Homework 3

CS 490A, UMass Amherst, Fall 2021

Deliverables

Create your writeup as a PDF with any tool you like, and submit to Gradescope. Do not include large amounts of code in your writeup. (Small snippets may be appropriate if they help you answer a question in a concise and useful way.) Submit your code to a separate Gradecope assignment entry.

1 NLP software setup

For the upcoming implementation questions, you will use the output of an NLP package that can produce POS tags, NER, and dependency parsing. We recommend using either SpaCy (https://spacy.io/) or Stanza (https://stanfordnlp.github.io/stanza/), which are Python libraries. Another option is CoreNLP (https://stanfordnlp.github.io/CoreNLP/), which is in Java and is easiest to use from the commandline. Whatever your choice, get it installed and running on your machine.

You'll also be using a dataset from the NLTK *corpora* package, which is another Python library, and finally, your Twitter dataset from HW1.

1.1

Report what NLP software package(s) you are using, including the software's version number, and the names of the specific models used for your analyses.

This information is very important for replicability, and will be required for your project reports. You may want to do this part after you've finished the rest of the assignment and you know which software versions you used, after confirming they worked for you.

2 JK discussion questions

We will refer to the Justeson and Katz (1995) reading as "JK."

2.1

JK discuss the difference between *lexical* and *nonlexical* noun phrases. Please describe them and the distinction between them, in your own words. Give examples for both, and discuss why they illustrate the criteria.

(Note: in NLP, the term "multi-word expression" is sometimes used for the concept that JK calls a "lexical phrase.")

2.2

Give two reasons why it might be useful to extract lexical noun phrases from a text. At least one of your reasons should be one not thought of by JK; say which one it is.

2.3

Consider the POS regular expression given in JK (printed page 16-17), which uses their system of 3 tags. They show the set of all length 2 and length 3 POS sequences that match the expression. Please list all length 4 POS sequences (that is, POS 4-grams) that match it.

3 Implementing JK

Using a POS tagger that outputs using the PTB tagset (don't use a coarse tagset), implement a noun phrase phrase extractor by using the JK pattern from the paper.

3.1 PTB tagset adaptation

You will have to make some adaptations to work with the PTB tagset; in particular, you have to decide which PTB tags should count for each of the three different tags in the JK pattern. List your decisions for these, and justify them. (There is not necessarily one right answer—recall our discussion in class of one tricky POS tag.)

3.2 Implementation and corpus report

Implement your phrase extractor in Python! Run it on the *nltk.corpora.webtext* corpus of text from a bunch of webpages, and print out the 20 highest frequency noun phrases, of length 2 or higher, in the corpus.

Implementation hints:

- We recommend defining a function that takes in a tagged sentence, then uses the JK pattern to identify a set of spans[1] matching the pattern, then returns either a list of spans, or maybe a list of strings for the phrases, that it identifies.
- There are a bunch of ways to implement the pattern. It might be convenient to use the Python regular expression library `re`, which requires turning the tagged sentence into a single string. You could do it in the format "WORD_TAG" for each token, or you could make a string only with the tags, then use start/end match positions to figure out the corresponding token positions in the original tokenized sentence.

3.3 Comparison study

When defining rules or heuristics, it's useful to experiment with them to see what may work better or worse. You could add a feature thing to try to improve it; or, you could remove something that's good about it, to figure out how useful that feature is (this is called an "ablation test"). When you're developing rules, you may not necessarily have labeled data like in supervised machine

[1] A *span* within a sentence is a pair integers describing the start and end tokens of subsequence of tokens in the sentence. For example, the span (3,5) for the previous entence refers to "within a", if you're using Python's convention of a closed-open interval in slicing notation, as in `tokens[3:5]`.

learning. But a very simple and useful empirical methodology is to compare the output of the new and old system on the same dataset.

Choose a well-motivated change to your system, and implement it as a separate function (or option to your current function, whichever). Let "OLD" be the system you used in the previous question. After your change, you now have a "NEW" system.

Describe what you changed. Then run both OLD and NEW on the webtext corpus, saving the output from both. Print out samples of phrases that OLD identifies but NEW does not, and ones that NEW identifies but OLD does not. (Maybe 5 or 10 or so from each set difference.) What has changed? Which seems better?

4 Social Media Error Analysis for NER

The NLP software package you're using was probably not developed for social media, or in particular, Twitter text. Let's test how well it does. This general problem is called *domain shift*, when something about the training data is systematically different than the data used at runtime; here, there's a shift in many things, including genre.

We'll look at named entity recognition (NER) in this problem.

Run the system's named entity recognizer on your tweet dataset from HW1. How well does it do? Spend some time reading through the system's output and thinking about what it's getting right and wrong, before summarizing your findings. This is called *error analysis*. In fancier error analysis you can manually annotate the error types and do a statistical report; but here it's fine to be qualitative.

4.1

Describe at least three types of errors the NER system tends to make on the tweets. For each error type, give an example. Make sure to describe both false positive and false negative errors.

4.2

Propose a feature that might help fix some of the errors you observed, and explain why it might help.

(This is a research hypothesis; if this were your project, you could then try actually doing it!)

5 Dependency Parsing

Please run the dependency parser on some example sentences, to familiarize yourself with what it outputs. Use any language you're familiar with (not necessarily English). Feel free to use examples from the previous questions, or modify them or make up examples of your own. We suggest looking at parses of short sentences, which are easier to understand.

Find an example where, according to the annotation guidelines of the dependency formalism that your dependency parser uses, the parser made an error.

5.1

Please show the sentence and its automatic parse. Show the parse both in a textual format you print out by using the parser API, and a graph visualization version you draw yourself, showing

the dependency tree over the sentence.

For the textual format, we recommend printing out each word token, and for each token, printing out what word its parent is and what the labeled edge is between them (assuming your parser is outputting a strict tree structure, where each token has exactly one parent).

For the visualization, we suggest drawing this by hand, since that tends to be easiest. This is why shorter sentences are better here! Please do not use spaCy's automatic visualizer (it's pretty terrible, in Prof. O'Connor's opinion).

5.2

Describe the error that this parse has for this sentence. Explain how it should be corrected, and draw a corrected version of the parse tree.

When explaining the error, please refer to documentation about the dependency parsing model you're using, with a precise citation to a section of a paper or URL webpage if appropriate,[2] that justifies your claim.

[2]For example, if your parser uses Universal Dependencies, you may want a webpage from its website: https://universaldependencies.org/