Grammar Evaluation Results

June 2019 Mini Dictionaries
ULL Dictionaries
July 2019 Info Dictionaries

Linas Vepstas

Version of 30 June 2019

This report presents measurements of the quality of various different dictionaries learned from two different language-learning pipelines; one is the ULL/Kolonin variant, the other is the Linas variant. The ULL/Kolonin dictionaries suggest two major breakthroughs: they suggest how the learning pipeline should be tuned, and they indicate how the pipeline is relatively insensitive to early stages of processing. The Linas variant results indicate that the sheer quantity of training data has a strong impact on the quality of the learned grammars.

This report briefly reviews the dictionaries, the algorithms used to obtain them, and measurement results. It is assumed that the reader has a general familiarity with the project.

1 Baseline Measurements

This provides the introduction and the baseline to the algorithm, the measurements, and the results.

1.1 Introduction

The language learning project attempts to extract symbolic grammars from a sampling of raw text. The general process is conceptually straight-forward:

1. Obtain provisional parses for a (very) large quantity of text.
2. Chop up up the resulting parses into individual words, with co-occurring connectors. That is, each word-instance in the text is associated with a set of connectors pointing at the other words it connected to in that particular parse. Thus, one has a word-instance, and a connector-set extracted from the parse in the first step. These connector-sets are variously referred to as "disjuncts" or "germs" in related texts.
3. Treat connector-sets as the basis vectors in a vector space. Thus, a word is actually a collection of (word, connector-set) pairs, with an associated count of the number of times that particular (word, connector-set) was seen during parsing.
4. The vector representation allows different kinds of word-similarity judgments to be used, and thus allows words to be clustered into classes. These classes should be called "grammatical classes", as all of the members of that class behave in a grammatically-similar fashion.
5. The grand-total collection of grammatical classes forms a dictionary or a lexis or a "grammar"; this dictionary is now a valid symbolic description of grammar, in that it can be use to parse new, previously-unseen sentences, and extract their syntactic structure, as well as some fair amount of semantic content.

The language-learning project contains two distinct implementations of the above pipeline. The first implementation, tightly coupled to the OpenCog AtomSpace, was created by Linas (the author), but put on hold as various other intervening priorities interceded. Development of that pipeline has recently resumed. A second implementation, termed "the ULL Project" was subsequently developed by a team lead by Anton Kolonin, including Andres Suarez Madrigal and Alexei Glushchenko. It is partly derived from the first implementation, but diverges in many important ways. In particular, it eschews the use of the AtomSpace, replacing it by a collection of ad-hoc text-file data formats and Python tools.

This paper reports on early results from these two different systems. It is structured as follows. The first several sections review the status and datasets used by the author, in his own pipeline. As such, this becomes a bit like a laboratory notebook diary, making notes of datasets in possession of the author, their names, their status, their general characteristics. The next section reviews some very early dictionaries that were produced by the author. They are highly preliminary: many important processing stages have not been implemented, various shortcuts and hacks were applied to the stages that do exist, and the evaluated dictionaries are just tiny.

The last section reviews the ULL datasets. These suggest not one, but two extremely important breakthroughs for the project. The first is that it seems like a method for tuning the learning pipeline has been discovered, as well as an objective measure for the fidelity of the pipeline. In short, it seems that there is a way of validating that, when given a fixed grammar, the pipeline preserves the structure of that grammar, despite the steps 2-through-5 above. That is, it seems that steps 2 through 5, when performed with some care and diligence, preserve the structure of a grammar, and do not wreck it, nor do they alter it into something else. This is still a preliminary result, and needs additional validation. But if it holds true, it is extremely important, as it allows the quality of the 2-through-5 pipeline to be measured and tuned. Once tuned, the pipeline can be trusted: whatever grammar goes in, that is the grammar that comes out. Thus, when an unknown grammar is put in, then whatever comes out must be correct.

A second breakthrough from the ULL datasets is that the resulting grammar is relatively insensitive to step 1. As long as the provisional parses have some accuracy above random chance, then, by accumulating enough samples, the errors in the provisional parses will cancel out. This is as it should be in radio receivers, or any kind of statistical sampling: The bigger the sample, the better the signal-to-noise ratio. Although the provisional parses of step 1 are noisy and often incorrect, they are good enough, when enough samples are collected.

There are several minor results worth mentioning. From the ULL datasets, it is clear that the Project Gutenberg tests provide an inadequate sample of modern English. From the Linas dictionaries, its clear that large sample size really makes a difference; the Linas dictionaries, although tiny, inadequate and hacked, trounce the ULL datasets when measured out-of-training-set.

One last result is worth mentioning: an "unknown word" system, described below, appears to be quite effective in offering broad coverage when new, unknown words are encountered in the test texts. Due to Zipfian distributions and long tails, the overlap of the test-vocabulary with the dictionary vocabulary is shockingly tiny. Thus, an unknown-word mechanism is critical for providing reasonable coverage.

1.2 June 2019 Restart

Work by the author was suspended in the Fall of 2018, and was restarted in June 2019. This is an attempt to pick up where things got left off. It summarizes the grammar learning dataset, the algorithms and some preliminary results, as it stands at the moment.

- The primary dataset, `en_dj_cfive,` appears to be in good condition and should be usable for a good long while, before updates are needed.

- The algorithms are incomplete; much work remains to be done. Work on the core ideas, such as the "sheaf" idea, has not yet started in earnest, although some scaffolding has been laid.
- Existing code is sufficient to generate word-classes, word-sense disambiguation and grammars.

An eyeball inspection of the word-classes suggests that they are quite healthy, but there are no automated tools to verify their quality. The clustering seems to be able to split words into word-senses, placing them into correct clusters. The assignments look reasonable, when judged by eye, but there are no automated measurements. These last two results were previously reported in detail in the Winter of 2017 and Spring of 2018. This report focuses on the resulting grammars that can currently be generated, and an automated process to measure their quality.

To repeat: these are highly preliminary, as most of the important parts of the grammar generation mechanism have not yet been implemented. The results here are a sniff-test or proof-of-concept, to show that the idea works at some minimal level.

1.3 Baseline Dataset

The baseline is the `en_dj_cfive` dataset. Let's recall how this dataset was obtained:

1. All five "tranches" of the training corpora were used to obtain word-pair MI scores. Unlike the earlier *en_rfive* datasets, this uses a bug-fixed tokenizer.
2. All five (the same five) tranches were run through the strict-MST parser, to obtain word-disjunct pairs. The disjunct connectors are all single-word connectors. The MST parser was "strict", in that it always created a tree (no loops) that always connected every word in a sentence. Each word-disjunct pair has an associated count, which is the number of times it was seen during MST parsing.

That's it. Starting with this dataset, various subsets were created:

- **en_micro_marg** – Keep only words with observation count>500, sections with count>10; discard all sections that contain connectors that cannot link to anything.
- **en_mini_marg** – Keep only words with observation count>40, sections with count>5; discard all sections that contain connectors that cannot link to anything.
- **en_large_marg** – Keep only words with observation count>8, sections with count>3; discard all sections that contain connectors that cannot link to anything.
- **en_huge_marg** – Keep only words with observation count>3, sections with count>1; discard all sections that contain connectors that cannot link to anything.
- **en_full_marg** – Discard all sections that contain connectors that cannot link to anything. (todo: rename to full-marg)

Recall that each dataset can be thought of as a (very) sparse matrix. Rows of the matrix are words; columns of the matrix are disjuncts. Each entry in the matrix is a word-disjunct pair; it is called a "Section" (a SectionLink in the AtomSpace). Below are the stats from the `(print-matrix-summary-report)` function.

Size	Secs	Obs'ns	Obs/sec	Sparsity	Entropy	MI	Dataset
1610 x 67K	184K	14.7M	80.3	9.20	15.17	4.60	en_micro_marg
7385 x 270K	608K	20.8M	34.2	11.67	16.79	4.96	en_mini_marg
17K x 947K	1.56M	26.2M	16.7	13.34	18.08	5.16	en_large_marg
55K x 6.03M	7.91M	42.3M	5.35	15.35	20.69	5.89	en_huge_marg
438K x 23.3M	31.7M	69.2M	2.18	18.30	23.08	8.15	en_full_marg
445K x 23.4M	31.9M	69.4M	2.18	18.32	23.09	8.16	en_dj_cfive

The columns are:

- **Size** – Dimensions of the matrix. Number of words x Number of disjuncts. *Notation:* $|w| \times |d|$.
- **Secs** – Total number of Sections. That is, total number of word-disjunct pairs in the matrix. *Notation:* $|(w,d)|$ (*sometimes written as* $|(*,*)|$).
- **Obs'ns** – Total number of observations of Sections. Sum total of how many times the sections were observed. *Notation*: $N(*,*)$.
- **Obs/sec** – The average number of observations per Section.
- **Sparsity** – The log (base two) of the fraction of non-zero entries in the matrix. That is, $\log_2 (|w| \times |d| / |(w,d)|)$ where $|(w,d)|$ is the number from the second column.
- **Entropy** – As usual for a matrix: $-\sum_{w,d} p(w,d) \log_2 p(w,d)$. Note that the log-base-two means this is in bits. Here, (w,d) are word-disjunct pairs, *i.e.* sections. The probability is actually a frequency: $p(w,d) = N(w,d)/N(*,*)$.
- **MI** – As usual for a matrix: $-\sum_{w,d} p(w,d) \log_2 p(w,d) / (p(w,*)\, p(*,d))$.

Note that the MI score drops as words are trimmed from the dataset. All of those infrequently-observed words carry a lot of information, it seems.

These are promising to be a lot cleaner, and stronger than the previous *en_rfive* dataset.

1.4 Grammars

A grammar is obtained by clustering words into grammatical classes. A variety of different clustering strategies and clustering parameters were used to obtain several grammars. See the files `learn/scm/gram-agglo.scm` and `learn/scm/gram-projective.scm` for details about how clustering is done. Results are reported in multiple tables.

First table: summary report for the first four survey sets. The second table is like the first; it just explores more parameter settings. These are just a rough first-cut, and are not expected to be good, for five reasons:

- These use the cosine-distance metric (I expect the MI metric to be much better).
- Parameters are not tuned (some parameters were picked that seemed reasonable to start with; the second table explores some tuning.)
- Clustering is only weakly sensitive to distinct word-senses; a quasi-linear merge of observation counts is used. When a word is placed into a cluster, the counts of any shared disjuncts are transfered to the cluster. This leaves behind a left-over word-vector, containing disjuncts that did not fit into the cluster. This remainder represents a second word-sense for that word; it's part of how WSD happens "automatically" in these algorithms. These left-over bits are either being reclustered, if possible, unless the remaining counts are tiny, in which case they are discarded. A more refined clustering algo would replace this quasi-linear merge with a disjunct-by-disjunct decision, to minimize the accidental mixing of disjuncts that should have been associated with different word-senses.
- The disjunct-shapes are not being clustered ("shapes" are disjuncts with wild-cards in them; see elsewhere for a detailed explanation. I expect that cluster quality will improve when shapes are included.)
- Link-types are being clustered badly; that is, connectors are not being assigned properly to word-classes (*i.e.* the "sheaves" concept is not being used.) The current code over-generalizes: if it finds a link between two words, then it automatically promotes that to a link between all word-classes that those word belong to. This creates linkages between word-senses that should have been kept distinct. It also leads to a combinatoric explostion in the parser. This desperately needs to be fixed.

Solutions to all of these issues are explored in Chapter 3.

All columns in the tables are obtained from the gram-class matrix report. That is, the matrix-rows are now grammatical classes, not words. The matrix columns are still word-disjuncts, not gram-class disjuncts (this is partly related to the "shapes" problem). A correct grammar would use gram-classes in the connectors as well. So again – this is a rough cut.

The table is much like the above:

- **Size** – Number of grammatical class x Number of disjuncts. Note that many of these classes are "singleton classes", containing only one word. See the **WC** and **Sing** columns for a breakout of the number of classes with one and more than one word. **WC+Sing** equals the total.
- **Secs** – Total number of Sections. That is, total number of word-disjunct pairs in the matrix.
- **Obs'ns** – Total number of observations of Sections. How many have been observed.
- **Entropy** – As before, just using gram-classes instead of words.

- **MI** – As before, just using gram-classes instead of words.
- **WC** – Number of grammatical classes with two or more words in them.
- **Sing** – Number of singleton classes – that is, classes with only one word in them.

The table below shows only the basic-baseline datasets. There are more, which explore additional parameters.

Size	Secs	Obs'ns	Entropy	MI	WC	Sing	Dataset
675 x 67K	142K	12.0M	13.75	4.03	135	540	en_micro_fuzz_exp
416 x 67K	133K	11.4M	13.23	3.62	124	292	en_micro_discrim
3692 x 269K	502K	16.6M	15.39	4.35	366	3326	en_mini_fuzz_exp
2468 x 269K	499K	15.5M	14.76	3.89	371	2097	en_mini_discrim

The datasets are:

- **en_micro_fuzz_exp** – Created with `(gram-classify-greedy-fuzz 0.65 0.3 4)`. This dataset does not have LEFT-WALL linkages in it. Created from `en_micro_marg`. Here, the parameter `0.65` is the cosine-distance cutoff; vectors with a cosine distance less than this are not clustered. The parameter `0.3` is the broadening parameter: it means that 30% of the counts of disjuncts NOT already in the cluster are added to the cluster. This makes the cluster boundaries "fuzzy", whence the name.
- **en_micro_discrim** – Created with `(gram-classify-greedy-discrim 0.5 4)`. Casual observation shows the clusters are less accurate than above. The parameter `0.5` is the cosine-distance cutoff. The broadening is not fixed; broadening is accomplished by assigning anywhere from 0% to 100% of the non-shared disjunct counts to the cluster, varying linearly by cosine distance (*e.g.* if the distance is 0.83, then (0.83-0.5)/0.5=66% of the counts will be merged).
- **en_mini_fuzz_exp** – Just like en_micro_fuzz_exp, but with a larger vocabulary, from en_mini_marg.
- **en_mini_discrim** – Just like en_micro_discrim, but with a larger vocabulary, from en_mini_marg.

The table below is much like that above, except that it explores a greater range of learning parameters. It is trying to get a sense of where the sweat spot is.

Size	Secs	Obs'ns	Entropy	MI	WC	Sing	Dataset
680 x 67K	143K	13.2M	14.08	3.95	136	544	en_micro_fuzzier
948 x 67K	155K	12.6M	14.23	4.29	115	833	en_micro_fizz
672 x 67K	142K	12.1M	13.75	4.00	135	537	en_micro_diss
948 x 67K	155K	12.5M	14.19	4.30	115	833	en_micro_dissier
1260 x 67K	168K	13.7M	14.66	4.46	85	1175	en_micro_dissiest
416 x 67K	133K	11.4M	13.23	3.62	124	292	en_micro_rediscrim
91 x 67K	160K	16.5M	13.51	2.79	34	57	en_micro_disinfo
279 x 67K	196K	14.3M	14.27	3.65	70	209	en_micro_disinfo3

The datasets are:

- **en_micro_fuzzier** – Created with `(gram-classify-greedy-fuzz 0.65 0.6 4)`. Note the fuzz parameter is 0.6 not 0.3 as before. This means that when a words is being merged into an existing word-class, it will merge in 60% of the non-shared disjuncts from the new word. This will probably harm word-sense disambiguation. However, WSD probably plays a small role, at this time.
- **en_micro_fizz** – Created with `(gram-classify-greedy-fuzz 0.75 0.3 4)`. The fuzz parameter is the same as for en_micro-fuzz, but the discriminator is tighter – 0.75 instead of 0.65.
- **en_micro_diss** – Created with `(gram-classify-greedy-disc rim 0.65 4)`. Similar to en_micro_discrim, but more discriminating (0.65 instead of 0.5).
- **en_micro_dissier** – Created with `(gram-classify-greedy-disc rim 0.75 4)`. Similar to en_micro_discrim, but more discriminating (0.75 instead of 0.5).
- **en_micro_dissiest** – Created with `(gram-classify-greedy-disc rim 0.85 4)`. Similar to en_micro_discrim, but more discriminating (0.85 instead of 0.5).
- **en_micro_rediscrim** – Created with `(gram-classify-greedy-discrim 0.5 4)`. These are exactly the same parameters as used for *en_micro_discrim*, but after fixing a bug in the algo. The cosines were being calculated incorrectly; the denominator of the cosine (the lengths) were not being recomputed after a merger; thus, all cosine angles were off – maybe by a lot... Despite this, it appears that the number of learned classes is the same as before, ditto the singletons, and the entropy and MI are unchanged! In fact, the matrix-summary report suggests that this dataset is identical to *en_micro_discrim*, which is unexpected, as the bug seemed to be real...
- **en_micro_disinfo** – Created with `(gram-classify-greedy-disinfo 2.0 4)`. This uses the same projective clustering as before, but uses MI instead of cosine for clustering decisions. Note that an MI=2.0 is very low; this placed almost all words into some cluster.
- **en_micro_disinfo3** – Created with `(gram-classify-greedy-disinfo 3.0 4)`. As above, but higher MI cutoff.

1.5 Measurements

There does not appear to be any good way to assess the quality of a grammar. So instead, a trick is used that is obviously flawed, but is "better than nothing". The trick is to compare the grammar to the LG English dictionary. That is, a sentence is parsed, once with the grammar-under-test, and once with the LG English grammar. The parses are compared side-by-side, looking to see if they have the same links between words, or not. This allows both precision and recall to be measured: A linkage is high-precision, if it does not contain link that it shouldn't. A linkage has high recall, if it contains most of the links that it should.

There are obvious problems with this:

- There is no particular reason to believe that the above algorithms will reproduce LG dictionaries. This is a comparison of apples-to-oranges: the learned dictionaries are obtained by apply probability theory to sparse observational data of strings of words. By contrast, the LG dictionary is hand-curated, by linguists applying innate, common-sense theories of grammatical relationships. The resulting linkages are just opinions expressed by linguists as to what English must be like. That there is significant overlap between statistical results and linguist opinion is itself remarkable.
- The LG English dictionary is imperfect, sometimes creating incorrect parses. Thus, it is possible that the grammar-under-test produced a better parse, while still being scored poorly, because it failed to match LG.
- LG parses in general contain loops, i.e. are not trees. The grammar-under-test might also contain loops, but maybe less than LG. Thus, the grammar-under-test will often have a low recall, simply because of how it was built. Using a non-strict MST at earlier stages might improve the situation.
- Some links are more important than others. It is more important to get links to subject and object correct, than it is to get links to punctuation correct. In particular, LG links to punctuation are rather *ad hoc*, especially for the end-of-sentence punctuation. It is unlikely that the grammar-under-test will handle punctuation the same way.
- If there is a combinatorial explosion of LG parses, then the parse-scoring system in LG is partially blinded, and the resulting chosen parses might not be the ones with the best scores. It might be possible to ameliorate this with changes to LG.

Thus, due to the above concerns, the actual meaning of the scores is unclear. Never the less, its a good starting point. Some future version of the measurement tool will keep track of subject and object links, and maybe other link types that are considered important.

Currently, all of the dictionaries-under-test are missing a LEFT-WALL marker. I'm not sure why; this will be revisited later. Thus, LEFT-WALL links are not compared, and are not a part of the evaluation process.

The test-corpora contain many words not in the dictionary, and vice-versa: the dictionaries contain many words not in the test-corpora. The overlap between these two vocabularies is shockingly small (but perhaps entirely normal – see extended commentary about Zipf's law, elsewhere). In particular, the "micro" dictionaries have only 1.6K words in them, and of these, less than a third show up in the test corpora. Thus, we have three testing options:

- Skip the evaluation of a sentence, if it contains words that are not in the dictionary-under-test. The test corpora contain a lot of proper names that never appeared in the training corpora, and this alone accounts for many of the sentence rejections.

- Evaluate the sentence anyway; words not in the dictionary will be null-linked (i.e. have no links going to them.) Mostly all that happens is that precision scores are mostly unaffected, while recall scores are obviously lower.

- Perform unknown-word-guessing. This is not hard to do, and ultimately is a required ability for any dictionary usable in the real world. The current guessing strategy is to try each unknown word with each gram-class, and then let the parse-scoring system pick the one with the highest score. Note that unknown-word guessing leads to a combinatoric explosion in LG, which can cause extremely long parse-times (hours!); see notes below. Combinatoric explosions can also result in sub-optimal choice of linkage. The determination of the pertinence of this effect is TBD.

Two sets of dictionaries were created: with and without unknown words. Results for both are reported below.

Comparisons were performed for five different corpora. These are also an imperfect choice, but adequate for this initial rough-cut evaluations. The are:

- **CDS** – The "Child Directed Speech" corpus from the ULL project. Contains 1837 sentences uttered by parents towards children. These have a characteristically small vocabulary, and a simple grammatical structure. Average length is 5 words per sentence. This corpus consistently results in the highest scores during testing, for all test dictionaries.

- **basic** – The `corpus-basic.batch` file from LG. This contains 1000 sentences, of which 421 are intentionally ungrammatical, leaving 579 valid English sentences. These contain a balanced selection of a wide variety of different kinds of English constructions; it was meant to showcase the rich variety of different sentences that LG can handle. (Note, however, that evolving fixes means that not all of these parse correctly). Most of these sentences are of short-to-medium length.

- **fixes** – The `corpus-fixes.batch` file from LG. This contains 4236 sentences, illustrating fixes to the early versions of the LG dictionary. Although there is a wide variety of different kinds of English constructions, it's not particularly "balanced", and the sentences tend to be short; just long enough to illustrate some phenomenon. Most of these sentences are of short length.

- **gold** – The "Golden Corpus" from the ULL project. This contains 229 sentences. Note that the parses provided by the ULL project are NOT used for evaluation. The LG parses are used instead. Most of these sentences are medium-short.

- **silver** – A subset of the "Silver Corpus" from the ULL project. This contains 2513 sentences, taken from three different Project Gutenberg books: "Kilmeny of the Orchard" by Lucy Maud Montgomery; "Peter Rabbit" by Thornton W Burgess; and a portion of "Under the Lilacs" by Louisa May Alcott. Average sentence length is 12 words.

- **wiki** – A concatenation of three Wikipedia articles: "Autoethnography", "Astor House Hotel (Shanghai)" and "History of Virginia". All of these sentences are long, averaging 24 words in length. The long sentence length can lead to combinatorial explosions in most of the dictionaries, resulting in painfully long parse times.

All five corpora are available in the `learn/run/3-gram-class/test-data` directory.

Measurements were performed with the script `learn/run/3-gram-class/dict-comp.scm` specifying the dictionary to test and the corpus as

```
guile -s learn/run/3-gram-class/dict-comp.scm <dict-to-test> <corpus>
```

The table below shows results. The columns in the table are as follows:

- **Sents** – The total number of sentences in the corpus. This repeats the number given above.
- **Skip** – Percentage of total sentences that were skipped, because they contained unknown words. (Reported only when unknown-word guessing is disabled.)
- **Parsed** – The number of sentences that were evaluated. Equal to number of sentences times (100% - Skip).
- **Diff** – The number of sentences that had parses that differed from the first parse given by the English dictionary in Link Grammar version 5.6.1.
- **Words** – The total number of words in the evaluated sentences. This varies from dictionary to dictionary, depending on whether sentences were skipped or not.
- **Vocab** – The size of the vocabulary in the evaluated sentences; that is, the number of unique words. When rejecting sentences with unknown words, this becomes the intersection of the vocabulary of the accepted sentences and the dictionary-under-test.
- **Links** – The number of links that the LG English parse generated. This varies from dictionary to dictionary, depending on whether sentences were skipped or not.
- **P** – Precision of links: TP/(TP+FP) where TP is the count of the links that both dictionaries generated, and FP is the count of the links that the dictionary-under-test generated, but LG did not.
- **R** – Recall of the links: TP/P where TP as above, and P the total number of links produced by the LG English parse.
- **F1** – The harmonic mean of precision and recall.

Results for the *micro-fuzz* dataset. This is a tiny dataset, with only 1600 words in it; its the first shot at anything even vaguely workable. Not expecting good results. But they seem passable. Most sentences got skipped, because they contained unknown words, but the ones that did parse seem to not be outrageous. Notice the intersection of the test-corpus vocabulary and the dictionary size is small – about 1/3rd of the dictionary size, or less. That is a small vocabulary; so 2/3rds of the dictionary contains words that are not in the test-corpus!

	micro-fuzz									
Dataset	Sents	Skip	Parsed	Diff	Words	Vocab	Links	P	R	F1
CDS	1832	69%	562	90%	2738	176	2369	0.705	0.473	0.566
basic	579	86%	82	99%	666	209	647	0.627	0.371	0.466
fixes	4236	82%	774	97%	4888	589	3770	0.534	0.399	0.457
gold	229	90%	24	92%	169	95	132	0.577	0.485	0.527
silver	2513	85%	377	100%	3323	527	2852	0.545	0.407	0.466

Not all link-types are created equal. Correct links to the subject and object should be considered to be more important, than ambient links to punctuation. Thus, consider placing the LG link-types into four categories: important, less important, and the others. The categories that seemed about right were these:

- **Primary** – the `S O MV SI CV` link types – these point out the subject, object, important modifiers and dependent clauses.
- **Secondary** – the `A A AN B C D E EA G J M MX R` link types – these connect to various modifiers, determiners, relative phrases.
- **Punctuation** – the `X` link type. The LG dictionary has rational rules for how to treat punctuation, but there is no particular reason to think that counting statistics will adhere to this.
- **Other** – all other link types. Yes, they are important, but one might expect that counting statistics might not adhere to the LG dictionary decisions.

Glancing at this table, there is no obvious indication that the important (primary) links are gotten correct more than the other links. Perhaps the `MV` link does not belong in the primary category?

	micro-fuzz							
Dataset	Primary	N	Secondary	N	Punct	N	Other	N
CDS	0.506	1019	0.465	617	-	0	0.434	733
basic	0.387	243	0.317	158	0.2	5	0.394	241
fixes	0.424	1317	0.388	880	0.237	207	0.406	1366
gold	0.483	58	0.447	47	-	0	0.600	25
silver	0.385	1071	0.402	731	0.434	76	0.434	974

Results for *micro-discrim* below. The gram classes are looser (the cutoff is lower - 0.5 instead of 0.65 as above), and casual inspection (*i.e.* just reading the list of words in each) suggests that they are lower quality (from eyeballing, it's clear that many dissimilar words are being classed together). This behaves in a not-too-surprising way: precision drops (more junk links produced), while recall improves (the junk links just happen to go to the right places). Overall F1 score is better, which is surprising. Same base vocab of 1600 words. The values in the Sents, Skip, Parsed, Words, Vocab, Links columns are unchanged, since both this and above are built on the *en_micro_marg* dataset.

	micro-discrim								
Dataset	Sents	Skip	Parsed	Diff	Words	Vocab	Links	P	R	F1
CDS	1832	69%	562	88%	2738	176	2369	0.679	0.542	0.603
basic	579	86%	82	99%	666	209	646	0.595	0.412	0.487
fixes	4236	82%	774	97%	4888	589	3770	0.529	0.476	0.501
gold	229	90%	24	92%	169	95	131	0.565	0.534	0.549
silver	2513	85%	377	99%	3323	527	2844	0.546	0.490	0.522

As before, there is no particularly obvious trend visible in the recall of the link-types.

	micro-discrim							
Dataset	Primary	N	Secondary	N	Punct	N	Other	N
CDS	0.562	1018	0.594	618	-	0	0.469	733
basic	0.388	240	0.357	157	0.2	5	0.475	244
fixes	0.489	1313	0.480	880	0.382	207	0.477	1370
gold	0.448	58	0.511	47	1	1	0.760	25
silver	0.457	1074	0.507	726	0.521	73	0.538	971

Results below for *mini-fuzz*. This dictionary has a much larger vocabulary: 7385 words instead of 1600 as above. Several large differences are apparent: considerably fewer sentences get skipped due to unknown vocabulary (but still more than half!) The mini dictionaries contain 7385 words, and at most 1/5th of them appear in the test corpora. This lack of overlap is remarkable! Compared to *micro-fuzz*, the results are mixed. Precision is mostly the same, sometimes lower; recall is mostly the same or lower and F1 is mostly the same, bouncing around a bit, sometimes higher and sometimes lower than the micro variant.

	mini-fuzz								
Dataset	Sents	Skip	Parsed	Diff	Words	Vocab	Links	P	R	F1
CDS	1832	54%	850	91%	4222	238	3649	0.710	0.480	0.573
basic	579	59%	232	100%	1944	465	1864	0.580	0.376	0.456
fixes	4236	57%	1767	97%	12122	1425	9748	0.540	0.399	0.459
gold	229	72%	64	100%	449	228	343	0.504	0.362	0.421
silver	2513	62%	943	99%	9859	1308	8793	0.531	0.392	0.451

Recall of link-types:

	mini-fuzz							
Dataset	Primary	N	Secondary	N	Punct	N	Other	N
CDS	0.503	1524	0.480	1022	-	0	0.450	1103
basic	0.320	640	0.364	544	0.344	32	0.443	648
fixes	0.398	3373	0.419	2570	0.432	463	0.379	3342
gold	0.314	140	0.384	125	0.6	5	0.397	37
silver	0.357	3111	0.393	2514	0.494	328	0.418	2840

Below for *mini-discrim*. Compared to *mini-fuzz*, the precision is mostly unchanged; mixed. Recall is consistently higher, a lot higher, which brings up the F1 scores across the board. This is a lot like the difference between *micro-discrim* and *micro-fuzz*. The difference between the *discrim* and *fuzz* variants is that the *discrim* variant uses a sigmoid merger proceedure, instead of a fixed fraction. The sigmoid appears to make an important quality difference.

	mini-discrim									
Dataset	Sents	Skip	Parsed	Diff	Words	Vocab	Links	P	R	F1
CDS	1832	53%	850	90%	4222	238	3648	0.658	0.538	0.592
basic	579	55%	232	99%	1944	465	1878	0.590	0.452	0.512
fixes	4236/3921	53%	1767	98%	12122	1425	9753	0.522	0.469	0.494
gold	229	67%	64	95%	449	228	343	0.516	0.475	0.495
silver	2513/2138	56%	943	100%	9859	1308	8840	0.532	0.475	0.502

1.6 Unknown word guessing

Having tiny vocabularies is disastrous for understanding text. Can one automate the guessing of unknown words? Of course one can! The default LG parser has an "UNKNOWN-WORD" mechanism, whereby any word that does not appear in the dictionary is defaulted to a collection of UNKNOWN-WORD entries, each carrying a set of associated disjuncts. Multiple UNKNOWN-WORD entries are allowed, in order to simplify disjunct management; however, when they are used, they are all used, and no preference is given for one over another.

The tactic obvious tactic to use in the current situation is to insert one UNKNOWN-WORD marker for each grammatical class. The effect will be that, during parsing, each unknown word will be tried out with each gram-class. When a parse is found, the tag on the UNKNOWN-WORD entry will indicate which class was used. When multiple parses are found, they are ranked according to the total word-disjunct MI for the sentence. The dictionaries evaluated in this section are the same ones as those above, except that one UNKNOWN-WORD entry was added for each gram class containing two or more members; singleton-classes were ignored.

The use of the UNKNOWN-WORD mechanism can (and does) lead to a combinatoric explosion during parsing. Naively, one might expect a runtime of $O\left(N^k\right)$ for k unknown words in the sentence; here N is presumably the total of all disjuncts in all candidate classes, as each is tried against the next as a possible match. For these dictionaries, N varies from 1K to 100K, and so it seems that sentences with 3,4,5 unknown words might take *hours* to parse. Hours is really bad. We are patient, here, but clearly, something needs to be done to avoid the bog-down. In practice, the actual slow-down is highly variable from dictionary to dictionary, as a preparse-pruning stage can decimate N down to a reasonable number quite effectively ... most of the time. However, some dictionaries choke on some sentences.

We expect 100% of the sentences to be covered, so this is a big change from before. Two columns from earlier tables are omitted, since all of the test-corpus is parsed and compared. Since LG will run on every sentence in each test-corpus, the total number of observed words, and the total known-vocabulary will be the same for all of the different dictionaries-under-test. These are reported just once, below.

Dataset	Sents	Words	Vocab	Links
CDS	1832	9274	300	7950
basic	579	4862	1117	4701
fixes	4236	28528	4082	23213
gold	229	1895	782	1539
silver	2513	30079	3657	27297

Here are the results for *micro-fuzz,* with unknown-word-guessing enabled. Recall, the parameters were `(gram-classify-greedy-fuzz 0.65 0.3 4)`. Compared to the same dataset without guessing, the resulting precision and recall changes erratically, sometime moving up, or down, depending on the test corpus. The overall F1 is unchanged or improved. Conclusion: adding unknown-word guessing vastly improves coverage, and also helps F1.

	micro-fuzz-unk `(fuzz 0.65 0.3)`				
Dataset	Sents	Diff	P	R	F1
CDS	1832	76%	0.660	0.541	0.595
basic	579	98%	0.555	0.440	0.491
fixes	4236	96%	0.502	0.459	0.479
gold	229	99%	0.484	0.468	0.476
silver	2513	100%	0.492	0.445	0.467

For *micro-fuzzier-unk*: almost no difference at all from the above. Note that both have almost exactly the same number of grammatical classes (135 *vs.* 136) and so unknown-word guessing seems to be unaffected.

	micro-fuzzier-unk `(fuzz 0.65 0.6)`				
Dataset	Sents	Diff	P	R	F1
CDS	1832	88%	0.657	0.538	0.592
basic	579	97%	0.557	0.441	0.493
fixes	4236	97%	0.501	0.458	0.478
gold	229	99%	0.479	0.461	0.470
silver	2513	100%	0.490	0.443	0.465

Below is *micro-fizz-unk*, which uses a cosine-cutoff of 0.75. Comparing to *micro-fizz-unk* with a cutoff of 0.65, it seems that precision is mostly unchanged. Recall is sharply down, and so, for the most part, the F1 scores are down.

There are two possible explanations for this difference:

1. The tighter discriminator results in narrower gram classes, which have trouble hooking up into a good parse. Here, “narrower” means that each gram class has fewer disjuncts in it.

2. The *micro-fizz-unk* dictionary has fewer grammatical classes (115 vs 135), which, in addition to each class being narrower, causes unknown word guessing to struggle.

Disentangling these effects is not obvious.

	micro-fizz-unk `(fuzz 0.75 0.3)`				
Dataset	Sents	Diff	P	R	F1
CDS	1832	90%	0.665	0.486	0.562
basic	579	98%	0.555	0.403	0.467
fixes	4236	97%	0.495	0.419	0.454
gold	229	98%	0.471	0.428	0.449
silver	2513	100%	0.483	0.402	0.439

Here is for *micro-discrim-unk*; it should be compared to *micro-discrim*, which uses the same algo and parameters, but doesn't have word-guessing built in. Changes to precision, recall and F1 are mixed, depending on the dataset. It seems that unknown-word guessing didn't have much of an effect.

Compared to *micro-fuzz-unk*, the change in precision is mixed, the recall is noticeably better, resulting in better F1 scores overall. This reaffirms an earlier observation: the *discrim* algo produces better data than the *fuzz* algo.

	micro-discrim-unk `(discrim 0.5)`				
Dataset	Sents	Diff	P	R	F1
CDS	1832	83%	0.651	0.580	0.614
basic	579	98%	0.571	0.482	0.523
fixes	4236	97%	0.496	0.490	0.493
gold	229	99%	0.475	0.497	0.485
silver	2513	100%	0.507	0.491	0.499

Here is *micro-diss-unk*. It should be compared to two different dictionaries: *micro-discrim-unk*, immediately above, which uses the same clustering algo, but a weaker cutoff, also to *micro-fuzz-unk*, which uses a different clustering algo, but the same cutoff. Note that when the cutoff is the same, both produce the same number of grammatical classes (135, either way). The looser cutoff produces fewer classes (124); presumably each class is larger. This does not resolve the question of whether it is the sharpness of the classes that matter, or their raw number. Perhaps these effects cannot be disentangled...

Based on prelim incomplete numbers: the algo don't matter very much, the discrim does.

	micro-diss-unk `(discrim 0.65)`				
Dataset	Sents	Diff	P	R	F1
CDS	1832	88%	0.654	0.537	0.590
basic	579	98%	0.560	0.440	0.493
fixes	4236	100%	0.503	0.459	0.480
gold	229	99%	0.482	0.466	0.474
silver	2513	100%	0.493	0.443	0.467

Here's *micro-dissier*-unk: it can be compared to the above, which uses a weaker cutoff, or to *micro-fizz-unk*, which uses the same cutoff. The sharper cutoff is having little effect on precision. It is forcing down recall, and thus F1.

	micro-dissier-unk `(discrim 0.75)`				
Dataset	Sents	Diff	P	R	F1
CDS	1832	92%	0.660	0.481	0.556
basic	579	98%	0.559	0.406	0.471
fixes	4236	97%	0.493	0.417	0.451
gold	229	99%	0.464	0.421	0.442
silver	2513	100%	0.487	0.405	0.442

Here's *micro-dissiest-unk*, with a sharp cutoff still. Precision drops, recall drops, F1 drops. Conclusion: it is quite possible to make the classes too narrow.

	micro-dissiest-unk `(discrim 0.85)`				
Dataset	Sents	Diff	P	R	F1
CDS	1832	93%	0.666	0.441	0.531
basic	579	98%	0.536	0.343	0.418
fixes	4236	97%	0.487	0.365	0.417
gold	229	99%	0.430	0.357	0.390
silver	2513	100%	0.462	0.339	0.391

Below is *mini-fuzz-unk.* It is to be compared to *micro-fuzz-unk*, which uses the same parameters, but has a smaller vocabulary.
it appears that ... mixed results ...

	mini-fuzz-unk `(fuzz 0.65 0.3)`				
Dataset	Sents	Diff	P	R	F1
CDS	1832	87%	0.677	0.529	0.594
basic	579/529	99%	0.559	0.393	0.461
fixes	4236	97%	0.498	0.412	0.451
gold	229	99%	0.451	0.387	0.416
silver	2513	100%	0.490	0.409	0.446

The basic corpus contains sentences that took excessive times to parse with this dictionary; as a result, those sentences were skipped. The second, smaller number indicates the number of setences that did not time-out. Compared to *mini-discrim* (same dictionary, but skipping sentences with unknown words), this is a wash – precision is mostly lower, just a bit, recall is mostly higher, just a bit, and F1 is about the same, varying. Conclusion: unknown word guessing does not hurt, and sometimes helps. Compared to *micro-discrim-unk*, (same algorithm applied to a much smaller dataset; both use unknown-word-guessing) this has a slightly better precision, slightly lower recall, and F1 is the same or slightly worse. Apparently, unknown-word-guessing with a small vocabulary is pretty much as good as that for a bigger vocabulary (and neither are terribly good). So, increasing vocabulary size, in and of itself, does not offer any benefit, with the algos as they currently stand.

	mini-discrim-unk `(discrim 0.5)`				
Dataset	Sents	Diff	P	R	F1
CDS	1832	85%	0.650	0.566	0.605
basic	579/491	55%	0.581	0.462	0.515
fixes	4236/3631	97%	0.508	0.478	0.492
gold	229	99%	0.460	0.444	0.452
silver	2513/2138	100%	0.512	0.474	0.492

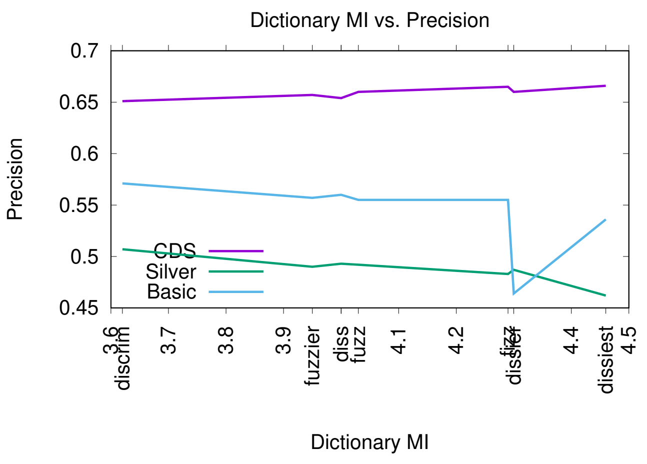
Cosine Redo

It appears that there was a bug in the cosine calculations for all of the datasets reported above. The results below come from a re-do. It appears to improve both precision and recall by a fair amount. Since the change did not affect the overall number of classes found, this would suggest that both the buggy and the revised dictionaries have clusters with the same members, and that the primary reason for the score difference is the differrent sigmoid! This is ... remarkable.

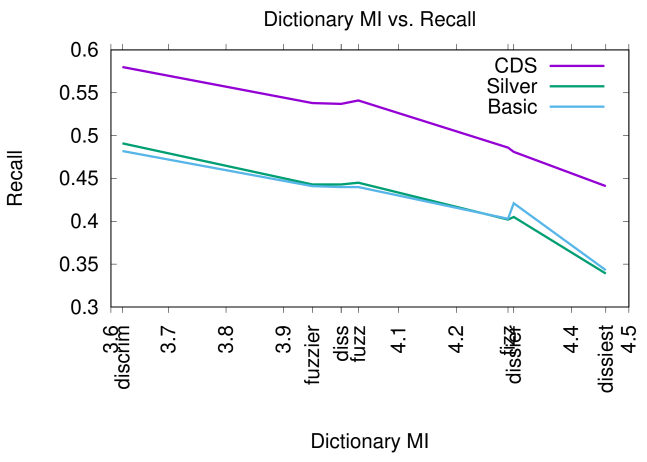
	micro-rediscrim-unk `(discrim 0.5)`				
Dataset	Sents	Diff	P	R	F1
CDS	1832	83%	0.660	0.590	0.623
basic	579	98%	0.569	0.481	0.521
fixes	4236	97%	0.510	0.505	0.508
gold	229	99%	0.498	0.517	0.507
silver	2513	100%	0.507	0.491	0.499

1.7 Graphs

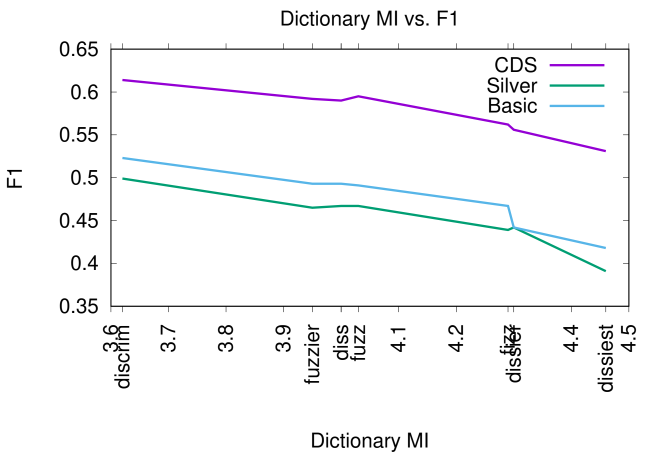
Reading the above tables is hard. It is easier to see what is going on with graphs. Here. The first figure shows precision as a function of dataset MI for three of the corpora (there is some missing data, still coming in.)



The one below shows recall as a function of dataset MI.



Finally, the reciprocal mean of the two, aka the F1-score. Judging from the slope, its now clear that the loss in recall dominates the gain in precision.



1.8 Conclusions for Part I

The following conclusions can be made for the above baseline measurements.

- Taken as a proof-of-concept, the pipeline seems to work, and provides a stable foundation for further research.
- Tighter clustering only marginally improves precision, but sharply damages recall. Given a choice, looser clustering improves the overall coverage of the dictionary. In effect, the dictionary understand more, even if it doesn't understand it quite as well.
- The sigmoid taper decision used in the *discrim* merge algo, vs. the *fuzz* algo seems to make an important improvement in quality. Accidental tweaks seem to further improve scores. This suggests that the manner of clustering has an important effect.
- Larger dictionaries with larger vocabularies provide only mixed results. The scores are roughly the same, which is surprising. Tiny vocabularies with effective unknown-word guessing are mostly just as good as larger vocabularies (with unknown word guessing). This indicates that the current algos are not effectively extracting any extra info from the larger vocabularies!
- Unknown-word guessing by the parser appears to be an effective strategy for broadening the coverage of the dictionary.

2 ULL Dictionary Measurements

Same procedures as above, applied to the ULL dictionaries. These are provided by Anton Kolonin and his crew, and are generated using completely different software pipeline, described elsewhere.

Results presented below; this uses the same measuring techniques as above. Each grammar is measured twice: once, by looking at all sentences; and a second time, skipping sentences that contained words not in the dictionary.

These measurements indicate that two major breakthroughs have been made in the ULL project. These are:

- Results from the *ull-lgeng* dataset indicates that the ULL pipeline is a high-fidelity transducer of grammars. The grammar that is pushed in is the effectively the same as the grammar that falls out. If this can be reproduced for other grammars, e.g. Stanford, McParseface or some HPSG grammar, then one has a reliable way of tuning the pipeline. After it is tuned to maximize fidelity on known grammars, then, when applied to unknown grammars, it can be assumed to be working correctly, so that whatever comes out must in fact be correct.

- The relative lack of differences between the *ull-dnn-mi* and the *ull-sequential* datasets suggests that the accuracy of the so-called "MST parse" is relatively unimportant. Any parse, giving any results with better-than-random outputs can be used to feed the pipeline. What matters is that a lot of observation counts need to be accumulated so that junky parses cancel each-other out, on average, while good ones add up and occur with high frequency. That is, if you want a good signal, then integrate long enough that the noise cancels out.

A third item should be mentioned:

- It appears that the Project Gutenberg training corpus does not appear to be a good sample of the English language. When the learned dictionaries are applied to other corpora, the scores are disastrously bad!

These are strong claims. Lets look at the results justifying them.

ULL Results

The ULL team provided four dictionaries. These are analyzed below.

ull-lgeng Based on LG-English parses: obtained from http://langlearn.singularitynet.io/data/aglushchenko_parses/
FULL-ALE-dILEd-2019-04-10/context:2_db-row:1_f1-col:11_pa-col:6_word-space:discrete/

I believe that this dictionary was generated by replacing the MST step with a parse where linkages are obtained from LG; these are then busted up back into disjuncts. This is an interesting test, because it validates the fidelity of the overall pipeline. It answers the question: "If I pump LG into the pipeline, do I get LG back out?" and the answer seems to be "yes, it does!" This is good news, since it implies that the overall learning process does keep grammars invariant. That is, whatever grammar goes in, that is the grammar that comes out!

This is important, because it demonstrates that the apparatus is actually working as designed, and is, in fact, capable of discovering grammar in data! This suggests several ideas:

- First, verify that this really is the case, with a broader class of systems. For example, start with the Stanford Parser, pump it through the system. Then compare the output not to LG, but to Stanford parser. Are the resulting linkages (the F1 scores) at 80% or better? Is the pipeline preserving the Stanford Grammar? I'm guessing it does...
- The same, but with Parsey McParseface.
- The same, but with some known-high-quality HPSG system.

If the above two bullet points hold out, then this is a major breakthrough, in that it solves a major problem. The problem is that of evaluating the quality of the grammars generated by the system. To what should they be compared? If we input MST parses, there is no particular reason to believe that they should correspond to LG grammars. One might hope that they would, based, perhaps, on some a-priori hand-waving about how most linguists agree about what the subject and object of a sentences is. One might in fact find that this does hold up to some fair degree, but that is all. Validating grammars is difficult, and seems *ad hoc*.

This result offers an alternative: don't validate the grammar; validate the pipeline itself. If the pipeline is found to be structure-preserving, then it is a good pipeline. If we want to improve or strengthen the pipeline, we know have a reliable way of measuring, free of quibbles and argumentation: if it can transfer an input grammar to an output grammar with high-fidelity, with low loss and low noise, then it is a quality pipeline. It instructs one how to tune a pipeline for quality: work with these known grammars (LG/Stanford/McParse/HPSG) and fiddle with the pipeline, attempting to maximize the scores. Built the highest-fidelity, lowest-noise pipeline possible.

This allows one to move forward. If one believes that probability and statistics are the correct way of discerning reality, then that's it: if one has a high-fidelity corpus-to-grammar transducer, then whatever grammar falls out is necessarily, a priori a correct grammar. Statistics doesn't lie. This is an important breakthrough for the project.

Lets now look at the actual data. First, the results when all sentences are parsed, including those with unknown words. Since the ULL dictionaries take no special steps to treat unknown words, the results are not terribly inspiring. The precision for *gold/silver* is quite high: recall that *gold/silver* are based on a selection of Gutenberg texts, the same class as the training set, and so high precision is commendable. Recall suffers a bit, but F1 is passable. When tested against corpora that are quite different from the training set, its a bit of a disaster: recall drops through the floor. It would seem that the *basic/fixes* corpora contain sentences that are very different than the Gutenberg texts. This is unfortunate; it suggests that the Gutenberg texts do not provide an adequate sample of the English language.

	ull-lgeng				
Dataset	Sents	Diff	P	R	F1
CDS	1832	93%	0.828	0.293	0.433
basic	579	100%	0.718	0.134	0.226
fixes	4236	98%	0.770	0.147	0.246
gold	229	100%	0.960	0.699	0.809
silver	2513	100%	0.904	0.599	0.720

The table above shows results when all sentences are tested; the table below when sentences with unknown words are skipped. Here, the picture brightens considerably! For *gold/silver*, the accuracy goes up, almost maxing out for *gold*. The recall shoots way up as well, with the F1 scores on *gold/silver* being just fantastic! It is based on these two lines that the above claims of a breakthrough are founded. If this can be reproduces for Stanford/McParse/*etc.* we're well on the way!

The table also emphasizes the incompleteness of the Gutenberg training set. The recall scores for *basic/fixes* are a complete disaster. Whatever is in that grammar, its not covering perfectly common 20th-century English.

	ull-lgeng									
Dataset	Sents	Skip	Parsed	Diff	Words	Vocab	Links	P	R	F1
CDS	1832	14%	1579	92%	7882	289	6757	0.832	0.310	0.452
basic	579	96%	25	100%	119	39	98	0.833	0.051	0.096
fixes	4236	93%	305	97%	1805	498	1552	0.749	0.146	0.245
gold	229	51%	113	12%	931	418	761	0.983	0.957	0.969
silver	2513	55%	1138	43%	12986	2146	11675	0.937	0.833	0.882

ull-sequential Based on "sequential" parses: obtained from http://langlearn.singularitynet.io/data/aglushchenko_
FULL-SEQ-dILEd-2019-05-16-94/GL_context:2_db-row:1_f1-col:11_pa-col:6_word-space:discrete/

I believe that this dictionary was generated by replacing the MST step with a parse where there are links between neighboring words, and then extracting disjuncts that way. This is an interesting test, as it leverages the fact that most links really are between neighboring words. The sharp drawback is that it forces each word to have an arity of exactly two, which is clearly incorrect.

	ull-sequential				
Dataset	Sents	Diff	P	R	F1
CDS	1832	99%	0.651	0.058	0.107
basic	579	100%	0.542	0.026	0.050
fixes	4236	99%	0.473	0.128	0.202
gold	229	100%	0.585	0.518	0.549
silver	2513	100%	0.595	0.497	0.542

The table above shows results when all sentences are tested; the table below when sentences with unknown words are skipped.

	ull-sequential									
Dataset	Sents	Skip	Parsed	Diff	Words	Vocab	Links	P	R	F1
CDS	1832	14%	1579	99%	7882	289	6757	0.658	0.062	0.114
basic	579	96%	25	100%	119	39	98	1.000	0.031	0.059
fixes	4236	92%	308	99%	1821	503	1564	0.480	0.075	0.130
gold	229	51%	113	100%	931	418	761	0.610	0.656	0.632
silver	2513	55%	1138	100%	12986	2146	11675	0.615	0.623	0.619

ull-dnn-mi Based on "DNN-MI-lked MST-Parses": obtained from http://langlearn.singularitynet.io/data/aglushch
GUCH-SUMABS-dILEd-2019-05-21-94/GL_context:2_db-row:1_f1-col:11_pa-col:6_word-space:discrete/

I believe that this dictionary was generated by using the standard MST parse step, but using "Bertram weights" derived from a neural net, instead of using word-pair MI scores.

	ull-dnn-mi				
Dataset	Sents	Diff	P	R	F1
CDS	1832	97%	0.662	0.325	0.436
basic	579	100%	0.563	0.184	0.277
fixes	4236	98%	0.445	0.180	0.256
gold	229	100%	0.533	0.459	0.493
silver	2513	100%	0.524	0.421	0.467

The table above shows results when all sentences are tested; the table below when sentences with unknown words are skipped. It appears that precision is higher or sharply higher, depending on the corpus (sharply higher for *basic/fixes*, which are dissimilar from the training set, but only a little higher for *silver/gold*, which are similar to the training set...) The effect on recall is the opposite: recall is mixed for *basic/fixes* but sharply higher for *silver/gold*. Conclude that testing with sets similar to the training sets does little for precision, but a lot for recall. Dis-similar test corpora flip the other way. Overall, F1 is mostly higher.

	ull-dnn-mi									
Dataset	Sents	Skip	Parsed	Diff	Words	Vocab	Links	P	R	F1
CDS	1832	14%	1579	96%	7882	289	6757	0.668	0.340	0.451
basic	579	96%	25	100%	119	39	98	0.790	0.153	0.256
fixes	4236	92%	301	98%	1784	494	1531	0.550	0.218	0.312
gold	229	51%	113	99%	931	418	761	0.550	0.573	0.561
silver	2513	55%	1137	100%	12966	2142	11655	0.539	0.541	0.540

Comparing either of these to the *ull-sequential* dictionary indicates that precision is worse, recall is worse, and F1 is worse. This vindicates some statements I had made earlier: the quality of the results at the MST-like step of the process matters relatively little for the final outcome. Almost anything that generates disjuncts with slightly-better-than-random will do. The key to learning is to accumulate many disjuncts: just as in radio signal processing, or any kind of frequentist statistics, to integrate over a large sample, hoping that the noise will cancel out, while the invariant signal is repeatedly observed and boosted.

ull-mst-mi Based on "GCB-FULL-ANY-dILEd": obtained from http://langlearn.singularitynet.io/data/aglushche
FULL-ANY-dILEd-2019-05-09-94/GL_context:2_db-row:1_f1-col:11_pa-col:6_word-space:discrete/

I believe that this dictionary was generated by using the standard MST parse step, using standard word-pair MI.

	ull-mst-mi				
Dataset	Sents	Diff	P	R	F1
CDS	1832	96%	0.724	0.327	0.451
basic	579	100%	0.595	0.173	0.268
fixes	4236	98%	0.509	0.171	0.256
gold	229	100%	0.507	0.417	0.457
silver	2513	100%	0.517	0.382	0.439

The table above shows results when all sentences are tested; the table below when sentences with unknown words are skipped.

	ull-mst-mi									
Dataset	Sents	Skip	Parsed	Diff	Words	Vocab	Links	P	R	F1
CDS	1832	14%	1579	95%	7882	289	6758	0.728	0.342	0.465
basic	579	96%	25	100%	119	39	98	0.667	0.122	0.207
fixes	4236	93%	298	98%	1758	492	1508	0.560	0.189	0.283
gold	229	51%	113	100%	931	418	760	0.514	0.542	0.528
silver	2513	55%	1138	100%	12986	2146	11745	0.521	0.509	0.515

Neither look as good as the *ull-dnn-mi* dataset.

Open Questions

There are some things I do not understand about the ULL pipeline. These are:

- When the ULL pipeline clusters words together, what distance metric is used? Cosine? Something else?
- How are link types clustered? That is, how does ULL determine if two links are similar, and belong in the same class? What metric is use? (Clustering links is a distinct step from clustering words; for ULL, I think tehse are somehow combined into one?)
- How are words merged into word-classes? I believe that when ULL decides that two words are similar, a whole-sale merge all disjuncts into the word-class is performed, without any attempt at setting aside the disjuncts that "don't belong". Is that right? (Disallowing the merger of disjuncts that "don't belong" is the primary mechanism of WSD, so this is important to understand.)

Without a clear description of these, its hard to tell what, exactly, ULL is doing.

2.1 Conclusions for Part II – ULL pipeline

There are several vitally important results gleaned from the ULL datasets. These suggest a method for how to tune the language learning pipeline: it should be tuned to maximize the fidelity of its action on known grammars. That way, when applied to new, unknown grammars, it can be trusted to produce good results. A second conclusion is that the results are only weakly dependent on the so-called “MST parse”. Anything that gives reasonably decent results at this stage is good enough to fish out a grammar. A third conclusion was that the Project Gutenberg corpora do not provide an adequate sample of modern English.

Contrasting to the Linas pipeline, one can also say:

- The Linas pipeline seems to be producing dictionaries with better coverage than the ULL dictionaries. This is almost certainly due to only three factors: a training set that is broader than just Gutenberg; a much larger training set, with probably an order of magnitude (or two) more samples in it; the use of the unknown-word mechanism.
- Unknown-word guessing by the parser appears to be an effective strategy for broadening the coverage of the dictionary; supporting this in the ULL pipeline is surely a good idea!
- The Linas pipeline should be tuned in the fashion that the tuning breakthrough suggests. Without this, one is working blind.

3 Info-theoretic Improvements

The results in Part 1 embodied several "known bugs", each of which needs to be fixed. These are:

- Failure to use the Kullback-Liebler divergence instead of cosine distance. I argued last summer about why KL is better than cosine distance, but my own code base has yet to catch up with my theory. All of the Part I dictionaries used cosine distance.
- Failure to perform a fine-grained merge that distinguishes different word-senses. The Part I merge algo was a quasi-linear merge that moved many/most of the disjunct counts from a word into the word-class it was assigned to. This has the unintended effect of blurring together distinct word-senses. A more fine-grained merge would make this decision on a disjunct-by-disjunct basis, deciding when a disjunct belongs, and when it doesn't. Such an algo will be (when I get done writing it) a "binary optimization" algo (That is, an "integer programming" algo with the integers restricted to 0 or 1: https://en.wikipedia.org/wiki/Integer_programming) The algo chooses, on a disjunct-by-disjunct basis, whether that disjunct belongs to the given word-sense, or not. I think I have an OK algo for this that can run in linear time; written up in the diary, but not yet converted into code. This should improve WSD by a lot. Maybe. Hmmm...
- Failure to cluster link-types/connectors. Part 1 used a brute-force combinatoric algo: if there was a link between two words (even if that link was wrong), then a link between all word-classes containing those words was created. This is wrong in three different ways:
  1. It excessively generalizes ("blurs") the grammar, allowing two word-classes to connect, just because some random error in the MST stage accidentally connected two words.
  2. Its wrong because it also blurs different word-senses: if a word belongs to two or more word-classes, each will be connected to, thus allowing links between word-senses that should have been disallowed.
  3. Finally, it is the direct and immediate cause of the combinatoric explosions that make parsing so terribly slow.

Here in Part III, solutions to the problems above will be explored.

3.1 Info-based distances

These dictionaries are generated by using MI scores instead of cosines for determining distances between words. They use the same projective clustering as the earlier datasets. These use a “sigmoid” taper; its not sigmoid in this case, but linear, running between 0% at the MI cutoff and 100% at the lesser of the two self-MI’s of the items being merged. All of these use the unknown-word mechanism.

The initial form of these dicts, where a connector is created for each distinct word-class that a word belongs to cannot be measured. The result is that there is a combinatoric explosion of expressions for each word, leading to an even worse combinatoric overflow during parsing. Here’s a snapshot example from the *micro-disinfo* dictionary:

```
Token "the" matches:
    the.54                     2879150524  disjuncts
    the.24                        3397807  disjuncts
Token "old" matches:
    old.71                        3568091  disjuncts
Token "was" matches:
    was.89                        4809781  disjuncts
    was.19                          20761  disjuncts
Token "on" matches:
    on.68                      3259929866  disjuncts
    on.24                         3397807  disjuncts
    on.17                           22768  disjuncts
Token "his" matches:
    his.24                        3397807  disjuncts
Token "shoulder" matches:
    shoulder.44                1274692269  disjuncts
    shoulder.39                    204008  disjuncts
    shoulder.28                 269503230  disjuncts
Token "." matches:
    ..81                        509547141  disjuncts
    ..34                       3730768318  disjuncts
```

Notice how “the” explodes into almost 3 billion disjuncts, “on” into more than 3 billion, and even a common noun like “shoulder” ends up with over a billion disjuncts. Clearly, this is intolerable.

Therefore ....

	micro-disinfo `(disinfo 2.0)`				
Dataset	Sents	Diff	P	R	F1
CDS	1832				
basic	579				
fixes	4236				
gold	229				
silver	2513				

The End