Lecture 19: Part-of-Speech Tagging

Ling 1330/2330 Intro to Computational Linguistics
Na-Rae Han, 11/2/2023

Outline

- Ex9 review

- Part-of-speech tagging
  - *Language and Computers*, Ch. 3.4 Tokenization, POS tagging
  - NLTK Book Ch.5 Categorizing and tagging words
- Parts of speech
- POS ambiguity
- POS-tagged corpora
- N-gram taggers

POS tagsets

- There are multiple POS tagsets for English in use.
  - Some are larger, some are smaller.
- **The Brown Corpus tagset** (87 tags)
  - http://clu.uni.no/icame/manuals/BROWN/INDEX.HTM
- In NLP, **the Penn Treebank tagset** (45 tags) has become de facto standard.
  - http://www.surdeanu.info/mihai/teaching/ista555-fall13/readings/PennTreebankTagset.html
  - This is the default tagset for `nltk.pos_tag()`.
- NLTK lets you load a POS-tagged corpus using "**Universal**" **POS tagset** (only 12 tags).
  - http://www.nltk.org/book/ch05.html#a-universal-part-of-speech-tagset

'so': three Parts-of-Speech

- RB (Adverb)
  - "I told you **so**."
- QL (Qualifier)
  - "We've always been **so** close."
- CS (Subordinating conjunction)
  - "**So** she couldn't choose Rev as a confidant;;"

What POS frequently precede each?

What POS frequently follow?

POS tags around 'so'

- Review Exercise 9
- We will look into the Brown corpus.

- The questions are best answered through **conditional frequency distribution**:
  - Condition: current word and its POS ("so/RB", "so/QL"…)
  - Outcome:
    - Preceding POS tag
    - Following POS tag

- Demo time!

  ← IDLE shell session posted separately

POS ambiguity in Penn Treebank

Some words can take on **multiple parts-of-speech:**

I asked him a question. / They wanted to question him.

*Time flies **like** an arrow, and fruit flies **like** bananas.*

← How are these represented in Penn Treebank?

← How to find different tags for a word?

```
>>> tb_cfd = nltk.ConditionalFreqDist(treebank.tagged_words())
>>> tb_cfd['question']
FreqDist({'NN': 12, 'VB': 1, 'VBP': 1})
>>> tb_cfd['flies']
FreqDist({'VBZ': 1})
>>> tb_cfd['like']
FreqDist({'IN': 44, 'VB': 8, 'VBP': 4, 'JJ': 1})
>>> nltk.help.upenn_tagset('IN')
IN: preposition or conjunction, subordinating
astride among uppon whether out inside pro despite on by throughout
below within for towards near behind atop around if like until …
>>> tb_cfd['share']
FreqDist({'NN': 116, 'VB': 3})
```

Build a Conditional Frequency Distribution where **condition**: word, **sample**: POS tag

Designing a POS tagger: simple but flawed

1. Tag everything a NOUN.
   - Why? Because NOUN is the most common POS.

   * Problem? Poor coverage.
2. Consider the morphology.
   - Ends in 'ly' → ADV
   - Ends in 'ed' → VERB

   *Problem? Can be wrong: "fly" is not an adverb. Not every word has an identifiable morphological marker.
3. Maintain a dictionary of word and its POS. For each word, simply look up its tag in the dictionary.

   * Problem? Ambiguity. 'question' can be both NOUN and VERB, depending on **context**!

Taking context into consideration

1. The dictionary lists the *most common* POS tag for a word.
   - 'question' → **NN** (more freq. than VB)
2. Instead of just individual word, the dictionary lists the most common tag for <u>the preceding POS + the word</u>.
   - 'MD question' → **VB**, 'AT question' → **NN**
3. Why stop at just *one* preceding POS? Consider *two*.
   - 'BEZ AT cold' (*<u>is a cold</u> month*) → **JJ**
   - 'HV AT cold' (*have a cold*) → **NN**

***Brown corpus tagset.**
NN: singular noun, VB: verb base form, MD: modal auxiliary, AT: determiner, JJ: adjective, BEZ: *is*, HV: *have*

N-gram taggers

1. The dictionary lists the *most common* POS tag for a word.
   - 'question' → **NN** (more freq. than VB)

Unigram Tagger

2. Instead of just individual word, the dictionary lists the most common tag for the preceding POS + the word.
   - 'MD question' → **VB**, 'AT question' → **NN**

Bigram Tagger

3. Why stop at just *one* preceding POS? Consider *two*.
   - 'BEZ AT cold' (*is a cold month*) → **JJ**
   - 'HV AT cold' (*have a cold*) → **NN**

Trigram Tagger

→ ***n*-gram tagger**.

→ The statistical patterns can be extracted from annotated corpora.

The bigger the context the better?

- So, a trigram tagger will always outperform a bigram tagger, right? And bigram taggers are better than unigram taggers?
  - Not in isolation.

spacing before . so I can use .split() for tokenization

```
>>> unigram_tagger.tag('It was a bright cold day in April .'.split())
[('It', 'PPS'), ('was', 'BEDZ'), ('a', 'AT'), ('bright', 'JJ'), ('cold',
'JJ'), ('day', 'NN'), ('in', 'IN'), ('April', 'NP'), ('.', '.')]
>>> bigram_tagger.tag('It was a bright cold day in April .'.split())
[('It', 'PPS'), ('was', 'BEDZ'), ('a', 'AT'), ('bright', 'JJ'), ('cold',
None), ('day', None), ('in', None), ('April', None), ('.', None)]
```

  - The larger the context, the more specific it gets.
  - The chance of a particular context not found in the corpus data increases.
  - This creates the **sparse data problem**.

Addressing sparse data problem

- Combine *n*-gram taggers as stacked **back-off models**:
  1. Look up `"POS`$_{n-2}$ `POS`$_{n-1}$ `word"` in the 3-gram tagger.
  2. If it's not found, look up `"POS`$_{n-1}$ `word"` in the 2-gram tagger.
  3. If it's not found, look up `"word"` in the 1-gram tagger.
  4. If it's not found (unknown word), use the Default Tagger where everything gets tagged NOUN.

- This is how NLTK's n-gram tagger is implemented:
  - https://www.nltk.org/book/ch05.html#n-gram-tagging

Building an n-gram tagger

```
>>> t0 = nltk.DefaultTagger('NN')
>>> t1 = nltk.UnigramTagger(train_sents, backoff=t0)
>>> t2 = nltk.BigramTagger(train_sents, backoff=t1)

>>> april = 'It was a bright cold day in April.'
>>> t2.tag(nltk.word_tokenize(april))
    [('It', 'PPS'), ('was', 'BEDZ'), ('a', 'AT'), ('bright', 'JJ'),
    ('cold', 'JJ'), ('day', 'NN'), ('in', 'IN'), ('April', 'NP'),
    ('.', '.')]
>>> t2.evaluate(test_sents)
    0.8452476038338658
```

- While building each n-gram tagger, the "n-1"-gram tagger is designated as the back-off model.

Preparing training/testing sets

- Training data: first 90% of 'news' section of Brown
- Testing data: last 10% of the same

```
>>> brown_tagged_sents = brown.tagged_sents(categories='news')
>>> len(brown_tagged_sents)
    4623
>>> cutoff = round(len(brown_tagged_sents) * 0.9)
>>> cutoff
    4161
>>> train_sents = brown_tagged_sents[:cutoff]
>>> test_sents = brown_tagged_sents[cutoff:]
>>> len(train_sents)
    4161
>>> len(test_sents)
    462
>>>
```

Tokenized SENTENCES, not word tokens, are units of training/testing

Now build t0, t1, t3 on `train_sents`.

Evaluating a tagger

- Compare the output of a tagger with a human-labelled (presumed "correct") **gold standard**

```
>>> len(test_sents)
    462
>>> t2.evaluate(test_sents)
    0.8452476038338658
>>> test_sents[341]
    [('None', 'PN'), ('of', 'IN'), ('these', 'DTS'), ('countries', 'NNS'),
    ('is', 'BEZ'), ('happy', 'JJ'), ('with', 'IN'), ('these', 'DTS'),
    ('arrangements', 'NNS'), ('.', '.')]
>>> [wd for (wd, tag) in test_sents[341]]
    ['None', 'of', 'these', 'countries', 'is', 'happy', 'with', 'these',
    'arrangements', '.']
>>> t2.tag([wd for (wd, tag) in test_sents[341]])
    [('None', 'NN'), ('of', 'IN'), ('these', 'DTS'), ('countries', 'NNS'),
    ('is', 'BEZ'), ('happy', 'JJ'), ('with', 'IN'), ('these', 'DTS'),
    ('arrangements', 'NNS'), ('.', '.')]
>>> t2.evaluate([test_sents[341]])
    0.9
```

A list of *one* sentence

Wrapping up

- Homework 7 out
  - Build a bigram POS tagger
- Next Wed: PyLing ➔
  - Over Zoom (link at MS Teams)

- Nov 16 (Thu) class will be remote, over Zoom.

- Final exam schedule announced!
  - 12/13 (Wed) 4-5:50pm
  - At LMC's PC lab (G17 CL)