Lab 5: Part of Speech Tagging

**This session covers:**

* Different types of POS Taggers 🡪 NLTK POS Tagger | Regex POS Tagger | TextBlob POS Tagger
* Universal Part-of-Speech Tagset
* Lemmatization with & without POS tags

**Learning Outcome:**

Apply text mining and natural language processing methodologies to textual data.

Quick Review

The process of classifying words into their **parts of speech** and labelling 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**.

Practices

* 1. NLTK POS Tagger

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

import nltk

from nltk.tokenize import sent\_tokenize, word\_tokenize

text1 = word\_tokenize("And now for something completely different")

print(nltk.pos\_tag(text1))

text2 = word\_tokenize("They refuse to permit us to obtain the refuse permit")

print(nltk.pos\_tag(text2))

#to get the meaning of the tags

nltk.help.upenn\_tagset('JJ')

The text.similar() method takes a word *w*, finds all contexts *w1 ww2*, then finds all words *w'* that appear in the same context, i.e. *w1 w'w2*.

text = nltk.Text(word.lower() for word in nltk.corpus.brown.words())

word\_list = ['woman', 'bought', 'over', 'the']

for w in word\_list:

print("\nwords in text similar to '"+ w + "' are: ")

text.similar(w)

* 1. 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')

print(tagged\_token)

print(tagged\_token[0])

print(tagged\_token[1])

We can construct a list of tagged tokens directly from a string. The first step is to tokenize the string to access the individual word/tag strings, and then to convert each of these into a tuple (using str2tuple()).

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 which/WDT it/PPS

said/VBD ``/`` ARE/BER well/QL operated/VBN and/CC follow/VB generally/RB

accepted/VBN practices/NNS which/WDT inure/VB to/IN the/AT best/JJT

interest/NN of/IN both/ABX governments/NNS ''/'' ./.

'''

[ ]

* 1. Reading Tagged Corpora

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

nltk.corpus.brown.tagged\_words()

nltk.corpus.brown.tagged\_words(tagset='universal')

Whenever a corpus contains tagged text, the NLTK corpus interface will have a tagged\_words() method. Here are some more examples, again using the output format illustrated for the Brown Corpus:

print(nltk.corpus.nps\_chat.tagged\_words())

print(nltk.corpus.conll2000.tagged\_words())

print(nltk.corpus.treebank.tagged\_words())

Not all corpora employ the same set of tags. Initially we want to avoid the complications of these tagsets, so we use a built-in mapping to the "Universal Tagset":

print(nltk.corpus.brown.tagged\_words(tagset='universal'))

print(nltk.corpus.treebank.tagged\_words(tagset='universal'))

Tagged corpora for several other languages are distributed with NLTK, including Chinese, Hindi, Portuguese, Spanish, Dutch and Catalan. These usually contain non-ASCII text, and Python always displays this in hexadecimal when printing a larger structure such as a list.

print(nltk.corpus.sinica\_treebank.tagged\_words())

print(nltk.corpus.indian.tagged\_words())

print(nltk.corpus.mac\_morpho.tagged\_words())

print(nltk.corpus.conll2002.tagged\_words())

print(nltk.corpus.cess\_cat.tagged\_words())

* 1. Universal Parts-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.

|  |  |  |
| --- | --- | --- |
| **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* |

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()

* 1. Nouns

Nouns generally refer to people, places, things, or concepts, e.g.: woman, Scotland, book, intelligence. Nouns can appear after determiners and adjectives, and can be the subject or object of the verb, as shown in 2.2.

|  |  |  |
| --- | --- | --- |
| **Word** | **After a determiner** | **Subject of the verb** |
| woman | *the* woman who I saw yesterday ... | the woman *sat* down |
| Scotland | *the* Scotland I remember as a child ... | Scotland *has* five million people |
| book | *the* book I bought yesterday ... | this book *recounts* the colonization of Australia |
| intelligence | *the* intelligence displayed by the child ... | Mary's intelligence *impressed* her teachers |

The simplified noun tags are N for common nouns like book, and NP for proper nouns like Scotland.

Let's inspect some tagged text to see what parts of speech occur before a noun, with the most frequent ones first. To begin with, we construct a list of bigrams whose members are themselves word-tag pairs such as (('The', 'DET'), ('Fulton', 'NP')) and (('Fulton', 'NP'), ('County', 'N')). Then we construct a FreqDist from the tag parts of the bigrams.

word\_tag\_pairs = 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)

[tag for (tag, \_) in fdist.most\_common()]

This confirms our assertion that nouns occur after determiners and adjectives, including numeral adjectives (tagged as NUM).

* 1. Verbs

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

|  |  |  |
| --- | --- | --- |
| **Word** | **Simple** | **With modifiers and adjuncts (italicized)** |
| fall | Rome fell | Dot com stocks *suddenly* fell *like a stone* |
| eat | Mice eat cheese | John ate the pizza *with gusto* |

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

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']

Note that the items being counted in the frequency distribution are word-tag pairs. Since words and tags are paired, we can treat the word as a condition and the tag as an event, and initialize a conditional frequency distribution with a list of condition-event pairs. This lets us see a frequency-ordered list of tags given a word:

cfd1 = nltk.ConditionalFreqDist(wsj)

cfd1['yield'].most\_common()

cfd1['cut'].most\_common()

We can reverse the order of the pairs, so that the tags are the conditions, and the words are the events. Now we can see likely words for a given tag. We will do this for the WSJ tagset rather than the universal tagset:

wsj = nltk.corpus.treebank.tagged\_words()

cfd2 = nltk.ConditionalFreqDist((tag, word) for (word, tag) in wsj)

list(cfd2['VBN'])

To clarify the distinction between VBD (past tense) and VBN (past participle), let's find words which can be both VBD and VBN, and see some surrounding text:

[w for w in cfd1.conditions() if 'VBD' in cfd1[w] and 'VBN' in cfd1[w]]

idx1 = wsj.index(('kicked', 'VBD'))

wsj[idx1-4:idx1+1]

idx2 = wsj.index(('kicked', 'VBN'))

wsj[idx2-4:idx2+1]

* 1. Exploring Tagged Corpora

Suppose we're studying the word *often* and want to see how it is used in text. We could ask to see the words that follow *often*

brown\_learned\_text = brown.words(categories='learned')

sorted(set(b for (a, b) in nltk.bigrams(brown\_learned\_text) if a == 'often'))

However, it's probably more instructive use the tagged\_words() method to look at the part-of-speech tag of the following words:

brown\_lrnd\_tagged = brown.tagged\_words(categories='learned', tagset='universal')

tags = [b[1] for (a, b) in nltk.bigrams(brown\_lrnd\_tagged) if a[0] == 'often']

fd = nltk.FreqDist(tags)

fd.tabulate()

Notice that the most high-frequency parts of speech following *often* are verbs. Nouns never appear in this position (in this particular corpus).

Next, let's look at some larger context, and find words involving particular sequences of tags and words (in this case "<Verb> to <Verb>"). In code-three-word-phrase we consider each three-word window in the sentence, and check if they meet our criterion. If the tags match, we print the corresponding words.

import nltk

from nltk.corpus import brown

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)

Finally, let's look for words that are highly ambiguous as to their part of speech tag. Understanding why such words are tagged as they are in each context can help us clarify the distinctions between the tags.

brown\_news\_tagged = brown.tagged\_words(categories='news', tagset='universal')

data = nltk.ConditionalFreqDist((word.lower(), tag)

for (word, tag) in brown\_news\_tagged)

for word in sorted(data.conditions()):

if len(data[word]) > 3:

tags = [tag for (tag, \_) in data[word].most\_common()]

print(word, ' '.join(tags))

* 1. Automatic Tagging

We'll begin by loading the data we will be using.

from nltk.corpus import brown

brown\_tagged\_sents = brown.tagged\_sents(categories='news')

brown\_sents = brown.sents(categories='news')

print(brown\_tagged\_sents)

print()

print(brown\_sents)

* **The Default Tagger**

The simplest possible tagger assigns the same tag to each token. This may seem to be a rather banal step, but it establishes an important baseline for tagger performance. In order to get the best result, we tag each word with the most likely tag. Let's find out which tag is most likely (now using the unsimplified tagset):

tags = [tag for (word, tag) in brown.tagged\_words(categories='news')]

nltk.FreqDist(tags).max()

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

tags = [tag for (word, tag) in brown.tagged\_words(categories='news')]

dt = nltk.FreqDist(tags).max()

print(dt)

print("~~~~~~~~~ Example ~~~~~~~~~")

raw = 'I do not like green eggs and ham, I do not like them Sam I am!'

tokens = word\_tokenize(raw)

default\_tagger = nltk.DefaultTagger('NN')

default\_tagger.tag(tokens)

Unsurprisingly, this method performs rather poorly. On a typical corpus, it will tag only about an eighth of the tokens correctly, as we see below:

default\_tagger.evaluate(brown\_tagged\_sents)

* 1. **The Regular Expression Tagger**

The regular expression tagger assigns tags to tokens on the basis of matching patterns. For instance, we might guess that any word ending in *ed* is the past participle of a verb, and any word ending with *'s* is a possessive noun. We can express these 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)

(r'^\d+$', 'CD'),

(r'.\*ing$', 'VBG'), # gerunds, i.e. wondering

(r'.\*ment$', 'NN'), # i.e. wonderment

(r'.\*ful$', 'JJ') # i.e. wonderful

]

regexp\_tagger = nltk.RegexpTagger(patterns)

tagger=nltk.tag.sequential.RegexpTagger(patterns)

print(tagger.tag('He was born in March 1991'))

* 1. **The Lookup Tagger**

A lot of high-frequency words do not have the NN tag. Let's find the hundred most frequent words and store 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.most\_common(100)

likely\_tags = dict((word, cfd[word].max()) for (word, \_) in most\_freq\_words)

baseline\_tagger1 = nltk.UnigramTagger(model=likely\_tags)

baseline\_tagger1.evaluate(brown\_tagged\_sents)

It should come as no surprise by now that simply knowing the tags for the 100 most frequent words enables us to tag a large fraction of tokens correctly (nearly half in fact). Let's see what it does on some untagged input text:

sent = brown.sents(categories='news')[3]

baseline\_tagger1.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**. We do this by specifying one tagger as a parameter to the other, as shown below. 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\_tagger2 = nltk.UnigramTagger(model=likely\_tags,backoff=nltk.DefaultTagger('NN'))

baseline\_tagger2.tag(sent)

Put all this together:

import nltk

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 # install matplotlib

word\_freqs = nltk.FreqDist(brown.words(categories='news')).most\_common()

words\_by\_freq = [w for (w, \_) in word\_freqs]

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()

* 1. **Text Blob Tagger**

import nltk

from textblob import TextBlob

wiki = TextBlob("Python is a high-level, general-purpose programming language. Python is a high-level, general-purpose programming language.")

print(wiki.tags)

import nltk

from nltk.tokenize import word\_tokenize

text1 = word\_tokenize("Python is a high-level, general-purpose programming language. Python is a high-level, general-purpose programming language.")

print(nltk.pos\_tag(text1))

import nltk

from textblob import TextBlob

wiki = TextBlob("Programming skill is very important for a analytics person.")

print(wiki.tags)

print(wiki.noun\_phrases)

print()

import nltk

from nltk.tokenize import word\_tokenize

text1 = word\_tokenize("Programming skill is very important for a analytics person.")

print(nltk.pos\_tag(text1))

print()

text2 = word\_tokenize("what men?")

print(nltk.pos\_tag(text2))

print()

print(nltk.help.upenn\_tagset("VBZ"))

**References:**

1. Speech and Language Processing, by Dan Jurafsky and James H. Martin. Prentice Hall Series in Artificial Intelligence, 2008.
2. Natural Language Processing with Python, by Steven Bird, Ewan Klein and Edward Loper, 2014.