Kevin Murphy
Swarthmore Honors Project – Computational Linguistics
Spring 2019
Examiner: Jane Chandlee

Introduction

For my project, I implemented several computational linguistic tools based on the the `nltk.cess_cat` and `nltk.cess_esp` corpora. In particular, I wrote a program that takes an `nltk` corpus and uses the provided parse trees to induce a grammar. Further, my program converts that grammar into a Chomsky Normal Form grammar, which is a grammar such that each nonterminal symbol generates exactly two nonterminal symbols. Such a grammar is important because it allows us to generate a binary tree for a given input sentence. In fact, I also implemented the CKY-parse algorithm, that allows us to produce a parse tree from a Chomsky Normal Form grammar and an input string. Finally, I induced and normalized grammars from both `nltk.cess_cat` and `nltk.cess_esp`, and then used those grammars to parse sentences from the `nltk.cess_cat` corpus. For the purpose of comparing the effectiveness of these two parsers, I developed some metrics, as described in my methods section.

Implementation

The source code can be downloaded from GitHub. To install, simply type these commands (the lines beginning with `$`) into a terminal:

```
$ git clone https://github.com/keggsmurph21/cky-parser
$ cd cky-parser
$ pip install --user -r requirements.txt
$ python src/main.py --help
usage: main.py [-h] [-i INDUCE] [-g INGRAMMAR] [-o OUTGRAMMAR] [-c CACHE]
               [-l LEXICON] [-n NUM_SENTS] [-t TEST [TEST ...]] [-T TRUTH]
               [-w WORDS]
               action

'CKY Parser & more : for more info, check out the README :)

positional arguments:
  action

optional arguments:
  -h, --help            show this help message and exit
  -i INDUCE, --induce INDUCE
                        corpus to induce a grammar from (cess_cat/cess_esp)
  -g INGRAMMAR, --ingrammar INGRAMMAR
                        filepath to read a grammar from
  -o OUTGRAMMAR, --outgrammar OUTGRAMMAR
                        filepath to write a grammar to
```

```
  -c CACHE, --cache CACHE
                        read tags to/from, saves lots of time when parsing
  -l LEXICON, --lexicon LEXICON
                        which lexicon to use when preparing the grammar for
                        parsing
  -n NUM_SENTS, --num_sents NUM_SENTS
                        consider only the first N sentences when analyzing
  -t TEST [TEST ...], --test TEST [TEST ...]
                        whitespaced-separated list of cache files to analyzed
  -T TRUTH, --truth TRUTH
                        cache file to use as the "ground truth" for analysis
  -w WORDS, --words WORDS
                        only consider the first W words of sentences (useful
                        for making sense out parse output)
```

Note: most of the important files (*e.g.*, all of the Python files) live in the `src/` directory, while `cache/` contains some generated files in order to avoid redundant work during development. In particular, this contains some copies of induced and Chomsky Normal Form grammars, along with some JSON files required by `src/parse.py`. Furthermore, `scripts/` contains some bash script utility scripts for regenerating grammars from scratch or other data. The program is meant to be mostly interacted with via the command line, and its entry point is the file `src/main.py`. Documentation for interacting with this program is provided primarily via the command line option `--help` (as shown above), although some documentation can also be found in the file `README.md`.

Methods

The implementation of the inducer, normalizer, and parser follow Chapters 11.1–11.2 of Daniel Jurafsky and James H. Martin's *Speech and Language Processing* (3rd edition). For reference, please consult the source code. For the remainder of this section, I will discuss and justify my choices of metrics on the parser's accuracy.

Once the CKY-parser has a fully equipped set of grammar rules and lexicon, we can attempt to gauge its effectiveness. Here, we distinguish two cases: **recognition** and **parsing**[1]. We consider developing metrics on each of these cases separately.

In recognizing, we only care whether a given input sentence in grammatical. That is, we check if cell $[0, n]$ contains a start symbol σ. Presumably, if we use a sentence to build a grammar, our grammar should always recognize that sentence. Since we are recognizing a sentence that our grammar has never "seen," it is probable that many inputs will fail to generate grammatical parses. Indeed, we would like a metric that is capable of measuring incomplete parses.

To this end, consider a grammar Γ and an input sentence S. Let $\alpha, \beta \in S$ be tokens at positions

[1] Jurafsky & Martin p. 231

i and j, respectively. Then we define the distance δ between α and β to be

$$\delta(\Gamma, \alpha, \beta) = \begin{cases} \mid i - j \mid & \exists A \in \Gamma \text{ such that } A \Rightarrow^* \alpha \text{ and } A \Rightarrow^* \beta \\ 0 & \text{otherwise} \end{cases}.$$

That is, if there is a nonterminal $A \in \Gamma$ that produces both α and β. Intuitively, this corresponds to the case where α and β "fall under" some tag in the parse tree. Further, we can define the "height" h of a token to be

$$h(\Gamma, \alpha) = \max_{\beta \in \sigma} \delta(\Gamma, \alpha, \beta).$$

Again, we conceptualize this for a binary tree to be the number of tags we can project up in a single direction. For a fully recognized sentence $\hat{S}$, both the first and last tokens $\alpha_1, \alpha_n \in \hat{S}$ will be such that $h(\Gamma, \alpha_1) = |\hat{S}| - 1 = h(\Gamma, \alpha_n)$. In addition, for incompletely recognized sentences (*i.e.*, ungrammatical with respect to Γ), there will be no token $\alpha \in S$ such that $h(\Gamma, \alpha) = |S| - 1$.

In fact, tokens in sentences that are "more completely recognized" (in the sense that the parse tree is more completely filled in) will tend to have higher values of h than tokens in sentences with sparse parse trees. This motivates three metrics for incompletely recognized sentences: h_1, h_2, h_3, given by

$$h_1(\Gamma, S) := \max_{\alpha \in S} h(\Gamma, \alpha) \quad ; \quad h_2(\Gamma, S) := \frac{1}{|S| - 1} \sum_{\alpha \in S} h(\Gamma, \alpha) \quad ;$$

$$h_3(\Gamma, S) := |\{\alpha \in S : h(\Gamma, \alpha) \neq 0\}|\,.$$

Note that for fully recognized, partially recognized, and completely unrecognized sentences S_1, S_2, S_3 (respectively) of the same length n, we have

$$n = h_1(\Gamma, S_1) > h_1(\Gamma, S_2) > h_1(\Gamma, S_3) = 0 \quad ; \quad h_2(\Gamma, S_1) > h_2(\Gamma, S_2) > h_2(\Gamma, S_3) = 0 \quad ;$$

$$0 = h_3(\Gamma, S_1) < h_3(\Gamma, S_2) < h_3(\Gamma, S_3) = n.$$

Thus, each of these metrics conveys some information about incompletely parsed sentences. If we let $\hat{\Gamma}$ be a grammar such that it will recognize any sentence we pass it as grammatical, then we can compare the performance of any other grammar Γ against $\hat{\Gamma}$ in terms of the relative scores of h_i for each grammar. Across a corpus of input sentences $\mathbb{S}$, we could simply define metrics on $\mathbb{S}$ by

$$H_i(\Gamma, \mathbb{S}) := \frac{1}{|\mathbb{S}|} \sum_{S \in \mathbb{S}} h_i(\Gamma, S) \quad (i = 1, 2, 3)$$

Finally, we realize that H_i in its current definition weights longer sentences more heavily. To address this, we simply normalize each metric by the size of the sentence. That is, define

$$h_i'(\Gamma, S) := \frac{h_i(\Gamma, S)}{|S|} \quad \text{and} \quad H_i'(\Gamma, \mathbb{S}) = \frac{1}{|\mathbb{S}|} \sum_{S \in \mathbb{S}} h_i'(\Gamma, S) \quad (i = 1, 2, 3).$$

In addition to measuring the degree to which a parse tree is incompletely recognized, we want to examine the cases in which our grammar Γ recognizes a sentence S. Once Γ recognizes a sentence as grammatical, it can then generate a set of grammatical parses (denote the set of such parses Π generated by a grammar Γ for a sentence S by $\Pi = \Gamma(S)$). In our particular case, we "know" what the parse $\hat{\pi}$ should be for each sentence S. Thus, in addition to checking whether $\Gamma(S) \neq \varnothing$ (*i.e.*, recognition), we also would like to check whether we generated the "right" parse (*i.e.*, whether $\hat{\pi} \in \Gamma(S)$). To this end, we define two more metrics P_1, P_2 by

$$P_1(\Gamma, \mathbb{S}) := |\{S \in \mathbb{S} : \Gamma(S) \neq \varnothing\}| \quad \text{and} \quad P_2(\Gamma, \mathbb{S}, \{\hat{\pi}\}) := |\{S \in \mathbb{S} : \hat{\pi} \in \Gamma(S)\}|.$$

As before, for fully correct, partially correct, and completely incorrect grammars $\Gamma_1, \Gamma_2, \Gamma_3$ (respectively) acting on a corpus $\mathbb{S}$ we have

$$|\mathbb{S}| = P_1(\Gamma_1, \mathbb{S}) > P_1(\Gamma_2, \mathbb{S}) > P_2(\Gamma_3, \mathbb{S}) = 0 \quad \text{and}$$

$$|\mathbb{S}| = P_2(\Gamma_1, \mathbb{S}, \{\hat{\pi}\}) > P_2(\Gamma_2, \mathbb{S}, \{\hat{\pi}\}) > P_2(\Gamma_3, \mathbb{S}, \{\hat{\pi}\}) = 0.$$

In our particular case, we would like to gauge the performance of a Catalan parser that we originally induced from Spanish trees (define this grammar to be Γ_s). As described above, in order to test its effectiveness, we would like a "perfect" grammar $\hat{\Gamma}$ (by perfect, we mean $P_2(\hat{\Gamma}, \mathbb{S}, \{\hat{\pi}\}) = |\mathbb{S}|$). Theoretically, if we were to train our parses on the very same Catalan trees that it about to parse, it would give such a grammar. However, as noted below, this grammar Γ_c is not quite so performant, yet it still performs better than Γ_s. In particular, if we let $\mathbb{S}_0$ be the corpus `nltk.cess_cat`, then we can compare the relative values of $\{H'_i\}$ and $\{P_j\}$ for Γ_s and Γ_c.[2]

Results

Unfortunately, we were unable to calculate the total result on all of $\mathbb{S}_0$, as the normalization step for the grammar resulted in tremendous increases in space and computing resources. For example, the following chart summarizes the number of rules required to specify a grammar for a given number N of sentences from the `nltk.cess_cat` corpus:

N	1	5	10	50
induced	92	222	297	671
normalized	140	2406	18050	966102

As we can clearly see, the size of the normalized corpus grows at an exponential rate relative to the size of the induced grammar. This appears to be due mostly to the nature of the corpus itself. That is, the `nltk` corpora have a very non-binary structure. In particular, they contain many, many unit productions, which require the introduction of an exponentially increasing number of new (binary) rules to compensate for.

As a result, it is not feasible for us to induce a complete Chomsky Normal Form grammar from the entire corpus, and so we are instead forced to make calculations on small chunks. Given this constraint, I chose to induce several different grammars Γ_s^N (where superscript N denotes the number of sentences from `nltk.cess_esp` the parser was induced from) and compare them against a reference grammar Γ_c^{100} on 100 sentences ($\mathbb{S}_1$) from `nltk.cess_cat`.

[2]Note: I do not include calculations for the P_2 metric in this report, as I encountered significant difficulty in comparing the two sets of parsed data (*i.e.*, one directly from the corpus and the other produced via backtracing the CKY-recognizer.

The following table gives a summary of the the "recognition metrics" $\{H_i'\}$ for Γ_s^N and Γ_c^{100} on $\mathbb{S}_1$:

	H_1' max h	$H_1'/H_1'(\Gamma_c^{100})$	H_2' average h	$H_2'/H_2'(\Gamma_c^{100})$	H_3' nonzero h	$H_3'/H_3'(\Gamma_c^{100})$
Γ_s^1	2.5	0.025	0.361	0.006	4.241	0.042
Γ_s^{10}	29.153	0.292	12.556	0.197	76.449	0.764
Γ_s^{25}	41.053	0.412	20.368	0.32	85.782	0.858
Γ_s^{50}	52.408	0.525	29.174	0.458	89.714	0.897
Γ_s^{100}	64.82	0.65	40.019	0.629	92.039	0.92
Γ_c^{100}	99.762	–	63.659	–	100.0	–

As we would expect, for each metric h_i', the performance of our grammars Γ_s^N depends monotonically on the size of the original corpus. This makes sense, as we have already seen how simply incuding a few new rules causes the Chomsky Normal Form grammar to grow very large. Thus, the parser is more flexible in its ability to handle new inputs. In particular, our calculations for h_3 show how important adding just a few more rules can be. For example, $H_3'(\Gamma_s^1)$ indicates that Γ_s^1 was unable to generate a single dependency relation for 96% of its input tokens. Yet even adding just 9 more sentences (as with Γ_s^{10}), improved our rate to just 23%.

Indeed, it appears we can achieve steady improvement among the first few sentence we induce from, although this tapers off quickly. Furthermore, while adding more sentences yields significant improvement for incomplete parses, it is less effective at recognizing Spanish sentences as grammatical, as we can see in the table below:

	P_1 count	$P_1/P_1(\Gamma_c^{100})$ proportion	$P_1/\lvert\mathbb{S}_1\rvert$ relative
Γ_s^1	0	0.0	0.0
Γ_s^{10}	0	0.0	0.0
Γ_s^{25}	4	0.04	0.04
Γ_s^{50}	9	0.09	0.091
Γ_s^{100}	20	0.2	0.202
Γ_c^{100}	99	0.99	–

From this chart, we can get a sense of the shortcomings of using the `nltk.cess_esp` to bootstrap a Catalan grammar. I suspect that most of the good number from the last chart are the result of a few frequently occuring subparses (like DET+NOUN, for example), sort of like Zipf's Law for CKY-parsers. However, parsing an entire sentence requires a very comprehensive grammar, as even a single missing rule will cancel the result. In addition, this poor performance is certainly a consequence of the small sample sizes. Finally, upon inspecting the corpus by hand, I noticed that most (all?) of the sentences very long, and that they tended to have complicated subclause structures.

Beyond surveying the performance of the Γ_s^N grammars on the `nltk.cess_cat` corpus, I also performed the symmetric task of inducing several Γ_c^N grammars from the `nltk.cess_cat`. As before, I tested their performance on the other corpus (in this case, `nltk.cess_esp`), as compared with a baseline grammar Γ_s^{100} induced from `nltk.cess_esp`. The results follow the same general pattern as before, although with much weaker performance. Especially noteworthy are the H_2' scores: approximately half the score (both relative and absolute) as our previous experiment. It is unclear to me why this might be the case, as I would expect a reasonable amount of symmetry, especially considering the closely entwined sociolinguistic history of the speech communities. Both metric tables are presented below:

	H_1' max h	$H_1'/H_1'(\Gamma_s^{100})$	H_2' average h	$H_2'/H_2'(\Gamma_s^{100})$	H_3' nonzero h	$H_3'/H_3'(\Gamma_s^{100})$
Γ_c^1	6.188	0.062	0.935	0.015	15.385	0.154
Γ_c^{10}	22.603	0.228	8.283	0.132	64.664	0.647
Γ_c^{25}	26.94	0.271	10.9	0.174	72.502	0.725
Γ_c^{50}	32.413	0.326	14.063	0.225	75.413	0.754
Γ_c^{100}	42.334	0.426	19.718	0.315	81.1	0.811
Γ_s^{100}	99.327	–	62.63	–	100.0	–

	P_1 count	$P_1/P_1(\Gamma_c^{100})$ proportion	$P_1/\lvert\mathbb{S}_1\rvert$ relative
Γ_c^1	0	0.0	0.0
Γ_c^{10}	0	0.0	0.0
Γ_c^{25}	0	0.0	0.0
Γ_c^{50}	0	0.0	0.0
Γ_c^{100}	2	0.02	0.021
Γ_s^{100}	97	0.97	–