LING 331
Text Processing for Linguists

Week 9

Python for Text (and Beyond)
Roadmap for Our Last Two Days

Wednesday 3/2

● Assignment 6 FYIs
● Content:
  Dependency Parsing
  WordNet
  Word Vectors
● Final Assignment

Monday 3/7

● Assignment 6 Notes
● Content:
  Classification
● Final Self-Evaluation
● Where To Go From Here
Notes for Assignment 6

- PRON- is a spaCy idiosyncrasy
  - Some weird version issues though...

- Stemming vs. Lemmatization
  - Stemmers are a much more coarse heuristic algorithm
  - Lemmatizers are machine learning models
    - more computationally expensive, but not crazily so
Notes for Assignment 6

- Sometimes you get a nice docstring, comments, etc sometimes you don’t!
- Figuring out types of objects:
  - `type(obj), dir(obj), help(obj), print(obj)`
- If you’re running into trouble this is the first thing to try!
- With dicts, useful also to `print(d.keys())`
- E.g. sentences in 2.g., what’s in a row in 5.a.
Notes for Assignment 6

- Nested dictionaries: dicts are key-value, but value can be anything, including another dict

```python
text = {}
...
# do some stuff

if not word in text:
    text[word] = {}
...
# do some stuff

text[word][emotion] = score
```
Notes for Assignment 6

- **left_adjectives:**
  This is another common sort of programming meme, requires a sort of “spatial orientation” / “navigation” skill

- **enumerate to maintain an index,**
  when current word matches, check index - 1

- **Working with dependency trees is a yet-trickier version of this meme!**
Dependency Parsing gives a syntax representation

- Words are connected to other words with a tag representing their relationship
- Main verb is the sentence root
- Directed: head → dependent
- Tag is role the played by the dependent

[Links]
https://spacy.io/usage/visualizers
https://explosion.ai/demos/displacy
Dependency Parsing gives a syntax representation

- Most common formalism for syntax in Comp Ling / NLP
  - Interesting contrast with formal syntax!
- Partially because of computational feasibility
- Very exciting project: Universal Dependencies
  - [https://universaldependencies.org/](https://universaldependencies.org/)
  - (you can contribute!)
Dependency Parsing gives a syntax representation

- spaCy does dependency parsing inherently (if you don’t disable "parser")
- Access dependency tag with `token.dep_`
  List of children with `token.children`
- More info:
  [https://spacy.io/usage/linguistic-features](https://spacy.io/usage/linguistic-features)
WordNet is a lexical resource for semantic relations

- Represents semantic relationships in a large network
- Allows to calculate e.g. “path similarity”
- Play with directly: 
  [http://wordnetweb.princeton.edu/perl/webwn](http://wordnetweb.princeton.edu/perl/webwn)

Figure 19.6 WordNet viewed as a graph. Figure from Navigli (2016).
WordNet is a lexical resource for semantic relations

- NLTK has an interface for working with WordNet
- ... but it’s not the most intuitive thing in the world
- More info here:
  https://www.nltk.org/howto/wordnet.html
Sparsity is a property of natural language

- Language is creative, flexible, and ever-evolving; there are many ways to say the “same thing”
- Translations for instance! But even within a language.

Q: Where is he?

He went to the store

Oh, Johnny left to get groceries

Out to grab the essentials
Sparsity is a property of natural language

- Zipf’s Law:

  If you order words by frequency rank, e.g.

  1  the  Counts will be inversely proportional to rank!
  2  and
  3  I
  4  to
  5  of

  So ‘the’ will appear roughly 2x as often as ‘and’, which will appear 2x as often as ‘I’, etc.
Zipf’s Law
Across languages on Wikipedia

https://en.wikipedia.org/wiki/Zipf%27s_law
Sparsity is a property of natural language

**Closed-Class Words**
- of, she, or, the, no, and
- a.k.a. ‘function words’
- Includes pronouns, articles, conjunctions, particles
- Rarely gain new members
- Very dense!
- Perform grammatical and discourse functions

**Open-Class Words**
- walrus, fleek, margarine, poindexter
- a.k.a. ‘content words’
- Includes nouns, verbs, adjectives, etc.
- Frequently gain new members
- Very sparse!
- Perform semantic functions, i.e. carry most of the meaning
Sparsity is a problem for computing with language

'cat' != 'cat,' != 'CAT' != ‘Cat’ != ‘cats’

We’ve seen some ways to deal with this:

- Stripping punctuation
- Downcasing
- Tokenization
- Stemming
- Lemmatization

... and more abstractly:

- POS tagging
- Lexicons (concreteness, emotion)
- Syntactic roles and relations
Sparsity is a problem for computing with language

But what if we want a different semantic operation than a pure exact match?

For instance, how can we know if words are more or less similar?

Answer: create a numerical representation that can be operated on mathematically - word vectors!
Word Vectors provide a numerical representation of the meaning of a word

- Key mathematical notes:
  - A vector is simply a list of numbers
  - Those numbers form an abstract representation of a word
  - Each “dimension” refers to the number at a certain index
  - Dimensions can be meaningful or not depending on how the vectors are constructed
Word Vectors provide a numerical representation of the meaning of a word.

You could imagine manually constructing them:

<table>
<thead>
<tr>
<th>word</th>
<th>cuteness</th>
<th>furriness</th>
<th>animacy</th>
<th>growth_stage</th>
</tr>
</thead>
</table>
Word Vectors provide a numerical representation of the meaning of a word.

You could imagine manually constructing them:

<table>
<thead>
<tr>
<th>word</th>
<th>cuteness</th>
<th>furrriness</th>
<th>animaey</th>
<th>growth_stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>
Word Vectors provide a numerical representation of the meaning of a word.

You could imagine manually constructing them:

<table>
<thead>
<tr>
<th>word</th>
<th>cuteness</th>
<th>furrriness</th>
<th>animaey</th>
<th>growth_stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>kitten</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>-4</td>
</tr>
</tbody>
</table>
Word Vectors provide a numerical representation of the meaning of a word.

You could imagine manually constructing them:

<table>
<thead>
<tr>
<th>word</th>
<th>cuteness</th>
<th>furrriness</th>
<th>animaey</th>
<th>growth_stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>kitten</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>-4</td>
</tr>
<tr>
<td>lizard</td>
<td>-3</td>
<td>-8</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
Word Vectors provide a numerical representation of the meaning of a word

You could imagine manually constructing them:

<table>
<thead>
<tr>
<th>word</th>
<th>cuteness</th>
<th>furriness</th>
<th>animacy</th>
<th>growth_stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>kitten</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>-4</td>
</tr>
<tr>
<td>lizard</td>
<td>-3</td>
<td>-8</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>houseplant</td>
<td>2</td>
<td>-4</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
Word Vectors provide a numerical representation of the meaning of a word.

You could imagine manually constructing them:

<table>
<thead>
<tr>
<th>word</th>
<th>cuteness</th>
<th>furriness</th>
<th>animacy</th>
<th>growth_stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>kitten</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>-4</td>
</tr>
<tr>
<td>lizard</td>
<td>-3</td>
<td>-8</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>houseplant</td>
<td>2</td>
<td>-4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>teddy_bear</td>
<td>6</td>
<td>6</td>
<td>-10</td>
<td>0</td>
</tr>
</tbody>
</table>
Word Vectors provide a numerical representation of the meaning of a word

- But this would be unthinkably time-consuming and arbitrary
- Solution: the distributional hypothesis
  “You shall know a word by the company it keeps.”
  -Firth 1957
- Intuitively:
  - “Cat” occurs near “furry”, “claws”, “cute”, “feline” in everyday speech, so does “kitten”, so they are similar.
Word Vectors provide a numerical representation of the meaning of a word

- So, use word vectors generated from co-occurrence statistics
- Methods described in more detail in SLP Ch. 6
  - Raw co-occurrence counts, TF-IDF, PPMI
Word Vectors provide a numerical representation of the meaning of a word

- These are still relatively sparse; most words don’t co-occur with most other words, matrix is full of many zeroes
- Solution: Machine learning approach (e.g. word2vec)
- Generates compressed vectors of dimension ~500
  - Pro: learn dense vectors implicitly from natural language!
  - Con: dimensions become much less interpretable!
“Embeddings” are the same as vectors

Representation is “embedded” in a shared “vector space” with other representations (i.e. they have comparable dimensions)

Figure 6.1 A two-dimensional (t-SNE) projection of embeddings for some words and phrases, showing that words with similar meanings are nearby in space. The original 60-dimensional embeddings were trained for sentiment analysis. Simplified from Li et al. (2015).
Final Assignment

- More and more on your own! Get creative!
- **Key point!**
  - If you want to use LDC or BYU data, let me know by Wednesday
- Please turn in on time! March 15th, where you’re at
  - For grading purposes, but I’m always available to talk more later if you keep working on it!