# Chapter 5 - Categorizing and Tagging Words

Jianzhang Zhang

Alibaba Business School
Hangzhou Normal University

May 25, 2022

# Table of Contents

# Table of Contents

## Using a Tagger

**Part-of-speech tagging (POS-tagging)**: the process of classifying words into their parts of speech and labeling them accordingly.

Parts of speech are also known as word classes or lexical categories.

POS tagging is the second step in the typical NLP pipeline, following tokenization.

A POS-tagger processes a sequence of words, and attaches a part of speech tag to each word.

```
import nltk
from nltk import word_tokenize

# ['And', 'now', 'for', 'something', 'completely', 'different']
text = word_tokenize("And now for something completely
↪  different")
# [('And', 'CC'), ('now', 'RB'), ... , ('different', 'JJ')]
print(nltk.pos_tag(text))
```

- *and* is CC, a coordinating conjunction;
- *now* and *completely* are RB, or adverbs;
- *for* is IN, a preposition;
- *something* is NN, a noun;
- *and* different is JJ, an adjective.

```
import jieba
import jieba.posseg as pseg

# jieba的词性标签含义见:
↪  https://github.com/fxsjy/jieba中的词性标注部分
for word, pos in pseg.cut('坚持把解决好“三农”
↪  问题作为全党工作重中之重 举全党全社会之力推动乡村振兴'):
print(word, pos)
```

Let's look at another example, this time including some homonyms (同形异义，如 refuse, permit).

```
# 他们拒绝让我们获得垃圾许可证
text = word_tokenize("They refuse to permit us to obtain the
↪  refuse permit")

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

*refUSE* is a verb meaning "deny," while *REFuse* is a noun meaning "trash"(名词重音在前，动词重音在后). For this reason, text-to-speech systems usually perform POS-tagging.

# Table of Contents

## Representing Tagged Tokens

By convention in NLTK, a tagged token is represented using a tuple consisting of the token and the tag.

```
tagged_token = nltk.tag.str2tuple('fly/NN')
# ('fly', 'NN')
print(tagged_token)

sent = '''
The/AT grand/JJ jury/NN commented/VBD on/IN a/AT number/NN of/IN
other/AP topics/NNS ,/, AMONG/IN them/PPO the/AT Atlanta/NP
↪  and/CC
Fulton/NP-tl County/NN-tl purchasing/VBG departments/NNS ./.
'''
[nltk.tag.str2tuple(t) for t in sent.split()]
```

## Reading Tagged Corpora

Several of the corpora included with NLTK have been tagged for their part-of-speech.

```
# [('The', 'AT'), ('Fulton', 'NP-TL'), ...]
nltk.corpus.brown.tagged_words()
# [('The', 'DET'), ('Fulton', 'NOUN'), ...]
nltk.corpus.brown.tagged_words(tagset='universal')

nltk.corpus.nps_chat.tagged_words()
nltk.corpus.conll2000.tagged_words()
nltk.corpus.treebank.tagged_words()
```

# A Universal Part-of-Speech Tagset

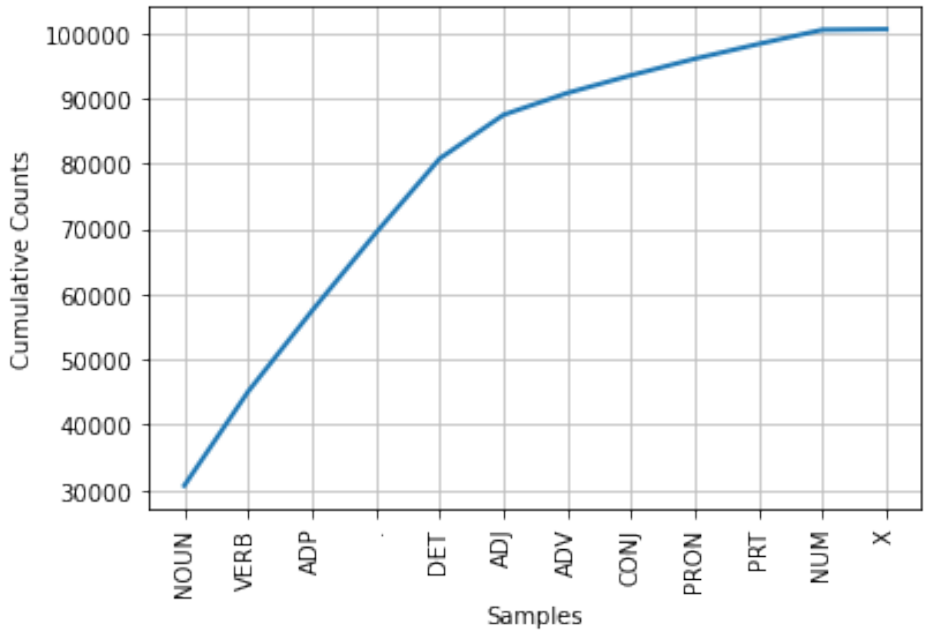
Tagged corpora use many different conventions for tagging words. To help us get started, we will be looking at a simplified tagset as below:

| Tag | Meaning | English Examples |
|---|---|---|
| ADJ | adjective | *new, good, high, special, big, local* |
| ADP | adposition | *on, of, at, with, by, into, under* |
| ADV | adverb | *really, already, still, early, now* |
| CONJ | conjunction | *and, or, but, if, while, although* |
| DET | determiner, article | *the, a, some, most, every, no, which* |
| NOUN | noun | *year, home, costs, time, Africa* |
| NUM | numeral | *twenty-four, fourth, 1991, 14:24* |
| PRT | particle | *at, on, out, over per, that, up, with* |
| PRON | pronoun | *he, their, her, its, my, I, us* |
| VERB | verb | *is, say, told, given, playing, would* |
| . | punctuation marks | *. , ; !* |
| X | other | *ersatz, esprit, dunno, gr8, univeristy* |

Figure 1: Universal Part-of-Speech Tagset

Let's see which of these tags are the most common in the news category of the Brown corpus:

```
from nltk.corpus import brown
brown_news_tagged = brown.tagged_words(categories='news',
↪  tagset='universal')
tag_fd = nltk.FreqDist(tag for (word, tag) in brown_news_tagged)
tag_fd.most_common()
tag_fd.plot(cumulative=True) # 80%的词使用前五个词性标签标注
```



## Nouns

Nouns generally refer to people, places, things, or concepts.

The simplified noun tags are **N** for common nouns (普通名词) like *book*, and **NP** for proper nouns (专有名词) like *Scotland*.

Let's inspect some tagged text to see what parts of speech occur before a noun, with the most frequent ones first.

```
word_tag_pairs = list(nltk.bigrams(brown_news_tagged))
noun_preceders = [a[1] for (a, b) in word_tag_pairs if b[1] ==
↪  'NOUN']
fdist = nltk.FreqDist(noun_preceders)
fdist.most_common()
```

## Verbs

Verbs are words that describe **events** and **actions**.

What are the most common verbs in news text? Let's sort all the verbs by frequency:

```
# [('Pierre', 'NOUN'), ('Vinken', 'NOUN'), (',', '.'), ...]
wsj = nltk.corpus.treebank.tagged_words(tagset='universal')
word_tag_fd = nltk.FreqDist(wsj)
[wt[0] for (wt, _) in word_tag_fd.most_common() if wt[1] ==
↪  'VERB']

# find words which can be both VBD and VBN
cfd1 = nltk.ConditionalFreqDist(wsj)
[w for w in cfd1.conditions() if 'VBD' in cfd1[w] and 'VBN' in
↪  cfd1[w]]
```

## Adjectives and Adverbs

**Adjectives describe nouns**, and can be used as **modifiers** (e.g. large in the large pizza), or in **predicates** (e.g. the pizza is large).

**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).

English has several categories of closed class words in addition to **prepositions**, such as **articles** (also often called determiners 限定词) (e.g., the, a), **modals** (e.g., should, may), and **personal pronouns** (e.g., she, they).

## Exploring Tagged Corpora

Let's look at some larger context, and find words involving **particular sequences of tags and words**.

```
# pattern is <Verb> to <Verb>
def process(sentence):
  for (w1,t1), (w2,t2), (w3,t3) in nltk.trigrams(sentence):
    if (t1.startswith('V') and t2 == 'TO' and
    ↪  t3.startswith('V')):
      print(w1, w2, w3)

from nltk.corpus import brown

for tagged_sent in brown.tagged_sents():
# seek to set
# like to see
process(tagged_sent)
```

# Table of Contents

Most often, we are mapping from a "word" to some structured object.

For example, a document index maps from **a word** (which we can represent as a string), to **a list of pages** (represented as a list of integers).

- Dictionaries in Python
- Defining Dictionaries
- Default Dictionaries
- Incrementally Updating a Dictionary
- Complex Keys and Values
- Inverting a Dictionary

For the above topics, please refer to the content of Chapter 4 in my programming basics course.

# Table of Contents

## The Default Tagger

We can create a tagger that tags everything as the same POS tag (e.g. NN). It establishes an important baseline for tagger performance.

```
tags = [tag for (word, tag) in
↪  brown.tagged_words(categories='news')]
# 标注数据集中最常见的标签是NN，因此将default tagger设定为NN
nltk.FreqDist(tags).max() # NN
default_tagger = nltk.DefaultTagger('NN')

# 0.13
default_tagger.evaluate(brown_tagged_sents)
```

## The Regular Expression Tagger

The regular expression tagger assigns tags to tokens on the basis of matching patterns.

```
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)
]

regexp_tagger = nltk.RegexpTagger(patterns)
# 0.20，相比全部标注为名词，准确率上升了7%
regexp_tagger.evaluate(brown_tagged_sents)
```

## The Lookup Tagger

Let's find the hundred most frequent words and store their most likely tag.

```
fd = nltk.FreqDist(brown.words(categories='news'))
cfd = nltk.ConditionalFreqDist(brown.tagged_words(categories='ne
↪  ws'))
# 最高频的100个词
most_freq_words = fd.most_common(100)
# 为每个高频词找出其最高频的词性标记
likely_tags = dict((word, cfd[word].max()) for (word, _) in
↪  most_freq_words)
baseline_tagger = nltk.UnigramTagger(model=likely_tags)
# 此时，我们已经把baseline的准确率提高到了45%,
↪  比正则表达式分词器提高了25%
baseline_tagger.evaluate(brown_tagged_sents) # 0.45
```

```
sent = brown.sents(categories='news')[3]
# [('``', '``'), ('Only', None), ('a', 'AT'), ..., ]
print(baseline_tagger.tag(sent))
```

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

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 (退避).

```
# 评估lookup tagger + default tagger的准确率
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'))

baseline_tagger = nltk.UnigramTagger(model=likely_tags,backoff=n
↪  ltk.DefaultTagger('NN'))
# 组合的tagger把baseline的准确率提升为58%，比lookup
↪  tagger提高了13%。
performance(cfd, words_by_freq[:100]) # 0.58
```

# Table of Contents

## Unigram Tagging

A unigram Tagger assign the tag that is most likely for that particular token. It behaves just like a lookup tagger.

We train a UnigramTagger by specifying tagged sentence data as a parameter when we initialize the tagger.

The training process involves inspecting the tag of each word and storing the most likely tag for any word in a dictionary.

```
brown_tagged_sents = brown.tagged_sents(categories='news')
unigram_tagger = nltk.UnigramTagger(brown_tagged_sents)
```

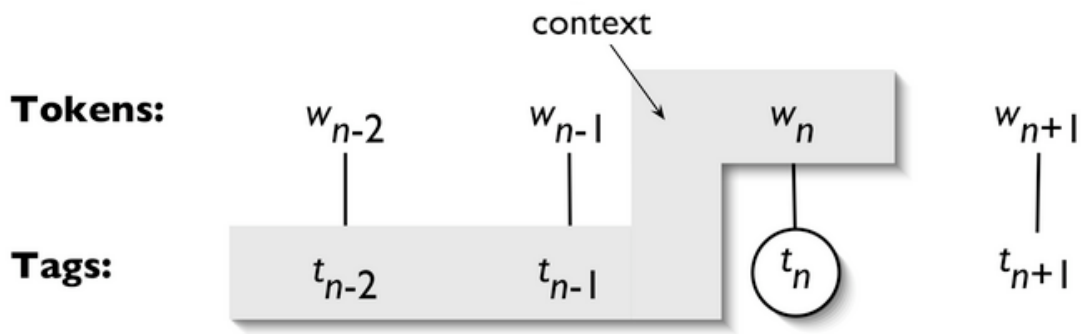
## Separating the Training and Testing Data

With the data division, we will have a better picture of the usefulness of this tagger, i.e. its performance on previously unseen text.

```
# 9:1
size = int(len(brown_tagged_sents) * 0.9)
train_sents = brown_tagged_sents[:size]
test_sents = brown_tagged_sents[size:]
unigram_tagger = nltk.UnigramTagger(train_sents)
# 0.81
unigram_tagger.evaluate(test_sents)
```

## 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, as shown in below:




An n-gram tagger picks the tag that is most likely in the given context.

```
bigram_tagger = nltk.BigramTagger(train_sents)
# 在训练数据中的句子上表现很好，因为在训练过程中见过该句子
print(bigram_tagger.tag(brown_sents[2007]))
```

```
unseen_sent = brown_sents[4203]
print(bigram_tagger.tag(unseen_sent))
# [('The', 'AT'), ('population', 'NN'), ('of', 'IN'), ('the',
↪  'AT'), ('Congo', 'NP'), ('is', 'BEZ'), ('13.5', None),
↪  ('million', None), (',', None), ('divided', None), ('into',
↪  None), ('at', None), ('least', None), ('seven', None),
↪  ('major', None), ('``', None), ('culture', None),
↪  ('clusters', None), ("''", None), ('and', None),
↪  ('innumerable', None), ('tribes', None), ('speaking', None),
↪  ('400', None), ('separate', None), ('dialects', None), ('.',
↪  None)]
bigram_tagger.evaluate(test_sents) # 0.10
```

当遇到训练数据中未见过的单词时，只能标记为 None，None 之后的词，也无法正确标注，因为其在训练数据中没见过 None 标签。

随着 n 变大，上下文的特异性增加，测试数据中包含的未知上下文也会增多，造成数据稀疏问题。

## Combining Taggers

We could combine the results of a bigram tagger, a unigram tagger, and a default tagger, as follows:

1. Try tagging the token with the bigram tagger;
2. If the bigram tagger is unable to find a tag for the token, try the unigram tagger;
3. If the unigram tagger is also unable to find a tag, use a default tagger.

```
t0 = nltk.DefaultTagger('NN')
t1 = nltk.UnigramTagger(train_sents, backoff=t0)
t2 = nltk.BigramTagger(train_sents, backoff=t1)
# 比仅使用unigram tagger准确性更好，提高了3%
t2.evaluate(test_sents) # 0.84
```

## Tagging Unknown Words

A useful method to tag unknown words (**out-of-vocabulary items, OOV**) 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`.

During training, a unigram tagger will probably learn that UNK is usually a noun.

However, the n-gram taggers will detect contexts in which it has some other tag.

For example, if the preceding word is *to* (tagged **TO**), then **UNK** will probably be tagged as a verb.

**Quiz**: please modify the above combining tagger based on the above idea, and evaluate its performance.

## Storing Taggers

```
# dump to file
from pickle import dump

output = open('./t2.pkl', 'wb')
dump(t2, output, -1)
output.close()

# load model from file
from pickle import load
input = open('t2.pkl', 'rb')
tagger = load(input)
input.close()
text = """The board's action shows what free enterprise is up
↪  against in our complex maze of regulatory laws ."""
tokens = text.split()
print(tagger.tag(tokens))
```

## Performance Limitations

A way to investigate the performance of a tagger is to study its mistakes (错误分析). Some tags may be harder than others to assign, and it might be possible to treat them specially by **pre- or post-processing the data**. (例如，介词比动词和名词更容易被正确标注出来)

Another convenient way to look at tagging errors is the **confusion matrix**. It charts expected tags (the gold standard) against actual tags generated by a tagger:

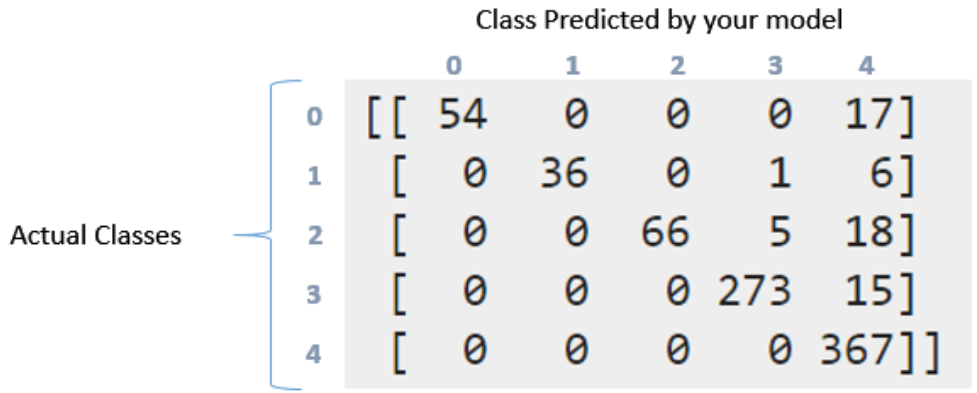



Figure 2: Confusion Matrix Example

# Table of Contents

The shortcomings of N-gram tagger are mainly: too large model size and sparse context.

Brill tagging is a kind of transformation-based learning: **guess** the tag of each word, then **go back and fix** the mistakes.

The rules used in Brill tagger are learned from annotated training data. The rules are linguistically interpretable.

We will examine the operation of two rules: (a) Replace NN with VB when the previous word is TO; (b) Replace TO with IN when the next tag is NNS.

| **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 |

Figure 3: Steps in Brill Tagging

# Table of Contents

How do we decide what category a word belongs to in the first place? (人类是如何确定一个词语的词性类别呢？)

In general, linguists use **morphological**, **syntactic**, and **semantic** clues to determine the category of a word.

- 形容词通常出现在名词之前或 be，very 之后;
- Dutch word *verjaardag* means the same as the English word *birthday* -> Noun;
- Nouns are called an open class while prepositions are regarded as a closed class;

# THE END