Language Learning Diary - Part Two

Linas Vepstas

February-August 2021

Abstract

The language-learning effort involves research and software development to implement the ideas concerning unsupervised learning of grammar, syntax and semantics from corpora. This document contains supplementary notes and a loosely-organized semi-chronological diary of results. The notes here might not always makes sense; they are a short-hand for my own benefit, rather than aimed at you, dear reader!

Introduction

Part two of the diary on the language-learning effort has three major sections. These are summarized here. This part has been closed, and part three has been started (there will be no further updates here.)

Experiments with random grammars. Feb-May 2021.

Attempt to run the full learning calibration pipeline on a randomly-generated grammar. This pipeline creates a random grammar, creates a random corpus with this grammar, runs the learning pipeline, learns a grammar, and then compares the learned grammar to the original one.

The goal of doing this is to be able to perform "calibration": by comparing the input and output grammars for fidelity, we could explore the parameter space, and find the best tuning for the learning pipeline. This sounds good in principle, it failed in practice.

Doing this entailed numerous bug fixes and some new code to get everything working (so that's a good thing). A bit of interesting mathematics popped up: merging is a non-commutative operation. When merging words into a word class, one must also merge connectors into connector classes. The initial idea was that merges can be done as (quasi-)linear operations: Attached to each word is a vector, and creating a word-class is as simple as summing vectors. Or possibly intersecting the basis elements, and then summing only those parts where the basis overlap (thus, the quasi-linear merge. This is needed, because non-overlapping vectors correspond to different word senses, and we need for factor those out.

One might imagine that merging connectors is as simple as creating vectors for those, the so-called "shape" vectors, and summing those. Because of how shapes are created, one might think that one could merge words, then shapes, or first shapes and only then words, and get the same thing. Not so: the merge operation is non-commutative. The result depends on the order it's done in. This is interesting, but troublesome.

The goal of merging is to find two things that are similar, and then place them in a common class. Since the "things" are word-disjunct pairs, one wishes to find some unambiguous, unique way of merging them. This seems not to be the case: there is some kind of "torsion" or "curvature" in the system. It's not clear how to even characterize this concept clearly. An unambiguous merge seems to be asking for some kind of flattening or abelianization of the merge procedure. It's not clear how to do this.

Mathematically, the formalities are still opaque. The notation for word-disjunct pairs is awkward. The notation for the linearizations are awkward. When I think about linearization, it feels very geometric: as if one is working with some tangent space to some manifold. But there's no manifold, and the analogies quickly falter. Progress requires a much clearer, a much more concrete description of what's going on with this network.

The experiments ended with confusion. Besides the above non-commutative aspect to merging, it became clear that many different initial grammars are effectively equivalent, in that they generate similar, if not identical corpora. Thus, the naive idea of calibration fails: one can't just learn the input grammar; at best one can learn the class of that grammar. But, as of now, we don't have any clear way of defining a "class of equivalent grammars", or more broadly, "a class of very similar grammars".

A secondary problem was that the random grammar didn't resemble a natural language grammar. That was not the intent. It was hoped that by picking a Zipfian distribution of vocabulary words, a Zipf distribution of disjuncts, throwing in some synonyms into the mix, the result would resemble a natural language. This was not the case. Despite having done a lot of distributional studies of English, it seems that I still don't know how to characterize a natural language very well. I don't know how to measure the complexity of a grammar, or the complexity of a corpus. There are formal questions: how can one tell when two grammars are isomorphic? Almost isomorphic? There are practical questions: what is the best way to factor the word-disjunct matrix?

In the end, it all got muddy, with now easy, obvious way of moving forward.

Back to English. July 2021

The advantage of working with English is that one can develop a gut-feel as to whether one is making progress. It's hard to look at a random grammar, and decide "does that make sense?" So, back to English. Earlier English experiments went so swimmingly well, that surely moving forward should be easy!

An embarrassing problem was immediately encountered: Setting the merge threshold at some apparently innocuous value, that would result in similar words getting merged immediately resulted in chaos and failure. This is very unlike the earlier experiments, where things went quite very well. Something went wrong. What is it?

Well, several things.

- Merging requires a merge threshold. Earlier experiments searched for this threshold carefully, and then applied it. Here, we complacently picked one, based on past experience and gut-sense, and it was set too low.
- Earlier merges used an overlap strategy, of assuming that similar words might have different disjunct sets just because some words were just too thinly sampled, to have seen such disjuncts. This seems like a good idea, but is perhaps naive. If one word-disjunct pair has been observed 100 times, and given another, similar word, that word-disjunct pair has been observed zero times, well, that might not be due to under-sampling. 100 is really different than zero, and assigning a fraction of the 100 to the merge cluster is perhaps a bad idea. There is a semi-formal proof that merging in this way will *always* reduce the MI. So if the goal is entropy maximization, the overlap merge strategy is a bad idea.
- Some confusion regarding the above ensues. The MI between words is explored, and compared to the MI between merged classes. After a lot of work, it really does become clear that overlap merging always makes grammatical class MI's worse, and that a pure intersection merge always maximizes MI.
- It was expected that using Shapes would improve similarity scores. After some work, it seems that this is not the case - they mostly seem to make things worse. It's not at all clear which this is so. Shapes should have helped, but they don't!
- To keep calculations relatively fast, trimmed datasets were used. It now seems that trimming damages the data. This is still a problem. Trimming seems like it should help learning, but again, it seems like it doesn't. The causes of this are still not intuitively clear.

The biggest thing that went wrong was that, when using MI as a measure of similarity, some rather dis-similar words had a high MI. How can that be? Turns out that if one compares a very large vector (a word with lots of disjuncts, i.e. a word with a large support) to another word with a very short vector (a word with only one disjunct, for example), the result can be a very high MI. Yet this is obviously not what we want: short vectors should not be similar to long vectors. This prompts an exploration of other kinds of similarity measures.

Similarity Smackdown, July-August 2021

The poor merge results prompted a step back, to judge different similarity measures (metrics). Twelve *ad hoc* "part of speech" clusters were defined. They were *ad hoc* in the sense of being hand-picked out of the first few thousand words with the highest observation counts. The various similarity measures were used to computer the inter-cluster and intra-cluster similarity. The quality of the metrics was assessed based on their ability to discriminate between clusters, by taking the ratio of the RMS sizes of the clusters to the RMS cluster separations. Higher discrimination is judged as being better.

Nine different similarity measures were evaluated:

- Mutual information
- Joint probability
- Variation of information
- Normalized variation of information
- Cosine distance
- Log cosine distance
- Vector overlap (plain Jaccard)
- Conditional Jaccard distance
- Probability Jaccard distance.

These were compared one datasets with and without shapes, at seven different levels of trimming (so, 14 datasets total). The results were as follows:

- Trimming is consistently deleterious to results. However, using the full dataset blows up both RAM and compute time. The best compromise, for the given datasets, appear to be to discard all words that have a small number of observation counts, and to discard sections with an observation count of less than two.
- Shapes mostly fail to improve similarity measurements, for any of the similarity measures. Using shapes makes all vectors more similar (smaller RMS intra-cluster, but also smaller RMS inter-cluster). The overall effect of shapes is mixed: they seem to improve things only for highly-trimmed datasets. Given that shapes add a large computational burden, they seem to offer no benefit. (They sharply increase the amount of time needed to compute similarity, and can triple the RAM usage.)
- The best overall similarity measure was vector overlap. Second place overall goes to Conditional Jaccard (which uses probabilities in the Jaccard min/max formula) Third place goes to Probability Jaccard. MI is consistently in fifth place.

A few comments. Although the motivation for these experiments was the large MI between short and long vectors, the experiments actually eliminated short vectors. That is, the hand-picked clusters consisted of words with relatively high counts, and thus relatively long vectors.

The failure of Probability Jaccard to dominate is also surprising. The Wikipedia article explains that it is in some sense optimal: it provides "maximally consistent sampling". This appears not to help us. It is also very computationally intensive; its the slowest similarity measure of all.

February 2021 - Closed Loop Calibration

Part two of the diary on the language-learning effort starts with the new task of of closed loop learning. The idea of closed loop learning is that the accuracy of the learned grammars can be very closely monitored and measured, thus allowing the learning algorithms to be tuned for speed and measured for accuracy. The closed loop is a basic five-step process:

1. Generating random but controlled grammars
2. Generate a text corpus from these grammars
3. Learn a new grammar from the corpus
4. Compare the controlled-grammar to the learned-grammar
5. Tune algorithms and procedures, and repeat.

The step-by-step instructions can be found in the file “README-Calibration.md”.

February 2021

Restart the project, finally making some headway. ToDo items:

- Describe how the “uniform random sampling of sentences” is performed.
- Enable weighted random sampling of sentences.
- Fix multi-sense bug in gen-dict.scm circa line 85

First run. Instructions in “README-Calibration.md”. Lots and lots of bugs fixed and lots of pipeline was automated.

March 2021

First real experiment, in expt-7/expt-8/expt-9. The dict in expt-7 fails to generate the correct corpus, because the generator does not expand synonyms (this is a combinatorial explosion and it just doesn’t do that.) Expanded by hand in expt-8. Same data, new scripts and config in expt-9.

Issues:

- After MST parsing, the grammar is correct, in that (I think) it will produce exactly the same corpus. However, the rules are different (and more verbose). TODO: check that the same corpus is actually produced. How? Answer: generate the corpus, and compare...
- After MST (MPG) parsing, the verbs link to obj-determiner instead of object. Why? Was there a cutoff that was missed? The corpus is just .. tiny.

- After (gram-classify-greedy-discrim 0.5 4) the right clusters are produced, but the connectors are not clustered; they need to be. TBD.
- Result: the correct corpus is produced (via manual checking), however, extra sentences are produced, which are missing the verb. This is because the wall links to the two determiners, and the wall can be skipped. However, the sentence with the verb has a higher MI. (11.51 instead of 8.92. See below.) This is due to a bug. See below.

Here's the result:

```
linkparser> the squirrel a dog
Found 1 linkage (1 had no P.P. violations)
        Unique linkage, cost vector = (UNUSED=0 DIS=-8.92 LEN=2)

     +-----------TB-----------+
     +---TB---+---TE---+      +-TE-+
     |        |        |      |    |
LEFT-WALL.2 the.1 squirrel.3 a.1 dog.3

linkparser> the squirrel saw a dog
Found 1 linkage (1 had no P.P. violations)
        Unique linkage, cost vector = (UNUSED=0 DIS=-11.51 LEN=4)

     +--------------TB--------------+
     |                 +-----TC-----+
     +---TB---+---TE---+---TF--+-TD-+-TE-+
     |        |        |       |    |    |
LEFT-WALL.2 the.1 squirrel.3 saw.4 a.1 dog.3
```

Hypothesis: The MI of wall-verb is lower than the MI of wall-determiner. Thus, the planar parser always picks the wall-determiner. Lets find out.

Heh. There is no MI wall-verb! Ouch. This was due to bad sampling; fixed github.com/opencog/opencog commit 895226228 Mar 16 2021. Sheesh.

Expt-10

expt-10, this is fixed. The parse trees are now very rich. Generated sentences:

- length 4 - none
- length 5 - the expected ones.
- 6 - LEFT-WALL LEFT-WALL plus valid sentence
- 7 - none
- 8 - LEFT-WALL the mouse saw the dog chased a bird

- 9 - double left wall
- 10 - the LEFT-WALL a cat saw the dog chased a squirrel – and also a triple-left-wall.

WTF. what's with the crazy multi-left-wall!? Heh. Here we go:

```
     +---------------------TI-------------------+
     +------------------TB-----------------+    |
     |           +------TI-----+           |    |
     |           +---TB--+--TC-+--TO--+-TE-+-TC-+
     |           |       |     |      |    |    |
LEFT-WALL.2 LEFT-WALL.2 a.1 mouse.5 saw.3 a.1 cat.5
```

Expt-11

So... expt-11 places a period at the end of every sentence. That terminates the infinite-recursive lengths being generated to only finite-length sentences. There is a total of three different parses. All have exactly the same cost. These are as follows:

```
Found 3 linkages (3 had no P.P. violations)
Linkage 1, cost vector = (UNUSED=0 DIS=-15.67 LEN=10)

     +-------------------TH-------------------+
     |              +------------TJ-----------+
     +------TG------+              +----TE----+
     +---TF---+--TD-+---TI--+--TC--+--TD-+-TJ-+
     |        |     |       |      |     |    |
LEFT-WALL.2 the.1 dog.3 chased.4 the.1 cat.3 ..5

        Linkage 2, cost vector = (UNUSED=0 DIS=-15.67 LEN=11)
     +-------------------TH-------------------+
     |              +------------TJ-----------+
     +------TG------+------TB------+----TE----+
     +---TF---+--TD-+---TI--+--TC--+--TD-+-TJ-+
     |        |     |       |      |     |    |
LEFT-WALL.2 the.1 dog.3 chased.4 the.1 cat.3 ..5

          Linkage 3, cost vector = (UNUSED=0 DIS=-15.67 LEN=12)

     +-------------------TH-------------------+
     |              +------------TJ-----------+
     |              |       +--------TK-------+
     +------TG------+       |      +----TE----+
     +---TF---+--TD-+---TI--+--TC--+--TD-+-TJ-+
     |        |     |       |      |     |    |
LEFT-WALL.2 the.1 dog.3 chased.4 the.1 cat.3 ..5
```

Notable in the above:

- Only one parse, the third one, links the main verb to a wall. And then its the right wall, not the left wall.
- The determiners seem to be over-linked, and judged to play a too-important role.
- The output grammar is much more highly detailed and constrained than the intended grammar.

Questions:

- What happens if there are a lot more verbs? Would this make the determiners less important, more important, or have no effect? (My guess is "no effect")
- To downgrade the importance of determiners would seem to require having sentences without them in it.

TBD:

- Waiting on completion of link-generator so that multiple-sense corpora can be generated. (enabled in lg pull req #1175) Or something like that ... what is the right strategy here? Need to rethink to avoid combinatorial explosion, while also verifying category contents.
- Fix bug to allow multiple-sense word definitions in multiple dict locations.
- Dict generation should auto-handle placing a period at the end of the sentence.

Expt-12

Start work on a single-sense random dictionary. Hit assorted issues with the scripts.

Results:

- non-classified dict has 134391 disjuncts, 11 uni-classes including left-wall. These are raw disjuncts.
- classified dict (i.e. that on which grammatical classification has been performed) has 11286 disjuncts

Time to generate 50 sentences, and all possible sentences, in seconds.

length	time for 50	time for all	num sents
3	3	3	108
4	3	3	779
5	4	4	7107
6	5	8	67935
7	13	31	673812
8	43	370	6855920
9	140		
10	638		
11	963		
12	2180		
13	3364		
14	5850		

Data is semi-meaningless, scripts were broken, data processing would start before data was fully loaded. Try again. Upon restart, the number of sentences generated is order of magnitude lower. Presumably due to corrected clustering; above clustered incomplete lists of disjuncts and thus over-generalized.

Expt-13, expt-14, expt-15

Try again with the same initial corpus. Expt-13 overflowed with fake warning message, so I couldn't see the log; thus expt-14 is an exact rerun. "Exact" in the sense of using the same config. However, the random sampling of pairs during pair counting was different.

Then expt-15 uses exactly the same corpus, with a period at the end of sentence placed manually.

Columns:

- length: length of sentence
- time to generate all sentences, in seconds (expt-13)
- num: number of sentences generated (expt-13)
- expt-14: number of sentences generated
- corpus: number of sentences in input corpus. Capped at 25K sentences for the longer sentences. Second number is how many could have been generated.
- expt-15: redo, but with a period at the end of the sentence.

length	time for all	num expt-13	expt-14	corpus	expt-15
3	3	19	21	10	24
4	2	142	163	23	104
5	2	1130	1356	75	485
6	2	9732	12090	254	2294
7	5	86872	111633	892	10845
8	13	794320	1054583	3402	51673
9	143	7393748	10134151	12728	248242
10	2558	69781807	98702133	25000/48364	1198418
11				25000/187541	5807783
12				25000/733525	28246686

Expt-13 and expt-14 differ only in how the pair-counts were collected (they are randomly different samplings of pairs). Both wildly over-sample the corpus.

The expt-13/14 input corpus lacks periods at the ends of sentences. This seems to be the most likely explanation for the over-generation; i.e. last time a period was lacking, the same thing happened. So expt-15 takes exactly the same corpus - identical copy, and adds a period. This does sharply cut down on the number of sentences, especially the long ones, but still over-generates.

General processing stats:

	expt-14	expt-15
time, pair-counting	109 minutes	107 minutes
pair aid	236/293	259/318
pair dimensions	11 x 10	11x10
pair counts	27367456	29988408
pair sparsity	0.0	0.0
pair entropy	5.96=2.90+3.06	6.14=3.08+3.08
pair MI	0.0014	0.012
time, mpg-parsing	12 minutes	19 minutes
mpg aid	246691	201594/298813
mpg dim	11 x 111712	12 x 97203
mpg counts	1067417	1159801
mpg sparsity	3.19	3.49
mpg MM^T support	134663	104049
mpg MM^T count	2341493237	9220550155
mpg MM^T entropy	2.95	0.232
gram dim	3 x 103552	4 x 92299
gram count	880090	1080580
gram sparsity	1.60	2.46
gram entropy	11.58=11.30+1.09-MI	8.83=8.75+1.15-MI
gram MI	0.806	1.08
dict records	102644	67140

Issues:

- There seems to be a dataset issue: both the disjunct pairs and gram pairs have 1/3rd of them without counts on them. .. They are not being saved, after clustering (clustering causes deletion of many disjuncts, and alteration of counts on all disjuncts.) I'm guessing this failure results in bad MI's? Anyway, its a bug is fixed in the new clustering shell scripts.

After above fix, verify export. gram-1 is a re-export of the original expt-13 run, while gram-4 is export of the fixed run. (Both start with the same disjuncts. I think there's nothing stochastic/random during processing, so it should be repeatable...)

	expt-13/gram-1	expt-13/gram-4
gram dim	3 x 103325	4 x 111320
gram count	881340	939886
gram sparsity	1.60	1.92
gram entropy	11.57=11.29+1.09-MI	11.89=11.61+1.17-MI
gram MI	0.806	0.894
dict records	101922	117807

So .. similar but not the same. How about sentence generation? expt-13-gram-1 and expt-13-gram-1a are identical (the 1a version is from a re-export; so we're exporting the same stuff).

The classes in gram-1 are <b f> <e j> and left-wall. The classes in gram-4 are the same plus <i#uni> ... there was no word i in gram-1 !! Wow, that's a big drop.

length	time for all	expt-13-gram-1	expt-13-gram-4
3	3	19	19
4	2	142	163
5	2	1130	1414
6	2	9732	13147
7	5	86872	125527
8	13	794320	1225346
9	143	7393748	12156101
10	2558	69781807	122141737

OK, so the numbers are dramatically larger. Apparently, this is due to the previously dropped word i. Yikes!

Well, the above is massively under-counting – it is only sampling one random word-draw per class. Since multiple words are in each class...

... anyway, this is nuts, because the connectors need to be classified, instead of issuing new connector types. So more work before something meaningful is possible.

expt-13 vs expt-15 precision, recall

So now that we've got things working, lets look at precision and recall. Clearly precision will be terrible, but maybe recall will be excellent? Compare expt-13-gram-4 to expt-15-gram-2 (which rebuilds after fixing the borked save.

Uhh .. No its not working yet, in the sense that the connector classes are being mishandled. Those need to be grouped correctly before export. More work...

expt-16

Due to absence of left-wall in the above, breaking the dict-compare step, tried again, generating a new dict with walls (after manually adding a wall to the dict of expt-15, and ending punctuation to the dictionary.) Well ... clustering worked quite differently. Here's a summary.

Pair counting seems to be more-or-less the same, slightly higher MI=0.030 which is still minuscule. MPG entropy, counts etc. look similar to the earlier runs.

Gram classification: only 2 words assigned to the same class. Oh, this used the "disinfo" classifier, whereas the earlier runs used the "fuzz" classifier. That could account for everything, I guess. Lets take a look.

	disinfo 3.0 4	discrim 0.5 4	fuzz 0.65 0.3 4
gram dim	11 x 75552	7 x 75667	7 x 75667
gram count	1011444	951215	960646
gram sparsity	3.37	2.77	2.77
gram entropy	11.48=10.01+3.15-MI	10.38=9.94+1.90-MI	10.45=9.91+2.06-MI
gram MI	1.68	1.46	1.52
dict records	80460	77735	77750

Clearly, the resulting clusters are sensitive to the parameters controlling classification. The above parameters seemed reasonable for the large English dataset. They may be unreasonable here!? But this is very unclear.

The MI's are larger, across the board (vs. 0.8 or 0.9 before.) How about sentence generation? We expect disinfo to be more accurate, since it did very little clustering.

length	time disinfo	disinfo	time discrim	discrim
3	61	60	3	18
4	67	523	3	107
5	131	5217	3	752
6	137	51368	3	5371
7	183	488376	5	37923
8	715	4514440	8	268248
9			37	1909447
10			262	13616410

Again, the discrim is under-counting, because of more categorization.

Next step: fix the conjoined clustering, with shapes.

Wow. So MM^T entropy with shapes is 5.03 which is huge compared to the dj-only MM^T so it really is something new and different! With shapes, and with gram-disinfo, there were no merges.

	disjunct disinfo 3.0 4	shape disinfo 3.0 4
MM^T MI		5.03
gram dim	11 x 75552	12 x 75552
gram count	1011444	1015356
gram sparsity	3.37	
gram entropy	11.48=10.01+3.15-MI	11.50=10.02+3.18-MI
gram MI	1.68	1.69
dict records	80460	80807

OK, so looks like shape created no categories at all. So how does that work out for generation?

```
link-generator -l learned -c 123123123 -s 3
```

length	corpus	time for all	expt-16-shape
3	4	38	61
4	21	69	566
5	50	147	5638
6	179	124	55546
7	621	172	531075
8	2246	642	4937036
9	8850		
10	> 25K		

OK, so wildly over-generating sentences, despite effectively no clustering being done. Didn't we do an experiment without clustering?? I can't find it above. Why are we over-generating? How to best explain it? Too small a vocabulary?

- Issue 1: given a fake-lang the generator is failing to generate all possible sentences. Fixed in link-grammar pull req #1175
- Issue 2: there are "accidental" synonyms cause of 1 above: many POS'es are shared in common between many words but are not completely sampled, thus creating "accidental synonyms".

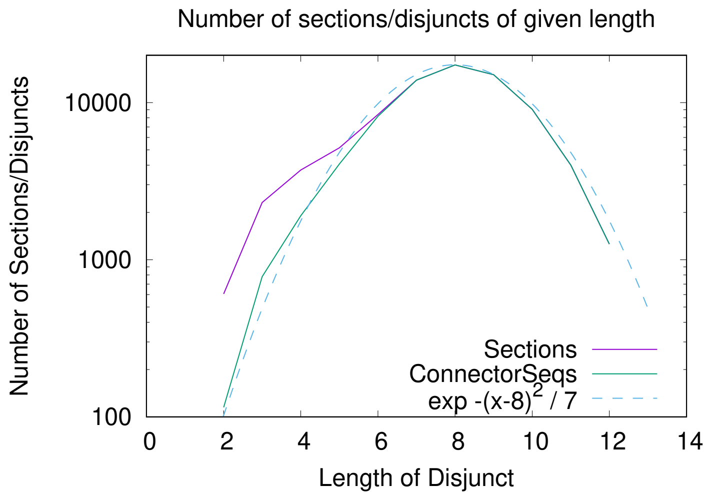
OK, so back to that one, except this time, its in the shape of a bug...

Sections and disjuncts

The Sections that were learned in expt-16 have a surprising number of connectors on them, averaging at 7.7 connectors per section. This seems way too large. What's up with that? Step one: get a more detailed view.

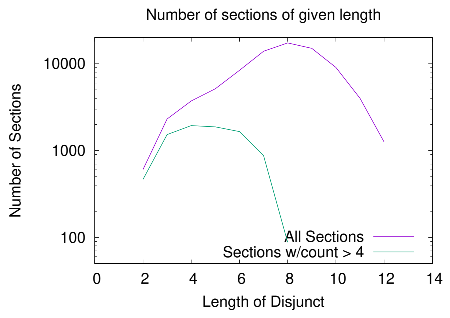
```
# column 1 length
# column 2: number of Section
# column 3: number of ConnectorSeq
#
1  0  0
2  606   115
3  2311  780
4  3723  1904
5  5150  4066
6  8387  8183
7  13907 13896
8  17362 17362
9  15043 15043
10 9061  9061
11 4000  4000
12 1257  1257
13 0
```

So ... long disjuncts appear on one and only one word (as witnessed by identical counts for length 8 and above). Short disjuncts might be shared with multiple words. This implies that words are not mergeable, or rather, the distinct long sections are preventing merger. Here a graph of above.



So its very Gaussian, both peak and tails. All these gaussians here and in earlier results imply that central-limit theorems hold. And all kinds of other classical theorems should hold. Have not been leveraging those theorems, so far. If we did, what would we get? Note it's Gaussian despite the fact that negative connector-seq lengths don't make sense! Right? What would a negative connector-seq length be? A repulsion? an anti-connect statement? "Must never connect"?

What happens if we exclude Sections with a low observation count?



The above shows all sections, and sections that were observed 5 or more times. Apparently, the long, complicated disjuncts are observed very rarely.

How should this be interpreted? Common sense seems to suggest that low-count observations are "noise" and should be cut before any merging is performed. Doing so will certainly increase the similarity of vectors. But are cuts really needed? Perhaps the similarity measures can already deal with these? If so, then the only reason for cuts would be performance, rather than accuracy.

How do these affect cosine-similarity vs MI-similarity? Clearly, for cosine-similarity, low counts means short vector components, and so these will not contribute much to the dot product. Likewise, the MI-similarity is built on a dot-product, so again, these should not contribute much. Even more-so, since the MI-similarity never takes a square-root of the dot product. It does, however, re-weight basis elements in a fashion that I do not yet have a good intuition for. (Or rather, developed an intuition, and now I've forgotten what it was... Hmmm.)

TODO: The Shapiro–Wilk test can be used to determine how close a distribution is to a Gaussian. The Wikipedia article on it is as clear as mud.

April-May 2021

Before starting expt-17 with fixed sampling, it is time to ponder connector merging. This is turning out to be non-trivial. The general upshot of the below is that the original 2019 concept for Shapes and CrossSections is a good, strong idea. However, assorted non-commutativities arise, and how to resolve them appropriately is not entirely clear. Thus, we embark on a journey of discovery...

But first: connector merging is needed because without it, the clustering fails to adequately reduce the size of dictionaries. The meta question is this: if words A and B are determined to be near-synonyms, and assigned to the same cluster, then what should be done with connectors that have A or B in them? Should all such connectors be automatically replaced by the cluster? The naive answer is, “yes, they should be”.

The less-naive response is “but multiple word-senses”. That is, A and B are not just words, but are word-vectors, and vector B may be a linear combination of two different word senses. One of these senses might be synonymous to A, and the other might be completely different. Thus, we want to merge that part of vector B that is (nearly) co-linear with A, while leaving behind a different vector B-prime that is associated with some other word-sense for the word B. When encountering B in a connector, is that B in the sense of wordclass-AB, or is it in the sense of B-prime? If the former, then obviously, that connector should be updated to use wordclass-AB; otherwise, it needs to be left alone, thus implicitly defaulting to B-prime.

TBD: Writing the above, it appears that the need for an explicit B-prime has been overlooked. This could cause havoc, in allowing inappropriate linkages! This is important, and needs to be addressed ...

Connector merging

Some of the questions that arise with connector merging, to figure out:

- How would connector merging affect clustering results?
- If connectors are merged, then how should vectors be handled? (Naive vectors no longer work, because the basis is now different.)
- What is the best or most correct merge algorithm?

Lets look at a toy model. Suppose the dict is

```
A: C+ & D+
B: C+ & D+

C: A- & E+
C: B- & F+

D: A- & G+
D: B- & G+
```

From this, conclude that A and B can be merged. However, the connectors on C cannot be merged, the connectors on D can be merged. The final dict is then (with some abuse of notation)

```
<wclass-AB>: C+ & D+
A B: <wclass-AB>

C: A- & E+
C: B- & F+

D: <wclass-AB>- & G+
```

Next suppose that we have

```
S: C+ & D+
T: C+ & D+

C: S- & J+
C: T- & F+

D: S- & H+
D: T- & G-
```

The determination to merge is now trickier. Naively, S can be merged into `<wclass-AB>` but doing so would wreck the connector set on D, as it implies an `(S- & G+)` which was not observed, and thus not mergeable. ... unless we are willing to create such new unobserved cases.

Merging T looks OK.

Perhaps this looks bad because we are not doing shape vectors. The shape vector for the above would be:

```
A: (C+ & D+) or (C:x- & E+) or (D:x- & G+)
B: (C+ & D+) or (C:x- & F+) or (D:x- & G+)

C: (A- & E+) or ...
C: (B- & F+) or ...

D: (A- & G+) or (A:C+ & x+)
D: (B- & G+) or (B:C+ & x+)
```

So the decision to merge A and B is less clear-cut, when using shapes (they are not perfect synonyms). The shape variant of the S thing is

```
S: (C+ & D+) or (C:x- & J+) or (D:x- & H+)
T: (C+ & D+) or (C:x- & F+) or (D:x- & G+)

D: (S- & H+) or (S:C+ & x+)
D: (T- & H+) or (T:C+ & x+)
```

So using shapes weakens the decision to merge S into AB. The decision to merge T remains strong. So it seems that shapes do offer a stronger foundation on which to make merge decisions. They examine similarity (almost-synonymy) out to a greater distance from the germ. It seems like they also allow things to continue to be treated as vectors, instead of muddling the concept of vectors. After the AB-merge[1] the result would be:

```
<wclass-AB>: (C+ & D+) or (C:x- & E+) or (C:x- & F+) or (D:x- & G+)
A B: <wclass-AB>

C: A- & E+
```

[1]When using the "union-merge" strategy, as described in in `src/gram-projective.scm`. In practice, the merge style used is typically `merge-project`, which would accept only a fraction of `(C:x- & E+)` and `(C:x- & F+)` into the final vector.

```
C: B- & F+

D: (<wclass-AB>- & G+) or (<wclass-AB>:C+ & x+)
```

Now, given this merged-AB thing, when happens when we look at merging S? Well, S is meh, T looks better. The vector for T is comparable to the vector for <wclass-AB> so vector similarity works for that merge decision.

Conclude: the original plan from a few years ago works and holds water. Use shape vectors for merge decisions. Once this is done, connectors can be swept up.

Non-commutativity

The above description pulls a sleight-of-hand, which presumes an algorithm that is able to crawl across individual disjuncts, compare them, and update words with merged word-classes. Such an algorithm can be written (and has been written/prototyped). It leads to some confusion, because the shapes/cross-sections are no longer consistent. Lets call the above the "connector sweep algorithm", or "sweep" for short.

Starting with the above example:

```
<wclass-AB>: (C+ & D+) or (C:x- & E+) or (C:x- & F+) or (D:x- & G+)
```

the sections can be reconstructed from the cross-sections. The reconstructed sections are

```
C: <wclass-AB>- & E+
C: <wclass-AB>- & F+

D: <wclass-AB>- & G+
```

Comparing, the reconstructed section on D is the same as what the sweep algo produced, but the sections on C are *not* what the sweep merge offers. That is, the sweep is not commutative with the creation of shapes. This is a problem for maintaining the consistency between sections and cross-sections as clusters grow. The lack of consistency will cause merge judgments to diverge...

Thus, we have at least two algorithms:

- Sweep-merge, as described above, where connectors are replaced by merged-connectors if and only if the the rest of the connector set is identical. This merge algorithm is naively described, since it does not explain what to do if there are multiple connectors in a connector set that might be merged. It's also naive in that it does not explain how counts (frequencies) are to be handled.

- Reshape-merge, which performs the basic projective merge on the germ-vectors, and then reconstructs Sections from CrossSections, thus restoring consistency between sections and cross-sections. It violates the intuitive correctness of the sweep-merge, but only perhaps because the sweep-merge, as naively described above, assumed the "union-merge" strategy of transferring observation counts for vectors that are not perfectly co-linear. The projective-merge count transfers recognize the non-colinearity, and obtain cluster centroids through weighting formulas.

The above ruminations suggest that reshape-merge enjoys an advantage over sweep-merge, as it keeps the section/cross-section duality consistent.

Anyway, the original 2019 plan for using shapes seems to have been a good plan.

Non-Commutativity, Again

The non-commutativity can be heightened with a slightly richer example. Consider

```
A: (P+ & Q+) or (R+ & S+) or (K- & B+)
B: (P+ & Q+) or (R+ & S+)
C: B- & T+
```

The dictionary entry of C is present to remind us of the fact that, if B+ appears as a connector, then B- must also appear as a connector, somewhere.

Based on the first two sections, a decision might be made to merge, with the count on `(K- & B+)` being small enough that it does not disrupt the merge decision. Expanding this into it's CrossSections, the full vectors are:

```
A: (P+ & Q+) or (R+ & S+) or (K- & B+)
B: (P+ & Q+) or (R+ & S+) or (K- & A:x+) or (C:x- & T+)
```

The merge result is then

```
<wclass-AB>: (P+ & Q+) or (R+ & S+) or (K- & B+)
<wclass-AB>: (K- & A:x+) or (C:x- & T+)
```

The cross-section leads to a reconstruction (reshape) of

```
A: K- & <wclass-AB>+
C: <wclass-AB>- & T+
```

How is this to be interpreted? Let's explore some "common sense" reasoning.

Case A: The count on `(K+ & B+)` is so small that it is considered to be noise, and is completely dropped before merging even starts. In this case, A and B are exactly co-linear (are exact synonyms). The (naive) merge of A and B is completely unproblematic, except that it leaves C without the ability to connect to anything. This can be handled in one of two ways. One way is to notice that, by detailed balance, the counts on this particular C section must also be tiny, and so this section can be dropped from the dictionary.

Another way to avoid this dangling-connector problem is to presume that the dictionary also includes

```
D: L- & B+
```

which would provide a place for that bar B to connect. but if this were the case, then we did the cross-sections on B wrong. Fixing these would have given

```
<wclass-AB>: (P+ & Q+) or (R+ & S+) or (K- & B+)
<wclass-AB>: (K- & A:x+) or (C:x- & T+) or (L- & D:x+)
```

which then reshapes to

```
A: K- & <wclass-AB>+
C: <wclass-AB>- & T+
D: L- & <wclass-AB>+
```

This is now fully linkable, and there are no dangling pure-B connectors.

In conclusion, this seems self-consistent either way: either we can drop the `A:(K+ & B+)` section entirely, and, by detailed balance, we can drop `D:(L- & B+)` also; or we can keep both, and doing it correctly leaves nothing dangling.

Note that we have to be careful with tracking in the merge algo: when reshaping to get the `C:<wclass-AB>- & T+` section, we have to be careful to notice that the `C:B- & T+` section was a donor, and so it should be removed (its counts driven to zero). Otherwise, C would end with both these sections on it, and it would be a bit wonky.

Case B: The count on `(K+ & B+)` is small but not ignorable. It is small enough to not block the merge decision. There are two issues to resolve. The first is easy: the `C:B- & T+` section should be recognized as a donor to `C:<wclass-AB>- & T+`, and removed (its counts driven to zero). This is easy enough to determine at the time of the merge.

The more difficult issue is what to do about the `<wclass-AB>:(K- & B+)` section, which appears to have a dangling B connector, and the reshape of `A:(K- & <wclass-AB>+)`, which seems to be a double-count. The first leaves a dangling A, the second leaves a dangling B. It seems fairly clear that these should be harmonized, merged together, to give `<wclass-AB>:(K- & <wclass-AB>+)`. It seems that this can be reasonably inferred and performed at the time of creation, since the donors are readily identified.

Case C: The count on `(K+ & B+)` is large, large enough to split. That is, A should be understood to be the direct sum of two distinct word-senses, with one word-sense being `<wclass-AB>` and the other being `<A-prime>:(K+ & B+)`. So, if were were able to be absolutely sure that A-prime was really a distinct word-sense, then we should transfer none of the counts from the originating section `A:(K+ & B+)` to the `<wclass-AB>` section.

So, starting with the vectors

```
A: (P+ & Q+) or (R+ & S+) or (K- & B+)[n]
B: (P+ & Q+) or (R+ & S+) or (K- & A:x+)[n] or (C:x- & T+)
```

where square-bracket-n is the count on that section, we create a merge result of the form

```
<wclass-AB>: (P+ & Q+) or (R+ & S+)
<wclass-AB>: (C:x- & T+)
<A-prime>:  (K- & B+)[n]
B: (K- & <A-prime>:x+)[n]
```

The cross-sections leads to a reconstruction (reshape) that appears to be self-consistent, so I don't see any problems here.

If we were to split [n] into some fractional parts, then this would reduce to a combination of case B and the current case, so that should also work.

Connector counts

(TBD, this section needs to be harmonized with the new text above... the below was written before the above was rewritten...) Lets go through above exercise, this time with counts. Suppose the dict, with observation counts in square brackets, is

```
A: C+ & D+ [na]
B: C+ & D+ [nb]

C: A- & E+ [nca]
C: B- & F+ [ncb]

D: A- & G+ [nda]
D: B- & G+ [ndb]
```

The shapes, with counts, are

```
A: (C+ & D+)[na] or (C:x- & E+)[nca] or (D:x- & G+)[nda]
B: (C+ & D+)[nb] or (C:x- & F+)[ncb] or (D:x- & G+)[ndb]

C: (A- & E+)[nca] or ...
C: (B- & F+)[ncb] or ...

D: (A- & G+)[nda] or (A:C+ & x+)[na]
D: (B- & G+)[ndb] or (B:C+ & x+)[nb]
```

Merging A and B, with 100% of count transfer, gives

```
A B: (C+ & D+)[na+nb] or (C:x- & E+)[nca] or (C:x- & F+)[ncb] or (D:x- & G+)[nda+ndb]

C: (A- & E+)[nca] or ...
C: (B- & F+)[ncb] or ...

D: (A- & G+)[nda] or (A:C+ & x+)[na]
D: (B- & G+)[ndb] or (B:C+ & x+)[nb]
```

Looking at the connectors on D, we see that they are mergeable, and that the counts are consistent. So, based on this toy model, we can either try to merge connectors directly, or we can, at a later date, merge connectors by reconstructing them from merged shapes. Doing it either way should give the same counts: the operations are commutative.

This is not entirely obvious. It seems to work for the toy example. It seems like the toy example could be converted into a full proof. Yet ... are we missing something? Best bet is to write the code both ways, and verify numerically that the operations are commutative.

Fractional counts

Lets try again, this time with fractional counts. Suppose that instead of merging 100% of B into A, we merge only a fraction $0 \leq y \leq 1$ of the count. This gives

```
A B: (C+ & D+)[na+y*nb] or (C:x- & E+)[nca] or (C:x- & F+)[y*ncb] or (D:x- & G+)[nda+y*ndb]
B: C+ & D+ [(1-y)nb] or (C:x- & F+)[(1-y)ncb] or (D:x- & G+)[(1-y)ndb]

C: (A- & E+)[nca] or ...
C: (B- & F+)[ncb] or ...

D: (A- & G+)[nda] or (A:C+ & x+)[na]
D: (B- & G+)[ndb] or (B:C+ & x+)[nb]
```

Then, apparently, the counts will be consistent if and only if the same fraction is used when merging connectors.

Connector merging, with counts

To get all of the above correct, there is a series of unit tests. They work well, but one of the more complex ones has become painfully difficult to understand and debug. It is reviewed here. But first, a change of notation to make it more compact:

- The entry `A:(B- & C+)` will be written as `(A, BC)`. Here, the parenthesis denote a pair (a matrix entry). The letter sequence is just the connector sequence with the directional indicators ignored.
- Entries with word-classes, such as `<wclass-AB>:(K- & B+)` will be written as `({AB},KB)`. The word-class is denoted with set-notation curly braces. Similarly, `A:(K- & <wclass-AB>+)` will be written as `(A,K{AB})`.
- Cross-sections, which were written above as `D:(A:C+ & x+)` will be written as `[D, <A, Cv>]`. The angle brackets denote the shape, and the lower-case `v` denotes the location of the variable in the connector sequence. The square brackets just serve to remind that a cross-section is being discussed.
- A property called "detailed balance" is introduced. This is the idea that corresponding sections and cross-sections should have exactly the same observation count on them. Thus for example, given a section `(A, BC)` which was observed N times, one expects that the two cross-sections derived from it, namely `[B, <A, vC>]` and `[C, <A, Bv>]` are also both observed N times each. Prior to any merging, detailed balance holds "automatically" or tautologically, as a trite statement about how counting is done. The goal is that connector merging should preserve detailed balance as a property. It is assumed to be a desirable property, and is enforced in the code and unit tests.
- Counts will be denoted with a lower-case `n` written in front of the pair. Thus, `n(A, BC)` would be the number of times that `(A, BC)` was observed.

First merge

The troublesome test is '`connector-merge-tricon.scm`'. The relevant portion is as follows. The dictionary is assumed to contain many entries; the troublesome subset is this:

```
(j, abe)
(f, abe)
```

A decision is made to merge the vectors for `e` and `j`, based on other dictionary entries not shown here. The "projective merge" strategy is used, so that a fraction $0 \le p \le 1$ of the count is merged whenever one of the two vectors is missing an entry at a given basis element. In this case, the merge, denoted with an arrow, is

```
none + (j, abe) -> p * ({ej}, abe) + (1-p) * (j, abe)
```

where '`none`' denotes that there is no section `(e, abe)` and so the projective merge was used. That is, the count on `(j, abe)` is reduced to $(1-p)$ of it's earlier value, and the remaining p is transferred over to `({ej}, abe)`. That is, the total counts are preserved. That is,

$$n'(\texttt{\{ej\}, abe}) = pn(\texttt{j, abe})$$

where n' denotes the count after the merge, and the unprimed n is the count before the merge.

From the connector merging discussion above, we conclude that `({ej}, abe)` should be rewritten to `({ej}, ab{ej})`. The count should be as above, that is:

$$n'(\texttt{\{ej\}, ab\{ej\}}) = pn(\texttt{j, abe})$$

The vector on `e` includes the cross-sections

```
[e, <j,abv>]
[e, <f,abv>]
```

These merge in a similar fashion:

```
[e, <j,abv>] + none -> p * [{ej}, <j,abv>] + (1-p) * [e, <j,abv>]
[e, <f,abv>] + none -> p * [{ej}, <f,abv>] + (1-p) * [e, <f,abv>]
```

From detailed balance, we deduce the two new sections

```
(j, ab{ej})
(f, ab{ej})
```

The counts on these are

$$n'(\texttt{j, ab\{ej\}}) = pn(\texttt{j, abe})$$

and

$$n'(\texttt{f, ab\{ej\}}) = pn(\texttt{f, abe})$$

From the connector merging discussion above, we conclude that `(j, ab{ej})` should be rewritten to `({ej}, ab{ej})`. The first identity arrives at the same count as before, so this rewrite appears to be self-consistent. Everything works out.

Second merge

The first merge is more-or-less straightforward. The trouble comes with the second merge. Here, it is decided that the vector for `f` should be merged into `{ej}`. The preservation of detailed balance creates subtleties and ambiguities.

The final count on `({ejf}, ab{ejf})` gets contributions from three sources:

- The starting count on `({ej}, ab{ej})`, which is

  $$n'(\texttt{\{ej\}, ab\{ej\}}) = pn(\texttt{j, abe})$$

  as given above.

- A contribution from `(f, ab{ej})`, which was created in the first merge, via detailed balance from the earlier cross-section `[{ej}, <f, abv>]`. This is merged in it's entirety into the the existing `({ej}, ab{ej})`. The projection merge is

  ```
  ({ej}, ab{ej}) + (f, ab{ej}) -> ({ejf}, ab{ej})
  ```

  This merge absorbs the entire count on `(f, ab{ej})` because `({ej}, ab{ej})` already exists. The contribution is thus $n'(\texttt{f, ab\{ej\}}) = pn(\texttt{f, abe})$. Rewriting then promotes `({ejf}, ab{ej})` to `({ejf}, ab{ejf})`.

- A contribution from `(f, abe)` via the projection merge

  ```
  none + (f, abe) -> q * ({ejf}, abe) + (1-q) * (f, abe)
  ```

  and then the subsequent rewrite of `({ejf}, abe) -> ({ejf}, ab{ejf})`. This contribution is $qn'(\texttt{f, abe}) = q(1-p)n(\texttt{f, abe})$.

The total of these three contributions is then

$$\begin{aligned} n''(\texttt{\{ejf\},ab\{ejf\}}) =& n'(\texttt{\{ej\}, ab\{ej\}}) + n'(\texttt{f, ab\{ej\}}) + qn'(\texttt{f, abe}) \\ =& pn(\texttt{j, abe}) + pn(\texttt{f, abe}) + q(1-p)n(\texttt{f, abe}) \end{aligned}$$

Note that this result is history-dependent: merging j into e first, then f gives a different result than merging f into e, then j (and presumably different than the third possibility, of merging f and j first, and only then adding e).

An open question is whether there is a way of performing the merges that are history-independent, and what would that mean.

Again, additional details are in the test file '`connector-merge-tricon.scm`'.

Connector Merging, Conclusion

After much work: there are ten unit tests, all passing, with the final fix in commit 5e1d7dfb94867f22642d7cdf0621a833bb96092e of 24 May 2021 which fixes a problem not found in the unit tests; it requires a real-world test-case. Need to run `(check-balance LLOBJ)` to evoke it.

expt-19 (May 2021)

Moving on... expt-19 reuses the same corpus as expt-16, and, in order to be comparable to earlier results, reuses the `pair-counts` and the `mpg-parse` disjuncts from expt-16. Here's the dataset statistics, from `print-matrix-summary-report`, without and with shapes:

	cset-only	w/ shapes
rows	12	12
columns	75688	587172
entries	80832	701606
sparsity	3.4901	3.3281
avg. obs./disjunct	12.561	6.4922
entropy	0.7221	5.0270
MMT support	80832	701606
obs. count	3.9e9	15e9

Time to compute the pair-distances was about 0.03 to 1.5 seconds for the cset-only vectors, and 6 to 16 seconds for the shapes. So, at least an order of magnitude slower. That's bad.

The entropy is much much higher, which I interpret as a good thing, indicating that the data is of higher quality.

Using `(gram-classify-greedy-disinfo psa 3.0 4)`, there weren't any merges that got done. That's because similarity never got above 3.0. Here's a table of all positive MI similarities, without shapes, and with. It's sorted on the shape column when the shape column MI is positive, otherwise sorted on the cset-only column.

pair	MI cset-only	MI w/ shapes
!-i	$-\infty$	2.3578
g-i	3.2143	1.8643
a-i	-2.388	1.2513
f-c	1.3032	1.1714
h-c	1.2824	1.0837
f-h	1.2800	1.0757
b-g	0.8982	1.0559
a-WALL	$-\infty$	0.9799
j-b	0.3929	0.5846
j-g	0.3929	0.5265
d-f	0.8141	0.5170
e-b	0.3725	0.5066
d-c	0.7817	0.5026
e-j	0.2484	0.4612
e-g	0.3896	0.4503
d-h	0.8126	0.4446
e-d	-0.320	0.0049
b-i	1.4862	-1.038
a-h	0.6048	-0.079
a-f	0.5596	-0.190
a-c	0.5544	-0.266
d-a	0.5501	-0.257
e-i	0.0819	-1.401
j-i	0.0185	-1.254

In general, any MI of less than 4 is ... pretty distant. Although that statement is true for large-vocabulary systems; here the vocabulary is tiny: 12 words, so I guess an MI greater than 1.0 is actually pretty good.

The distance '!-i' is alarming. I guess. Seems to indicate that 'i' is usually at the end of sentences. The 'a-WALL' distance suggests that 'a' is frequently at the start of the sentence. If so, this is not obvious from casual inspection of the corpus.

A core problem is that this is a very mixed grammar: lots of ambiguity, lots of word senses, no particular clean factorization. Just looking at it, its quite cloudy as to the actual structure. In particular, although each word belongs to only one word-class, the word-classes have lots of mixed, shared POS entries, and each POS is a seemingly random (duhh) unstructured mess. For example:

- pos-e has single, divalent, trivalent disjuncts on it. Except for a few words, most of English is not like that.
- pos-c has one disjunct. It's identical to pos-d, which has two disjuncts.
- pos-e has a huge number of disjuncts on it, as do pos-i and pos-j. This seems to allow very grammatically complex sentences to be generated, which "of course" are going to be very hard to decode.

Overall, the grammar appears to be over-complex, and very unlike a natural language grammar. it seems unlikely, just from eyeballing it, that the grammar could be untangled without a very large, exhaustive examination of the corpus. This is a bad experimental base.

Issues:

- Is there any sense in which the word "mixing" is appropriate, in it's technical sense (from ergodic theory?) Can we define mixing from the point of view of disjunct ambiguity? Of the indiscernability of grammars given a corpus?
- Is the automatic grammar generation controlling sufficiently for word-senses? Yes, there's a tune-able parameter for that, but some disjuncts accidentally appear in multiple POS, thus making those POS at least partly synonymous.
- Is the current automatic grammar generation API appropriate? It was based on an intuitive sense of factorization, but the randomness seems to easily generate ambiguous grammars.
- How does one measure the complexity of a grammar?
- Is there some easy way of writing down its factorizability?
- Is there a way of proving that two grammars are equivalent? If two different grammars generate the exact same corpus, then is there some algorithm that can transmute one grammar into the other? How is this found/discovered?
- How can one characterize human natural language grammars? That is, if an artificial grammar is generated, how can we know if it is similar to a natural human language? I don't think the rainbow of human natural languages lines up well (or at all) with the axes of tune-able parameters in the grammar generator.

Conclude: the expt-19 grammar is over-complicated, ambiguous, mixed, ugly. We need to restart with a simple grammar.

Also conclude: I do not understand how the ambiguity of grammars works. I do not really understand how factorization works. The artificial grammar generator "works" but I don't understand what it is generating. I don't know how close it is to typical human grammars. There's a bit of a "back to the drawing board" moment here.

expt-20 (May 2021)

Start again, this time with a simple, relatively unambiguous, relatively unmixed grammar. Perhaps even with a tiny artificial subset of English!? Just to make eyeballing easier?

July 2021 - Projective Merge and Entropy Maximization

OK, Above work resulted in a bunch of automation scripts, and a few bug-fixes, a completion of the "shapes" work, but not much in the way of insight. So, restart processing of English. A rather small corpus reveals a conceptual bug in the projective-merge strategy, when used with mutual information. Projective merge works great (I think .. I guess??) with cosine distance, but not MI. This section describes the problem, and explores solutions.

The small corpus is the "run-2" corpus of the English work, its a truncated copy of "tranche-1" consisting of 3026 articles, 426941 sentences, 8133834 word instances. Word-pair counting went well (except for the handling error that truncated the corpus), as did disjunct formation. The MM^T stats were computed for the disjuncts, including shapes.[2] Everything is fine (nominal) until merge. Projective merge fails dramatically – unrelated words are being merged. Why?[3]

What's the pair-wise MI? The MI here is called the "entropic similarity" in other texts in this directory. It's kind-of the same thing, once one adjusts definitions appropriately. Lets call it MI for short. Lets explore the top seven words that got merged, shown in the table below.

Symmetric-MI							
	of	to	in	he	it	that	said
of	7.460	3.720	4.443	-1.74	-1.44	0.619	-1.43
to		6.848	3.019	0.449	-0.18	0.935	0.070
in			5.699	1.348	0.255	1.517	-0.53
he				3.968	1.854	1.691	1.198
it					2.405	0.819	0.646
that						3.664	0.468
said							3.223

Looks pretty reasonable, right? The self-MI of "it" is shockingly low. Some of the other self-MI's are on the low side, too. But whatever. Small dataset. Asking for an MI > 3.0 for a merge to take place seems like it should provide good results.

The current code base for projective merge recommends a fraction of 0.187 for merging "of" and "to", which seems low, but acceptable. So lets do the merge, by hand (see notes in 'run-2/README' for details. The result is '(WordClassNode "of to")'. What is the MI between this, and the other words? A disaster. See table below.

Symmetric-MI								
	of-to	of	to	in	he	it	that	said
of-to	10.009	8.411	8.216	6.044	1.937	1.629	2.970	1.900
of		11.70	$-\infty$	5.294	-4.44	-3.34	1.012	-3.25
to			11.92	3.558	0.874	-0.25	1.432	-1.97

[2]It contains 6029 words, 98102 disjuncts, a total of 171922 word-disjunct pairs, has a sparsity of 11.748, a total of 4006152 observations for an average of 23.302 per pair.

[3]One possible answer to this question is that there is a procedural error at play. It is this: the MM^T marginals were computed for the full dataset, and then the dataset was trimmed to discard low-count rows and columns. However, the full marginals were still used in computing the MI. This throws everything off, because .. rows and columns are missing. The correct process would have been to recompute the MM^T marginals *after* trimming! The MM^T marginals depend on the support marginals as well. Which were also cached. This has lead to some confusion in the below. I think the results are unchanged, but I'm no longer confident. 27 July 2021

OK, so what's wrong with that? The naive expectation is that the MI(cluster, "of") and MI(cluster, "to") should be low or negative. It's not - its huge. The naive expectation is that the MI of the cluster to the non-preposition words should remain near zero (small positive or negative) and instead it got higher, not lower! Yow!

Note that the "of" and "to" vectors are the new vectors, with the projected parts removed. Thus, the new MI (of,to)=$-\infty$ is as expected: all overlap is now completely gone. That the MI(of, non-prep) and MI(to, non-prep) has gotten lower is a good sign.

So what's going wrong, here? Well, we failed to define the projective merge in such a way that we minimize and maximize the assorted overlaps. So lets fix that.

Conclusion

The following sections explore the problem above. The answer, in retrospect, appears to be obvious: of the parameterized projective merge, only the overlap merge maximizes entropy. Adding in any fraction of the union merge lowers the total mutual information. It smears out the word-senses, and damages word-sense disambiguation.

If the answer is so obvious, then how did we get into this mess to begin with? Well, it seemed, at the time, that, because the input data is so noisy, that the number of observations is so incomplete, that there might be some advantage for extending the overlap merge with some "small" fraction of the union merge. In retrospect, this appears to have been a failed idea.

The overall concern is still valid: for a small corpus, there is a lot of noise, and perhaps there does need to be some kind of generalization taking place. But perhaps the smearing provided by the union merge is ... not a good idea ... at present. However ...

However, trying to always maximize the entropy runs the risk of stumbling into local maxima, and being trapped by them. This is an old machine-learning problem: how to avoid local maxima. The union merge might eventually be a good way of jumping out of local maxima. However, at this time, I don't understand what's going on clearly enough. It seems to have been a premature feature. Maybe. At any rate, future work should set a baseline of pure overlap merges, zero union merges. If any part of the union is to be mixed in, it should probably be quite small – well less than a percent, rather than the 30% which almost all earlier work made use of.

If the union-fraction is greater than zero, then it should be kept small enough so that the MI between the newly created cluster, and the remainder of the words that were put into the cluster remains less than the cutoff MI that is used to determine if a merge is to be undertaken. Examining the experimental data, below, indicates that this fraction is indeed tiny: its about 0.003 (a third of a percent) for the case that was examined.

Rule of thumb: if there is a merge fraction, then if should be set to

$$f = 1/2^{\max(MI(a,a),MI(b,b))-MI_{cut}}$$

where a,b are the two words to be merged, $MI(a,a)$ is the self-MI and MI_{cut} is the cutoff, below which a merge will not be performed. Any fraction larger than this will results in an MI between the new class, and the remainder of the words a,b that is *larger* than the original $MI(a,b)$ itself ... and that's a disaster. This is a rule of thumb, because, I guess a more precise value could be given, but getting it requires some more algebra which I'm too lazy to do.

Similarity and Projection

Let's review the definition for the MI, and the projection.

Entropic Similarity and Cosine Distance

This is a copy of what is in the 'connector-sets-revised.pdf' paper, chapter 6 (pages 39-44). There some additional experimental data in the 'diary-part-one.pdf', pages 102-103.

A word w is associated with a vector $N(w,d_j)$ where the d_j are the disjuncts observed on the word, and N is a count of the number of times that the word-disjunct pair was observed.

For words w,u, define the dot product (inner product) between the words as

$$i(u,w)=\sum_d N(u,d)\,N(w,d)$$

This can be turned into a *bona-fide* joint probability by writing

$$p(u,w)=i(u,w)/i(*,*)$$

where, as always, * denotes a wild-card – here, a sum over all words:

$$i(*,*)=\sum_{u,w} i(u,w)$$

There is a corresponding marginal probability

$$p(w)=p(w,*)=\frac{i(w,*)}{i(*,*)}$$

The entropic similarity between two words is

$$MI(u,w)=\log_2\frac{p(u,w)}{p(u)\,p(w)}$$

and written in this form, this is clearly the conventional (fractional) MI between two words.

It's worth comparing this to the cosine distance

$$\theta(u,w)=\arccos\frac{p(u,w)}{\sqrt{p(u,u)\,p(w,w)}}$$

Note that they both start with the dot product in the numerator. The cosine distance is invariant under rotations (orthogonal transformations) in Euclidean space. However, probability space is not Euclidean, so it is a "category error" to work with cosine distance applied to probabilities. By contrast, the MI is "obviously correct" when working with probabilities. In practice, the two are correlated. See page 43 of ‘connector-sets-revised.pdf‘ for experimental data.

The code for computing this stuff can be found in the AtomSpace github repo, https://github.com/opencog/atomspace/blob/master/opencog/matrix/symmetric-mi.scm To avoid absurdly long compute times, this code uses the MM^T concept (described in connector-sets) to perform and cache partial results that can be quickly combined to obtain the desired MI value. Note, its impossible to pre-compute the MI for any but the smallest vocabularies, as there are just too many words.

Projection Merge

Given two words (word-vectors), the projection merge create three new vectors: a merged vector, and remainders of the two original vectors. This is most easily described in terms of the basis vectors. Define the set of all disjuncts with non-zero counts on word w:

$$D(w) = \{d : N(w,d) \neq 0\}$$

Given a real number $0 \leq f \leq 1$, the merged vector g of the two words w,u has counts

$$N(g,d) = \begin{cases} N(w,d) + N(u,d) & d \in D(w) \cap D(u) \\ fN(w,d) & d \in D(w) \setminus D(u) \\ fN(u,d) & d \in D(u) \setminus D(w) \end{cases}$$

Clearly, the support for g is $D(g) = D(w) \cup D(u)$ whenever $f > 0$ and it is $D(g) = D(w) \cap D(u)$ when $f = 0$. The two new words are u' and w' which are just the old words, with the overlaps removed:

$$N(w',d) = \begin{cases} 0 & d \in D(w) \cap D(u) \\ (1-f)N(w,d) & d \in D(w) \setminus D(u) \end{cases}$$

and likewise for $N(u',d)$.

This projection merge is designed to preserve the total count:

$$N(g,d) + N(w',d) + N(u',d) = N(w,d) + N(u,d) \tag{1}$$

This equation can be called “detailed balance”, as it is in thermodynamics.

The merged vector g is meant to correspond to a “grammatical class”; it is a vector just as any other word-vector, but it is meant to capture the “average” of the two words it is made out of. The role of the fraction f is to handle the situation of the words u and w having multiple word-senses. The idea here is that the set $D(w) \cap D(u)$ captures those disjuncts for which both u and w have the same word-sense (for example, the parts of u and w that are nouns) whereas $D(w) \setminus D(u)$ and $D(u) \setminus D(w)$ are the disjuncts that belong to other word-senses (*e.g.* the parts of u and w that are verbs). Due to inadequate statistics and systemic noise, a non-zero f might help in smoothing out erroneous assignments of disjuncts to word-senses. In the example above, a value of $f = 0.187$ was used.

The projection described above is the same as that described and implemented in https://github.com/opencog/learn/blob/master/scm/gram-projective.scm (as of this writing).

Maximum Mutual Information

Well, obviously as the example above demonstrates, there's something not quite right with the projection merge, at least, when one works with mutual information.

The projection merge does seem to make sense (mostly) when one thinks of vectors that inhabit Euclidean space: One can easily define the sum of two vectors; and to deal with the fact that the word vectors are themselves sums of multiple senses, one can try to project back out the parts that don't have shared support. The problem with working in Euclidean space is that the counts can go negative: the inner product on Euclidean space really is the cosine (dot) product, and simple linear algebra sense some vector components negative. This is undesirable, as it prevents counts from being interpreted as statistical frequencies.

For probabilities, the correct goal is to maximize the mutual information (maximize the entropy). There seem to be two ways to define the total MI of the system. One is to write

$$MI_{tot} = \sum_{u,w} p(u,w)\, MI(u,w)$$

Note that since both p and MI are symmetric, this has the effect of double-counting the off-diagonal entries. This seems to underweight the diagonal. Another possibility is to double-weight the diagonal:

$$MI_{alt} = \sum_{u \geq w} p(u,w)\, MI(u,w)$$

Not clear, right now, which is better or more correct.

A bit of articulation helps clarify things. From the definition of MI, one has

$$\begin{aligned}
MI_{tot} =& \sum_{u,w} p(u,w)\, MI(u,w) \\
=& \sum_{u,w} p(u,w) \log_2 \frac{p(u,w)}{p(u)\, p(w)} \\
=& \sum_{u,w} p(u,w) \log_2 p(u,w) - \sum_{u,w} p(u,w) \log_2 p(u) - \sum_{u,w} p(u,w) \log_2 p(w) \\
=& \sum_{u,w} p(u,w) \log_2 p(u,w) - 2 \sum_{w} p(w) \log_2 p(w) \\
=& H_{joint} - 2H_{marg}
\end{aligned}$$

where

$$H_{marg} = \sum_{w} p(w) \log_2 p(w)$$

is the "marginal entropy".

Detailed balance

Detailed balance, eqn 1 means that the merge affects only the rows and columns of the merged words. This is somehow "intuitively obvious", but I'll belabor the topic here, to make sure we're not making any mistakes. In the following, let a,b be the two words to be merged, creating a category g.

Lemma: For $w \neq a,b$, the row and column sums are unaffected by the merge. That is, one has $i(w,*) = i'(w,*)$.

Proof: Let a,b be the two words to be merged. Then for $w \neq a,b$, one has

$$\begin{aligned}
i(w,a) + i(w,b) =& \sum_d N(w,d)\left[N(a,d) + N(b,d)\right] \\
=& \sum_d N(w,d)\left[N(g,d) + N\left(a',d\right) + N\left(b',d\right)\right] \\
=& i(w,g) + i\left(w,a'\right) + i\left(w,b'\right)
\end{aligned}$$

No other sums are affected, so that, for $u,w \neq a,b$ one has

$$i(w,u) = i'(w,u)$$

Therefore,

$$\begin{aligned}
i(w,*) =& \sum_u i(w,u) \\
=& i(w,a) + i(w,b) + \sum_{u\neq a,b} i(w,u) \\
=& i(w,g) + i\left(w,a'\right) + i\left(w,b'\right) + \sum_{u\neq a,b} i(w,u) \\
=& i'(w,*)
\end{aligned}$$

where the i' sum includes (runs over) a',b' and g. □

This enables a key theorem.

Theorem: The total count $i(*,*)$ is unaffected by the merge. That is, $i'(*,*) = i(*,*)$ where the prime denotes the post-merge sum.

Proof: Split out the affected rows and columns. First, the corner case:

$$\begin{aligned}
i(a,a) + i(a,b) + i(b,a) + i(b,b) =& \sum_d \left[N(a,d) + N(b,d)\right]\left[N(a,d) + N(b,d)\right] \\
=& \sum_d \left[N(g,d) + N\left(a',d\right) + N\left(b',d\right)\right]\left[N(g,d) + N\left(a',d\right) + N\left(b',d\right)\right] \\
=& i(g,g) + i\left(g,a'\right) + i\left(g,b'\right) + i\left(a',g\right) + i\left(a',a'\right) + i\left(b',g\right) + i\left(b',b'\right) \\
=& i(g,g) + i\left(a',a'\right) + i\left(b',b'\right) + 2i\left(g,a'\right) + 2i\left(g,b'\right)
\end{aligned}$$

Two of the terms vanish: $i(a',b') = i(b',a') = 0$, which is arrived at by noting that the merge has arranged that $N(a',d) = N(b',d) = 0$ for $d \in D(a) \cap D(b)$.

Even if these terms did not vanish, one still arrives at

$$i(a,*) + i(b,*) = i(g,*) + i\left(a',*\right) + i\left(b',*\right)$$

Plugging through,

$$\begin{aligned}
i(*,*) =&\sum_u i(u,*)\\
=&i(a,*)+i(b,*)+\sum_{u\neq a,b} i(u,*)\\
=&i(g,*)+i(a',*)+i(b',*)+\sum_{u\neq a,b} i(u,*)\\
=&\sum_{u'} i(u',*)\\
=&i'(*,*)
\end{aligned}$$

Phew. That was complicated, given that the result seems obvious. □

Corollary: Likewise, for the probabilities: if a,b are the two words to be merged into g,then $p(a,*)+p(b,*)=p(g,*)+p(a',*)+p(b',*)$.

Proof: Divide by $i(*,*)$. □

Corollary: The marginal probability of rows/columns not being merged is unaffected by the merge. That is, $p(w)=p'(w)$ for $w\neq a,b$.

Proof:

$$p(w)=p(w,*)=\frac{i(w,*)}{i(*,*)}=\frac{i'(w,*)}{i'(*,*)}=p'(w)$$

Follows from the lemma and the theorem above. □

Corollary: The mutual information of rows/columns not being merged is unaffected by the merge. That is, $MI(u,w)=MI'(u,w)$ for $u,w\neq a,b$.

Proof:

$$\begin{aligned}
MI(u,w) =&\log_2\frac{p(u,w)}{p(u)\,p(w)}\\
=&\log_2\frac{p'(u,w)}{p'(u)\,p'(w)}=MI'(u,w)
\end{aligned}$$

Follows as before. □

From the above, it is clear that the MI_{tot} splits into an invariant part, and a part affected by the merge. Write

$$\begin{aligned}
MI_{tot} =&\frac{1}{2}\sum_{u,w} p(u,w)\,MI(u,w)\\
=&\frac{1}{2}\sum_{u,w=a,b} p(u,w)\,MI(u,w)+\frac{1}{2}\sum_{u,w\neq a,b} p(u,w)\,MI(u,w)\\
=&MI_{merge}+MI_{invariant}
\end{aligned}$$

The focus is then on how MI_{merge} changes. The change in the mutual information due to merging is captured by the difference, defined as

$$S=\sum_{u,w=g,a',b'} p(u,w)\,MI(u,w)-\sum_{u,w=a,b} p(u,w)\,MI(u,w) \qquad (2)$$

A suitable name for this would be the “relative entropy”, I guess. Seems reasonable.

Experimental Exploration

The goal of later sections will be to find the extrema (maxima, minima) of eqn 2 by algebraic means: that is, to find a value for the parameter f that yields the projective merge that maximizes the mutual information. As it turns out, a whole lot of rather tedious algebra is required. Thus, its time for an experimental interlude. Given the dataset above, we can vary the merge parameter by hand, and see what happens.

First, a review of the dataset and the computational techniques. The dataset is ‘r2-mpg-trim-40-8-5.rdb‘. It contains 6029 words and 98102 disjuncts, and a total of 171922 word-disjunct pairs that were observed a total of 4006152 times. That is, each word/disjunct pair was observed an average of 23.302 times. The dataset is very sparse, with a log2 sparsity of 11.748. This is a trimmed dataset: all words with less than 40 observations were discarded; all disjuncts with less than 8 observations were discarded, and all pairs with less than 5 observations were discarded. See the experiment README file for more info on row and column support, average lengths, etc.

Here are the probabilities and MI, before merge:

word-pair	$p(u,w)$	$MI(u,w)$
of,of	1.199e-3	7.460
of,to	7.599e-5	3.720
to,to	5.623e-4	6.848

The total probability is then 1.913e-3 and the weighted MI is 1.336e-2 and the average MI is then 6.9829. To repeat:

$$\sum_{u,w=a,b} p(u,w) = 1.913\times 10^{-3}$$

$$\sum_{u,w=a,b} p(u,w)\,MI(u,w) = 1.336\times 10^{-2}$$

$$\frac{\sum_{u,w=a,b} p(u,w)\,MI(u,w)}{\sum_{u,w=a,b} p(u,w)} = 6.9829$$

for $a,b = of,to$.

Lets repeat this calculation, after merging, with $f = 0.18715$ as before.

pair	$p(u,w)$	$MI(u,w)$
of,of	4.831E-4	11.695
to,to	3.153E-4	11.923
g,g	7.465E-4	10.009
g,of	1.105E-4	8.4107
g,to	7.208E-5	8.2156
total	1.910E-3	10.431

The MI reported in the total column is the parameter-dependent entropy from above:

$$S' = \sum_{u,w=g,a',b'} p(u,w) MI(u,w)$$

where a' is what's left of the vector for the word "of", after projection-merging the common part. Likewise, b' is what's left of "to". It is numerically confirmed that a' and b' are orthogonal to each-other: that $p(a',b') = 0$ after the merge. This is as expected — the common components have been merged. That $p(g,a') \neq 0$ is not surprising: an additional portion of a was merged into g, beyond what is strictly possible with the overlap merge. That is, g and a' are intentionally not orthogonal. That's what this is all about – finding out if something broader than a pure overlap merge is somehow advantageous.

Annoyingly, the detailed balance seems to be a bit off — 1.910e-3 *vs.* 1.913e-3 before. That's a fairly hefty amount of rounding error. Surprisingly large... There are 7620 sections and 7620 shapes being merged; calculations should be double-precision, so this rounding error is irritating.

Anyway, that's the baseline. Lets try a range of merge fractions. Again, to reiterate: $f = 0$ corresponds to a pure overlap projection merge: the cluster g consists of only those disjuncts that are shared in common by a and b. Setting $f = 1$ corresponds to the union merge: all disjuncts on a and b are moved into g so that a' and b' are empty after the merge. Other fractions $0 < f < 1$ interpolate linearly between the overlap merge and the union merge. The table below shows what happens.

frac	total MI	MI(g,g)	MI(g,of)	MI(g,to)	p(of,of)	p(g,of)
0.0	11.312	10.499	$-\infty$	$-\infty$	7.312E-4	0
1e-6	11.312	10.499	-8.816	-9.011	7.311E-4	7.27E-10
1e-5	11.311	10.499	-5.494	-5.689	7.311E-4	7.267E-9
1e-4	11.310	10.498	-2.173	-2.368	7.310E-4	7.266E-8
0.001	11.297	10.495	1.1478	0.9527	7.297E-4	7.260E-7
0.002	11.285	10.492	2.1461	1.9510	7.282E-4	1.450E-6
0.005	11.253	10.482	3.4630	3.2679	7.239E-4	3.615E-6
0.01	11.207	10.465	4.4546	4.2595	7.166E-4	7.194E-6
0.02	11.129	10.433	5.4379	5.2428	7.022E-4	1.424E-5
0.05	10.943	10.340	6.7110	6.5159	6.599E-4	3.452E-5
0.1	10.714	10.203	7.6330	7.4379	5.922E-4	6.540E-5
0.187	10.431	10.009	8.4107	8.2156	4.831E-4	1.105E-4
0.3	10.184	9.8344	8.9426	8.7474	3.583E-4	1.526E-4
0.4	10.030	9.7367	9.2374	9.0423	2.632E-4	1.744E-4
0.7	9.7714	9.6468	9.7346	9.5394	6.580E-5	1.526E-4
0.95	9.7000	9.6849	9.9595	9.7644	1.828E-6	3.452E-5
0.99	9.6978	9.6950	9.9873	9.7922	7.312E-8	7.194E-6
1.0	9.6976	9.6976	$-\infty$	$-\infty$	0	0

So, this numeric exploration of the post-merge mutual information reveals ... something that perhaps should have been obvious, in retrospect. Sigh.

In retrospect, it should be clear that the union merge serves only to make word-sense disambiguation cloudier and mushier, by placing unrelated disjuncts into the same class/cluster. This seemed somewhat harmless at some conceptual level, by employing the argument that the observed disjuncts are rather noisy, and that perhaps there are commonalities that simply were not observed, due to insufficient sampling. This may still be true, but, viewed from the MI angle, it is clear that anything beyond a pure overlap merge is harmful to the entropy maximization.

Deriving the Entropy Extrema

The goal of this section is to find the extrema of the relative entropy as given in eqn 2. This requires a lot of algebraic calculation. The calculations below were undertaken before the numeric exploration immediately above. The numerics show that this effort was ... pointless. The entropy is maximized by doing the overlap merge, setting $f = 0$; it's it, that's that. The below is simply not necessary. It's kept below for, uhh, posterity. Otherwise, its useless and can be skipped over.

Merge parameterization

It's worth considering the most general merge that maintains detailed balance, and symmetry between the two merged words. This is done by defining a merge vector - a merge parameter for each disjunct, so that one gets a vector $0 \le f(d) \le 1$. The merged class is then

$$N(g,d) = \begin{cases} f(d)\left[N(a,d) + N(b,d)\right] & d \in D(a) \cap D(b) \\ f(d) N(a,d) & d \in D(a) \setminus D(b) \\ f(d) N(b,d) & d \in D(b) \setminus D(a) \end{cases}$$

The above tries to distinguish the overlapping regions. But it is overly complicated, since on the non-overlapping regions, the counts vanish. That is, $N(b,d) = 0$ when $d \in D(a) \setminus D(b)$. Thus, its enough to write

$$N(g,d) = f(d)\left[N(a,d) + N(b,d)\right]$$

To maintain detailed balance as before, one must have

$$N\left(a',d\right) = (1 - f(d)) N(a,d)$$

Note that the number of disjuncts is huge: in practice, a vocabulary of a few thousands words might have hundreds of thousands of disjuncts.[4] The vector $f(d)$ is a large vector.

In what follows, this full vector will be retreated from, going back to the original vision of a projection merge. There are two reasons for this:

[4]The ‘run-2‘ dataset used in the example above had 6029 distinct words and 98102 distinct disjuncts. This was a “trimmed” dataset, 40-8-5: all words with less than 40 observations were discarded. All disjuncts with less than 8 observations were discarded, and all (word,disjunct) pairs with less than 5 observations were discarded. I've been assuming that trimming filters out a lot of the noise in the dataset, without much compromising the data integrity. However, this has not been shown experimentally, it just seems like it should be so.

- The expressions get hopelessly non-linear and intractable,
- It is hard to imagine what sort of additional information might arrive, that would distinguish one disjunct from another. Either a disjunct is associated with a or with b or with both; there really aren't any other possibilities.

However, until that point, a vector of independent f's will be assumed.

Variational problem

A change of notation is in order. Parenthesis have been useful to emphasize what depends on what. Parenthesis are also convenient for plain-ASCII source code documentation. However, for the following, subscript notation will improve readability. Let

$$\begin{aligned} f_d &\equiv f(d) \\ N_{wd} &\equiv N(w,d) \\ M_{uw} &\equiv MI(u,w) \end{aligned}$$

The variational problem is then to obtain

$$\frac{\partial S}{\partial f_d}$$

I see no easy way out, other than to brute-force this. First, note that

$$\begin{aligned} \frac{\partial}{\partial f_d} \sum_{u,w=a,b} p(u,w) MI(u,w) &= \frac{\partial}{\partial f_d} \left[p(a,a) MI(a,a) + 2p(a,b) MI(a,b) + p(b,b) MI(b,b) \right] \\ &= 0 \end{aligned}$$

since none of these have $f(d)$ appearing in the expressions. Lets blast away.

$$\frac{\partial S}{\partial f_d} = \sum_{u,w=g,a',b'} M_{uw} \frac{\partial p_{uw}}{\partial f_d} + p_{uw} \frac{\partial M_{uw}}{\partial f_d}$$

So

$$\begin{aligned} \frac{\partial p_{gg}}{\partial f_d} &= \frac{1}{i_{**}} \frac{\partial i_{gg}}{\partial f_d} \\ &= \frac{1}{i_{**}} \frac{\partial}{\partial f_d} \sum_d N_{gd} N_{gd} \\ &= \frac{2}{i_{**}} f_d \left(N_{ad} + N_{bd} \right)^2 \end{aligned}$$

and

$$\begin{aligned}\frac{\partial p_{ga'}}{\partial f_d} =& \frac{1}{i_{**}}\frac{\partial}{\partial f_d}N_{gd}N_{a'd} \\ =& \frac{1}{i_{**}}(N_{ad}+N_{bd})N_{ad}\frac{\partial}{\partial f_d}f_d(1-f_d) \\ =& \frac{1}{i_{**}}(N_{ad}+N_{bd})N_{ad}(1-2f_d)\end{aligned}$$

and

$$\begin{aligned}\frac{\partial p_{a'a'}}{\partial f_d} =& \frac{1}{i_{**}}\frac{\partial}{\partial f_d}N_{a'd}N_{a'd} \\ =& \frac{N_{ad}^2}{i_{**}}\frac{\partial}{\partial f_d}(1-f_d)^2 \\ =& -2\frac{N_{ad}^2}{i_{**}}(1-f_d)\end{aligned}$$

and

$$\begin{aligned}\frac{\partial M_{uw}}{\partial f_d} =& \frac{\partial}{\partial f_d}\log_2\frac{p_{uw}}{p_u p_w} \\ =& \frac{1}{\log 2}\left[\frac{1}{p_{uw}}\frac{\partial p_{uw}}{\partial f_d} - \frac{1}{p_u}\frac{\partial p_u}{\partial f_d} - \frac{1}{p_w}\frac{\partial p_w}{\partial f_d}\right]\end{aligned}$$

so

$$\begin{aligned}\frac{\partial p_g}{\partial f_d} =& \frac{1}{i_{**}}\frac{\partial i_{g*}}{\partial f_d} \\ =& \frac{1}{i_{**}}\frac{\partial}{\partial f_d}\sum_d N_{gd}N_{*d} \\ =& \frac{1}{i_{**}}\frac{\partial}{\partial f_d}N_{gd}N_{*d} \\ =& \frac{1}{i_{**}}N_{*d}\left[N_{ad}+N_{bd}\right]\end{aligned}$$

and

$$\begin{aligned}\frac{\partial p_{a'}}{\partial f_d} =& \frac{1}{i_{**}}\frac{\partial i_{a'*}}{\partial f_d} \\ =& -\frac{1}{i_{**}}N_{*d}N_{ad}\end{aligned}$$

Putting all this together is painfully tedious. But it must be done. (Is there some easier way I don't see yet?)

$$
\begin{aligned}
\sum_{u,w=g,a',b'} p_{uw}\frac{\partial M_{uw}}{\partial f_d} =& p_{gg}\frac{\partial M_{gg}}{\partial f_d} + p_{a'a'}\frac{\partial M_{a'a'}}{\partial f_d} + p_{b'b'}\frac{\partial M_{b'b'}}{\partial f_d} \\
&+ 2p_{ga'}\frac{\partial M_{ga'}}{\partial f_d} + 2p_{gb'}\frac{\partial M_{gb'}}{\partial f_d} \\
=& \frac{1}{\log 2}\times \\
&\frac{2}{i_{**}} f_d\left(N_{ad}+N_{bd}\right)^2 - 2\frac{p_{gg}}{p_g}\cdot\frac{1}{i_{**}}N_{*d}\left(N_{ad}+N_{bd}\right) \\
&-2\frac{N_{ad}^2}{i_{**}}\left(1-f_d\right) - 2\frac{p_{a'a'}}{p_{a'}}\cdot\left(-\frac{1}{i_{**}}N_{*d}N_{ad}\right) \\
&-2\frac{N_{bd}^2}{i_{**}}\left(1-f_d\right) + 2\frac{p_{b'b'}}{p_{b'}}\cdot\left(\frac{1}{i_{**}}N_{*d}N_{bd}\right) \\
&+2\frac{1}{i_{**}}\left(N_{ad}+N_{bd}\right)N_{ad}\left(1-2f_d\right) + 2\frac{p_{ga'}}{p_{a'}}\cdot\frac{1}{i_{**}}N_{*d}N_{ad} + 2\frac{p_{ga'}}{p_{b'}}\cdot\frac{1}{i_{**}}N_{*d}N_{bd} \\
&+2\frac{1}{i_{**}}\left(N_{ad}+N_{bd}\right)N_{bd}\left(1-2f_d\right) + 2\frac{p_{gb'}}{p_{a'}}\cdot\frac{1}{i_{**}}N_{*d}N_{ad} + 2\frac{p_{gb'}}{p_{b'}}\cdot\frac{1}{i_{**}}N_{*d}N_{bd}
\end{aligned}
$$

Lets hope there are no mistakes. Gathering terms,

$$
\begin{aligned}
\frac{i_{**}}{2}\log 2\sum_{u,w=g,a',b'} p_{uw}\frac{\partial M_{uw}}{\partial f_d} =& f_d\left[\left(N_{ad}+N_{bd}\right)^2 + N_{ad}^2 + N_{bd}^2 - 2\left(N_{ad}+N_{bd}\right)^2\right] \\
&-\frac{p_{gg}}{p_g}N_{*d}\left(N_{ad}+N_{bd}\right) + \frac{p_{a'a'}}{p_{a'}}N_{*d}N_{ad} + \frac{p_{b'b'}}{p_{b'}}N_{*d}N_{bd} \\
&-N_{ad}^2 - N_{bd}^2 \\
&+\left(N_{ad}+N_{bd}\right)^2 \\
&+\frac{p_{ga'}}{p_{a'}}N_{*d}N_{ad} + \frac{p_{ga'}}{p_{b'}}N_{*d}N_{bd} \\
&+\frac{p_{gb'}}{p_{a'}}N_{*d}N_{ad} + \frac{p_{gb'}}{p_{b'}}N_{*d}N_{bd} \\
=& 2\left(1-f_d\right)N_{ad}N_{bd} \\
&+N_{*d}N_{ad}\left[-\frac{p_{gg}}{p_g} + \frac{p_{a'a'}+p_{ga'}+p_{gb'}}{p_{a'}}\right] \\
&+N_{*d}N_{bd}\left[-\frac{p_{gg}}{p_g} + \frac{p_{b'b'}+p_{ga'}+p_{gb'}}{p_{b'}}\right]
\end{aligned}
$$

Huf. This appears to be quadratic rational in f because

$$
\begin{aligned}
p_{gg} =& \frac{1}{i_{**}}\sum_d N_{gd}N_{gd} \\
=& \frac{1}{i_{**}}\sum_d f_d^2\left(N_{ad}+N_{bd}\right)^2
\end{aligned}
$$

while

$$\begin{aligned} p_g &= \frac{1}{i_{**}} \sum_d N_{gd} N_{*d} \\ &= \frac{1}{i_{**}} \sum_d f_d \left(N_{ad} + N_{bd}\right) N_{*d} \end{aligned}$$

So those fractions are nasty. There seem to be two choices in front of us. These are

- Set $f_d = 1$ for $d \in D(a) \cap D(b)$ and $f_d = f$ otherwise. This is the projection merge.
- Set $f_d = f$ for all d. This seems to be pointless; it smears together multiple word senses.

Abandon the second option. To disentangle the location of f in the various quantities, some additional notation is needed.

July 2021 - Bad Merge Redux

Variation on above theme. After merging the first two words with the overlap merge, the MI to additional words gets large, and so merges to those are proposed, too. This is presumably some bug. What's the bug? Attempt to find out, below.

The problem dataset is '`r2-mpg-trim-40-8-5.rdb`' (the same trimmed dataset as before, above.) Using the merge strategy '`(gram-classify-greedy-mifuzz star-obj 5.0 0 4)`' i.e. MI cutoff of 5.0, zero fraction and min obs count of 4, the following is printed:

(bad) merger MI		
be	have	5.5520
be-have	him	5.6404
be-have	do	7.8926
be-have	this	6.3487
be-have	all	7.0597
be-have	her	7.4523

The be-have merger is plausible. The remaining ones are not.

Lets look at the MI before merger. As before, load 'mi-tools.scm' to load scaffolding to make this easy. It is

Symmetric MI - csets+shapes							
	be	have	him	do	this	all	her
be	7.349	5.552	1.464	3.107	-0.29	0.684	1.316
have		6.922	0.010	2.953	-0.93	0.242	-0.24
him			2.201	0.830	0.181	0.500	1.961
do				5.242	0.431	0.723	0.683
this					6.245	1.223	1.011
all						4.421	0.573
her							2.942

The above is the MI for disjuncts plus shapes. It all looks pretty darned reasonable.

Lets check to see what's going on with the csets-only, without the shapes. This requires a batch-transpose recompute.

Symmetric MI - csets only							
	be	have	him	do	this	all	her
be	8.709	5.643	4.284	4.608	-1.70	3.119	2.909
have		8.267	1.772	4.655	-3.98	1.234	-0.14
him			7.538	3.334	1.737	3.240	6.082
do				8.306	-1.07	1.384	2.392
this					6.959	1.309	2.209
all						6.976	1.758
her							6.971

So that's .. a wash. The shapes really lowered the MI on him, her. Made them much much more distant. It lowered all the self-MI's. It lowered some cross-MI's and raised others, with no apparent overall effect.

How Shapes affect MI

The effect on him-her deserves further pondering. Without the shapes, it seems to be saying that him-her are being used in sentences the same way. But, when shapes are added, and the self-MI drops, this seems to be saying "hey, besides him/her, there are other words in this cluster, besides just him/her". That's what I interpret this as saying.

Lets take a detour, and see if that is true. Table below shows top MI between "him" and other words, ranked by cset-only-MI. A second column shows the corresponding cset-only-MI to "her", although this is no longer ranked. The third column shows the cset+shape-MI to "him".

	cset-only		cset+shape
	him	her	him
him	7.538	6.081	2.201
me	7.190	5.808	1.860
example	7.012	6.618	1.178
us	6.938	5.716	1.758
them	6.904	5.714	1.884
speak	6.901	6.049	1.061
instance	6.894	6.499	0.113
happen	6.876	6.055	0.800
difference	6.819	$-\infty$	0.653
themselves	6.794	6.020	1.784
himself	6.631	5.716	1.817
live	6.630	5.808	1.155
why	6.490	4.140	-1.044
listen	6.436	5.986	0.967
herself	6.406	5.557	1.730
behind	6.345	4.678	1.280
myself	6.082	5.292	1.559
her	6.081	6.970	1.961
try	5.945	5.495	0.635
say	5.911	4.965	1.135

It's OK, but a bit underwhelming. First, looking at the first column. It seem OK that other pronouns are in there. The presence of a handful of verbs, nouns and prepositions is not confidence-inspiring. One might have hoped that these would not be there. The second column is OK, I guess... although "her" is also a possessive, and "him" is not.

The third column is peculiar. It looks at the cset+shape MI. It's a *lot* lower. I'm surprised by this; I was expecting the MI to be improved. There's only a hint of a silver lining: the MI to verbs, nouns and preps seems just a little lower, than it is to pronouns. But just a smidgen. It's not much. Overall, we conclude that the MI values fail to distinguish between pronouns, verbs, nouns and preps.[5]

July 2021 - Vector Sizes and MI

Above, we see that, for a given dataset, the MI values (seem to) fail to distinguish between pronouns, verbs, nouns and prepositions. This section explores why that is the case. The results so far show that the MI between a large word-vector and a small

[5]So this is intermixed with a lot of confusion. The MM^T marginals (*i.e.* the values of $\sum_d N(w,d)N(*,d)$) were originally from the large dataset, kept as a cache. Then the dataset was trimmed, without recomputing marginals. Then I caught the error. Then I discovered that the $N(*,d)$ values were cached, too. So, several different mistakes – in some cases, the sum ran over the set of disjuncts d for the full set, and for some it did not. The dot-product $\sum_d N(w,d)N(u,d)$ used in computing the MI always runs only over the disjuncts d in the trimmed dataset. This is obviously not compatible with the cached marginals.

word-vector can be quite high, if most of the small ("short") vector lies in the same direction as the large ("long") vector.[6]

This is...well, its good and bad, depending on what one wants. If the goal is to take a short fragment and identify where it belongs, then this is unalloyed good. However, if the dataset is small, and almost all words have short vectors, then this is bad. The natural Zipfian distribution indicates that only a small fraction (ballpark of 10%) have long vectors. Thus, it is not surprising that most short vectors will mostly align with one of these long vectors.

There are several conclusions one can make:

- Don't use small datasets with lots of short vectors. This is hard, because (1) the Zipfian distribution means most vectors will be short, no matter the size of the dataset. (2) it is CPU intensive to use large datasets.
- Don't run merge algos on short vectors. This is already done, more or less: the current merge algo considers merging only vectors of similar counts (and thus, presumably of similar sizes.) Stop merging long before reaching down into the short-vector regions of the dataset.
- Don't trim datasets. The goal of trimming was to discard "noise": i.e. infrequently-observed word-disjunct pairs, with the general argument that such low-count pairs are likely to be garbage. However, we have not shown that they actually are "garbage", and such low-count trimming might actually be discarding important information. That is, trimming might be creating short vectors that incorrectly align with large vectors, when the untrimmed variants would have shown that they are quite different.
- Use a different measure of MI that emphasizes non-colinearity.

Terrible MI values

Here's the disaster were looking at. The top-MI cset+shape words to "him" are these:[7]

[6]The word "length" is troublesome. In this paragraph, we are talking about "the size of the support" *i.e.* the number of non-zero entries in the vector. This should not be confused with the root-mean-square length of the vector, *i.e.* its Euclidean length. This confusion easily extends to "short" vs. "small" and "long" vs. "large". Hopefully, things will be clear from context.

[7]Use '(almi "him")' from the tools file.

	him
sink	4.452
fill	4.223
Instead	4.181
grow	3.613
1880	3.529
scoundrel	3.423
accepted	3.186
serve	3.179
appealed	3.018
springing	3.018
...14 more ...	
him	2.201

Wow. That's ugly. Bad. Wrong. None of the skipped 14 words are pronouns. The self-MI for "him" is 25th on the list. Why isn't the self-MI always the highest? How should this be understood?

The word "her" is no better:

	her
outstretched	5.855
flashing	4.946
appearances	4.710
Catherine	4.682
hybrids	4.307
jail	4.058
sink	3.909
his	3.903
fill	3.680
God's	3.462
... 7 more ...	
her	2.878

Here, the self-MI for "her" is 17th in the list. At least the possessive "his" appears 8th.

Close-up inspection

So what is going on here? Lets look close up.[8] The word "him" has 404 Sections and 430 CrossSections. A random sample of the CrossSections looks mighty healthy:

[8]After the analysis in this section was performed, a procedural issue was detected. It's not clear how this affects results. The procedural problem is that the MM^T marginals were computed first, and then the dataset was trimmed. For computing the MI, the MM^T marginals from the full dataset were used. I'm thinking that they should have been recomputed after trimming.

```
at: (gazing- & him+) or (gazing- & the+)
upon: (came- & him+) or (came- &  a+)
for: impossible- & (him+ or me+ or any+ or them+)
do: to- & (him+ or her+ or it anything in a something them what ?  so with as the that this . , )
```

The Sections look good too:

```
him: (assure- & that+) or (put- & in+) or (Let- & go+) or (turned- &out+)
```

How about the word "sink"? Wow, it is .. bad. It has one cross-section, and zero sections:[9]

```
to: seemed- & sink+
```

The total observation count is 9, which is small. Just above the dataset cutoff. The self-MI is MI(sink, sink)=12.27. The cross section is "seemed to xxx" where xxx can be any of "like them Pierre grow her have go consider see be take feel say think him make come sink fill me". Count on "seemed to him" is 151. Count on the product is $\sum_d N(sink,d)N(him,d) = 9 \times 151 = 1359$. So that's perfect. $\sum_d N(sink,d)N(*,d) =$ 10215 Since there is one d, we have $\sum_d N(*,d) = 1135$ which was verified manually. The other marginal is $\sum_d N(him,d)N(*,d) = 41576753$ which was not verified. The total count is 6842054908. Multiplying out:

$$\begin{aligned}
\log_2 \frac{\sum_d N(sink,d)N(him,d)}{\sum_d N(*,d)N(*,d)} &= \log_2 \frac{1359}{6842054908} = -22.263 \\
\log_2 \frac{\sum_d N(sink,d)N(*,d)}{\sum_d N(*,d)N(*,d)} &= \log_2 \frac{10215}{6842054908} = -19.353 \\
\log_2 \frac{\sum_d N(him,d)N(*,d)}{\sum_d N(*,d)N(*,d)} &= \log_2 \frac{41576753}{6842054908} = -7.362
\end{aligned}$$

and so

$$MI(sink,him) = -22.263 + 19.353 + 7.362 = 4.452$$

So the numerics verify. What does this teach us? A word with a single cross-section can give us absurd MI values.

Lets compare this to the cosine distance. For that, we need:

$$\begin{aligned}
\sum_d N(sink,d)N(him,d) &= 1359 \\
\sum_d N(sink,d)N(sink,d) &= 81 \\
\sum_d N(him,d)N(him,d) &= 1161813
\end{aligned}$$

[9]It has zero sections because these were filtered out. The filter removes all sections with a count of less than five. It seems that the word "sink" only appeared in a small handful of sentences. This makes it a rare word.

and the cosine would be

$$\begin{aligned}\cos\theta &= \frac{\sum_d N(sink,d)N(him,d)}{\sqrt{\left(\sum_d N(sink,d)N(sink,d)\right)\left(\sum_d N(him,d)N(him,d)\right)}}\\ &= \frac{1359}{\sqrt{81\times 1161813}}\\ &= 0.1401\end{aligned}$$

which is a far more reasonable value. Why is the MI misleading us? What went wrong here? What can we do to fix it?

What's the problem?

Well, the primary problem with the above is that .. the MI is correct, but unwanted. We have a word with a tiny sample - one disjunct, and that one disjunct overlaps nicely with anything and everything else that has that disjunct in it.

To with: the cross-section came from the phrase "seemed to xxx". It's plausible to assume that $MI(sink, xxx)$ will be high for all other words xxx appearing in that phrase. Indeed this is the case:

	MI	support
sink	12.37	1
fill	12.15	2
grow	11.51	3
consider	9.345	5
be	8.302	1040
have	6.931	863
feel	6.602	33
take	6.391	122
make	5.689	135
come	4.966	235

	MI	support
think	4.801	221
say	4.786	257
him	4.452	834
me	4.206	612
her	3.909	810
like	3.852	171
Pierre	3.611	31
go	3.555	231
see	2.984	329
them	1.617	372

The column labeled "support" is the length of the vector for that word (*i.e.* it is the number of disjuncts observed for that word.) It appears to show that the length of the vector has little impact on the MI. That is, it is the word "sink" and it's single disjunct that is in the drivers seat: it can overlap some very long vectors in such a way that a high MI results.

What is meant by "high MI"? (Computing the average MI is impossible, as the CPU demand to sum over all pairs is simply too high. We need to develop techniques for estimating this from the distribution.)

The MI is high, because this word "sink" has nothing else it can match to. That is, the vector for "sink" is so short, that it matches anything else that has that same vector element in it. So, in that sense, the result is correct; it is just, perhaps, not quite what we wanted.

What do we want, then?

Double check

The above focuses on the word “sink”. Perhaps its exceptional. How about the other problematic words: “fill Instead grow 1880 scoundrel accepted serve appealed springing”? Let’s take a quick peak.

The word “fill” shows up in two cross-sections: “to: seemed- & fill+” and “up: fill- & the+”, both with low counts (9 and 10) and no Sections.

The word “Instead” appears in one Section: “Instead: WALL- & ,+ & he+” and one cross-section: “,: Instead- & he+” with counts of 6 and 14.

The word “grow” appears in three cross sections: “to: seemed- & grow+” and “up: grow- & .+” and “to: began- & grow+” with counts of 18, 6 and 25. There are no Sections.

The word “1880” appears in one Section: “1880: in- & ,+” with a count of 10.

The word “scoundrel” appears in two cross-sections: “a: am- & scoundrel+” and “,: scoundrel- & but+”.

The word “accepted” appears in a number of Sections and cross-sections. Most of these appear to be the result of parsing a sentence in the Project Gutenberg boilerplate, something about accepting donations.

The word “serve” appears in one cross-section: “as: serve- & a+” with a count of 32.

We can safely conclude that “grow” is prototypical of the other problematic words. All of the problematic words are those with very short vectors.

As noted before, the lack of Sections is due to this being a filtered dataset: Sections with a count of 5 or less have been removed. Effectively, all of the above words are rare, and occur only in a handful of sentences in the text.

This reinforces an apparent paradox: the corpus seems to be quite large: maybe a quarter of a million sentences! (The exact count is not known, this is a damaged dataset.) Even after filtering, there seem to be a lot of words (just over six thousand!) Yet, there’s an average of 23 observations per word, which is not all that many. This should be compared to mother-baby talk in human infants: by the age of two, a baby will have heard a small vocabulary of hundreds of words, but with a sample of a hundred-thousand or a million for each of these words.

The size of the datasets used here is deceiving. Also: one can presume that almost all sentences in the Project Gutenberg corpus are distinct, with very few repeated sentences. By contrast, in human mother-baby talk, there is a vast amount of repetition. Now, most of this repetition is for auditory training, not syntactic training, but still, the repetitiveness must surely be quite reinforcing.

Other kinds of mutual information

The above suggests that perhaps other measures should be examined. Intuitively, one might want a measure that treats vectors of similar size as being more similar, than those of dis-similar sizes.

(Conclusion for this section: (1) the conventional MI seems to work best, and (2) never got around to making a size-sensitive distance in this particular section.)

Since we're discarding cosine distance for *a priori* reasons (probabilities do not live in Euclidean spaces), then only other MI-like constructions are allowed. One obvious one is a Banach-like MI_s measure:

$$\begin{aligned} MI_{s,t}(w,u) &= \log_2 \frac{\left(\sum_d N^s(w,d)N^s(u,d)\right)\left(\sum_d N^t(*,d)N^t(*,d)\right)}{\left(\sum_d N^s(w,d)N^t(*,d)\right)\left(\sum_d N^s(u,d)N^t(*,d)\right)} \\ &= \log_2 \frac{\left(\sum_d p^s(w,d)p^s(u,d)\right)\left(\sum_d p^t(*,d)p^t(*,d)\right)}{\left(\sum_d p^s(w,d)p^t(*,d)\right)\left(\sum_d p^s(u,d)p^t(*,d)\right)} \end{aligned}$$

where $p(w,d) \equiv N(w,d)/N(*,*)$ or where $p(w,d) \equiv N(w,d)/N(w,*)$ as desired; the normalization cancels out. The two exponents s and t are independent of one-another. it's not entirely clear if they should be forced to be equal, *i.e.* to set $s=t$ or if it is better to set $t=1$ and allowing s to remain free. Even setting $t=0$ is plausible. Setting $s=t=1$ gives the earlier MI. Setting $s=t=0$ gives a Hamming-distance-like MI that only considers those disjuncts that have a non-zero count. Setting $s=t=2$ results in a weighting where the high-count disjuncts dominate, and thus naturally suppresses "noise": the word-disjunct pairs with low observation counts. This assumes that low-count word-disjunct pairs actually are "noise". This has *not* been experimentally established. In fact, one can go in the opposite direction: perhaps these carry some kind of important signal. In this case, a weighting of $s=t=1/2$ may provide a good intermediate balance between $s=t=0$ and $s=t=1$. There's a strange intellectual appeal to an exponent of 1/2, in that it is "quantum mechanical": it is the square root of the probability. That this is meaningful in a statistical realm is demonstrated by the Fisher information metric, which reduces to the flat-space Euclidean metric when working with the square-root. This is "interesting", the meaning of which is not yet entirely clear to this author. (Asking about complex values of s seems like a step too far, at this time.)

How might this work out for "sink"?

	$D(w,*)$	MI_0	$MI_{0.5}$	MI_1	MI_2
him	834	5.657	28.954	2.201	-13.495
me	612	4.382	27.300	1.860	-13.661
example	12	-0.056	19.292	1.179	-8.622
us	153	3.250	25.031	1.759	-11.774
them	372	4.235	26.589	1.884	-12.781
speak	58	1.271	21.562	1.061	-10.114
instance	19	2.295	20.724	0.114	-13.463
happen	17	0.605	19.379	0.800	-9.381
difference	20	-0.174	18.974	0.653	-8.862
themselves	17	0.683	21.598	1.785	-9.302
himself	95	2.671	24.065	1.817	-10.951
say	257	1.709	23.289	1.136	-11.831

The above table shows four different kinds of MI between "him" and the other words shown. The $D(w,*)$ is the support for that word vector. Recall the definition $D(w,d)=1$ if $N(w,d)>0$ and so $D(w,*)$ is just the sum – the size of the vector.

All of these seem to discriminate, but weakly. That is, its plausible to have a high MI between "him me us them themselves himself" and a lower MI to "example speak instance happen difference say" and all of these do this. The discrimination is imperfect. For MI_0 "themselves" is lower than "instance speak say", so these two clusters overlap. That's not good. For MI_1 the lowest pronoun is "us" at 1.76 while the highest other word is "example" at 1.18, so these two clusters are separated. However, the information content is not large: MI is measured in bits, there is less than one bit difference. This is not big enough to get the clustering to work well, as currently designed/implemented.

The huge values for $MI_{0.5}$ are surprising. It doesn't seem to be a great discriminator: the width of the clusters is about the same as the distance between them. The lowest pronoun for $MI_{0.5}$ is "themselves" at 21.60; the highest non-pronoun is "speak" at 21.56. So this seems weaker than the usual MI for discrimination.

The negative values for M_2 are surprising. Not sure what to make of that.

Table below shows cluster statistics. The pronoun cluster is "him me us them themselves himself" the "other" cluster is "example speak instance happen difference say".

	pronouns	other
min MI_0	0.6833	-0.174
max MI_0	5.6577	2.2954
avg MI_0	3.4799	0.9418
rms MI_0	1.5628	0.9021
min $MI_{0.5}$	21.598	18.974
max $MI_{0.5}$	28.955	23.289
avg $MI_{0.5}$	25.590	20.537
rms $MI_{0.5}$	2.3741	1.5272
min MI_1	1.7585	0.1135
max MI_1	2.2012	1.1789
avg MI_1	1.8844	0.8238
rms MI_1	0.1479	0.3683
min MI_2	-13.66	-13.46
max MI_2	-9.302	-8.622
avg MI_2	-11.99	-10.38
rms MI_2	1.5296	1.7358

These are small, eyeballed clusters, but they provide a guide to discrimination. Consider the difference between the averages in the two clusters, and the sum of the rms of the two, and the ratio of these:

	distance	sum rms	stddev
MI_0	2.5381	2.4649	1.03
$MI_{0.5}$	5.053	3.9013	1.30
MI_1	1.0606	0.5162	2.05
MI_2	1.610	3.2654	0.49

Thus we conclude – for this small, unbalanced, *ad hoc* sample, the conventional MI provides the greatest amount of discrimination.

Length Matching

Above section has no considerations for negative matching – mismatches due to size differences. That is, the foundation-stone is built on

$$\sum_d N(w,d) N(u,d)$$

where, by definition, the sum over the disjuncts d run over the set where both counts N are non-zero. This suggests that perhaps, we could use a measure that includes negative matching. This suggests using an MI inspired by the Jaccard distance (aka Tanimoto distance).

Is this really needed? Maybe not – and here's why. Merging words into clusters (minimally) means that only the disjuncts in the intersection are merged. The intent of this minimalist merge is to split up words having multiple meanings, associating a collection of disjuncts with a particular meaning. But what this also implies is that having a similarity measure that is likewise tuned to being able to pick out common subsets is a good thing.

That said, it is worth exploring some alternate similarity measures in greater detail.

Flavors of Jaccard similarity

There are multiple variants of Jaccard similarity that apply to the current situation. This leads to a bit of a naming crisis.

Ruzicka similarity

Given some arbitrary function $f(w,d)$ of word-disjunct pairs (w,d), the Ruzicka similarity for this problem is

$$J_R(f;u,w) = \frac{\sum_d \min(f(w,d), f(u,d))}{\sum_d \max(f(w,d), f(u,d))}$$

The Jaccard similarity corresponds to the case where $f(w,d)$ is either zero or one (is the indicator function: is or is not in the set) so the Ruzicka similarity is a kind of weighted Jaccard.

The corresponding distance is

$$J(f;u,w) = 1 - J_R(f;u,w)$$

Wikipedia suggests that this obeys the triangle inequality, i.e. is a true metric, but this is not obvious to me.

There are several plausible functions from which f can be built. One is the raw observation count $N(w,d)$ and the other is the conditional probability of observing a particular word-disjunct pair, for that word:

$$p(d|w) = \frac{N(w,d)}{N(w,*)}$$

This is nicer than just using the plain counts, and here's why. Consider first:

$$J_R(N;u,w) = \frac{\sum_d \min(N(w,d), N(u,d))}{\sum_d \max(N(w,d), N(u,d))}$$

If word u is a (perfect) synonym for w but is used only rarely, then the comparison of the raw counts will lead to mostly-garbage. That is, if $N(u,d) = \alpha N(w,d)$ for some constant α, these two vectors would come out as dis-similar, even though it is clear that, for this case, we want them to be considered to be identical, as they are synonyms. Thus, using raw N is just wrong, for the present case. By contrast, using the conditional probability appears to provide the desired result: the two synonyms would be judged to have perfect similarity.

The simplest, narrowest, original definition of the Jaccard similarity is the unweighted similarity given by $f(w,d) = [N(w,d) > 0]$. This just uses the support of N to determine set membership.

The ‘add-similarity-compute‘ object provides three Jaccard methods:

- ‘jaccard‘ which implements $J_R(N;u,w)$
- ‘cond-jacc‘ which implements $J_R(p;u,w)$
- ‘overlap‘ which implements $J_R(N > 0;u,w)$

Probability Jaccard

The “probability Jaccard” provides “maximally consistent sampling” per WP ref. It is defined as

$$J_P(u,w) = \sum_d \left[\sum_c \max\left(\frac{N(w,c)}{N(w,d)}, \frac{N(u,c)}{N(u,d)} \right) \right]^{-1}$$

Note that

$$1 - J_P(u,w)$$

is a *bona fide* metric over the probability space.

How might this work for “sink” with one disjunct, and “him”, with 832 disjuncts on it? The sum over d must necessarily run over just this one disjunct. The sum over c runs over all 832 of them. For 831 of them, $N(sink,c) = 0$. Thus,

$$J_P(sink, him) = \left[1 + \sum_{c \neq d} \frac{N(him,c)}{N(him,d)} \right]^{-1}$$

and clearly, this will judge these two words to be very different.

Entropic Similarity

The notation for entropy and MI being used here are susceptible to confusion, when being compared to the contents of Wikipedia. To avoid confusion, a review is worthwhile. This was already presented in part one of the diary, section titled “Entropic Similarity”, but is repeated here.

Given a count $N(w,d)$ define the *joint probability*

$$p_M(w,d) = \frac{N(w,d)}{N(*,*)}$$

The subscript M is a reminder that these counts are obtained from MST/MPG parsing (maximum spanning tree/maximum planar graph parsing).

The *conditional probability* of observing a word w given a fixed disjunct d is then

$$p_M(w|d) = \frac{p_M(w,d)}{p_M(*,d)}$$

That is, the conditional probability is just the fraction of all pairs having this disjunct.

The *marginal probability* of observing a word w is just the sum of observing it for any disjunct:

$$p_M(w) \equiv p_M(w,*) = \sum_d p_M(w,d)$$

The above is all fine and everything, but inappropriate for comparing two words. The inner product between two words w,u is

$$i(w,u) = \sum_d N(w,d)\,N(u,d)$$

and can be written as a proper (joint) probability as

$$p(w,u) = \frac{i(w,u)}{i(*,*)} = \sum_d p_M(w,d)\,p_M(u,d)$$

Note the absence of a subscript M; this is a different probability than above.

The marginal probability is then

$$p(w) \equiv p(w,*) = \sum_d p_M(w,d)\,p_M(*,d)$$

The fractional MI between two words is then

$$MI(w,u) = \log_2 \frac{p(w,u)}{p(w)\,p(u)}$$

and so is now manifestly compatible with Wikipedia.

This has the desirable property of giving the same score to synonyms. That is, if $N(u,d) = \alpha N(w,d)$ for some constant α, then the two words u and w should be judged as perfect synonyms, and so should be perfectly similar. Plugging through, we find that the MI obeys this condition:

$$MI(u,u) = MI(w,w) = MI(u,w)$$

when $N(u,d) = \alpha N(w,d)$ holds.

The (fractional) joint entropy is

$$H(w,u) = -\log_2 p(u,w)$$

Note the minus sign.

Wikipedia then defines the "variation of information" $d(u,w)$ as

$$\begin{aligned} d(u,w) &= H(u,w) - MI(u,w) \\ &= \log_2 p(w)\,p(u) - 2\log_2 p(u,w) \end{aligned}$$

It then claims that

$$0 \leq \sum_{w,u} p(w,u)\,d(w,u) \leq \sum_{w,u} p(w,u)\log_2 p(w,u)$$

which is not manifestly obvious to me, but I guess it's plausible. This suggests the "normalized metric"

$$D(w,u) = \frac{d(w,u)}{H(w,u)} = 1 - \frac{MI(w,u)}{H(w,u)}$$

Is this really always positive? If so, its not superficially obvious to me. This is the fractional part of the "Jaccard distance", arrived at when given the appropriate measure-theoretic interpretation. There is also a different "metric":

$$D'(w,u) = 1 - \frac{MI(w,u)}{\max(H(w),H(u))}$$

Jaccard MI

This section defines a kind-of "Jaccard mutual information", in a kind of *ad hoc* manner, by inspiration. It's not particularly well-founded; rather, its plausible. Except that it's not – it's broken.

Define the support of a vector as

$$S(w) = \{d : N(w,d) > 0\}$$

so that the Jaccard union is

$$S(w) \cup S(u)$$

How about the rest, then? Well, lets just try it out. Abusing the notation for conditional probability, write

$$\begin{aligned} p(w;u,d) &= \frac{N(w,d)}{\sum_d N(w,d) + N(u,d)} \\ &= \frac{N(w,d)}{N(w,*) + N(u,*)} \end{aligned}$$

Then define a Jaccard MI as

$$MI_J(w,u) = \log_2 \frac{\left(\sum_d p(w;u,d)p(u;w,d)\right)\left(\sum_d p(*;u,d)p(*;w,d)\right)}{\left(\sum_d p(w;u,d)p(*;u,d)\right)\left(\sum_d p(u;w,d)p(*;w,d)\right)}$$

Plugging through, this reduces to

$$MI_J(w,u) = \log_2 \frac{\left(\sum_d N(w,d)N(u,d)\right)\left(\sum_d p(*;u,d)p(*;w,d)\right)}{\left(\sum_d N(w,d)p(*;u,d)\right)\left(\sum_d N(u,d)p(*;w,d)\right)}$$

where

$$p(*;w,d) = \sum_x \frac{N(x,d)}{N(x,*)+N(w,*)}$$

which is computationally awkward.

Upshot is: the conventional dot product is not altered. The computation of this Jaccard MI is far more complex and time consuming. Curiously, the obvious attempt to simplify/approximate it just takes us back to the conventional MI.

The primary problem here is that this violates one of the desired properties for the similarity of synonyms. That is, if $N(u,d) = \alpha N(w,d)$ for some constant α, then the two words u and w should be judged as perfect synonyms, and the resulting similarity should be perfect. This distance violates that.

July 2021 - Similarity Smackdown

So, above, we have a whole variety of different similarities. Lets see how they perform on real data. This test is to be done on several datasets: ‘run-1-en_mpg-tranche-123.rdb‘ and trimmed versions of this.[10] Its our midsize English dataset. It’s pretty big. The comparison will be between the top ten most frequent common nouns, the top ten verbs, the top ten adjectives, the top ten pronouns, the top ten prepositions and the top ten adverbs. The min, max, mean and rms similarity will be computed in-group, for each group. This will be between the 10*9/2=45 pairs in each group. Next, the min, max, avg and rms similarity between groups will be computed; this will run over 10*10=100 between-group pairs. This will be repeated for the trimmed dataset, too. (Maybe several trims?) Should probably do this with and without shapes. This is a pretty massive experiment, with a whole lot of compute going into it. This is going to take a while, and spew a lot of data.

Lets go. The word categories (these are hand-picked):

[10]The ‘run-1-en_mpg-tranche-123.rdb‘ dataset contains 113M atoms. Of these, 391548 (391K) are WordNodes

POS	Difficult set
determiner	the a that this an some any these those most -
preposition	of to in with for at on by from up -
verb-ir	was is had be have were are would been will -
verb-act	said like do did know made see go came come -
pronoun	I he it you him she they me them we -
possessive	his her my their your its our ones +
adverb	that as not no there about only then very just -
conjunction	and but or so if than both either yet nor -
adjective	all one other little even good great long old many -
WH-words	which who when what where why how while whatever whose -
noun	time man way eyes day hand head men face place -
aux	can should could may must might shall cannot am has -

Note that some of these are ambiguous: "as" can be an adverb or preposition. "that" can be an adverb or a determiner. As such, they might be excessively broad categories, and might overlap a fair bit.

The table below defines an "easier" set, that is, hand-picked, because they seem easier to me.

POS	Easier set
determiner	the a this an some any these those all most -
preposition	in with for at on by from up out into -
verb-ir	was is be have were are been has having being -
verb-act	like do know see go come make say think want -
verb-past	said did thought looked found took asked saw told knew -
pronoun	I he it you him she they me them we -
possessive	his her my their your its our ones +
adverb	not no there about only then very just rather often -
conjunction	and but or so if than both either yet nor -
adjective	one little good great long old many young black hard -
WH-words	which who when what where why how while whatever whose -
comparative	better longer greater lower higher closer earlier larger older younger -
noun	man eyes day hand head men room house night door -
aux	had can could should would will may must might shall -
given	Jack Ross God John Jim Demelza Mai Richard George Mary -
places	England London Dallas Paris America Rome Canada Italy Ireland China -
titles	Mr Mrs Miss King St Sir Dr Captain General Madame -

A better selection technique might be to start with the "difficult set" and kick out the two outliers (according to conventional MI) and obtain an easier set that way.

Anyway, the experiments below were all performed with the difficult set.

Summary of Results

The sections below provide detailed results. To summarize, these are:

- MI beats the pants off of cosine similarity; of all the distance measures, cosine similarity is in a distant last place.
- Surprisingly, three different metrics provide superior results to MI: overlap similarity (unweighted Jaccard similarity), weighted Jaccard similarity and “probability” Jaccard similarity, in order of quality of results. These improve the inter-intra-cluster separation by over 10% compared to using MI. (So - 10% is not a lot, but we’ll take what we can.)
- This last result is ... surprising, and seems to contradict the naive belief in information theory. How is this to be explained? Is there some deeper information-theoretic explanation that has not been grasped?
- The experiment fails to measure the thing that originally inspired it: There are word pairs with very high MI but very low overlap similarity, simply because one of the word vectors is very short. The hand-picked clusters above contain no short vectors!
- Surprisingly, the use of shapes makes the MI scores considerably worse. It’s possible, but not likely, that this is due to dataset filtering problems. That experiment still needs to be performed. Naive imagination suggests shapes should have made everything better.
- Shapes do seem to make things better, when using the various forms of the Jaccard distance. This is especially the case for the smaller/smallest datasets.

See also a different summary of results at the bottom, and another at the top.

Trimmed dataset (Poorly trimmed)

Above is to be run on the following datasets. Attention: The table immediately below summarizes datasets generated with a poorly designed filtering operation. The problem is that the ‘add-zero-filter‘ was not run, resulting in words in Connectors that do not appear in Sections. This not only alters product computations, but it also leads to cross-sections whose head word doesn’t appear in a section. This turns out to skew results, but not all that much - mostly less than a few percent. So these datasets appear to be “pretty good”, but not entirely fully self-consistent. See further below for the revised results. Funny thing: those revised results are also incompletely self-consistent. They’re better, but it turns out a single pass of ‘add-zero-filter‘ is not enough. A recursive multi-pass will be required.

file name	total pairs	words	disjuncts	sparsity	entropy
r3-filt-1280-256-160-marg.rdb	78	28	70	4.65	-17.06
r3-filt-640-128-80-marg.rdb	6035	434	4238	8.25	-2.83
r3-filt-320-64-40-marg.rdb	15733	810	10808	9.120	0.4877
r3-filt-160-32-20-marg.rdb	40357	1687	27298	10.156	3.5924
r3-filt-80-16-10-marg.rdb	93758	2995	61359	10.937	6.6372
r3-filt-40-8-5-marg.rdb	213735	5422	137120	11.764	9.6489
r3-filt-20-4-2-marg.rdb	546345	10770	325604	11.903	13.326
r3-mpg-marg.rdb	28436901	377553	25698949	18.380	24.100
filt-640-128-80-shape	18507	1294	10737	9.552	0.7650
filt-320-64-40-shape	48818	2674	26981	10.529	4.7756
filt-160-32-20-shape	129247	5180	71280	11.480	8.5478
filt-80-16-10-shape	301284	9517	157543	12.281	12.164
filt-40-8-5-shape	693785	16668	351734	13.045	15.560
filt-20-4-2-shape	1794658	29080	876922	13.795	11.747
mpg-shape					

The first two datasets do not have all of the words for all of the clusters. The "marg" datasets use the Sections (pseudo-csets) only; the "shape" datasets include both Sections and Cross-Sections. The number of words differ between the corresponding datasets, because the filtering cut things in such a way that there are words appearing in CrossSections that don't have a matching Section (because that Section was cut by the filter). That is, there are words in connectors that don't appear in Sections.

XXX The above should not have happened; 'add-linkage-filter' should have prevented the above. Is there a bug? What's up with that? Answer: above failed to run 'add-zero-filter' after the row-column filter, but before 'add-linkage-filter'. The problem was that 'add-linkage-filter' was given a list of words that had zero counts, because the generic filter does not knock out rows whose counts are empty. All this is recomputed further below.

r3-filt-640-128-80-marg.rdb – MI (for the poorly-filtered dataset)

So lets try it out. Attention: this is done with the poorly-filtered datasets above, but apparently can still be taken to be "representative". Better filtering changes results by only a few percent, or less.

The r3-filt-640-128-80-marg.rdb dataset is the first dataset big enough to measure. The stats are for the MI. The cluster stats are:

cluster	sz	pr	min pair	min	max pair	max	avg	rms
wl-1-det	10	18	the . most	$-\infty$	this . any	5.646	3.437	1.581
wl-2-prep	10	38	of . up	$-\infty$	by . from	4.595	2.777	1.252
wl-3-verb-ir	10	21	was . be	$-\infty$	would . will	6.840	3.817	2.319
wl-4-verb-act	7	7	said . like	$-\infty$	do . did	8.289	5.482	1.782
wl-5-pronoun	10	30	I . him	$-\infty$	him . me	7.807	3.889	1.843
wl-6-posessive	3	3	her . my	5.60	his . her	6.934	6.238	0.546
wl-7-adverb	10	12	that . about	$-\infty$	not . only	5.960	1.917	2.487
wl-8-conj	8	10	and . than	$-\infty$	or . so	2.410	2.055	0.357
wl-9-adj	10	7	all . one	$-\infty$	good . great	9.674	5.514	2.772
wl-10-wh	5	5	who . when	$-\infty$	which . what	4.744	3.121	1.292
wl-11-noun	10	36	time . man	$-\infty$	eyes . hand	5.222	4.554	0.180
wl-12-aux	9	21	can . shall	$-\infty$	should . shall	8.342	6.771	0.692

This dataset is small enough that not all of the words in the cluster appear. The ‘sz‘ column counts how many of those words are present. The $-\infty$ in the ‘min‘ column shows that this dataset has word-pairs with zero overlap. Yow! The ‘pr‘ column counts how many good pairs there were (there should be 49=10x9/2, if they were all comparable). The average and rms exclude such pairs.

Conclusion: this dataset is not big enough to sample many common English words. It is big enough to classify at least some words into clusters, though.

How about between clusters? Worth a quick look.

clusters	prs	max pair	max	avg	rms
1-det 2-prep	15	that . for	2.220	-1.90	0.944
1-det 5-pronoun	9	this . it	3.565	-0.08	2.279
1-det 7-adverb	20	any . no	6.980	1.671	2.267
1-det 9-adj	11	some . many	9.918	3.179	2.899
3-verb-ir 4-verb-act	31	been . come	9.886	4.036	2.021
3-verb-ir 5-pronoun	10	be . him	4.434	1.306	1.871
4-verb-act 5-pronoun	8	see . him	4.364	2.704	1.654
4-verb-act 11-noun	1	come . time	7.095	7.095	0.0
5-pronoun 6-posessive	5	him . her	6.451	4.739	2.189
7-adverb 8-conj	28	just . so	3.770	1.134	1.534
7-adverb 9-adj	13	only . other	4.478	2.319	1.504
8-conj 9-adj	11	either . one	8.629	1.960	3.318

There are 66=12x11/2 of these cluster pairs. The above samples just some of them. it seems to show some problems with these hand-picked clusters: they overlap more than imagined, and not surprisingly so. The only real surprise here is the “see-him” pair.

r3-filt-40-8-5-marg.rdb – MI (for the poorly-filtered dataset)

A much bigger dataset than the above. Attention: this is done with the poorly-filtered dataset described above. The results seem to still be "representative"; better filtering changes them by a few percent, sometimes less.

The stats are for the MI. The cluster stats are:

cluster	sz	pr	min pair	min	max pair	max	avg	rms
wl-1-det	10	44	an . these	$-\infty$	these . those	8.501	2.095	2.900
wl-2-prep	10	45	of . up	-1.39	by . from	4.238	2.586	1.330
wl-3-verb-ir	10	43	would . been	$-\infty$	were . are	6.348	2.013	2.476
wl-4-verb-act	10	45	see . came	-3.31	go . come	7.579	2.917	1.976
wl-5-pronoun	10	45	I . him	-6.96	him . me	6.796	0.771	4.555
wl-6-posessive	7	15	his . ones	$-\infty$	its . our	6.727	5.879	0.616
wl-7-adverb	10	43	no . about	$-\infty$	not . only	4.853	0.965	1.769
wl-8-conj	10	45	if . either	-3.27	or . nor	4.744	1.378	1.451
wl-9-adj	10	43	even . great	$-\infty$	good . great	8.513	1.887	3.199
wl-10-wh	10	33	which . whose	$-\infty$	what . whatever	6.485	2.761	1.635
wl-11-noun	10	45	man . hand	1.827	man . day	3.912	3.046	0.412
wl-12-aux	10	43	cannot . has	$-\infty$	can . cannot	7.201	4.665	2.629
group mean and rms for 489 pairs							2.378	2.716

The green colors are the words that are the same as in the 640-128-80 dataset. Many means and most rms are similar to the smaller dataset. So that's good: the datasets are reflective of one-another. The means are .. low, but perhaps not surprisingly low, as they are a rather disparate collection of words...

Rather than reporting on all 66 inter-cluster distances, it seems simplest to provide a single score: the grand-total average and rms of the inter-cluster scores. There are 4617 inter-cluster pairs, with a mean of -1.32 and rms of 3.007. Comparing to the intra-cluster mean and rms, it looks like the difference is 3.70, or about 1.3 std-dev distance between the two.

Inter-intra MI summary (for the poorly-filtered datasets)

Pursuing above, the table below records the inter and intra cluster average MI for the datasets.

Attention: The table below is generated with the poorly-filtered datasets described above. A table further below presents the same data for better-filtered datasets. Comparing tables, one sees they are mostly the same: most numbers change by a few percent; some differences are as large as 10% and some are under one percent. So this is mostly representative.

The big question here is why shapes make things worse; this remains a big unresolved question.

filename	tot	intra MI	intra RMS	tot	inter MI	inter RMS	separation
filt-640-128-80-marg	208	3.967	2.042	556	1.312	2.609	1.15
filt-320-64-40-marg	296	3.700	2.139	1110	0.754	2.674	1.22
filt-160-32-20-marg	377	3.050	2.513	2120	-0.118	2.856	1.18
filt-80-16-10-marg	448	2.559	2.802	3429	-0.91	2.989	1.20
filt-40-8-5-marg	489	2.378	2.716	4617	-1.32	3.007	1.29
filt-20-4-2-marg	501	2.394	2.603	5656	-1.572	3.013	1.42
mpg-marg							
filt-640-128-80-shape	311	2.489	1.947	2386	0.288	1.992	1.13
filt-320-64-40-shape	448	1.990	2.102	4349	-0.063	2.015	1.00
filt-160-32-20-shape	479	1.747	2.420	5367	-0.391	2.178	0.93
filt-80-16-10-shape	500	1.884	2.385	5885	-0.38	2.232	0.98
filt-40-8-5-shape	513	1.974	2.252	6245	-0.254	2.131	1.02
filt-20-4-2-shape	516	2.024	2.198	6266	-0.096	1.989	1.01
mpg-shape							

The “separation” is the number of stddev between the intra and inter MI values.

Trimmed dataset

Redo of trimming, as explained above. These are more tightly trimmed, but still not fully trimmed. See below for discussion. Time to perform the filtering is about 4200 seconds for each file.

Notation: filename-WW-DD-SS-foo where WW is the cutoff for the minimum time this word has been observed. DD is the cutoff for the number of times a disjunct has been observed, and SS is the cutoff for the number of times a given section has been observed. This is in keeping with how the filter object is designed in the main atomspace repo – these are the three filter parameters.

file name	total pairs	words	disjuncts	sparsity	entropy
r3-zfil-640-128-80-marg	4006	409	2496	7.9934	-3.191
r3-zfil-320-64-40-marg	11157	749	6754	8.8247	0.2809
r3-zfil-160-32-20-marg	30675	1587	18840	9.9288	3.4436
r3-zfil-80-16-10-marg	73480	2816	43687	10.709	6.5297
r3-zfil-40-8-5-marg	172853	5076	101593	11.543	9.5523
r3-zfil-20-4-2-marg	470567	10110	261323	12.455	13.289
r3-mpg-marg.rdb	28436901	377553	25698949	18.380	24.100
zfil-640-128-80-shape	12340	437	6961	7.9455	-1.338
zfil-320-64-40-shape	34737	811	18274	8.7369	2.4048
zfil-160-32-20-shape	99342	1777	53238	9.8953	6.0846
zfil-80-16-10-shape	238735	3081	120546	10.603	9.6970
zfil-40-8-5-shape	566970	5483	277064	11.388	13.206
zfil-20-4-2-shape	1558158	10916	738026	12.336	17.160
mpg-shape					

Some of the thinner datasets do not have all of the words for all of the clusters. The "marg" datasets use the Sections (pseudo-csets) only; the "shape" datasets include both Sections and Cross-Sections.

So here's the trimming problem: the algo used is not yet fully recursive. So: first round of trimming removes sections which have connectors that don't appear as words in the left-basis. This leaves some words with no sections on them. These are removed. However, those words might still appear in some connectors. Thus a second round of trimming is needed. And so on, and so on, until a stable collection is arrived at.

Some additional confusion: there might be words in the left basis that never appear in connectors. These also have to be removed. When these are removed, corresponding sections disappear as well, necessitating another recursive round. Yech. So lets proceed without this, and see what happens. We can try again, later.

Notes: 'r3-mpg-marg.rdb' is just a copy of 'run-1-en_mpg-tranche-123.rdb' with MMT marginals for Sections only (no CrossSections).

Inter-intra MI summary

The table below records the inter and intra cluster average MI for the datasets.

This proceeds as follows. Given one cluster, the MI was computed for each pair of words in the cluster. These were then averaged together to obtain the average cluster MI, and also the RMS for that cluster. The averaging is unweighted (all word-pairs get a uniform weight). Next, all of these cluster scores were averaged together, to give the "intra MI" for the entire dataset. This second averaging was weighted by the number of pairs in each cluster; the result is that all word-pairs get a uniform weight. The "tot" column records the total number of pairs. In formulas:

$$intra = \sum_{c\in C} \sum_{w,u\in c} MI(u,w)$$

where C is the set of all of the clusters.

The “inter MI” column computes the average MI over all word-pairs, where the two words are in distinct clusters. That is, each word is in a different cluster:

$$inter = \sum_{c\in C} \sum_{c'\in C; c'\neq c} \sum_{w\in c; u\in c'} MI(u,w)$$

The “tot” column records the total number of such pairs.

The “separation” is the number of stddev’s between the intra and inter MI values. The stddev is the larger of the two RMS’s.

filename	tot	intra MI	intra RMS	tot	inter MI	inter RMS	separation
zfil-640-128-80-marg	202	4.077	2.029	548	1.475	2.604	1.000
zfil-320-64-40-marg	288	3.752	2.198	1094	0.834	2.654	1.100
zfil-160-32-20-marg	374	3.044	2.521	2104	-0.079	2.834	1.102
zfil-80-16-10-marg	448	2.576	2.791	3420	-0.886	2.985	1.160
zfil-40-8-5-marg	489	2.3831	2.711	4610	-1.315	2.990	1.237
zfil-20-4-2-marg	501	2.3970	2.601	5654	-1.565	3.009	1.317
mpg-marg	512	2.5315	2.385	6250	-1.101	2.604	1.395
zfil-640-128-80-shape	307	2.5024	1.863	2359	0.2723	1.975	1.129
zfil-320-64-40-shape	441	1.8892	2.012	4310	-0.204	1.959	1.041
zfil-160-32-20-shape	479	1.5841	2.350	5365	-0.547	2.091	0.907
zfil-80-16-10-shape	500	1.7075	2.290	5884	-0.551	2.151	0.986
zfil-40-8-5-shape	504	1.7782	2.176	6148	-0.437	2.044	1.018
zfil-20-4-2-shape	516	1.8864	2.134	6266	-0.233	1.937	0.993
mpg-shape							

So we conclude:

- Less filtering gives better results.
- Shapes make all vectors more similar to each other. The RMS with shapes is smaller, considerably smaller, with shapes, than without them.
- Unfortunately, this is true not only within clusters, but also between them! Shapes fail to improve the discrimination between clusters. This is counter-intuitive.
- Shapes are stunningly disappointing; the reason for this is quite unclear. It may be due to the faulty filtering, and so deserves one more round. Yet there’s a hint that even better filtering still won’t improve things. So this is an open problem.

Inter-intra joint probability summary

The table below records the inter and intra cluster average joint probability for the datasets. The joint probability is as defined previously; it is the normalized inner product.

$$p(w,u) = \frac{i(w,u)}{i(*,*)} = \sum_d p_M(w,d)\, p_M(u,d)$$

with

$$p_M(w,d) = \frac{N(w,d)}{N(*,*)}$$

Several problems arise when computing the "average probability" for words in a cluster or between clusters. Some words are observed far more frequently than others, and we really do want to adjust for that. Thus, it is best to compare the joint probability divided by the marginal probabilities for each word:

$$\frac{p(w,u)}{p_M(w,*)\, p_M(u,*)}$$

where the wild-cards range over the disjuncts.

Next, computing the RMS for the above doesn't really make sense, cause probabilities cannot go negative. It seems better to compute the averages and RMS for $\log_2 p$, that is, for

$$\log_2 \frac{p(w,u)}{p_M(w,*)\, p_M(u,*)}$$

This should be quite similar to the MI.

Here, "larger is better" – larger log2 numbers mean a better match.

filename	tot	intra joint	intra RMS	tot	inter joint	inter RMS	separation
zfil-640-128-80-marg	202	3.295	2.621	548	0.563	2.677	1.020
zfil-320-64-40-marg	288	2.634	2.661	1094	-0.363	2.913	1.029
zfil-160-32-20-marg	374	1.6787	2.945	2104	-1.493	3.047	1.041
zfil-80-16-10-marg	448	1.1077	3.309	3420	-2.398	3.235	1.059
zfil-40-8-5-marg	489	0.8812	3.210	4610	-2.907	3.220	1.177
zfil-20-4-2-marg	501	0.7920	3.126	5654	-3.218	3.179	1.262
mpg-marg	512	0.7484	2.855	6250	-2.971	2.787	1.303
zfil-640-128-80-shape	307	2.1093	2.033	2359	0.0777	2.365	0.859
zfil-320-64-40-shape	441	1.5268	2.310	4310	-0.530	2.422	0.849
zfil-160-32-20-shape	479	1.0244	2.433	5365	-1.099	2.412	0.873
zfil-80-16-10-shape	500	0.8585	2.317	5884	-1.416	2.384	0.954
zfil-40-8-5-shape	504	0.6339	2.171	6148	-1.580	2.271	0.975
zfil-20-4-2-shape	516	0.3893	2.150	6266	-1.758	2.154	0.997
mpg-shape							

So we conclude:

- Less filtering gives better results.
- Quality is close to the MI results, and only slightly poorer.
- Shapes offer no advantage; they're a clear disadvantage.

Inter-intra variation of information summary

The table below records the inter and intra cluster average "variation of information" for the datasets. This is defined as

$$VI(u,w) = H(u,w) - MI(u,w)$$

where $H(u,w)$ is the log of the joint probability, as defined before:

$$H(w,u) = -\log_2 p(w,u)$$

Here, "smaller is better" – smaller numbers mean a better match. Per Wikipedia, this is a metric (obeying the triangle inequality).

filename	tot	intra vi	intra RMS	tot	inter vi	inter RMS	separation
zfil-640-128-80-marg	202	10.121	2.911	548	13.619	3.656	0.957
zfil-320-64-40-marg	288	11.583	3.725	1094	15.957	4.241	1.031
zfil-160-32-20-marg	374	13.206	4.739	2104	18.499	4.895	1.081
zfil-80-16-10-marg	448	14.451	5.516	3420	20.701	5.535	1.129
zfil-40-8-5-marg	489	15.044	5.678	4617	22.067	5.847	1.201
zfil-20-4-2-marg	501	15.064	5.480	5656	22.925	6.115	1.286
mpg-marg	512	15.294	5.309	6250	22.677	5.607	1.317
zfil-640-128-80-shape	307	12.469	3.178	2359	16.093	3.610	1.004
zfil-320-64-40-shape	441	14.340	3.995	4310	18.162	4.037	0.947
zfil-160-32-20-shape	479	15.235	4.682	5365	19.448	4.521	0.900
zfil-80-16-10-shape	500	15.550	4.526	5884	20.040	4.702	0.955
zfil-40-8-5-shape	504	15.769	4.310	6148	20.299	4.558	0.994
zfil-20-4-2-shape	516	16.067	4.260	6266	20.369	4.280	1.005
mpg-shape							

So we conclude:

- A touch better than the joint-probability results above.
- Separation is not as good as MI, but are within 5% of the MI separation.

Inter-intra normalized variation of information summary

The table below records the inter and intra cluster average "normalized variation of information" for the datasets. This is defined as

$$NVI(u,w) = \frac{MI(u,w)}{H(u,w)}$$

where $H(u,w)$ is the log of the joint probability, as defined before.

Here, "larger is better" – larger numbers mean a better match.

filename	tot	intra nvi	intra RMS	tot	inter nvi	inter RMS	separation
zfil-640-128-80-marg	202	0.2925	0.145	548	0.1047	0.175	1.075
zfil-320-64-40-marg	288	0.2553	0.150	1094	0.0595	0.162	1.207
zfil-160-32-20-marg	374	0.2056	0.163	2104	0.0083	0.156	1.212
zfil-80-16-10-marg	448	0.1746	0.171	3420	-0.030	0.151	1.200
zfil-40-8-5-marg	489	0.1607	0.166	4617	-0.049	0.144	1.260
zfil-20-4-2-marg	501	0.1597	0.160	5656	-0.058	0.138	1.364
mpg-marg	512	0.1627	0.150	6250	-0.039	0.122	1.347
zfil-640-128-80-shape	307	0.1728	0.121	2359	0.0244	0.120	1.223
zfil-320-64-40-shape	441	0.1276	0.125	4310	-0.003	0.108	1.039
zfil-160-32-20-shape	479	0.1095	0.136	5365	-0.019	0.107	0.941
zfil-80-16-10-shape	500	0.1122	0.131	5884	-0.018	0.106	0.994
zfil-40-8-5-shape	504	0.1134	0.126	6148	-0.013	0.101	0.995
zfil-20-4-2-shape	516	0.1164	0.124	6266	-0.003	0.097	0.968
mpg-shape							

So we conclude:

- Separation is a fair bit better than the un-normalized variation of information.
- Separation is about 10% better than the MI separation, for the plain sections.
- Separation is about 5% better than the MI separation, for the shapes.

Inter-intra cosine summary

The table below records the inter and intra cluster average cosine distance for the datasets. Although this is "obviously" inappropriate for this work, it is the industry standard benchmark, so we want to see how it stacks up.

filename	tot	intra cos	intra RMS	tot	inter cos	inter RMS	separation
zfil-640-128-80-marg	404	0.1798	0.267	4646	0.0158	0.077	0.614
zfil-320-64-40-marg	475	0.1614	0.227	5741	0.0162	0.073	0.641
zfil-160-32-20-marg	484	0.1571	0.210	5844	0.0182	0.071	0.663
zfil-80-16-10-marg	501	0.1600	0.203	6054	0.0200	0.071	0.690
zfil-40-8-5-marg	507	0.1614	0.200	6163	0.0211	0.070	0.701
zfil-20-4-2-marg	516	0.1620	0.197	6270	0.0223	0.069	0.710
mpg-marg	516	0.1465	0.180	6270	0.0212	0.064	0.695
zfil-640-128-80-shape	404	0.1773	0.193	4646	0.0467	0.099	0.676
zfil-320-64-40-shape	475	0.1851	0.184	5741	0.0515	0.095	0.726
zfil-160-32-20-shape	484	0.1863	0.179	5844	0.0530	0.091	0.743
zfil-80-16-10-shape	501	0.1905	0.177	6054	0.0546	0.090	0.770
zfil-40-8-5-shape	507	0.1908	0.175	6163	0.0559	0.089	0.770
zfil-20-4-2-shape	516	0.1962	0.176	6270	0.0581	0.091	0.784
mpg-shape							

So we conclude:

- This is not competitive with the probability-based methods, not even close.
- Shapes improve the cosine distance! Even though this is still with non-optimal filtering! Hypothesis: the shape vectors really do improve vector alignment (as this clearly shows) but the extra MI terms dissolve it back out. That is, the shapes do not carry any extra information not already present in the vectors.
- This may be a misinterpretation, though. By comparing inter scores with and without shapes, we see that the shapes make the vectors more similar. This is also the case for the intra scores. In particular, the RMS for the intra scores goes down (cause they're more similar) which makes the separation improve.
- Conclude: shapes make all vectors appear to be more similar.

Inter-intra log cosine summary

The table below records the inter and intra cluster average log2 cosine distance for the datasets. This is similar to above, except that the log2 makes the numbers more directly comparable to the probabilities. The cosine ranges from 0.0 to 1.0, with 1.0 meaning "highly similar". Thus, the log emphasizes dissimilarities. Larger values are better.

Doing this does alter the contributions to the averages: for the smaller datasets, zero cosines are frequent. This is due to the dataset size: just not enough disjuncts. These do contribute to the average cosine distance, but not to the log distance; they're discarded (since log zero is minus infinity). This seems to be an OK thing to do for the present situation.

filename	tot	intra cos	intra RMS	tot	inter cos	inter RMS	separation
zfil-640-128-80-marg	202	-2.168	1.716	548	-4.267	2.193	0.957
zfil-320-64-40-marg	288	-2.867	2.119	1094	-5.325	2.448	1.004
zfil-160-32-20-marg	374	-3.694	2.616	2104	-6.501	2.714	1.034
zfil-80-16-10-marg	448	-4.179	2.997	3420	-7.431	2.993	1.085
zfil-40-8-5-marg	489	-4.344	2.981	4610	-7.958	3.071	1.177
zfil-20-4-2-marg	501	-4.251	2.905	5654	-8.203	3.114	1.269
mpg-marg	512	-4.344	2.722	6250	-8.040	2.804	1.318
zfil-640-128-80-shape	307	-2.744	1.652	2359	-4.595	2.006	0.923
zfil-320-64-40-shape	441	-3.263	2.074	4310	-5.207	2.163	0.899
zfil-160-32-20-shape	479	-3.480	2.303	5365	-5.581	2.283	0.912
zfil-80-16-10-shape	500	-3.359	2.156	5884	-5.605	2.298	0.977
zfil-40-8-5-shape	504	-3.274	1.990	6148	-5.498	2.166	1.027
zfil-20-4-2-shape	516	-3.198	1.906	6266	-5.334	2.002	1.067
mpg-shape							

So we conclude:

- (Looking at the ‘separation‘ column:) This is competitive with the joint-probability scores, maybe even a hair better, and is just a little worse (about 8% worse) than the MI scores.
- Unlike the plain cosine, the shapes make things worse. The log2 has the effect of magnifying differences (since the log is almost linear for high similarity, but stretches the scale strongly for low similarity.) And yet, if we compare the inter scores with and without shapes, it seems that shapes are making vectors more similar. This is consistent with the non-log cosine, and the separation is controlled by the RMS.

Inter-intra overlap summary

The table below records the inter and intra cluster average overlap similarity for the datasets. This is provided per API for the ‘add-similarity-compute‘ object. The overlap is simply the unweighted Jaccard similarity. This ranges from 0.0 to 1.0, with the interpretation that 1.0 means very similar.

That is, define the indicator function as

$$\sigma(w,d) = [N(w,d) > 0]$$

Then the overlap similarity is

$$\begin{aligned}\text{overlap}(w,u) &= \sum_d \frac{\min(\sigma(w,d),\sigma(u,d))}{\max(\sigma(w,d),\sigma(u,d))} \\ &= \frac{|S(w)\cap S(u)|}{|S(w)\cup S(u)|}\end{aligned}$$

where the support of a vector is defined as

$$S(w) = \{d : N(w,d) > 0\}$$

The averages and RMS are computed for $\log_2 \text{overlap}(w,u)$. This is done for two reasons. One is that it makes the reporting scale similar to that for the probability, above. The log is approx linear for high similarity, and thus emphasizes differences for low similarity. As before, "larger is better" – larger log2 numbers mean a better match.

filename	tot	intra olap	intra RMS	tot	inter olap	inter RMS	separation
zfil-640-128-80-marg	202	-3.281	1.454	548	-4.925	1.412	1.131
zfil-320-64-40-marg	288	-4.021	1.518	1094	-5.828	1.490	1.191
zfil-160-32-20-marg	374	-4.689	1.770	2104	-6.761	1.626	1.170
zfil-80-16-10-marg	448	-4.949	1.969	3420	-7.338	1.779	1.213
zfil-40-8-5-marg	489	-5.102	1.982	4610	-7.750	1.872	1.336
zfil-20-4-2-marg	501	-5.162	1.894	5654	-7.968	1.915	1.465
mpg-marg	512	-7.869	1.619	6250	-10.28	1.678	1.440
zfil-640-128-80-shape	307	-3.948	1.409	2359	-5.842	1.319	1.344
zfil-320-64-40-shape	441	-4.444	1.616	4310	-6.408	1.427	1.215
zfil-160-32-20-shape	479	-4.610	1.709	5365	-6.697	1.508	1.221
zfil-80-16-10-shape	500	-4.531	1.590	5884	-6.668	1.499	1.343
zfil-40-8-5-shape	504	-4.546	1.493	6148	-6.639	1.412	1.402
zfil-20-4-2-shape	516	-4.699	1.476	6266	-6.620	1.306	1.302
mpg-shape							

So we conclude:

- For the shapeless sections, this provides superior separation to the MI scores. Its about 10% or 15% better.
- With shapes, the separation is even better – about 15% to 40% better than the MI scores.
- The separation with shapes is mostly better than that without, so this breaks a trend.
- Shapes give the most improvement for the small datasets! They're not useful, once the dataset gets large.
- However, as before, the addition of shapes seems to make vectors more similar, in general, no matter what kind they are.

Inter-intra Jaccard summary

The table below records the inter and intra cluster average “conditional Jaccard similarity” for the datasets. This is provided by the ‘cond-jacc‘ method on the ‘add-similarity-compute‘ object. This is the same as that defined, using above: its the Ruzicka similarity for the conditional probabilities. To recap, so that I don’t have to scroll up and re-read, the conditional Jaccard similarity is given by

$$J_R(N;u,w) = \frac{\sum_d \min\left(p(w,d), p(u,d)\right)}{\sum_d \max\left(p(w,d), p(u,d)\right)}$$

where $p(w,d) = N(w,d)/N(*,*)$.

The conditional Jaccard similarity ranges from 0.0 to 1.0. As above, log2 is taken.

filename	tot	intra jac	intra RMS	tot	inter jac	inter RMS	separation
zfil-640-128-80-marg	202	-3.508	1.760	548	-5.606	1.933	1.086
zfil-320-64-40-marg	288	-4.277	1.892	1094	-6.487	1.971	1.121
zfil-160-32-20-marg	374	-4.970	2.213	2104	-7.428	2.113	1.110
zfil-80-16-10-marg	448	-5.272	2.381	3420	-8.072	2.254	1.176
zfil-40-8-5-marg	489	-5.362	2.311	4610	-8.400	2.285	1.314
zfil-20-4-2-marg	501	-5.271	2.159	5654	-8.513	2.276	1.424
mpg-marg	512	-6.925	1.873	6250	-9.700	1.927	1.440
zfil-640-128-80-shape	307	-4.147	1.566	2359	-6.100	1.682	1.161
zfil-320-64-40-shape	441	-4.563	1.762	4310	-6.665	1.798	1.170
zfil-160-32-20-shape	479	-4.723	1.869	5365	-6.903	1.780	1.166
zfil-80-16-10-shape	500	-4.603	1.692	5884	-6.866	1.732	1.307
zfil-40-8-5-shape	504	-4.543	1.573	6148	-6.760	1.605	1.382
zfil-20-4-2-shape	516	-4.554	1.495	6266	-6.630	1.430	1.389
mpg-shape							

So we conclude:

- For the shapeless sections, this provides almost as good separation as the overlaps, and about 5% better than the MI scores.
- Using shapes seems to improve separability by a fair amount – again, bucking a trend. Most of the improvement is in the smaller datasets.
- As before, using shapes seems to make all vectors more similar to one-another.

Inter-intra Probability Jaccard summary

The table below records the inter and intra cluster average "probability Jaccard distance" for the datasets. This is provided by the `‘add-similarity-compute‘` object, by calling the `'right-prjacc` method. The probability-Jaccard similarity ranges from 0.0 to 1.0. As above, log2 is taken.

filename	tot	intra prjac	RMS	tot	inter prjac	RMS	separation
zfil-640-128-80-marg	202	-3.023	1.629	548	-5.029	1.812	1.107
zfil-320-64-40-marg	288	-3.765	1.768	1094	-5.905	1.879	1.139
zfil-160-32-20-marg	374	-4.466	2.086	2104	-6.851	2.020	1.144
zfil-80-16-10-marg	448	-4.792	2.295	3420	-7.501	2.171	1.181
zfil-40-8-5-marg	489	-4.887	2.238	4610	-7.840	2.214	1.319
zfil-20-4-2-marg	501	-4.813	2.107	5654	-7.966	2.224	1.418
mpg-marg							
zfil-640-128-80-shape	307	-3.696	1.466	2359	-5.559	1.561	1.193
zfil-320-64-40-shape	441	-4.132	1.700	4310	-6.108	1.675	1.162
zfil-160-32-20-shape	479	-4.294	1.814	5365	-6.368	1.687	1.143
zfil-80-16-10-shape	500	-4.171	1.639	5884	-6.342	1.663	1.306
zfil-40-8-5-shape	504	-4.109	1.516	6148	-6.239	1.545	1.379
zfil-20-4-2-shape	516	-4.114	1.438	6266	-6.111	1.382	1.388
mpg-shape							

So we conclude:

- For the shapeless sections, this provides the best separation of all, and is just a tad better than the conditional Jaccard, above.
- For shapeless separation, this is about 5% to 10% better than MI.
- Adding shapes further improves separability, again bucking a trend.

This is stunningly painfully slow for the larger datasets, and so the algo needs to be refactored to move common computations out of the inner loop. If this cannot be done, then this also is useless.

Summary

Lets try to summarize the results above. The table below reproduces the separations for the various different measures. The column titles are in the same order as in the text. There are nine columns.

filename	MI	joint	vi	nvi	cos	logcos	ovrlp	condj	prjac
zfil-640-128-80-marg	1.000	1.020	0.957	1.075	0.614	0.957	1.131	1.086	1.107
zfil-320-64-40-marg	1.100	1.029	1.031	1.207	0.641	1.004	1.191	1.121	1.139
zfil-160-32-20-marg	1.102	1.041	1.081	1.212	0.663	1.034	1.170	1.110	1.144
zfil-80-16-10-marg	1.160	1.059	1.129	1.200	0.690	1.085	1.213	1.176	1.181
zfil-40-8-5-marg	1.237	1.177	1.201	1.260	0.701	1.177	1.336	1.314	1.319
zfil-20-4-2-marg	1.317	1.262	1.286	1.364	0.710	1.269	1.465	1.424	1.418
mpg-marg	1.395	1.303	1.317	1.347	0.695	1.318	1.440	1.440	
zfil-640-128-80-shape	1.129	0.859	1.004	1.223	0.676	0.923	1.344	1.161	1.193
zfil-320-64-40-shape	1.041	0.849	0.947	1.039	0.726	0.899	1.215	1.170	1.162
zfil-160-32-20-shape	0.907	0.873	0.900	0.941	0.743	0.912	1.221	1.166	1.143
zfil-80-16-10-shape	0.986	0.954	0.955	0.994	0.770	0.977	1.343	1.307	1.306
zfil-40-8-5-shape	1.018	0.975	0.994	0.995	0.770	1.027	1.402	1.382	1.379
zfil-20-4-2-shape	0.993	0.997	1.005	0.968	0.784	1.067	1.302	1.389	1.388
mpg-shape									

So we conclude:

- In all cases, the less that the dataset is trimmed, the greater the separation between clusters. That is, trimming is unambiguously harmful to the quality of the data.
- Cosine distance is stunningly bad. It's in a league of it's own. Other than that, the scores are actually pretty similar, differing by at most dozens of percent.
- The MI is a middle-of-the-road measure. It is better than four columns: joint-probability, variation of information, cosine, log cosine, but worse than the remaining four: normalized VI, overlap, conditional Jaccard and probability Jaccard.
- Shapes have a mixed effect on the discrimination between clusters. They make things much worse for the five worst metrics. They make things better for the three best metrics, but only when the datasets are heavily trimmed. For the least-trimmed dataset, shapes make things worse.

Ranking depends on the amount of trimming. Different rankings result, depending on how much trimming one does. The table below shows first through fifth places for rankings.

filename	1st	2nd	3rd	4th	5th
zfil-640-128-80-marg	overlap	prjac	condj	nvi	joint
zfil-320-64-40-marg	nvi	overlap	prjac	condj	MI
zfil-160-32-20-marg	nvi	overlap	prjac	condj	MI
zfil-80-16-10-marg	overlap	nvi	prjac	condj	MI
zfil-40-8-5-marg	overlap	prjac	condj	nvi	MI
zfil-20-4-2-marg	overlap	condj	prjac	nvi	MI
mpg-marg					
zfil-640-128-80-shape	overlap	nvi	prjac	condj	MI
zfil-320-64-40-shape	overlap	condj	prjac	MI	nvi
zfil-160-32-20-shape	overlap	condj	prjac	nvi	logcos
zfil-80-16-10-shape	overlap	condj	prjac	nvi	MI
zfil-40-8-5-shape	overlap	condj	prjac	logcos	MI
zfil-20-4-2-shape	condj	prjac	overlap	logcos	vi
mpg-shape					

So we conclude:

- Overlap is a clear winner, across the board.
- Conditional Jaccard and Probability Jaccard are close to overlap, both for the plain disjuncts, and for the shapes.
- For the shape datasets, 4th and 5th place are very distant from 3rd place.
- From first principles, one would have concluded that prjac should have done best, as it is in some sense “optimal”. Yet here, it rarely gets out of third place, hitting second place only a few times. The fact that it is very computationally expensive is a further strike against it.
- Overlap beats the dominant 2nd place holder by a fair bit. One upset: large shapes. When overlap was in second place, it was by a slim margin.
- nvi makes a few showings, but does not seem to be worth considering any further. When its not in 1st place, its pretty distant from overlap.
- condj seems to be overall second place, and a fairly strong contender.

Not visible in the above table are the above factoids, observed elsewhere:

- prjac is very compute intensive.
- Shapes consistently reduce the differences between vectors. Both inter-cluster differences are smaller but also intra-cluster differences are also smaller. Both RMS scores are pretty much always tighter, with shapes.
- Shapes have a mixed effect on discrimination between clusters. Discussed in the paragraph above.

- Using shapes has a large impact on RAM and CPU usage. It takes a lot of time to create them, they triple or quadruple RAM usage, and all vector similarity computations take longer, because the vectors are longer.
- Most of the tables have no results for the full dataset. That's because computing with the full dataset is very CPU intensive. We conclude that a very mild amount of trimming, to eliminate the onesie-twosie counts is likely to be well worth the effort. Particularly notable is that the scores for overlap and condj are tied for the full dataset, and MI is in a close 3rd place. Rerunning the experiment with more lightly trimmed datasets might be worthwhile. (If rerun, then the "better" clusters should be used. This will prevent direct comparison, but so what.)

To summarize: moving forward, it seems like we should eschew trimming, and use overlap for the similarity measure. However, understanding conditional Jaccard is also worthwhile. Retaining MI as a benchmark also seems prudent, given how much effort has been sunk into it. If there's time, then understanding how the "optimal" probability Jaccard seems not optimal for this problem is worthwhile.

Apparently, the use of shapes is not worth the effort.

July 2021 - Big English

Attempt to do the big English data set again, "correctly", this time. The corpus comes in five parts, four from project Gutenberg, a fifth that is fan fiction. This naturally lends itself to datasets of increasing size. Table below shows some stats.

Tranche	1	2	3	4	5
Ingestion of corpora for word-pair counting					
Files	3027	4499	5712	5418	8221
Sentences	426941	547704	752788	694302	1003799
Sentences (cumulative)	426941	974645	1727433	2421735	3425534
Words	8133834	6632258	14677945	13217488	19532255
Words (cumulative)	8.134M	14.766M	29.444M	42.661M	62.194M
Wall-clock time (hours)		13.2 hr	31.0 hr		40.6
Highest RAM Usage		12GB	30GB		37GB
CPU time (hours)		51.4 hr	133 hr		199 hr
Atoms in database		29287757	56675645	76425718	104150297
RocksDB file size		2.0 GB	3.5 GB	4.75 GB	6.6 GB
Computation of MI for word pairs					
Wall-clock time (hours)	1.51 hr	2.185	4.43	6.14 hr	9.25
Highest RAM usage				72 GB	96 GB
Left words (rows)	104879	140140	304085	397229	531345
Right words (columns)	105701	142101	306920	402896	538513
Unique word pairs	9.797M	14.573M	28.184M	38.011M	51.806M
Sparsity (log_2)	10.144	10.416	11.693	12.040	12.431
Total observation count	268.9M	492.6M	985.5M	1.431G	2.091G
Observations/pair	27.449	33.804	34.966	37.660	40.360
Entropy Total	17.827	17.889	18.378	18.503	18.631
Entropy Left	9.7963	9.8102	10.069	10.109	10.148
Entropy Right	9.5884	9.5463	9.8321	9.8801	9.9265
Total MI	1.5572	1.4677	1.5227	1.4863	1.4431
Atoms in database	20122297	29852254	57897670	78025983	106290028
RocksDB file size	2.34 GB	3.44 GB	6.11 GB	8.03 GB	12.08 GB

So... interestingly, the entropy is roughly independent of the dataset size. Sparsity is high and increasing. That's good.

Table below is for MPG (Max Planar Graph) Parsing.

Tranche	1	2	3	4	5
Time to load pairs	944 s	1501 s	3261 s	4214s	6488s
Wall-clock time (hours)	7.0 h	4.3 h	13.1	11.6	
RAM to load pairs			41.3	67.8	93.1
Highest RAM usage	30.2 GB	47.9	92.6		
CPU time (hours)	52.1	85.7	180.3	267	
Atoms in database	36M	58M	113M	156M	
RocksDB file size	4.43 GB	6.95 GB	13.8 GB	18.8GB	

Wall-clock time is time to process that one particular tranche. Cumulative wall-clock is that, plus the sum of the earlier ones.

Loading speed appears to be about 1GB/minute, or 1M atoms/min or 17K atom-s/sec.

The End

This is the end of the diary. It spans the time period of February-August 2021. The next part is Part Three.