Language Learning Diary

Linas Vepstas

July 13, 2018

Abstract

The language-learning effort involves research and software development to implement the ideas described in ArXiv abs/1401.3372[GV14]. This document contains supplementary notes and a loosely-organized semi-chronological diary of results.

Introduction

The language-learning effort involves research and software development to implement the ideas described in in ArXiv abs/1401.3372[GV14]. This document contains supplementary notes and a loosely-organized semi-chronological diary of results. Its not actully chronological: in general, it is organized so that theory preceeds data analysis. Usually.

The initial stages of this work require the extraction of word-pair probabilities from raw text, and the use of these to induce a Link Grammar[ST91, ST93]. This extends prior work on MST parsers[Yur98], by inducing link types for word-pair relations.

Later stages further extend beyond what is possible with Link Grammar by inducing synonymous words and phrases. The goal here is to unifiy into a consistent framework various techniques for unsupervised semantic discovery that have already been proven in narrower contexts[PD09, Lin98, LP01].

The first section of this document is a review of various defintions of probabilites that can be obtained from natural langauge text. This is followed by a roughly chronological diary of further observaions and results. Many revisions are made out of chronological order.

Lexical Attraction, Mutual Information, Interaction Information

The goal of this section is to clarify some of the formulas used by Deniz Yuret in his PhD thesis "*Discovery of Linguistic Relations Using Lexical Attraction*", MIT 1998 (http://www2.denizyuret.com/pub/yuretphd.pdf). These formulas are vitally important, because they provide a strong tool when working with text; this has been shown by Yuret in his thesis, as well as by many others, as well as by my own practical exprience with using them.

Possibly the most useful formula is the one in the middle of page 40. By the time that we get to it, the terms "mutual information" and "lexical attraction" are being used interchangably. This formula states the $MI(x,y)$ for two words x and y; yet it is manifestly not symmetric in x and y, since x is the word on the left, and y is the word on the right. By constrast, textbook (wikipedia) definitions of MI are symmetric in thier variables. Below I try to dis-entangle the resulting confusion a bit, and give a more correct derivation of the formula. The key is to observe that the formula contains an implicit pair-wise relationship between two words, and that there are actually three variables: two words, and thier relationship. If this implicit relationship is made explicit, then the confusion evaporates. It also opens the door to talking about the MI (or the interaction information InI) of more complex relationships, not just pair-wise ones.

Being able to correctly write down the MI and the InI for complex relationships is important for NLP: relationtionships can be labelled by types (subject, object) and by word classes (noun, verb), and have various dependency constraints between them. Thus, we need to be able to talk both about a labelled directed graph, and the entropy or mutual information contained in it's various sub-graphs.

In defense of Yuret, he does say, on page 22, that "lexical attraction is the likelihood of a syntactic relation." However, the relation starts becoming implicit by eqn 12 on page 29. An unexplained leap is then made from eqn 12 to the formula on page 40. The below gets fairly pedantic; this seems unavoidable to avoid confusion.

Definitions

Let $P(R(w_l,w_r))$ represent the probability (frequency) of observing two words, w_l and w_r in some relationship or pattern R. Typically, R can be a (link-grammar) linkage of type t connecting word w_l on the left to word w_r on the right; implicitly, both w_l and w_r occur in the same sentence. The goal of this discussion is to enable relations R that are more general than this; for now, though, R is a word-pair occuring in a single sentence.

The simplest dependency grammar language model has only one type t, the ANY type. This is the type that Yuret uses: it makes no distinction at all between subject, object relations (that is, all depdencies are unlabelled), and it does not make a head-dependent distinction (all dependencies are bi-directional). Thus, in what follows, we do the same: initially, the relation $R(w_l,w_r)$ is simply the statement that the words w_l and w_r are connected by an unlabelled, un-directed edge. For this simplest case, what $R(w_l,w_r)$ does is to capture that w_l is to the left of w_r.

In what follows, the relation $R = R(w_l,w_r)$ refers to a generic two-word relation, and not necessarily this simplest one. To regain Yuret's formula, use the simplest relation, the ordered word-pair relation, given just above.

The quantity of interest is the (unconditional) probability $P(R(w_l,w_r),w_l,w_r)$ of observing the two words w_l and w_r in a relation $R = R(w_l,w_r)$. To correctly understand and work with this quantity, some care must be taken with the notation for several related probabilities. First, one has $P(w)$, the probability of observing the word w in the data sample. Next, one has $P(S(w_1,w_2),w_1,w_2)$, the probability that the two words occur in the same sentence. Again, $S(w_1,w_2)$ denotes a relation between the two words; it differs from $R(w_1,w_2)$ in that the word-order does not matter. A third kind of pair relation is the unconditional probability of observing two words, which can be *defined* as $P(w_1,w_2) = P(w_1)P(w_2)$. In this case, instead of assuming independence of two random variables, we define them to be so. This is possible, because we have a notation for specifying when there is a correlation. That is, if there was some correlation (relation) $C(w_1,w_2)$ between them, then one should write this explicitly, as $P(C,w_1,w_2) = P(C(w_1,w_2),w_1,w_2)$. The notation here allows the various needed probabilities to be defined without ambiguity.

Thus, assumptions of independent variables are now replaced by a notational infrastrcture. Note, in particular, that if one uses a frequentist definition for the probabilities (as will be done in what follows), then the probabilities are not independent of the data sample from which they are drawn. Thus, all probabilities here have an implicit dependence on the data sample. This dependency is not explicitly shown. Some care must be taken to use the same data sample throughout.

The above notation allows the definition of conditional probabilities, in the conventional sense. For example, one has that

$$P(R,w_l,w_r) = P(R|w_l,w_r)P(w_l,w_r)$$

or that

$$P(R|w_l,w_r) = \frac{P(R,w_l,w_r)}{P(w_l,w_r)}$$

as the conditional probability of observing the relation R, given that it's component parts are observed. From the earlier definitions, the denominator factors, and so we conclude that the correct expression for the conditional probability is:

$$P(R|w_l,w_r) = \frac{P(R,w_l,w_r)}{P(w_l)P(w_r)} \tag{1}$$

This is the probability of observing the relationship R given that the individual parts of the relationship have been observed. The relation R includes all correlations between the two words: thier ordering as well as thier co-occurance in a sentence.

Take care, however: $P(R|w_l,w_r)$ is NOT the probability of seeing R, given that w_l and w_r occur in the same sentence. This would instead by given by $P(R,w_l,w_r)/P(S,w_l,w_r)$. This is an entirely different.

Frequentism - Counting words and pairs

In order to be usable, a computable definition for the probabilities must be given. For this, the definition can only be frequentist. That is, the probabilities are to be obtained from empircal data; from counting frequencies as they occur in data samples taken from nature. The frequency $P(w)$ of observing a word w is obvious:

$$P(w) = \frac{N(w)}{N(*)}$$

where $N(w)$ is the count of observing word w and $N(*)$ is the total number of words observed. That is, by definition, it is the wild-card summation

$$N(*) = \sum_w N(w)$$

How to count words is not entirely obvious, so even these defintions need care. There are several ways in which one can count words. One way is to simply count how many times a word occurs in the block of sample text. Another way is to count how many times a word occurs in parses of the sample text. These are not the same! For example, if a parse connects words by edges (by dependency-grammar relations), then one can count each word once, for each time that it occurs at the end of an edge. In this counting, the word-count is exactly double the word-pair count. A word is then counted multiple times, if it participates in multiple edges. If the sample text is parsed multiple times, then addtional counts can result that way. To maintain consistency with the defintions given in the previous section, $N(w)$ is defined to be the number of times that the word w occurs in the data sample, and independent of any other relations that w might be engaged in. For now, it is assumed that the segmentation of the text sample into words is unambiguous.

Let $F(S(w),w)$ be the number of times (frequency) of observing word w in any sentence S. This can be computed as

$$F(S,w) = \frac{N(w)}{NS}$$

where $N(w)$ is the number of times a word w was observed in a data sample, and NS is the number of sentences in that same sample. This counts with "multiplicity", in that w can appear in a sentence more than once. That is, F is not a probability, rather, it is an expectation value of the number of times that a word is observed. This can be made explicit, by writing

$$F(S,w) = \frac{N(w)}{N(*)}\frac{N(*)}{NS} = P(w)L(S)$$

with $L(S) = F(S,*)$ being the average sentence length (the expectation value of the number of words in a sentence).

Three different word-pair relationships are interesting. First, define the relation $S(w_1,w_2)$ as being the relation that both words w_1 and w_2 occur in the same sentence, but in arbitrary order. It is symmetric: $S(w_1,w_2) = S(w_2,w_1)$. Define $A(w_l,w_r)$ as being the relation that both words w_l and w_r occur in the same sentence, and that w_l is to the left of w_r. By this definition, the counts for the two are related: one has that

$$N(S,w_1,w_2) = N(A,w_1,w_2) + N(A,w_2,w_1)$$

This is the symmetrized count.

Neither of S or A is yet the relation $R(w_l,w_r)$ mentioned above, which is defined as being the relation that both words w_l and w_r occur in the same sentence, that w_l is to the left of w_r, and, most importantly, that there is a link-grammar link (of type "R") connecting the two. Observe that although A can be deduced from S, there is no simple or obvious relation between S and R; these are essentially independent relations.

The way that the statistics are collected for A and for R are different. To count the A-type relations, one tokenizes a sentence into words, and then, counts every possible word-pair in the sentence. Effectively, one draws a clique of edges between the words, and then counts each edge. The statistics for R are collected by parsing the sentence into a random planar tree, and then counting the edges in the tree. The result for this counting is NOT the same as that for type-A edges. The reason for this is demonstrated in depth, in the section Edge-counting 27 March 2017 on page 59, below.

Initially, there is only one link relation "R" between two words: this is the "ANY" link-type. However, in general, "R" can be other kinds of link-types. Note that "R" can also have a head-tail dependency order: either w_l or w_r can be the head-word of a directional link. Thus, there are three different symmetrizations that can be obtained from "R": by failing to make a left-right distinction, by failing to make a head-tail distinction, and failing to do either.

The definition for the probability of observing a relation can be taken to be

$$P(R,w_l,w_r) = \frac{N(R,w_l,w_r)}{N(R,*,*)} \tag{2}$$

where

$$N(R,*,*) = \sum_{w_l,w_r} N(R,w_l,w_r)$$

This can be roughly understood as being the conditional probability of observing the relation $R(w_l,w_r)$ between two specific words, given that the relation R between any two words was seen.

Is it possible to define the unconditional probability $P(R,*,*)$ of seeing the relationship? The path to the answer is not entirely straight-forward. First consdier the probability $P(S,w_1,w_2)$ of seeing two words in the same sentence. This probability is defined just as in eqn 2; that is, $P(S,w_1,w_2) = N(S,w_1,w_2)/N(S,*,*)$. From this, one can define the frequency of seeing a relation in a sentence, as

$$F(R|S,w_1,w_2) = \frac{P(R,w_1,w_2)}{P(S,w_1,w_2)}$$

This gives the expectation value of seeing the relation R in a sentence, given that the two words are already known to be in the sentence. That this is an expectation value should be clear, as the relation might appear multiple times in one sentence (e.g. if one of the words is repeated). The sum

$$F(R|S,*,*) = \sum_{w_l,w_r} F(R|S,w_l,w_r)$$

then counts the average number of relations per sentence. For the any-type ordered-pair relation, clearly one must have that there are at least as many relations as there are words in the sentence, minus one, since each word must appear in at least one (distinct) relation. That is, $F(S,*) - 1 \leq F(R|S,*,*)$ with $F(S,*)$ the expected length of a sentence.

Similarly, one can consider the ratio

$$F(S,w_1,w_2) = \frac{P(S,w_1,w_2)}{P(w_1)P(w_2)}$$

which captures the frequency at which two words are seen in the same sentence. The summation $F(S,*,*)$ then counts how many pairs are seen per sentence. Assuming that the counting was performed with a uniform distribution, this should then equal the number of edges in a clique. That is, for a sentence of length m, there should be $m(m-1)/2$ word-pairs (edges) counted for that sentence. This should hold approximately, on average, so that $F(S,*,*) \approx F(S,*)(F(S,*)-1)/2$.

From the development above, it should be clear that it is not really possible to define a quantity $P(R,*,*)$ that is the "probability of seeing a relation". We can count the number $N(R,*,*)$ of times the relation occurs in a data sample. We can count the average number of times the relation is seen in a sentence. However, as long as the relation occurs at least once in the data sample, one would have to say that the "probability of seeing the relation in the data sample" is one. The problem is one of normalization: there is no universe, of which $N(R,*,*)$ is a fractional measure.

That said, once can still consider an interesting ratio:

$$F(R,*,*) = \sum_{w_l,w_r} F(R,w_l,w_r) = \sum_{w_l,w_r} \frac{P(R,w_l,w_r)}{P(w_l)P(w_r)}$$

This can be interpreted as a kind-of centrality. So, for example, for the any-pair relation, every word in the data sample must participate in at least one such pair-relation, and thus, we expect that $F(R,*,*) \approx 1$. The precise value is related to the tree-parse that is being used to generate the any-relation. If the (random) parse-tree is acyclic, then the number of edges is comparable to the number of words. If the parse-tree contains cycles, then there may be more relations than there are words.

Yuret's Mutual Information

Deniz Yuret introduces the concept of "lexical attraction". It is reviewed breifly, here. He defines a probability $\mathscr{P}(w_l,w_r)$ of seeing an ordered pair; as compared to the above, the relation is implicit. To make it explicit, one should write:

$$\mathscr{P}(w_l,w_r) = P(A(w_l,w_r),w_l,w_r) \tag{3}$$

which indicates the relation explicitly, as well as noting that the order of the positions in the relation matter. To avoid confusion, the cursive $\mathscr{P}$ is used for the Yuret notation, instead of the roman P which is reserved for the defintions above.

The letter A used here reminds us that in Yuret's work, the pair-counting method used is the clique-edge-counting mechanism, described above, rather than the random-planar-tree relation. One expects the two to be similar, but not the same.

Yuret also uses the notation $\mathscr{P}(w_l,*)$ and $\mathscr{P}(*,w_r)$ for wild-card summations, defined as

$$\mathscr{P}(w_l,*) = \sum_{w_r} \mathscr{P}(w_l,w_r) \qquad \text{and} \qquad \mathscr{P}(*,w_r) = \sum_{w_l} \mathscr{P}(w_l,w_r)$$

It is tempting to conflate $\mathscr{P}(w_l,*)$ with $P(w_l)$ but that would be wrong; not every possible word can occur on the w_r position. This suggests a different, but tempting, error, that $\mathscr{P}(w_l,*) \leq P(w_l)$. This is also not the case! A word might occur more frequently as the left side of a pair, than it does all by itself in the sample text. This follows from the frequentist definitions; the denominators for the two probabilities are not compatible; they do not range over the same universe.

Yuret defines the "lexical attraction" as

$$\mathrm{MI}(w_l, w_r) = \log_2 \frac{\mathscr{P}(w_l, w_r)}{\mathscr{P}(w_l, *)\mathscr{P}(*, w_r)} \tag{4}$$

so that large positive MI is associated with words that rarely seen one without the other (e.g. '*Northern Ireland*' from his examples.) Note the absence of a minus sign in the above! See below for an explanation. Large-MI word pairs occur when $\mathscr{P}(w_l, w_r)$ is roughly comparable to $\mathscr{P}(w, *) \approx \mathscr{P}(*, w)$.

It is worth reviewing Yuret's example, at this point. He looks at the word pair 'Northern Ireland' and states (based on a particular corpus that was analyzed) that $-\log_2 P(\text{'Northern'}) = 12.60$ and that $-\log_2 P(\text{'Ireland'}) = 14.65$ and finally that $-\log_2 \mathscr{P}(\text{'Northern'}, \text{'Ireland'}) = 16.13$. What these numbers mean is that although either word alone occurs at a rate of roughly once in ten-thousand words, the word-pair together occurs at the rate of one in thirty-thousand or so: the word pair occurs almost as often as either word alone. Thus, the resulting MI is very large: $\mathrm{MI} = -16.13 + 12.60 + 14.65 = 11.12$. The choice of sign in eqn 4 is such that words that co-occur have a large positive value. In practice, the distribution of the MI for word-pairs runs from about -15 to about +35, and, when ranked accoring to MI, the probabilites form a rounded mountain-peak, two-sided, each side being linear (Zipfian) with the peak at about MI=4 or 6. (See my other notes for a graph.)

1 January 2014

OK, after that side distraction, which helped clear up notation, back to the main show ...

The main show is this: We want to model language, and specifically, find a 'minimal' set of relations R that are accurately generative. The meaning of 'minimal' seems obvious, intuitively, but a lot harder to pin down mathematically. We need to pin it down to get an algorithm that works in a trust-worthy, understandable fashion.

So: what is the total space of relations, and how do we find it? The simplest model is then a Zipfian distribution of words, but placed in random order. This model has a total entropy of

$$H = -\sum_w P(w) \log_2 P(w)$$

For a recent swipe at parsing a few hundred articles from the French wikipedia, I get H=7.2. This is on 17K words, observed a total of 35M times (actually, observerd each sentence 100 times, so really just 350K 'true' observations of words).

How does one count the entropy of the rule-set? Elucidating this is the goal-set.

But first, step back: describe the rules.

OK ... so, once again ... sentence structure is to be described via link-grammar, using disjoined conjunctions of connectors. This is theoretically sound, as it seems to be isomorphic to categorical grammars (via type-theory of the connectors; need a formal proof of this someday, but for now it seems 'obvious'). Also link-grammar is fully compatible with dependency grammar. So lets move forward. But this is an old debate, off to the side, immaterial for now.

How to count relations

Consider a sentence with n words in it, numbered $w_1, w_2, \cdots, w_n$ left to right. We want to constrain grammar by discovering a set of relations $R(w_1, w_2, \cdots, w_n)$ such that $P(R(w_1, w_2, \cdots, w_n)) > 0$ when the sentence is gramatically valid (*i.e.* such an R exists), and P is zero when no such R exists (*i.e.* the sentence is not gramatically valid.) The first and most obvious simplification rule is to observe that R can be replaced by $R(W_1, W_2, \cdots, W_n)$ where each W_k is a set of words. That is, instead of listing each sentences individually, we list certain classes of sentences. In other words, the relations $R(w_1, w_2, \cdots, w_n)$ are in one-to-one correspondance with a list of grammatical sentences $(w_1, w_2, \cdots, w_n)$, so simply listing all possible sentences is a very verbose way of specifying a grammar. It is linguistically 'obvious' that sentences fall into classes, and so the two relations $R('this', 'is', 'a', 'dog')$ and $R('this', 'is', 'a', 'cat')$ can be replaced by $R('this', 'is', 'a', W_n)$ where $W_n = \{'dog', 'cat'\}$. In fact, W_n can be a rather large set of nouns.

So ... the question is: what is the reduction of complexity, by performing this classification? What is the correct way of counting? I assume that 'complexity' is a synonym for 'entropy', so we are looking to do two things: enumerate the states of the system, and proivde a measure for complexity. So, lets consider a language with N nouns, so that the cardinality of W_n is $|W_n| = N$ and the only valid sentences are $('this', 'is', 'a', w)$ with $w \in W_n$. Before simplification, we had N relations R, one per sentence. We also had $N+3$ sets, each set containing a single word; the N nouns, and the three words $'this', 'is', 'a'$. After simplification, we have one relation R, and four sets; three of the sets have cardinality 1, the fourth set has cardinality N.

Revision: July 2014

There seem to be several ways of counting. Some of these seem to give wrong answers. Some just seem wrong. This is all very confusing, so I've altered the entries to explicitly show the different ways of counting.

Method 1 (naive counting): One counting rule is to count set-membership relations on equal footing with structural relations. Thus, before simplification, we had $N+3$ sets, each a singleton, and thus $N+3$ set membership relations. After simplification, we have four sets, but still have $N+3$ set membership relations. Thus, this particular simplification step does not reduce the number of membership relations at all. This seems disconcerting... Let's provisionally go with this and see what happens. Thus, before simplification, we had $2N+3$ relations grand-total, and afterwords, we have $N+4$ relations grand-total.

What is the correct 'thermodynamic' picture of what's going on? In this toy problem, we have a grand-total state space of size $(N+3)^4$ since any of the $N+3$ words can appear in any of the four slots in a four-word sentence (micro-canonnical ensemble). The entropy, at 'infinite temperature' where all possible four-word sequences occur with equal probability is then $4\log_2(N+3)$. The entropy of the set of grammatical sentences is $\log_2 N$ since there are only N possible grammatical sentences. In this toy grammar, there are also invalid setences of length 1,2,3,5,6,7,... and so the total size of the space of word-sequences is clearly infinite.

OK, so the space of word-sequences is very concrete, and easy to describe and measure, at least for toy grammars. What about the space of relations? Well, the claim is that the entropies of the before-and-after models are $\log_2(2N+3)$ and $\log_2(N+4)$, respectively. Neither of these matches the entropy of the set of allowed sentences (which is $\log_2 N$), so this seems paradoxical, and begs the questions 'did we count correctly?' and 'did we actually simplify anything by making the above change of description?' Hmm. The correct answer seems to be 'no' and 'no'.

Method 2 (subtract one): To 'fix' the oddball results above, an alternative counting methodology is to subtract 1 from the cardinality of every set. This would then give both $\log_2 N$ as the entropy for both the before and after relation-sets. Thus, before, we had N relations and $N+3$ sets, each of weight zero, for a total weighted-relation count of N. After, we have one relation and four sets; three of the sets have weight zero, one set has a weight of $N-1$ so the total weighted relations is again N. This seems to resolve the paradox. But why subtract one? That's a bizarre rule, almost unheard-of in information theory.

Method 3 (naive log addition): Total complexity is given by:

$$K = \log_2 |Rel| + \sum_{W \in Wrds} \log_2 |W|$$

where Rel is the set of relations, and $Wrds$ is the set of word-lists, and $|W|$ is the cardinality of each word-list. We then get, before simplification, $|Rel| = N$ and $|W| = 1$ for each of the word-sets. The total complexity is thus $K = \log_2 N$ as expected (i.e. equal to the log of the total number of possible sentences). After simplification, there is $|Rel| = 1$ and 3 sets with $|W| = 1$ and one set with $|W| = N$, thus yeilding a total of $K = \log_2 N$ again. This seems to give a plausible answer, and provides a plausible argument.

Method 4 (relational complexity): Treating each relation as being equally complex seems odd. It would seem to make more sense to have each relation contribute according to its complexity, so that the contribution of the relations to the total complexity is:

$$\sum_{R \in Rel} C_R$$

with C_R the complexity of each relation, itself the log of some measure. But how do we measure complexity? Is it Kolmogorov complexity? There's no obvious *a priori* defintion for this. The defintion of this complexity would seem to depend on the particular algorithm machinery of the grammar; that is, on the 'programming language' used to represent the relation. This is the traditional ambiguity attached to the Kolmogorov complexity.

Method 5 (corpus distribution): Instead of measuring the complexity of a grammatical expression (in an as-yet unkown grammar), instead, use the corpus frequency ass a proxy. For the above example, if the N sentences are equi-distributed (i.e. occur equally likely in the corpus), then, before simplification, each of the relations has a complexity

$$C_R = -\frac{1}{N} \log_2 \frac{1}{N}$$

so that, before simplification,

$$K = \sum_{R \in Rel} C_R = \log_2 N$$

which again seems to be the desired answer. After simplification, there is one relation that applies to the entire corpus, so that $C_R = 0$ after simplification.

Method 6 (corpus word-counts): If we are taking word-relation frequencies from the corpus, then we should be taking word-set frequencies from the corpus as well. That is, the word-set contribution $\log_2 |W|$ is assuming an equi-distribution. This should be replaced by the corpus contribution

$$-\sum_{w \in W} p(w) \log_2 p(w)$$

Summary. Provisionally, the last two methods seem to be the best way to move forward. To summarize, the complexity is given by

$$K = -\sum_{R \in Rel} P_R \log_2 P_R - \sum_{W \in Wrds} \sum_{w \in W} P_w \log_2 P_w \tag{5}$$

where $P_R = P(R) = P(R(W_1, W_2, \cdots, W_n))$ is the probability of observing the relation R in a sample corpus, and $P_w = P(w|W)$ is the probability of observing word w in the corpus, conditioned on its appearence in the corpus having to do with it belonging to the word-class W.

Counting Link-Grammar Relations

Per link-grammar, each relation is decomposable into pair-wise relations; this is the so-called 'parse' of a sentence. If the relation is a single word-per-slot sentence relation, then the 'parse' is literal. We write

$$R(w_1, w_2, \cdots, w_n) = \prod_{j,k,m} R_\alpha(w_j, w_k, t_m)\; Q(R_\alpha, R_\beta, \cdots, R_\omega) \tag{6}$$

where $R_\alpha(w_j, w_k, t_m)$ is a single connected pair of words, connected by the connector t_m. The product symbol $\prod$ implies that all such binary relations must hold. The awkward $Q(R_\alpha, R_\beta, \cdots, R_\omega)$ at the end is the additional no-links-cross constraint in the current link-grammar parser. Its a non-local constraint involving all of the binary relations. It also subsumes any 'post-processing' rules, although, for the language learnign exercise, there won't be any post-processing rules. At any rate, Q is a place where higher ordrer constraints can be applied. In particular, the most genneral form for Q should be $Q(R_\alpha, R_\beta, \cdots, R_\omega, w_1, w_2, \cdots, w_n)$ since, in principle, it could depend on the word-choice, although the no-links-cross constraint does not.

Yuret proposes a way of discovering the pair-wise relations[Yur98]. He makes the implicit, unvoiced assumption that there is a single, unique connector type t_m for every ordered pair of words w_j, w_k; that is, that $t_m = t_m(w_j, w_k)$. Viz, specifically, that such connectors are in 1-1 correspondance with word-pairs. (I don't think he's aware of this assumption; I don't think anyone has ever before realized that he's making such an assumption; certainly, I haven't). Yuret then makes two claims: first, that the only possible grammatically correct parses are those of the above form (eqn (6)) for which the relations $R_\alpha(w_j, w_k, t_m(w_j, w_k))$ have been previously observed; secondly, that there is a natural ranking of such allowed parses by summing the total mutual information associated with each word-pair.

These two concepts give rise to the idea of minimum-spanning-tree parsers. Such parsers work in a two-step process: a training phase, and a parse phase. In the training phase, one gathers a lot of statistics about mutual information. The important point here is that this is unsupervised training. To parse, one first creates a graph clique, with every word connected to every other. One uses the gathered MI to define graph edge lengths. Finally, the corrrect parse is then the maximum spanning tree of the graph (maximizing the MI, summed over the tree edges in the graph).

Here, we use the same idea, but then take the next step. The spanning tree can be decomposed into a set of link-grammar disjuncts, one disjunct per word. The disjunct is merely a list of the connections that one word makes. It consists of the type, and the direction. The direction is left or right. The type is the $t_m = t_m(w_j, w_k)$ defined above. By parsing a large number of sentences, we can now automatically discover a large number of disjuncts, in an unsupervised manner.

The goal, the next step, is then to reduce the total number of disjuncts, and the total number of types, by clustering and discovering similarities.

3 January 2014

No-crossing Minimum Spanning Trees

It turns out that writing an algorithm for a no-crossing minimum spanning tree is surprisingly painful; enforcing the no-crossing constraint requies treatment of a number of special cases. But perhaps this is not actually required! R. Ferrer i Cancho in "Why do syntactic links not cross?"[iC06] shows that, when attempting to arrange a random set of points on a line, in such a way as to minimize euclidean distances between connected points, one ends up with trees that almost never cross!

Other related references:

- Crossings are rare: Havelka, J. (2007). Beyond projectivity: multilingual evaluation of constraints and measures on non-projective structures. In: Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics

(ACL-07): 608-615. Prague, Czech Republic: Association for Computational Linguistics.

- Hubbiness is a better model of sentence complexity than mean dependency distance: Ramon Ferrer-i-Cancho (2013) "Hubiness, length, crossings and their relationships in dependency trees", ArXiv 1304.4086 — also states: maximum number of crossings is bounded above by mean dependency length. Also, mean dependency length is bounded below by variance of degrees of vertexes (i.e. variance in number of connectors a word can have).

- Language tends to be close to the theoretical minimum possible dependency distance, if it was legal to re-arrange words arbitrarily. See Temperley, D. (2008). Dependency length minimization in natural and artificial languages. Journal of Quantitative Linguistics, 15(3):256-282.

- Park, Y. A. and Levy, R. (2009). Minimal-length linearizations for mildly context-sensitive dependency trees. In Proceedings of the North American Chapter of the Association for Computational Linguistics - Human Language Technologies (NAACL-HLT) conference.

- Sentences with long dependencies are hard to understand: The original claim is from Yngve, 1960, having to do with phrase-structure depth. See – Gibson, E. (2000). The dependency locality theory: A distance-based theory of linguistic complexity. In Marantz, A., Miyashita, Y., and O'Neil, W., editors, Image, Language, Brain. Papers from the first Mind Articulation Project Symposium. MIT Press, Cambridge, MA.

- (Cite this, its good) Mean dependency distance is a good measure of sentence complexity – for 20 languages – Haitao Liu gives overview starting from Yngve. [Liu08]. Haitao Liu "Dependency distance as a metric of language comprehension difficulty", 2008, Journal of Cognitive Science, v9.2 pp 159-191 http://www.lingviko.net/JCS.pdf

- Sentences with long dependencies are rarely spoken: Hawkins, J. A. (1994). A Performance Theory of Order and Constituency. Cambridge University Press, Cambridge, UK. —-Hawkins, J. A. (2004). Efficiency and Complexity in Grammars. Oxford University Press, Oxford, UK. —-Wasow, T. (2002). Postverbal Behavior. CSLI Publications, Stanford, CA. Distributed by University of Chicago Press.

- Dependency-length minimzation is universal: Richard Futrell, Kyle Mahowald, and Edward Gibson, "Large-scale evidence of dependency length minimization in 37 languages" (2015), doi: 10.1073/pnas.1502134112

- The longest links, observed statistically, are of length 6 or less. This is based on computing the mutual information of words at different distances for the Brown corpus. Xuedong Huang, Fileno Alleva, Hsiao-wuen Hon, Mei-Y uh Hwang, Kai-Fu Lee and Ronald Rosenfeld. The SPHINX-II Speech Recognition System:

An Overview . Computer , Speech and Language , volume 2, pages 137–148, 1993.

So, rather than imposing no-crossing as a constraint on the parser, instead, let it find its own way into the grammar. Just implement a plain-old MST parser, punt on crossing.

11 January 2014

Clustering Redux

OK, so what is the very next algorithmic step? Up to here, we've generated a large number of unique disjuncts. Now what?

Back to counting. Lets do the French dictionary. The database fr_pairs contains table atoms_mi_snapshot. So:

- `select count(*) from atoms_mi_snapshot;` returns 415532

15 January 2014

Embodied Learning

OK, so maybe learning syntax before emantics puts the cart before the horse. Can we learn a world-model first, and then gradually annotate and correct it as our linguistic comprehension improves? So, for example, can we start with a world-model obtained via document summarization? How do we annotate this model with newly discovered data?

Related question: how to automatically discover ontologies? Automated, unsupervised concept, entity extraction? Semantic context change over time?

Steps:

1. How do I extract entities out of a text? The extraction doesn't have to be perfect; having candidate entities is enough. How do I put a confidence rating on the entiy, and how do I discard the low-cnfidence ones?

2. Once entities are extracted, I want to start decorating them with attributes (adjectives, modifiers), to build a network.

3. Once a network is built, it needs to be factually reconciled, using logical reasoning and an ontology (is-a and has-a relations). Need to do this so that upon reading "colorless green ideas", we can deduce that ideas are either colorless or green, but not both.

4. How to automatically extract an ontology from free text?

The above seem to be the central steps/core issues for creating a world-model, unsupervised, from text.

Entropy

Some refresher notes:

- "The Boltzmann distribution is the so-called canonical distribution, meaning it maximizes entropy subject to a constraint on the expected value of energy." (viz, this is the MaxEnt principle. Except for MaxEnt, the constraint is not on energy, but rather a set of constraints obtained on some other theoretical grounds.)
- Define "Shanon Entropy" as $S_s = -k_B \sum p \log p$
- The "Boltzmann Entropy" S_B is the shanon entropy of the microcanonical ensemble: it maximizes the entropy (MaxEnt) for a fixed value of the energy. (MaxEnt: not the energy, but for a fixed set of constraints). (viz, $S_B = k_B \log\left(\varepsilon \frac{d\Omega}{dE}\right)$ with Ω being number of states, E the energy, ε a constant of dimension energy to make arg of the log dimensionless.) (MaxEnt: replace E by the individual constaints. This suggests that there are many Boltzmann entropies: one for each constraint that is applied!)
- The "Gibbs Entropy" is the Shanon entropy, maximized for a system held to the constraint that energy is less-than-or-equal to E. (!) This gives $S_G = k_B \log \Omega$ (duhh, take $p = 1/\Omega$ for Ω states. For a non-sharp cutoff, the Shannon entropy is primal.). (MaxEnt: one gets a different Gibbs entropy for each applied constraint.)
- Gibbs and Boltzmann entropies give different results for N-particle systems when N is very small. Viz, an off-by-one error for N. In some ways, S_G is more correct (at low temp, quantum systems). See Jörn Dunkel and Stefan Hilbert (2014) "Consistent thermostatistics forbids negative absolute temperatures" Nature Physics DOI: 10.1038/NPHYS2815

Why does Yuret's MST work?

There is an interesting simplification that happens with minimum-spanning tree parsers driven by entropy. If we use Yuret's definition of the MI of word-pairs, then, Yuret says (I should re-read his stuff) that we should maximize the entropy

$$\sum_{w_l, w_r} MI(w_l, w_r) \tag{7}$$

Why? Why this, instead of the maybe "more obviously correct" sum:

$$\sum_{w_l, w_r} P(w_l, w_r) MI(w_l, w_r) \tag{8}$$

I think I can hand-wave the answer. The answer is that we don't really know the probability of $P(w_l, w_r)$ *for the given sentence!* We know $P(w_l, w_r)$ for a large corpus, but its somewhat of a mistake to assume that this identical to what it would be for expressing a particular idea in a certain specific way. Its possible that, to express the idea, the only sentences that one could ever possibly use would have $P(w_l, w_r)$ that strongly deviate from a large-corpus average. Unfortunately, there is no easy way of knowing what this sentence-specific $P(w_l, w_r)$ is. So, instead we make the uniform distribution assumption, that they're all the same, and thus get eqn (7) instead of (8). Does Yuret ever make this argument himself? Dunno.

A supporting argument is that we also ignored 3,4,5-point interactions as well. Which brings us to the next point: why should we expect a link-parse to work better than an MST parse? Because Yuret-MST ignores the valence of words, whereas the disjuncts don't! The disjuncts provide a better, more accurate way of capturing valency!

Entity Extraction

See Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates (2005) "Unsupervised Named-Entity Extraction from the Web: An Experimental Study". So: KNOWITALL utilizes a set of eight domain-independent extraction patterns to generate candidate facts. For example, the generic pattern "NP1 such as NPList2" ... "cities such as Paris,..."" Of course, this is not really unsupervised, since it uses human-generated search patterns ("such as") and also applies constraints (the targets must be proper nouns, which is not a-priori known).

Partition Function

Some notes about the Boltzmann distribution. Let $F(w_l, w_r)$ be a numeric score associated with the edge (w_l, w_r) – for example, this might be (minus) the MI. By convention, one introduces a Lagrange multiplier β and writes $F = \beta f$. The probability of a parse tree T constructed solely out of edge-pairs is then defined as

$$P(T|S) = \frac{\exp\left(-\beta \sum_{(w_l,w_r)\in T} f(w_l, w_r)\right)}{Z(S)}$$

Here, S is the sentence: a sequence of words, and $Z(S)$ is the partition function: the sum of the probabilities of all different possible parses for that setence:

$$Z(S) = \sum_T \exp\left(-\beta \sum_{(w_l,w_r)\in T} f(w_l, w_r)\right)$$

The MST parse is then the single parse T that maximizes the probability $P(T|S)$ and this can be easily seen to be the parse that maximizes the sum $\sum_{(w_l,w_r)\in T} f(w_l, w_r)$ on the spanning tree T.

Written in this form, it suggests how parsing can be generalized to include other scores. If, for example, one has some other score $g(w_1, w_2, w_3)$ over triples of words, then one sums as above, using g in place of f. In general, one can consider scoring functions $f = f(R; S)$ for some relation R over the sentence S. The prototypical example would be a scoring function that uses Link Grammar disjuncts for the relation

R, together with some weighting for link lengths (so as to avoid long links), together with some weighting for word pairs (to give greater weight to idioms and set phrases, as opposed to grammatically valid but non-idiomatic parses).

In many papers on supervised training, it is common to call the relation R a "feature", and to consider many different possible features R. Thus, $f = f_R(S)$ becomes a vector $\vec{f}$ over the features R. The multiplier $-\beta$ is replaced by a vector of weights $\vec{w}$, and so one considers the probability

$$P = \frac{\exp\left(\sum \vec{w} \cdot \vec{f}\right)}{Z(S)}$$

Supervised training then uses a training corpus marked up with both a feature vector $\vec{f}$ and the correct parse; training consists of using various supervised learning algos to find the best-possible weight vector $\vec{w}$ that maximizes the fraction of correct parses (or optimizes the ROC curve, or other measure of accuracy). In unsupervised training, we don't have a training corpus, and thus, do not focus on optimizing the weight vector.

Now we do the funky chicken dance. Write

$$Z(S) = \det L(S)$$

This is commonly done when working with fermions; this is the Berezin determinant or Berezin integral, so named because one may write

$$\det A = \int \exp\left[-\theta A \eta\right] d\theta d\eta$$

for Grassman variables θ and η and A a matrix. Here's the part that surprises me: Koo et al state that $L(S)$ can be taken to be a Laplacian matrix of a graph. Wow! The mind boggles.

References:

- Koo, T., Globerson, A., Carreras, X., and Collins, M. (2007). Structured prediction models via the matrix-tree theorem. In Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP- CoNLL) , pages 141–150, Prague, Czech Republic, June. Association for Computational Linguistics.

3 March 2014

Start again, after long distraction.

Finding patterns

To problem. Consider an alphabet of $N = 5$ letters, $\alpha = \{A, B, C, D, E\}$ and a corpus built from those letters. The five letters occur with probability $p(w)$ with $w \in \alpha$. Assume the corpus consists entirely of pairs AB, CB and DE, each occurring equally often: so: $p(A, B) = p(C, B) = p(D, E) = 1/3$. From this, we can reconstruct that

$p(A) = p(C) = p(D) = p(E) = 1/6$ and $p(B) = 1/3$. This follows because the corpus can be reduced to $\{AB, CB, DE\}$, so A occurs 1 out of 6 times, B two out of 6 times, etc. The total single-letter entropy is thus

$$\begin{aligned} h_{SING} &= -\sum_{w\in\alpha} p(w)\log_2 p(w) \\ &= -\frac{4}{6}\log_2\frac{1}{6} - \frac{1}{3}\log_2\frac{1}{3} \\ &= \frac{2}{3} - \log_2\frac{1}{3} = 2.25163 \end{aligned}$$

By contrast, in a random 2-letter corpus, we expect all possible letter pairs to occur equally often, i.e. $p(w) = 1/5$, which would result in $h_{RAND} = -\log_2 1/5 = 2.321928$ and so we see that the total entropy for this corpus is less than the random corpus.

The total double-word entropy is

$$\begin{aligned} h_{PAIR} &= -\sum_{w_1,w_2\in\alpha} p(w_1,w_2)\log_2 p(w_1,w_2) \\ &= -\log_2\frac{1}{3} = 1.5849625 \end{aligned}$$

Compare this to $h_{PR-RAND} = -\log_2 1/25 = 4.643856$ for the random 2-letter corpus. The pair-entropy is sharply lower.

What do we know about mutual information? We can also deduce that $p(*,B) = 2/3$, $p(A,*) = 1/3$ and so

$$\begin{aligned} MI(A,B) &= \log_2 p(A,B) - \log_2 p(A,*) - \log_2 p(*,B) \\ &= \log_2 3/2 = 0.585 \end{aligned}$$

and likewise $MI(C,B) = MI(A,B)$ while $MI(D,E) = \log_2 3 = 1.585$.

By contrast, in a random 2-letter corpus, we expect all possible letter pairs to occur equally often, i.e. $p(w_1,w_2) = 1/25$, which would result in an $MI(w_1,w_2) = \log_2 1 = 0$ for all word pairs.

Given this corpus, we wish to deduce the following answer: there is a cluster $\gamma = \{A,C\}$ and two link relations $R(\gamma,B)$ and $R(D,E)$ occuring with probabilities $p(\gamma,B) = p(A,B) + p(C,B) = 2/3$ and $p(D,E) = 1/3$. Note that $p(\gamma,*) = p(A,*) + p(C,*) = 2/3$ so that

$$\begin{aligned} MI(\gamma,B) &= \log_2 p(\gamma,B) - \log_2 p(\gamma,*) - \log_2 p(*,B) \\ &= \log_2 3/2 = 0.585 \end{aligned}$$

So how do we deduce this?

Well, consider the reduced space, with $N = 4$ letters: $\beta = \{\gamma, B, D, E\}$. In this space, only two pairs are observed in the corpus, γB and DE with probabilities as above. The single-letter probabilities are $p(D) = p(E) = 1/6$ and $p(\gamma) = p(B) = 1/3$.

The single-letter entropy is

$$\begin{aligned} h^{red}_{SING} &= -\sum_{w\in\beta} p(w)\log_2 p(w) \\ &= -\frac{2}{6}\log_2\frac{1}{6} - \frac{2}{3}\log_2\frac{1}{3} \\ &= \frac{1}{3} - \log_2\frac{1}{3} = 1.9182958 \end{aligned}$$

This can be compared to the entropy of the random 4-word corpus: $h^{red}_{RAND} = -\log_2 1/4 = 2$. Note that

$$h^{red}_{RAND} - h^{red}_{SING} = 0.081704 > 0.070298 = h_{RAND} - h_{SING}$$

In other words, the reduced corpus shows more order than the comparable unreduced corpus! Interesting! The above can be written as:

$$h_{SING} - h^{red}_{SING} = 0.333334 > 0.321928 = h_{RAND} - h^{red}_{RAND}$$

What about the reduced pair entropy? For this case, we have

$$\begin{aligned} h^{red}_{PAIR} &= -\sum_{w_1,w_2\in\beta} p(w_1,w_2)\log_2 p(w_1,w_2) \\ &= -\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3} \\ &= -\frac{2}{3} - \log_2\frac{1}{3} = 0.9182958 \end{aligned}$$

which can be compared to the random-pair entropy of $h^{red}_{PR-RAND} = -\log_2 1/16 = 4$. The comparable reduction is

$$h^{red}_{PR-RAND} - h^{red}_{PAIR} = 3.081704 > 3.0588935 = h_{PR-RAND} - h_{PAIR}$$

So again, this wins, but not by a lot. Re-ordering, this can be written as:

$$h_{PAIR} - h^{red}_{PAIR} = 0.6666667 > 0.643856 = h_{PR-RAND} - h^{red}_{PR-RAND}$$

So we seem to have two ways of winning: reducing the overall entropy, for for single letters, and for pairs, and also finding reductions that are strong, even compared to the reduced vocab.

Reductio ad absurdum? No.

What if we continue on this path, and (incorrectly) reduce to $N = 3$ letters, with $\delta = \{\gamma, \eta, D\}$ where $\eta = \{B, E\}$? Then $p(\eta) = p(B) + p(E) = 1/2$

$$\begin{aligned} h^{rr}_{SING} &= -\sum_{w\in\delta} p(w)\log_2 p(w) \\ &= -\frac{1}{6}\log_2\frac{1}{6} - \frac{1}{3}\log_2\frac{1}{3} - \frac{1}{2}\log_2\frac{1}{2} \\ &= \frac{2}{3} - \frac{1}{2}\log_2\frac{1}{3} = 1.4591479 \end{aligned}$$

and the reduction inequality is

$$h^{red}_{SING} - h^{rr}_{SING} = 0.4591479 > 0.4150375 = h^{red}_{RAND} - h^{rr}_{RAND}$$

So this inequality allows an inappropriate reduction to take place. That implies that we must not use the SING inequality to obtain reductions!

For the pairs, $p(\gamma,\eta) = p(\gamma,B) = 2/3$ and $p(D,\eta) = p(D,E) = 1/3$ and everything else being zero. Thus one gets:

$$\begin{aligned} h^{rr}_{PAIR} &= -\sum_{w_1,w_2\in\delta} p(w_1,w_2)\log_2 p(w_1,w_2) \\ &= -\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3} \\ &= -\frac{2}{3} - \log_2\frac{1}{3} = 0.9182958 \end{aligned}$$

so that

$$h^{red}_{PAIR} - h^{rr}_{PAIR} = 0 \not> 0.830075 = h^{red}_{PR-RAND} - h^{rr}_{PR-RAND}$$

Here, nothing is gained, so the pair inequality blocks the inappropriate reduction. Consider a diffent inappropraite reduction to $N = 3$: let $\varepsilon = \{\zeta, B, E\}$ with $\zeta = \{D,\gamma\}$ Then the pair probabilities are $p(\zeta,B) = p(\gamma,B) = 2/3$ and $p(\zeta,E) = p(D,E) = 1/3$ and again, there is no entropy reduction. The other groupings look to be equally ineffective.

Finding Patterns, General Formula

OK, recast the above section for the (semi-)general case of word-pairs (not structures in general). So, given a vocabulary of N words, we have $h_{RAND} = -\log_2\frac{1}{N} = \log_2 N$ and $h^{red}_{RAND} = \log_2(N-1)$ so that for large N, $h_{RAND} - h^{red}_{RAND} = \log_2 N/(N-1) = \log_2(1 + 1/(N-1)) \approx 1/N$ and so we have a word-combine winner if we can combine words A and C into a cluster $\gamma = \{A,C\}$ such that

$$\begin{aligned} \frac{1}{N} &\lesssim h_{SING} - h^{red}_{SING} \\ &= -\sum_{w\in\alpha} p(w)\log_2 p(w) + \sum_{w\in\beta} p(w)\log_2 p(w) \\ &= -p(A)\log_2 p(A) - p(C)\log_2 p(C) + p(\gamma)\log_2 p(\gamma) \\ &= p(A)\log_2\left(1 + \frac{p(C)}{p(A)}\right) + p(C)\log_2\left(1 + \frac{p(A)}{p(C)}\right) \end{aligned}$$

where $p(\gamma) = p(A) + p(C)$. What's not clear: is this inequality *ever* broken? Or does it always hold? At any rate, from the previous example, it seems clear that we should not use the SING inequalities to obtain clusters.

For pairs, its clear that $h_{PR-RAND} - h^{red}_{PR-RAND} \approx 2/N$ which follows as above, given that $h_{PR-RAND} = 2\log_2 N$, etc. The corresponding inequality is now

$$
\begin{aligned}
\frac{2}{N} \lesssim & h_{PAIR} - h^{red}_{PAIR} \\
= & -\sum_{w_1,w_2\in\alpha} p(w_1,w_2)\log_2 p(w_1,w_2) + \sum_{w_1,w_2\in\beta} p(w_1,w_2)\log_2 p(w_1,w_2) \\
= & -\sum_{w\in\alpha\setminus A,C} [p(A,w)\log_2 p(A,w) + p(C,w)\log_2 p(C,w) - p(\gamma,w)\log_2 p(\gamma,w)] \\
& -\sum_{w\in\alpha\setminus A,C} [p(w,A)\log_2 p(w,A) + p(w,C)\log_2 p(w,C) - p(w,\gamma)\log_2 p(w,\gamma)] \\
& \quad - p(A,A)\log_2 p(A,A) - p(C,A)\log_2 p(C,A) + p(\gamma)\log_2 p(\gamma) \\
& \quad - p(A,C)\log_2 p(A,C) - p(C,C)\log_2 p(C,C)
\end{aligned}
$$

So...

8 March 2014

Morphology

Notes: https://en.wikipedia.org/wiki/Nonconcatenative morphology

25 March 2014

Information-Theoretic Clustering

New references:

- http://www.cs.utexas.edu/users/inderjit/public_papers/kdd_cocluster.pdf Information-Theoretic Co-clustering Inderjit S. Dhillon, Subramanyam Mallela, Dharmendra S. Modha
- http://pdf.aminer.org/000/472/364/a_generalized_maximum_entropy_approach_to_bregman_co_clustering_and.p A Generalized Maximum Entropy Approach to Bregman Co-clustering and Matrix Approximation Arindam Banerjee, Inderjit Dhillon, Joydeep Ghosh, Srujana Merugu, Dharmendra S. Modha Journal of Machine Learning Research 8 (2007) 1919-1986

30 March 2014

The below was going to be a brief note, but I'm turning it into a rough draft blog post. But after sleeping on it, it seems silly.

Freedom and Constraint

The concepts of freedom and constraint are central to the definition of algebra in mathematics. So for example, in group theory, the algebraic symbols denoting the elements of the group may be arranged freely, in any order desired. A given group is then defined as a 'presentation', a set of equivalences between different orderings. Thus, there is the notion of a 'free group', which is merely a set of symbols that can be written in arbitrary order, and no further constraints other than those of it being a group. Groups that aren't free are presented by a collection of equations, which state that one certain order of symbols is equivalent to another. One says that groups are 'equationally presented'.

A more complex example is the term algebra, where the terms may be arranged in free order; but the combination of the written symbols on the page are constrained to those of the 'signature' of the algebra. One then has the notion of an 'equational theory', which is a term algebra with additional equations between expressions, indicating which expressions should be taken as equivalent.

These have strong, and even precise analogues in linguistics. But first, continuing with the mathematical observations: the signature of a term algebra can be viewed as defining the 'syntax' of the symbolic notation: a Turing machine, tasked with the need to recognize the 'language' of the term algebra, would process input symbols one by one. It would appear that term algebras have a context-free syntax, and are thus recognizable by a push-down automata. That is, one must recognize the function symbol, the open and close parens, the commas separating arguments, and the constant symbols. The arbitrary-depth recursiveness is the only reason why the push-down is needed; otherwise the language seems 'almost regular'. (Hmm ... is there any formal definition/distinction of this case? i.e. for very simple constext-free languages, vs. 'more complex' ones? Not that I know of ...)

In linguistics, similar notions of freedom and constraint arise, but seem to be more of a surprise and mystery to linguists. Thus, for example, in [And12], Anderson describes the syntax and morphotactics of Kwakw'ala, a Wakashan language of coastal British Columbia. The syntax of the language (that is, the order in which the words can appear in a sentence) is very strict: the verb must be followed by a subject, optionally followed by the object, and then a prepositional phrase. Similarly adjectives must always preceed the noun. The language also has a rich morphology: words are assembled from stems and suffixes. The rules for assembling a word out of stems and affixes is refered to as the 'morphotactics'. In Kwakw'ala, it would appear that the morphotactics is utterly distinct from the syntax: here, object-denoting prefixes can preceed verbs, adjective-denoting suffixes follow a noun. Anderson finds this quite remarkable: the language has two distinct kinds of structure-imposing systems: the syntax and the morphtactics, and they are quite different. He notes that this dual structure in turn allows the same thing to be said in multiple ways. One may take meaning-parts, as morphemes, and glue them together morphtactically into words, and aranging these in a sentence. Allternately, one may take the meaning-parts separately, as individual words, and glue them together into a sentence, having a different sequence of the meaning-parts.

The part that struck me with Anderson's analysis is the similarity of the phenomena to the analogous behaviour formalized in mathematics. Lets first look at a second example: Lithuanian has a rich morphotactical structure: verbs and adverbs are conjugated, nouns and adjectives are declined; the rules for doing so are rather fixed and uniform, making adjustments mostly for phonological reasons (i.e. with exceptions based on constraints that come from the natural flow of the sound sequences constrained by the use of vocal cords, mouth, tongue and lips). Curiously, Lithuanian is almost devoid of syntactic constraint: word-order can be chosen freely (in the mathemaitcal sense!), and the meaning of the resulting sentences are esentially the same (if I am allowed to gloss over the notion that different word orders can serve to highlight or emphasize different themes and rhemes). So again: a language with very distinct syntax and morphtactics; in this case, the syntax being almost absent.

I used the theory of Link Grammar for performing structural linguistic analysis. The theory was originally developed to model syntactic structure, but it also appears to be entirely adequate for morphotactic analysis as well (certainly, for 'agglutinative' or 'concatenative' languages, with ongoing research into more complex morphologies). From the point of view of a linguist, Link Grammar appears to be 'just another theory of syntax', being a kind of dependency grammar. From the point of view of a mathematician, the situation is entirely more remarkable. It appears that the mathematical defintion of what constitutes a 'link grammar' is isomorphic to that of a 'categorical grammar', and that the correspondance is immediate and direct. Categorical grammars are interesting because they have a direct, formal mathematical definition that is studied and classified by mathematicians: roughly speaking, categorical grammars are 'non-symmetric compact closed monoidal categories'. The precise definition here has been championed by Bob Coecke ref [xxx]

It takes some study of category theory to understand what this means, but, roughly speaking, it means that sequences of sounds, morphemes, words are analyzed in sequential order: by means of short-distance groupings of left-right arrangements. This may sound silly, as, of course, sequential things occur in a sequence, but it helps highlight the difference between dependency grammars and phrase-structure grammars, or computer-science grammars in general. An example of a 'computer-science grammar' is the so-called 'context-free grammar'. A hallmark of such grammars is that they allow recursion to arbitrary depths. An English-language example would be the sequence of sentences: "This is a house", "This thing is a house." "This thing is a thing that is a house." "This thing is a thing that is a thing that is ... a house." The example is silly because no one ever talks that way. The phrase-structure analysis of this would be "(S (VP (NP (VP (NP ...)))))", with the heirarchical arrangement emphsized. Dependency grammars can also parse such sentences, but here, the arrangment of dependencies are in the form of arrows that point from head word to dependent word; the arrows are only rarely long-range, and usually point to the immediately-surrounding words. There is strong psycho-linguistic evidence for such local structure in language, see for example [xxxx]. That is, the workings of the human mind is not recursive in nature, pushing and popping an arbitrarily deep stack as each new noun-phrase or verb-phrase is enountered. Indeed, psychological studies with constructed sentences similar to the above, but varying the 'thing' and 'house' at each depth, show that humans quickly loose track after just two or three nestings [need ref]. In essence, the human mind is adapted for linear sequential analysis, and long-range order between words is challenging: this is the psycho-linguistic argument for dependency grammars. From the mathematical point of view, the statement is that human languages are not so much context-free, as they are non-symmetric compact closed monoidal categories. That Link Grammar is an example of the latter is why it seems so appropriate to use for syntactic and morpho-tactic structural analysis.

Which theories of langauge are mathematically isomorphic? That is, Link Grammar and categorical grammars seem to be isomorphic because there is a simple way of translating the one into the other, and vice-versa (although no formal mathematical proof of this has been written down). A mathematical proof of equivalence is a mechanical device: given one representation, one turns a crank to obtain the other. More generally, its been argued that phrase-structure grammars and dependency grammars are equivalent in the same sense: there is an algorithm that converts the one into the other, and v.v.[where's the ref for this?]. Does this mean that non-symmetric compact closed monoidal categories have context-free grammars as their internal language, and that every context-free language has a corresponding monoidal category? I think not, but the answer to that, the 'why not', and the 'what, then, is the difference?' is entirely unclear. Clarifying these relationships seems important for putting language study on a firmer basis.

Anyway, the point here was to clarify the boundaries between freedom and constraint. Traditional phrase-structure grammars were inspired by notions from 1960's-era computer science, but now seem slavishly wedded to the same ideas, to the detriment of closer linguistic understanding. Dependency grammars seem to be more psycho-linguistically valid, but have suffered from a lack of mathematical formalism that ellucidates freedom and constraint. This lack of formalism makes it hard to explain why some constructions are grammatically correct, and others are not. It also seems to draw an artificial and confusing line between syntactic and morphotactic structure, when, in fact, these really should be taken as a part of a continuum of structure. I see no reason why a single grammar could not also describe the allomorphic variations in pronunciation. After all, these are just a set of rules that govern how a morpheme is pronounced, and this is essentially a linear, sequential phenomenon, with only (mostly?) nearest-neighbor morphemes affecting one-another. The nearest-neighbor aspect of this fairly well screams out 'dependency'.

Another curious and interesting language-constraint structure emerges with the study of idioms and institutionalized, set phrases. Because these are 'phrases', built of 'words', it would naively seem that tese lie in the domain of syntax. But this is misleading. Institutionalized utterances are those where neither the word-choice nor the word-order are directly governed by syntax alone, but instead seem to be frozen into a fixed form. So, one talks of the 'time of day', but never of 'pressure of air' or 'height of mountain' – “What pressure of air should I put in this tire?” “What height of mountain do you plan to climb?” “What time of day do you expect to come over?”. There is nothing syntactic that prevents such a choice of wording, and the semantic meaning is more or less clear: its just that such word arrangements simply don't happen. Its as if the lexis for English has a phrase in it: “time-of-day”, which should be treated as a single word, rather than the three words it is written as. This provides the first hint of the role of probability in this discussion: the probability of seeing the phrase 'height of mountain' in English approaches zero: in fact, this text that you are reading right now just might be the only place ever in the history of the world in which this phrase has appeared ... despite it being 'grammatically valid'. Freedom and constraint aren't just governed by true-false distinctions, but by probabilities. The question then is, 'what is the most natural way in which to express such probabilities?'

The last is not just some idle intellectual question, but in fact, an engineering question: the proper structure should have an immediate and direct effect on how well, and how quickly a language could be learned, via unsupervised machine-learning algorithms. A universal but naive attitude in the artificial-intelligence community is that 'oh. everything is a neural net, and we should use neural nets to build AI.' Less frequently, one may seem a similar attitude regarding Hidden Markov Models (HMMs). The fact that such naive approaches lead to algorithms that fail to converge quickly leads to ideas such as 'deep learning': a modification that explicitly splits a problem into layers, with explicit feedback between layers. Another variation used to escape the trap is to explicitly model what is un-known: this is the notion of maximum entropy (MaxEnt). Traditional AI was also founded on logic and reasoning, and, for many decades, AI was dominated by the exploration of boolean-valued logic. By this I mean anything with crisp, sharp truth values: whether first-order logic, boolean satisfiability, satisfiability-modulo-theories, stable-model semantics, and so on. Another corner was fuzzy logic, but that didn't seem to have legs. Notions of maximum entropy and probability can be unified: thus, one has Markov Logic Networks (MLN). What I'm wodering about here is that maybe none of these approaches are correct, because they are ignoring the actual structure that is in front of us.

So, perhaps, the correct approach is not to marry maximum entropy with first-order logic, but to marry maximum entropy to dependency grammars (or, equivalently, to appropriate monoidal categories). The question then becomes: what is the appropriate monoidal category? Picking the wrong one will lead to disasterous machine learning performance (this, I think, is the lesson from neural networks). Picking something too easy doesn't get you far enough (the lesson of HMM's – excellent for certain classes of problems, but lacking in scale). There are more choices than that: but the chices, and their inter-connectedness, and trade-offs, seem to be unarticulated. For any given monoidal category, there would seem to be some probabilistic model corresponding to that category's internal langauge. That is, there is a way of describing the transition probabilities from state to state. Indeeded, (finite) monoidal categories, in the form of acts, can be partly understood to be finite state machines acting on a set. The probabilistic generalization of this leads both to probabilistic and quantum finite automata, with the former having a strong resemblance, if not identity, to Markov chains, with the corresponding acts being HMM's. My hypothesis is that probabilistic dependency grammars will lead to machine learning algos that converge more rapidly than the similar-but-different HMM that can also be mapped onto the same problem. Unfortunately, my hypothesis is impedded by my lack of understanding of precisely, exactly how the different approaches named above may be equivalent, isomorphic, or merely similar.

2 April 2014

Link Grammar and Finite State Transducers

Claim: Finite state transducers, such as those used for morphological analysis, can be mapped to a Link Grammar. This implies that Link Grammar parsing can be used for morphological analysis, thus unifying syntactic parsing and morphological analysis into a unified framework.A finite state transducer (FST) is defined as:

- A set of states Q
- A set Σ of input symbols (surface form)
- A set Γ of output symbols (lexicalized form)
- A transition function $\delta \subset Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \times Q$

A member $(r,a,b,s) \in \delta$ should be thought of as the arrow from state r to state s, the arrow being taken when the input symbol is a and as a result producing the output symbol b. The corresponding link-grammar dictionary entry for this would be

```
a.b: r- & s+;
```

This states that no linkage is possible, unless the previous link resulted in the emission of the r+ connector. No transition to the next state is possible, unless that state has an s- connector on it.

The current link-grammar notation a.b is awkward for printing, and perhaps some new style is needed to distinguish the output to be printed from the input that is recognized. Thus, perhaps, it would be better to invent a new notation, perhaps a$b to denote that a is recognized, and that b is printed.

Note that the above definition of link-grammar rules results in a very simple, linear linkage: state transitions follow one-another in linear order. Link grammar allows richer, more complex linkage diagrams, and so the question arises: can a given FST be compactified into a smaller system by making use of the richer possibilities that link-grammar offers? How can this compactification be acheived?

Suppose that the FST δ includes as a subset the state transitions $\{(r,a,?,s),(s,\varepsilon,\varepsilon,t),(s,b,?,t),(t,c,?,u)\}$. The symbol ? is used here as a don't-care state, as it is irrelevant to the discussion that follows. The above state transitions indicates that when the system is in state s, it may spontaneously transition to state t, or may do so upon reading b. That is, the presence of b is optional in the state transition. The "natural" way of indicating this with link-grammar notation is using the link-grammar dictionary entries:

```
a: r- & s+;
b: t+;
c: {t-} & s- & u+;
```

Because the transition $(s,\varepsilon,\varepsilon,t)$ reads no input, and produces no output, the state transitions would more likely be written as $\{(r,a,?,t),(r,a,?,s),(s,b,?,t),(t,c,?,u)\}$, that is, by collapsing the transition $(s,\varepsilon,\varepsilon,t)$ into the prior state. This would have the entries

```
a: r- & (s+ or t+);
b: s- & t+;
c: t- & u+;
```

How should it be understood? These are, in fact, two distinct, inequivalent LG grammars, as can be seen by considering the parse of the strings “ac” and “abc” for the two cases.

When would weighting schemes interfere? when would output interfere?

15 April 2014

Elegant Normal Form

Or, more precisely, “Minimal Normal Form”. Instead of writing out LG disjuncts in long strings of DNF or CNF, where they blow up into the thousands or tens of thousands, we really need to write then in Craig Holman’s "Elegant Normal Form", (http://www.patterncraft.com/Blog/Blog-080609.html#ElegantNormalForm) format. This is to be done by entropy minimization, in two different ways: first, ENF reduces the total count of terms, for just one single expression. Second, and maybe more important: different words will share significant subsets of the ENF expression. So, for example, the LG English dicts define:

```
<verb-rq>: Rw- or ({Ic-} & Q- & <verb-wall>) or [()];
```

which is (1) in ENF, not DNF or CNF, and (2) shared by several dozen words. There should be a strong push to discover such common sub-expressions across many words.

28 April 2014

Isotopy

The concept of “isotopy” (https://en.wikipedia.org/wiki/Isotopy_(semiotics)) was introduced by Algirdas Greimas in 1966. Example: “I drink some water”, with the meanings of “drink” and “water” re-inforcing one-another. But this is eactly what the Mihalcea WSD algo does, eh?

14 May 2014

Tree similarity

“Similarity Evaluation on Tree-structured Data” Rui Yang Panos Kalnis Anthony K. H. Tung SIGMOD 2005 June 14-16, 2005. Quote from abstract: “propose to trans- form tree-structured data into an approximate numerical multidimensional vector”. Funny – that’s what Bob Coecke proposes for any kind of monoidal category: vector spaces being a special monoidal cat. Hmmm.

Approaches:

- Tree-edit distance: many variants proposed, all high cpu/memory intensive.
- convert tre to pre or post-oerder, and use string edit distance.
- Convert to binary tree. For combo trees, this makes sense, due to the associative property of most of the operators. In particular, in combo any oper that can have multi-sibilings is also associative and thus convertible to binary tree. What's more, trees with binary branch distance of zero really are equivalent for us: See Figure 4 in above reference. Yay! this fits very very well with combo!.

29 June 2014

Morphology Basic Claims

We have two tasks to adress: the automated discovery of morpheme boundaries, and the automated discovery of "morphtactics", the syntax of connected morphemes. We make two claims: first. the automated discovery of morpheme boundaries can be accomplished by searching for breaks between word-parts that have the lowest mutual information. Second, the discovery of morphotactics is identical to the discovery of syntax, as outlined above.

The simplest approach to finding the breaks between morphemes is to randomly break up words into two parts. A worked example of this is given below. Several questions present themselves:

- To discover morphemes of words that split into three or more parts, is it better to always split pairwise, and then perform recursion, or is it easier to split into multiple parts immediately? Perhaps the answer is language-dependent?
- Does one obtain better mophological splits by immediately including morphtactic analysis, or can this be deferred?

Morphology Worked Example

OK, this will be tedious, but I see no alternative. Suppose we have the corpus "test gift tester testy gifty tester gifter" so that "tester" appears twice in the corpus. Explore all possible splits into two parts. The 4-letter splits split 3 ways, the 5-letter splits split 4 ways, etc. so there is a total of N(*,*)=3+3+5+4+4+5+5=29 pairs. All pairs appear once, except for tester, which appears twice. Viz.

P(x,y)=1/29 for (x,y) in {(t,est), (te,st), (tes,t), (g,ift), (gi,ft), (gif,t), (t,esty), (te,sty), (tes,ty), (test,y), (g,ifty), (gi,fty), (gif,ty), (gift,y), (g,ifter), (gi,fter), (gif,ter), (gift,er), (gifte,r)}

and

P(x,y)=2/29 for (x,y) in {(t,ester), (te,ster), (tes,ter), (test,er), (teste,r)}

Table 1: Word Split Table

	g	gi	gif	gift	gifte	t	te	tes	test	teste	row total
ifter	1										1
fter		1									1
ter			1					2			3
er				1					2		3
r					1					2	3
ifty	1										1
fty		1									1
ty			1					1			2
y				1					1		2
ift	1										1
ft		1									1
t			1					1			2
ester						2					2
ster							2				2
esty						1					1
sty							1				1
est						1					1
st							1				1
column total	3	3	3	2	1	4	4	4	3	2	29

The above is a sparse matrix showing the possible word splits. empty cells contain a count of zero.

There is a bit of a procedural error in the above; we would like to discover the "null suffix", that is, that "test", "gift", with nothing following it, are morphemes, so that the possible suffixes are "-y", "-er" and "-nothing". However, the above failed to count this possibility separately. Thus, given the above data, what we expect to find are two roots: "gif-" and "tes-" and three suffixes: "-t", "-ty" and "-ter". This is not so bad. If we did split and count in such a way as to allow a null suffix, it would be ambiguus as to whether the stems end with a "t" or not. That is, the with-t and without-t stems would have been equally likely... Anyway, moving on... the possible splits are shown in the table below 1:

Next, lets do the partial sums. Recall the notation for the partial summation of pairs. writing P(x,y) for the probability of observing the *ordered* pair of items (x,y), the partial sums are:

$$P(x,*) = \sum_{y\in Y} P(x,y)$$

and

$$P(*,y) = \sum_{x\in X} P(x,y)$$

The left-hand sums are the column totals in the table above, table 1

P(t,*) = (1+1+2)/29 = 4/29 = P(te,*) = P(tes,*)

P(g,*) = (1+1+1)/29 = 3/29 = P(gi,*) = P(gif,*)

P(test,*) = (1+2)/29 = 3/29

P(teste,*) = 2/29

P(gift,*) = 2/29

P(gifte,*) = 1/29

Next, the right-hand partial sums. These are the row totals for the table above, table 1:

P(*,est) = 1/29 = P(*,st) = P(*,esty) = P(*,sty) =P(,ift) = P(*,ft) = P(*,ifty) = P(*,fty) = P(*,ifter) = P(*,fter)

P(*,t) = (1+1)/29 = 2/29 = P(*,ty) = P(*,y)
P(*,ester) = 2/29 = P(*,ster)
P(*,ter) = (1+2)/29 = 3/29 = P(*,er) = P(*,r)

Now, the compute the MI (we use log=log_2 in all cases below, for measuring the entropy in units of bits). Recall the definition of mutual information for *ordered* pairs, previously discussed and given above:

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

So, working these by hand:

MI(t,est) = log P(t,est)/P(t,*)P(*,est) = log (1/29)(29/4)(29/1) = log(29/4) = 2.857981 = MI(te,st) = MI(t,esty) = MI(te,sty)

MI(g,ift) = log P(g,ift)/P(g,*)P(*,ift) = log (1/29)(29/3)(29/1) = log(29/3) = 3.273018 =MI(gi,ft) = MI(g,ifty) = MI(gi,fty)=MI(g,ifter)=MI(gi,fter)

MI(tes,t) = log P(tes,t)/P(tes,*)P(*,t) = log (1/29)(29/4)(29/2) = log(29/8) = 1.857981 =MI(tes,ty)

MI(gif,t) = log P(gif,t)/P(gif,*)P(*,t) = log (1/29)(29/3)(29/2) = log(29/6) = 2.273018

MI(test,y) = log P(test,y)/P(test,*)P(*,y) = log (1/29)(29/3)(29/2) = log(29/6) = 2.273018

MI(gif,ty) = log P(gif,ty)/P(gif,*)P(*,ty) = log (1/29)(29/3)(29/2) = log(29/6) = 2.273018

MI(gift,y) = log P(gift,y)/P(gift,*)P(*,y) = log (1/29)(29/2)(29/2) = log(29/4) = 2.857981

MI(gif,ter) = log P(gif,ter)/P(gif,*)P(*,ter) = log (1/29)(29/3)(29/3) = log(29/9) = 1.688056

MI(gift,er) = log P(gift,er)/P(gift,*)P(*,er) = log (1/29)(29/2)(29/3) = log(29/6) = 2.273018

MI(gifte,r) = log P(gifte,r)/P(gifte,*)P(*,r) = log (1/29)(29/1)(29/3) = log(29/3) = 3.273018

MI(t,ester) = log P(t,ester)/P(t,*)P(*,ester) = log (2/29)(29/4)(29/2) = log(29/4) = 2.857981
= MI(te,ster)

MI(tes,ter) = log P(tes,ter)/P(tes,*)P(*,ter) = log (2/29)(29/4)(29/3) = log(29/6) = 2.273018

MI(test,er) = log P(test,er)/P(test,*)P(*,er) = log (2/29)(29/3)(29/3) = log(58/9) = 2.688056

MI(teste,r) = log P(teste,r)/P(teste,*)P(*,r) = log (2/29)(29/2)(29/3) = log(29/3) = 3.273018

Phew. I think that's all of them. So, what can we conclude? The basic claim is that the morpheme boundaries occur at the places where the letters are the least sticky, the most likely to be de-correlated, i.e. those with the lowest MI. In the above, these are: MI(gif,ter)=1.69 followed by MI(tes,t)=MI(tes,ty)=1.86. These are the most likely splits for these three words. Lets look up each possible split, for each word. We get:

Word	Split	MI	Split	MI	Split	MI	Split	MI	Split	MI	Best
gift	(g,ift)	3.27	(gi,ft)	3.27	(gif,t)	2.27					(gif,t)
gifty	(g,ifty)	3.27	(gi,fty)	3.27	(gif,ty)	2.27	(gift,y)	2.86			(gif,ty)
gifter	(g,ifter)	3.27	(gi,fter)	3.27	(gif,ter)	1.69	(gift,er)	2.27	(gifte,r)	3.27	(gif,ter)
test	(t,est)	2.86	(te,st)	2.86	(tes,t)	1.86					(tes,t)
testy	(t,esty)	2.86	(te,sty)	2.86	(tes,ty)	1.86	(test,y)	2.27			(tes,ty)
tester	(t,ester)	2.86	(te,ster)	2.86	(tes,ter)	2.27	(test,er)	2.69	(teste,r)	3.27	(tes,ter)

The best results from the above table are summarized below

Word	Lowest MI split(s)	MI
gift	(gif,t)	2.27
gifty	(gif,ty)	2.27
gifter	(gif,ter)	1.69
test	(tes,t)	1.86
testy	(tes,ty)	1.86
tester	(tes,ter)	2.27

What looks like the best split has been found; it certainly matches what was expected. Yay! After this, link-type clustering proceeds just as before, as if these were distinct words. That is, the above has 6 distinct link types; clustering will then proceed discover one link type, between the cluster {gif, tes} and {t,ty,ter}.

Morfesssor

An alternative algorithm is presented in:

- Mathias Creutz Krista Lagus, "Unsupervised Morpheme Segmentation and Morphology Induction from Text Corpora Using Morfessor 1.0" http://users.ics.aalto.fi/mcreutz/papers/Creutz05tr.pdf

That algorithm works only for concatenative languages, and does not provide a morphotactic structure; that is, it cannot learn the grammar governing the morphemes. It also requires several (plausable) assumptions about Bayesian priors. One assumption is that morpheme frequency follows a modified Zipfian distribution, this is used to make estimates for morphemes that are observed only once in the corpus. Another assumption is is that the morpheme length distribution can be approximated by either a Poisson or a (two-parameter) gamma distribution.

12 July 2014

Link-type discovery, worked example

In keeping with the previous, lets look at a super-simplified version of link-type discovery, continuing imediately from the previous morpheme-discovery example. We begin with the intial observations, given in the table below:

Pair	Initial Link Type	# observations
gif–t	GA	1
gif–ty	GB	1
gif–ter	GC	1
tes–t	TA	1
tes–ty	TB	1
tes–ter	TC	2

The "initial link type" is handed out randomly; the actual letter string has no bearing on the outcome. Notice the above has 6 different, unique link types. These correspond to the following link-grammar dictionary, written in the classic link-grammar notation:

Algorithm 1 Morpheme grammar

```
gif.=: GA+ or GB+ or GC+;
tes.=: TA+ or TB+ or TC+;
=t: GA- or TA-;
=ty: GB- or TB-;
=ter: GC- or TC-;
```

From the above initial dictionary, we want to deduce that a single link type is sufficient to full describe what is happening. That is, we wish to discover the following dictionary:

```
gif.= tes.=: LL+;
=t =ty =ter: LL-;
```

This is inuitively obvious, because the morphemes obviously form a clique: each stem has been observed with each suffix. Technically, this is a bipartite clique or complete bipartite graph of order (2,3). Here, we see it immediately; however, in general, it is very hard to search for bipartite cliques in a grammar; general algorithms are provably NP-complete and run in exponential time.

So how should we find grammar reductions? How is this to be done?

Out vocabulary consists of N=5 morphemes α ={gif.=, tes.=. =t, =ty, =ter}. We begin by recomuting the MI for observed pairs, once-again starting with the initial corpus “test gift tester testy gifty tester gifter”, same as before, with “tester” appearing twice in the corpus. This time, se split strictly according to the learned morphology. The word split table is:

	gif	tes	row total
ter	1	2	3
ty	1	1	2
t	1	1	2
column total	3	4	7

Note that this table is a stric subset of the previous table; the column and row totals are completely unchanged. However, the total number of observations has diminished from 29 to 7, and so all P amd MI values need to be recomputed. Proceeding long-hand, as before:

P(x,y)=1/7 for (x,y) in {(tes,t), (gif,t), (tes,ty), (gif,ty), (gif,ter)}

and

P(x,y)=2/7 for (x,y) in {(tes,ter)}

The partial sums are:

P(gif,*) = (1+1+1)/7 = 3/7

P(tes,*) = (1+1+2)/7 = 4/7

P(*,t) = 2/7 = P(*,ty)

P(*,ter) = 3/7

The MI values are all different, as well:

MI(gif,t) = log P(gif,t)/P(gif,*)P(*,t) = log (1/7)(7/3)(7/2) = log (7/6) = 0.222392
= MI(gif,ty)

MI(gif,ter) = log P(gif,ter)/P(gif,*)P(*,ter) = log (1/7)(7/3)(7/3) = log (7/9) = -0.362570

MI(tes,t) = log P(tes,t)/P(tes,*)P(*,t) = log (1/7)(7/4)(7/2) = log (7/8) = -0.192645
= MI(tes,ty)

MI(tes,ter) = log P(tes,ter)/P(tes,*)P(*,ter) = log (2/7)(7/4)(7/3) = log (7/6) = 0.222392

Note that three of the MI values are negative, and three are positive.

Following the previous formulas, we compute the total pair entropy:

$$
\begin{aligned}
h_{PAIR}^{observed} &= -\sum_{w_1,w_2\in\alpha} p(w_1,w_2)\log_2 p(w_1,w_2) \\
&= -\frac{5}{7}\log_2\frac{1}{7} - \frac{2}{7}\log_2\frac{2}{7} = 2.521641
\end{aligned}
$$

This is a bit of a misnomer, or misleading; we are actually computing the link-entropy: so the set is actually $\beta = \{GA, GB, GC, TA, TB, TC\}$ the first five of which were observed once, and the last was observed twice. So really we should write:

$$
h_{PAIR}^{observed} = -\sum_{t\in\beta} p(t)\log_2 p(t)
$$

with $p(t)$ being the probability of observing link-type t.

The above is the observed entropy, given the corpus, and the grammar shown in listing 1. However, this grammar does not have any probability indicators attached to it, so that if it was used to generate a corpus, the entropy would be different. Basically, the probability of observing any of the link-types would be identical, and so the entropy would be:

$$\begin{aligned} h_{PAIR}^{generated} &= -\sum_{t\in\beta} p(t)\log_2 p(t) \\ &= -\frac{6}{6}\log_2\frac{1}{6} = 2.584963 \end{aligned}$$

This is obtained by observing that there are 6 link types in the set β and so, if chosen equi-probably, the resulting entropy is just $\log_2 6$. For a given number N of link types, the entropy of the generated grammar will be $\log_2 N$, for this extremely simply type of grammar, where all disjuncts have only one connector in them. The generated entropy will always be maximal for the grammar, as the observed distribution will surely never be equi-distributed. Thus, we have as a general principle:

$$h^{observed} \leq h^{generated}$$

Note that the equi-distributed link-types is the same as having each of the words in the corpus appear with equal frequency. The morphemes, however, do NOT appear with equally frequency (although individually, all stems do, and all suffixes do).

Link type reductions can be many ways. In each case, we look to see if adding a new word to a category improves the score. The possibilites are:

1. Group =ter and =ty together.
2. Group =ter and =t together.
3. Group =ty and =t together.
4. Group gif.= and tes.= together.

After this, we have more reductions:

1.a. Add =t to {=ter, =ty}, and finally group together gif.= and tes.=
1.b. Group together gif.= and tes.=, and finally, add =t to {=ter, =ty}
2.a, 2.b. 3.a 3.b variations of above
4.a. Group =ter and =ty together, then add =t.
4.b. Group =ter and =t together, then add =ty.
4.c. Group =ty and =t together, then add =ter.

This gives 9 different orders in which the reductions can take place. Actually, only 6: case 1b and 4a are the same, as are 2b=4b and 3b=4c. Lets do at least some of them.

Case 1. Let γ = {=ter, =ty}. Then the link types GB and GC need to be consolidated: GG={GB, GC} and likewise TT={TB, TC}. The dictionary becomes

```
gif.=: GA+ or GG+;
tes.=: TA+ or TT+;
=t: GA- or TA-;
=ty =ter: GG- or TT-;
```

The observed pair probabilities become:

p(GA) = p(gif,t) = 1/7 = p(TA) = p(tes,t)

p(GG) = p(gif,ter) + p(gif,ty) = 2/7

p(TT) = p(tes,ter) + p(tes,ty) = 3/7

So the observed entropy is now

$$h_{PAIR}^{red1.} = -\frac{2}{7}\log_2\frac{1}{7} - \frac{2}{7}\log_2\frac{2}{7} - \frac{3}{7}\log_2\frac{3}{7} = 1.842371$$

The generated entropy is $h_{gen}^{red1.} = \log_2 4 = 2$ since there are four total link types in the reduced grammar. Pursuant to equation 5, we should add the log of the cardinality of the word-sets. Here, only one word-set has a cardinality greater than one: {=ty, =ter}. So, one gets:

$$h_{gen}^{wrds1.} = \log_2 2 = 1$$

The conditional entropy, based on the textual observations, is

$$h_{obs}^{wrds1.} = -\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} = 0.970951$$

Case 1.a. Let $\delta = \{$=ter, =ty, =t$\}$. Then the link types GA and GG need to be consolidated: G={GA, GG} and likewise T={TA, TT}. The dictionary becomes

```
gif.=: G+;
tes.=: T+;
=t =ty =ter: G- or T-;
```

The observed pair probabilities become:

p(G) = p(gif,t) + p(gif,ty) + p(gif,ter) = 3/7

p(T) = p(tes,t) + p(tes,ty) + p(tes,ter) = 4/7

So the observed entropy is now

$$h_{PAIR}^{red1.a.} = -\frac{3}{7}\log_2\frac{3}{7} - \frac{4}{7}\log_2\frac{4}{7} = 0.985228$$

The generated entropy is $\log_2 2 = 1$ since there are only two link types in the grammar. The word-counting entropy for the set δ contributes an additional

$$h_{gen}^{wrds1.a.} = \log_2 3 = 1.584963$$

while the observed entropy is

$$h_{obs}^{wrds1.a.} = -\frac{4}{7}\log_2\frac{2}{7} - \frac{3}{7}\log_2\frac{3}{7} = 1.556657$$

Case 1.b. Let $\gamma = \{$=ter, =ty$\}$ as before, and $\varepsilon = \{$gif.=, tes.=$\}$. The link types consolidate: EA={GA, TA} and EM={GG, TT}. The dictionary becomes

```
gif.=  tes.=:  EA+  or  EM+;
=t:  EA−;
=ty  =ter:  EM−;
```

The observed pair probabilities become:

p(EA) = p(gif,t) + p(tes,t) = 2/7

p(EM) = 5/7

So that the entropy is

$$h_{PAIR}^{red1.b.} = -\frac{2}{7}\log_2\frac{2}{7} - \frac{5}{7}\log_2\frac{5}{7} = 0.863121$$

The generated entropy is $\log_2 2 = 1$ since there are only two link types in the grammar. The word-set counting probability adds

$$h_{gen}^{wrds.1.b.} = 2\log_2 2 = 2$$

while the observed probabilities are

$$h_{obs}^{wrds.1.b.} = -\frac{3}{7}\log_2\frac{3}{7} - \frac{4}{7}\log_2\frac{4}{7} - \frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5} = 1.956179$$

Other cases. Case 2.a. and case 2.b. are identical to cases 1.a. and 1.b. because =t and =ty are interchangeable, from the probability point of view.

Case 3.a. and case 3.b. are similar, but with different probabilities.

Case 4. and the subcases are different, but not illuminating.

Final Case. The final consolidation gives $\gamma = \{$=ter, =ty,=t$\}$, and $\varepsilon = \{$gif.=, tes.=$\}$. The dictionary becomes

```
gif.=  tes.=:  LL+;
=t  =ty  =ter:  LL−;
```

The observed pair probabilities become:

p(LL) = 7/7

So that the entropy is

$$h_{PAIR}^{final} = -\frac{7}{7}\log_2\frac{7}{7} = 0$$

The generated entropy is $\log_2 1 = 0$ since there is only one link type in the grammar. The word-set counting probability adds

$$h_{gen}^{wrds.fin} = \log_2 2 + \log_2 3 = 2.584963$$

while the observed word count is

$$h_{obs}^{wrds.fin} = -\frac{4}{7}\log_2\frac{2}{7} - \frac{3}{7}\log_2\frac{3}{7} - \frac{3}{7}\log_2\frac{3}{7} - \frac{4}{7}\log_2\frac{4}{7} = 2.541885$$

Summary. The table below summaries these results. The sum columns show the entropy according to the equation 5 for the observed frequencies, and the generated frequencies.

	h_{obs}^{red}	h_{gen}^{red}	h_{obs}^{wrds}	h_{gen}^{wrds}	$h_{obs}^{red}+h_{obs}^{wrds}$	$h_{gen}^{red}+h_{gen}^{wrds}$
Initial	2.521641	2.584963	0	0	2.521641	2.584963
Case 1.	1.842371	2	0.970951	1	2.813322	3
Case 1.a.	0.985228	1	1.556657	1.584963	2.541885	2.584963
Case 1.b.	0.863121	1	1.956179	2	2.819299	3
Final	0	0	2.541885	2.584963	2.541885	2.584963
Case 3.	1.950212		1.0		2.950212	
Case 3.a.	0.985228		1.556656		2.541884	
Case 3.b.	0.985228		1.985228		2.970456	
Case 4.	1.556657		0.985228		2.541885	

Arghhh. Such a simple case, so much complexity... anyway, the case 3 and 4 are computed from the script “link-type/gifty.scm” in this same directory.

Conclusions: based purely on entropy maximization, all cases advance, but none go to the final case. But we are not imposing any ’complexity penalty’ on this.

Results on some alternate distributions, for this ranking: “tester testy test gifter gifty gift”

- Pure Zipf: $(rank)^{-1.0}$: none advance ($h_{initial} = 2.281979$ and $h_{final} = 2.293598$)
- Zipf $(rank)^{-1.05}$:none advance ($h_{initial} = 2.251204$ and $h_{final} = 2.263603$)
- Zipf $(rank)^{-1.5}$:none advance ($h_{initial} = 1.930661$ and $h_{final} = 1.948128$)

None of these advance because the initial and final entropies are so very close. But, as before, there are advnces, with the biggest ones to case 4.c and 3.b. The alternative rankings “tester testy test gift gifty gifter” and “tester gifter testy gifty test gift” give only slightly different results.

Link-type discovery, better example

In the previous, the unified link-type discovery is inevitable, so a more complex version is needed, with a less-obvious outcome. So lets take the original example of link-type discovery, and add some confounding link types. We begin with the intial observations, plus some extras, given in the table below:

Algorithm 2 Example morpheme grammar

```
gif.=: GA+ or GB+ or GC+;
tes.=: TA+ or TB+ or TC+;
blo.=: BB+ or BF+;
=t: GA− or TA−;
=ty: GB− or TB− or BB−;
=ter: GC− or TC−;
=fu: BF−;
```

A more complex grammar showing morpheme linkages.

Table 2: Example Link Frequency Table

Pair	Initial Link Type	# observations
gif–t	GA	1
gif–ty	GB	1
gif–ter	GC	1
tes–t	TA	1
tes–ty	TB	1
tes–ter	TC	2
blo–ty	BB	3
blo-fu	BF	1

Example distribution of link frequencies obtained from an example corpus.

The addition of "bloty" to the link table, and with a strong weight, will tend to derail the consolidation of the =ty suffix with the others. The addition of "blofu" helps make sure that there's some confusion about the "blo=" stem.

The corresponding link-grammar dictionary is:

From the above initial dictionary, we hope to deduce one word class that contains gif.= and tes.= and another that contains =t and =ter; eactly how the rest plays out is unclear. Lets begin by starting with the un-clustered entropy, and then see what hapens if we try various different clusters. So, as before, let $\beta = \{GA, GB, GC, TA, TB, TC, BB, BF\}$ and write:

$$\begin{aligned} h_{PAIR}^{observed} &= -\sum_{t\in\beta} p(t)\log_2 p(t) \\ &= -\frac{6}{11}\log_2\frac{1}{11} - \frac{2}{11}\log_2\frac{2}{11} - \frac{3}{11}\log_2\frac{3}{11} \\ &= 2.845351 \end{aligned}$$

with $p(t)$ being the probability of observing link-type t. Since there are 8 different link types, the generated entropy is $h_{PAIR}^{generated} = \log_2 8 = 3$. The different between these two is $h^{gen} - h^{obs} = 0.154649$. The observed corpus also has 8 words in it (not counting multiplicity): this is by design; before reduction, there is always exactly one link type for each morpheme pair.

Lets look at several cases:

1. Group gif.= and tes.= together.
2. Group gif.= and blo.= together.
3. Group =t and =ty together.
4. Group =ty and =fu together.
5. Group =ter and =fu together.

Here, we expect case 1 to go easily, cases 2 and 3 to to be ambiguous or blocked, case 4 to be weakly blocked, and case 5 to be strongly blocked. So, proceeding:

Case 1. Group gif.= and tes.= together. Let $\gamma = \{$gif.=, tes.=$\}$. Then the link types G* and T* need to be consolidated: A=$\{$GA,TA$\}$ and likewise B=$\{$GB,TB$\}$ and C=$\{$GC,TC$\}$. The dictionary becomes

```
gif.=  tes.=: A+ or B+ or C+;
blo.=: BB+ or BF+;
=t: A-;
=ty: B- or BB-;
=ter: C-;
=fu: BF-;
```

The observed pair probabilities become:

p(A) = p(gif,t) + p(tes,t) = 2/11 = p(B) = p(gif,ty) + p(tes,ty)

p(C) = p(gif,ter) + p(tes,ter) = 3/11

p(BB) = p(blo,ty) = 3/11

p(BF) = p(blo,fu) = 1/11

So the observed entropy is now

$$h_{PAIR}^{red1.} = -\frac{4}{11}\log_2\frac{2}{11} - \frac{6}{11}\log_2\frac{3}{11} - \frac{1}{11}\log_2\frac{1}{11} = 2.231270$$

The generated entropy is $h^{gen} = \log_2 5 = 2.321928$. The difference is $h^{gen} - h^{obs} = 0.090658$. This clearly brings the entropy closer to the theoretical (equidistributional) maximum; the grouping goes. However, $h^{lang} = \log_2 8 = 3$ as before, since the generated language still has 8 words in it.

Case 2. Group gif.= and blo.= together. Let $\delta = \{$gif.=, blo.=$\}$. Then the link types GB and BB can be consolidated, because they share the common suffix =ty: B={GB,BB}. No other link consolidation is possible, without permitting impermissible (previously unseen) linkages. The dictionary becomes

```
gif.= blo.=: GA+ or B+ or GC+ or BF+;
tes.=: TA+ or TB+ or TC+;
=t: GA- or TA-;
=ty: B- or TB-;
=ter: GC- or TC-;
=fu: BF-;
```

Note that this dictionary does allow several previously unobserved words: giffu, blot, bloter. This is what happens when one hypothesizes unions between classes that merely overlap, instead of being subsets. What happens next depends on whether the overlap was large, or small.

The observed pair probabilities become:

p(GA) = p(gif,t) = 1/11 = p(GC) = p(TA) = p(TB)

p(TC) = p(tes,ter) = 2/11

p(B) = p(gif,ty) + p(blo,ty) = 4/11

p(BF) = p(blo,fu) = 1/11

So the observed entropy is now

$$h_{PAIR}^{red2.} = -\frac{5}{11}\log_2\frac{1}{11} - \frac{2}{11}\log_2\frac{2}{11} - \frac{4}{11}\log_2\frac{4}{11} = 2.550341$$

The generated entropy is $h^{gen} = \log_2 7 = 2.807355$. The difference is $h^{gen} - h^{obs} = 0.257014$. The entropy is not getting closer to the equidistributional maximum; this grammar is rejected.

Case 3. Group =t and =ty together. Let $\varepsilon = \{$=t, =ty$\}$ Then we may group G={GA, GB} and T={TA, TB}. The corresponding link-grammar dictionary is:

```
gif.=: G+ or GC+;
tes.=: T+ or TC+;
blo.=: BB+ or BF+;
=t =ty: G- or T- or BB-;
=ter: GC- or TC-;
=fu: BF-;
```

The above again allows a new, unobserved word: “blot”. The observed pair probabilities become:

p(G) = p(gif,t) + p(gif,ty) = 2/11 = p(T) = p(tes,t) + p(tes,ty)

p(GC) = p(gif,ter) = 1/11

p(TC) = p(tes,ter) = 2/11

p(BB) = p(blo,ty) = 3/11

p(BF) = p(blo,fu) = 1/11

So the observed entropy is now

$$h_{PAIR}^{red3.} = -\frac{6}{11}\log_2\frac{2}{11} - \frac{2}{11}\log_2\frac{1}{11} - \frac{3}{11}\log_2\frac{3}{11} = 2.481715$$

The equidistributional entropy is $h^{gen} = \log_2 6 = 2.584963$. The difference is $h^{gen} - h^{obs} = 0.103248$. This difference means we are getting closer to the maximum; the grouping is acceptable! Its really not much worse than case 1, which was unambiguous.

Case 4. Group =ty and =fu together. Let $\zeta = \{$=ty, =fu$\}$. Then we must group B={BB,BF} together. The dictionary is:

```
gif.=: GA+ or GB+ or GC+;
tes.=: TA+ or TB+ or TC+;
blo.=: B+;
=t: GA- or TA-;
=ty =fu: GB- or TB- or B-;
=ter: GC- or TC-;
```

No new unobserved words are allowed by this grouping! The observed pair probabilities are:

p(GA) = p(gif,t) = 1/11 = p(GB) = p(GC) = p(TA) = p(TB)

p(TC) = p(tes,ter) = 2/11

p(B) = p(blo,ty) + p(blo,fu) = 4/11

The observed entropy is then:

$$h_{PAIR}^{red4.} = -\frac{5}{11}\log_2\frac{1}{11} - \frac{2}{11}\log_2\frac{2}{11} - \frac{4}{11}\log_2\frac{4}{11} = 2.550341$$

Curiously, this entropy is identical to the completely different case 2. The equidistributional entropy is $h^{gen} = \log_2 7 = 2.807355$ and the difference is thus $h^{gen} - h^{obs} =$

0.257014 which is sharply further away from the equidistributional maximum. Thus, this grouping is rejected. This is perhaps surprising ... First, this grammar did not generate any new unobserved words; thus, it is a faithful grammar. Also, it suceeds in reducing the total number of link-types, and thus is naively acceptable for that reason. However, the frequency distribution of th generated grammar move away from the observed frequency distribution, leading to the rejection. This begs a question: when and how might we annotate the grammar with frequency information?

Case 5. Group =ter and =fu together, so that $\eta = \{$=ter, =fu$\}$. It does not appear that any link types get consolidated! This is not much of a grouping, then ...

```
gif.=: GA+ or GB+ or GC+;
tes.=: TA+ or TB+ or TC+;
blo.=: BB+ or BF+;
=t: GA- or TA-;
=ty: GB- or TB- or BB-;
=ter =fu: GC- or TC- or BF-;
```

Many new, unobserved words are allowed: bloter, giffu, tesfu. The observed pair probabilities are:

p(GA) = p(gif,t) = 1/11 = p(GB) = p(GC) = p(TA) = p(TB)

p(TC) = p(tes,ter) = 2/11

p(BB) = p(blo,ty) = 3/11

p(BF) = p(blu,fu) = 1/11

The observed entropy is then:

$$h^{red5.}_{PAIR} = -\frac{6}{11}\log_2\frac{1}{11} - \frac{2}{11}\log_2\frac{2}{11} - \frac{3}{11}\log_2\frac{3}{11} = 2.845351$$

This is identical to the unreduced entropy: no surprise, because no link consolidation was performed. The equidistributional entropy is the same as well: $h^{gen} = \log_2 8 = 3$ since there are still 8 link types. The language entropy increased: there are now 11 possible words in the language, so $h^{lang} = \log_2 11 = 3.459432$. This is a very unsatisfying situation: the difference in entropies is no better or worse than the starting point, and so this seems like a reasonable sideways slide, and yet, this grammar allows a bunch of nonsense words to be generated. That seems wrong.

Summary Of the 5 cases, three are blocked (cases 2,4,5), and two are acceptable (1,3). Case 1 looks to be the best. Lets see what might happen next:

- Case 1a. Group =t and =ty

- Case 1b. Group =t and =ter
- Case 1c. Group =ty and =ter

Case 1a resembles Case 3 so we expect it to advance. Likewise for case 1c. Its reasonable to guess that case 1b will be the strongest. Lets try some of these.

Case 1a. Group =t and =ty. The link merges are T={A,B}; the resulting gramar is:

```
gif.= tes.=: T+ or C+;
blo.=: BB+ or BF+;
=t =ty: T- or BB-;
=ter: C-;
=fu: BF-;
```

This grammar allows a new unobserved word: “blot”. The observed pair probabilities become:

p(T) = p(gif,t) + p(tes,t) + p(gif,ty) + p(tes,ty) = 4/11

p(C) = p(gif,ter) + p(tes,ter) = 3/11

p(BB) = p(blo,ty) = 3/11

p(BF) = p(blo,fu) = 1/11

So the observed entropy is now

$$h_{PAIR}^{red1.} = -\frac{4}{11}\log_2\frac{4}{11} - \frac{6}{11}\log_2\frac{3}{11} - \frac{1}{11}\log_2\frac{1}{11} = 1.867634$$

The generated entropy is $h^{gen} = \log_2 4 = 2$. The difference is $h^{gen} - h^{obs} = 0.132366$ which is not closer than the previous delta of 0.090658, so this is rejected.

Casse 1b. Group =t and =ter. This consolidates links T={A,C} and so the dictionary becomes

```
gif.= tes.=: T+ or B+;
blo.=: BB+ or BF+;
=t =ter: T-;
=ty: B- or BB-;
=fu: BF-;
```

This does not generate any new words. The observed pair probabilities become:

p(A) = p(gif,t) + p(tes,t) + p(gif,ter) + p(tes,ter) = 5/11

p(B) = p(gif,ty) + p(tes,ty) = 2/11

p(BB) = p(blo,ty) = 3/11

p(BF) = p(blo,fu) = 1/11

So the observed entropy is now

$$h^{red1.}_{PAIR} = -\frac{5}{11}\log_2\frac{5}{11} - \frac{2}{11}\log_2\frac{2}{11} - \frac{3}{11}\log_2\frac{3}{11} - \frac{1}{11}\log_2\frac{1}{11} = 1.789929$$

The generated entropy is $h^{gen} = \log_2 4 = 2$. The difference is $h^{gen} - h^{obs} = 0.210071$ which does not get closer; the best still stands at 0.090658. This is surprising: it sems to be blocking the discovery of the clique.

Case 1c. Group =ty and =ter. This consolidates T={B,C}, so the dictionary becomes

```
gif.= tes.=: A+ or T+;
blo.=: BB+ or BF+;
=t: A-;
=ty =ter: T- or BB-;
=fu: BF-;
```

This does not generate any new words. The observed pair probabilities become:

p(A) = p(gif,t) + p(tes,t) = 2/11

p(C) = p(gif,ty) + p(tes,ty)+ p(gif,ter) + p(tes,ter) = 5/11

p(BB) = p(blo,ty) = 3/11

p(BF) = p(blo,fu) = 1/11

The changes are the same as for case 1b. Again, this is blocked.

Case 1f. This is the "final" case: group together =t =ty =ter into one. This consolidates T={A,B,C} so that

```
gif.= tes.=: T+;
blo.=: BB+ or BF+;
=t =ty =ter: T- or BB-;
=fu: BF-;
```

This allows new words "blot", "bloter". The observed pair probabilities become:

p(T) = p(gif,t) + p(tes,t) + p(gif,ty) + p(tes,ty)+ p(gif,ter) + p(tes,ter) = 7/11

p(BB) = p(blo,ty) = 3/11

p(BF) = p(blo,fu) = 1/11

The observed entropy is

$$h_{PAIR}^{red1.} = -\frac{7}{11}\log_2\frac{7}{11} - \frac{3}{11}\log_2\frac{3}{11} - \frac{1}{11}\log_2\frac{1}{11} = 1.240671$$

Hmm. 3 link types

Summary Movement to Cases 1a, 1b and 1c are all blocked. This seems surprising. The relatively high-frequency observation of =ter makes the distribution of the consolidated grammar to deviate strongly from the distribution of the observed corpus. This seems like an undesired effect, as the point of learning how to simplify the grammar is to obtain a smaller grammar, rather than to preserve the the distribution of the corpus. Mostly.

Intuition suggests that the grammar for "common" cases should be consolidated. The grammar for quite rare cases should indeed be handled distinctly. To avoid this seemingly perverse outcome, perhaps the grammar should contain frequency information, which is to be consolidated appropriately. This is truly tedious, but seems to be neccessary. So we have to start from scratch.

And we do, below, and its a total failure, as now, the corpus frequencies are recorded faithfully, so the consolidation process doesn't tell us anything we didn't know. Its the same calculation done differently.

Worked Link Consolidation Example, with Frequencies (XXX Fail)

(XXX The below fails, don't bother reading it). So we start all over again, using the same corpus frequences as before, namely, those of table 2. The grammar is essentially identical to that of 2, except that it is now annotated with probabilities.

```
gif.=: (GA+)(1/11) or (GB+)(1/11) or (GC+)(1/11);
tes.=: (TA+)(1/11) or (TB+)(1/11) or (TC+)(2/11);
blo.=: (BB+)(3/11) or (BF+)(1/11);
=t: GA- or TA-;
=ty: GB- or TB- or BB-;
=ter: GC- or TC-;
=fu: BF-;
```

The above only annotates the +-going links; it seems like annotating the –going links would cause double-counting. This is somewhat confusing, since the probability has nothing to do with directionality. A better notation is not obvious. Lets go through the cases as before.

Case 1. Group gif.= and tes.= together. Let $\gamma = \{$gif.=, tes.=$\}$. Then the link types G* and T* need to be consolidated: A=$\{$GA,TA$\}$ and likewise B=$\{$GB,TB$\}$ and C=$\{$GC,TC$\}$. The dictionary becomes

```
gif.= tes.=: (A+)(2/11) or (B+)(2/11) or (C+)(3/11);
blo.=: (BB+)(3/11) or (BF+)(1/11);
=t: A-;
=ty: B- or BB-;
=ter: C-;
=fu: BF-;
```

The observational probabilities are unchanged, as the dictionary probabilities have no bearing on the parsing. However, the entropy of the generated language is different, as it is no longer $\log_2 5$ but instead

$$h^{gen} = -\frac{6}{11}\log_2\frac{1}{11} - \frac{2}{11}\log_2\frac{2}{11} - \frac{3}{11}\log_2\frac{3}{11}$$

That is, it is now identical to $h^{observed}$. No surprise, as we made it like that, by encoding the frequency information in the dictionary.

Case 2. Group gif.= and blo.= together. Let $\delta = \{$gif.=, blo.=$\}$. Then the link types GB and BB can be consolidated, because they share the common suffix =ty: B=$\{$GB,BB$\}$. No other link consolidation is possible, without permitting impermissible (previously unseen) linkages. The dictionary becomes

```
gif.= blo.=: (GA+)(1/11) or (B+)(4/11) or (GC+)(1/11) or (BF+)(1/11);
tes.=: (TA+)(1/11) or (TB+)(1/11) or (TC+)(2/11);
=t: GA- or TA-;
=ty: B- or TB-;
=ter: GC- or TC-;
=fu: BF-;
```

The generated entropy is

$$h^{gen} = -\frac{5}{11}\log_2\frac{1}{11} - \frac{4}{11}\log_2\frac{4}{11} - \frac{2}{11}\log_2\frac{2}{11}$$

which is identical to the corpus entropy, again. Not surprising, I guess ... we seem to be doing the same calculation, but in a different way. Dohh. Never mind ...

Alternate Distributions

Instead of looking for an equi-distribution, how about a Zipf distribution, which seems far more plausible? The distribution is

$$p(k,n) = \frac{1}{kH_n}$$

where the normalization is $H_n = \sum_{k=1}^{n} 1/n$. The entropy is then

$$\begin{aligned} h_n^{Zipf} &= -\sum_{k=1}^{n} p(k,n) \log_2 p(k,n) \\ &= \frac{1}{H_n} \sum_{k=1}^{n} \frac{\log_2 kH_n}{k} \\ &= \log_2 H_n + \frac{1}{H_n} \sum_{k=1}^{n} \frac{\log_2 k}{k} \end{aligned}$$

and the first few values are shown below. For comparison, $h_n^{equi} = \log_2 n$ is also shown.

n	H_n	h_n^{Zipf}	h_n^{equi}
2	1.5	0.918296	1
3	1.83333	1.435371	1.584963
4	2.033333	1.792488	2
5	2.283333	2.063860	2.321928
6	2.45	2.281979	2.584963
7	2.592857	2.463914	2.807355
8	2.717857	2.619715	3

The question is then: how would the above cases go if this was used as the deciding factor? This is shown below:

Case	# lnk	$h^{observed}$	h^{equi}	$h^{eq} - h^{obs}$	OK	h^{Zipf}	$h^{Zipf} - h^{obs}$	C1	C2
Base	8	2.845351	3	0.154649		2.619715	-0.225636		
1.	5	2.231270	2.321928	0.090658	Y	2.063860	-0.16741	Y	N
2.	7	2.550341	2.807355	0.257014	N	2.463914	-0.086427	Y	N
3.	6	2.481715	2.584963	0.103248	Y	2.281979	-0.199736	Y	N
4.	7	2.550341	2.807355	0.257014	N	2.463914	-0.086427	Y	N
5.	8	2.845351	3	0.154649		2.619715	-0.225636		
1a.	4	1.867634	2	0.132366	N	1.792488	-0.075146	Y	N
1b.	4	1.789929	2	0.210071	N	1.792488	0.002559	Y	N
1c.	4	1.789929	2	0.210071	N	1.792488	0.002559	Y	N
1f.	3	1.240671	1.584963	0.344292	N	1.435371	0.1947	N	N

There seem to be two different decision criteria to apply:

1. Does the reduced entropy come closer to the Zipfian entropy?
2. Does the reduced entropy increase, relative to the Zipfian entropy?

The first is shown in column C1, the second in C2. Naively, C2 seems like a better chooser. Does it also work for the simple case (with the original 7-word corpus)? Lets see:

Case	#lnk	$h^{observed}$	h^{equi}	$h^{eq} - h^{obs}$	OK	h^{Zipf}	$h^{Zipf} - h^{obs}$	C1	C2
Base	6	2.521641	2.584963	0.063322		2.281979	-0.239662		
1.	4	1.842371	2	0.157629	N	1.792488	-0.049883	Y	N
1a	2	0.985228	1	0.014772		0.918296	-0.066932	N	Y
1b.	2	0.863121	1	0.136879		0.918296	0.055175	N	N

Basically, this is really irritating.

Thoughts

What have we learned from the above?

- The problem of condensing together morphemes into classes which share common link types is the bipartite clique problem. It is a known-hard problem.
- Bad grammars increase the size of the language. This could be acceptable, if the increase is small. What's the criteria? Unclear.

Consciousness - 27 July 2014

Two works:

- Masafumi Oizumi, Larissa Albantakis, Giulio Tononi, "From the Phenomenology to the Mechanisms of Consciousness: Integrated Information Theory 3.0" (2014) PLOS Computational Biology, http://www.ploscompbiol.org/article/info%3Adoi%2F10.1371%2Fjournal.p
- Max Tegmark, Consciousness as a State of Matter (27 Feb 2014) arXiv:1401.1219v2 [quant-ph]

Curious points and thoughts:

- CEI – "Cause-effect information" – Tononi – sound like a time-ordered variant of mutual information. How should this be defined? Answer: my guess is that its just like the mutual information defined in eqn**??**, right? Because the relational complexity can deal with arbitrary structures, so that seems appropriate.

13 Sept 2014

The Zipfian distribution is typical of a scale-free network. So why is language scale-free? Crudely, because we attempt to recycle existing concepts/words.

Next, from this:

- Christoph Adami "Information-Theoretic Considerations Concerning the Origin of Life" http://arxiv.org/pdf/1409.0590v1.pdf

Come the following thoughts:

- Never assume a uniform distribution of parts. This makes it very unlikely that an imprtant assemblage of parts can arise at random. For Adami, this is used to argue that biotic and abiotic strings should have very siilar distributions (or rather, the converse: a non-uniform abotic distribution makes it much more likely to find a replictor with a similar distribution.)

- The information content of (grammatical sentences of length L is

$$I_{grammatical} = -\log_2(N_{grammatical}/N_{total})$$

where N_{total} is the total number of sentences of length L, assuming a *uniform* distribution of words picked from a vocabulary of D words. That is, $N_{total} = D^L$. But this is weird ... because the vocabulary isn't really a constant, and the natural distribution is not uniform, so its not clear what kind of "information" the above actualy is ...

Thermodynamics - 24 March 2015

Some quick short notes: blog post: "Thermodynamics with Continuous Information Flow" https://johncarlosbaez.wordpress.com/2015/03/21/19395 with arxiv paper: http://arxiv.org/pdf/1402.3276v3.pdf Jordan M. Horowitz and Massimiliano Esposito study the master equation for a probability p(x,y) over two distributions X,Y, which are connected via a bipartite graph. The total system is also connected to a thermal bath. The eqn is

$$\frac{dp(x,y)}{dt} = \sum_{x',y'} H^{y,y'}_{x,x'} p(x',y') - H^{y',y}_{x',x} p(x,y)$$

i.e. its Markovian; we've written two indexes, which makes it clearer when H is bipartite i.e.

$$H^{y,y'}_{x,x'} = \begin{cases} H^{y}_{x,x'} & \text{if } y = y' \\ H^{y,y'}_{x} & \text{if } x = x' \\ 0 & \text{otherwise} \end{cases}$$

The interesting part is the entropy, and the thermal bath, which is not in the master eqn(!) The total entropy is $S_{tot} = S_{XY} + S_{env}$. Per usual, the information entropy is $S_{XY} = -\sum_{x,y} p(x,y) \log p(x,y)$. Two tricks now happen: (1) taking the timer derivative of S_{XY} results in something that naturally splits into an X piece and a Y piece. Trick (2) is that S_{env} cannot be written down directly, but its time derivative can be; it is proportional to the heat current: $\dot{S}_{env} = -\dot{Q}/T$ Observer the tiny dot over S,Q these are the usuaul rate-of-change dots, (i.e. just rates, not functions we are taking time derivative of). Q is heat, Q-dot is heat flow, T is temp. Local detailed balance requires that

$$\log \frac{H^{y,y'}_{x,x'}}{H^{y',y}_{x',x}} = \frac{-(E_{x,y} - E_{x',y'})}{kT}$$

is the change in energy due to a state transition: the change in energy is supplied by the heat reservior. Where does this mystery equation come from? Answer:

Detailed balance requires that, when the system reaches equilibirum, that the transition rate into and out of the equlibrium state $p_i = \pi_i$ are equal:

$$H_{ji}\pi_i = H_{ij}\pi_j$$

(there is NO repeated-index summation). Then, just write $\pi_i = \exp -E_i/kT$, and turn the crank. The general principle: *the log of the ratio of the forward and backward transition rates between two states must be proportional to the energy difference between those states!*

BTW the detailed-balance equation resembles Bayes Theorem, in that, if we wrote $H_{ji} \to P(j|i)$ and $\pi_i \to P(i)$, then detailed balance is written as $P(j|i)P(i) = P(i|j)P(j)$. So the master equation describes "non-equilibrium Bayes statistics", in a strange sense. Hmm. But, of course, this is just a Markov chanin/process.

26 March 2015

The Inverse Relationship Principle of Channel theory: "*Whenever there is an increase in available information there is a corresponding decrease in possibilities, and vice versa.*" Barwise, "Information and Impossibilities." Notre Dame J. Formal Logic Volume 38, Number 4, 488-515. Barwise, Jon and Jerry Seligman 1997. "Information Flow: The Logic of Distributed Systems". Cambridge: Cambridge University Press

Linear networks - 3 May 2015

Another Baez post: "*A Compositional Framework for Passive Linear Networks"* blog: https://johncarlosbaez.wordpress.com/2015/04/28/a-compositional-framework-for-passive-linear-networks/

So first, we have the table:

	mechanics	electronics	information geometry	geometric mechanics
q	position	charge	entropy	point on manifold
$\dot{q}$	velocity	current	entropy change	tangent vector
p	momentum	flux linkage	temperature momentum	covector (vector in cotangent bundle)
$\dot{p}$	force	voltage	temperature	map from tangent bundle to cotangent bundle
	principle of least action	principle of least power dissipation	?	principle of least action

This table is slightly oversimplified; the first four columns show only the linear case. The fifth column makes clear that force isn't really p-dot; that only holds when the manifold is flat. Anyway..

Key concepts: (*) monoidal categories are needed, and (*) symplectic geometry is needed.

Baez does the linear passive-component electronics example, viz a network of passive resistors, capacitors, inductors. For the resistor network, voltages at each node are taken from the field $\mathbb{F} = \mathbb{R}$ while for the inductive network, the field is the field of rational functions of one variable $\mathbb{F}=\mathbb{R}(t)$ with t time: i.e. voltage varying over time. A Dirichlet form is a quadratic form

$$P(\phi)=\frac{1}{2}\sum_{i,j}\frac{(\phi_i-\phi_j)^2}{r_{ij}}$$

where r_{ij} is the reistance (impedance) between nodes i and j, and $\phi_i=\phi(i)$ is the voltage at node i. (Actually, we should be summing over edges, so as to handle parallel resistors). Note that the space of Dirichlet forms is smaller than the space of quadratic forms: Dirichlet forms do not have diagonal entries. Note that P is (half) the power dissipation.

The principle of least power dissipation is this: Given fixed voltages ψ on the boundary of the network, i.e. on the input/output terminals, the actual power dissipated is

$$Q(\psi)=\min_{\phi\in\mathbb{R}^N,\phi|_{\partial N}=\psi}P(\phi)$$

Notation: there are N nodes, so voltages live in $\mathbb{R}^N$. The boundary of the network (input/output terminals) is written as ∂N and the voltages are held fixed at the boundary. Note that Q is also a Dirichlet form. Viz its a map $Q:\mathbb{R}^{\partial N}\to\mathbb{R}$. The black-box principle of equvalent resistor networks is that any two resistor networks are black-box equivalent when they have the same Q.

For the correct generalization to impedance, it is not enough to just replace $\mathbb{F}=\mathbb{R}$ by $\mathbb{F}=\mathbb{R}(t)$ because this fails to deal with the time variation correctly. Put it another way: for the pure resistor network, we are free to fix voltages at both the input and output terminals arbitrarily; the internal currents are determined entirely by these. For the general case with impedance, we are not free to fix both voltages and currents at both the input and output terminals. Out of the total set of $2\dim(\partial N)$ voltages and currents, we can fix only half the set, i.e. a mixture of voltages, currents of $\dim(\partial N)$.

To handle this, we need to construct a symplectic vector space, with a symplectic form on it, and work in the Lagrangian subspace of it. Thus, we have $\psi\in\mathbb{F}^{\partial N}$ as the potentials at the network terminals, and $dQ_\psi\in\left(\mathbb{F}^{\partial N}\right)^*$ as the conjugate currents. Out of the total space $\mathbb{F}^{\partial N}\oplus\left(\mathbb{F}^{\partial N}\right)^*$ of states, the subspace of actually attainable states is

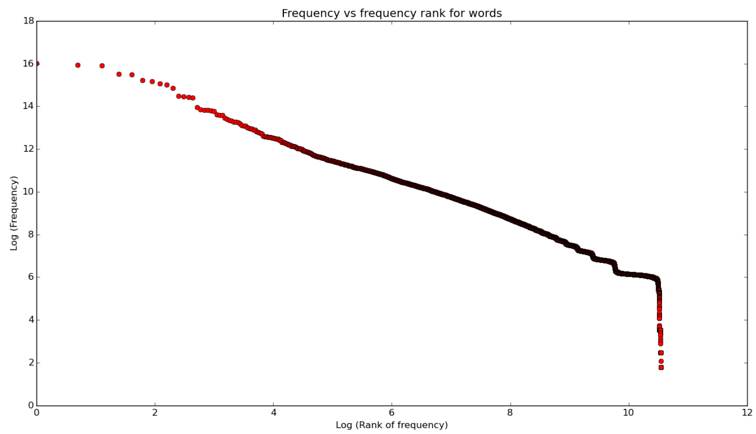
$$\text{Graph}\,(dQ)=\left\{\left(\psi,dQ_\psi\right)|\,\psi\in\mathbb{F}^{\partial N}\right\}\subseteq\mathbb{F}^{\partial N}\oplus\left(\mathbb{F}^{\partial N}\right)^*$$

The set of Lagrangian subspaces is an algebraic variety, the Lagrangian Grassmanian.

Baez primary result on impedence networks is that the black box is describable by the symplecitification of .. OK I don't get it.

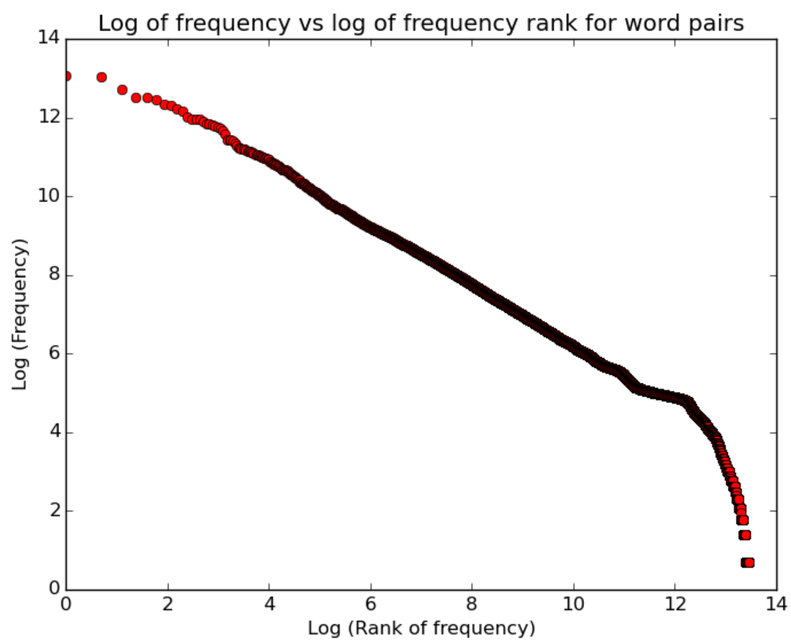
MI graphs – 31 May 2015

Results from Rohit:

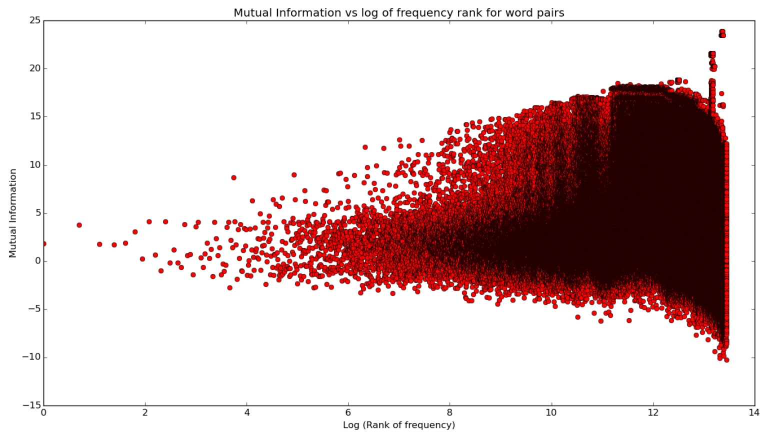


Above is for single words. Frequency is the number of times the word was observed.

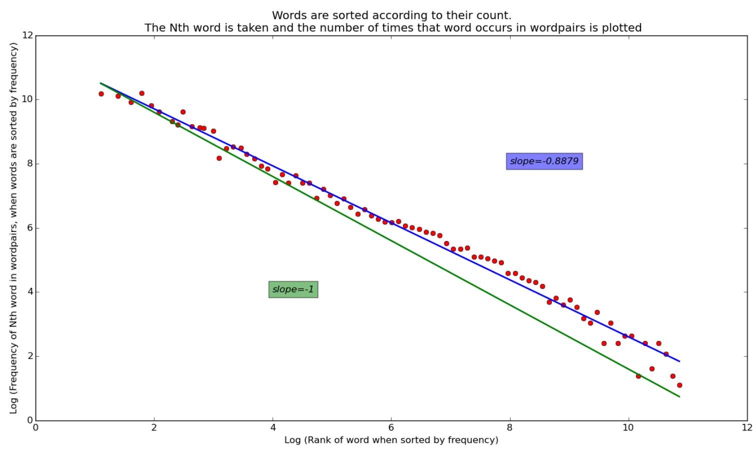
The log is the natural log. Below is for word-pairs.



Below is a scatterplot for mutual information of word-paris vs rank.



number of word pairs as function of word rank



Summary: these are more or less exactly as expected. Will need to make cuts to get rid of the low-frequency word pairs...

Mining Grammatical Categories – 20 June 2015

Now that we have a database filled with disjunct statistics, how do we datamine that for grammatical categories, which is, after all, the main point of this exercise? Let me explain in several steps; at first illustrative, and then, more precisely. So first, consider a corpus containing these sentences:

```
the big tree
a green tree
the big bush
```

```
a green bush
```

I want to conclude that "tree" is a lot like "bush", and the two should be considered as being "similar enough to be merged into a common grammatical category". That is, the words "tree" and "bush" always occur in similar contexts, or even the same contexts. The word “context” here means “the dependency parse context”, and not “the n-gram context”. More precisely, it means “the accumulated statistics for the disjuncts obtained from MST dependency parses”.

Suppose the following parses were observed:

```
+---MA---+
|   +-MB-+
|   |    |
the big  tree

+---MC---+
|   +-MD-+
|   |    |
a  green tree

+---ME---+
|   +-MF-+
|   |    |
the big bush

+---MG---+
|   +-MH-+
|   |    |
a  green bush
```

Recall that the above parses were obtained by performing a Maximum-Spanning-Tree (MST) parse based on word-pair mutual information (MI). The MST is obtained by considering the graph clique joining all words in the sentence, and then keeping only those edges that have the greatest MI between pairs of words. This is the “Yuret parse”. The Yuret parse does not have labelled edges, and so we assign arbitrary (but unique!) link labels to the edges that were kept. Every unique word pair gets a unique link type. Then, using the standard Link Grammar theory, each link is broken into a + and a - connector, and the ordered set of connectors on a word are called a disjunct.

The disjuncts extracted from the above parses would then be:

```
tree: (MA- & MB-) or (MC- & MD-)
bush: (ME- & MF-) or (MG- & MH-)
```

No two disjuncts are alike, so naively, these seem completely uncomparable. Of course, this is wrong; we need to compare the “decoded disjunct”. The “decoded disjunct” is NOT a part of the standard Link Grammar theory, so let me explain it here: it is simply the disjunct where the connector is replaced by the word or word-class that it connects to. For example, `MA-` connects to the word "the", so the "decoded connector" for `MA-` is `$the$-`. So, the decoded disjuncts are then:

```
tree: ($the$- & $big$-) or ($a$- & $green$-)
bush: ($the$- & $big$-) or ($a$- & $green$-)
```

Now we can see that the decoded disjuncts are identical, for this example. Based on this, we conclude that perhaps "tree" and "bush" indeed belong to the same grammatical category. The remainder of the clustering algorithm is now "obvious": rewrite the dictionary so that it has a single entry for both words:

```
tree bush: (MA- & MB-) or (MC- & MD-)
```

This leaves the ME+, MF+, *etc*. connector dangling: thus, we need to search for all occurances of ME+ and replace it by MA+, and likewise all occurances of MF+ need to be replaced by MB+, and so on.

Similarity metrics

The above conveys the general idea, but is over-simplifies a few aspects. First of all, it is very unlikely that two words will appear in sentence contexts that are exactly identical. Secondly, some constructions may be very common, and others, very rare; that is, some disjuncts may be very common, and some very rare. So, for example: suppose we read a text which used the phrase "*the big idea*" a lot, but we also read an obscure linguistics text that said that "*a green idea sleeps furiously*". It would probably be a mistake to lump "idea" in with "tree, bush", given that "green idea" is a very rare construction. Thus, we need a better way of comparing collections of disjuncts.

One obvious way is to treat a collection of decoded disjuncts as a vector in a high-dimensional vector space. The similarity between two vectors could be given by the cosine between two vectors. Alternately, perhaps the vectors could be treated as points, and similarity be given by the distance between points. There are other possibilities; the best choice is not obvious; several need to be explored.

Thus, for example, let $\{e_1, e_2, e_3, \cdots\}$ be the basis of a high-dimensional vector space. For the previous example, we let e_1 correspond to the decoded disjunct `($the$- & $big$-)` while e_2 corresponds to `($a$- & $green$-)`. The word "tree" is then some vector ... what vector should it be? There are several choices. Suppose that `($the$- & $big$-)` was observed with a frequency p_1 and that `($a$- & $green$-)` was observed with frequncy p_2. The corresponding vector is then obviously $p_1 e_1 + p_2 e_2$ and we can construct another vector that corresponds to the the word "bush", say, for example: $q_1 e_1 + q_2 e_2$.

The dot-product between "tree" and "bush" is then given by $p_1 q_1 + p_2 q_2$, so that the larger the product, the closer the two words are. The cosine angle is $(p_1 q_1 + p_2 q_2)/|p||q|$ where $|p| = \sqrt{p_1^2 + p_2^2}$ and so on. The closer that the cosine is to 1.0, the closer the two words are. There are other possibilities: we have the Cartesian distance

$$dist(tree, bush) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2}$$

and we can contemplate *lp*-metrics as well.

None of the above metrics take into account the mutual information (MI) of the disjunct. This is almost surely a mistake. Due to the vagaries of MST parsing, there will be many disjuncts with a low MI value. This is not uncommon in sentences with prepositions, where MST gives some poor choices for the links to the prepositions, and thus results in disjuncts with low MI values. Recall, the higher the MI, the stronger the structure is. Thus, perhaps a better vector for "tree" might be

$$tree = e_1 m_1 p_1 + e_2 m_2 p_2$$

The above seems to be the most entropic-like in its expression. However, the probabilites might weight the terms too strongly, and so a weaker weighting would be the below. It is not yet clear to me which of these expressions are the most "elegant", or which work the best...

$$tree = e_1 (m_1 - \log_2 p_1) + e_2 (m_2 - \log_2 p_2)$$

Here m_1 and m_2 are the mutual information of the disjuncts `(MA- & MB-)` and `(MC- & MD-)`, respectively. The last two seem to be closer to the intended spirit of the maximum entropy principle. There are even more possibilities, though.

Frequency and Mutual Information

The above section makes explicit use of the frequency and the mutual information of a disjunct. It is useful to define these. Given a disjunct `(MA- & MB-)` let N`(MA- & MB-)` be the number of times that this disjunct has been observed. It will usually be an integer (except when obtained in certain unusual situations not discussed here). Let N`(*- & *-)` be the numer of times that any two-connector disjunct has been observed, as long as both connectors point in the - direction. That is,

$$N(*-\&*-) = \sum_{c_1 \in -, c_2 \in -} N(c_1 \& c_2)$$

the summation taking place over all connectors in the - direction. The frequency of observing `(MA- & MB-)` is then

$$p(MA-\&MB-) = \frac{N(MA-\&MB-)}{N(*-\&*-)}$$

The mutual information associated with the disjunct is then

$$m(MA-\&MB-) = \log_2 \frac{p(MA-\&MB-)}{p(MA-\&*-)p(*-\&MB-)}$$

The reason for this possibly unexpected form was developed earlier in this diary.

Semantics

There is another interesting issue that arises in the above discussion: the problem of syntax-semantics correspondance. Consider, for example, the sentence *"the dog treed* *the squirrel*". Here the word "tree" is used as a verb, meaning "the dog chased the squirrel up into the tree". Such sentences will cause the the word "tree" to accumulate disjuncts that the word "bush" will not have. Likewise, "*I'm bushed*" is a verb usage that has no analogous "tree" version. Thus, not only do the words "bush" and "tree" have different sets of disjuncts, but the differences are hiding semantic differences ...

There are several strategies that can be used to deal with this. More on this later.

Finding word pairs

We need a good way of finding word-pairs that are likely to be related. I think that perhaps the pattern matcher may be ideal for this. Details are TBD... but the basic idea is that the hypergraph for "tree: (MA- & MB-)" is connected to "big" because MB- is connected to "big", and "big" is connected to other lg-connectors, which in turn are connected to other disjuncts, which are then connected to other words. Thus, we search the local neighborhood of "tree", which causes us to dsicover the word "big", and then we search the neighborhood of "big" to find candidates such as "bush" which might be comparable to "tree". This search graph is not small, but it is not large: There may be thousands of words that are two hops away from "tree", but not millions.

Putting it all together

These are the things that need to be done:

1. compute the MI for the disjuncts
2. pick a common noun, compute the similarity scores for that word and every word that is linked to it. created ranked graph of similarity.
3. repeat step 2 for several different similarity formulas
4. repeat steps 2,3 for several verbs, several adjectives, several adverbs, several determiners, several prepositions.
5. Write code for creating grouping words into grammatical clusters.
6. Pick the most promising metric, and start clustering in bulk.

Step 5 requires writing a lot of code; it can all be written before the final metric has been determined.

The end.

That's all for now. More later.

Not LSA – 1 July 2015

NotLSA – a way to do LSA-like things without actually using LSA (Latent Semantic Analysis). Two very low-brow approaches, maybe well-known in the industry; I have no idea. Both of these approaches attempt to automatically extract keywords from documents. What cool about this is that its ... unsupervised; requires no training, and is based on very simple, proven ideas. Obvious, even: compute the mutual information between pairs of things ... between words and documents, between words and word-pairs, etc. Heh.

But how do we do this? How do we compute the MI between a page of text, and a word? No way to answer this without diving into the details.

Text-keyword correlation

Lets take a text, say – 1000 pages of .. something. Some corpus. We want to compute the mutual information between the page itself, and the words on the page. We do this by analogy to MI of word pairs.

Call the k'th page g_k. Count the number of times that word w_m appears on this page; let this count be N_{mk}. Define $N_m = \sum_k N_{mk}$ be the total number of times that the work w_m appear in the document, and let $N = \sum_m N_m$ be the total number of words in the document. Then, as usual, define probabilities, so that

$$p_m = P(w_m) = N_m/N$$

is the frequency of observing word w_m in the entire corpus, and

$$p_{mk} = P(w_m|g_k) = N_{mk}/\sum_m N_{mk}$$

be the (relative) frequency of the same word on page g_k. Notice that the definition of p_{mk} is independent of the page size. Pages do not all have to be of the same size. Define the mutual information as

$$\mathrm{MI}(g_k, w_m) = -\log_2 \frac{p_{mk}}{p_m} = -\log_2 \frac{N_{mk}N}{\sum_m N_{mk} \sum_k N_{mk}} = -\log_2 \frac{p(m,k)}{p(m,*)p(*,k)}$$

This is essentially a measure of how much more often the word w_m appears on page g_k as compared to its usual frequency. The highest-MI words are essentially the topic words for the page. The right-most form introduces a new notation, to make it clear that it resembles the traditional pair-MI expression. The notation is

$$p(m,k) = \frac{N_{mk}}{N}$$

so that

$$p(m,*) = \sum_k p(m,k) \qquad \text{and} \qquad p(*,k) = \sum_m p(m,k)$$

are the traditional-looking pair-MI values.

TODO: – this does not have the feature-reduction/word-combing aspects of LSA...

Variants

Instead of working with words, we could work with word-pairs, which is a stand-in for working with (named) entities. Thus, we can identify if a named entity occurs in a document more often than average.

Unsupervised Morphology Learning References

Here's some:

- John Goldsmith, "The unsupervised learning of natural language morphology", Journal Computational Linguistics archive Volume 27 Issue 2, June 2001 Pages 153-198 MIT Press http://delivery.acm.org/10.1145/980000/972668/p153-goldsmith.pdf
- John Goldsmith, "An algorithm for unsupervised learning of morphology" Natural Language Engineering Volume 12 / Issue 04 / December 2006, pp 353-371 Cambridge University Press DOI: http://dx.doi.org/10.1017/S1351324905004055 http://people.cs.uchicago.edu/~jagoldsm/Papers/algorithm.pdf
- Survey Article Unsupervised Learning of Morphology Harald Hammarström Lars Borin http://www.mitpressjournals.org/doi/pdf/10.1162/COLI_a_00050

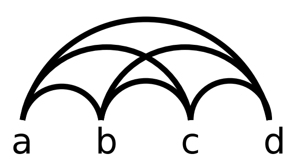
Predicate-Argument structure

Here's one:

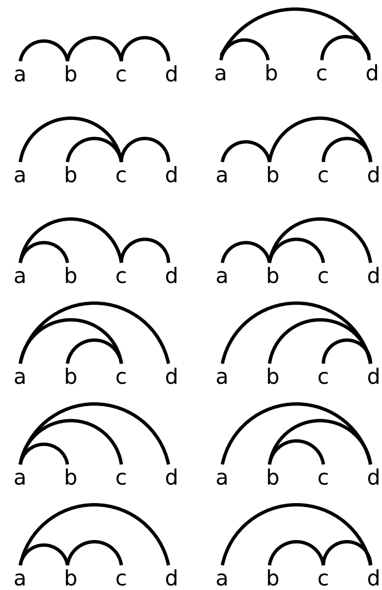
- The Darwinian evolution of natural language comes from a combination of Expressive FSM's and Lexical predicate-argument FSM's within the human brain. Shigeru Miyagawa, Robert C. Berwick and Kazuo Okanoya "The emergence of hierarchical structure in human language" Front. Psychol., 20 February 2013 | http://dx.doi.org/10.3389/fpsyg.2013.00071 http://alpha-leonis.lids.mit.edu/wordpress/wp-content/uploads/2014/01/shigeru-berwick-kaz-frontiers13.pdf

Edge-counting 27 March 2017

Counting edges in a clique is not the same as counting edges in planar trees. The diagram below shows the clique of a four-word sentence. The "words" are 'a', 'b', 'c' and 'd'. There are a total of six edges, with one edge between every possible word-pair. Each edge occurs only once.



Pair counting in planar diagrams gives different results. The diagram below shows the twelve planar trees, containing no cycles, that can be formed by parsing a sentence of four words.



The general formula for the number of different planar dependency parses is

$$\frac{1}{2n-1}\binom{3n-1}{n-1}$$

This formula is given by Deniz Yuret in "Lexical Attraction Models of Language" ISCIS 2006 (http://www2.denizyuret.com/pub/lex-attr/lam-iscis06.pdf)

It is important not to confuse this with the "matrix-tree theorem" aka Kirchoff's Theorem, which counts the number of spanning trees of a graph. In short, it states that the number of spanning trees is equal to any cofactor of its Laplacian matrix. In our case, we are dealing with a complete graph (a clique) and so on might hope that Cayley's formula applies. In fact, neither theorem works, because we are interested in non-self-intersecting planar trees, constrained by linear word-order.

There are 36 edges grand total, and these are unequally distributed. The counts are:

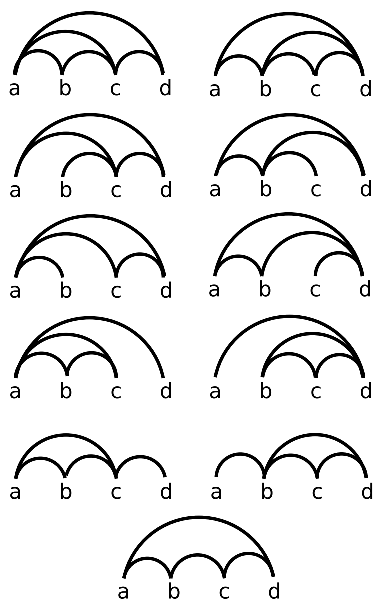
word-pair	count
(ab)	7
(bc)	7
(cd)	7
(ac)	4
(bd)	4
(ad)	7

Note that the most frequent edges occur almost twice as often as the least-frequent edges. The distribution, by length, is:

Length	Count
1	21
2	8
3	7

Note that the progressively-longer edges get less frequent.

If graphs with cycles are also allowed, (but no edge crossings) then, in addition to the above, there are eleven more diagrams. These are shown below.



Again, we count the number of edges, as before. The 'tree' column shows he counts from before; the loop count count the edges from the additional eleven diagrams; the total is just that.

word-pair	tree	loop	total
(ab)	7	9	16
(bc)	7	9	16
(cd)	7	9	16
(ac)	4	5	9
(bd)	4	5	9
(ad)	7	9	16

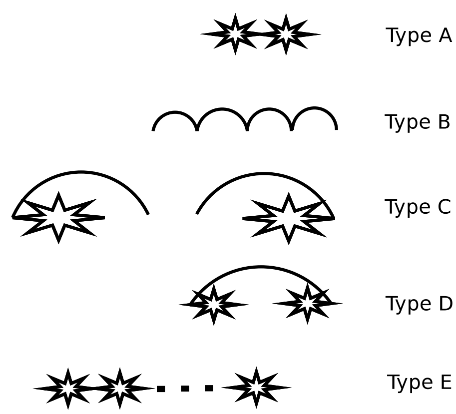
Likewise, the number of arcs of the given length is now given below:

Length	Count
1	48
2	18
3	16

What are the actual distributions, for these two cases? Begin by counting the number of planar trees. I currently do not know of any published work on this, so below, I make a half-baked attempt to count these myself. Its ... incomplete. Maybe there's some simpler approach.

Counting planar tree graphs

Let's count the number of planar tree graphs; i.e. those without any loops. First, we need a generic formula for sentences of length N. This is not so very easy. The diagram below shows one way to count. (I think what follows is correct, but I might be making a mistake. I am unaware of any literature that presents this information).



Here, the star represents some planar tree connecting all of the words of a smaller sentence. Assume that there are $T(n)$ such trees, connecting n words. Tree diagrams of of Type A are assembled by placing two adjacent smaller trees next to each other. Naively, one can then count how many such pairs there are; the issue is that the Type B diagram will occur mutiple times in this pairing; we would rather NOT count it with this mutiplicity. To avoid this problem, we should only allow pairs, such as Type A, to be assembled of sub-parts of the shapes C and D. Because of the over-arching arc, these can never result in double-counting. However, counting only pairs results in an under-counting: graphs of Type B never occur. Thus, one should count pairs, triples, and so on – graphs of Type E. Now we have a way of getting the formula. Define $D(n)$ as the count of the number of planar trees, connecting n words, having an arc connecting the first and last word: i.e. trees of type C or D. (Think "D = dome") One then has that

$$D(n) = \sum_{j=1}^{n-1} T(j)T(n-j)$$

It is convenient, here, to define $T(1) = 1$. The first and last terms of this sum then correspond to trees of Type C, while the middle terms are trees of type D.

To count trees of Type E, is is convenient to break this up into the problem of counting chains of length k, so that there are $C_k(n)$ trees, consisting of a sequence of k

domes, making up a total of n words. One then has that

$$T(n) = D(n) + C_2(n) + \cdots + C_{n-1}(n)$$

It's convenient, here, to define $C_1(n) = D(n)$. Writing down the $C_k(n)$'s requires some combinatorial magic. The first one is

$$C_2(n) = \sum_{j=2}^{n-1} D(j)D(n-j+1)$$

Next comes

$$C_3(n) = \sum_{j=2}^{n-1} \sum_{m=2}^{n-j+1} D(j)D(m)D(n-j-m+2)$$

which is awkward to write down. It's easier to count partitions of sets. Thus, what really is happening here is that the sums range over all k-way partitions of sets containing $n+k-1$ elements. Not the partition is NOT over n elements: to get connected graphs, we have to identify end-points of each link in the chain. Thus,

$$C_k(n) = \Pi_\sigma \cdots$$

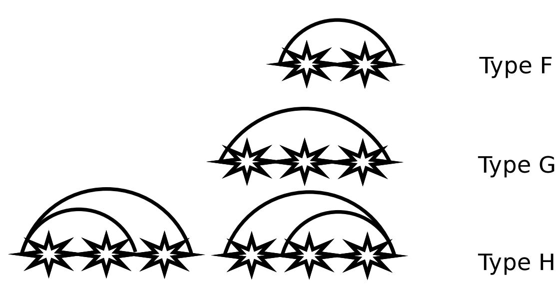
The table below summarizes the first few sums:

n	$T(n)$	$D(n)$	$C_2(n)$	$C_3(n)$	$C_4(n)$	$C_5(n)$
1	1	1				
2	1	1				
3	3	2	1			
4	12	7	4	1		
5	45	20	18	6	1	
6		123				1

Either I am computing this wrong, or the sequences are not in OEIS. Surprising!

Counting planar loop graphs

The above process can be repeated, except that this time, we consider the planar graphs containing loops. To get started, consider the diagram below.



Here, the stars represent either "domed" diagrams, or the empty set (a set containing no words and no edges. The type F concatenates two domes, and puts a dome over those, in turn. Since both of the stars are domed (or empty), it is impossible to add any additional edges to this graph. So, for graphs constructed out of a pair domes (one or both possibly empty), Type F is all that there is. For three domes in a row, there are only three ways of adding edges: these are shown in type G and H in this diagram. Again, this exhauses all possibilities. This process constructs both looped and tree diagrams. The general idea is to repeat this, for sequences of four or more stars.

The counting is similar to that before. Let $F(n)$ count the number of domed graphs, connecting n words. Let $G_2(n)$ count the number of type F graphs, made of two parts, and containing n words. Consulting the diagram, we have

$$G_2(n) = F(n-1) + \sum_{k=2}^{n-1} F(k)F(n-k+1) + F(n-1)$$

Likewise, let $G_3(n)$ count the number of graphs of type G and H, combined. Consulting the diagram, this has a more complex expression:

$$G_3(n) = \sum_{k=2}^{n-2} F(k)F(n-k) + \sum\sum F()F()F()... + \sum_{k=2}^{n-2} F(k)F(n-k) + ...$$

The total number of domed graphs having n words is then

$$F(n) = \sum_{k=2}^{n-1} G_k(n)$$

Let $S(n)$ be the count of a string of domed graphs, but NOT having connecting arcs: that is, graphs of type A or E.

A table of these is given below.

n	$L(n)$	$S(n)$	$F(n)$	$G_2(n)$	$G_3(n)$	$G_4(n)$	$G_5(n)$
1	1	0	1				
2	1	0	1				
3	4	1	3	3			
4	23				x		
5	156					x	
6	1162						x

English Dataset Sample 28 April 2017

Samples for English

This section reports on data collected for a small sample of English sentences, taken from Wikipedia, late April 2017. [1]. It was collected over the course of a few days,

[1]The 'en_snapshot' dataset

and so should be considered to be a medium-sized sample: larger ones, collected over weeks or months, are possible, as well as smaller samples collected over a few hours. Note that the current atomspace infrastructure does have some serious performance limitations: the atomspace is designed to be a very general-purpose hypergraph tool, and not a fast statistics-collecting tool.

There were 106696 unique words observed in the dataset[2]. This number is fairly large, as it includes not only common nouns, but also surnames, geographical location names, and a variety of foreign-language words, as would be observed in typical wikipedia articles. These words were observed for a total of 24417409 times[3].

These were observed in 80613 sentences[4] with 15.88 parses per sentence[5]. On average, there were 19.07 words per sentence[6].

There were 9376710 (about 9M) unique 'clique pairs' observed[7] for a total of XXX observations[8]

Graphs. TODO.TODO graphs P(w), P(t,w), etc. show zipf

XXX

Define the relation $E(w_1, w_2)$ as being the relation that both words w_1 and w_2 occur at the ends of an edge in the same sentence, but in arbitrary order. It is symmetric: $E(w_1, w_2) = E(w_2, w_1)$. By this definition, one has that

$$N(E, w_1, w_2) = N(R, w_1, w_2) + N(R, w_2, w_1)$$

This is the symmetrized count. It is useful to mod out one of the two words, and to consider the sum

$$N(E, w) \equiv N(E, w, *) = \sum_{w_2} N(E, w, w_2)$$

This counts how often the word w occurs at one end or the other of a word-pair. It is a distinct count from $N(w)$, which, by definition, counts only once per word in a sentence.

Left-right asymmetry

This section explores how often a given word occurs on the left side of a word pair, vs. how often it occurs on the right. This, of course, depends on the word. If sentences and words were randomly generate, one would expect that a given word would occur on the left, or on the right, exactly half the time. That is, in a random world

$$N(w, *) \approx N(*, w)$$

Of course, this cannot hold for a given human language: the exclamation point, question mark and period occur exclusively on the right hand-side of any randomly-generated

[2]Obtained with (fetch-all-words)(length (get-all-words))

[3](get-total-atom-count (get-all-words)) which is the same as (total-word-observations)

[4](get-sentence-count)

[5](/ (get-parse-count) (get-sentence-count))

[6](avg-sentence-length)

[7](fetch-clique-pairs) (length (get-all-clique-pairs))

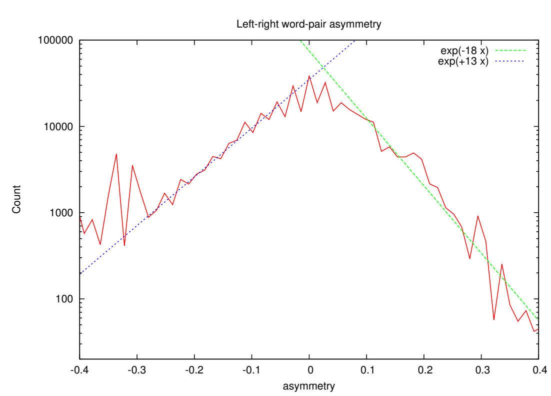
[8](get-total-atom-count (get-all-clique-pairs))

pair. This is not limited to punctuation: for Japanese, sentences usually end in a verb, and thus, for Japanse verbs v, one expects that $N(v,*) \ll N(*,v)$.

Some typical values for English words are given in the table below. Here, by defintion, $N(w) \equiv N(*,w) + N(*,w)$ is the number of times the word w is observed in a pair relation. The word-pairs were generated by creating random parse trees of the sentences in the data-set, and then counting a pair, if two words are connected by a parse link.

w	$N(*,w) - N(w,*)/N(w)$
how	-0.0058
when	-0.0051
,	-0.0021
will	-0.00114
usually	-0.00112
a	-0.00085
the	0.0035
finally	0.0043
word	0.0094
hope	0.0128
?	0.3197
.	0.8578

A histogram summarizing the above table is shown below. It bin-counts the asymmetry $N(*,w) - N(w,*)/N(w)$. About 100 bins are used, from an observation of 8.88 million distinct English-language word pairs, where were observed for a total of 420 million times. These pairs connected about 400K distinct words. The number of distinct words is large, because these include surname, and geographical names, as well as an assortment of foreign-language words as might be encountered in a sampling of English wikipedia pages. (This is the dataset collected by Rohit, summer of 2015).



Observe that the y-axis is drawn with a logarithmic scale. The two sides to this peak are conjectured to be linear. Two lines guessing at the slopes are indicated in the graph; the (natural logarithm) slopes are +13 and -18.

Total Entropy

Of some

Asymmetric Mutual Information

Reviewing some old results suggests that asymmetric mutual information should be attempted. For the results reported below, this will be defined as

$$AMI(w_l, w_r) = MI(w_l, w_r) / \log$$

Connector Sets 7 May 2017 (revised July 2017)

Tho better manage the size of this diary, this has been moved to its own file. See the "connector-sets-revised.lyx" file.

Abstract

This is a report on a dataset of disjuncts and connector sets, extracted from MST parses of a batch of sentences. First, a recap of what these are, then a characterization of the database contents, and finally, a report on the grammatical similarity of words in the dataset.

MST parsing algos

There are multiple MST algos, some better than others. A short list with some references.

- The current implementation in the (opencog sheaf) directory is an MST algo for generating projective MST trees from undirected edges. Its a simple-minded projective adaptation of Borůvka's algo (see wikipedia for a description). I just measured it to run at $O\left(n^3\right)$ for n words. See atomspace/opencog/benchms/README.md.
- The $O\left(n^3\right)$ is for the case of arbitrary-length links. If the scoring function is altered to give bad scores to link lengths >6 long, then the algo kicks over to $O\left(n^{2.3}\right)$ after about $8 < n$ or so. Awesome! See graphs in atomspace/opencog/benchms/ for a better look.
- This isn't bad, per se, since Yuret published his best projective MST algo which ran at $O\left(n^3\right)$ for n words. So we are in the right ballpark...
- The state-of-the-art projective algo (for directed edges) is supposed to be Eisner's algo, which runs at $O\left(n^3\right)$ for n words. So we are still in the right ballpark. Eisner, Jason, 1996. "Three New Probabilistic Models for Dependency Parsing." In Proceedings of the 16th Conference on Computational Linguistics (CoLING 96) . Saarbruecken: ACL, 340–345.

- The Chu-Liu-Edmonds algorithm finds (non-projective) spanning trees in directed graphs. It is described by Ryan McDonald, Fernando Pereira, Kiril Ribarov Jan Hajič, “Non-projective Dependency Parsing using Spanning Tree Algorithms” http://www2.denizyuret.com/ref/mcdonald/nonprojectiveHLT-EMNLP2005.pdf Its supposed to run in $O\left(n^2\right)$ time.

- An non-projective algorithm that is “super-linear” in the number of edges is described by Effi Levi, Roi Reichart, Ari Rappoport, “Edge-Linear First-Order Dependency Parsing with Undirected Minimum Spanning Tree Inference” (2016) https://www.aclweb.org/anthology/P/P16/P16-1198.pdf Since this is edge-linear, I think that, for us, it claims $O\left(n^2\right)$ for n words. (Since, for us, we don’t know, a priori, if we have an edge, or not). Its also not projective. https://arxiv.org/pdf/1510.07482.pdf

Dataset report 3 June 2017

Some summary reports from various different datasets.

Word-Pair datasets

First, datasets that hold word-pairs, parsed using the LG “ANY” link type: i.e. random parse trees.

Size	Pairs	Obs’ns	Obs/pr	Entropy	MI	Dataset
395K x 396K	8.88M	418M	47.0	19.28	3.02	en_pairs_sim
138K x 140K	4.89M	140M	28.6	17.73	2.03	en_pairs_tone_mst
183K x 187K	8.05M	268M	33.3	17.83	1.84	en_pairs_ttwo_mst
425K x 432K	15.2M	557M	36.6	18.32	1.93	en_pairs_tthree
134K x 135K	5.54M	174M	31.4	17.67	1.94	en_pairs_rone_mst
185K x 188K	8.95M	321M	35.9	17.77	1.79	en_pairs_rtwo_mst
428K x 434K	16.4M	639M	38.9	18.27	1.90	en_pairs_rthree
839K x 851K	30.1M	1.35G	44.9	18.54	1.84	en_pairs_rfive
619K x 581K	27.9M	1.25G	44.7	18.65	1.80	en_pairse_cfive_mst
158K x 159K	5.92M	729M	123	18.45	2.02	zh_pairs_sone
60K x 60K	1.68M	87.8M	52.3	17.47	2.88	zen_pairs
351K x 351K	14.6M	632M	43.4	19.35	3.37	zen_pairs_three

The legend is as follows:

Size The dimensions of the array. This is the number of unique, distinct words observed occurring on the left-side of a word pair, times the number of words occurring on the right. We expect the dimensions to be approximately equal, as most words will typically occur on both the left and right side of a pair.

Pairs The total number of distinct pairs observed.

Obsn's The total number of observations of these pairs. Most pairs will be observed more than once. Distributions are typically Zipfian, as previous sections point out.

Obs/pr The average number of times each pair was observed.

Entropy The total entropy of these pairs in this dataset, as defined previously: for word-pairs (w_L, w_R) it is $H = -\sum_{w_L,w_R} p(w_L, w_R) \log_2 p(w_L, w_R)$.

MI The total mutual information for the pairs in this dataset, as defined previously: $MI = \sum_{w_L,w_R} p(w_L, w_R) \log_2 [p(w_L, w_R)/p(w_L, *)p(*, w_R)]$

The datasets are as below.

en_pairs_sim This contains text parsed from Wikipedia, only. As noted previously, Wikipedia is painfully short of verbs and pronouns. Compared to the Gutenberg datasets below, it is also very rich in foreign words and proper names (product and brand names, geographical place names, biographical mentions and other named entities). Issue: missing connectors the LEFT-WALL.

en_pairs_tone_mst Text from Project Gutenberg "tranche one", mostly all "famous authors", popular, well-known 19th century books. Includes six modern sci-fi/fantasy novels from other sources, and some 20th century non-fiction, including a military appraisal of Vietnam.

en_pairs_ttwo_mst Tranche two - Everything from tranche one, plus fan-fiction from http://archiveofourown.org. Most of the selected texts were 10K words or longer. See the 'download.sh' file for the precise texts. Issues: tone_mst and ttwo_mst are missing connectors the LEFT-WALL. Certain types of punctuation is mishandled.

en_pairs_tthree Tranche three - Everything in tranche two, plus several hundred of the most recently created Project Gutenberg texts, whatever they may be. See the 'download.sh' file for the precise texts. The _mst version has the same issues that ttwo_mst has, although some connectors to LEFT-WALL do get added. The _mst version is probably not useful for similarity measurements.

en_pairs_rone_mst Same as en_pairs_tone_mst, but with minor issues fixed. However, links to LEFT-WALL still missing.

en_pairs_rtwo_mst As above, tranche 1 & 2.

en_pairs_rthree As above, tranche 1,2 & 3.

en_pairs_rfive As above, tranche 1,2,3,4 & 5.

en_pairs_rfive_mtwo The MST parses of tranches 1-2, performed on the word-pairs computed from en_pairs_rfive. That is, the word-pair stats for tranches 1-5 were accumulated to completion first, before the MST parsing is started.

en_pairs_rfive_mst As above, except that this is the MST parse of all of the tranches 1-5. That is, the MST corpus is the same corpus as the word-pair corpus. (Same as en_pairs_rfive_mfive, but with dj marginals)

en_pairs_cfive_mst Same as en_pairs_rfive_mst above, but with all words that contained bogus infix punctuation removed. (hyphenated words remain, as do decimal numbers and abbreviations).

zh_pairs_sone A parse of Mandarin Wikipedia, with each individual character (hanzi) treated as a single item (so that, during pair-counting, pairs are formed between items). Non-Chinese characters are grouped into words in the normal way, by splitting according to white-space (and punctuation). Thus, the total dimensions of the dataset are given by the number of observed Chinese characters (hanzi) plus the number of observed non-Chinese words (and punctuation).

zen_pairs A parse of a small set of Mandarin novels, with text segmented into words by external third-party tools (provided by Ruiting).

zen_pairs_three Word-pairs for tranch-1 and tranche-2-part-1 of Mandarin novels, segmentation by Ruiting.

Now, for some commentary, as to the summary stats. For English, as the number of pair observations increase, so do the number of unique, distinct words. The relation even seems to be linear: double the number of pair observations, and the number of different words also increases. This suggests something Zipfian at work. The explosion of words is hypothesized to be given names, although these datasets all fail to split hyphenated words, and so some may be due to that. The point is that the average observations per pair increases with difficulty, and the entropy and MI does not budge at all.

Comparing the English _sim dataset to the _rone, _rtwo and _rthree datasets does provide some contrast: The _sim dataset, built from Wikipedia, is distinctly different from the Gutenberg datasets. Certainly, the prose style in the two datasets is quite different, with Wikipedia consisting of statements of facts (“is”, “has” relational statements) concerning a broad range of named entities, whereas the Gutenberg texts are primarily narrative adventures (“did”, “went” activity statements) involving fictional personages.

Comparing English to Chinese is very interesting. The Chinese “zh” dataset has three times, almost four times more observations per pair; equivalently about 3-4 fewer “words”. This is partly due to the fixed number of ideograms in the language. Remarkably, the entropy and MI are untouched. This suggests that the entropy and MI are capturing something about the human nature of language use, as opposed to something descriptive of the language itself. However, a lot more data would be needed to see if this is really true.

By contrast, the “zen_pairs” dataset, where the Mandarin was pre-segmented into words by 3rd-party tools, behaves much more like English in it’s statistics. This is also evident in the table below, where the “zen” dataset behaves like “en”, and not like “zh”.

There’s something else interesting going on, shown in the table below.

Size		Support		Count		Length		Dataset
L	R	L	R	L	R	L	R	Name
158K	159K	6819	6411	548	487	41.7	37.7	zh_pairs_sone
60K	60K	8170	8702	191	156	18.1	15.1	zen_pairs
351K	351K	28.8K	27.1K	239	203	22.9	19.8	zen_pairs_three
619K	581K	53.8K	73.1K	282	254	34.7	25.9	en_pairs_cfive_mst
839K	851K	80.6K	80.6K	249	230	28.2	24.5	en_pairs_rfive
428K	434K	45.6K	45.1K	208	187	22.9	19.4	en_pairs_rthree
185K	188K	24.7K	23.8K	199	173	21.5	17.8	en_pairs_rtwo_mst
134K	135K	17.4K	17.4K	143	129	16.6	14.0	en_pairs_rone_mst

The columns are as follows:

Size The left and right dimensions, as before. Viz, the number of unique, distinctly different words observed on the left and the right side of a pair. Viewed as a matrix, this is the number of columns and rows in the matrix.

Support The support is the average number of word-pairs that a word participates in (on the left, or on the right). Viewed as a matrix, this is the average number of non-zero entries in each row or column. Viewed as (row or column) vectors, this is the "support" of a (row or column) vector. Mathematically, this is the l_0 norm of each vector: $|(w_L,*)| = \sum_{w_R} (0 < N(w_L,w_R))$ and likewise $|(*,w_R)| = \sum_{w_L} (0 < N(w_L,w_R))$.

Count The count is the average number of observations that a word-pair was observed, for a given word. Viewed as a matrix, this is the average value of each non-zero entry (averaged over rows, or columns). Viewed as vectors, this is the l_1 norm divided by the l_0 norm. The l_1 norm is just the wild-card counts $N(w_L,*)$ and $N(*,w_R)$, where as always, the wild-card counts are defined as $N(w_L,*) = \sum_{w_R} N(w_L,w_R)$. The count shown in the table is then the average count: $N(w_L,*)/|(w_L,*)|$ for the rows, and likewise for the columns.

Length The length is the average length of the row and column vectors. This is the l_2 norm divided by the l_0 norm. The l_2 norm is just the standard concept of the length of a vector in Euclidean space. Here, $L(w_L,*) = \sqrt{\sum_{w_R} N^2(w_L,w_R)}$, and likewise $L(*,w_R) = \sqrt{\sum_{w_L} N^2(w_L,w_R)}$. The length is interesting, because it "penalizes" word-pairs with only a small number of counts. The act of squaring the count has the effect of giving much higher "confidence" to large observation counts: a word-pair observed twice as often is given four times the credit. The length shown in this table is the "average" length: it is $L(w_L,*)/|(w_L,*)|$ for the rows, and likewise for the columns.

So here's what is so interesting in this table: the support, for Chinese, is outrageously different than it is for English. For a given item (hanzi, for Chinese, word, for English), the Chinese hanzi participates in three to four fewer item-pairs! Since pairs are formed on a sentence-by-sentence basis, this means that the variety of different hanzi that can occur in a single sentence is much more constrained, much more strongly correlated. Now, perhaps this comparison is not quite valid: because we are not comparing words to words, but rather English words to Chinese "morphemes" (in the sense that Chinese words are typically composed of 1, 2 or 3 hanzi). Still, its interesting and surprising. This has knock-on effects: the observational counts are much higher, as are the average lengths. It would be interesting to repeat the previously given analysis of the various distributions, and see how they differ.

Disjunct datasets

Next, datasets that hold disjuncts. This section used to report more data, but it was all flawed: the MI had a minus sign in it, causing all computed disjuncts to be maximally bad. Despite this, the results were similar to the below: observations and entropy fit in line, as expected. The H_{left} entropy values were lower, hovered around 15, and the MI was in the 3-5 range, while H_{right} was unchanged and fit in line. You can find the original data in the git commit 9244905afdff191a39af8c5a6deab592d5a1558c.

Size	Csets	Obs'ns	Ob/cs	Entropy	H_{left}	H_{right}	MI	Notes
37K x 291K	446K	661K	1.48	18.30	16.00	10.28	7.98	en_pairs_sim
137K x 6.24M	8.63M	18.5M	2.14	20.96	19.14	9.71	7.90	en_pairs_rfive_mtwo
522K x 25.2M	34.2M	77.8M	2.27	22.82	20.92	10.09	8.18	en_pairs_rfive_mst
445K x 23.4M	31.9M	69.4M	2.18	23.09	21.14	10.11	8.16	en_pairs_cfive_mst
60K x 602K	801K	1.19M	1.48	18.86	17.99	10.13	9.26	zen_pairs_mst
85K x 4.88M	7.02M	17.8M	2.54	20.48	17.06	9.52	6.10	zen_pairs_three_mst

An updated legend for the columns:

Size The dimensions of the array. The left dimension counts words, the right dimension counts the number of unique, distinct pseudo-disjuncts.

Left-Right The left and right entropies, as defined previously. Note that $MI = H - H_{left} - H_{right}$ holds, by definition. Not given for the word-pairs table, because these two are nearly equal, and are half the difference between the entropy and the MI.

Note how the MI is considerably larger than that for the word-pairs. Higher MI implies a stronger correlation, and this is good: this suggests that the disjuncts are capturing meaningful structures in the language.

The behavior of the "zen" dataset might be explained by two issues that this dataset has. The smaller "zen_pairs_mst" dataset is tiny, with a large fraction (most?) words observed only once, most disjuncts observed only once, and so the high MI being a false signal, an artifact of the tiny size of the set. By contrast, the unexpectedly low MI on the "zen_pairs_three_mst" dataset might be blamed on the 3rd-party word segmentation tool. It is known not to be very accurate, and the low MI might be a by-product of that.

Thresholding PCA Classifier

The next step is what I've called "clustering" in the past, but it really needs to be something more like factor analysis, or better yet, sparse PCA. Except that's not right, either.

What is needed is a recognizer,as follows. Consider $\overrightarrow{b} = \sum_n b_n w_n$ be a vector, with the w_nT being individual words, and the b_n being weights. Plain-old Principal Component Analysis (PCA) computes real-valued weights b_n. It's problematic, because potentially all of the weights are non-zero for all of the words. Sparse PCA computes real-valued weights b_n such that only some small number of them are non-zero. This is much better. But what is really needed is a classifier: a set of b_n that are either zero or one, indicating the membership of a word w_n in some class of words. (Note, by the way, that a word might belong to multiple classes, for example, according to its part-of-speech, or it's meaning.) This suggests a neural-netish variant on iterative PCA, described below. But, before giving this, some general remarks.

Preliminary comments

The definition of PCA requires a matrix X that connects columns and rows in some way. In the conventional definition, it is a matrix connecting variables and measurements. The variables (the features being measured) are organized in the columns; the measurements in rows. The PCA algorithm effectively computes the eigenvectors of the matrix X^TX, with X^T denoting the transpose of X.

The matrix X^TX is proportional to the covariance matrix between the different features being observed. The principal component is the the direction of the greatest variation in this matrix.

What plays the role of X in the current situation, and how should the principal component be understood and interpreted?

What we have on hand, foundationally, is the frequency matrix P with components $p(w,d)$ connecting words with disjuncts. It was defined previously as $p(w,d) = N(w,d)/N(*,*)$, and where $N(w,d)$ is the number of times word w has been observed with disjunct d. As noted earlier, $N(w,d)$ is very large and very sparse: typically $200K \times 4M$ in recent datasets, with only 1 entry out of 2^{15} being non-zero.[9] The extreme sparsity indicates that a power-iteration algorithm will be the most efficient for implementing a PCA algorithm.

What we will examine will be the results on several different kinds of matrices M derived from (constructed from) the base data matrix P. In all cases, the features are words, and so in all cases, it is appropriate to write $X = M^T$; that is, we work mostly with the transpose of the matrix X as usually given in standard texts. This follows from the standard Link Grammar dictionary: the word is followed by the disjuncts it can employ.

[9] I plan to send out the revised, expanded statistical analysis "real soon now".

PCA of the frequency matrix

Should we identify P and X^T, so that $X^TX = PP^T$? We can, but then we don't get what we want. Computing the principal component of this matrix for a recent dataset (see later section below), we get the following vector shown below. The "weight" gives the magnitude of the vector component. The other two columns are the support for the word, and the number of observations, and are shown for comparison.

XXXX the table below is still from the broken corpus!!!! discard it!!!

word	weight	$\lvert(w,*)\rvert$	$N(w,*)$
.	0.9358	2031	341112
the	0.1212	1403	106378
and	0.1098	1225	96276
to	0.1035	1276	96308
”	0.0920	506	45809
,	0.0904	1703	111982
a	0.0842	958	73760
in	0.0808	750	56751
of	0.0783	890	64753
his	0.0666	691	48728
it	0.0567	606	44211
with	0.0531	480	33681
him	0.0482	425	30345
that	0.0464	729	49714
for	0.0450	479	33092

What does this mean? What can we do with this? Why is the weight of the period so high? In essence, this vector is stating that the greatest variety of disjuncts are associated with the period. Since periods are sentence enders, and every sentence has one, a link to the end of the sentence will attach to just about any word. That is, the period almost single-handedly accounts for almost all of the variance of the disjuncts in the dataset. The rest of the list is filled out with words that also attach freely and easily to just about anything: "the" should attach only to nouns, but common nouns wildly outnumber all of the other parts of speech put together. Similar remarks for "and" and "to". The comma can connect in a large variety of situations, and the closing quotation mark ” behaves much like a sentence-ender (This particular dataset contains a lot of dialog). The two columns labeled as $|(w,*)|$ and $N(w,*)$ confirms this interpretation: so, $|(w,*)|$ is the total number of unique, different disjuncts that were observed with w, and $N(w,*)$ is the summation over all of the counts with which these disjuncts were seen.[10] The top words, in terms of the variety and number of disjuncts, are more or less the makeup of the principal component of PP^T. This should not be a surprise. Anyway, this is not what we wanted: we want to classify sets of similar words; discovering which words account for the greatest variation in disjuncts is of secondary interest.

[10] If these numbers seem small, it is because they were taken from a sharply filtered dataset, the en_pairs_ttwo_mst dataset with the (50,30,10) cut applied. This cut is discussed later, below.

PCA of the cosine similarity

We had previously defined the cosine similarity of two words as

$$\text{sim}(w_1, w_2) = \frac{\sum_d p(w_1, d) p(w_2, d)}{\sqrt{\sum_d p^2(w_1, d)} \sqrt{\sum_d p^2(w_2, d)}}$$

and so, perhaps we should use this as the basis for judging the similarity of words. This suggests defining a matrix S with matrix components

$$S(w, d) = \frac{p(w, d)}{\sqrt{\sum_d p^2(w, d)}}$$

and then setting setting $X = S^T$ so that $X^T X = SS^T$. The idea here is that PCA allows a whole-set analysis of similarity, rather than point-wise similarity. That is, for normal clustering algorithms, one computes a large number of values for $\text{sim}(w_1, w_2)$ and then employs a clustering algorithm to categorize these, word by word. Here, instead, perhaps PCA can reveal entire clusters in one gulp, by simultaneously evaluating the similarity between all words in a cluster.

Power iteration converges at about half of the rate as for the frequency matrix, which is not a surprise, as the off-diagonal entries are closer to one-another. The PCA vector, however, is not all that different: 0.978 "." + 0.137 "," + 0.080 "the" + 0.068 "to" + 0.067 "and" + 0.039 "a" + ... and so on, the remaining entries filled out in roughly the same order, by the same words, as in the frequency PCA.[11] Why is this? It's worth taking a look at the matrix:[12]

	.	,	the	to	and	a
.	1	0.549	0.731	0.6435	0.668	0.627
,		1	0.711	0.824	0.888	0.765
the			1	0.790	0.744	0.896
to				1	0.906	0.857
and					1	0.755
a						1

So what is this saying? There are plenty of pairs that have greater similarity; here's an arbitrary sampling:

[11] Created as follows:

```
(define fsi (add-subtotal-filter psa 50 30 10))
(define pci (make-cosine-matrix fsi))
(define pti (make-power-iter-pca pci 'left-unit))
(define lit (pti 'left-iterate feig 8))
(pti 'left-print lit 20)
```

[12] Created with

```
(define poi (add-pair-cosine-compute fsi))
(poi 'right-cosine WORD-A WORD-B)
```

pair	sim
(he, she)	0.982
(this, that)	0.894
(run, walk)	0.878
(big, small)	0.908
(high, low)	0.910
(soft,hard)	0.809
(easy,hard)	0.846
(easy,soft)	0.749

So why aren't some of these in the PCA vector?

IMPORTANT:

Some readers have misunderstood this section. We are NOT doing PCA to obtain similarity! We are examining it as an algo for CLUSTERING! That is, instead of doing k-means clustering, or agglomerative clustering, or something like that, the idea is/was to use a thresholded PCA for CLUSTERING! and NOT for similarity, because we've already got reasonable adequate similarity. Higher-quality similarity might be nice, but that is of secondary importance, right now.

A bit of sheaf theory

I recently realized that much of what is being discussed here can be anchored in the vocabulary of a generic mathematical theory, namely, sheaf theory. Sheaves allow topological structure to be discussed in a local way: sheaves describe how the local neighborhoods of a point glue together, to form a manifold as a whole. Link Grammar disjuncts and connector sets are really just the stalks and germs of sheaf theory, in mild disguise. This can be seen as follows.

A standard way of expressing a graph is to list all of the vertexes in the graph, and to list all of the edges. Knowing these, one knows the graph. However, this is a global description: One does not know the local structure until one looks at specific vertexes, and what they attach to.

A different way of describing a graph is to make a list of pairs: a vertex v and all the edges that attach to it. More generally, one can consider pairs where a vertex v is attached to a vertex w by means of a path of length N or less.

$$(\text{vertex } v, \{w \text{ s.t. vertex } w \text{ is attached to } v\})$$

This describes the graph, as a whole, just as well as the simpler vertex+edge list does. However, the language is different: these pairs are presheaves, obeying all the axioms of a presheaf, e.g. the composition of restriction morphisms. They become a sheaf because they also obey the gluing or collation axiom as well: they can be glued together to form the original graph from which they were taken.

Thus, we can see that the set

$$(\text{vertex } v, \{\text{ edges attached to } v\})$$

is the same thing as a Link Grammar dictionary entry.

To be more precise, we need to distinguish the graph-sheaf that arises for a single sentence (from the dependency parse of the sentence) from the sheaf that arises from the entire language. If we take the language to consist of the set of all possible sentences, then the sheafification is to parse each of the sentences in the language, to get a dependency graph for each sentence, create the individual (word, connector-set) lists, and then take the quotient, identifying together all words that have the same spelling. This gives the sheaf of the entire language.

From what I can tell, this realization that language can be sheafified is not new; when the language is not a natural language, but is instead first-order logic, then it's sheafification gives the Kripke–Joyal semantics. According to Wikipedia, this was noted in 1965 for existential quantification. I don't know if this was ever noted for natural language before, but, as I've blathered on the mailing list before, this provides the "answer" to why the logic of Link Grammar appears to be modal logic: Link Grammar dictionary entries are sheaves, and the disjuncts are the different "possible worlds" that a given word can inhabit. For a natural-language sentence, "there exists" (existential quantification) a collection of disjuncts that can parse the sentence. Bingo.

I used to say that LG disjuncts had something to do with linear logic, because linear logic also has the general whiff of "possible worlds" around it. I now see that in fact its actually modal logic, and it is the language of sheaves that provides the direct route from Link Grammar, to modal logic. It would be very interesting to see all the details worked out.

Most interesting is perhaps this: the sentences of a language are observed with some a priori frequency or probability. What's the correct way of converting this to a probability distribution on the sheaf? Next, given a probability distribution on the sheaf, what is the corresponding probability distribution on the corresponding modal logic?

It seems to me that one could make this very generic: every language, and not just first-order logic, but any language, as considered in model theory, has a set of sentences. This sentences are composed of the terms in their term algebra, and these terms and how they connect, define a graph. That graph can be viewed in terms of sheaves, germs, stalks, etale spaces. This implies that every model, of model theory, has a corresponding cohomology. Writing this out could be interesting. Perhaps this has already been done; perhaps this is what topos theory is. But I suspect that it's not been sufficiently popularized: certainly, the standard computer-science textbooks that tell you what a language is do not tell you that it has a cohomology associated with it. And yet, this seems blatantly obvious, in retrospect, and naggingly it might actually be important for some reason or another.

Anyway: this is not just all-talk, no-action. I've written some code that implements some sheaf-based parsing on the atomspace. It is in the github atomspace repo. The README file there explains more.

Disjuncts are compositional

One reason that a disjunct representation of a graph is important is that disjuncts can be composed, so that the product is again a disjunct. This is in contrast to the vertex-edge model. where neither vertexes nor edges can be composed to obtain a new vertex or edge. That is, disjuncts form a monoidal category, and, specifically, a compact-closed category. This section tries to spell out very clearly what this means, if it is not already apparent.

So, for example: the Link Grammar parse for "this is a test" involves four disjuncts:

- this: S+
- is: S- & O+
- a: D+
- test: D- & O-

The determiner connectors D+ and D- can be composed to form a determiner D link, leaving a phrase that is still describable by a disjunct, a single object O- connector that can attach to verbs:

- "a test": O-

This has only one connector, but it is a perfectly ordinary connector, not differing from that which might be found on a single word. That is, Link Grammar makes no particular distinction between words and word-phrases. Using the same argument, it is why Link Grammar can work for morpho-syntax. One can continue composing:

- "is a test": S-

which has a subject connector S- that can connect to any subject. One can also, perhaps foolishly, perform some net-very-sensical disjuncts:

- "is a": S- & O+ & D+

or

- "this ... a": S+ & D+

This last has to use elipses as an awkward notation to indicate the projectivity constraint. Projectivity can be discard, provided some other means ensures a tight parse.

The point here is that the category of disjuncts can be taken to be a monoidal category, i.e. a category with a tensor product $\otimes$, with tensoring simply being the writing of two disjuncts next to each other. As the first three examples illustrate, the typical usage is not only to tensor together two disjuncts, but also to contract some of the connectors, as well.

The contractability of connectors into links means that the Link Grammar forms a compact-closed category. I've been through this one too many times, so won't try to sketch this here. It is a good homework exercise for novices.

Bob Coecke has written repeatedly on this topic, any one of his papers on pregroup grammars or closed monoidal categories applied to linguistics is adequate to grasp the concept. His notation is easily and readily translated into Link Grammar notation. The primary insigh is to understand that the Link Grammar connector letters should be understood as type labels: they provide a simple, easy notational device, overcoming the notational complexity that is otherwise required when presenting categorial grammars.

What's the point of all this? Well categorical grammars are all the rage, and the fact that LG is a categorical grammar seems to be frequently overlooked or misunderstood.

Conclusions

Conclusions from the above:

- Pair-wise similarity is very promising.
- The cosine similarity measure penalizes better, more accurate measurements, because better measurements are more likely to find dissimilarity. We need a better measure.
- PCA and sparse PCA, in the naive sense, applied to frequencies, or to cosine similarities, are inappropriate for classification. Its still possible that perhaps PCA applied to some sigmoid of the cosine similarity (e.g. cosine to the fourth power) might work better, but the selection of this sigmoid seems ad-hoc, and not anchored in any principles.
- First principles suggest something Bayesian, based on the Gibbs measure, maybe some sort of hidden multi-variate logistic regression. Hidden, because we don't know the grammatical categories in any a priori sense; we must deduce them.
- It would be great if someone worked out the precise details in going from sheaves to modal logic. This was already done, in 1965, for topos theory; no one has done this for natural language, though.

Other similarity measures

Its worth noting that one of the other similarity measures, such as qim and pim, discussed previously, can also be treated in this way. Note also that a matrix can be constructed so that X^TX becomes explicitly Markovian. This is given by

$$A(w,d) = \frac{p(w,d)}{p(*,d)} \quad \text{and} \quad B(d,w) = \frac{p(w,d)}{p(w,*)}$$

and then setting $X^TX = AB$. This has the property that

$$\sum_{w_1} [AB]\,(w_1,w_2) = 1$$

That is, the matrix AB is a Markov matrix. It is straight-forward to compute using the standard power-iteration algorithm employed throughout this section.

A modified PCA algorithm

This suggests a feed-foreward-neural-netish variant on iterative PCA, described below. I is entirely of my own design, cribbed from nowhere at all, just popped into my head as I sit still immobilized.

0. Pre-condition, filter the data. See step 10, below, for what to filter, and why.

1. Start with $b_n = 1/\sqrt{|w|}$ where $|w|$ is the number of unique words. This starting point is a unit-length vector, i.e. $\left|\overrightarrow{b}\right| = 1$. Its convenient to change notation, here, and write $b(w)$ for the value of $\overrightarrow{b}$ at word w. That is, $b(w_n) = b_n$ is the same thing.

2. Let M be the matrix for which the PCA is to be computed, with matrix components $M(w,d)$ for word w and disjunct d. This matrix is derived from (defined in terms of) the frequency matrix $p(w,d)$ describing the base dataset. Compute the double-sum

$$s(v) = \left[MM^T b\right](v) = \sum_d M(v,d) \sum_w M(w,d) b(w)$$

which is basically a pair of dot products. Its still a large, time-consuming computation, even for sparse vectors.

3. Normalize: set $\overrightarrow{b} \leftarrow \overrightarrow{s} / \left|\overrightarrow{s}\right|$ so that $\overrightarrow{b}$ is of unit length. In theory, this is not needed; in practice, each iteration can sharply shrink the value of $\overrightarrow{b}$, making it very small, eventually leading to exponent underflow.

4. Repeat these steps k times: go to step 2 and run the summation again. The repetition here is the 'power iteration' or the 'von Mises iteration' method for computing the largest eigenvalue of $\left[MM^T\right]$. It is not guaranteed to converge, and if it does, it might not do so quickly. But we deal with this in the next step, so its sufficient to keep k small, just enough to get a trend going. Another way to think of this is as a Markov process (specifically, a Markov chain). That is, the matrix $\left[MM^T\right]$ will behave essentially as a Markov chain, and iteration on it just identifies the primary Peron-Frobenius stable state (step 3 makes it Markovian, by preserving to total probability measure). That is, $\left[MM^T\right]$ defines a weighted adjacency matrix for a graph, and iteration creates a measure-preserving process (walk) on this graph.

5. After the above repetitions, apply some standard neural-net sigmoid function to $\overrightarrow{b}$. That is, set $b(w) \leftarrow \sigma(b(w))$ for some sigmoid. This has the effect of driving some of the elements to zero, and others to one.

6. Repeat this m times: go to step 2, and repeat steps 2-5. Viewing this as a dynamical system, the effect of the sigmoid function is to force the system into a block-diagonal form, with the vector $\overrightarrow{b}$ identifying a highly-connected block. Another way to look at this is as a graph factorization algorithm: the vector $\overrightarrow{b}$ is identifying a well-connected subgraph, which is only weakly connected to the rest of the graph. The vector (viewed as a measure-preserving dynamical system) is spending most of its time in one particular block. Again, $\left[MM^T\right]^k$, the k-th power iterated matrix from step 4, can be thought of as a surrogate for a weighted graph adjacency matrix. A third way of thinking of this is as an m-layer neural net, with the link weights between one layer and the next being given by

$\left[MM^T\right]^k$. All three ways of looking at this are essentially equivalent: a measure-preserving dynamical system, a chaotic and mixing process on a graph, or as an m-layer neural net. Pick your favorite.

7. Classify. Pass the vector $\overrightarrow{b}$ through the step function, i.e. $b(w) \leftarrow \Theta(b(w))$ where $\Theta(x) = 0$ if $x < 1/2$ and $\Theta(x) = 1$ if $x > 1/2$. The step function is a super-sharp sigmoid. This step identifies and isolates an active, well-connected subgraph of $\left[MM^T\right]$. It identifies a square block, of dimension $|b| \times |b|$ where $|b|$ is the total number of non-zero entries in this final $\overrightarrow{b}$. To belabor the point: the block-matrix is explicitly

$$B(v,w) = b(v)b(w)\sum_d M(v,d)M(w,d)$$

The non-zero elements of this final $\overrightarrow{b}$ identify a class of words that can be considered to be grammatically similar or identical. This is the "clustering" step.

8. Associated with this class of words is a disjunct set, the "average disjunct" for the class. It can be taken to be the set $\{d|0 < \sum_w b(w)N(w,d)\}$. The observed counts associated with this set can be taken to be $N(b,d) = \sum_w b(w)N(w,d)$ and the frequencies similarly: $p(b,d) = \sum_w b(w)p(w,d)$. From here-on, the set of words $b \equiv \{w|0 \neq b(w)\}$ can be treated as if it was an ordinary word, behaving like any other, with the indicated disjuncts, counts and frequencies.

9. Since words can have have multiple meanings, or rather, multiple different kinds of grammatical behaviors based on their part of speech, the identified words need to be subtracted, en block, from the matrix $p(w,d)$, and then the process repeated, to identify another class of words. Put another way, if b is to be added to the set of words, as "just another word", then the frequencies $p(b,d)$ have to be subtracted from the matrix P, and shunted to this new "word", so as not to loose the overall normalization. That is, one must preserve the identity $\sum_{w,d} p(w,d) = 1$. So define, in the next iteration

$$p(w,d) \leftarrow \begin{cases} p(b,d) & \text{if } w = b \\ p(w,d) - b(w)p(b,d) & \text{otherwise} \end{cases}$$

(Hmmm. This may not be right, its late and I'm tired). This still sums to the identity except that now some of the values might go negative, and we don't want that.

10. And so we get to what should be called step zero: We want to truncate, and discard the negative entries. This should have been carried out as an actual step 0: a pre-conditioning of the matrix: some noise filtering, e.g. discarding all words that were observed less than a handful of times, discarding rare or preposterous disjuncts. Pre-conditioning in this way will have the effect of removing some (possibly many) of the words from the matrix: the size of the matrix shrinks. This is the step where the actual dimensional reduction takes place: the size of the set of words is shrinking, as they get classified into sets.

11. Go to step 0 and repeat, until the preconditioning and noise-removal has left behind an empty matrix (or alternately, a matrix where all words have been classified into some group). So, for example, words which have only one part-of-speech or meaning would (hopefully should) get classified after just one step; words that are more complex, and have two parts of speech, would require at least two iterations. This is perhaps optimistic; I expect dozens of iterations to get anything vaguely accurate.

12. There's one more step. After the formation of the class b, we arrive at a situation where no (pseudo-)connectors connect to b directly. Instead, all disjuncts connect to words inside of b. But this is a problem: we don't know if any given connector actually connects to some $w \in b$ or if it connects to the same w, but outside of b. (e.g. if b are nouns, then does "saw+" connect to "saw" the noun, or "saw" the verb?) Thus, after some small number of iterations of step 11, there needs to be a re-parse of the entire text, using these newly discovered classes of words.

That's it. I think this should work fairly well. Clearly, there are many nested loops, and so this is potentially a very time-consuming computation. The number of iterations k and m need to be kept small, and the classification in step 11 needs to be kept greedy, because step 12 is expensive. An alternate strategy is to brutally precondition $p(w,d)$ to make it as small as possible; but this risks throwing out the baby with the bathwater: early on, we want to cluster together the rare, obscure, unused words as best as possible into large bins, and then devote large CPU resources to correctly classifying the remaining much smaller set of verbs and prepositions, which we know, *a priori,* to be complex and difficult, due to their grammatical variability.

Dataset

The previous dataset EN_PAIRS_SIM, analyzed above, proved to be inadequate in many respects. Thus, data analysis here resumes with a different, considerably larger dataset, collected on a higher-quality corpus. This will be the EN_PAIRS_TTWO_MST dataset, listed above. To recap, it is this one:

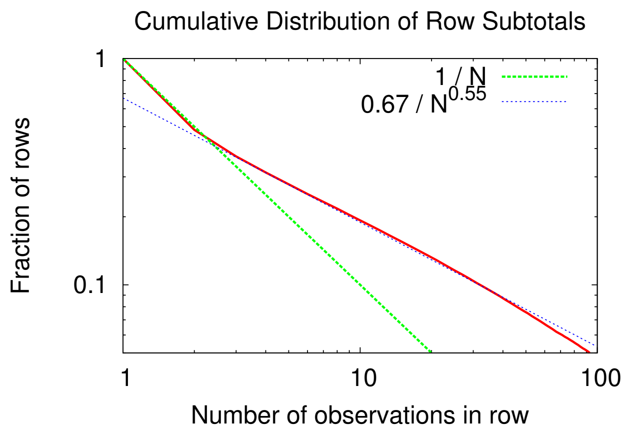
Size	Csets	Obs'ns	Ob/cs	Entropy	H_{left}	H_{right}	MI	Notes
176K x 3.4M	6.43M	14.3M	2.23	21.01	14.91	10.01	-3.91	en_pairs_ttwo_mst

To avoid accidental corruption of this dataset, a copy was made, in which assorted sporadic results are maintained. The copy is the EN_PAIRS_TTWO_SIM dataset.

Filtering, Step 0

The filtering performed in step 0 (described in step 10, above) removes some of the noise in the dataset. Basic filtering is implemented in the (OPENCOG ANALYSIS) scheme module, and specifically in the FILTER.SCM file. One can remove rows and columns that have subtotal counts less than a cutoff, and also remove individual entries that have fewer than some number of counts. By removing very infrequently observed connector sets, some of drivers of accidental similarity or dis-similarity between words should be ameliorated.

How much data does filtering actually discard? This dataset has 175559 rows. Each row corresponds to one unique, distinct word (columns correspond to disjuncts). Of these words, only 84984 were observed twice, or more: slightly less than half! Only 64882 words were seen three times or more; only 10% of the words were seen 32 times or more. The distribution is shown in the graph below.[13]



The fraction of rows with more than N observations drops a little faster than $\sqrt{N}$. Note that this graph is not scale-free; for larger datasets, the graph should progressively flatten. Since cumulative distributions are integrals of distributions, this is essentially the integral of some of the graphs shown before. A table of plausible cutoffs to use with this dataset is given below. There are three cuts one can make: discard words that are observed N or fewer times; discard disjuncts that were observed N or fewer times, and discard connector-sets (word-disjunct pairs) that were observed N or fewer times. These three cuts are given in the first three columns; the resulting dataset is given in the remaining columns.[14]

[13]From the cnt-obs-rows function.

[14]Stats can be gotten by creating the add-support-compute object on the filter object, and then invoking 'left-basis-size, 'right-basis-size, 'total-support and 'total-count.

Word cut	Dj cut	Cset cut	Size	Csets	Obs'ns	Ob/cs
0	0	0	176K x 3.4M	6.43M	14.3M	2.23
1	1	0	85K x 1.15M	4.10M	12.0M	2.94
4	4	0	49K x 145K	2.81M	9.51M	3.39
10	10	0	32.2K x 40.3K	2.42M	8.73M	3.60
30	30	0	17.9K x 14.4K	1.98M	8.05M	4.07
30	10	0	17.9K x 40.3K	2.26M	8.48M	3.75
30	10	4	17.9K x 40.3K	269K	5.52M	20.5
50	30	0	13.1K x 14.4K	1.88M	7.86M	4.19
50	30	4	13.1K x 14.4K	256K	5.41M	21.1
50	30	10	13.1K x 14.4K	101K	4.36M	43.4

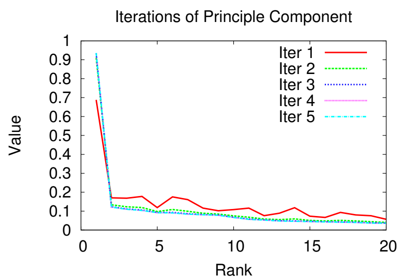
The last cut seems plausible for further work: it suggests that each disjunct is observed a fairly strong number of times; and that given the word/disjunct ratio, a lot of words are using disjuncts in similar fashion; thus, there should be a lot of similarity.

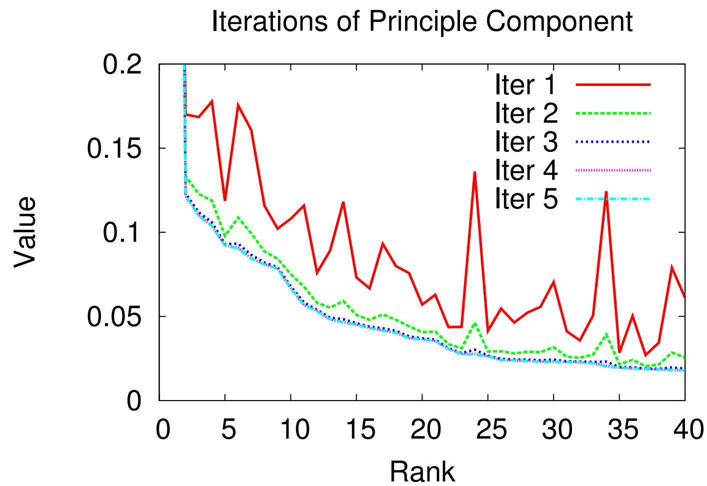
Note, by the way, that the previous sections carefully described entropy and mutual information distributions that no longer hold for the cut dataset. Filtering changes these!

Power iteration, Steps 1-4

Step 1-4 are implemented in the (OPENCOG ANALYSIS) scheme module, and specifically in the THRESH-PCA.SCM file. The implementation uses lazy evaluation to avoid unneeded computation, and caching of evaluation results to avoid repeated evaluations. This seems like the best way of working with the extremely sparse matrices involved.

Iteration appears to converge very rapidly. After three iterations, the ranking, by weight in the vector, appears to be established. This is shown in the figures below. Six iterations are performed, and the words w are then ranked according to the strength $b(w)$ in the sixth iteration. Then the values of $b(w)$ are plotted for the first five iterations, using this rank. After the third iteration, there is no discernible change in the weights.





In this case, the principle component is revealed to be[15]

word	weight
.	0.9358
the	0.1212
and	0.1098
to	0.1035
”	0.0920

The significance and interpretation of this vector was already discussed in a previous section.

Reboot: 23 June 2017

The cosine similarity gives OK pair-wise similarity results; the overlap similarity gives noticeably worse results. There’s no obvious matrix-based algorithm that will group together multiple words into a cluster, efficiently, in one shot. Its time for a rethink.

What are we trying to do here, really?

- Grammatical categories: By grouping multiple words into categories, we hope to discover grammatical categories of words that behave similarly, with respect to grammar.

[15]Computed as follows (simplified, various checks were done to verify correctness):

```
(define psa (make-pseudo-cset-api))
(psa 'fetch-pairs)
(define fsi (add-subtotal-filter psa 50 30 10))
(define pti (make-power-iter-pca fsi))
(define feig (pti 'make-left-unit (fsi 'left-basis)))
(define lit (pti 'left-iterate feig 4))
(pti 'left-print lit 20)
```

- Compression: By grouping multiple words into categories, we hope to compress the size of the overall dataset of connector-sets, without loosing much fidelity.
- Idioms: By observing word-disjunct pairs with high MI scores, we hope to discover idioms and set phrases.
- Meaning: By developing a coherent framework for working with graph sections (of which the connector sets are a special case), we hope to discover synonymous phrases.
- Reference resolution: By developing a coherent framework for working with graph sections, we hope to discover reference resolution (of pronouns and of given names) across multiple sentences.

These seem like they should be achievable. Preliminary results looking promising, but not yet great. What's the grand scheme of things?

- Discover that certain nouns refer to objects in the physical world.
- Discover that certain verbs refer to actions in the physical world.
- Discover that certain nouns refer to abstract, non-physical concepts.
- Discover the meaning of the verb-phrases "is-a", "has-a", "is-a-part-of", "belongs-to" ... or, generally, discover the meanings of prepositional phrases.
- Perform reasoning on relationships; specifically, on "is-a", "has-a", ... relationships.
- Develop a database of common-sense knowledge.
- Translate between multiple languages, by employing common-sense knowledge and reasoning.

The first two seem impossible without embodiment. The third bullet holds out hope that progress might be possible for textual-only analysis. The fourth bullet asks for an algebraic structure to be discerned: "is-a" relations are symmetric: "A is-a B" if "B is-a A", and it should be possible to data-mine such symmetric relations. Likewise for "next-to" and "near" relationships. It is at least plausible that such relations could be data-mined, with no *a priori* knowledge of the words. Anyway, this provides the setting for the initial grammatical tasks. So, back to the initial grammatical tasks.

Idioms: based on preliminary evidence, we could make lists of idiomatic phrases, now, based only on high-MI word-disjunct relations. But these are useless, until we can build a list of synonymous words.

Discerning synonymous words based on grammatical usage is tricky. First, it is often antonyms that get observed: e.g. the (black,white) pairs reported above. To discern antonymy, we would also need to discern is-a relationships, apply common-sense reasoning, and notice that antonyms never describe the same objects, never describe the properties of the same objects. So, antonym detection appears to be an advanced topic.

Clustering

We could go full-speed ahead on trying to discern grammatical categories, but for several issues:

- Merging two items into one necessarily entails a loss of information. That is, one necessarily has that $-(p_a + p_b)\log_2(p_a + p_b) \leq -p_a \log_2 p_a - p_b \log_2 p_b$. That is, information is necessarily lost. How can we minimize information loss?

- If a classification error is made early on, can it be spotted, and later corrected? What is the mechanism, and how might it work?

We can use the "information loss" to our advantage if the "lost information" is in fact just noise in the data. That is, the data is necessarily noisy, and the naive calculation of the entropy and mutual information encodes both that noise and the signal we are searching for. Clustering together items, and the associated information loss, is desirable if the loss results in the filtering out of noise. How can we characterize the noise in the observations?

Word Meanings

Take as an assumption that word-meaning is strongly correlated with grammatical usage. That is, "saw", the noun, has a different meaning than "saw", the verb. Thus, as a hypothesis, write

$$[w,m_1] = \{(w,d_a),\, (w,d_b),\, (w,d_c),\, \cdots\}$$

that is, the word w might have meaning m_1 whenever is used with any of the disjunts d_k from the indicated set. The meaning m_2 of a word will be associated with a different set of disjuncts. In general, the sets $[w,m_1]$ and $[w,m_2]$ will overlap.

Individually, $[w,m]$ is just a set, and has no weights or probabilities associated with it. However, if the disjunct (w,d) is observed, one can say that there is a frequency or probability $p([w,m]|(w,d))$ that the meaning m of word w is intended when (w,d) is obeserved. This is written as a conditional probability, so that one has

$$\sum_m p([w,m]|(w,d)) = 1$$

That is, given that (w,d) was observed, there must be some meaning m that was intended; the list of possible meanings is complete and exhaustive. I'm assuming that one possible meaning is "nonsense" or "junk" or "unknown"; just add it to the list of possible meanings.

One of the tasks is to discover the complete set of meanings $\{m_i\}$ for a word. Another task is to discern the probabilities $p([w,m]|(w,d))$.

Word Classes

Any given word might belong to one of many different word classes (noun, verb, ...) and the collected disjunct usage observations on that word will in general be a linear combination of such different usages. Distinguishing word-classes require untangling these relationships.

The setup for this problem is mostly identical to the problem above, except that the "meaning" m is re-interpreted as the word-class g, short for "grammatical class". That is, the above did not specify the definition of m, rather, it was presented in general terms. Here, likewise, but a stricter defintion is proposed for a "word class".

A word class g is a set of words $\{w_i\}$, together with a set of disjuncts $\{d_j\}$, such that all words in the word-class are commonly used with any of the disjuncts in the disjunct-set. That is,

$$g = (\{w_i\}, \{d_j\})$$

subject to the constraint that the vector

$$[w_1, g] = \{(w_1, d_a), (w_1, d_b), (w_1, d_c), \cdots\}$$

is judged to be similar to the vector

$$[w_2, g] = \{(w_2, d_a), (w_2, d_b), (w_2, d_c), \cdots\}$$

according to some similarity measure (e.g. cosine similarity). The idea is that any of the words in g use any of the disjuncts in g in similar ways.

Any given word might belone to multiple different classes g. For example, the verb "saw" will belong to a different class than the noun "saw". As a general rule, whenever $g_1 \neq g_2$ then the set of disjuncts in g_1 and g_2 will not overlap very much, if at all.

Assigning Word to Word Classes

Assigning a word to a word-class has a knock-on network effect. That is, words appear not only in isolation, but also as connectors in disjuncts. If two words are considered to be similar, then perhaps two connectors should be judged to be similar. If two connectors are similar, then perhaps the disjuncts they appear in are similar. If two disjuncts are similar, then perhaps some other pair of words can now be considered to be similar. The question arises of how far to follow this network effect, and how to assign cutoffs.

Note that the network can be traced in either one of two directions. Given a pair of similar words, one can ask if any of the disjuncts attached to those words are similar to one-another, or not. In the other direction, one can ask how similar connectors imply similar disjuncts. This is made more explicit below.

Starting with a single word, one can examine all of the disjuncts on that word, to see if any of them are similar. For example, the word "the" should have disjuncts "book+" and "novel+" on it, and one can ask if "book" and "novel" are similar. If so, they can be merged into a grammatical class $g = \{\text{book}, \text{novel}\}$ and the two disjuncts "book+" and "novel+" replaced by $g+$. The observation count (and likewise the probability) on (the, g+) should be the sum of N(the, book+) and N(the, novel+). The process is then repeated recursively, examining each of the words appearing in the disjuncts.

Alternately, one may walk the network in the "other" direction, and merge disjuncts as they appear in similar words. Let disjunct d be a sequence of connectors: $d =$

$(c_1, c_2, c_{3,} \cdots)$ and each connector is a word and a direction indicator: $c = (w, \pm)$. Given two similar words w_a and w_b, one can trace through the connectors $c_{a+} = (w_a, +)$ and $c_{b+} = (w_b, +)$ and likewise for the - direction. One then forms the set of all disjuncts in which c_{a+}appears:

$$\left\{d_k = (c_1, c_2, c_3, \cdots) | c_j = c_{a+} \text{ for some } j\right\}$$

Then, given one d_k, one constructs $\widetilde{d_k}$ so that c_{b+} replaces c_{a+}. One then constructs the set of all words that appear with $\widetilde{d_k}$:

$$\upsilon = \left\{w | N(w, \widetilde{d_k}) > 0\right\}$$

and ask whether any of these words already belong to the same grammatical class. If not, then they should be compared to one-another, to see if they might.

If a pair of words in υ already belong to the same grammatical class, then the two disjuncts d_k and $\widetilde{d_k}$ can be merged into one. Do this by forming the grammatical class $g = \{w_a, w_b\}$ and construct the connector $c_{g+} = (g, +)$. Then construct $\overbrace{\ d_k\ }$ so that that c_{g+} replaces c_{a+}, and replace both d_k and $\widetilde{d_k}$ by $\overbrace{\ d_k\ }$ in the relevant sections. The observation counts are copied over. The process is recursive, repeating for each pair of words judged sufficiently similar. Alternately, one might defer the creation of g until one has walked enough of the network to determine general similarity.

Merging Words to form Word Classes

After the grammatical behavior of two words is considered to be similar, how should a merged word-class be created? How should the merger be performed? There are several different ways in which words can be merged together to form word classes. These are reviewed below.

Merging is not straight-forward, because the process needs to result in an orthogonalization of the space for grammatical behavior. That is, the disjunct counts on any given word might be partly representative of its behavior as one of many different kinds of fine-grained parts of speech. That is, the goal is to take the disjuncts on any given word, separate them into two classes, and merge one class into an existing (or new) grammatical class, while leaving the rest as-is, which might be subsequently re-organized into some other class, ad infinitum.

Linear merging

Linear merging treats words as vectors, computing thier sum to define the new merged class, and then computing the perpendicular components as the left-over, un-accounted-for remainders. More precisely, the disjuncts are considered to be the basis elements of the vector space, and the count (or frequency) of each disjunct defines the vector.

Consider merging two words or word-classes w_a and w_b (that is, each of w_a and w_b can be either a word, or a word-class). Let $M(w, d)$ be a number associated with the word-disjunct pair (w, d). Typically it will be the count $N(w, d)$ that the pair was observed (or equivalently, the normalized frequency $p(w,d) = N(w,d)/N(*,*)$). The corresponding vector is then

$$\vec{w} = \sum_d M(w,d)\, \widehat{d}$$

where $\widehat{d}$ is the basis element. The merged word-class can then be defined as $\vec{w}_c = \vec{w}_a + \vec{w}_b$.

Erasure vs. orthogonal replacement

After the merger, there are two alternatives for what to do with $\vec{w}_a$ and $\vec{w}_b$ in the dataset. One alternative is to remove both $\vec{w}_a$ and $\vec{w}_b$ entirely. The other alternative is to compute the components of $\vec{w}_a$ and $\vec{w}_b$ that are orthogonal to $\vec{w}_c$, and replace $\vec{w}_a$ and $\vec{w}_b$ by these orthogonal components. That is, given a vector $\vec{v}$ (which might be $\vec{w}_a$ or $\vec{w}_b$), compute

$$\vec{v}_\perp = \vec{v} - \widehat{w}_c\,(\widehat{w}_c \cdot \vec{v})$$

where $\widehat{w}_c = \vec{w}_c / |\vec{w}_c|$ is the normalized unit vector pointing in the $\vec{w}_c$ direction.

The goal of maintaining the orthogonal components is that perhaps $\vec{w}_a$ and $\vec{w}_b$ have admixtures of other grammatical categories in them; what these are cannot be known a-priori. The discard option effectively discards these admixtures, hiding them from later iterations. This kind of hiding/data-destruction seems undesirable.

The orthogonal component potentially has negative coefficients appearing in it; it seems that these must be zeroed out to preserve "physicality".

Linear overlap

This would work like linear merging, described above, except that the intersection of the sets of disjuncts on these two words is computed first, and the vector basis is taken only over this intersected set. The intersection is presumably substantial, if the two words are grammatically similar. For the replacement step, one has three alternatives: total discard, which seems inappropriate (as the non-intersected disjuncts get discarded); partial discard, which discards only the intersected components, and orthogoanl replacement.

Noise

If an event is normally distributed, then we can characterize the uncertainty as being $1/\sqrt{N}$ after N observations. We don't actually know if our observations are normally distributed. Its not even clear quite how to even obtain the distribution. But lets assume they are. Then, given that $p(x,y) = N(x,y)/N(*,*)$ and estimating the noise to go as $\sqrt{N(x,y)}$ we get that the error in the frequentist probability estimate is given by

$$\frac{N(x,y) \pm \sqrt{N(x,y)}}{N(*,*)} = p(x,y) \pm \sqrt{\frac{p(x,y)}{N(*,*)}}$$

where only the pair observations $N(x,y)$ are considered to be noisy and the value of $N(*,*)$ is held fixed (the natural variation in it is ignored).

The error in the frequentist estimate of the entropy to be

$$-\log_2\left[p(x,y) \pm \sqrt{\frac{p(x,y)}{N(*,*)}}\right] = -\left[\log_2 p(x,y)\right] \pm \frac{1}{\log 2}\sqrt{\frac{1}{p(x,y)N(*,*)}}$$

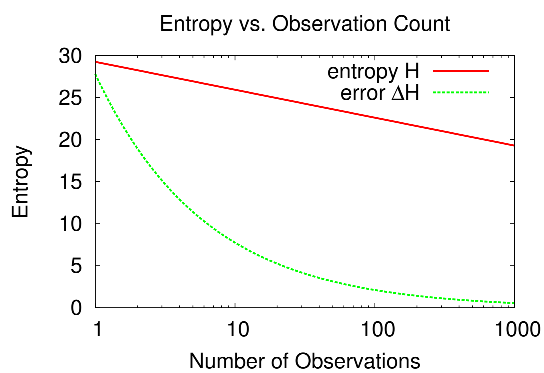
where the estimate $\log(1+\varepsilon) \approx \varepsilon$ is used for small ε. Summing to estimate the total entropy, one gets

$$H \pm \Delta H = -\sum_{x,y}\left[p(x,y) \pm \sqrt{\frac{p(x,y)}{N(*,*)}}\right] \log_2\left[p(x,y) \pm \sqrt{\frac{p(x,y)}{N(*,*)}}\right]$$

which expands out to

$$\Delta H = \frac{-1}{\sqrt{N(*,*)}}\sum_{x,y}\sqrt{p(x,y)}\left(\frac{1}{\log 2} + \log_2 p(x,y)\right)$$

What might these values be, in practice? As a worked example, consider the word-pair (big, deal) in the EN_PAIRS_RTHREE dataset. It is observed 1039 times, out of 638845863 pair observations (639M) total. Plugging and chugging, one gets $H = -\log_2 p(\text{big}, \text{deal}) = 19.23$ and $\Delta H = 0.552$ which seems to be eminently reasonable. The value of ΔH depends only on $N(x,y)$ and $N(*,*)$ and is graphed below, for fixed $N(*,*) = 639M$. It does not take very many observations to drive the uncertainty to a fairly small value.



foo

Error correction

If a classification error is made early on, can it be spotted, and later corrected? What is the mechanism, and how might it work?

Worked example

Bothe the cosine similarity and the overlap similarity suggested that the words "black" and "white" are similar. What happens when these are grouped together? We not only consider throwing both words into the same bag, but we then may want to consider what happens to other disjuncts, that connected to other words.

Also, when clustering, should we create a single common category "bw" holding both words, or should we create three categories: bw, white-prime and black-prime, where bw just has the common disjuncts, and white-prime and black-prime is what's left after taking differences?

English wordpair small dataset July 2017

This report provides a quick sketch of a small dataset containing English wordpairs. This is the EN_PAIRS_RONE dataset described in section . To recap, its this one:

Size	Pairs	Obs'ns	Obs/pr	Entropy	MI	Dataset
134K x 135K	5.54M	174M	31.4	17.67	1.94	en_pairs_rone_mst

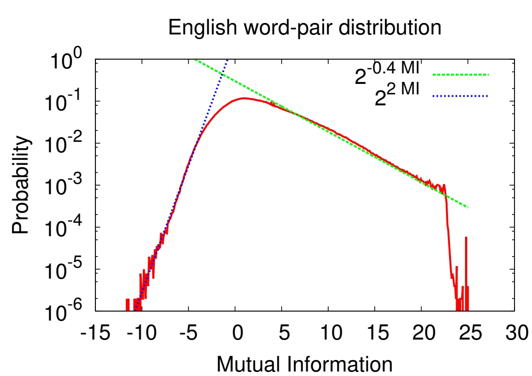
The size and support is recapped here below, just copied from section . The explanation of the column labels can be found there.

Size		Support		Count		Length		Dataset
L	R	L	R	L	R	L	R	Name
134K	135K	17.4K	17.4K	143	129	16.6	14.0	en_pairs_rone

The dataset contains 5544578 pairs.

Distribution of Mutual Information

The figure below shows the distribution of the mutual information of English wordpairs.



The peak of the distribution occurs near an MI=1.0. The two straight lines are eyeballed to fit the bimodal distribution. The "meaningful" mode, with positive MI, has a slope of -0.4. The negative-MI mode has a slope of 2.[16] Note that this is qualitatively similar to the Chinese hanzi pairs distribution, shown below, although the slopes are different, and the peak is slightly shifted.

Chinese character pair dataset July 2017

This report provides a quick sketch of a dataset containing Mandarin Chinese character pairs. This differs from English in two important ways. First, obviously, its not English. Second, there was no word segmentation done: each character (hanzi, ideogram) is treated as being distinct, and so all pairs are between hanzi. The goal/hope here is that word segmentation will appear "naturally", as a by-product of high-MI hanzi pairs. The dataset is the ZH_PAIRS_SONE dataset described in section . To recap, its this one:

Size	Pairs	Obs'ns	Obs/pr	Entropy	MI	Dataset
158K x 159K	5.92M	729M	123	18.45	2.02	zh_pairs_sone

The size and support is recapped here below, just copied from section . The explanation of the column labels can be found there.

Size		Support		Count		Length		Dataset
L	R	L	R	L	R	L	R	Name
158K	159K	6819	6411	548	487	41.7	37.7	zh_pairs_sone

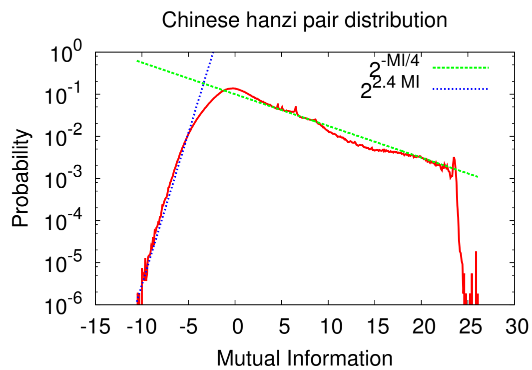
As mentioned before: the dimensions of the dataset are larger than the number of hanzi, because the dataset treats all Latin-alphabet words as single words. Since this dataset is generated from Wikipedia, we can expect that many of the entries correspond to English-language technical terms and named entities, such as product names, geographical place names and the names of people. There is also likely to be a mixture of simplified and traditional hanzi in the dataset.

The dataset contains exactly 5922477 pairs.

Distribution of Mutual Information

The figure below shows the distribution of the mutual information of the hanzi pairs.

[16] Graph obtained from the binned-enpr-mi data, in the en-pairs.scm file.



The peak of the distribution occurs near an MI=-0.25. The two straight lines are eyeballed to fit the bimodal distribution. The "meaningful" mode, with positive MI, has a slope of -0.25. The negative-MI mode has a slope of 2.4.[17] Note that this is qualitatively similar to the English word-pairs distribution, shown above, although the slopes are different, and the peak is slightly shifted.

Idioms and word boundary detection

Higher-level structures in language are important. By "higher level" I mean both the problem of detecting idioms in English, and segmenting words in Chinese. I claim that applying traditional algorithms to sheaves is sufficient to get good results.

In English, one is interested in discovering idioms, entity names, set phrases and institutional phrases set in English: one is looking for a sequence of neighboring "words" that commonly occur together. Examples include: "Sun Trust Bank" (an entity name), "gone fishin" "out to lunch" (set phrases), "blessing in disguise", "dime a dozen" (idioms). The words are not necessarily sequential: there are set circumpositions: "if... then..." "first... second...", "not only ... but also ..."

The Chinese word-segmentation problem is discerning when two hanzi characters belong to the same word, or not. It is similar to the problem of discerning idioms in English.

What is a word?

As background knowledge: there are multiple definitions of a word: Jerome Packard, in "The Morphology of Chinese A Linguistic and Cognitive Approach" (2000) Cambridge University Press lists the following:

- Orthographic word
- Sociological word
- Lexical word

[17] Graph obtained from the binned-hanpr-mi data, in the zh-pairs.scm file.

- Semantic word
- Phonological word
- Morphological word
- Syntactic word
- Psycholinguistic word

Sheaf structures are important

The proposal being advanced here is that the general sheaf-theoretic techniques can be used to discover all of these structures.

The simplest case would seem to be word-boundary detection in Chinese. Here, a word boundary might be one, two or three hanzi characters in a row. It seems that basic MST techniques should be enough to discover these. So, for example, given a hanzi sequence A B C D E, if the MST parse provides a link B-C and C-D but no link A-B and no link D-E, then the sequence BCD is a candidate for being identified as a word. However, observing this once is not statistics: only if the sequence BCD is observed many times, can one consider it to be a word.

More complex structures can be found using sheaf-theoretic techniques. The example below is taken from the existing Link-Grammar lexis to illustrate a search for circumpositions. Consider the sentence “I will do it if you say so”. It has the parse:

```
    +-------->WV-------->+---MVs--+----CV->+
    +--Wd--+-Sp*i+--I---+Osm+  +Cs+-Sp-+--O-+
    |      |     |      |   |  |  |    |    |
LEFT-WALL I.p will.v do.v it if you say.v so
```

Inside of this, there is a single “germ”, “gerbe” or “disjunct” located at the word “if”:

```
 +Cs+
 |  |
if  ?
```

Extending out from this are numerous “sections” or “partial linkages”. One of these is

```
+--MVs--+----CV->+
|      +Cs+-Sp-+--O--+
|      |  |    |     |
?      if ?   say    ?
```

The above structure might occur in many other sentences, and not just in this sentence. One can keep an eye out for this structure. If it occurs more often than usual, one can deduce that it is some set phrase or idiom. This particular example is not a set phrase in English, but it does illustrate how one can describe structure, and search for it, in a way that is more sophisticated than using n-grams.

Word boundaries - Chinese

So ... how does one find word-boundaries in Chinese? The basic idea is to count the frequency of patterns such as the below, where BCD are sequentially linked, and there are no links AB or DE. There may be additional links from the triple BCD going elsewhere, but not to neighboring words. Ideally, those links attach to just one morpheme.

```
..−−+            +−...   +−−...
    |   +−−+−−+          |
    |   |  |  |          |
    A   B  C  D          E
```

If this was a European language, we would expect any extra links to attach to the last morpheme; this is due to the morphology of Indo-European, where the semantic (meaning-carrying) stem is always to the left of the grammatically active suffixes. Note Japanese, although it has a minimal morphology, as also similarly structured; i.e. the suffix carries the syntactic structure. With Chinese, this is less obviously the case, and ideally, the correct attachment will be discovered.

Meaning

So here's one approach to meaning. It is already clear that disjuncts are correlated with meaning, so one provisional approach might be to assign each disjunct a unique meaning. Alternately, this can be used as a doorway to the intensional meaning of a word.

Consider the phrases "the big balloon", "the red balloon", "the small ballon"... The pseudo-disjuncts on balloon in these three cases would be "the- big-" "the - red-" and "the- small-" (plus an additional connector to the verb). Examining this connector-by-connector, we expect that the MI for the word pair (the, balloon) to be small, while the MI for the word-pairs (big, balloon), (red, balloon) and (small, balloon) to be large(r). Its thus tempting to identify the set {big, red, small} as the set of intensional attributes associated with "balloon". The strength of the MI values to each of the connectors might be taken as a judgement of how much that attribute is prototypical of the object (see other section on "prototype theory").

The disjuncts associated with "balloon" will also connect to a verb. These verb connectors may be taken as another set of intensional attributes, for example {floats, drifts, rose, popped}. It should be possible to distinguish these as an orthogonal set of attributes, in that one might observe "the- red- floats+" and "the- red- drifts+" but never observe "floats- drifts+".

Meaning bibliography:

- "The Molecular Level of Lexical Semantics", EA Nida, (1997) International Journal of Lexicography, 10(4): 265–274. https://www.academia.edu/36534355/The_Molecular_Level_of_Lexica

Meaning Redux

(9 June 2018) I keep explaing, over and over, why K-means and SVD cannot be used. Here's a snapshot from a recent email, explaining it again:

Why SVM and K-means don't work

Here's WHY both SVM and K-means are fundamentally wrong, and are total failures for this particular task. Lets start with K-means. One minor issue with K-means is that you have to pick K in advance, but you don't know what K is. But whatever, that's not important. Its OK to guess that K=100 or so. The big problem is that K-means then takes the *MEAN* (the *AVERAGE*) of the vectors. That's what the word "mean" in "K-means" means. But we already know, a priori, that taking the average is wrong -- it wipes out, erases the different word senses.

For example: the word-token "saw" is going to have a vector that contains disjuncts for both "cutting tool", "the verb cut" "the past tense of to see". With K-means, this word token can only be assigned to just one cluster: it will be the cluster for nouns, or the cluster for past tenses, or the cluster for cutting-manipulation-actions. No matter which cluster its assigned to, when the average/sum of the vector is merged into the cluster, the wrong disjuncts will be averaged in as well.

So, for example: lets assume k-means places "saw" into the "nouns cluster". After averaging, the noun cluster will now contain disjuncts for both past-tense verbs, and also disjuncts for present-tense manipulation-verbs. Clearly, noun-clusters should not contain these. Two bad things happen: (a) the noun cluster is polluted with verb-vector components, and (b) the vector has not been factorized, and so "saw" cannot also be placed into other clusters as well.

Ergo -- K-means is fundamentally incorrect -- it cannot correctly cluster linguistic data!

Lets write some formulas: let v be a vector. The MST observation counts give us v_{saw}. We know that, a priori,

$$v_{saw} = v_{tool} + v_{past-tense} + v_{cutting}$$

However, we do NOT know what these parts: v_{tool},$v_{past-tense}$, $v_{cutting}$ what they are. We need to factorize them out. K-means erases them, lumps them all into one. It does not factorize.

SVM is a little bit better, if you use it correctly. I am not convinced that you are using it correctly. So, for example, lets say we had only four words: v_{saw}, v_{look}, v_{heard} and v_{tool}. Suppose that SVM was told to decompose into three dimensions, and that the three that were picked were mostly pointing along the direction of v_{look} and v_{heard} and v_{tool} -- these were the three principle components.

Where should v_{saw} go? In traditional SVDM, the three principle components are used to define three hyperplanes or classifiers, and so the single vector v_{saw} would then be classified as to being either "on the same side of the hyperplane as v_{look}, and thus a part of the v_{look} singular value", or "on the same side of the hyperplane as v_{tool}, and thus a part of the v_{tool} singular value", etc.

But this is again wrong. We want to factor or decompose v_{saw} along these three different principle components, thereby automatically discovering that some of the disjuncts on v_{saw} are tool-like, that others are verb-like, etc.

THIS is where word-sense disambiguation comes from. It is *NOT* done in some pre-cleaner, pre-disambiguator stage. It is done at the clustering stage.

But, as I hope is now clear, both SVM and K-means are fundamentally wrong approaches, because both ERASE word-sense information from the dataset!

Now, IF you are very careful, you might be able to modify SVM, and after finding principle components, go back for a second pass, and perform the factorization needed to extract the different word-senses. Maybe. I can see/guess at a way of doing this, but its hard.

Dimensional reduction

There's a completely different issue - "dimensional reduction" which must not be ignored; its important, a big part of the task.

So: My large dataset has 24 million disjuncts in it -- that's the dimension of the vector space -- all vectors are 24M-dimensional. How do you perform dimensional reduction? Well, its "easy" -- if two disjuncts have connectors that belong to the same word-class, replace them by one disjunct in that word-class. (The vector space is now (24M minus one)-dimensional) Lets suppose that one of the disjuncts is

```
bird:  the- & saw-
```

When doing the dimensional reduction, the saw- needs to be replaced by: ??? either `TOOL-` or `PASTTENSE-` or by `CUTTING-`.Obviously, that disjunct was obtained from an MST parses of childrens-lit sentences like "John saw the bird. Susan saw the bird too. Mary saw it also". When you dimensionally reduce the saw- in this disjunct, which cluster do you assign it to?

Well, if the text has the sentences: "John knew the bird was there. John heard the bird", and if clustering determined that "knew", "heard" belongs to `PASTTENSE` then the dim reduction is clear:

```
bird:  the- & PASTTENSE-
```

and we know that the following is wrong:

```
bird-:  the- & TOOL-
```

The problem here is that both K-means and SVD are completely ignorant of the structure of the basis elements of the vector space. Both assume that the basis elements of the vector space are irreducible, atomic, indivisible. Its a natural assumption for some machine-learning tasks, but completely wrong for language-learning where we know, a priori, that the basis elements have structure. This is kind of a key idea from sheaves!

Merge Results 5 June 2018

After a very long hiatus, restart. All earlier merge data lost!?

Here's a sample of automatically-discovered grammatical classes, using the '`ortho-merge`' strategy from '`gram-class.scm`'. I seem to have lost/corrupted a previous, larger dataset, so this was remade from scratch the last few days. Source dataset is '`en_pairs_cfive_class`'. The merge paramters are: cosine-similarity-accept cutoff = 0.65; union-merge-fraction = 0.3.

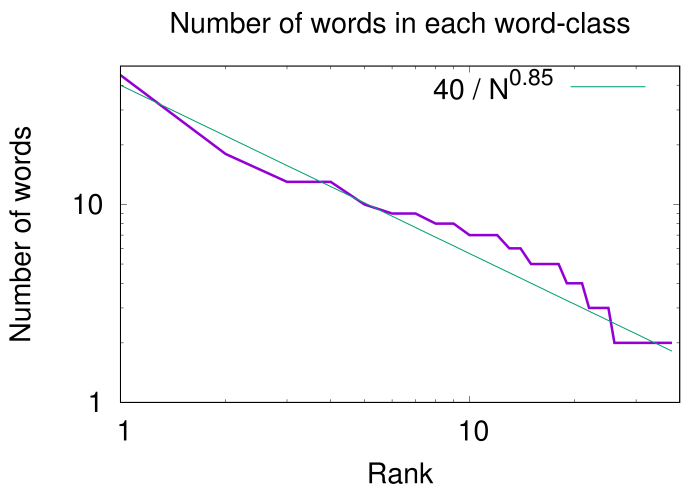
This run took 48 hours (its very far from done, this is a snapshot), it found 230 words that it could classify into 38 classes. (Seven more words got classified as I prepared this, so the counts may be off.) The table below lists them exhaustively. Note that some words appear in multiple classes: for example, "mother father". Some words are clearly mis-classified, but there are not many of those. Some classes are a bit confusing as to thier content, but most seem very clear. The classes are clearly semantic in nature; for example, there are two distinct classes of prepositions. The semantics is entertainingly insightful: "voice mother hands heart head father mind face feet" are parts of oneself, with some unexpected members: "mother, father" are not normally considered to be body parts, but are, in some sense, deeply, "parts of oneself". Similarly, "wife arm daughter friend mouth friends brother" are mostly relatives and relationships, yet "arm mouth" are not. Perhaps the arm and mouth have a mind of thier own, functioning a bit independently from the true self?

I'll try to run this a few more days, and present a newer report. While reviving this old code, I realized that the classification algorithm being used here has multiple faults and is a bit crude. I'm writing a nicer algo right now. I don't really know how to compare the quality of the algos, at this point.

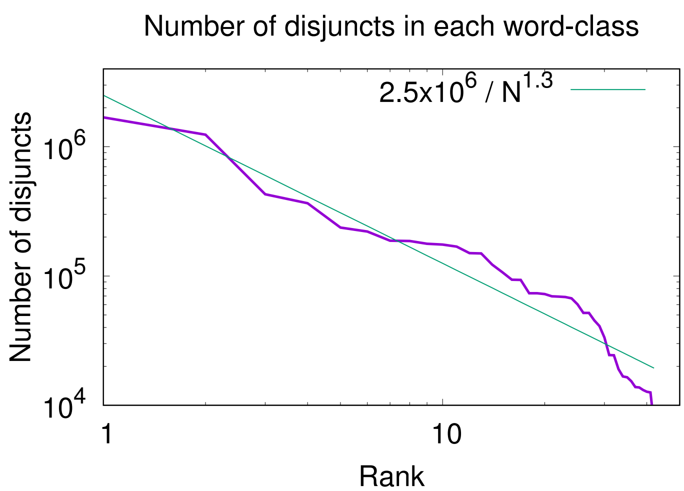
Meanwhile, you shuld be able to get similar results, by applying the code in '`gram-class.scm`' to a dataset that contains disjuncts dervied from MST parses.

Size	Members	Comments
43	village city question subject sea town girl public land French English fire King war boy air morning words others poor best second world door book heart body case night room whole light country people house children last present ground water family first other	Nouns, mostly
18	for in from at on by of with all towards within near against under through over upon into	Prepositions
12	help hear keep leave take find get make see give say go	Personal verbs
12	fine word large moment certain small woman new good man great little	Adjectives, mostly
10	fall action history character state position sense force knowledge pleasure	
9	full nature part death power most some out one	
9	voice mother hands heart head father mind face feet	Body-parts
8	till whether since because until where if when	Time
7	will would might should may can must could	Imperatives
7	or but perhaps nor though And while	Conjunctions
7	wife arm daughter friend mouth friends brother	Relatives
6	feel believe myself am know think	Beingness
6	rest end body name side power	
6	heard taken given already done seen	Past perfect - action
5	really always still also now	
5	year place same day way	
5	kept held called made found	Possesive verbs
4	son arms own eyes	
4	her me him us	Anaphora
4	heard felt knew saw	Simple past - action
3	making such like	
3	our its their	Possesives - plural
3	five three four	
3	during between among	Prepositions
2	once least	
2	thus sometimes	Deduction
2	therefore indeed	Deduction
2	is was	to be - Singular
2	are were	to be - Plural
2	! ?	Sentence end
2	, ;	Punct
2	And The	Sentence start
2	they we	Anaphora
2	cannot shall	Imperatives
2	mother father	
2	sort number	
2	France England	

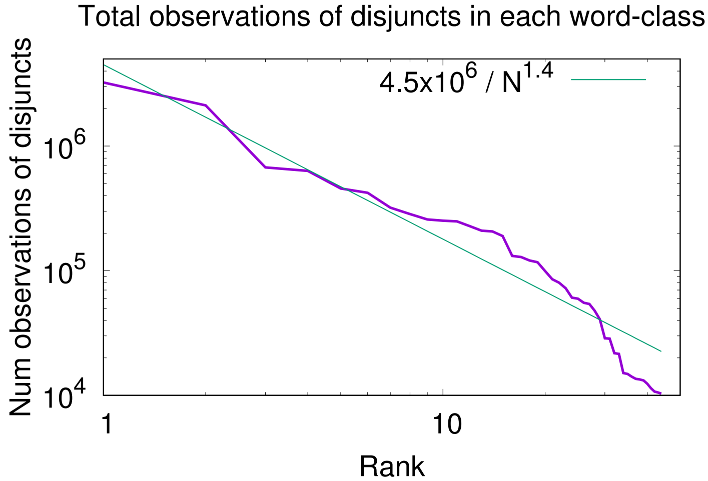
Here's a graph of the above distribution. Its on a log-log scale. It looks to be approximately Zipfian. That's no surprise. Total of 38 classes shown.



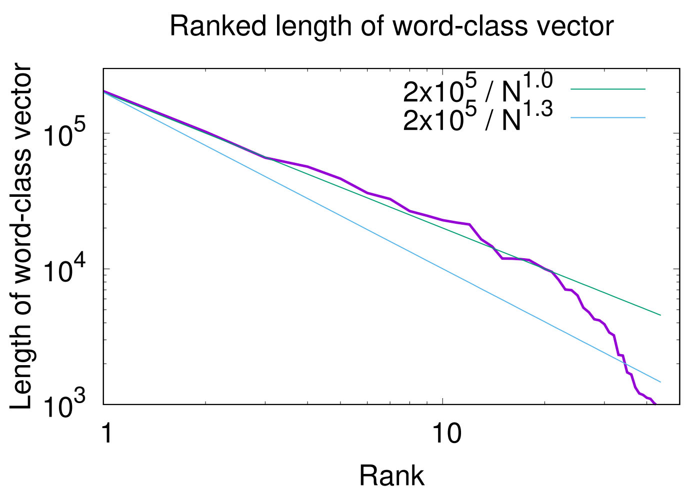
Here's a graph of the number disjuncts in each grammatical class. (There were 259 words classified into 42 classes when this was prepared). The "number of disjuncts" is the same as the "support" or l_0 norm of the vector.



Continuing, below is the total count of the number of observations of the disjuncts (There were 269 words classified into 44 classes when this was prepared). The "count of disjuncts" is the same as the "count" or "Manhatten distance" or l_1 norm of the vector.



And again, below is the length of each vector, viz, the root-mean-square count of the number of observations of the disjuncts (There were 269 words classified into 44 classes when this was prepared). The "RMS count of disjuncts" is the same as the "length" or l_2 norm of the vector. The initial part of this graph is the most Zipfian so far, with a slope of exactly 1.0, as eyeballed.



Conclusion: Looks good, more proccessing needed; comparison of experiments needed.

Merge Experiments

Three different merge experiments are being run. These are reported below. The summary is here:

Block-5x5 Same as above, pushed out farther. This is run on a copy of the `en_pairs_cfive_mst` dataset (all five MST tranches) on the LXC container. Using a cutoff of 20 observations minimum per word, this contains 62607 words to be classified. The classifier is the `merge-ortho` classifier, using a minimum cosine of 0.65 to propose a merge, and a fixed fraction of 0.3 for the union-merge. The agglomeration algorithm is the block-diagonal algorithm. (This dataset has 40M sections; viz about 80M atoms, viz about 120GB to load in full)

Fuzz-5x2 This is run on a copy of the `en_pairs_cfive_mtwo` dataset (only two MST tranches) and thus has fewer words: a total of 25505 words with more than 20 observations. As above, this uses the `merge-ortho` classifier, using a minimum cosine of 0.65 to propose a merge, and a fixed fraction of 0.3 for the union-merge. The agglomeration algorithm is the greedy algorithm. (this dataset has about 13M sections) Run this as: '(gram-classify-greedy-fuzz 0.65 0.3 20)'.

Discrim-5x2 This is run on a copy of the `en_pairs_cfive_mtwo` dataset, as above: a total of 25505 words with more than 20 observations. This uses the `merge-discrim` classifier, which is like the `merge-ortho` classifier, but uses a variable fraction for union-merge. Because the variable fraction should behave nicely, the minimum cosine is set to 0.50. The agglomeration algorithm is the greedy algorithm. Run this as: '(gram-classify-greedy-discrim 0.5 20)'.

Basically, the last two are directly comparable: they differ only in the merge strategy. The first two are harder to compare: they use different datasets and different agglomeration algos. All three are using the screwy `merge-ortho` classifier, which is almost right, but altered counts in a somewhat screwy way. Thus, these experiments need to be repeated ... again.

The table below is a "progress report" on **Fuzz-5x2**, as its being computed. Each row represents a snapshot in a different point in time for the computation. **Fuzz-5x2** crashed; the last row are the stats at the time of the crash.

num-classes	num-words	doubles	singletons	dupes	dup-cls	uniq-dup	cpu
46	429	18	259	12	24	15	3400
56	482	21	487	14	28	17	6230
65	533	26	682	15	30	19	8800
70	581	28	915	18	36	21	11760
75	635	28	1138	24	48	26	14760
81	670	31	1301	25	50	28	17200
85	707	33	1544	25	50	28	20720
90	822	33	1846	29	58	31	25256
93	835	33	1927	33	66	34	26624

The columns are as follows:

num-classes Total number of grammatical categories (word classes).

num-words Total number of words assigned to grammatical classes, with two or more words per class.

doubles Total number of classes having exactly two words in them.

singletons Total number of words examined, but could not be assigned to any existing grammatical class. These can be thought of as classes that have only one member; they may eventually grow to more than one member.

dupes Total number of words belonging to more than one class. These roughly correspond to words that have been found to have more than one syntactic form (i.e. more than one "meaning")

dup-cls Total number of classes that the dupes belong to. Thus, dupe-cls / dupes = average number of "meanings" that a multi-meaning word has.

uniq-dup Total number of unique classes that have multi-meaning words in them.

cpu The CPU-minutes accumulated so far. This is ad-hoc, it doesn't count for time spent in postgres, or inefficient parallelism. It just provides a scale for forward progress.

Some examples of multi-category words:

what belongs to <that as when if what before where because until> and also <what how why whether> – propostional words and question words.

her belongs to <his her> and <her him me us> – possesives and determiners.

with belongs to <of in with for on by from into upon over through under among> and also <with like such having> – prepositions and membership-property words.

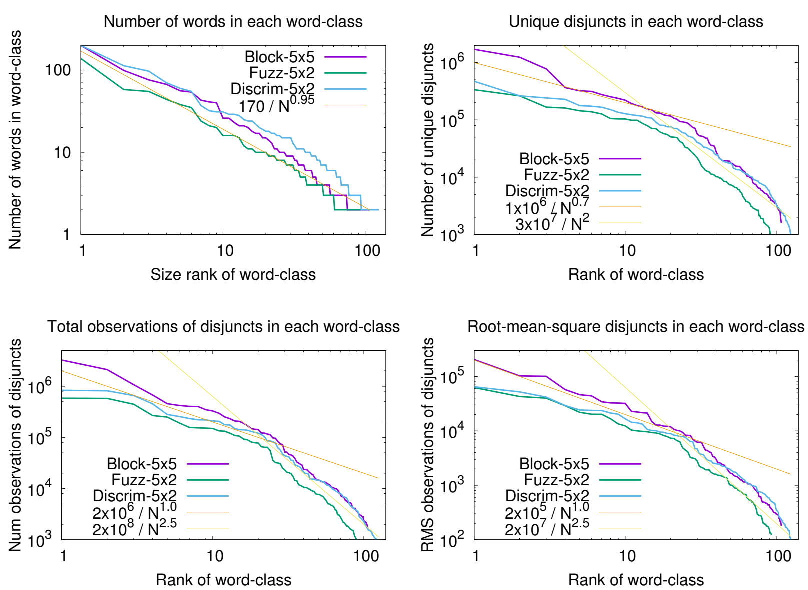
Here's a progress report for **Discrim-5x2**. A quick look-see shows that this is lower-quality; the cosine=0.50 seems to accept too much, mixing nouns and verbs, although it is better at placing given names into one category (for example)...

num-classes	num-words	doubles	singletons	dupes	dup-cls	uniq-dup	cpu
31	427	8	67	17	34	17	2370
47	715	14	89	62	127	23	4130
63	946	16	169	87	182	30	6375
82	1145	19	343	106	222	41	9700
96	1268	24	573	125	260	44	13420
106	1415	27	757	136	282	49	16400
121	1518	32	994	142	294	54	20600
125	1545	33	1020	145	300	56	21149

Its clear that **Discrim-5x2** is assigning more words into fewer classes than **Fuzz-5x2** is. Particularly notable is the much much smaller size of the 'doubles' classes and the 'singletons' classes. This crashed mysteriosuly after running for 21K seconds... ‘(compute-right-cosine (WordNode "”" #)

Below are distribution graphs for the word-clases obtained using the two differrent datasets, and three different classification schemes. The general similarity of the graphs is immedately apparent. One can conclude:

- The different classifcation schemes all generate the same distribution of class sizes, and that distribution is very nearly purely Zipfian. (Upper-left graph)
- The distribution of disjuncts in the classes is bimodal, and the modality and inflection is the same for the l_0, l_1 and l_2-norms.
- The distribution of disjuncts is determined primarily by the dataset, and not by the classification algo. That is, Fuzz-5x2 and Discrim-5x2 are two different classifiers running on the same dataset; they are in many ways similar, and differing a bit from the larger dataset Block-5x5.



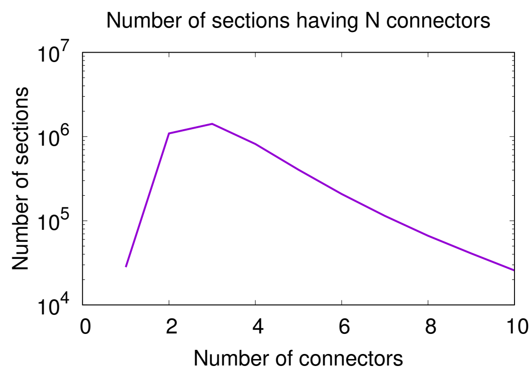
The overall lack of dramatic differences in the distributions is remarkable. Visual inspection of the classes indicates that they are all arriving at the same general and mostly-correct classification of words. It could be interesting to see how much these classifications differ; measuring this, however, is difficult, as they are in distinct datasets, and there's no infrastructure for that.

Quality evaluation

Evaluating the quality is hard. Quick looks suggest its all going as planned... unclear how to be quantititive, except by tedious hand-scoring.

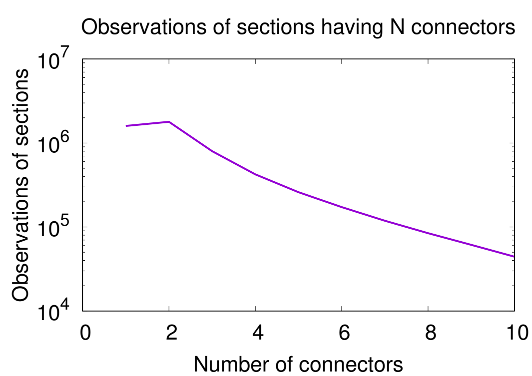
Connector distribution from MST parses

(9 June 2018) This is kind-of a repeat of earlier work reported in ‘connector-sets-revised.lyx‘ but is (a) graphed differently and is (b) for a different dataset. Its actually a commentary on the quality of data comeing out of MST. First graph: number of sections having N connectors.



It shows how many sections there are that have the indicated number of connectors on them. That is, each section has one and only one disjunct in it. Each disjunct can have N connectors in it. So, fixing N, how many sections are there that have N connectors? For “real” linnguistic data, we excpet a much much sharper falloff. Determiners (the, a this, that ...) should have one connector. Nouns should have 2 or 3 or 4: zero or one to a determiner, zero, one or two (maybe rarely three) to adjectives, one to a verb. Transitive verbs should have 3 or 4 connectors: one to the subject, one to the object, one to LEFT-WALL, zero or one to adverbs, particles, prepositions, etc. Thus, six or more connectors should be very very rare. its not. This suggests that the MST parser is not producing enitrely beleivable data.

Perhaps the high-connector count disjuncts are observed only infrequently? The next graph shows the counts, weighted by the number of observations: i.e. how often that particular disjunct was observed.



Hmm. The good news: the observation counts for 1- and 2-connector disjuncts are much higher, with 3-connector disjuncts seen a lot less often. The number of 4-connector disjuncts remains dishearteningly high, and the fall-off is slower than before.

Both of the above graphs were generated by considering only the sections on word-clusters. This comes from the **Block-5x5**, when it had 700 words assigned to clusters. Its not clear if this pattern is true, in general, for all words. Note that, to obtain these 700 words, the top-most-frequent 1300 words were examined: so the 700 words are among the most frequent. Note that, due to the orthogonalization algorithm, not all of the counts are transfered from the words to the clusters; only some are.

Entropy-similarity

Some stats for entropy-similarity. Consider '**en_rfive_mtwo**'. There are:

- Rows: 137078 – viz that many words. Viz $\sum_w 1 = 137078$
- Columns: 6239997 - viz that many disjuncts – i.e. (Section * dj) with * the wild-card, and dj held fixed. Viz $\sum_d 1 = 6239997$
- Size: 8629163 - viz this many sections, viz explicit (Section w dj) for fixed w,dj) Viz $\sum_{w,d} [0 < N(w,d)] = 8629163$

Let $N(w,d)$ be the count of disjunct d on word w. Then, for **en_rfive_mtwo** we have:

$$N(*,*) = \sum_{w,d} N(w,d) = 18489594.0$$

viz 18.5M observations total, while

$$\sum_{u,w,d} N(u,d)N(w,d) = \sum_d N(*,d)N(*,d) = 63598403588.0$$

viz 63.6G. Note that

$$\sum_d \frac{N(*,d)}{N(*,*)} \frac{N(*,d)}{N(*,*)} = 1.8603 \times 10^{-4}$$

Define the product between words as

$$f(u,w) = \sum_d N(u,d)N(w,d)$$

Normalize this as

$$p(u,w) = f(u,w)/f(*,*)$$

and define

$$MI(u,w) = \log_2 \frac{p(u,w)}{p(u)\,p(w)}$$

where

$$p(u) = p(u,*) = \sum_{w,d} N(u,d)N(w,d) = \sum_d N(u,d)N(*,d)$$

What's this like? Some pairs below. Note that $-\log_2 1.8603 \times 10^{-4} = 12.3921$

u	w	$-\log_2 p(u)$	$-\log_2 p(w)$	$-\log_2 p(u,w)$	$MI(u,w)$	$\cos(u,w)$
other	same	21.52	22.13	27.13	4.1232	0.5866
nice	fine	26.94	25.21	35.24	4.5210	0.5256
him	me	20.81	21.34	25.50	4.2583	0.7842
men	women	23.69	25.79	33.24	3.8489	0.6077
up	down	22.46	23.31	27.87	5.5085	0.5630
found	called	23.62	24.57	32.09	3.7066	0.5576
came	went	24.19	24.72	31.32	5.1930	0.5900
eyes	hand	23.47	23.19	28.91	5.3589	0.7284
men	nice			38.71	-0.477	0.0233
men	went			36.08	-0.061	0.0249
nice	went			38.22	+1.0490	0.0353
called	eyes			38.14	-2.499	0.0049
nice	eyes			40.21	-2.193	0.0038
nice	called			35.63	+3.4810	0.3794

A table yea.

TODO

Explain how mutual exclusion of concepts as performed by humans when learning new concepts, resembles optimal strategies for the channel coding theorem, by minimizing confusion between similar concepts. This is the "mutual exclusion" principle. Well, MI already provides a certain measure of exclusivity.

The End

References

[And12] Steven R. Anderson. Dimensions of morphological complexity. In Greville G. Corbett Matthew Baerman, Dunstan Brown, editor, *Understanding and measuring morphological complexity*, 2012.

[GV14] Ben Goertzel and Linas Vepstas. Langauge learning. ArXiv abs/1401.3372, 2014.

[iC06] R. Ferrer i Cancho. Why do syntactic links not cross? *EPL (Europhysics Letters)*, 76(6):1228–1234, 2006.

[Lin98] Dekang Lin. Automatic retrieval and clustering of similar words. In *Proceedings of the 17th international conference on Computational linguistics*, pages 768–774, Morristown, NJ, USA, 1998. Association for Computational Linguistics.

[Liu08] Haitao Liu. Dependency distance as a metric of language comprehension difficulty. *Journal of Cognitive Science*, 9(2):159–191, 2008.

[LP01] Dekang Lin and Patrick Pantel. Dirt: Discovery of inference rules from text. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'01)*, pages 323–328. ACM Press, 2001.

[PD09] Hoifung Poon and Pedro Domingos. Unsupervised semantic parsing. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 1–10, Singapore, August 2009. Association for Computational Linguistics.

[ST91] Daniel Sleator and Davy Temperley. Parsing english with a link grammar. Technical report, Carnegie Mellon University Computer Science technical report CMU-CS-91-196, 1991.

[ST93] Daniel D. Sleator and Davy Temperley. Parsing english with a link grammar. In *Proc. Third International Workshop on Parsing Technologies*, pages 277–292, 1993.

[Yur98] Deniz Yuret. *Discovery of Linguistic Relations Using Lexical Attraction*. PhD thesis, MIT, 1998.