictions demanding a strong form of non-redundancy: (1) no pair of symbols S, T , in a given order, may appear twice in the set of rules, and (2) every rule is used more than once. Such sets of rules can be viewed as compressions of the original data

which reveal redundancies in the data. An example, taken from [?] will make this clear.

Suppose the data is *abcdbcabcd*. The algorithm will begin with a single rule expanding the root symbol S as the first symbol, here a : $S \rightarrow a$. As we scan the next letter, we extend the rule to $S \rightarrow ab$, and then to $S \rightarrow abc$, to $S \rightarrow abcd$, to $S \rightarrow abcdb$, and finally to $S \rightarrow abcdbc$. Now a violation of the first principle has occurred, because bc occurs twice, and the repair strategy invoked is the creating of a non-terminal symbol (we choose to label it ‘A’) which expands to the twice-used string: $A \rightarrow bc$, which allows us to rewrite our top rule as $S \rightarrow aAdA$, which no longer violates the principles. We continue scanning and extended the top rule, now to: $S \rightarrow aAdAa$, still maintaining the second rule, $A \rightarrow bc$. Scanning the next symbol, b , the top rule becomes $S \rightarrow aAdAab$; and the next, c , the rule becomes $S \rightarrow aAdAabc$. The bc just found is replaced by A , so we have $S \rightarrow aAdAaA$, but this rewrite creates a new violation, since aA appears twice. This leads to the creation of a new non-terminal symbol ‘B’ and the rule $B \rightarrow aA$, and the top rule is shortened to $S \rightarrow BdAB$. Not surprisingly, the highest level rule is quickly losing its terminal symbols in favor of non-terminal symbols. As the next symbol, d , is scanned, the top rule becomes $S \rightarrow BdABd$, and this triggers a cascade of changes. A new symbol C is created which expands $C \rightarrow Bd$, and the top rule becomes $S \rightarrow CAC$. The rule expanding B (which is $B \rightarrow aA$) is no longer licit, because its only application is to expand the node B which occurs only once in the grammar, in the expansion of the new C . A rule must appear at least twice to survive, so we remove the rule $B \rightarrow aA$ by expanding C in this way: $C \rightarrow aAd$. All the conditions are satisfied, and we have a small hierarchical compression of the original string of symbolic data.

Where are the words, now?

4.2 Minimum Description Length

4.2.1 Brent, de Marcken

4.2.2 MDL approaches

Minimum Description Length (or MDL) is an approach to data analysis developed by Jorma Rissanen, [?, ?] developing ideas of algorithmic complexity discussed by a range of scholars, notably Solomonoff, Chaitin, Wallace, and notably Kolmogorov. At its heart is the notion that the fundamental challenge of analyzing data is the correct division of a set of observations into information, complexity, and noise, in Rissanen’s terminology, each of which can be measured in (Shannon’s) *bits*. This terminology is not ideal in the context of applying MDL to the problem of unsupervised language acquisition, because Rissanen’s *information* corresponds to the *grammar* that generated the data, the *complexity* is a measure related to the *message* that is encoded by the data, and *noise* is, well, noise.

An MDL approach to the analysis of a set of data D , where $D \subset \Sigma^*$, with Σ an alphabet, begins with an assumption about the class of models \mathcal{M} that will be taken into consideration, and a background assumption about how much encoding, in bits, is required to make any particular grammar completely explicit, typically given in terms of a universal Turing machine. However we choose to define that class of models, each member will be a grammar capable of generating (or accepting) the data D . If we have chosen a model class that contains the

grammar g_1 (see 1) (generate all strings), then obviously it accepts the data D , but it imposes little structure, and it accepts not only D , but a very, very large and infinite superset of D ; this account posits very little *information* (in our terms, very little *grammar*) in the data. We are not obliged to choose so large a model class. Indeed, the artistry that we call science includes the judgment of just what that model class should be.

To repeat: one makes a background assumption about how algorithms will be encoded, and then that assumption allows us to measure the information contained in a grammatical model g that we are entertaining (again, the information is very closely related to the length of the encoding of the grammatical model, or grammar, which we indicate as $|g|$). In addition, we make the assumption that all algorithms are probabilistic. This assumption can be interpreted in two equivalent ways. From the point of view of string accepting, the grammar generates a positive number (< 1) associated with any string in Σ^* ; that number is the string's probability; and these probabilities sum to 1.0. From the point of view of generation, any real finite binary sequences of 0's and 1's will be interpreted as a binary fraction beginning "0." (and hence as a rational number between 0 and 1), and a probabilistic grammar can always be interpreted as a device that takes such finite binary expansions, and produces a string; the length of the shortest such binary expansion that generates D is the *complexity* of the message D (written $|D|_g$), and is closely related to the inverse binary logarithm of the probability assigned to the string by an accepting model.

Thus we see that given a model/data pair g/D , we generate two numbers, the information of the model g and the complexity of the data D given the model g . We say that the model/pair then has the *description length* $\mu_{g,D} = |g| + |D|_g$, and we choose a particular model \hat{g} :

$$\hat{g} = \arg \min_g \mu_{g,D} = \arg \min_g |g| + |D|_g \quad (15)$$

Since a lexicon typically includes its alphabet as a subset (which is to say, each individual letters is also a word, in actual practice), any given model will typically be able to generate a given string D in many different ways (remember the case of *anicecream*), each with its own probability. In practice, we typically define the probability of a string D as the probability associated with one particular *parse* of D .

Putting these threads together, we can see that an MDL approach to word-breaking consists of two things: a suitable definition of a class of lexicon models, where each model assigns a probability distribution over the strings which it generates; and second, a probability distribution over that class of lexicon models. In addition, a model of word-breaking may be linked with a theory of acquisition, which takes a string, its corpus (and hence immediate access to the alphabet Σ in which it is inscribed, so to speak) and proposes a lexicon for it.

Work by Brent [?]; [1]; [?] [2]

4.3 Problems with this approach to word discovery

The first set of problems that one encounters in looking at the results of this approach are these: (i) the approach makes pieces that are too large, like *of the, to the, of course*, etc. (ii) the approach also makes pieces that are too

small when the word is relatively infrequent but is composed of pieces that are relatively frequent, like finding *manage ment* as two words, rather than one.

When we put it that way, the problem is obvious. The word-unigram model of language is simply way too simple and simplistic for dealing with natural language. Natural language has an enormous amount of structure, at many different levels, and all that structure is on display in samples of any size from any language. If we expect a model as simple as the word-unigram model to work, we are going to be as sadly disappointed. We need a model as complex as the reality that in fact lies behind the data from the natural languages we look at. We need a model that includes an explicit play for word-internal structure, and an explicit place for word-external structure. We call the first *morphology*, and the second *syntax*.

5 Morphology: Making a lexicon

5.1 General remarks on morphology

The field of morphology has as its domain the study of internal word structure, and in practice that has meant the study of three relatively autonomous aspects of natural language, which one can identify as morphophonology, morphosyntax, and morphological decomposition. To explain what each covers, we must introduce the notion of *morph*—a natural, but not entirely uncontroversial notion. If we consider the written English words *jump*, *jumps*, *jumped*, and *jumping*, we note that they all begin with the string *jump*, and three of them are formed by following *jump* by *s*, *ed*, or *ing*. When words can be decomposed directly into such pieces, and when the pieces recur in a functionally regular way, we call those pieces *morphs*.

- Morphophonology. It is often the case that two (or more) morphs are similar in form, play a nearly identical role in the language, and each can be analytically understood as the realization of a single abstract element—abstract merely in the sense that it characterizes a particular grammatical function, and abstracts away from one or more changes in spelling or pronunciation. For example, the regular way in which nouns form a plural in English is with a suffixal *-s*, but words ending in *s*, *sh*, and *ch* form their plurals with a suffixal *-es*. Both *-s* and *-es* are thus morphs in English, and we may consider them as forming a class which we call a *morpheme*: *s*, *-es* whose grammatical function is to mark plural nouns. The principles that are involved in determining which morph is used as the correct realization of a morpheme in any given case is the responsibility of morphophonology. Morphophonology is, in a real sense, the shared responsibility of the disciplines of phonology and morphology.
- Morphosyntax. Syntax is the domain of language analysis responsible for the analysis of sentence formation, given an account of the words of a language. In the very simplest case, the syntactic structure of a well-formed sentence could conceivably be described as **noun-verb-noun**, where the first noun is the subject and the second the object, but grammar is never that simple; in reality, the morphs that appear in one word (for example,

verbal suffixes) may also specify information about the subject or the object (for example, the verbal suffix *-s* in *Sincerity frightens John* specifies that the subject of the verb is grammatically singular). Morphosyntax is the shared responsibility of the disciplines of syntax and morphology.

- Morphological decomposition. While English has many words which contain only a single morpheme (e.g., *while*, *class*, *change*), it also has many words that are decomposable into morphs, with one or more suffixes (*helpful*, *thought-less-ness*), one or more prefixes (*out-last*,) or combinations (*un-help-ful*). But English is rather on the tame side as natural languages go; many languages regularly have several affixes in their nouns, adjectives, and even more often, their verbs. (e.g., Spanish *bon-it-a-s*).

Three interrelated questions:

- Word segmentation: How can we develop a *language-independent* algorithm that takes as input a large sequence of symbols representing letters or phonemes and provides as output that same sequence with an indication of how the sequence is divided into words?
- How can we develop a language-independent algorithm that takes as input a list of words and provides as output a segmentation of the words into morphemes, appropriately labeled as prefix, stem, or suffix—in sum, a morphology of the language that produced the word list?
- How can we implement our knowledge of morphology in computational systems in order to improve performance in natural language processing?

General comments here.

Morphological decomposition. Conversion; compounding.

Inflectional and derivational morphology. A useful distinction is generally made between derivational and inflectional morphology. The distinction falls squarely on whether the phenomenon one is considering is relevant to morphosyntax or not. If it is relevant, then it is considered inflectional morphology, and otherwise it is considered derivational morphology.

Users of natural languages (which is to say, all of us) need no persuasion that words are naturally occurring units. We may quibble as to whether expressions like “of course” should be treated as one word or two, but there is no disagreement about the notion that sentences can be analytically broken down into component words.

In all, or virtually all, languages, it is appropriate to analytically break words down into component pieces, called morphemes; such an analysis is called a morphology, and is the central subject of this chapter. Morphologies are motivated by three considerations: (1) the discovery of regularities and redundancies in the lexicon of a language (such as the pattern in *walk:walks:walking :: jump:jumps:jumping*); (2) the need to predict the occurrences of words not found in a training corpus (e.g.); and (3) the usefulness of breaking words into parts in order to achieve better models for statistical translation and other models particularly sensitive to the meaning of a message.(explain).

Thus morphological models offer a level of segmentation that is typically larger than the individual *letter*, and typically smaller than the *word*. For

example, the English word *unhelpful* can be analyzed as a single word, as a sequence of nine letters, or from a morphological point of view as a sequence of the prefix *un*, the stem *help*, and the suffix *ful*.

5.2 Putting phonology into the lexicon

5.3 Putting segmentation structure in the lexicon: morphology 1

5.4 String Edit Distance

5.5 Rich morphologies : morphology 2

6 Proto-syntax

6.1 Word-based transition models

6.2 Syntactic categories, for smoother transitional probabilities

Using the lexicon for transitional probabilities in the sentence. Brown et al.

6.3 Brill POS tagging

7 Prior distributions over grammars: grammar complexity and Universal Turing machines

8 Semantics and meaning

8.1 Latent semantic indexing

8.2 Word sense disambiguation

Word sense disambiguation [3]

References

- [1] Michael R. Brent and Timothy A. Cartwright. Distributional regularity and phonotactic constraints are useful for segmentation. *Cognition*, 61:93–125, 1996.
- [2] Carl de Marcken. *Unsupervised Language Acquisition*. PhD thesis, MIT, 1996.
- [3] David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. pages 189–196, 1995.