Categorizing and Tagging Words

CHAPTER 5

Syed Kamran Ali Ahmad

Goal:

- What are lexical categories, and how are they used in natural language processing
- What is a good Python data structure for storing words and their categories?
- How can we automatically tag each word of a text with its word class?
- Cover fundamental techniques in NLP, including sequence labeling, n-gram models, backoff, and evaluation

Tagging:

- The process of classifying words into their parts-of-speech and labeling them accordingly is known as part-of-speech tagging, POS tagging, or simply tagging.
- Parts-of-speech are also known as **word classes** or **lexical categories**.
- The collection of tags used for a particular task is known as a tagset.

Using a Tagger

processes a sequence of words, and attaches a part of speech tag to each word

```
>>> import nltk
>>> text = nltk.word_tokenize("And now for
    something completely different")
>>> nltk.pos_tag(text)
[('And', 'CC'), ('now', 'RB'), ('for', 'IN'),
    ('something', 'NN'),
('completely', 'RB'), ('different', 'JJ')]
```

we need to know which word is being used in order to pronounce the text correctly

```
>>> text = nltk.word_tokenize("They refuse to
    permit us to obtain the refuse permit")
>>> nltk.pos_tag(text)
[('They', 'PRP'), ('refuse', 'VBP'), ('to', 'TO'), ('permit',
    'VB'), ('us', 'PRP'), ('to', 'TO'), ('obtain', 'VB'), ('the',
    'DT'), ('refuse', 'NN'), ('permit', 'NN')]
```

- refUSE is a verb meaning “deny,” while REFuse is a noun meaning “trash”
- text-to-speech systems usually perform tagging

text.similar() method

- The takes a word w, finds all contexts w1w w2, then finds all words w' that appear in the same context, i.e. w1w'w2.

```
>>> text = nltk.Text(word.lower() for word in nltk.corpus.brown.words())
>>> text.similar('woman')
Building word-context index...
man time day year car moment world family house country child boy state job way war girl place room word
```

- categories arise from superficial analysis of the distribution of words in text

Tagged Corpora(Representing Tagged Tokens)

- By convention in NLTK, a tagged token is represented using a tuple consisting of the token and the tag
- We can create one of these special tuples from the standard string representation of a tagged token, using the function str2tuple()
- >>> tagged_token = nltk.tag.str2tuple('fly/NN')

Tagged Corpora(Reading Tagged Corpora)

- Brown Corpus with a text editor:

The/at Fulton/np-tl County/nn-tl ….

- NLTK's corpus readers provide a uniform interface so that you don't have to be concerned with the different file form

```
>>> nltk.corpus.brown.tagged_words()
[('The', 'AT'), ('Fulton', 'NP-TL'), ('County', 'NN-TL'), ...]
```

Note: (Reading Tagged Corpora)

- Whenever a corpus contains tagged text, the NLTK corpus interface will have a tagged_words() method.
- part-of-speech tags have been converted to uppercase (standard practice)
- Not all corpora employ the same set of tags
- Tagged corpora for several other languages are distributed with NLTK, including Chinese, Hindi... These usually contain non-ASCII text, and Python always displays this in hexadecimal when printing

A Simplified Part-of-Speech Tagset

- Tagged corpora use many different conventions for tagging words.

Tag	Meaning	Examples
ADJ	adjective	new, good, high,
ADV	adverb	really, already, still,
CNJ	conjunction	and, or, but, if, while,
DET	determiner	the, a, some, most,

Nouns (a word class in simplified Tags)

- generally refer to people, places, things, or concepts, e.g., woman, Scotland.
- can appear after determiners and adjectives, and can be the subject or object of the verb.

Verbs (a word class in simplified Tags)

- are words that describe events and actions, e.g., fall and eat,
- In the context of a sentence, verbs typically express a relation involving the referents of one or more noun phrases.

Adjectives (word classes in simplified Tags)

- Adjectives describe nouns, and can be used as modifiers (e.g., large in the large pizza), or as predicates (e.g., the pizza is large).
- English adjectives can have internal structure (e.g., fall+ing in the falling stocks).

Adverbs (word classes in simplified Tags)

- Adverbs modify verbs to specify the time, manner, place, or direction of the event described by the verb (e.g., quickly in the stocks fell quickly).
- Adverbs may also modify adjectives (e.g., really in Mary's teacher was really nice).

Other closed class words

- English has several categories of closed class words in addition to **prepositions**, such as **articles** (also called determiners) (e.g., the, a), **modals** (e.g., should, may), and personal **pronouns** (e.g., she, they).
- Each dictionary and grammar classifies these words differently.

Unsimplified Tags:

Program to find the most frequent nouns

You will see that there are many variants of NN;

```
def findtags(tag_prefix, tagged_text):
    cfd = nltk.ConditionalFreqDist((tag, word) for (word, tag) in tagged_text
                                   if tag.startswith(tag_prefix))
    return dict((tag, cfd[tag].keys()[:5]) for tag in cfd.conditions())
>>> tagdict = findtags('NN',
      nltk.corpus.brown.tagged_words(categories='news'))
>>> for tag in sorted(tagdict):
...     print tag, tagdict[tag]
...
NN ['year', 'time', 'state', 'week', 'man']
NN$ ["year's", "world's", "state's", "nation's", "company's"]
NN$-HL ["Golf's", "Navy's"]
NN$-TL ["President's", "University's", "League's", "Gallery's", "Army's"]
NN-HL ['cut', 'Salary', 'condition', 'Question', 'business']
```

Exploring Tagged Corpora

```
>>> brown_learned_text =
    brown.words(categories='learned')
>>> sorted(set(b for (a, b) in
    nltk.ibigrams(brown_learned_text) if a ==
    'often'))
[',', '.', 'accomplished', 'analytically', 'appear',
    'apt', 'associated', 'assuming', ...]
```

Mapping Words to Properties

- **Using Python Dictionaries data type**

```
>>> pos = {}
 >>> pos {'colorless': 'ADJ'}
>>> pos['ideas'] = 'N'
>>> pos['sleep'] = 'V'
>>> pos['furiously'] = 'ADV'
>>> pos
{'furiously': 'ADV', 'ideas': 'N', 'colorless': 'ADJ', 'sleep': 'V'}
```

- **the dictionary methods keys(), values(), and items()**

```
>>> pos.keys()
['colorless', 'furiously', 'sleep', 'ideas']
>>> pos.values()
['ADJ', 'ADV', 'V', 'N']
>>> pos.items()
[('colorless', 'ADJ'), ('furiously', 'ADV'), ('sleep', 'V'), ('ideas', 'N')]
```

Automatic Tagging

- **tag** of a word depends on the **word** and its **context** within a sentence
- Simply loading the data

```
>>> from nltk.corpus import brown
>>> brown_tagged_sents =
    brown.tagged_sents(categories='news')
>>> brown_sents =
    brown.sents(categories='news‘)
```

The Default Tagger: assigns the same tag to each token.

- **Let's find out which tag is most likely**

```
>>> tags = [tag for (word, tag) in
    brown.tagged_words(categories='news')]
>>> nltk.FreqDist(tags).max()
'NN‘
```

- **Now we can create a tagger that tags everything as NN.**

```
>>> raw = 'I do not like green eggs and ham, I do not like them Sam I
    am!'
>>> tokens = nltk.word_tokenize(raw)
>>> default_tagger = nltk.DefaultTagger('NN')
>>> default_tagger.tag(tokens)
[('I', 'NN'), ('do', 'NN'), ('not', 'NN'), ('like', 'NN'), ('green', 'NN'),
('eggs', 'NN'), ('and', 'NN'), ('ham', 'NN'), (',', 'NN'), ('I', 'NN'),
```

The Regular Expression Tagger

assigns tags to tokens on the basis of matching patterns

- our guess that any word ending in **ed** is the **past participle of a verb**. We can express all our guesses as a list of regular expressions:

```
>>> patterns = [
...     (r'.*ing$', 'VBG'),               # gerunds
...     (r'.*ed$', 'VBD'),                # simple past
...     (r'.*es$', 'VBZ'),               # 3rd singular present
...     (r'.*ould$', 'MD'),                # modals
...     (r'.*\'s$', 'NN$'),               # possessive nouns
...     (r'.*s$', 'NNS'),                # plural nouns
...     (r'^-?[0-9]+(.[0-9]+)?$', 'CD'),  # cardinal numbers
...     (r'.*', 'NN')                   # nouns (default)
... ]
```

- Note that these **are processed in order**, and final regular expression «.*» is a catch-all that tags everything as a noun. (equivalent to less efficient version of the default tagger)

- Now we set up a tagger and use it to tag a sentence.

```
>>> regexp_tagger = nltk.RegexpTagger(patterns)
>>> regexp_tagger.tag(brown_sents[3])
[('``', 'NN'), ('Only', 'NN'), ('a', 'NN'), ('relative', 'NN'),
   ('handful', 'NN'),
('of', 'NN'), ('such', 'NN'), ('reports', 'NNS'), ('was',
   'NNS'), ('received', 'VBD'),
("''", 'NN'), (',', 'NN'), ('the', 'NN'), ('jury', 'NN'),
   ('said', 'NN'), (',', 'NN'),
('``', 'NN'), ('considering', 'VBG'), ('the', 'NN'),
   ('widespread', 'NN'), ...]
>>> regexp_tagger.evaluate(brown_tagged_sents)
0.2032639178948624 5
```

The Lookup Tagger

- Let's find 100 **most frequent words** and their most **likely tag.** **We can then use this information as the model for a "lookup tagger" (an NLTK UnigramTagger):**

```
>>> fd = nltk.FreqDist(brown.words(categories='news'))
>>> cfd = nltk.ConditionalFreqDist(brown.tagged_words(categories='news'))
>>> most_freq_words = fd.keys()[:100]
>>> likely_tags = dict((word, cfd[word].max()) for word in most_freq_words)
>>> baseline_tagger = nltk.UnigramTagger(model=likely_tags)
>>> baseline_tagger.evaluate(brown_tagged_sents)
0.4557849513694134
```

- **simply knowing the tags for the 100 most frequent words enables us to tag a large fraction of tokens correctly**

- Let’s see what it does on some untagged input text:

```
>>> sent = brown.sents(categories='news')[3]
>>> baseline_tagger.tag(sent)
[('``', '``'), ('Only', None), ('a', 'AT'), ('relative',
None), ('handful', None), ('of', 'IN'), ('such',
None), ('reports', None), … ]
```

- Many words have been assigned a tag of None, because they were not among the 100 most frequent words

backoff

- In these cases we would like to assign the default tag of NN. In other words, we want to use the lookup table first, and if it is unable to assign a tag, then use the default tagger, a process known as backoff
- Now the lookup tagger will only store word-tag pairs for words other than nouns, and whenever it cannot assign a tag to a word, it will invoke the default tagger.

```
>>> baseline_tagger =
    nltk.UnigramTagger(model=likely_tags,
...                                 backoff=nltk.DefaultTagger('NN'))
```

Lookup tagger with varying model size.

```
def performance(cfd, wordlist):
    lt = dict((word, cfd[word].max()) for word in wordlist)
    baseline_tagger = nltk.UnigramTagger(model=lt, backoff=nltk.DefaultTagger('NN'))
    return baseline_tagger.evaluate(brown.tagged_sents(categories='news'))
def display():
    import pylab
    words_by_freq = list(nltk.FreqDist(brown.words(categories='news')))
    cfd = nltk.ConditionalFreqDist(brown.tagged_words(categories='news'))
    sizes = 2 ** pylab.arange(15)
    perfs = [performance(cfd, words_by_freq[:size]) for size in sizes]
    pylab.plot(sizes, perfs, '-bo')
    pylab.title('Lookup Tagger Performance with Varying Model Size')
    pylab.xlabel('Model Size')
    pylab.ylabel('Performance')
    pylab.show()
>>> display()
```

Evaluation

- We evaluate the performance of a tagger relative to tags a human expert would assign

- **gold standard test data.**

This is a corpus which has been manually annotated and accepted as a standard against which the guesses of an automatic system are assessed.

The tagger is regarded as being correct if the tag it guesses for a given word is the same as the gold standard tag.

N-Gram Tagging

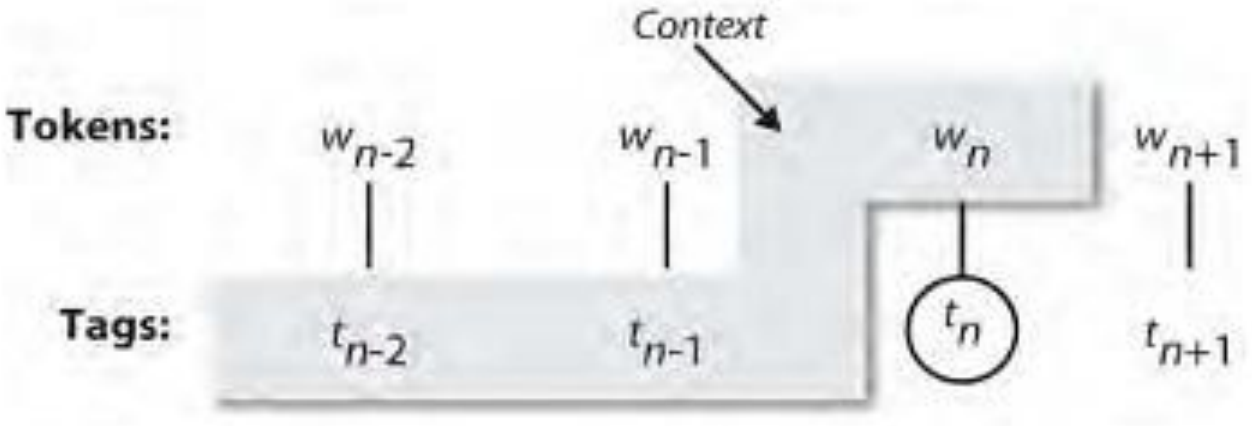
- assign the tag that is most likely for that particular token (like a lookup tagger, except technique for setting it up, called training. In the following code sample, we train a unigram tagger, use it to tag a sentence, and then evaluate:

```
>>> from nltk.corpus import brown
>>> brown_tagged_sents = brown.tagged_sents(categories='news')
>>> brown_sents = brown.sents(categories='news')
>>> unigram_tagger = nltk.UnigramTagger(brown_tagged_sents)
>>> unigram_tagger.tag(brown_sents[2007])
[('Various', 'JJ'), ('of', 'IN'), ('the', 'AT'), ('apartments', 'NNS'),
('are', 'BER'), ('of', 'IN'), ('the', 'AT'), ('terrace', 'NN'), ('type', 'NN'),
(',', ','), ('being', 'BEG'), ('on', 'IN'), ('the', 'AT'), ('ground', 'NN'),
('floor', 'NN'), ('so', 'QL'), ('that', 'CS'), ('entrance', 'NN'), ('is', 'BEZ'),
('direct', 'JJ'), ('.', '.')]
>>> unigram_tagger.evaluate(brown_tagged_sents)
0.9349006503968017
```

Separating Training and Testing Data

```
>>> size = int(len(brown_tagged_sents) * 0.9)
>>> size
4160
>>> train_sents = brown_tagged_sents[:size]
>>> test_sents = brown_tagged_sents[size:]
>>> unigram_tagger =
    nltk.UnigramTagger(train_sents)
>>> unigram_tagger.evaluate(test_sents)
0.8120203329014252 8
```

General N-Gram Tagging



An n-gram tagger is a generalization of a unigram tagger whose context is the current word together with the part-of-speech tags of the n-1 preceding tokens

```
>> bigram_tagger =
   nltk.BigramTagger(train_sents)
>>> bigram_tagger.tag(brown_sents[2007])
[('Various', 'JJ'), ('of', 'IN'), ('the', 'AT'),
   ('apartments', 'NNS'), … ]
>>> unseen_sent = brown_sents[4203]
>>> bigram_tagger.tag(unseen_sent)
[('The', 'AT'), ('population', 'NN'), ('of', 'IN'),
   ('the', 'AT'), ('Congo', 'NP'), …]
```

- Note: it does badly on an **unseen** sentence.

Combining Taggers

- Most NLTK taggers permit a **backoff** tagger . The backoff tagger may itself have a backoff tagger:

```
>>> t0 = nltk.DefaultTagger('NN')
>>> t1 = nltk.UnigramTagger(train_sents,
    backoff=t0)
>>> t2 = nltk.BigramTagger(train_sents, backoff=t1)
>>> t2.evaluate(test_sents)
0.8449117910894049 5
```

Tagging Unknown Words

- A useful method to tag unknown words based on context is to limit the vocabulary of a tagger to the most frequent n words, and to replace every other word with a special word UNK

Storing Taggers

```
>>> from cPickle import dump
>>> output = open('t2.pkl', 'wb')
>>> dump(t2, output, -1)
>>> output.close()

>>> from cPickle import load
>>> input = open('t2.pkl', 'rb')
>>> tagger = load(input)
>>> input.close()
```

Transformation-Based Tagging

- Brill tagging is a kind of transformation-based learning. The general idea is very simple:
  guess the tag of each word, then go back and fix the mistakes.
- Analogies to **Painter**

```
>>> nltk.tag.brill.demo()
Training Brill tagger on 80 sentences...
Finding initial useful rules...
    Found 6555 useful rules.
……
>>> print(open("errors.out").read())
```

Phrase	to	increase	grants	to	states	for	vocational	rehabilitation
Unigram	TO	*NN*	NNS	*TO*	NNS	IN	JJ	NN
Rule 1		*VB*						
Rule 2				*IN*				
Output	TO	VB	NNS	IN	NNS	IN	JJ	NN
Gold	TO	VB	NNS	IN	NNS	IN	JJ	NN

- In table above, steps in Brill tagging, we see two rules.
  All such **rules** are generated from a **template** of the following form: "**replace T1 with T2 in the context C**."
  During its training phase, the tagger guesses values for T1, T2, and C, to create thousands of candidate rules. Each rule is scored according to its net benefit: the number of incorrect tags that it corrects, less the number of correct tags it incorrectly modifies**.**
  Brill taggers have another interesting property: the rules are linguistically interpretable.

How to Determine the Category of a Word?

Morphological Clues

e.g., happy → happiness, ill → illness.

Syntactic Clues

- typical contexts in which a word can occur.
- an adjective in English is that it can occur immediately before a noun, or immediately following the words be or very.

Semantic Clues

- the meaning of a word is a useful clue as to its lexical category. For example, the best-known definition of a noun is semantic: “the name of a person, place, or thing.”

New Words(Open and Close Class)

- All languages acquire new lexical items. A list of words recently added to the Oxford Dictionary of English includes cyberslacker, fatoush, blamestorm, SARS, cantopop,
- Notice that all these new words are nouns, and this is reflected in calling nouns an **open class**. By contrast, prepositions are regarded as a **closed class**. That is, there is a limited set of words belonging to the class

OpenNLP (enhance the tm package)

```
>library("openNLP")
> sentence<-"This is a short sentence consisting of
+   some nouns,verbs, and adjectives."
>tagPOS(sentence,language="en")
[1]"This/DT  is/VBZ a/DT short/JJ sentence/NN
   consisting/VBG of/IN"
[2]"some/DT nouns,/JJ verbs,/NNS and/CC
   adjectives./VBG"
```