Language Learning Diary - Part Three

Linas Vepstas

Sept 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 Three of the diary on the language-learning effort continues work on the English dataset. This part is concluded, the next part is Part Four.

Summary Conclusions

A short summary of what is found here:

- Trimming the word-disjunct dataset to remove words or disjuncts or word-disjunct pairs observed only once is a good thing, and reduces dataset size by 90% or more. Trimming more than this is a bad thing.
- Part of the above result may be due to a bug in the word-segmentation code: leading/trailing double-quotes are not split from words. The processing code then tries to escape these with backslashes, and repeated processing creates a shower of backslashes. This appears to pointless inflate the number of words in a dataset. This bug needs to be fixed.
- It is not known if trimming word-pair files to remove low-count word-pairs is a good idea or not. This will require new (and time-consuming) experiments. Some trimming is surely good, but how much?
- The mutual information between word-vectors is given by a Gaussian distribution. This was seen before, but is confirmed again here. Recall that a word-vector consists of a word, and the vector basis elements are disjuncts. These word-vectors are reminiscent of SkipGrams, but are constructed differently. I

don't know of any theoretical work that could explain this experimental result (or any kind of result, for that matter).

- The self-MI of a word-vector against itself appears to be given by a weak log-normal distribution. It is almost but not quite uniformly distributed, ranging from about 2 and gradually tapering to 30 for the dataset explored here. An easy theoretical argument can be made that the self-MI must be positive, but that's all. I don't know of any theoretical work that could explain this experimental result.

- Other ways of comparing word vectors, including basic Jaccard (vector overlap) distance, and a conditional-Jaccard distance do correlate in general with the MI. An exploration of these alternate ways of exploring word similarities was driven by work reported in the Diary part Two. Here, nothing unusual stands out. The distribution of these appears to be given by a log-normal curve, or possible a log logistic curve, but the data is not refined enough to tell. There is no theoretical basis to expect any of this.

- Examining the top-100 most similar word pairs by hand (by gut feel) suggests that all but one or two word-pair recommendations are excellent. This visual exam also seems to suggest that maybe MI provides better recommendations than the other metrics (contradicting earlier results). Certainly, computing MI is much faster.

- Both of the Jaccard measures have a self-distance score of exactly zero. By contrast, the self-MI is not zero, and it suggests that there it can be used as a natural scale to help determine when words should be classified together. That is, a low self-MI word is similar to everything, and maybe should not be merged with high self-MI words. Or perhaps it is an indicator of a word with many distinct meanings. I don't understand this yet, and it might be a fruitful topic to examine.

- In order to make a determination to merge a pair of words, one must look at both their similarity, and their frequency of occurrence. One wishes to merge words that are similar and are also frequently-occurring. To this end, the text proposes that the similarity should be combined with the average log-frequency (log-probability) of the two words. Combining this idea with the formula for MI gives the explicit formula

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

  The square-root arises from the factor of 1/2 that arises from averaging two things together. The square-root makes this formula look unusual, but it seems to provide an excellent way of ranking word-pairs for merging. The distribution again follows the normal Gaussian Bell curve. As mentioned, the top 100 suggestions for merge candidates look very healthy.

The diary Part Four will examine a merge in gruesome detail.

September 2021

After the previous setback with grammatical clustering, it seems like more exploratory work is in order. Part of the exploratory effort is to discover faster ways of finding similar words, and to find algorithms that can do incremental or continuous learning. Some questions along this path:

- What is the distribution of disjuncts? (Yes, we've done this before, but lets do it again for the latest "clean" dataset.) Create a graph showing rank vs number of times a disjunct has been observed.
- How are disjuncts shared? Create a graph showing the rank vs. the number of words that share this disjunct.
- Scatterplots: ...

Expt-4 – Trimming – Sept 2021

The above questions will be aimed at a specific dataset. The initial choice of dataset was '`run-1-en_mpg-tranche-123.rdb`', as described earlier in the diary. The intent is to explore two versions: the full dataset, and a lightly trimmed dataset.

What is explored here (in expt-4) is how the size of the dataset varies with the amount of trimming. Trimming is desirable, since it can be used to manage dataset sizes. Computations run faster on smaller datasets.

The outcome of trimming proves to be surprising: even "mild" trimming removes large parts of the word–disjunct datasets. It removes much more than one might naively guess from a Zipfian distribution. The underlying reason for this is confusing: it turns out that a bug in tokenization may be ruining the distribution. The tokenizer does not split double-quotes from the beginning/ending of words, and later stages add back-slashes to escape the quotes. A waterfall of quotes results, polluting the dataset with oodles of backslashes. These are observed infrequently, and how they interact with the quality of the datasets is unclear.

Coupled with the earlier observation that even mild trimming damages the data (as seen in expt-3, at the end of diary Part Two), this suggests that trimming should be kept to a minimum. By the conclusion, it emerges that perhaps the ideal trim is one that removes all words, disjuncts and word–disjunct pairs that are observed only once.

"Trimming" means removing all words, disjuncts and word–disjunct pairs with observation counts below some given thresholds. The thresholds for these three counts can be set independently. These are the three numbers below: e.g. 10-4-2 means "discard all words with an observation count of 10 or less, all disjuncts with a count of 4 or less, and all word–disjunct pairs with a count of 2 or less."

The trimming process is funny: trimming (as implemented) is not idempotent (but it should be). Basically, the linkage filter, which guarantees that connectors have corresponding word entries, knocks out disjuncts and so pushes words below the cut-off. Repeating this process then knocks out more connectors. It needs to be run until its stable. This is CPU consuming.

The table below shows the result of repeated trimming, interleaved with a linkage filter to remove disjuncts with connectors in them that do not have words any more. Each was written to a distinct file.

filename	num words	num disjuncts	total pairs	sparsity	obs/pair
r4-mpg-marg	377553	25698949	28436901	18.380	1.2796
r4-zfil-10-4-2-pass-1	11755	262705	473877	12.670	12.651
r4-zfil-10-4-2-pass-2	6402	174036	368566	11.562	14.623
r4-zfil-10-4-2-pass-3	6097	169970	362504	11.481	14.709
r4-zfil-10-4-2-pass-4	6059	169578	361967	11.471	14.716
r4-zfil-10-4-2-pass-5	6046	169400	361746	11.467	14.720
r4-zfil-10-4-2-pass-6	6046	169384	361726	11.467	14.720
r4-zfil-10-4-2-pass-7	6046	169384	361726	11.467	14.720

So that took an alarmingly large number of iterations to stabilize. It also trimmed the dataset extremely sharply. Alarmingly sharply. Lets try again.

Same idea, modified algo: here, instead of filtering, just delete outright. This will shrink the AtomSpace as we go along, avoiding some of the hassle with filtering. Hopefully it will run faster, too.

filename	num words	num disjuncts	total pairs	sparsity	obs/pair
r4-mpg-marg	377553	25698949	28436901	18.380	1.2796
xxx-trim-1-1-1-pass-1	191265	2366262	2208776		
r4-trim-1-1-1-pass-1	58491	1733439	2208776	15.486	4.6001
r4-trim-1-1-1-pass-2	55190	1588635	2060121	15.377	4.7418
r4-trim-1-1-1-pass-3	55180	1588397	2059878	15.377	4.7421
r4-trim-1-1-1-pass-4	55175	1588357	2059837	15.377	4.7421
r4-trim-1-1-1-pass-5	55174	1588338	2059817	15.377	4.7421
r4-trim-1-1-1-pass-6	55174	1588338	2059813	15.377	4.7421

Notes:

- The ‘xxx-trim-1-1-1-pass-1‘ version is not a file, but is rather the number of WordNodes, ConnectorSeqs and Sections in the AtomSpace, after removing those with only one observation count. This is not the same as the matrix dimensions, because the matrix is built by finding all Sections, and then taking the Words and ConnectorSeqs in those Sections. There are fewer of those, because some disjuncts have connectors to words that have been removed.
- The ‘r4-1-1-1-pass-1‘ version removes all words, disjuncts and sections that have one observation count. This is followed by removing all disjuncts and sections that do not have words as the germs of the sections. This results in words that don’t appear anywhere; these are removed.

- The ‘r4-1-1-1-pass-n‘ versions repeat this process, until it terminates. Iterations until termination are the same as for the earlier cut.

It is quite remarkable how much the dataset shrinks, simply by removing sections and words with just one count! This suggests several activities:

- Plot the distribution of words, disjuncts and sections and verify that they are Zipfian. I thought that the Zipfian distribution implied that half of the dataset would have a count of one, and not 5/6th or 9/10th's of it.
- Take a look at how trimming affects the other datasets.
- Introduce a periodic trim step into the processing pipeline. The hope is that this could dramatically shrink the size of the datasets, with very little loss of information.

Let's look at the previous graphs of the Zipfian distributions. These can be found in ‘`connector-sets-revised.pdf`‘ page 9ff. For words, the graph on page 9, left hand side shows that the count drops precipitously as a function of the rank. The logarithmic scale is deceiving; it looks like more than half of words have a count of only 1. The right-hand graph suggests the same for disjuncts. Likewise for the graphs on page 11 and 12. Thus, the present case is not new: this was seen in earlier datasets.

Lets try some deeper trimming. What happens then? Shown in table below.

filename	num words	num disjuncts	total pairs	sparsity	obs/pair
r4-mpg-marg	377553	25698949	28436901	18.380	1.2796
r4-trim-1-1-1-pass-6	55174	1588338	2059813	15.377	4.7421
r4-trim-2-2-2	16922	414956	629654	13.445	10.381
r4-trim-5-2-2	9313	357614	561309	12.535	10.976
r4-trim-10-4-2	6046	169384	361726	11.467	14.720

There is a very dramatic drop-off in the number of vocabulary words, as trimming is increased. Each of the later trims was created by starting with the earlier version, and trimming that. This history dependence should not matter. Looks like the earlier r4-trim-10-4-2 result is reproducible.

Single-count trim

Based on general results, it seems that a trim of removing all words, disjuncts and word–disjunct pairs with an observation count of one is a good thing. Based on this, some “master” trimmed files are prepared, stored under “run-1”. These were carefully processed. monitored and double-checked. They contain MMT values for the word–disjunct pairs. (They also contain word-pairs, and updated MI states for the word-pairs.)

Results of trimming are shown in the table below.

filename	nwords	ndisjuncts	total pairs	sparsity	obs/pair	MM^T ent
run-1-marg-tranche-1	134199	7470276	8131679	16.912	1.2079	22.107
run-1-t1-trim-1-1-1	16129	289038	388629	13.550	4.9890	14.506
run-1-marg-tranche-12	190699	13132418	14517517	17.396	1.2479	22.780
run-1-t12-trim-1-1-1	23865	559870	785426	14.054	5.3421	15.793
r4-mpg-marg	377553	25698949	28436901	18.380	1.2796	
r4-trim-1-1-1-pass-6	55174	1588338	2059813	15.377	4.7421	
run-1-marg-tranche-123	377553	25698949	28436901	18.380	1.2796	23.583
run-1-t123-trim-1-1-1	55174	1588334	2059813	15.377	4.7421	16.912
run-1-marg-tranche-1234	497690	36562917	40648095	18.772	1.3016	
run-1-t1234-trim-1-1-1	75431	2354562	3070162	15.820	4.8913	17.522

The shrinkage is dramatic, in each case. Overall effects in each case are quite similar.

I wonder what would happen, if MPG parsing was performed a second time, with the trimmed word-pairs...

Support Trim

The single-count trim above was the right idea, but it missed a spot: we also need to trim away all words/disjuncts that have a support of just one! So trimming is repeated, and summarized below. Trim steps alternate between support-trim and count trim.

filename	nwords	ndisjuncts	total pairs	sparsity	obs/pair	MM^T ent
run-1-t1-trim-1-1-1	16129	289038	388629	13.550	4.9890	14.506
run-1-t1-tsup-1-1-1	3737	29210	114147	9.9013	7.3261	12.868
run-1-t12-trim-1-1-1	23865	559870	785426	14.054	5.3421	15.793
run-1-t12-tsup-1-1-1	7110	65792	269822	10.760	7.7701	14.544
run-1-t123-trim-1-1-1	55174	1588334	2059813	15.377	4.7421	16.912
run-1-t123-tsup-1-1-1	11277	135962	560478	11.418	8.1629	15.756
run-1-t1234-trim-1-1-1	75431	2354562	3070162	15.820	4.8913	17.522
run-1-t1234-tsup-1-1-1	15083	205003	855718	11.819	8.5462	16.352

Holy Smokes! That is one heck of a trim! That is one heck of a trim! Word counts reduced by factors of 3 to 6, disjunct counts reduced by factors of 8 to 12, total pairs reduced by a factor of 3 to 4. And that's compared to the trimmed files!

Compared to the untrimmed:

filename	nwords	ndisjuncts	total pairs
run-1-marg-tranche-1	134199	7470276	8131679
run-1-t1-tsup-1-1-1	3737	29210	114147
ratio	36	256	71
run-1-marg-tranche-12	190699	13132418	14517517
run-1-t12-tsup-1-1-1	7110	65792	269822
ratio	27	200	54
run-1-marg-tranche-123	377553	25698949	28436901
run-1-t123-tsup-1-1-1	11277	135962	560478
ratio	33	189	50
run-1-marg-tranche-1234	497690	36562917	40648095
run-1-t1234-tsup-1-1-1	15083	205003	855718
ratio	33	178	48

Those are just huge numbers! Oddly reassuring though. I wonder how much of this is due to the bad tokenization and bad quote-escaping in the MPG parse pipeline...

Expt-5 – Trimming pairs – Sept 2021

What happens if we trimmed the pair files? Scripts in `experiments/run-5`. Table below.

Summary of results: the effect of trimming on quality is ambiguous. Because there are 24 observations made of each sentence, most word-pairs have observation counts of 24 or higher, and thus trimming less than this has almost no effect on the word-pair dataset size. Whether or not trimming more than this is a good idea is unknown. Conclusion: don't trim the word-pair dataset, at least not without further experimentation and measurements.

filename	lwords	rwords	tot pairs	sparsity	obs/pair	Entropy	MI
run-1-en_pairs-tranche-1	104879	105701	9797694	10.144	27.449	17.827	1.5572
r5-prs1-trim-1-1-1	104758	105425	6568020	10.716	40.455	17.686	1.5688
r5-prs1-trim-5-5-2	103555	104209	5159723	11.030	50.949	17.571	1.5734
r5-prs1-trim-10-10-4	101428	102030	3807387	11.408	67.826	17.391	1.5742
r5-prs1-trim-20-20-6	73735	74267	2984091	10.842	84.855	17.207	1.5400
r5-prs1-trim-40-40-8	52211	52360	2424861	10.139	102.46	17.035	1.5009
r5-prs1-trim-80-80-12	49757	49888	2412289	10.007	102.91	17.028	1.4956

So... trimming has a very different effect, here. Trimming to thresh of one does not have much of an effect. Trimming 10 raises the MI to an all-time high. Presumably it is cutting out the low-MI pairs. Don't we have a graph of MI vs. count, somewhere? Trimming to 10 also raises the sparsity to an all-time high. Is it a good idea to trim pairs? Maybe.

Its possible that the low-MI pairs never show up in sections anyway, to the trimming might not affect the computation of the sections, except maybe to speed up MPG parsing slightly, by offering fewer edges.

Word pairs in disjunct files

So ... it appears that the disjunct files all have the word-pairs in them. If the disjuncts are trimmed, then, as a side effect, many word-pairs are also trimmed. How does this work out? Lets take a look. The table below reports the word-pair subset of the indicated data files, after trimming and after recomputing the MI for the word-pairs.

filename	lwords	rwords	tot pairs	sparsity	obs/pair	Entropy	MI
run-1-marg-tranche-1	104879	105701	9797694	10.144	27.449	17.827	1.5572
run-1-t1-trim-1-1-1	11414	11447	4305534	4.9234	52.229	16.490	1.1341
run-1-marg-tranche-12	140140	142101	14572805	10.416	33.804	17.889	1.4677
run-1-t12-trim-1-1-1	15301	15395	7387010	4.9950	58.751	16.853	1.1425
run-1-marg-tranche-123	304085	306920	28184319	11.693	34.966	18.378	1.5227
run-1-t123-trim-1-1-1	43719	43970	18544906	6.6957	49.343	17.721	1.2360
run-1-marg-tranche-1234	397229	402896	38011356	12.040	37.660	18.503	1.4863
run-1-t1234-trim-1-1-1	58251	58573	26173771	7.0263	51.439	17.934	1.2433

Note that there are fewer words in the word-word pairs, than there are words in the word-disjunct pairs; about half as many (15350 avg vs. 23865). Note that MI dropped by "a lot"; i.e. by much more than the variation between files. The avg word-pair MI of the untrimmed tranches ranges from 1.55 to 1.44.

The explanation for this might be as follows: there are lots of unique words in the file, arising from poor cleanup of marked up text. Examples include "state,\\\\\\" "shot.\\" "\\\"Always" "party!\\\\\\" "Sit.\\" "hide,\\" "\\\\\\\"Prince" "\\\\\\\"Tick" "\\\\\\\"A\\\\\\\" "\\\\\\\"Exactly,\\\\\\" ... There's actually a vast number of these, indicating that our cleanup scripts are broken. (XXX TBD Fix these. It appears that they have something to do with quotations.) Anyway, such words will be observed only a few times, in fixed combinations, and thus have a naturally high MI. Yet they are effectively garbage. When parsed via MPG, they appear only once, and thus become singletons, and get trimmed away. Untrimmed, they inflate the MI.

After trimming, these words with these backslashes are mostly absent.

Expt-6 – Similarity – Sept 2021

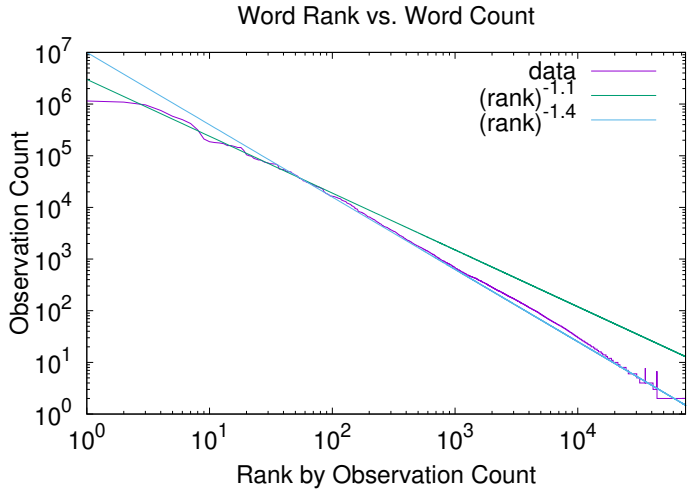
OK, so, finally onto similarity. The '`experiments/run-6`' directory contains scripts to compute similarity, although the actual scripts are in '`learn-lang-diary/utils/similarity-jacc.scm`'. Three similarity measures are computed: conventional MI, as before, overlap, and conditional-Jaccard. See diary Part Two for the definitions of these.

Even the most casual peek at the MI data shows that its pretty darned good, and its not obvious that overlap and cond-jacc are actually better. Interestingly, the self-MI of words are all over the place, and sometimes quite high, in the 7 to 11 range for the top most frequent 300 words, or even extremely high, in the 20-32 range for the least-frequently observed words. Many (most) infrequently-observed words have an MI of 31.809. This might be due to the fact that these words appear to have only one disjunct,

grand-total?! This needs to be checked, and it suggests that perhaps a further trim step should be taken: get rid of words with only one disjunct on them! One the other hand, they might be idioms or set phrases. Are they? Lets look. But first, a characterization of the basic dataset.

Dataset generics

We are using the '`run-1-t1234-trim-1-1-1.rdb`' dataset. It has 75431 words in it. Datasets generated with code in '`utils/similarity-graphs.scm`'. First up: rank vs observation count. This is the classic Zipf graph.

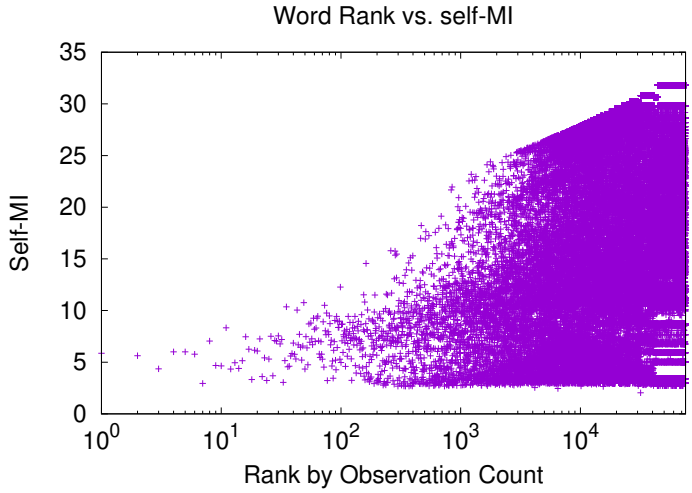


The above graphs the classic Zipf distribution graph, although the distribution is not quite classic: it seems to have two regimes: the first 80 or 90 words follow a $rank^{-1.1}$ distribution, which is more-or-less the classic slope, while later words follow a $rank^{-1.4}$ slope, which is considerably steeper than conventional Zipf distributions. I don't think I've seen this before. Is this due to trimming?

Self-MI

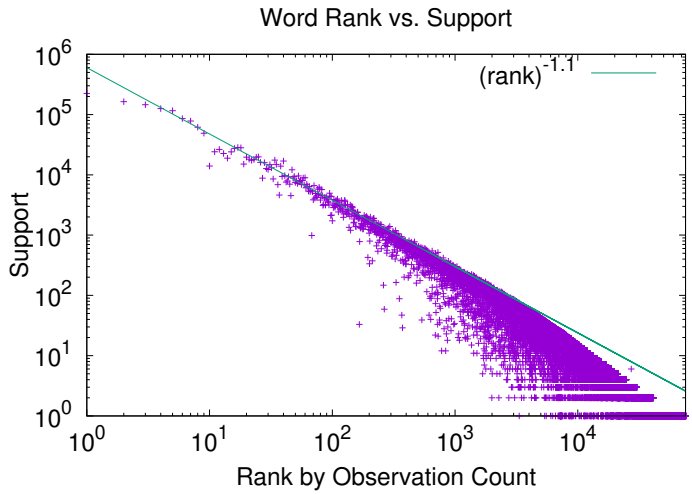
Lets look at the structure of Self-MI. Self-MI is defined as the MI between a word and itself.

Here is a scatterplot of the rank vs self-MI.



It was noted that some words have a very high self-MI value, and it was hypothesized that these are words with a low number of disjuncts. The scatterplot show that there's no particularly clear self-MI vs observation-count-rank correlation. The above can be re-graphed with a ranking taken from the word-vector support size; it does not change substantially. Looks nearly exactly the same.

Lets explore further. Support vs. rank-by-observation-count.



The "support" of a word-vector is the number of disjuncts it has on it. This does have the expected shape: the fewer observation counts on a word, the fewer disjuncts on it, too. It even seems to follow the classic Zipf slope.

So, lets take a look at some specific cases, those with a low count, but with very high or very low MI. Here's a table of words with an MI of greater than 25 and a rank of greater than 3000:

rank	count	support	word	self-MI
3011	170	85	units	25.399
3154	160	80	aircraft	25.398
3185	158	73	Seigneur	25.216
3312	150	73	antenna	25.433
3349	148	69	Battalion	25.416

And again, with a rank greater than 12000 and MI greater than 28:

rank	count	support	word	self-MI
12127	22	11	Sauch	28.349
12129	22	11	trails	28.349
12132	22	11	contractors	28.349
12134	22	11	fishermen	28.349
12139	22	11	villagers	28.349

For comparison, words with a very low MI, in the same range:

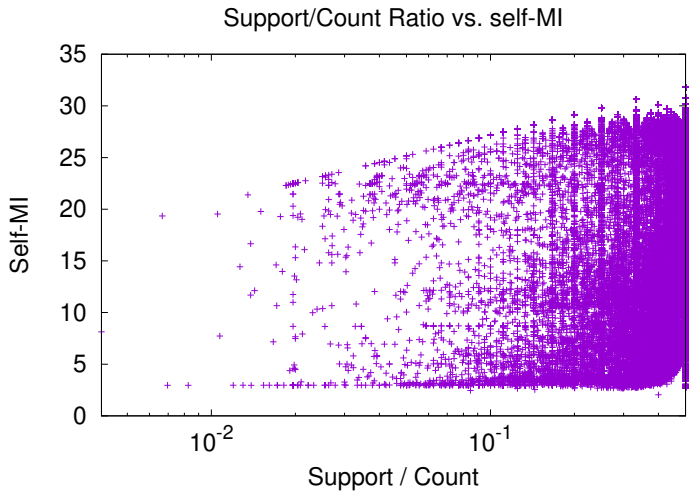
rank	count	support	word	self-MI
3020	170	25	Sch	2.9911
3043	168	13	3s	2.9844
3058	167	2	Scribner	2.9647
3141	161	57	destruction	2.8399
3178	159	51	wisdom	2.9809

And again, with a rank greater than 12000:

rank	count	support	word	self-MI
12122	23	2	810w	2.9878
12563	22	4	CONCLUSION	2.8572
12848	21	2	1140w	2.9987
12859	21	1	Wessels	2.9627
12861	21	1	Leipzig	2.9627

Comparing the tables, we see that, for the same rank (same observation count), the low-MI words have a much smaller support than the high-MI words. A low support/count ratio suggests idioms or set phrases: they may have been counted a lot, but in only a few contexts. Yet, this result is counter-intuitive: shouldn't idiomatic, set phrases have a high self-MI, as the contexts are not shared?

The below is a scatterplot of MI vs. the ratio of support to observation count. There is no obvious correlation, other than a cap on the highest possible MI for a fixed ratio, and some sampling under-observations when the support-count ratio approaches 1.0.



The lack of correlations in the above graph seems to make it hard to say anything interesting about set phrases. Lets look at some details.

- Both Leipzig and Wessels appear to be in single-word sentences (a connector to LEFT-WALL to left, and a period on the right). So, in some list or index or table. Likewise for 810w and 1140w.
- The word CONCLUSION appears in four sentences:
  - CONCLUSION.
  - CHAPTER CONCLUSION.
  - RECAPITULATION AND CONCLUSION.
  - CONCLUSION (without the period at the end)
- The word Scribner appears by itself, and as "New York: Scribner, Armstrong Co." So, clearly a set phrase.

But how about "destruction" which has a very low self-MI but a support of 57, and "fishermen" which has a very high MI but low support?

- Most of the disjuncts on the word "destruction" are quite large: 4 or 5 or 6 connectors. Many of the connectors are prepositions. Almost all connector sequences have a determiner in them. Almost all sections (disjuncts) have a count of two. We conclude: no particular grammatical structure seems to be emerging. This word is not mergeable with others.
- The disjuncts on the word "fishermen" are crazy-large: the lengths are (as sampled): 9 5 7 7 10 7 10 6 3 3 3. All of these sections were sampled twice.

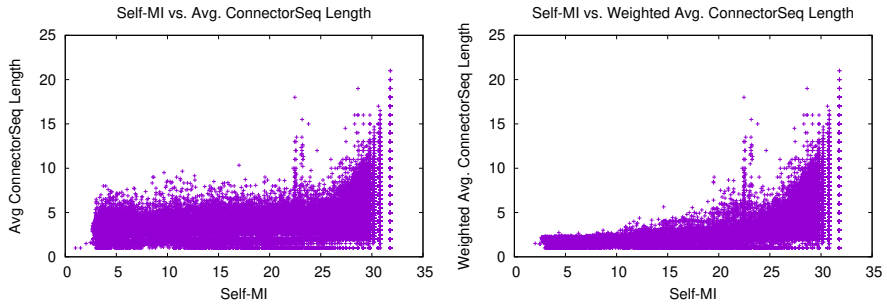
Comparing "destruction" to "fishermen" now reveals the origin for the extremely large self-MI value: the long connector sequences mean that the contexts are extremely specific, and effectively no other words will appear in these contexts.

Is this correct? Lets see...

- "fishermen" – all of the connector sequences on "fishermen" have a support of one!
- "destruction" – of the 57 connector sequences on "destruction", one has a support of 2385, and 13 have a support of more than ten. There are 38 connector sequences with a support of just one.

Yikes! The above reveals that the trimming was too conservative. The trim should remove all connector sequences with a support of one! Before we get too far, maybe we should do this. But first, some additional characterizations of the existing dataset, so that we can see how trimming affects it.

What is the scatterplot of the MI vs the average connector sequence length? How about the weighted avg sequence length? Shown below.



Given a fixed word w, the average connector sequence length is give by

$$\langle \text{len} d \rangle = \frac{\sum_d I(w,d)\,\text{len} d}{\sum_d I(w,d)}$$

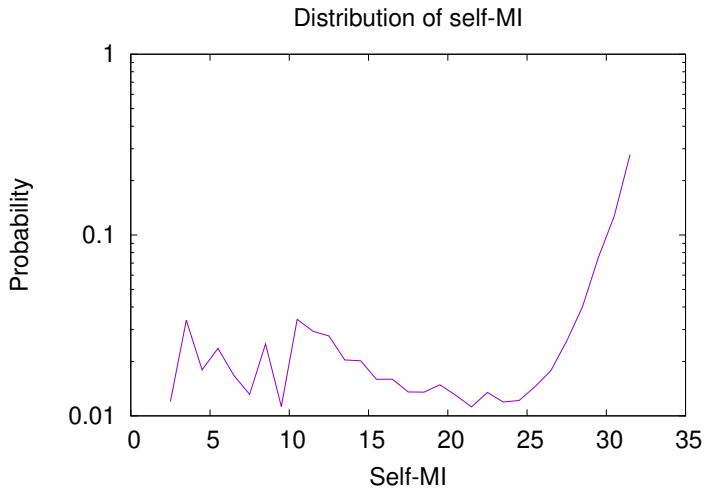
where $I(w,d)$ is the indicator function, equal to 1 if the word-disjunct pair (w,d) exists, else equal to zero. The len d is the number of connectors in the connector sequence (aka disjunct) d. The weighted average is

$$\langle \text{len} d \rangle_{\text{weighted}} = \frac{\sum_d I(w,d)\,\text{len} d \sum_v I(v,d)}{\sum_d I(w,d) \sum_v I(v,d)}$$

so that the weighting is the total number of words in which a disjunct $d$appears in.

This seems to suggest that the commonly used connector sequences, those having a large support, have a small number of connectors in them. The long connector sequences appear to have a small support. Again, this points at trimming away connector sequences that have a small support.

How is the self-MI distributed? Shown below.

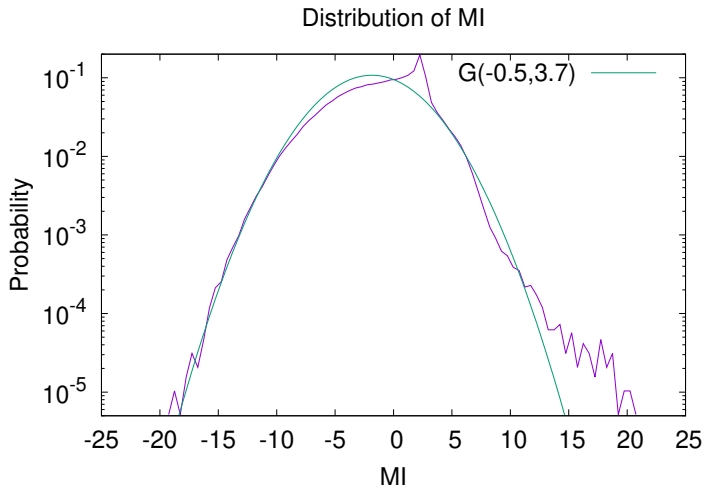


The above shows the probability distribution of the self-MI of the 75K words in the dataset. As can be seen, almost all of these have a high MI. Based on the earlier scatterplots, almost all of these have low observation counts. The high MI is almost surely due to the untrimmed connector-seq support. Ignoring these shows that the self-MI distribution is otherwise rather noisy and quasi-uniform.

Pair distributions

Lets take a quick look at the distributions of the MI, log overlap and log conditional Jaccard of word pairs. The similarity between pairs of the top-ranked 1200 words. In principle, there are $N(N+1)/2 = 720600$ such pairs. In practice, 386380 pairs are observed; the remaining pairs have no overlap! (and thus a $-\infty$ for the three measures).

Here's is the MI distribution:

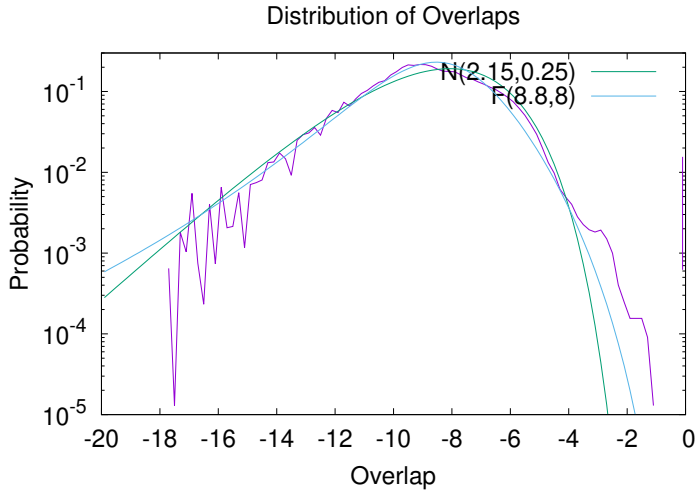


The distribution appears to be an almost perfect Gaussian, centered at $\mu = -0.5$ and stddev $\sigma = 3.7$. That is, the smooth parabola superimposed on the data is

$$G(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

The two parameters are an eyeballed fit. The fact that the MI distributions follow a Gaussian has been noted before.

Distribution of overlaps is shown below.



Two eyeballed fits are shown. One is to the log normal distribution

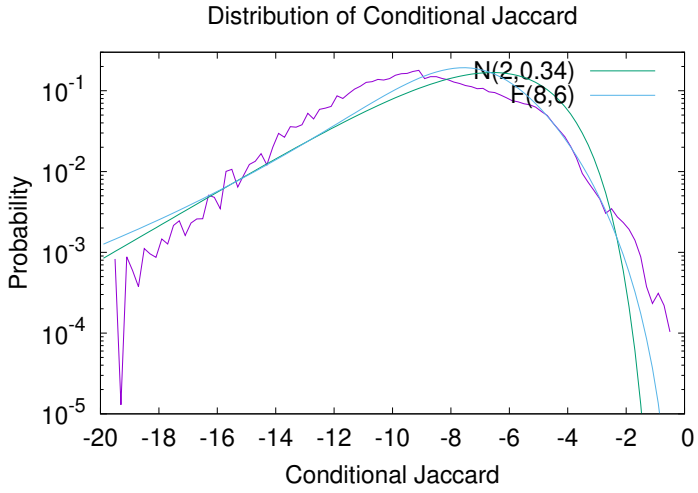
$$N(x;\mu,\sigma) = \frac{1}{x\sigma\sqrt{2\pi}} \exp\left(-\frac{(\ln x-\mu)^2}{2\sigma^2}\right)$$

with $\mu = 2.15$ and $\sigma = 0.25$. Note that the normal peak occurs at and overlap of $\exp\mu = 8.6$. The other curve is an eyeballed fit to the log logistic distribution

$$F(x;\alpha,\beta) = \frac{\beta\,(x/\alpha)^{\beta-1}}{\alpha\left(1+(x/\alpha)^{\beta}\right)^2}$$

with $\alpha = 8$ and $\beta = 8.8$. Both curves seem to provide reasonable descriptions of the data, and I know of no theoretical reason to prefer one or the other.

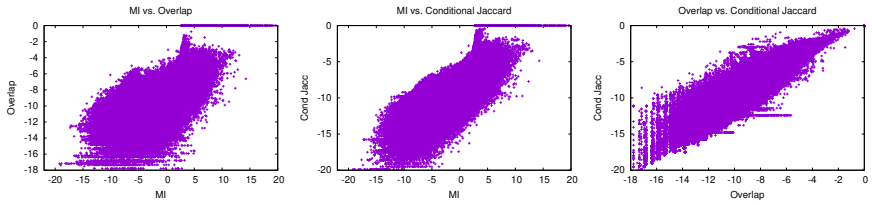
Distribution of the Conditional Jaccard is shown below.



Two fits are shown; this time, neither is very convincing, as they get either one end wrong, or the other, as parameters are adjusted. The log normal fit shown is for $\mu = 2$ and $\sigma = 0.32$ while the log logistic curve is for $\alpha = 6$ and $\beta = 8$.

MI vs. Overlap vs. CondJacc scatterplots

These should be correlated. What do they look like? Below.



Well, yes, all three are roughly correlated. Roughly; the clouds are quite fat. They're not very good proxies for each other. Note that the projections of these distributions to the x and y axes give the three distributions given in the subsection immediately previous.

Note that MI below 5, and overlap, condjacc below -4 mean that the two words of the word-pair are very different from one-another. They're junk, for the purposes of merging.

MI Inversion

There are some words whose self-MI is lower than the MI to another word! Out of the list of the 700 top-ranked words, there are 135 words whose self-MI is less than the MI to some other word. Out of the 1200 top-ranked words, there are 286 such instances. That's a lot.

Why this matters: There are several merge strategies: one is to merge a pair of words, whenever their MI is above some threshold. A second merge strategy is to compare the pair MI to the two self-MI's, and merge whenever it is close enough. That is, we should compare the MI of the word-pair to the average of the two self-MI's, and accept for merge if that is high enough.

Lets look at a few cases.

Here, the self-MI of "I" is lower than the MI of "I" to four other words.

		MI			MI
I	I'll	5.0460	I'll	I'll	13.395
I	You	4.6618	You	You	8.3167
I	We	4.5361	We	We	8.0321
I	I've	4.3938	I've	I've	12.304
I	I	4.3380			

Note all of these are capitalized. Note the single-quote is a UTF8 tilted quote. The characteristic here is that the self-MI of "I" is fairly low, and that it has an MI that is higher to four other words, all of which have a high self-MI. Its as if these other words pull up the MI.

The above table is awkward. Lets try a matrix.

	I'll	You	We	I've	I
I'll	13.395				
You	7.4164	8.3167			
We	7.2146	7.7200	8.0321		
I've	5.9883	4.8425	6.3143	12.304	
I	5.0460	4.6618	4.5361	4.3938	4.3380

This matrix shows that in this cluster, it is only the word "I" that has the low self-MI. None of the others do.

A few more. Here, "will" has a low self-MI.

		MI			MI
will	cannot	6.3204	cannot	cannot	8.3846
will	may	6.2399	may	may	7.2852
will	must	6.2347	must	must	7.2428
will	would	6.0957	would	would	7.0066
will	can	5.9624	can	can	7.6600
will	can't	5.8143	can't	can't	8.8750
will	will	5.7761			

Let's do that again, in matrix form:

	cannot	may	must	would	can	can't	will
cannot	8.3846						
may	6.9085	7.2852					
must	7.0241	6.9907	7.2428				
would	6.5425	6.8959	6.9021	7.0066			
can	7.2619	6.2926	6.4708	6.0848	7.6600		
can't	8.2076	5.9178	6.5754	5.5990	7.5074	8.8750	
will	6.3204	6.2399	6.2347	6.0957	5.9624	5.8143	5.7761

Of all the word-pairs above, only "will" has the self-MI inversion.

Here's another one: The inversion sequence is straight, down, forth, forward, up, out, off, back.

	straight	down	forth	forward	up	out	off	back
straight	10.18							
down	8.102	9.431						
forth	8.127	7.732	13.34					
forward	7.563	7.264	8.004	10.57				
up	7.922	8.372	7.608	7.433	8.789			
out	7.556	7.246	8.251	7.412	7.382	9.257		
off	8.209	7.080	8.069	7.210	7.213	7.842	9.852	
back	7.061	6.701	6.689	6.662	6.388	6.274	6.099	5.943

What should be done here?

Let's fill in these tables with the difference, the delta:

$$\Delta(w,u) = MI(w,u) - \frac{1}{2}MI(w,w) - \frac{1}{2}MI(u,u)$$

Why the average, instead of the geometric mean? Well, MI has a logarithm in it, and it gives the geometric mean after consolidation:

$$\begin{aligned}
\Delta(w,u) &= \log_2 \frac{f(w,u)}{f(w)f(u)} - \frac{1}{2}\log_2 \frac{f(w,w)}{f(w)f(w)} - \frac{1}{2}\log_2 \frac{f(u,u)}{f(u)f(u)} \\
&= \log_2 \frac{f(w,u)}{f(w)f(u)} - \frac{1}{2}\log_2 \frac{f(w,w)}{f(w)f(w)} \frac{f(u,u)}{f(u)f(u)} \\
&= \log_2 \frac{f(w,u)}{f(w)f(u)} - \log_2 \frac{\sqrt{f(w,w)f(u,u)}}{f(w)f(u)} \\
&= \log_2 \frac{f(w,u)}{\sqrt{f(w,w)f(u,u)}}
\end{aligned}$$

Well, that's .. interesting. So, fill it in. BTW, caution with the definitions of the above. As before, the joint probabilities are sums over the normalized word-disjunct probabilities:

$$f(w,u) = \sum_d P(w,d)\,P(u,d)$$

and that, in the above,

$$f(w) = \sum_d P(w,d)\,P(*,d) = f(w,*)$$

Below the diagonal, as before. Above the diagonal is the delta.

	I'll	You	We	I've	I
I'll	13.395	-3.440	-3.499	-6.861	-3.821
You	7.4164	8.3167	-0.454	-5.468	-1.665
We	7.2146	7.7200	8.0321	-3.854	-1.649
I've	5.9883	4.8425	6.3143	12.304	-3.927
I	5.0460	4.6618	4.5361	4.3938	4.3380

Hmm. So anything with "I'll" is deeply negative, because the self-MI of "I'll" is so high. Should we use Δ as the merge-recommendation value?

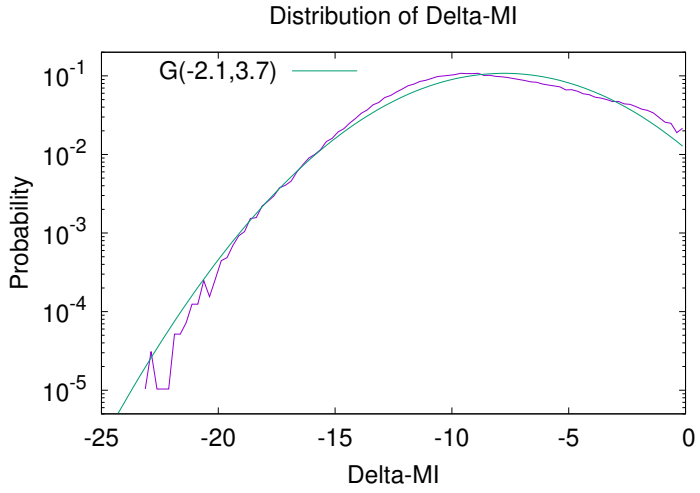
Another

	cannot	may	must	would	can	can't	will
cannot	8.3846	-0.926	-0.790	-1.153	-0.760	-0.422	-0.760
may	6.9085	7.2852	-0.273	-0.250	-1.180	-2.162	-0.291
must	7.0241	6.9907	7.2428	-0.223	-0.981	-1.483	-0.275
would	6.5425	6.8959	6.9021	7.0066	-1.249	-2.342	-0.296
can	7.2619	6.2926	6.4708	6.0848	7.6600	-0.760	-0.756
can't	8.2076	5.9178	6.5754	5.5990	7.5074	8.8750	-1.511
will	6.3204	6.2399	6.2347	6.0957	5.9624	5.8143	5.7761

These are pretty mildly behaved.

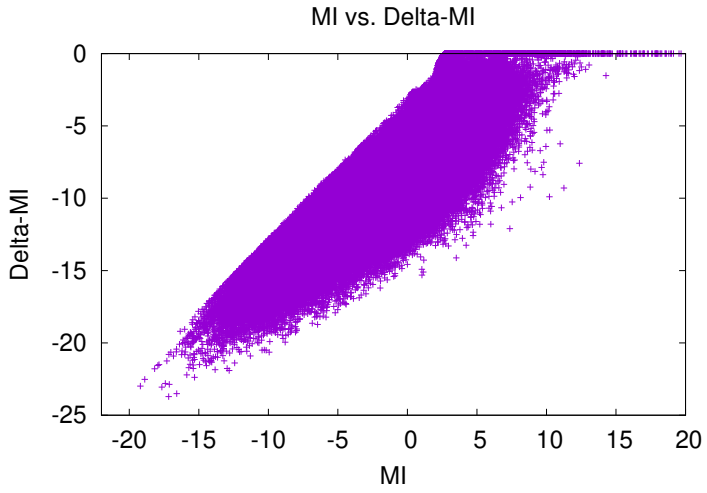
Distribution of Delta-MI

Here's the distribution of Delta-MI:



Compared to the distribution of the MI, given further up, this has exactly the same width, just a different offset. The distribution is smoother: the little prong is missing. There are no Δ values that are positive, thus, although the above was fit with a Gaussian normal distribution, this cannot be the theoretically-correct distribution, because of that cutoff. The theoretical reasons and implications of normal distributions for MI remain unknown to me.

How does this compare to MI? A scatterplot:



Well ... we see that it performs a kind of clamping. High MI values seem to be clamped to a maximum of 0.0. Many, but not all of the $\Delta = 0$ word-pairs are self-pairs (a word against itself).

Word-pair Rankings

Lets look at some top-ten lists of word pairs. Oh no! Inhabiting the top of the pops are word-pairs where the words have low observation counts. That's not nice, or, at least, not very informative. So lets re-rank taking into account the average log probability of seeing each word. Well, that leads to a curious situation. Let

$$\text{common}sim(w,u) = sim(w,u) + \frac{1}{2}\log_2 f(w) + \frac{1}{2}\log_2 f(u)$$

For the *MI* this leads to

$$\text{common}MI(w,u) = \log_2 \frac{f(w,u)}{\sqrt{f(w)f(u)}}$$

which is .. also curious. Oh, hang on... what do we mean by $f(w)$? Because this can be interpreted two different ways:

$$f(w) = f(w,*) = \sum_d P(w,d)P(*,d)$$

which is appropriate for MI. But there is also the plain-old marginal

$$p(w) = \sum_d P(w,d)$$

which is ... a direct count of how often a word is observed. From here on out, we'll try to use lower-case p for the word-disjunct marginal. (This is OK, since it's normalized: $p(*) = 1$.)

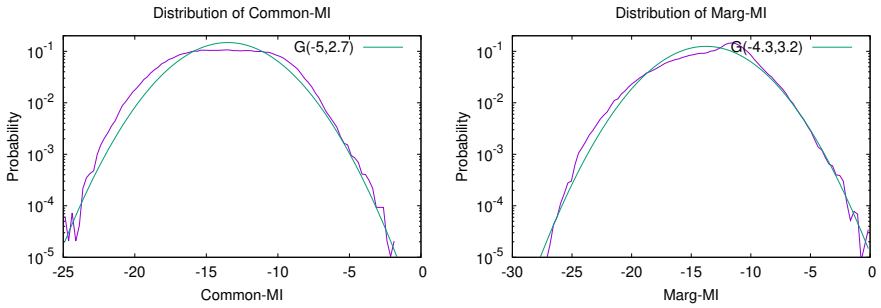
This raises yet more questions. Define

$$\text{marg}sim(w,u) = sim(w,u) + \frac{1}{2}\log_2 p(w) + \frac{1}{2}\log_2 p(u)$$

Lets take some samples. The top-ten word pairs:

common-MI		marg MI	
is	was	Old	New
It	He	i	ii
she	he	hundred	thousand
i	ii	;	,
and	but	her	his
well	soon	ii	iii
Old	New	His	Her
her	his	the	a
It	There	is	was
good	great	down	up

Looks pretty healthy, and there's some commonality. Some distributions:



Both look approximately normal. Common-MI is curiously flat-topped. Marg-MI retains a bit of the horn that the MI distribution has. Perhaps that horn is what makes Common-MI flat-topped.

Anyway, either of these seems like a good way to chose which words to rank. Marg-MI might be slightly preferable, as it seems to give slightly higher priority to frequent words (?).

How about the other two?

marg-overlap		marg-condjacc	
the	a	the	a
the	his	she	he
and	,	the	his
he	I	and	but
she	he	;	,
is	was	is	was
of	,	it	he
it	he	he	I
her	his	her	his
to	,	She	He

This does not seem as high-quality as either of the MI pair lists. the of-comma and the to-comma pairing is disturbing. The the-his is also disturbing. And these are in the top-ten! Yow! What's up with that?

Lets look at the top-100 from common-MI and see how that works out.

common-MI			common-MI			common-MI		
1	is	was	21	We	They	41	the	a
2	It	He	22	;	,	42	there	it
3	she	he	23	well	long	43	and	that
4	i	ii	24	my	his	44	and	so
5	and	but	25	him	me	45	he	I
6	well	soon	26	would	must	46	In	But
7	Old	New	27	would	may	47	”	’
8	her	his	28	would	will	48	as	but
9	It	There	29	His	Her	49	we	they
10	good	great	30	would	should	50	ii	iii
11	‘	“	31	:	,	51	far	much
12	This	It	32	down	up	52	You	We
13	She	It	33	old	young	53	far	long
14	....	–	34	didn’t	don’t	54	him	them
15	well	far	35	had	has	55	ii	v
16	and	as	36	not	never	56	do	did
17	it	he	37	and	for	57	hundred	thousand
18	well	much	38	the	this	58	You	They
19	She	He	39	There	He	59	would	might
20	don’t	don’t	40	This	He	60	He	I

There’s not much to object to, here. Maybe 39 and 40. How about marg-MI?

marg-MI			marg-MI			marg-MI		
1	Old	New	21	ii	v	41	us	him
2	i	ii	22	i	iii	42	Her	Their
3	hundred	thousand	23	up	out	43	the	our
4	;	,	24	more	less	44	out	off
5	her	his	25	.	?	45	don’t	don’t
6	ii	iii	26	him	them	46	United	several
7	His	Her	27	fifty	twenty	47	sighed	nodded
8	the	a	28	—	+	48	iii	v
9	is	was	29	their	the	49	old	young
10	down	up	30	her	him	50	Chapter	Footnote
11	my	his	31	old	ago	51	us	me
12	His	The	32	longer	more	52	down	out
13	et	à	33	the	this	53	their	its
14	FIG	Fig	34	good	great	54	we	they
15	:	,	35	etc	&c	55	away	out
16	him	me	36	The	Their	56	ten	twenty
17	had	has	37	thirty	twenty	57	his	its
18	His	Their	38	your	his	58	i	v
19	their	his	39	!	.	59	me	them
20	the	his	40	the	an	60	said	replied

There's one screwy item in this list: United-several at position 46. It also notably elevates common Project Gutenberg markup: lower-case roman, FIG-Fig, Chapter-Footnote but otherwise looks pretty healthy.

Looking at the top-100 marg-overlap entries, these look pretty healthy, except for five more questionable comma-preposition suggestions! The top-100 cond-jacc looks reasonable, too, except for some oddball determiner-pronoun similarities.

Overall, common-MI really looks best overall! I might be biased. It's hard to be scientific here.

TODO Items

Other things:

- Explore the word "idea" and everything it is similar to!!! Wow!
- Graph of similarity cutoff vs number of things at are similar at that cutoff - ranked.
- Scatterplot: rank of word vs. number of disjuncts on it
- Histogram of length of disjuncts.

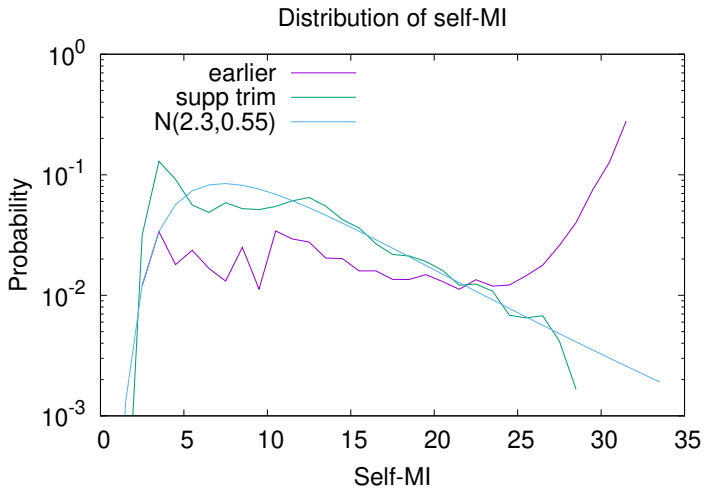
Important Hypothesis

One important task: Compute the marginal MI of individual words, and rank those. The hypothesis is that words with low marginal MI are "filler words", not "function words": they are syntactic place-holders but are secondary in the transmission of meaning. The function words presumably carry most of the meaning. Is this true? Can we look?

Correctly trimmed similarity

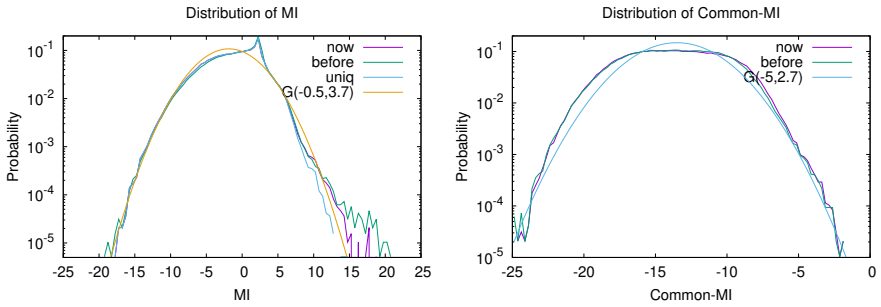
Lets redo the similarity calculations, after trimming away all disjuncts, words with a support of one. As illustrated in table way above, this is a *huge* trim, removing most words and disjuncts. So copy '`run-1-t1234-tsup-1-1-1.rdb`' to '`r6-similarity-tsup.rdb`' and compute MI pairs for top-ranked words. Computation is a *lot* faster. There are 15083 words in this set ... still too many for computing all possible pairs ... but far more manageable. A skim of these reveals that a large fraction contain backslashes, *i.e.* are still polluted by bad tokenization.

One of the "bad charts" above was the self-MI. Words with only a single disjunct on them had a huge self-MI. Upon trimming, this goes away, as shown below:



This time, there is a clear downward trend. It can be fit with a log-normal distribution, the formula given previously above, with $\mu = 2.3$ and $\sigma = 0.55$. Recall that the "N" label in the graph is a shorthand for the log-normal distribution, the formula for which is given previously.

Of the $N(N+1)/2 = 720600$ possible pairs for the top $N = 1200$ words, a total of 398431 are observed. This is a bit more than the earlier dataset. The pair distribution is the same as before, and only the high-MI excess has been removed. Three data curves are shown: "now", "before" and "uniq". The "before" curve is as before, and the "now" curve is for the current dataset. The "uniq" curve repeats the the current dataset, but with all self-MI pairs removed. This makes it especially clear that the goings-on at the high MI end of the curve is entirely due to the high-self-MI of some words.



The Common-MI graph from before is nearly identical to the one from before. No comment.

Expt-7 – Merging – Sept 2021

Time to open a new chapter. We are ready to merge word-pairs, with a mostly-trustworthy dataset. The goal here will be to merge just one word at a time, by hand,

and see what happens. We're doing this because of the disastrously bad July 2021 merge attempt, which generated complete garbage, quite in contrast to the merges from previous years.

Part of the reason for the failure of the July merge was that the merge threshold was set much too low. Part of the problem is that the datasets had too much grunge in them; they needed to be trimmed. Part of the problem is that the default merge used a proxy for the common-MI value, based on ranking. Probably most of the failure can be ascribed to the first part: the merge threshold was set too low.

We've never studied the merges carefully, before. Mostly, they seemed to "just work", and we unleashed the machinery, and let it go. The first time, it worked, the second time, it failed. Anyway, now that we've got a better dataset, its worth looking at these in detail.

This work resumes in the Diary Part Four.

The End

This is the end of the diary. The next part is Part Four.