NLP (for Social Science)

Robert Henderson

The University of Arizona
S/WCISS

Outline

What is NLP?

Natural Language Processing, broadly, concerns the automatic manipulation of language—e.g., speech, text—through software.

What is NLP?

We have discovered that these manipulations become much easier and more accurate when using massive amounts of data paired with statistical models—because of this NLP has close connections with

1. Big Data
   State of the art language models are trained on corpora of hundreds of billions of words.
2. Artificial Intelligence
   e.g., GPT-3, a neural net language model with 175 billion parameters.

Common NLP Tasks

Because NLP concerns manipulating natural language, it is natural to define NLP through extension, pointing at the kinds of manipulations that are most common / actively researched.
Phonetic / Phonological Tasks

- **Speech recognition:** Given an audio recording of language, convert it to text.
- **Text-to-speech:** Given language text, convert it to spoken audio.

Common NLP Tasks

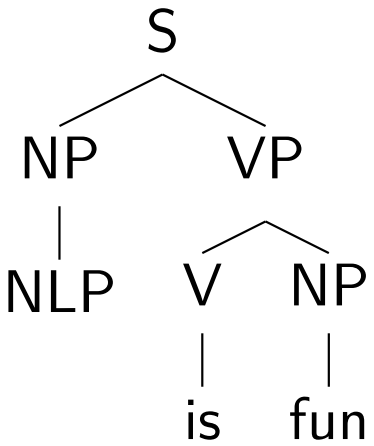
Syntactic Tasks

- **Word segmentation:** Given a string of text, recover the phonological words. Trivial for written English, but not for many languages—e.g., in Mandarin 简单 'simplicity', but 简 'simple' and 单 'single'.
- **Morphological segmentation:** Given a word, break it down into its constituent parts—e.g., unbreakable → un-break-able
- **Part-of-speech tagging:** For each token of text, assign a grammatical category—e.g., The$_{\textit{article}}$ boy$_{\text{NOUN}}$ left$_{\text{VERB}}$.

Common NLP Tasks

Syntactic Tasks

- **Named entity recognition:** Given text, identify every proper name and type (person, organization, location, business, etc.)
- **Parsing:** Given a sentence, assign a grammatical representation / parse tree of that sentence—e.g,



- **Grammar induction:** Given text in a language, produce a description of that language's syntax—e.g., a strong form would be to induce a parser from that text.

Common NLP Tasks

Semantic Tasks

- **Coreference resolution:** Given text, determine which expressions refer to the same entities—e.g. The doctor$_i$ greeted the patient$_j$ and then he$_i$ listened to his$_j$ heart.
- **Relationship extraction:** Given text, find determine the relationships between entities—e.g., Ferid was born in Topeka, Kansas. He played football for Topeka High School, before moving to Austin, Texas. He lives there today. [BIRTHPLCE(Ferid, Topeka), AIMA_MATER[Ferid, Topeka HS], RESIDENCE(Ferid, Austin)]
- **Semantic Parsing:** Given a sentence, produce a formal representation of its semantics—e.g., Every student hugged a teacher $\rightsquigarrow \forall x[student(x) \rightarrow \exists y[teacher(y) \wedge hug(x, y)]]$

Common NLP Tasks

Semantic Tasks

- **Recognizing Textual entailment:** Given text and a sentence, determine whether the text (defeasibly) ensure the sentence is true—e.g., (1) A soccer game with multiple males playing. (2) Some men are playing a sport. Does (1) $\Rightarrow$ (2)?
- **Sentiment analysis:** Given a text, determine the emotional valence, especially towards a given entity—e.g., "This paint is top of the line don't know how to describe the vibrancy of the color" <— Guess how many stars was attached to this amazon review.

Common NLP Tasks

High level tasks

- **Text summarization:** Produce fluent, readable summaries of given text.
- **Machine translation:** Translate text from one human language to another.
- **Natural language generation:** Convert information from a database into readable human language.
- **Natural language understanding:** Covert text into a formal semantic representation or set of representations (a kind of end case of semantic parsing)
- **Question answering:** Given a human language question, determine its answer.

How I use NLP

I primarily use NLP to help speed the documentation, description, and revitalization of endangered minority languages.

Preparing natural language data

As with all data, we often want to clean natural language data before working with it. The most common steps (which will vary depending on task):

- Strip markup (HTML, XML, etc., Punctuation)
- Tokenize
- lemmatize (or Stem)
- Remove stop words

Tokenization

The task of chopping a text up in to expressions—tokens—at the level of interest.

- Most commonly, one tokenizes at the word level. This is pretty much trivial to do in English using whitespace (but beware, e.g., *New York*.)
  - If you need to be careful dictionary-based tokenizers are standard in text processing packages.
- It is often common to tokenize at the level of the morpheme (i.e., minimal unit of meaning). This must be done in tandem with with morphological segmentation
  The cats have been walking on the building. $\rightsquigarrow$
  the cat s have be en walk ing on the build ing
- lemmatize (or Stem)
- Remove stop words

Lemmatization

Sparsity is a common problem when analyzing natural language data. In many applications we would like to reduce the amount of non-relevant variation in forms as much as possible. Grammar often generates a proliferation of form:

- car, cars, car's, cars'
- culture, cultures, culturing, cultural, acculturate
- is, are, be

Lemmatization reduce all expressions in a text to their root forms, e.g., car, culture, be.

- We can also lemmatize at the morpheme level. For instance, -s and -es are two ways of writting plural that we might want to collapse.

Lemmatization

Full lemmatization can be hard if working on low resource languages. It requires the ability to morphologically parse text. One can often get by with stemming, i.e., chopping off part of the word in hopes of reaching the stem.

- for instance, in Mayan languages you can stem pretty effectively by deleting a few strings if they occur at the beginning of words (prefixes), and then taking the first three characters.
- there are tricks for English as well, e.g., Portner's algorithm.

Stop words

Once again, we often want to reduce the ways in which texts can vary in order to deal with sparsity. We can reduce the size of a text and decrease variation without losing much information by eliminating super frequent words.

- e.g., a, an, and, are, as, at, be, by, for, from, has, he, in, is, it, its, of, on, that, the, to, was, were, will, with

You may have to hand-make a list of stop words for the corpus you are working with. In many state of the art algorithms removing stop words doesn't affect performance much.

- But before throwing the big guns at a problem it is often helpful to run simpler, older alogorithms at problems and here stop words can help.

Representing language

We rarely want to work with raw language / text, but instead transform it in some way.

- **Approximating syntax:** n-grams
- **Approximating morphology:** character n-grams
- **Approximating a text:** bag-of-words
- **Richer syntax:** dependency parsing
- **Richer semantics:** vector-based representations

N-grams

Syntax concerns how words are grouped into phrases. Instead of dealing with the actual syntax of the language we can get approximations by working with n-grams—usually bi- or tri-grams.

- The quick horse raced past the barn.
  the quick, quick horse, horse raced, raced past, past the, the barn
  with stop words:
  quick horse, horse raced, raced past, past barn

Knowing that these word units occur to together gives us a much better sense of the sentence than an undifferentiated list of words.

- we then can plug our n-grams into whatever other NLP task we have—e.g., a language model, comparing document similarity, etc.

Character n-grams

We can play the same trick with characters and get a rough approximation of morphology—i.e., how words are constructed out of roots and affixes. Consider the character tri-gram representation of the phrase "reheating the pots"

- reheating the pots
  reh hea ati ing the pot ots

Once again, these n-grams can be fed as features into whatever NLP task we have at hand, but people have found character n-grams are great for language models, authorship attribution, plagiarism detection, identifying language varieties.

Bag-of-words

Language is highly structured, and we have been considering was of reducing structure, while still capturing aspects of it (syntax / morphology). The most common way to reduce structure in a way that scales up to entire documents, is the bag-of-words model.

- The idea is to treat a chunk of text as a set of words with a count of how often each word occurs. Think of it as like the fingerprint of a text.

Bag-of-words

The idea is to treat a chunk of text as a set of words with a count of how often each word occurs.

- { a:3, of:2 the:1, idea:1, is:1, to:1, treat:1, treat:1, chunk:1, words:1, with:1, count:1, how:1, often:1, each:1, words:1, occurs:1}

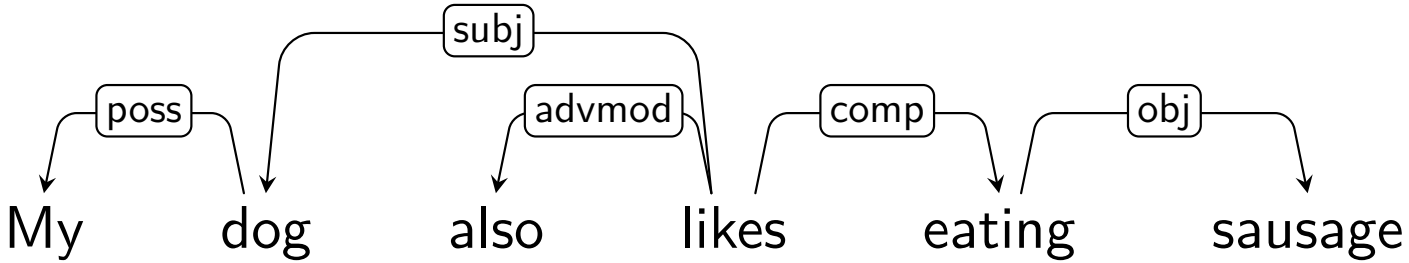
We often combine notions we have seen before. That is, we could lemmatize and use stop words to reduce the variety of words in a corpus of texts, and then create a bag-of-words model over text bi-grams.

- We can then think of two sentences or texts as similar if their fingerprints are similar.

Dependency grammars

The most common richer kind of syntactic representation you will see are dependency parses.

- Idea: the syntactic structure of a sentence is represented as a direct graph with an arrow from *a* to *b* if *b* modifies *a*.
- We can get fancy and label arrows with the kind of modification.



Lots of off the shelf parsers, like the Stanford Neural Network Dependency Parser—part of CoreNLP. See also Python NLTK, which provides a wrapper to MaltParser.

Vector-based representations

Currently, the richest of king of lexical semantic representation you will see are *(word) embeddings*.

- Idea: Assign to each word a vector of real numbers that approximates the the meaning of the word in the following way—words that are similar along some axis of meaning will be (cosine) close in vector space in some dimension.

Vector-based representations

But how do we get these vectors?

- We train a neural network to predict the likelihood of a context give a word—e.g. predict how likely a given word is to occur in an n-gram with it.
- The weights of neurons that minimize the error in this prediction task for each word become our vector.

You will most likely not be training these yourself but can get these vectors, these word-embeddings that other have computed—e.g., Word2Vec, ELMO, BERT, GloVe, fastText.