LING 300 - Topics in Linguistics: 
Introduction to Programming and Text Processing for Linguists

Week 9

Python for Text 2 (and Beyond)
Roadmap for This Week

Monday

- Content: Dependency Parsing
  WordNet
- Final Assignment
- Some Applications of What We’ve Learned
  (from my research)

Wednesday

- Final Self-Evaluation
- Assignment 6 Notes
- Content: Word Vectors Classification
- Where To Go From Here
Notes from Assignment 6

- _PRON_ is a spaCy idiosyncrasy

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

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

```python
intensity = {}
for row in csv.DictReader(open(f), delimiter='\t'):
    word = row['word']
    emotion = row['emotion']
    score = row['emotion-intensity-score']
    if not word in intensity:
        intensity[word] = {}
    intensity[word][emotion] = score
```
Notes from 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!**
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
  2 and
  3 I
  4 to
  5 of

  Counts will be inversely proportional to rank!

  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>
<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!
Word Vectors == Word embeddings

- “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).
Classification is the task of assigning labels

- Use known input-label pairs to train an algorithm to decide which category a previously unseen input belongs to
Features are leveraged to make predictions

- Features can take many forms:
  - Counts of particular words
  - Counts of $n$-grams
    - multi-word phrases of length $n$:
      - e.g. trigrams are three-word phrases ("so it goes")
  - Numerical values (e.g., average concreteness)
  - Word vector dimensions

- Each is part of a mathematical representation of a document
Features are leveraged to make predictions

- “Learning” is most frequently the process of assigning numerical weights to each feature

NLTK movie review classification example:

```python
>>> print(nltk.classify.accuracy(classifier, test_set))
0.81

>>> classifier.show_most_informative_features(5)
Most Informative Features
contains(outstanding) = True          pos : neg  =  11.1 : 1.0
contains(seagal) = True               neg : pos  =   7.7 : 1.0
contains(wonderfully) = True          pos : neg  =   6.8 : 1.0
contains(damon) = True                pos : neg  =   5.9 : 1.0
contains(wasted) = True               neg : pos  =   5.8 : 1.0
```

https://www.nltk.org/book/ch06.html
Congratulations!

You are all officially computational linguists!
Programming is very useful

- The skills you’ve learned are broadly applicable to linguistic and non-linguistic applications

- Try out your new computational tools and thinking in other parts of your life!
Other things you are now well-equipped to start learning

- Version control (git, see these lectures)
- Data science (see e.g. pandas and numpy)
- Machine learning (see e.g. scikit-learn)
- Web scraping (see e.g. BeautifulSoup)
- Dynamic web programming (see e.g. Flask or Django)
- App development (see e.g. Kivy)
- Game programming (see e.g. pygame or Godot)
Natural Language Processing (NLP) and Computational Linguistics (CL)

- NLP = more engineering, everything is a “task”, focus on system performance
- CL = computational social science, using and developing NLP tools for social, linguistic, humanistic questions
- No need, of course, to strictly pick a camp!
AI and Deep Learning


- and these more advanced lectures (Stanford CS224N):
  https://www.youtube.com/playlist?list=PLoROMvodv4rOhcuXMXkNm7j3fVwBBY42z
Interacting with me!

- *Next Spring*
  Ling 334 - Introduction to Computational Linguistics
  - Covering more advanced and modern methods
  - This class is sufficient background

- Always interested to chat about research projects etc!
Thank you!

It’s been a privilege and a joy to teach this class.