

Default tagger: use the most frequent tag

```
>>> tokens = "Thieves leave young athletes in  
the dark".split();  
>>> default_tagger = nltk.DefaultTagger('NN')  
>>> default_tagger.tag(tokens)  
>>> nltk.tag.accuracy(default_tagger,  
nltk.corpus.brown.tagged_sents(categories =  
'a'))
```

Is a default tagger good for anything?

The regular expression tagger

- VB: base form, e.g., 'go'
- VBZ: 3rd person singular present, e.g.,
goes
- VBN: past participle, e.g., 'gone'
- VBG: gerund, e.g., 'going'
- VB: simple past, 'went'

The regular expression tagger

```
>>> patterns = [  
    (r' .*ing$', 'VBG'),  
    (r' .*ed$', 'VBD'),  
    (r' .*es$', 'VBZ'),  
    (r' .*ould$', 'MD'),  
    (r' .*\'s$', 'NN$'),  
    (r' .*s$', 'NNS'),  
    (r' .*', 'NN')  
]  
  
>>> regexp_tagger = nltk.RegexpTagger(patterns)  
>>> regexp_tagger.tag("It got more words  
right".split())
```

The lookup tagger

- The idea: for each word, identify its most frequent tag, and use this information to tag words in next text

```
>>> fd =  
    nltk.FreqDist(nltk.corpus.brown.words(categories='a'))  
>>> most_freq_words = fd.sorted()[:100]  
>>> cfd =  
    nltk.ConditionalFreqDist(nltk.corpus.brown.tagged_words(categories='a'))  
>>> likely_tags = dict((word, cfd[word].max()) for word  
    in most_freq_words)  
>>> lookup_tagger =  
    nltk.UnigramTagger(model=likely_tags)
```

The role of context in POS tagging

- Water
- Water the
- The water

Categorizing context

- Morphological
- Syntactic
- Semantic

Morphological context

- Inflectional morphology

Verb: destroy, destroying, destroyed

Noun: destruction, destructions

He watered the plant.

- Derivational morphology

Noun: destructiontion

Syntactic context

- Verb: The bomb destroyed the building.
He decided to water the plant.
- Noun: The destruction of building

Semantic context

- Verb: action, activity
- Noun: state, object, etc.

Unigram tagger

```
>>> brown_a =  
    nltk.corpus.brown.tagged_sents(categories='a'  
    ' )  
  
>>> unigram_tagger =  
    nltk.UnigramTagger(brown_a)  
  
>>> sent = "Thieves leave young athletes in  
    the dard".split()  
  
>>> unigram_tagger(sent)
```

What is the difference between the unigram tagger and the lookup tagger?

Affix tagger: taking advantage of morphological context

```
>>> affix_tagger = nltk.AffixTagger(brown_a,  
    affix_length=-2,min_stem_length=3)  
>>> affix_tagger.tag(sent)
```

What is the difference between the affix tagger and the regular expression tagger?

Ngram taggers: taking advantage for syntactic context

- Lookup tagger is a unigram tagger
- Need to take more context into account
- Bigram, trigram taggers

w0	w1	w2	w3	w4	w5	...	wn
t0	t1	t2	t3	t4	t5	...	tn

Bigram tagger

```
>>> bigram_tagger = nltk.BigramTagger(brown_a,  
    cutoff=0)  
>>> bigram_tagger.tag(sent)
```

The more context, the better?

Data sparseness problem

- The larger the context, the more specific, the less likely to find the same context for the training and test data, e.g., "order", "executive order", "an executive order"
- Finding the right tradeoff is important

Combining taggers

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

Which tagger works best?

HW5

I) Extract collocations out of a corpus.

Assume that collocations are simply pairs of adjacent words occurring in a text.

Make sure that these pairs of words are lemmatized. List the collocations you extract from a corpus by frequency.

Output both the collocations and their frequencies.

HW5

II) Compile a POS-tagged dictionary out of Section 'a' of the Brown corpus. You just use the Brown Corpus provided in the NLTK package. For each word, list the POS tags for that word, and put the word and its POS tags on the same line, e.g., "word tag1 tag2 tag3 ... tagn". Sort the list of words alphabetically. Which words are the most ambiguous (having the most POS tags)?

HW5

III). Use the combined tagger described in the NLTK book to tag the article used for Homework 4. For this homework, you just need to write a simple python program calling the functions provided in the NLTK package. You don't have to reinvent the wheel and reimplement the tagger(s) yourself.

Next week: functions and classes

- I'll be out of town for a conference
- Jinho Choi will be teaching Python functions and classes in my place
- Jinho is a very smooth programmer