Homework 3

Edward Hernández
College of William & Mary

Homework 3

Introduction

To start, I'm going to import the modules and data that I need. I'm importing the nltk tools individually for two reasons: to reduce load times, and to give me access to these functions directly, instead of calling them as, for example `nltk.ConditionalFreqDist()`, which gets unwieldy.

```
from nltk import FreqDist
from nltk import ConditionalFreqDist
from nltk import RegexpTagger
from nltk.corpus import brown
import re
```

Question 1

I'll start out by comparing the use of days of the week in the news and in romance novels. I'll make tuples[1] of the genres and target words I'm examining:

```
genres = ('news','romance')
days   = ('Monday', 'Tuesday', 'Wednesday', 'Thursday',
          'Friday', 'Saturday', 'Sunday')
```

I'll then make a conditional frequency distribution by genre for those words:

```
cfd = ConditionalFreqDist(
    (genre, word)
    for genre in brown.categories()
    for word in brown.words(categories=genre))
```

Then we'll print that in a table:

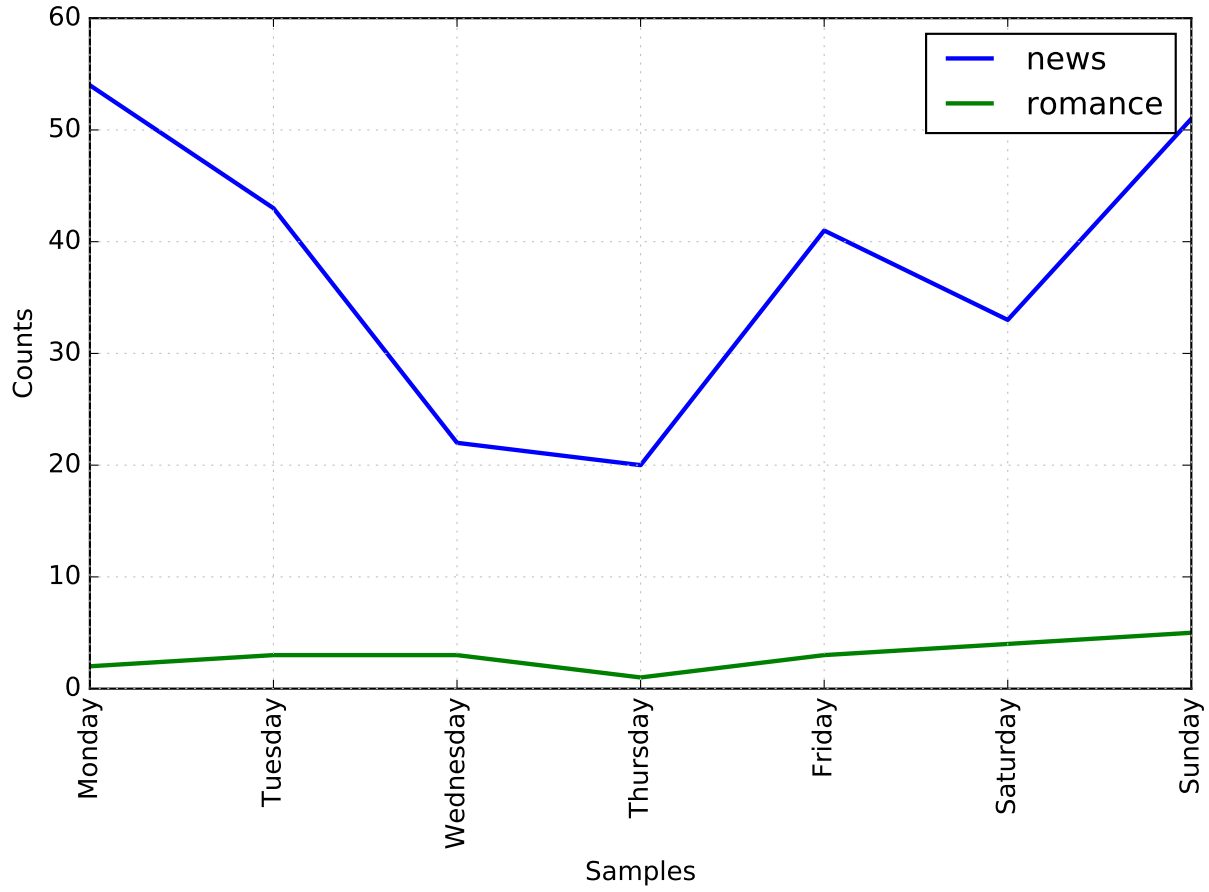
[1] Using tuples over lists here is not just a matter of preference (though it is my preference to pass immutable objects to functions I do not fully understand, where possible). When I tried to call `ConditionalFreqDist()` with list arguments I inconsistently received an error "`unhashable type: list`".

```
cfd.tabulate(conditions=genres, samples=days)
```

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
news	54	43	22	20	41	33	51
romance	2	3	3	1	3	4	5

And plot it:[2]

```
cfd.plot(conditions=genres, samples=days)
```



It looks like talking about days of the week is newsworthy, but not so romantic.

Question 2A

I'm also interested in the relative frequencies of different parts of speech in English. I'll make a frequency distribution of tagged words (according to the 'universal' tagset) in the news

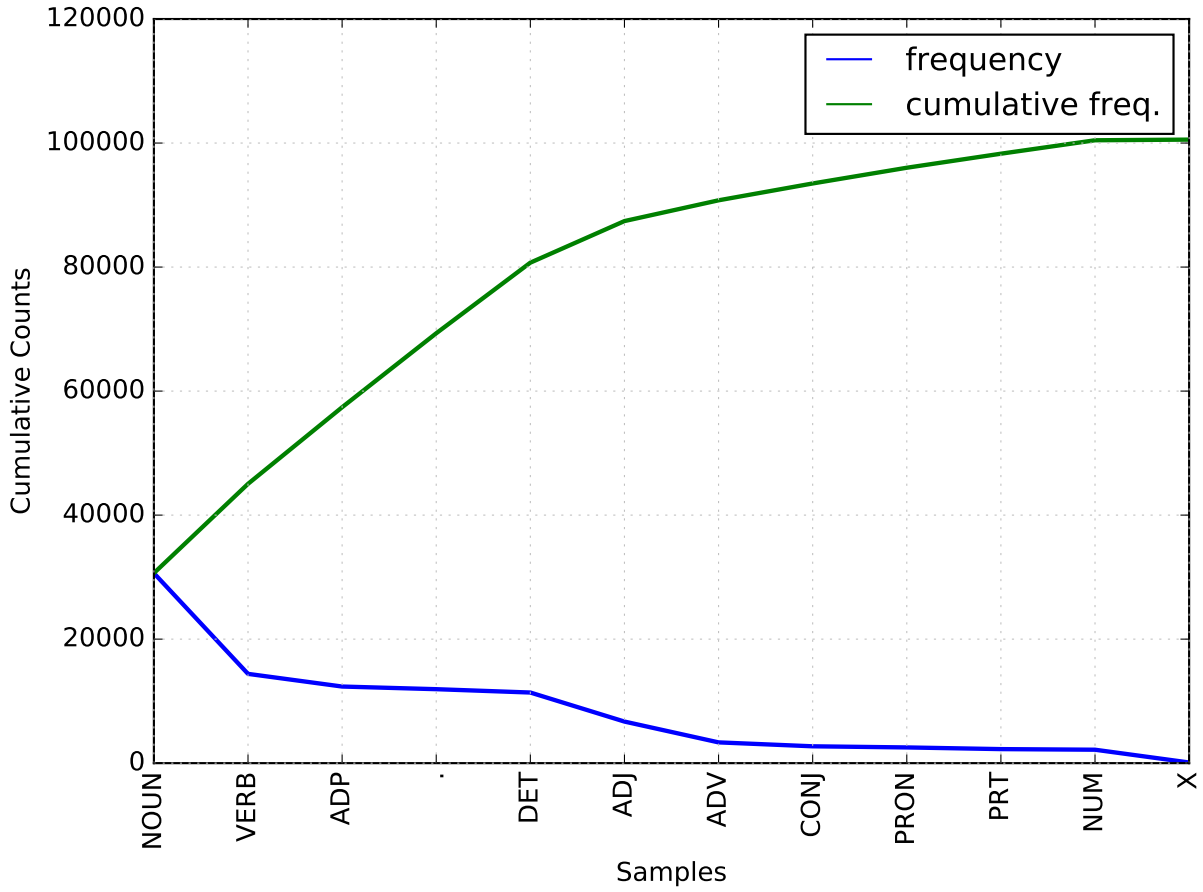
[2] matplotlib is finicky about the length of words in x-labels. To properly format the plot with fully visible labels, this code should be run in a directory containing a `matplotlibrc` containing "`figure.autolayout : True`".

genre of the Brown corpus.

```
news_tagged = brown.tagged_words(categories='news',
                                 tagset='universal')
tag_fd      = FreqDist(tag for (word, tag) in news_tagged)
```

I'll then plot the frequencies and cumulative frequencies of those parts of speech:

```
tag_fd.plot()
tag_fd.plot(cumulative=True)
```



From this plot, it looks like 80% of "words" are nouns, verbs, prepositions, determiners, and punctuation.

Question 2B

The NLTK book provides a toy example of building a part-of-speech tagger using regular expressions:

```
patterns = [
    (r'.*ing$',               'VBG'),
    (r'.*ed$',                'VBD'),
    (r'.*es$',                'VBZ'),
    (r'.*ould$',               'MD'),
    (r'.*\'s$',               'NN$'),
    (r'.*s$',                 'NNS'),
    (r'^-?[0-9]+(.[0-9]+)?$',  'CD'),
    (r'.*',                    'NN')
]

toy_tagger = RegexpTagger(patterns)
```

Though this seems like a perfectly reasonable way to tag text on first glance, this example is not all that accurate. Here's a quick function to check the accuracy of the tagger against the entire Brown Corpus:[3]

```
def evaluate(tagger):
    """evaluate a tagger against the Brown corpus"""
    acc = tagger.evaluate(brown.tagged_sents())
    return (round((acc * 100), 2))
```

The toy example is only 19.54% accurate when run over the whole Brown corpus. I initially figured that this was due to its size, and that adding more regular affixes as expressions, I could achieve a reasonably high accuracy. Here's my attempt:

```
patterns = [
    (r'.*[ai]ble$',           'JJ'),
    (r'.*al$',                'JJ'),
    (r'.*esque$',             'JJ'),
    (r'.*ful$',               'JJ'),
    (r'.*less$',              'JJ'),
```

[3] All accuracy ratings provided below are calculated and printed inline using this function.

```
    (r'.*ic$|.*ical$',        'JJ'),
    (r'.*ish$',               'JJ'),
    (r'.*ive$',               'JJ'),
    (r'.*ous$',               'JJ'),
    (r'.*er$',                'JJR'),
    (r'.*est$',               'JJT'),
    (r'.*ould$',              'MD'),
    (r'.*self$',              'PPL'),
    (r'.*selves$',            'PPLS'),
    (r'.*ly',                 'RB'),
    (r'.*ate$',               'VB'),
    (r'.*fy$$',               'VB'),
    (r'.*i[sz]es?$',          'VB'),
    (r'.*ed',                 'VBD'),
    (r'.*ing$',               'VBG'),
    (r'.*ing$',               'VBG'),
    (r'.*ed$',                'VBD'),
    (r'.*es$',                'VBZ'),
    (r'.*\'s$',               'NN$'),
    (r'.*s$',                 'NNS'),
    (r'^-?[0-9]+(.[0-9]+)?$', 'CD'),
    (r'.*',                   'NN'),
]


pattern_tagger = RegexpTagger(patterns)
```

This is still only 21.52% accurate. Unfortunately, this is largely because English words are not especially regular. It's very difficult to guess a part of speech by strings of characters in words (or even from knowing the whole word out of context). Critically, many of the most common words do not fit these patterns: 'a', 'in', 'for', 'the', etc.

Here is a numerically better regex tagger:

```
patterns = [
```

```
    (r'^\.$|^\?$|^:$|^;$',              '. '),
    (r'^,$',                            ',' ),
    (r'^``$',                           '``'),
    (r'^\'\'$',                         "''"),
    (r'^--$',                           '--'),
    (r'^\($',                           '('),
    (r'^\)$',                           ')'),
    (r'^[Nn][o\']t$',                   '*'),
    (r'^[Aa]ll$',                       'ABN'),
    (r'^[Tt]he$|^[Aa]n?$|^[Nn]o$',     'AT'),
    (r'^[Ll]ast$|^[Oo]ther$|^[Mm]ore$', 'AP'),
    (r'^[Mm]any$',                      'AP'),
    (r'^[Bb]e$',                        'BE'),
    (r'^[Ww]ere$',                      'BED'),
    (r'^[Bb]een$',                      'BEN'),
    (r'^[Aa]re$',                       'BER'),
    (r'^[Ii]s$',                        'BEZ'),
    (r'^[Ww]as$',                       'BEDZ'),
    (r'^[Aa]nd$|^[Oo]r$|^[Bb]ut$',     'CC'),
    (r'^[Oo]ne$|^[Tt]wo$|^[Tt]hree$',  'CD'),
    (r'^[Tt]hat$|^[Aa]s$|^[Ii]f$',     'CS'),
    (r'^[Ll]ike$',                      'CS'),
    (r'^[Tt]his$',                      'DET'),
    (r'^[Dd]o$',                        'DO'),
    (r'^[Dd]on\'t',                     'DO*'),
    (r'^[Dd]id$',                       'DOD'),
    (r'^[Dd]idn\'t',                    'DOD*'),
    (r'^[Dd]oes$',                      'DOZ'),
    (r'^[Ee]ach$',                      'DT'),
    (r'^[Ss]ome$|^[Aa]ny$',             'DTI'),
    (r'^[Tt]h[eo]se$',                  'DTS'),
    (r'^[Tt]here$',                     'EX'),
```

```
    (r'^have$',                         'HV'),
    (r'^[Hh]as$',                       'HVZ'),
    (r'^[Hh]ad$',                       'HVD'),
    (r'^[Oo]f*$|^[Ff]or$|^[Ww]ith$',   'IN'),
    (r'^[Oo]n$|^[Ff]rom$|^[Oo]ver$',   'IN'),
    (r'^[Bb]y$|^[Aat]$|^[Aa]bout$',    'IN'),
    (r'^[Ii]nto$|^[Tt]hrough$|^[Ii]n$', 'IN'),
    (r'^[Tt]o$',                        'TO'),
    (r'^[Nn]ew$|^[Ss]uch$',             'JJ'),
    (r'^[Ww]ill$|.*ould$|^[Cc]an$',    'MD'),
    (r'^[Mm]ay$|^[Mm]ust$',             'MD'),
    (r'^Mrs?\.$',                       'NP'),
    (r'^[Pp]eople$',                    'NNS'),
    (r'^[Ff]irst$|^[Ss]econd$',         'OD'),
    (r'^him$|^her$|^them$',             'PPO'),
    (r'^[Hh]e$|^[Ss]he$|^[Ii]t$',      'PPS'),
    (r'^I$|^[Tt]hey$|^[Ww]e$|^[Yy]ou$', 'PPSS'),
    (r'^[Hh]is$|^[Tt]heir$|^[Yy]our$',  'PP$'),
    (r'^[Ii]ts$|^[Mm]y$|^[Oo]ur$',     'PP$'),
    (r'^[Mm]ore$',                      'QL'),
    (r'^[Aa]lso$|^[Nn]ow$|^[Tt]hen$',   'RB'),
    (r'^[Ee]ven$',                      'RB'),
    (r'^[Uu]p$|^[Oo]ut$',               'RP'),
    (r'^[Ww]ho$',                       'WPS'),
    (r'^[Ww]here',                      'WRB'),
    (r'^[Ww]hen$',                      'WRD'),
    (r'^[Ww]hich$|^[Ww]hat$',           'WDT'),
    (r'^[Mm]ake$',                      'VB'),
    (r'.*',                             'NN')
]

cheating_tagger = RegexpTagger(patterns)
```

This is 56.51% accurate, but obviously it is not in the spirit of a regex tagger. All it does is account for approximately one hundred of the most common English words (which are almost all irregular). To optimize the regex tagger in English, I've had to crudely approximate a lookup tagger.[4] In fact, when I added the regular expressions from above the tagger only becomes 3.98% more accurate.

Combining the faux-lookup tagger and the affix expressions gives me an accuracy of 60.49%, which is pretty nearly the best I can do. Here is the final tagger that I used to get that accuracy:

```
patterns = [
    (r'^\.$|^\?$|^:$|^;$',              '. '),
    (r'^,$',                            ',' ),
    (r'^``$',                           '``'),
    (r'^\'\'$',                         "''"),
    (r'^--$',                           '--'),
    (r'^\($',                           '(' ),
    (r'^\)$',                           ')' ),
    (r'^[Nn][o\']t$',                   '*' ),
    (r'^[Aa]ll$',                       'ABN'),
    (r'^[Tt]he$|^[Aa]n?$|^[Nn]o$',     'AT'),
    (r'^[Ll]ast$|^[Oo]ther$|^[Mm]ore$', 'AP'),
    (r'^[Mm]any$',                      'AP'),
    (r'^[Bb]e$',                        'BE'),
    (r'^[Ww]ere$',                      'BED'),
    (r'^[Bb]een$',                      'BEN'),
    (r'^[Aa]re$',                       'BER'),
    (r'^[Ii]s$',                        'BEZ'),
    (r'^[Ww]as$',                       'BEDZ'),
    (r'^[Aa]nd$|^[Oo]r$|^[Bb]ut$',     'CC'),
    (r'^[Oo]ne$|^[Tt]wo$|^[Tt]hree$',  'CD'),
```

[4] Lookup taggers still have issues (such as not considering context), but they seem to systematically outperform regex taggers.

```
    (r'^[Tt]hat$|^[Aa]s$|^[Ii]f$',      'CS'),
    (r'^[Ll]ike$',                      'CS'),
    (r'^[Tt]his$',                      'DET'),
    (r'^[Dd]o$',                        'DO'),
    (r'^[Dd]on\'t',                     'DO*'),
    (r'^[Dd]id$',                       'DOD'),
    (r'^[Dd]idn\'t',                    'DOD*'),
    (r'^[Dd]oes$',                      'DOZ'),
    (r'^[Ee]ach$',                      'DT'),
    (r'^[Ss]ome$|^[Aa]ny$',             'DTI'),
    (r'^[Tt]h[eo]se$',                  'DTS'),
    (r'^[Tt]here$',                     'EX'),
    (r'^have$',                         'HV'),
    (r'^[Hh]as$',                       'HVZ'),
    (r'^[Hh]ad$',                       'HVD'),
    (r'^[Oo]f*$|^[Ff]or$|^[Ww]ith$',    'IN'),
    (r'^[Oo]n$|^[Ff]rom$|^[Oo]ver$',    'IN'),
    (r'^[Bb]y$|^[Aat]$|^[Aa]bout$',     'IN'),
    (r'^[Ii]nto$|^[Tt]hrough$|^[Ii]n$', 'IN'),
    (r'^[Tt]o$',                        'TO'),
    (r'^[Nn]ew$|^[Ss]uch$',             'JJ'),
    (r'^[Ww]ill$|.*ould$|^[Cc]an$',     'MD'),
    (r'^[Mm]ay$|^[Mm]ust$',             'MD'),
    (r'^Mrs?\.$',                       'NP'),
    (r'^[Pp]eople$',                    'NNS'),
    (r'^[Ff]irst$|^[Ss]econd$',         'OD'),
    (r'^him$|^her$|^them$',             'PPO'),
    (r'^[Hh]e$|^[Ss]he$|^[Ii]t$',       'PPS'),
    (r'^I$|^[Tt]hey$|^[Ww]e$|^[Yy]ou$', 'PPSS'),
    (r'^[Hh]is$|^[Tt]heir$|^[Yy]our$',  'PP$'),
    (r'^[Ii]ts$|^[Mm]y$|^[Oo]ur$',      'PP$'),
    (r'^[Mm]ore$',                      'QL'),
```

```
    (r'^[Aa]lso$|^[Nn]ow$|^[Tt]hen$',   'RB'),
    (r'^[Ee]ven$',                      'RB'),
    (r'^[Uu]p$|^[Oo]ut$',               'RP'),
    (r'^[Ww]ho$',                       'WPS'),
    (r'^[Ww]here',                      'WRB'),
    (r'^[Ww]hen$',                      'WRD'),
    (r'^[Ww]hich$|^[Ww]hat$',           'WDT'),
    (r'^[Mm]ake$',                      'VB'),
    (r'.*ed',                           'VBD'),
    (r'.*ly',                           'RB'),
    (r'.*s',                            'NNS'),
    (r'.*ate$',                         'VB'),
    (r'.*fy$$',                         'VB'),
    (r'.*i[sz]es?$',                    'VB'),
    (r'.*ing$',                         'VBG'),
    (r'.*es',                           'VBZ'),
    (r'.*self$',                        'PPL'),
    (r'.*selves$',                      'PPLS'),
    (r'.*[ai]ble$',                     'JJ'),
    (r'.*al$',                          'JJ'),
    (r'.*esque$',                       'JJ'),
    (r'.*ful$',                         'JJ'),
    (r'.*ic$|.*ical$',                  'JJ'),
    (r'.*ive$',                         'JJ'),
    (r'.*ous$',                         'JJ'),
    (r'.*ish$',                         'JJ'),
    (r'.*less$',                        'JJ'),
    (r'.*er$',                          'JJR'),
    (r'.*est$',                         'JJT'),
    (r'.*',                             'NN')
]
```

```
final_tagger = RegexpTagger(patterns)
```