Homework 3

Due Monday April 8

CS 485, UMass Amherst, Spring 2024

Deliverables

Create your writeup as a PDF with any tool you like, and submit to Gradescope. Do not include cky.py directly in your writeup. Submit cky.py or any other additional code to a separate Gradescope assignment entry.

1 CKY Implementation

In this section, you will implement the CKY algorithm for an unweighted CFG. Start with the starter code *cky.py*.

1.1 CKY acceptance

Implement the acceptance version of CKY as `cky_acceptance()`, which returns `True` if there is an "S" covering the entire sentence (in other words, assume S is the only start symbol). Use the "GRAMMAR1" default grammar. Does it return `True` or `False` for the following sentences? Please `pprint()` the chart cells for each case as well.

- the the
- the table attacked a dog
- the cat

Hint: A simple way to implement the chart cells is by maintaining a list of nonterminals at the span. This list represents all possible nonterminals over that span.
Hint: `pprint()`ing the CKY chart cells may be useful for debugging.
Hint: Python dictionaries allow tuples as keys. For example, `d={}; d[(3,4)] = []`. A slight shortcut is that `d[3,4]` means the same thing as `d[(3,4)]`.

1.2 CKY parsing

Implement the parsing version of CKY, which returns one of the legal parses for the sentence (and returns None if there are none). If there are multiple real parses, we don't care which one you print. Implement this as `cky_parse()`. You probably want to start by copying your `cky_acceptance()` answer and modifying it. Have it return the parse in the following format, using nested lists to represent the tree (this is a simple Python variant of the Lisp-style S-expression that's usually used for this.)

```
['S',
        [['NP', [['Det', 'the'], ['Noun', 'cat']]],
         ['VP', [['Verb', 'attacked'],
                 ['NP', [['Det', 'the'], ['Noun', 'food']]]]]]]
```

Print out the parses for the following sentences.

- the cat saw a dog
- the cat saw a dog in a table
- the cat with a table attacked the food

Hint: In the chart cells, you will now have to store backpointers as well. One way to do it is to store a list of tuples, each of which is "(nonterminal, splitpoint, leftchild nonterm, rightchild nonterm)". For example, if the state "('NP', 3, 'Det', 'Noun')" is in the cell for span (2,4), that means this is a chart state of symbol NP, which came from a "Det" at position (2,3) and a Noun at position (3,4). *Hint*: It may be useful to use a recursive function for the backtrace.

1.3 Grammar improvement

Please revise the grammar as follows.

- Add four words to the lexicon: two verbs "attack" and "attacks", and the nouns "cats" and "dogs".
- Revise the rules to enforce subject-verb agreement on number.

The new grammar should accept and reject the following sentences. Please run your parser on these sentences and report the parse trees for the accepted ones. Also, describe how you changed the grammar, and why.

- ACCEPT: "the cat attacks the dog"
- REJECT: "the cat attack the dog"
- ACCEPT: "the cats attack the dog"
- REJECT: "the cat with the food on a dog attack the dog"

Hint: you will need to introduce new nonterminal symbols, and modify the currently existing ones.

1.4 Extra credit: Grammar engineering

Revise the grammar to incorporate coordination or other syntactic phenomena in English. Show examples of what it can analyze, and describe what you did.

2 Dependency parser errors

Run a dependency parser that uses the Universal Dependencies representation and try it out on some examples you make up. Feel free to use a web demo like from CoreNLP (https://corenlp.run/) or from SpaCy (https://demos.explosion.ai/displacy-ent) for this.

Show a sentence where the parser makes an error. Show the parse itself (for example, from a web visualization). Describe the error (including false positive edge(s) involved with the problem), and how to fix the error if you were to manually fix it.

Do this and draw a correct parse for the sentence.

Notes:

- We recommend trying shorter or intermediate length sentences, whose parses are easier to understand. But at the same time, longer sentences are more likely to have errors. You could start with something basic, then iteratively add more and more and see what happens.
- You'll want to read the Universal Dependencies documentation for this, at https://universaldependencies.org/
- Specifically for English relations, the page is: https://universaldependencies.org/en/dep/index.html

3 Word embedding questions

3.1 Cosine similarity

Consider the setting where you want to evaluate cosine similarity between pairs of word embeddings. Section 6.3.3 of your reading discusses creating word vectors based on contextual term co-occurrence counts. (As the reading states, this method can work ok, though typically not as well as model-based dense embeddings.) When using this method, what range of values can cosine similarity take?

3.2 Co-occurrence versus contextual similarity

To model semantic similarity between two words u and v, one way to do it would be to measure how often u and v co-occur; call this a direct co-occurrence measure. This is different than distributional similarity, which is based on similarity of u and v's *contexts*; thus, distributional similarity is based on a a second-order form of co-occurrence. (This applies both to the direct context methods in 6.3.3, but also to word2vec/SGNS or other model-based embeddings, which are trained to summarize this contextual information.) The following questions ask you to explain the difference between co-occurrence versus distributional similarity. You should assume the context window is fairly short.

3.2.1

Give an example of a pair of words where you might expect them to have high co-occurrence, but low distributional similarity, and why. (Assume there is plenty of data about them; explain any assumptions you have about the corpus, or how contexts are defined, if necessary.)

3.2.2

Give an example of a pair of words where you might expect them to have low co-occurrence, but high distributional similarity, and why.

4 Exploring pre-trained word embeddings

Download a version of the GLoVE word embeddings.

4.1 Nearest neighbors

Implement cosine similarity and choose six words. For each, display the top-10 most similar words, and comment on what the model seems to have learned, or failed to learn.

- a name (for example: person, organization, or location)
- a common noun
- a verb
- a word that has an antonym

4.2 Extra credit: Clustering

Consider higher-level structure of what the embeddings have learned. For something like a few hundred words (for example, you could use the 200 most frequent terms in the GLoVE file), run a clustering algorithm from sklearn (for example, k-means or hierarchical agglomerative clustering, ideally using cosine similarity), and print out the words belonging to each cluster.[1]

Comment on a few things that the model seems to heave learned ok, and a few things that don't make sense.

[1]For k-means, just look through each cluster and print out words in each. For hierarchical clustering, words will be leaves in the tree; you can use a dendrogram or just print out the hierarchy yourself with textual or HTML-based formatting as you prefer.