Introduction to Computational Linguistics

Follow along / resources at www.github.com/jonathandunn/Lectures

Modules

Data

Representations

Models

Data

Text:

Data Sources, Data Formats, Encoding, Tokenization, Documents, Strings

Data

Knowledge:

Annotations, Mark-Up, Word Alignment, Knowledge Graphs, Meta-Data

Representations

N-Grams, Embeddings, Concepts / Entities, Strings and Trees, Sequences

Models

Families of Models:

Classification, Clustering, Language Models

Models

Training Models:

Training/Testing, Cross-Validation, Updateable Models, Data Sampling

Models

Pipelines

Background data (unsupervised and semi-supervised), chains of models

N-Grams and Sequences

This lesson focuses on **sequences**:

list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

(1) Indexing

We can access an item in a sequence by its location:

line[0] is 1

line[1] is 2

line[2] is 3

(2) Iterating

We can access all items in order by looping: **for**

```
In [1]: for item in [1, 2, 3, 4, 5]:
            print(item)
```

```
1
2
3
4
5
```

(3) Data Types

strings are sequences of characters: "This is a string"

lists are arrays of any data type: ["this", 1, 2.0, False]

We start by adding our previous functions to the workspace.

In [23]:

```
def read_corpus(file):
    import codecs
    with codecs.open(file, "r", encoding = "utf-8") as fo:
        for line in fo:
            yield line
```

Now read_corpus() creates a **generator** that feeds lines from a file.

In [24]:

```
for line in read_corpus("./Corpora/eng.Test.txt"):
    print(line)
```

```
This - here - is the first test sentence.


This sentence: after the first test sentence!


HERE IS THE THIRD (TEST) SENTENCE.
```

In [25]:

```
def tokenize(line_in):

    punctuation = ['"', ".", ",", "?", "!", "-", ")", "(", ":", ";"]
    line_in = line_in.strip().lower()
    line_out = ""

    for char in line_in:
        if char not in punctuation:
            line_out += char

    return line_out.replace("  ", " ")
```

Now tokenize() returns a lowercase string with certain symbols removed.

In [26]:

```
for line in read_corpus("./Corpora/eng.Test.txt"):
    line = tokenize(line)
    print(line)
```

```
this here is the first test sentence
this sentence after the first test sentence
here is the third test sentence
```

With *strings* we can iterate over characters:

In [27]:

```
for char in "This is a line":
    print(char)
```

```
T
h
i
s
 
i
s
 
a
 
l
i
n
e
```

To iterate over words, we need to split the string. This creates a *list* of words.

In [28]:

```
line = "This is a line"
line_list = line.split(" ")

for word in line_list:
        print(word)
```

```
This
is
a
line
```

An **index** refers to a specific location in a sequence. In Python, indexes begin at 0.

In [29]:

```
print(line[0])
print(line[1])
```

```
T
h
```

Both strings and lists can be accessed by index.

In [30]:

```
print(line_list[0])
print(line_list[1])
```

```
This
is
```

The *range()* function iterates over integers.

In [31]:

```
for i in range(1, 5):
    print(i)
```

```
1
2
3
4
```

The *len()* function tells us how many units are in a sequence (for strings and lists).

In [32]:

```
print(len(line))
print(len(line_list))
```

```
14
4
```

We can iterate over a sequence by index by combining range() and len()

In [33]:

```
for i in range(len(line)):
    print(line[i])
```

```
T
h
i
s
 
i
s
 
a
 
l
i
n
e
```

In [34]:

```
for i in range(len(line_list)):
    print(line_list[i])
```

```
This
is
a
line
```

We can use the index to find the *window* of neighboring words or characters.

In [35]:

```
for i in range(1, len(line_list)):
    print(line_list[i-1], line_list[i])
```

```
This is
is a
a line
```

These sequences are called **n-grams**, where *n* is the length. Bigrams have length 2. Trigrams have length 3.

We can use this to make a function that takes a string and returns all the bigrams.

In [36]:

```python
def get_bigrams(line):

    bigrams = []  #Initialize list of bigrams

    line = tokenize(line)
    line = line.split(" ")

    for i in range(1, len(line)):
        bigrams.append((line[i-1], line[i]))

    return bigrams
```

The function get_bigrams() allows us to easily collect bigrams across lines from a corpus.

In [37]:

```
for line in read_corpus("./Corpora/eng.Test.txt"):
    print(get_bigrams(line))
```

```
[('this', 'here'), ('here', 'is'), ('is', 'the'), ('the', 'first'), ('first',
'test'), ('test', 'sentence')]
[('this', 'sentence'), ('sentence', 'after'), ('after', 'the'), ('the', 'first
'), ('first', 'test'), ('test', 'sentence')]
[('here', 'is'), ('is', 'the'), ('the', 'third'), ('third', 'test'), ('test',
'sentence')]
```

Here we see three *lists*, each containing a *tuple* with two *strings*.

We will count bigrams by using a *dictionary*.

The keys of the dictionary will be bigrams and the values will be the current count.

In [38]:

```
from collections import defaultdict

bigram_count = defaultdict(int)
```

In [39]:

```
for line in read_corpus("./Corpora/eng.Test.txt"):
    for bigram in get_bigrams(line):
        bigram_count[bigram] += 1
```

We can look at the contents of bigram_count using *keys()*

In [40]:

```
for key in bigram_count.keys():
    print(key, bigram_count[key])
```

```
('this', 'here') 1
('here', 'is') 2
('is', 'the') 2
('the', 'first') 2
('first', 'test') 2
('test', 'sentence') 3
('this', 'sentence') 1
('sentence', 'after') 1
('after', 'the') 1
('the', 'third') 1
('third', 'test') 1
```

Let's make this a function so it can be reused.

In [41]:

```
def count_bigrams(filename):

    corpus = read_corpus(filename)
    bigram_count = defaultdict(int)

    for line in corpus:
        for bigram in get_bigrams(line):
            bigram_count[bigram] += 1

    return bigram_count
```

In [42]:

```
bigram_count = count_bigrams("./corpora/eng.Test.txt")
print(bigram_count)
```

```
defaultdict(<class 'int'>, {('this', 'here'): 1, ('here', 'is'): 2, ('is', 'th
e'): 2, ('the', 'first'): 2, ('first', 'test'): 2, ('test', 'sentence'): 3, ('
this', 'sentence'): 1, ('sentence', 'after'): 1, ('after', 'the'): 1, ('the',
'third'): 1, ('third', 'test'): 1})
```

Now we can view the bigrams by the most frequent.

In [43]:

```
top_bigrams = sorted(bigram_count, key = bigram_count.get)

for i in range(0, 6):
    print(top_bigrams[i])
```

```
('this', 'here')
('this', 'sentence')
('sentence', 'after')
('after', 'the')
('the', 'third')
('third', 'test')
```

We have five real-world corpora in the ./corpora/ folder:

```
eng.Europarl.txt: European Parliament proceedings
eng.NewsCommentary.txt: News articles and editorials
eng.OpenSubs.txt: Movie subtitles
eng.TED_Talks.txt: TED Talk scripts
eng.Web.txt: Web-crawled data
```

Each corpus contains about a million words. All are from the same language. Do they have the same top bigrams?

In [44]:

```
europarl_bigrams = count_bigrams("./corpora/eng.Europarl.txt")
news_bigrams = count_bigrams("./corpora/eng.NewsCommentary.txt")
opensubs_bigrams = count_bigrams("./corpora/eng.OpenSubs.txt")
ted_bigrams = count_bigrams("./corpora/eng.TED_Talks.txt")
web_bigrams = count_bigrams("./corpora/eng.Web.txt")
```

In [45]:

```
top_europarl = sorted(europarl_bigrams, key = europarl_bigrams.get, reverse = True)
top_news = sorted(news_bigrams, key = news_bigrams.get, reverse = True)
top_opensubs = sorted(opensubs_bigrams, key = opensubs_bigrams.get, reverse = True)
top_ted = sorted(ted_bigrams, key = ted_bigrams.get, reverse = True)
top_web = sorted(web_bigrams, key = web_bigrams.get, reverse = True)
```

```
In [51]: print("Europarl")
         for i in range(10):
             print(top_europarl[i], europarl_bigrams[top_europarl[i]])
```

```
Europarl
('of', 'the') 3822
('in', 'the') 2108
('to', 'the') 1386
('the', 'european') 1293
('on', 'the') 1229
('it', 'is') 1182
('that', 'the') 1099
('and', 'the') 1007
('the', 'commission') 908
('for', 'the') 894
```

```
In [52]: print("NewsCommentary")
         for i in range(10):
             print(top_news[i], news_bigrams[top_news[i]])
```

```
NewsCommentary
('of', 'the') 2002
('in', 'the') 1821
('to', 'the') 864
('’', 's') 784
('and', 'the') 713
('it', 'is') 584
('that', 'the') 568
('on', 'the') 548
('the', 'us') 545
('to', 'be') 541
```

In [53]:

```
print("OpenSubs")
for i in range(10):
    print(top_opensubs[i], opensubs_bigrams[top_opensubs[i]])
```

```
OpenSubs
('i', "'m") 2041
('it', "'s") 1975
('don', "'t") 1863
('you', "'re") 1239
('', 'i') 972
('in', 'the') 961
('that', "'s") 876
('of', 'the') 805
('i', "'ll") 779
('i', 'don') 662
```

In [54]:

```
print("TED Talks")
for i in range(10):
    print(top_ted[i], ted_bigrams[top_ted[i]])
```

```
TED Talks
('of', 'the') 1925
('in', 'the') 1733
('this', 'is') 1036
('and', 'i') 793
('and', 'the') 777
('going', 'to') 754
('to', 'be') 723
('to', 'the') 665
('on', 'the') 633
('is', 'a') 596
```

In [55]:

```
print("Web-Crawled")
for i in range(10):
    print(top_web[i], web_bigrams[top_web[i]])
```

```
Web-Crawled
('of', 'the') 2496
('in', 'the') 1624
('to', 'the') 1044
('on', 'the') 759
('and', 'the') 631
('for', 'the') 585
('at', 'the') 534
('to', 'be') 521
('with', 'the') 422
('it', 'is') 417
```

Questions

(1) Can you find the top trigrams for each corpus?

(2) Can you compare the frequencies across corpora for each n-gram?