Learning a Lexis

Linas Vepstas

June 2019

Abstract

Lexical descriptions of language can be learned using information theoertic techniques. This is a rough draft (collection of notes) for a planned future paper describing the language-learning system, and justifying why it works.

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.

A primary claim is that the above algo preserves lexical structure. This is, if the provisional parses generated by step 1 came from some lexical grammar, then the output from step 5 will be the same lexis.

Learning a Lexis

Consider step 1 where parse 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.

Again

Stating the above, again. From an email:

Claim: (1) natural language grammar has the "decomposition property" (aka "lexical property"), and (2) The above algo not only learns a lexis, but preserves lexical structure

The "decomposition property" states that "grammar is lexical". Natural language is "lexical" when it's structure can be described by a "lexis" -- a dictionary, whose dictionary headings are words, end whose dictionary entries are word-definitions of some kind -- disjuncts for LG; something else for Stanford/McParseface/HPSG/etc.

If you take some lexical grammar (Stanford/McParseface/whatever) and generate a bunch of parses, run it through the ULL pipeline, and learn a new lexis, then, ideally, if your software works well, then that *new* lexis should get close to the original input lexis. And indeed, that is what you are finding with F1-for-LG-English.

Your F1-for-LG-English results indicate that if you use LG as input, then ULL correctly learns the LG lexis. That is a good thing. I believe that ULL will also be able to do this for any lexis... provided that you take enough samples. (There is a lot of evidence that your sample sizes are much too small.)

Let's assume, now, that you take Stanford parses, run them through ULL, learn a dict, and then measure F1-for-Stanford against parses made by Stanford. The F1 should be high. Ideally, it should be 1.0. If you measure that learned lexis against LG, it will be lower - maybe 0.9, maybe 0.8, maybe as low as 0.65. That is because Stanford is not LG; there is no particular reason for these two to agree, other than in some general outline: they probably mostly agree on subjects, objects and determiners, but will disagree on other details (aux verbs, "to be", etc.)

Do you see what I mean now? The ULL pipeline should preserve the lexical structure of language. If you use lexis X as input, then ULL should generate something very similar to lexis X as output. You've done this for X==LG. Do it for X=Stanford, McParseface, etc. If you do, you should see F1=1.0 for each of these (well, something close to F1=1.0)

Is Language lexical?

Now for part two: what happens when X==sequential, what happens when X==DNN-MI (aka "bertram weights") and what happens when X=="honest MI" ?

Let's analyze X==sequential first. First of all, this is not a lexical grammar. Second of all, it is true that for English, and for just about *any* language, "sequential" is a reasonably accurate approximation of the "true grammar". People have actually measured this. I can give you a reference that gives numbers for the accuracy of "sequential" for 20 different languages. One paper measures "sequential" for Old English, Middle English, 17th, 18th, 19th and 20th century English, and finds that English becomes more and more sequential over time! Cool!

If you train on X==sequential and learn a lexis, and then compare that lexis to LG, you might find that F1=0.55 or F1=0.6 -- this is not a surprise. If you compare it to Stanford, McParseface, etc. you will also get F1=0.5 or 0.6 -- that is because English is kind-of sequential.

If you train on X==sequential and learn a lexis, and then compare that lexis to "sequential", you will get ... kind-of-crap, unless your training dataset is extremely

large, in which case you might approach F1=1.0 However, you will need to have an absolutely immense training corpus size to get this -- many terabytes and many CPU-years of training. The problem is that "sequential" is not lexical. It can be made approximately lexical, but that lexis would have to be huge.

What about X==DNN-Bert and X==MI? Well, neither of those are lexical, either. So you are using a non-lexical grammar source, and attempting to extract a lexis out of it. What will you get? Well -- you'll get ... something. It might be kind-of-ish LG-like. It might be kind-of-ish Stanford-like. Maybe kind-of-ish HPSG-like. If your training set is big enough (and your training sets are not big enough) you should get at least 0.65 or 0.7 maybe even 0.8 if you are lucky, and I will be surprised if you get much better than that.

What does this mean? Well, the first claim is "ULL preserves lexical grammars" and that seems to be true. The second claim is that "when ULL is given a non-lexical input, it will converge to some kind of lexical output".

The third claim, "the Linas claim", that you love to reject, is that "when ULL is given a non-lexical input, it will converge to the SAME lexical output, provided that your sampling size is large enough". Normally, this is followed by a question "what non-lexical input makes it converge the fastest?" If you don't believe the third claim, then this is a non-sense question. If you do believe the third claim, then information theory supplies an answer: the maximum-entropy input will converge the fastest. If you believe this answer, then the next question is "what is the maximum entropy input?" and I believe that it is honest-MI+weighted-clique. Then there is claim four: the weighted clique can be approximated by MST.

It is now becoming clear to me that MST is a kind-of mistake, and that a weighted clique would probably be better, faster-converging. Maybe. The problem with all of this is rate-of-convergence, sample-set-size, amount-of-computation. it is easy to invent a theoretically ideal NP-complete algorithm; its much harder to find something that runs fast.

Anyway, since you don't believe my third claim, I have a proposal. You won't like it. The proposal is to create a training set that is 10x bigger than your current one, and one that is 100x bigger than your current one. Then run "sequential", "honest-MI" and "DNN-Bert" on each. All three of these will start to converge to the same lexis. How quickly? I don't know. It might take a training set that is 1000x larger. But that should be enough; larger than that will surely not be needed. (famous last words. Sometimes, things just converge slowly...)

Tuning

There are two extremely important concepts underpinning 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.

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.

Conclusions

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.

- Unknown-word guessing by the parser appears to be an effective strategy for broadening the coverage of the dictionary.