DATA SOCIETY®

The premiere data science training for professionals

"If you can't explain it simply, you don't understand it well enough."

- Albert Einstein

Which problems will you solve?

Objectives for this lecture:

1. Create your own tokens
2. Automate parts-of-speech taggers
3. Define your own dictionaries

Overview

1. Using a tagger
2. Identifying syntactic patterns
3. Using default dictionaries
4. Automating tags
5. Building and using taggers

Categorizing and tagging words

- Word classes (also known as lexical categories) are ways for us categorize words for language processing tasks

- The process of classifying words into their parts of speech (nouns, verbs, adjectives, etc) and labeling them accordingly is known as part-of-speech tagging or POS-tagging

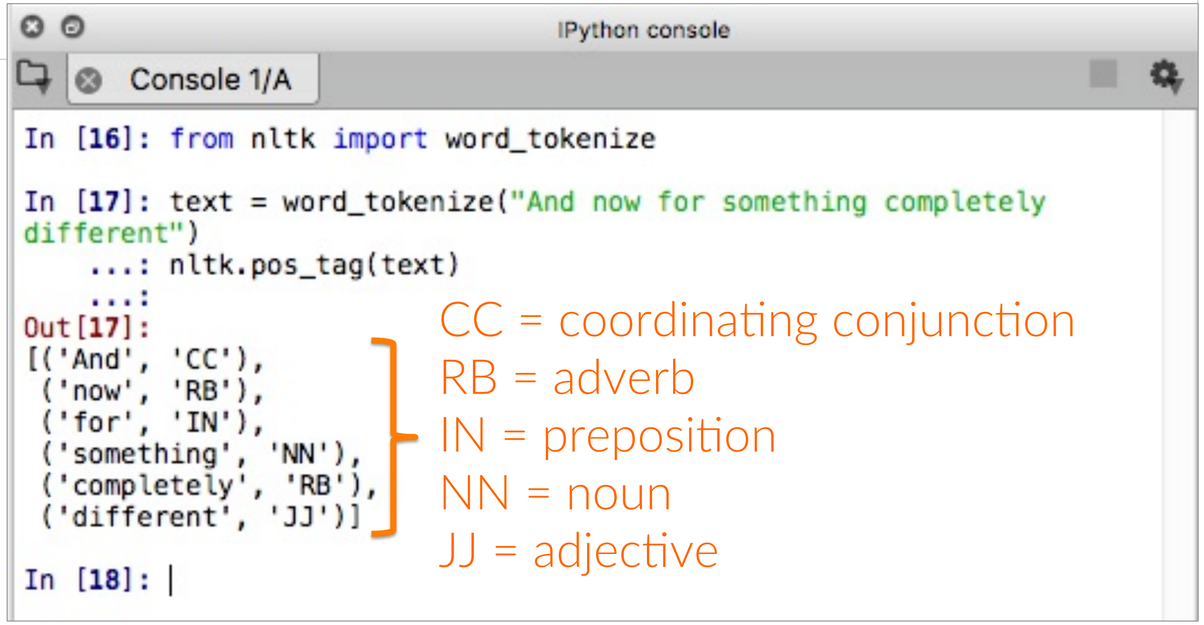
- We can do a simple analysis to automatically tag words

Using a tagger

Script

```
# Import nltk and word_tokenize
import nltk
from nltk import word_tokenize

# What are the parts of speech of the sentence below?
text = word_tokenize("And now for something completely different")
nltk.pos_tag(text)
```

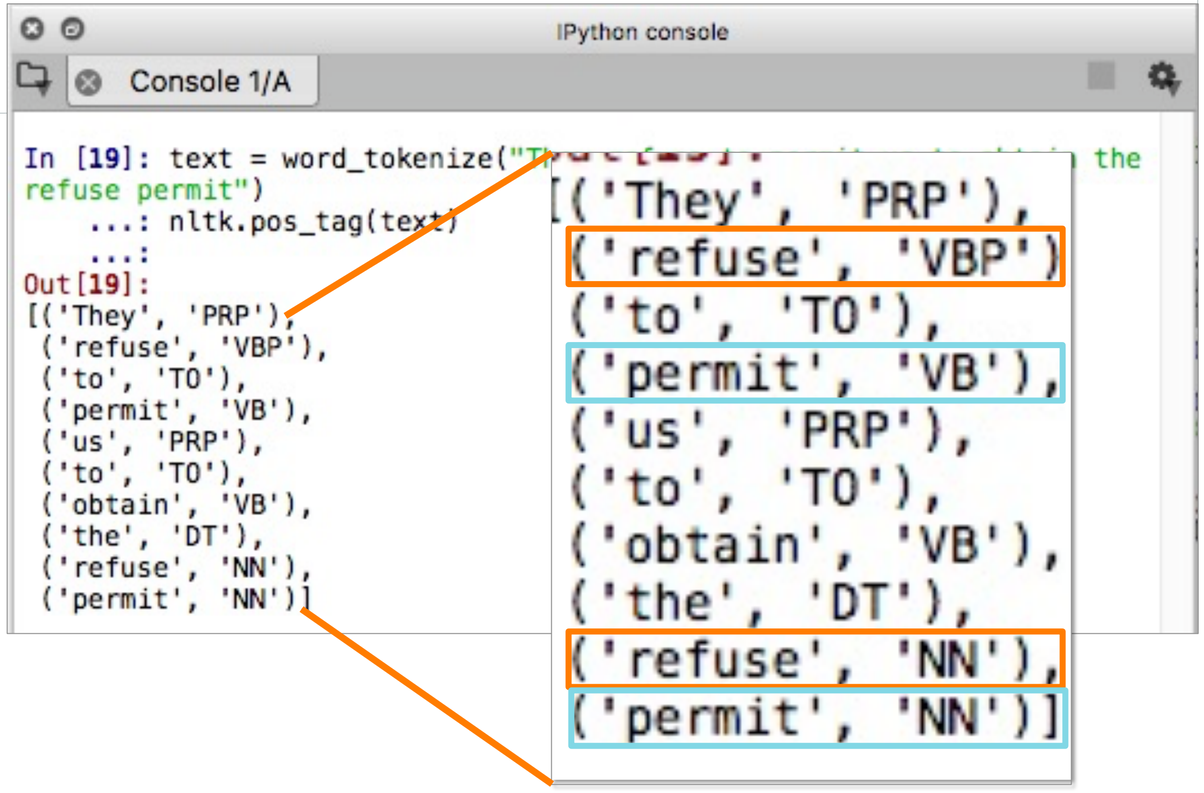


If you want to look at the entire list of categorizations, you can type in `nltk.help.upenn_tagset()` to see the full list

Using a tagger

Script

```
# What if we use homonyms (same word with a different meaning)?
text = word_tokenize("They refuse to permit us to obtain the refuse permit")
nltk.pos_tag(text)
```



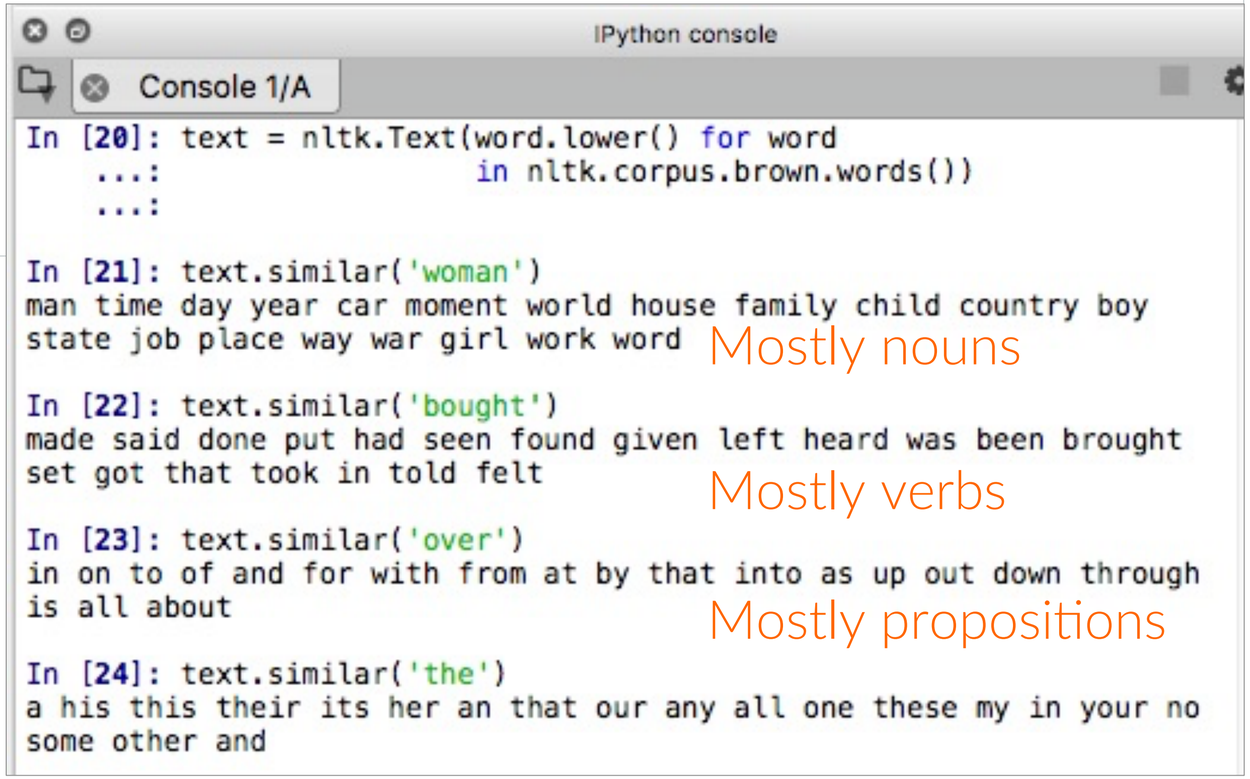
The POS tagger was able to distinguish between the different meanings of "refuse" and "permit"!

Using a tagger

Script

```
# How can we identify words that are used in similar context?
# First, we need to create the word list.
text = nltk.Text(word.lower() for word in nltk.corpus.brown.words())

text.similar('woman')
text.similar('bought')
text.similar('over')
text.similar('the')
```



Mostly nouns

Mostly verbs

Mostly propositions

The `text.similar` method takes the word we input, finds all the contexts of that word, and then finds words that appear in the same context

Parts of speech can help us understand the word distribution for analysis

Tagged tokens

- A tagged token is represented by a tuple (a sequence of elements) where the first element is the word, and the second element is the tag

Script

```
# 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_token
tagged_token[0]
tagged_token[1]
```

Creating tagged tokens

Script

```
# We can create a list of tokens manually
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 ''/'' ./.  '''

[nltk.tag.str2tuple(t) for t in sent.split()]
```

Built-in taggers

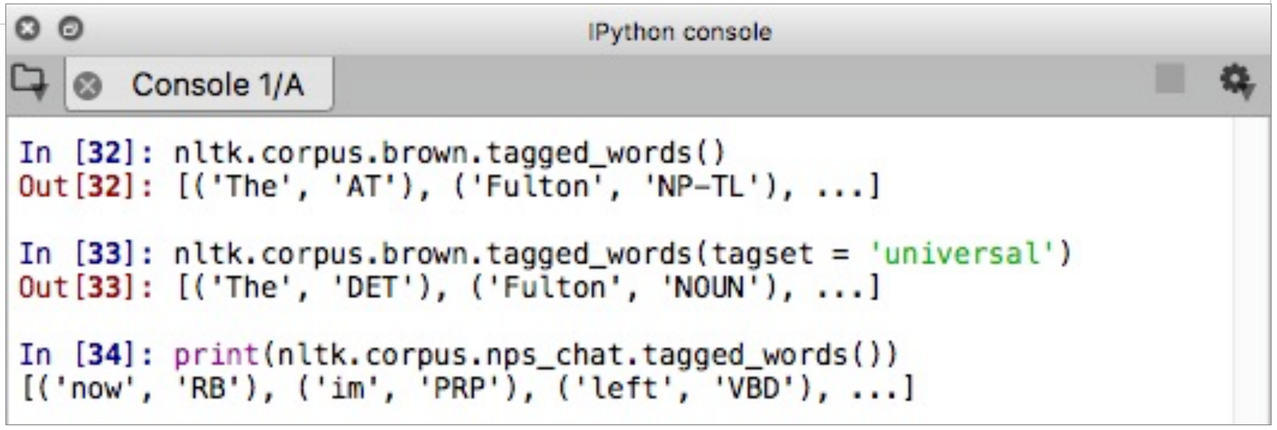
- Several corpora in NLTK have been tagged already

Script

```
# We can use the tagged_words() syntax to view the tags of the words.
nltk.corpus.brown.tagged_words()

# The universal tagset contains different terms for parts of speech.
nltk.corpus.brown.tagged_words(tagset = 'universal')

# If a corpus has tagged text, you'll be able to use the tagged_words() method.
print(nltk.corpus.nps_chat.tagged_words())
```



You can learn more about the universal tagset here:
https://github.com/slavpetrov/universal-pos-tags

Universal POS tags

Tag	Meaning	English example
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*

Tabulating tags

- Let's see which of these tags are most common in the news

Script

```
# Make sure to import the Brown corpus first.
from nltk.corpus import brown

# Now we'll pull all the words in the "news" category with a universal tag.
brown_news_tagged = brown.tagged_words(categories = 'news', tagset = 'universal')

# We'll use the FreqDist() syntax from nltk to calculate
# the frequency distribution of the tags.
tag_fd = nltk.FreqDist(tag for (word, tag) in brown_news_tagged)

# What are the most common parts of speech?
tag_fd.most_common()
```

Console 1/A

```
In [38]: tag_fd.most_common()
Out[38]:
[('NOUN', 30654),
 ('VERB', 14399),
 ('ADP', 12355),
 ('.', 11928),
 ('DET', 11389),
 ('ADJ', 6706),
 ('ADV', 3349),
 ('CONJ', 2717),
 ('PRON', 2535),
 ('PRT', 2264),
 ('NUM', 2166),
 ('X', 92)]
```

Exercise time!

Overview

1. Using a tagger
2. Identifying syntactic patterns
3. Using default dictionaries
4. Building and using taggers

Syntactic patterns: nouns

- What parts of speech typically appear before a noun?

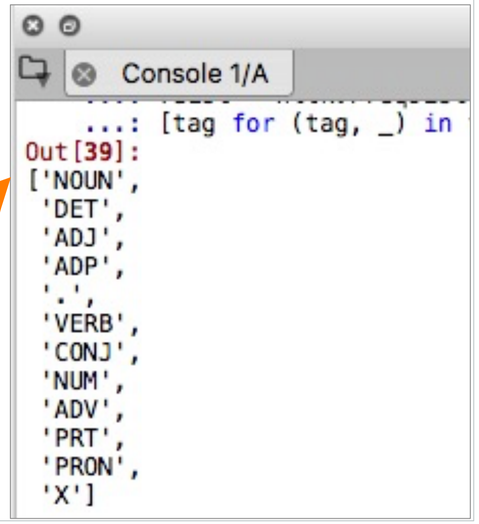
Script

```
# Let's create bigrams with the bigrams() syntax from NLTK.
word_tag_pairs = nltk.bigrams(brown_news_tagged)

# Now we will identify the words that appear before a noun in the bigrams
# we just created.
noun_preceders = [a[1] for (a, b) in word_tag_pairs if b[1] == 'NOUN']

# We'll use the FreqDist() method on the word list we
# just created to see which words occur most often before
fdist = nltk.FreqDist(noun_preceders)

# Let's pull out only the part of speech term
# (not the frequency).
[tag for (tag, _) in fdist.most_common()]
```



Looks like nouns and determiners are the top two parts of speech before nouns!

Syntactic patterns: verbs

- What are the most common verbs in news?

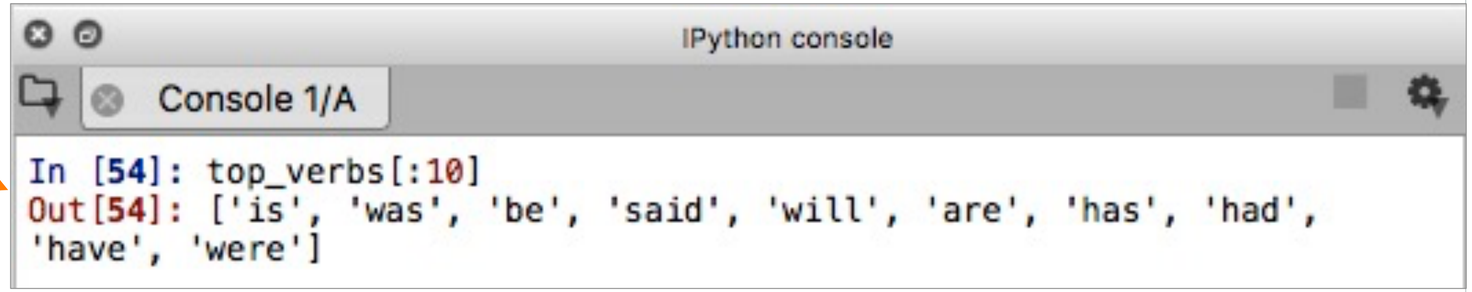
Script

```
# Let's create bigrams with the bigrams() syntax from NLTK.
word_tag_fd = nltk.FreqDist(brown_news_tagged)

# Now we will identify the most common words that are tagged as verbs in the
# frequency distribution.
top_verbs = [wt[0] for (wt, _) in word_tag_fd.most_common() if wt[1] == 'VERB']

# We can pull the top terms from the 'top_verbs variable.
top_verbs[:10]
```

Here are the top verbs in news



Conditional frequency distribution

- Since words and tags are paired, we can treat the word as a condition and the tag as an event

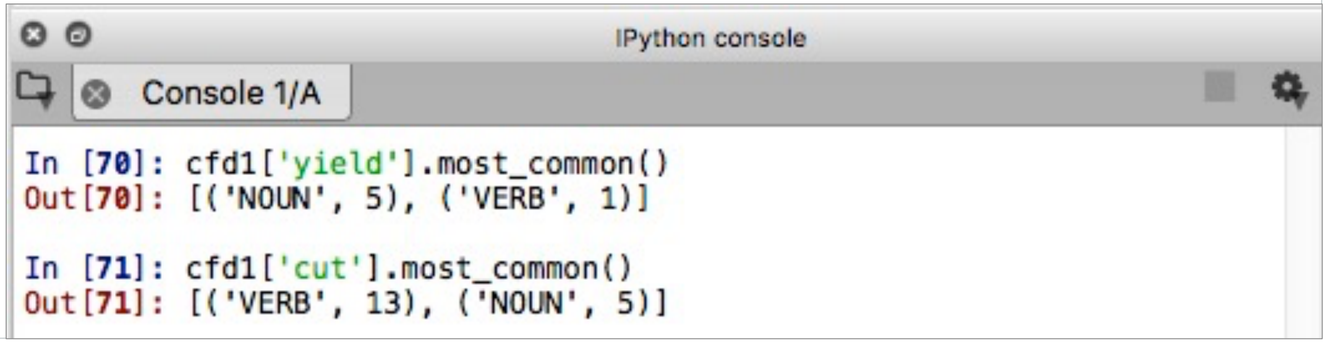
Script

```
# First, we'll set up the Conditional Frequency Distribution
cfd1 = nltk.ConditionalFreqDist(brown_news_tagged)

# Let's look at some words that have multiple meanings to see how they're most
# commonly used.
cfd1['yield'].most_common()

cfd1['cut'].most_common()
```

Remember, Conditional Frequency Distribution shows the distribution across multiple variables – in this case, it's tallying up the word across the different parts of speech



Conditional frequency distribution

- We can also reverse it and set the tags as the conditions and the word as the event so we can search by part of speech.

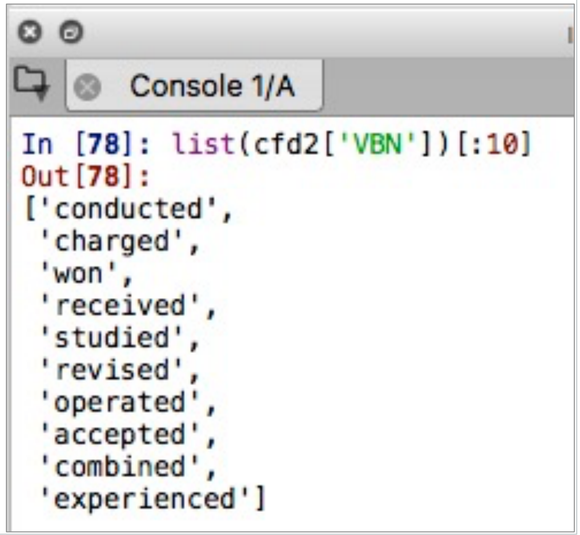
Script

```
# First, we'll set up the Conditional Frequency Distribution
brown_news_tagged_1 = brown.tagged_words(categories = 'news')


cfd2 = nltk.ConditionalFreqDist((tag, word) for (word, tag)
                                in brown_news_tagged_1)

# We can pull the top ten terms that
# are tagged as "verb past participle"
list(cfd2['VBN'])[:10]
```

Here we're switching the tags and the words so we can search by the tag



Conditional frequency distribution

- 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 surrounding text

Script

```
# First, we'll create the list where verbs are labeled as both VBD and VBN
[w for w in cfd1.conditions() if 'VBD' in cfd1[w] and 'VBN' in cfd1[w]]

# Let's shorten the variable to 'bnt' to make it easier to work with.
bnt = brown_news_tagged_1

# Now, we can look at 'kicked' labeled as VBD and
# pull out the indexed words before and after it
# to see the words and parts of speech around it.
idx1 = bnt.index(('kicked', 'VBD'))
bnt[idx1 - 4: idx1 + 1]

# Let's see if the words are different for VBN.
idx2 = bnt.index(('kicked', 'VBN'))
bnt[idx2 - 4: idx2 + 1]
```

Exercise time!

Overview

1. Using a tagger
2. Identifying syntactic patterns
3. Using default dictionaries
4. Automating tags
5. Building and using taggers

Default dictionaries

- Normally, we get errors when we try to access a key in a dictionary, but we can use a special dictionary to create new entries
- defaultdict automatically creates an entry for this new key and give it a default value, such as 0 or the empty list
- In order to use it, we have to supply a parameter which can be used to create the default value, e.g. int, float, str, list, dict, tuple

Default dictionaries

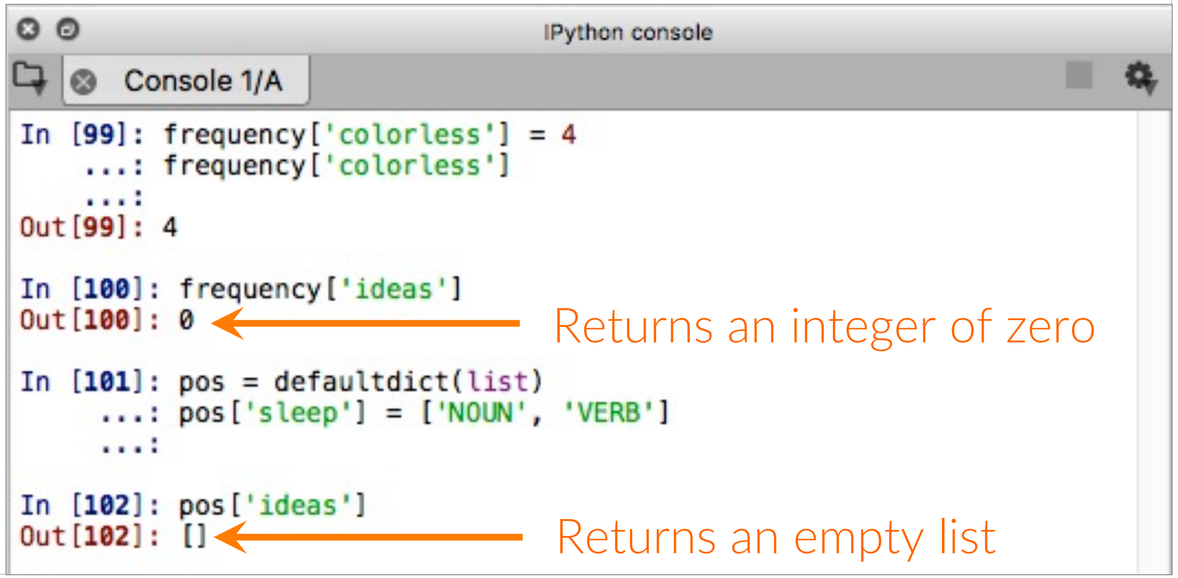
Script

```
# Let's import defaultdict.
from collections import defaultdict

# Set the defaultdict to accept integers as the parameter, and name it 'frequency'.
frequency = defaultdict(int)

# We can define the term 'colorless' to have the parameter '4'. We can check it
# by running it - what happens when we enter a new term that's not defined?
frequency['colorless'] = 4
frequency['colorless']
frequency['ideas']

# Set the defaultdict to set lists as
# the parameter, and name it 'pos'.
pos = defaultdict(list)
pos['sleep'] = ['NOUN', 'VERB']
pos['ideas']
```



Default dictionaries

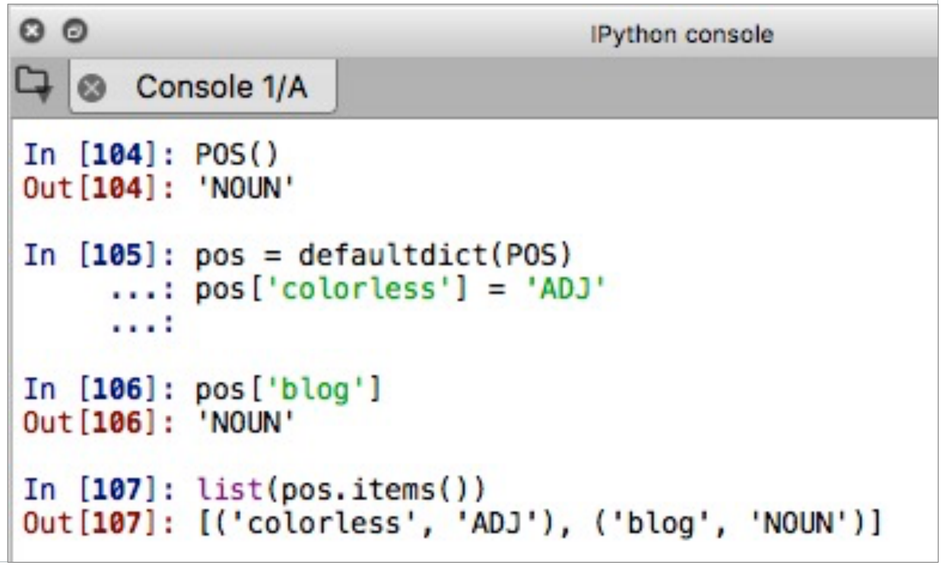
- We can set a default value for new entries in our dictionary

Script

```
# We first define the function that tells us if the POS is unknown,
# label it as a noun.
def POS():
    return 'NOUN'
POS()

# Let's set our default dictionary to the function
# that will return a noun. Now, if we put in a
# non-existent entry, it's automatically added.
pos = defaultdict(POS)
pos['colorless'] = 'ADJ'
pos['blog']

# We can look at the list of
# items from the dictionary.
list(pos.items())
```



Default dictionaries

- Many language processing tasks, such as tagging, have trouble identifying words that only appear once in a text (also known as hapaxes)
- They can perform better with a fixed vocabulary and a guarantee that no new words will appear.
- We can create a token, UNK, as a special 'out of vocabulary' token and assign it to all the low-frequency words

Default dictionaries

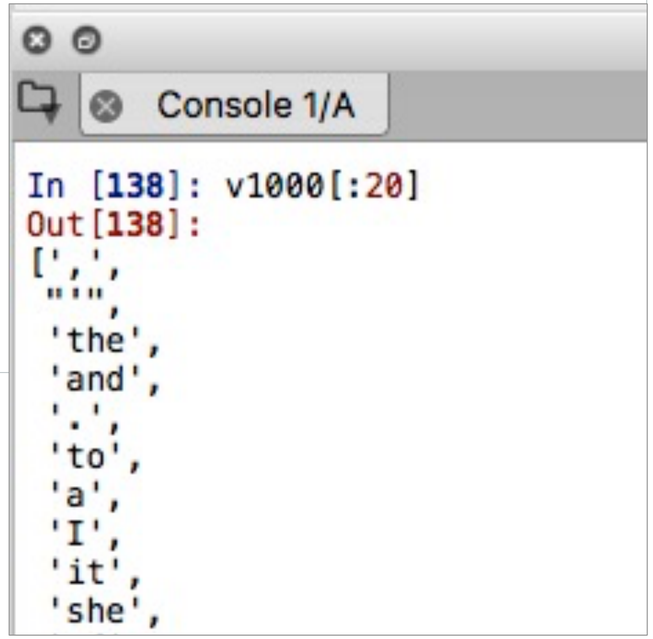
- We can set a default value for new entries in our dictionary

Script

```
# Let's read in "Alice in Wonderland" and save it as 'alice'.
alice = nltk.corpus.gutenberg.words('carroll-alice.txt')

# We'll calculate the Frequency Distribution of words, and then subset out the
# 1000 most common words.
vocab = nltk.FreqDist(alice)
v1000 = [word for (word, _) in vocab.most_common(1000)]

# Let's check the first 20 words of the set we just created.
v1000[:20]
```



Default dictionaries

- We can set a default value for new entries in our dictionary

Script

```
# We need to define a new function to automatically return the special
# token that we can define - let's call it 'UNK'.
def default_POS():
    return 'UNK'

# Set the defaultdict to return our special token and label it 'mapping'. Then,
# we will map 'v' to itself to assign
mapping = defaultdict(default_POS)
for v in v1000:
    mapping[v] = v
mapping

# Now, we'll use the mapping syntax again, but this time for all of Alice
# - now it will assign 'UNK' to any word that is not in the top 1000 common words.
alice2 = [mapping[v] for v in alice]
alice2[:100]
```

Exercise time!

Default dictionaries

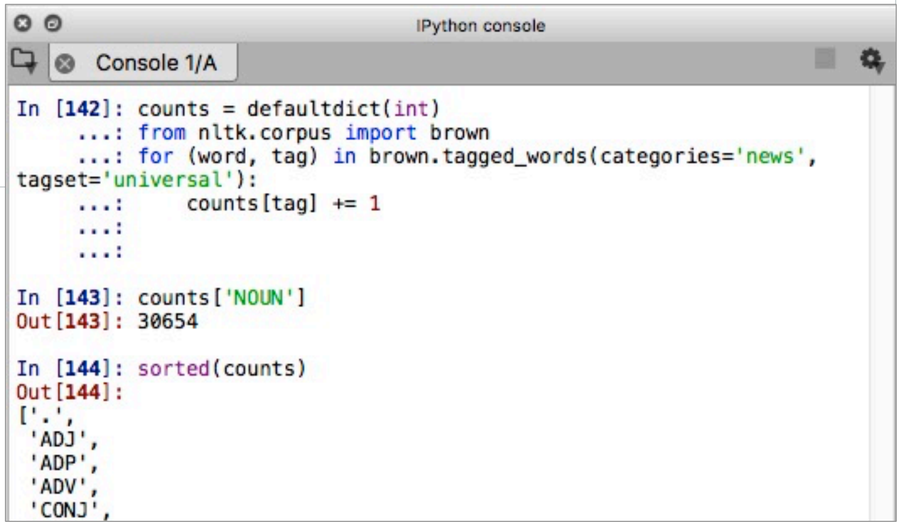
- We can add new occurrences incrementally to dictionaries

Script

```
# First, set the default dictionary for integers, and import brown.
counts = defaultdict(int)
from nltk.corpus import brown

# If the tag hasn't been seen before, it will have a zero count by default. Each time
# we encounter a tag, we increment its count using the += operator.
for (word, tag) in brown.tagged_words(categories = 'news', tagset = 'universal'):
    counts[tag] += 1

counts['NOUN']
```



Default dictionaries

- Itemgetter is important in helping us sort a dictionary by its values

Script

```
# First, import itemgetter.
from operator import itemgetter

sorted(counts.items(), key = itemgetter(1), reverse = True)

# Let's set pair equal to a tuple and look at how we can pull out a particular
# object from the tuple.
pair = ('NP', 8336)
pair[1]

itemgetter(1)(pair)
```

Items to sort – a list of tuples with POS tag and frequency

Returns a callable object that fetches an item

Tells us that the item should be returned in decreasing order

Exercise time!

Overview

1. Using a tagger
2. Identifying syntactic patterns
3. Using default dictionaries
4. Automating tags
5. Building and using taggers

Default dictionaries

- Let's review the concept of default dictionaries:

Sets the default value

```
my_dictionary = defaultdict(function to create default value)

for item in sequence:
     my_dictionary[item_key] is updated with information about item
```

Cycle through the data set and update information

Default dictionaries

- Let's index words according to their last two letters.

Script

```
# First, set 'last_letters' as a default dictionary for lists
last_letters = defaultdict(list)

# Set the wordlist to the NLTK corpus in English
words = nltk.corpus.words.words('en')

# Build the for loop where the key is equal to the last
# two letters, and then, we'll append the words
# to the matching key (which is the last two letters)
for word in words:
    key = word[-2:]
    last_letters[key].append(word)

# Let's try out our function, and pull the top ten terms!
last_letters['ly'][:10]
last_letters['zy'][:10]
```

IPython

Console 1/A

```
In [159]: last_letters['ly'][:10]
Out[159]:
['abactinally',
 'abandonedly',
 'abasedly',
 'abashedly',
 'abashlessly',
 'abbreviately',
 'abdominally',
 'abhorrently',
 'abidingly',
 'abiogenetically']

In [160]: last_letters['zy'][:10]
Out[160]:
['blazy',
 'bleezy',
 'blowzy',
 'boozy',
 'breezy',
 'bronzy',
 'buzzy',
 'Chazy',
 'cozy',
 'crazy']
```

Default dictionaries: anagrams

- Let's index words according to their last two letters.

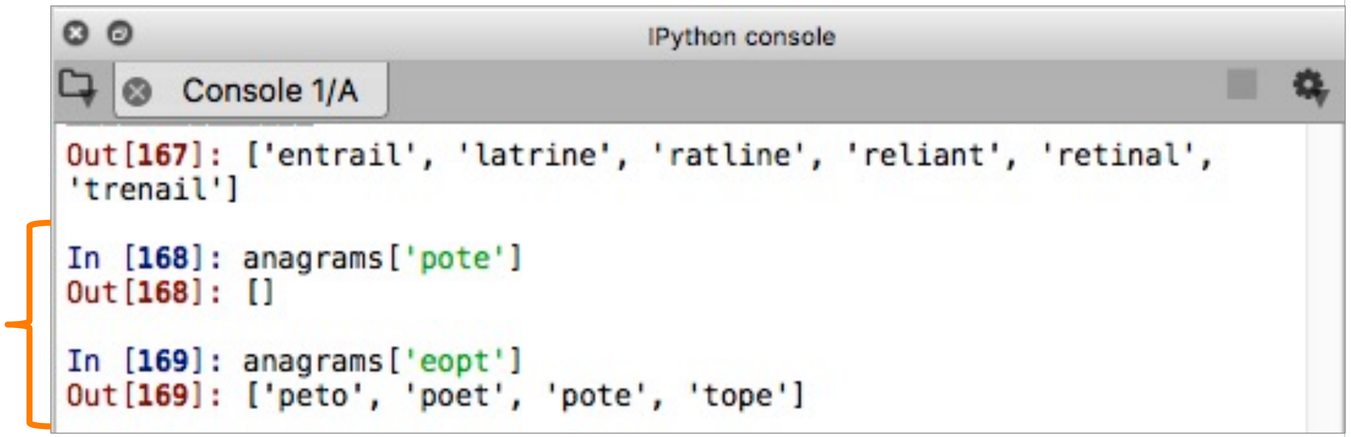
Script

```
# First, set 'last_letters' as a default dictionary for lists.
anagrams = defaultdict(list)

# Let's set a for loop, where the key is equal the sorted letters in a word, and
# the event is equal to the word itself - this way, for every word that has the
# same letters, it will have the same key.
for word in words:
    key = ''.join(sorted(word))
    anagrams[key].append(word)

# Let's test it out.
anagrams['aeilnrt']

# Note: this will only work if we put
# the letters in alphabetical order!
```



Exercise time!

Default taggers

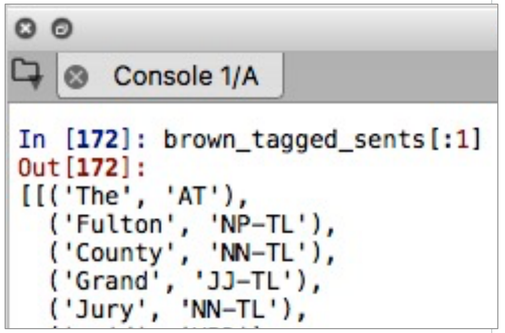
- Let's explore how to automatically add POS tags to text
- Since parts of speech can depend on the context of the sentence, we'll need to work with data at the sentence level

Script

```
# First, set 'last_letters' as a default dictionary for lists.
from nltk.corpus import brown

# We'll set up the regular sentences
# and the tagged sentences to compare them.
brown_tagged_sents = brown.tagged_sents(categories = 'news')
brown_sents = brown.sents(categories = 'news')

# Now we will see which tag is the most common in the text.
tags = [tag for (word, tag) in brown.tagged_words(categories = 'news')]
nltk.FreqDist(tags).max()
# NN
```



Default taggers

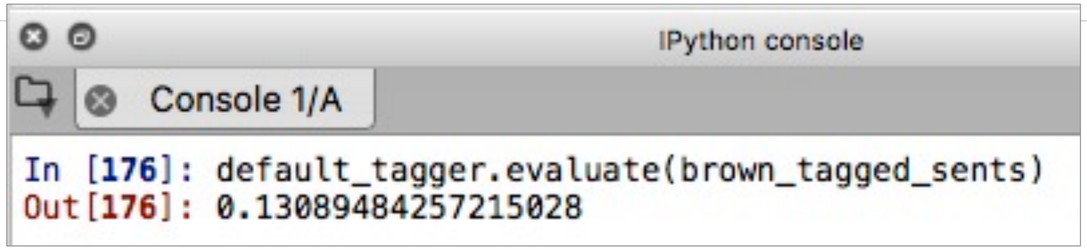
Script

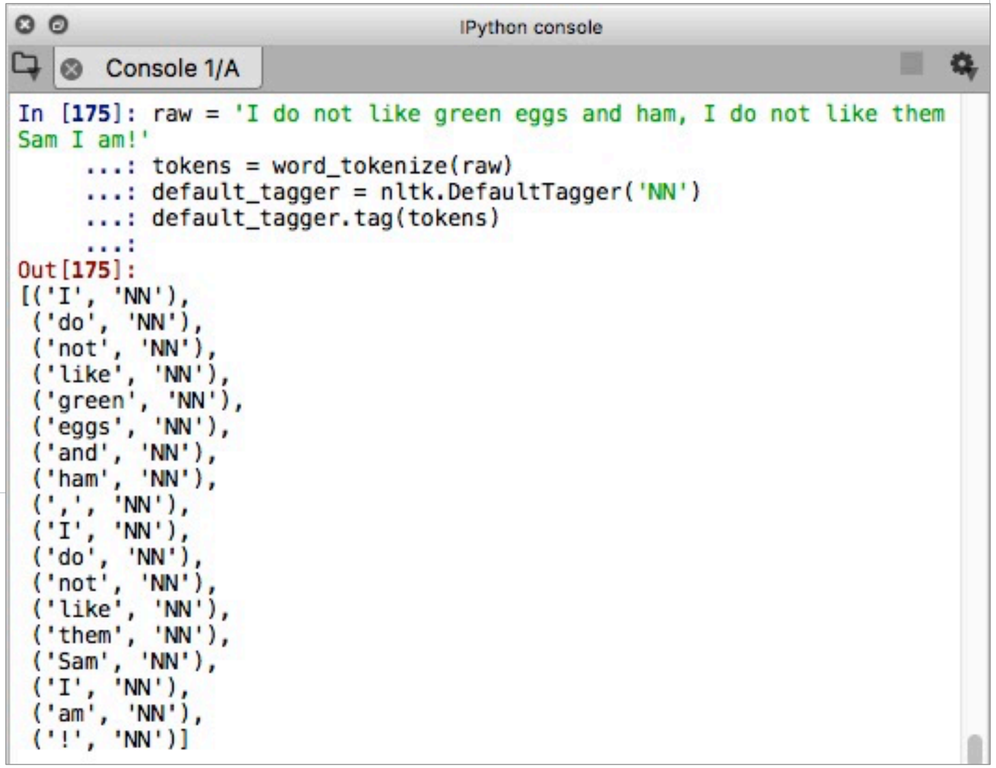
```
# Now we can create a tagger that will tag all words as 'NN'.
raw = 'I do not like green eggs and ham, I do not like them Sam I am!'

# We'll tokenize the words in the sentence and then use the default tagger to 'NN'.
tokens = word_tokenize(raw)
default_tagger = nltk.DefaultTagger('NN')

# Let's check to make sure it worked.
default_tagger.tag(tokens)


# Now we can evaluate the method
default_tagger.evaluate(brown_tagged_sents)
```





Overview

1. Using a tagger
2. Identifying syntactic patterns
3. Using default dictionaries
4. Automating tags
5. Building and using taggers

Lookup taggers

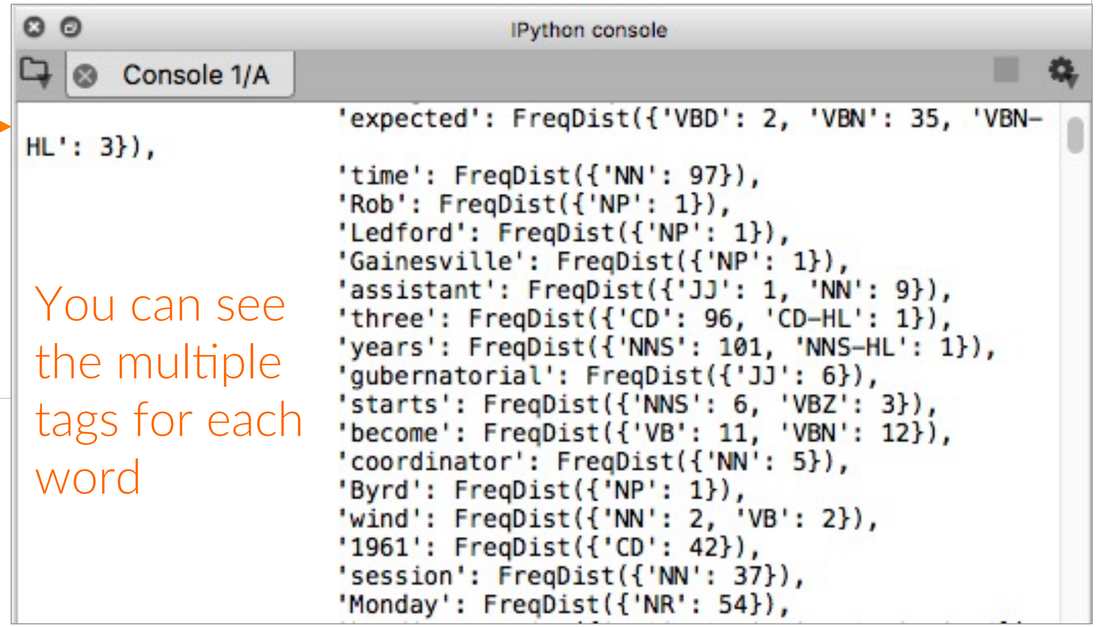
- Since a lot of high frequency words don't have the NN tag, let's find the most frequent words and store their tag so that we can refer to it later.

Script

```
# First we'll get the frequency distribution and the conditional
# frequency distribution for the Brown corpora categorized as 'news'.
fd = nltk.FreqDist(brown.words(categories = 'news'))
cfd = nltk.ConditionalFreqDist(brown.tagged_words(categories = 'news'))

# What does cfd look like?
cfd

# Now we will store the most common words
# from the Brown corpus.
corpus.most_freq_words = fd.most_common(100)
```



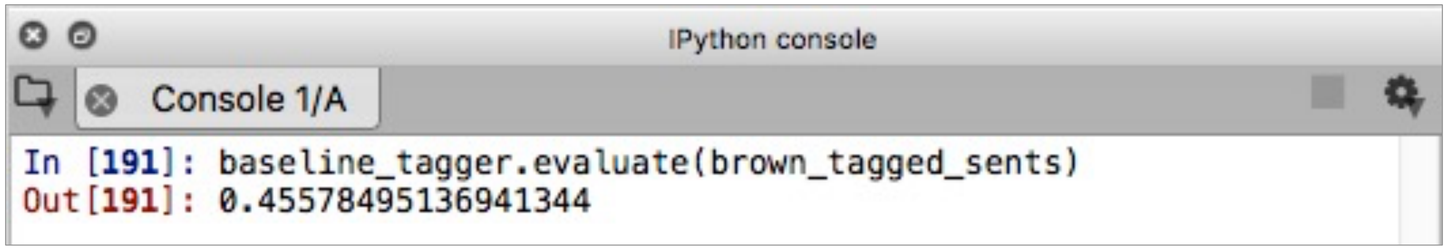
Lookup taggers

- Since a lot of high frequency words don't have the NN tag, let's find the most frequent words and store their tag so that we can refer to it later.

Script

```
# We'll set 'likely_tags' equal to the max count of tags for the words that
# are in the 'most_freq_words' data - remember, since some of the words have
# multiple parts of speech, we only want to take the most frequent one.
likely_tags = dict((word, cfd[word].max()) for (word, _) in most_freq_words)

# Now, we can use the UnigramTagger() method and set it as the baseline tagger.
# We can see how much more accurate we are if we tag the sentences with
# the lookup tagger.
baseline_tagger = nltk.UnigramTagger(model = likely_tags)
baseline_tagger.evaluate(brown_tagged_sents)
```

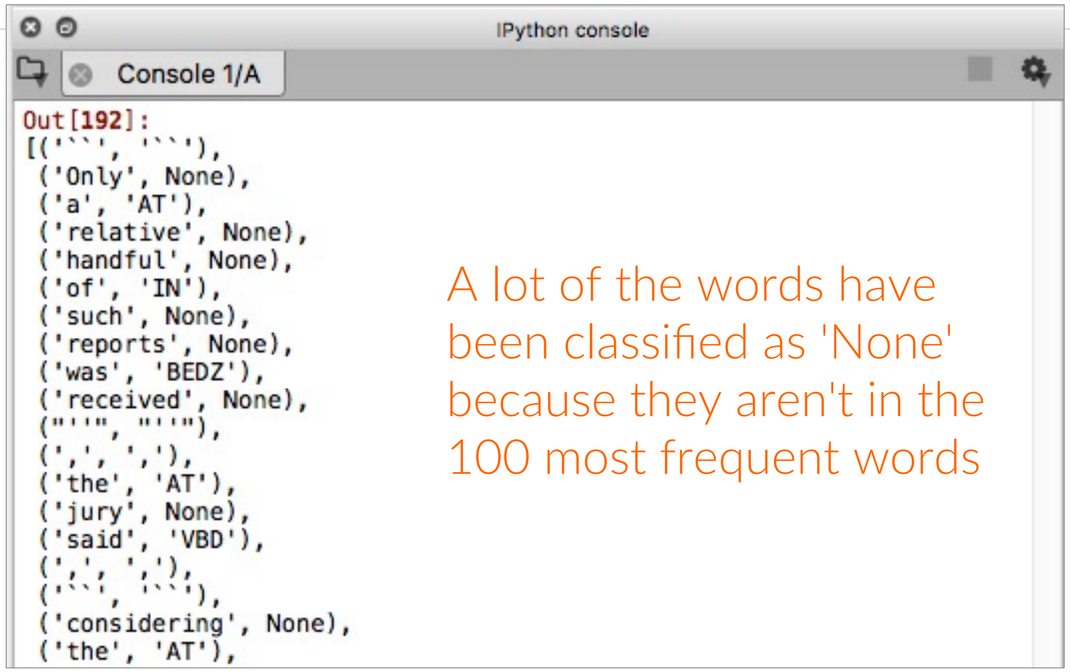


Lookup taggers

- Since a lot of high frequency words don't have the NN tag, let's find the most frequent words and store their tag so that we can refer to it later.

Script

```
# Let's test this out on untagged input text:
sent = brown.sents(categories = 'news')[3]
baseline_tagger.tag(sent)
```



A lot of the words have been classified as 'None' because they aren't in the 100 most frequent words

Lookup taggers

- We want to create a method where we want to lookup if the word is in the most frequent words, then if it isn't, assign it an 'NN' tag.

Script

```
# This is what that method we described would look like - if the word
# doesn't appear in the 'likely_tags' set, it will be assigned an 'NN' tag.
baseline_tagger = nltk.UnigramTagger(model = likely_tags,
                                     backoff = nltk.DefaultTagger('NN'))
```

Lookup taggers

- Let's put this all together and write a program to evaluate lookup taggers that have a range of sizes

Script

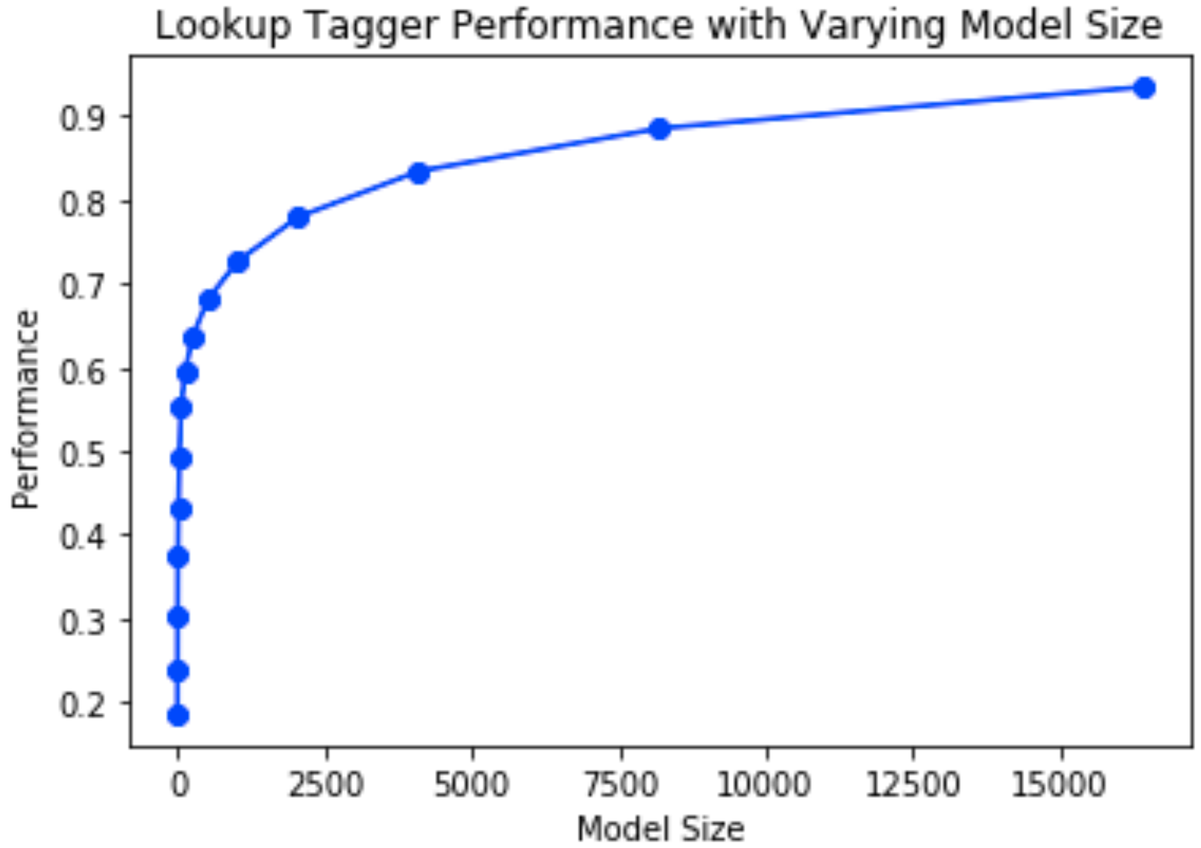
```
def performance(cfd, wordlist):
    lt = dict((word, cfd[word].max()) for word in wordlist)
    baseline_tagger = nltk.UnigramTagger(model = lt, backoff = nltk.DefaultTagger('NN'))
    return baseline_tagger.evaluate(brown.tagged_sents(categories = 'news'))

def display():
    import pylab
    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()
```

Lookup taggers

```
display()
```

Script



Exercise time!

Unigram taggers

- A unigram tagger is based on a simple statistical algorithm based on the likelihood of the tag for a particular token.

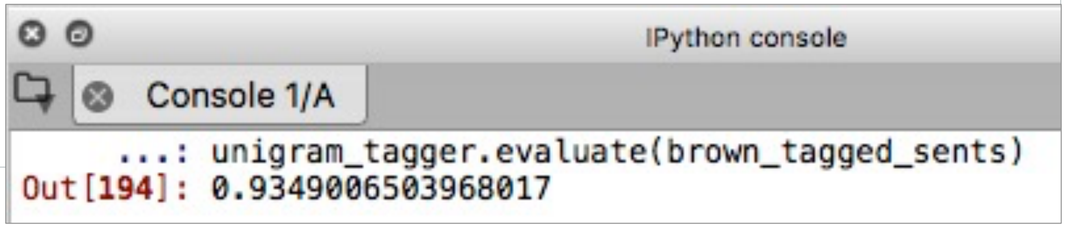
Script

```
# Let's import brown and set up the tagged sentences and regular sentences
# word lists. Then, we'll train the unigram tagger, use it to tag a sentence,
# and evaluate it.
from nltk.corpus import brown

brown_tagged_sents = brown.tagged_sents(categories = 'news')
brown_sents = brown.sents(categories = 'news')

# How does the unigram tagger perform on the sentences of 2007?
unigram_tagger = nltk.UnigramTagger(brown_tagged_sents)
unigram_tagger.tag(brown_sents[2007])

unigram_tagger.evaluate(brown_tagged_sents)
```



Separating training and testing data

- In order to make sure that our model is generalizable to new data, we have to train and test it. Let's split the data to train 90% of it, and test 10% of it.

Script

```
# We can split up the data to 90% of the tagged sentences to train.
size = int(len(brown_tagged_sents) * 0.9)
size

# Now, we will label a data set 'train_sents' and 'test_sents' to train the model.
train_sents = brown_tagged_sents[:size]
test_sents = brown_tagged_sents[size:]

# We'll train our model with UnigramTagger, and then test it with 'evaluate'.
unigram_tagger = nltk.UnigramTagger(train_sents)
unigram_tagger.evaluate(test_sents)
```

Even though the performance is worse, we have a better picture about the usefulness of this tagger →