Preprocessing:

Normalizing Text

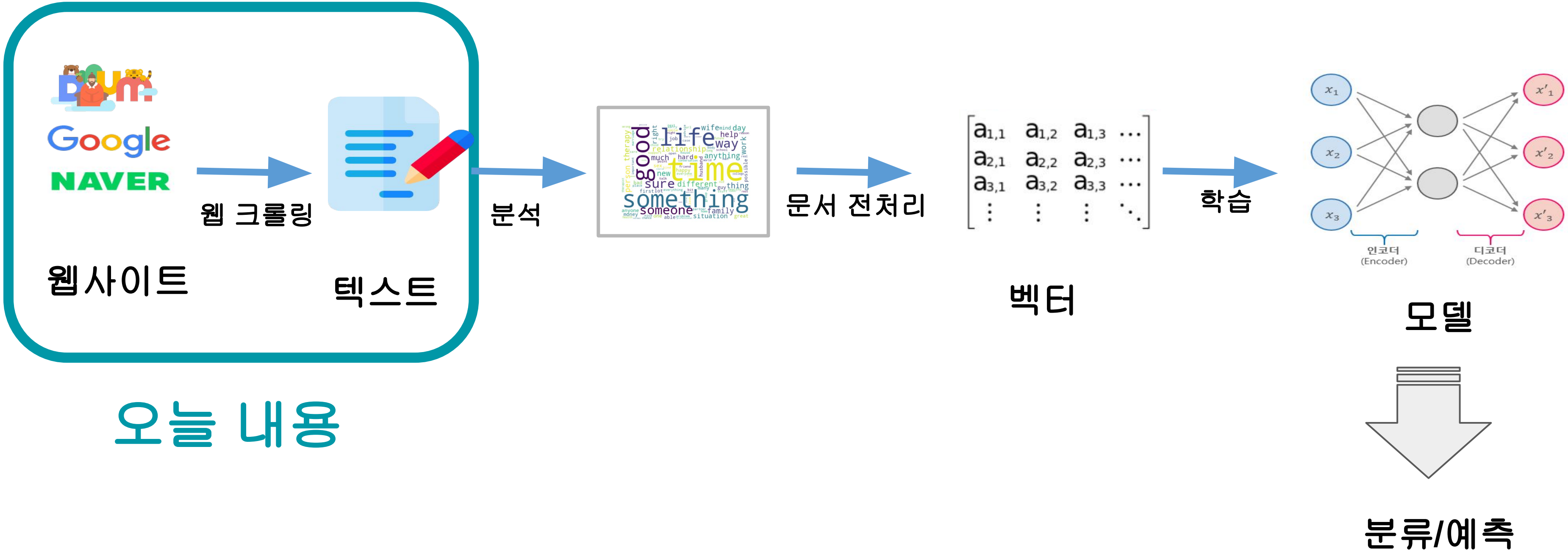
NLTK 3. Processing Raw Text

http://www.nltk.org/book/ch03.html

SLP 2. Regular Expressions, Text Normalization, and Edit Distance

https://web.stanford.edu/~jurafsky/slp3/2.pdf

한눈에 보는 자연어 처리 과정



웹사이트

웹 크롤링

텍스트

오늘 내용

분석

문서 전처리

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

벡터

학습

모델

분류/예측

Outline

- Loading your own text sources from Web, word, and text files
- Split documents up into words
- Converting words into standard forms for analysis

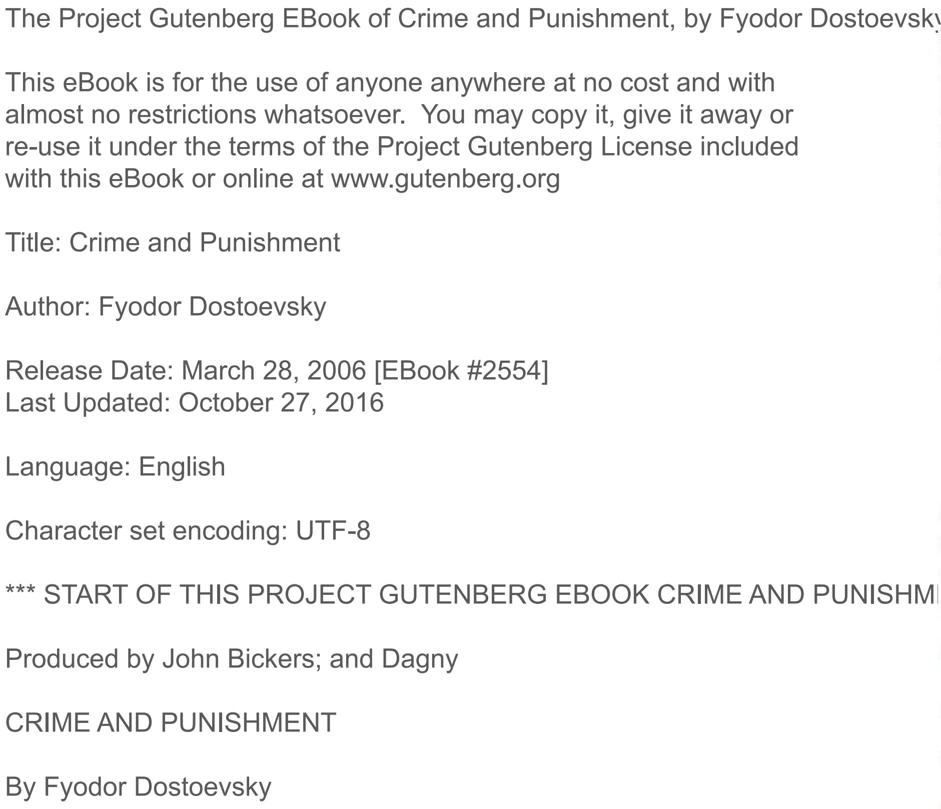
Text from the Web

- NLTK corpus contains some texts from Project Gutenberg
- *raw* contains a string with 1,176,893 characters, including many details we are not interested in such as whitespace

```
>>> from urllib import request
>>> url = "http://www.gutenberg.org/files/2554/2554-0.txt"
>>> response = request.urlopen(url)
>>> raw = response.read().decode('utf8')
>>> type(raw)
<class 'str'>
>>> len(raw)
1176893
>>> raw[:75]
'The Project Gutenberg EBook of Crime and Punishment, by
Fyodor Dostoevsky\r\n'
>>> tokens = word_tokenize(raw)
>>> type(tokens)
<class 'list'>
>>> len(tokens)
254354
>>> tokens[:10]
['The', 'Project', 'Gutenberg', 'EBook', 'of', 'Crime', 'and',
'Punishment', ',', 'by']
```

- Create an NLTK text from this list to carry out all of the other linguistic processing

```
>>> text = nltk.Text(tokens)
>>> text[1024:1062]
['CHAPTER', 'I', 'On', 'an', 'exceptionally', 'hot', 'evening',
'early', 'in', 'July', 'a', 'young', 'man', 'came', 'out', 'of', 'the',
'garret', 'in', 'which', 'he', 'lodged', 'in', 'S.', 'Place', 'and',
'walked', 'slowly',',', '.']
>>> text.collocations()
Katerina Ivanovna; Pyotr Petrovitch; Pulcheria Alexandrovna;
Avdotya Romanovna; Rodion Romanovitch; Marfa Petrovna;
Sofya Semyonovna; old
woman; Project Gutenberg-tm; Porfiry Petrovitch; Amalia
Ivanovna; great deal; Nikodim Fomitch; young man; Ilya
Petrovitch; n't know; Project Gutenberg; Dmitri Prokofitch;
Andrey Semyonovitch; Hay Market
>>> raw.find("PART I")
5338
>>> raw.rfind("End of Project Gutenberg's Crime")
1157743
>>> raw = raw[5338:1157743]
>>> raw.find("PART I")
0
```



The Project Gutenberg EBook of Crime and Punishment, by Fyodor Dostoevsky

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever. You may copy it, give it away or
re-use it under the terms of the Project Gutenberg License included
with this eBook or online at www.gutenberg.org

Title: Crime and Punishment

Author: Fyodor Dostoevsky

Release Date: March 28, 2006 [EBook #2554]
Last Updated: October 27, 2016

Language: English

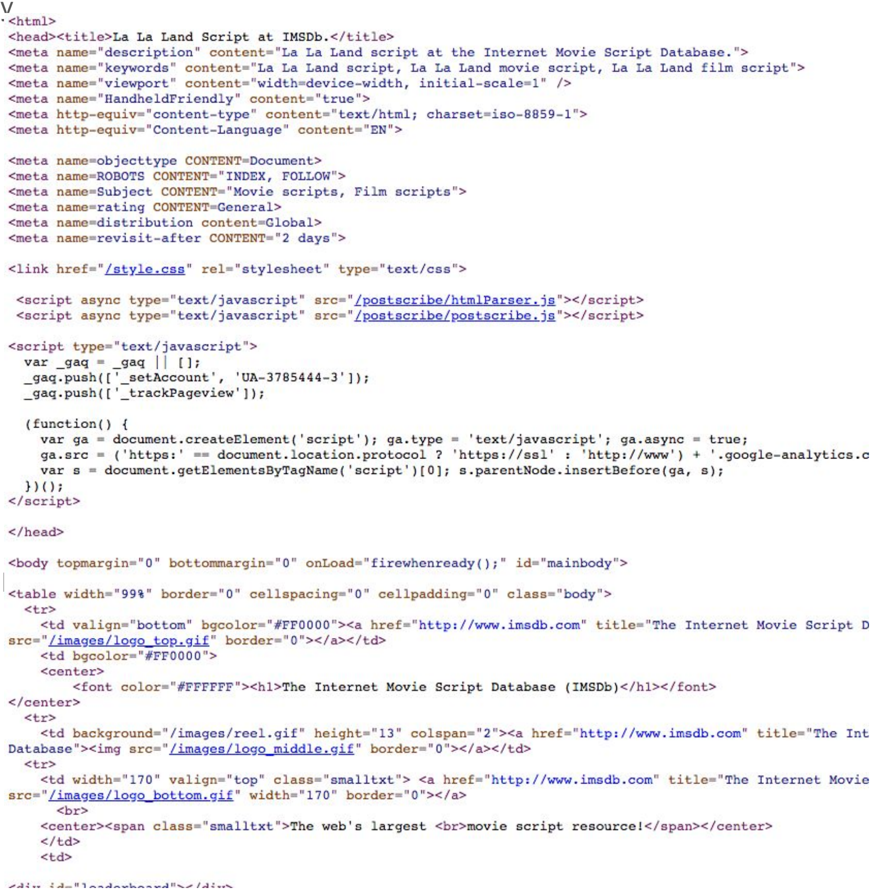
Character set encoding: UTF-8

*** START OF THIS PROJECT GUTENBERG EBOOK CRIME AND PUNISHM

Produced by John Bickers; and Dagny

CRIME AND PUNISHMENT

By Fyodor Dostoevsky



```
<html>
<head><title>La La Land Script at IMSDb.</title>
<meta name="description" content="La La Land script at the Internet Movie Script Database.">
<meta name="keywords" content="La La Land script, La La Land movie script, La La Land film script">
<meta name="viewport" content="width=device-width, initial-scale=1" />
<meta name="HandheldFriendly" content="true">
<meta http-equiv="content-type" content="text/html; charset=iso-8859-1">
<meta http-equiv="Content-Language" content="EN">

<meta name=objecttype CONTENT=Document>
<meta name=ROBOTS CONTENT="INDEX, FOLLOW">
<meta name=Subject CONTENT="Movie scripts, Film scripts">
<meta name=rating CONTENT=General>
<meta name=distribution content=Global>
<meta name=revisit-after CONTENT="2 days">

<link href="/style.css" rel="stylesheet" type="text/css">

 <script async type="text/javascript" src="/postscribe/htmlParser.js"></script>
 <script async type="text/javascript" src="/postscribe/postscribe.js"></script>

<script type="text/javascript">
  var _gaq = _gaq || [];
  _gaq.push(['_setAccount', 'UA-3785444-3']);
  _gaq.push(['_trackPageview']);

  (function() {
    var ga = document.createElement('script'); ga.type = 'text/javascript'; ga.async = true;
    ga.src = ('https:' == document.location.protocol ? 'https://ssl' : 'http://www') + '.google-analytics.c
    var s = document.getElementsByTagName('script')[0]; s.parentNode.insertBefore(ga, s);
  })();
</script>

</head>

<body topmargin="0" bottommargin="0" onLoad="firewhenready();" id="mainbody">

<table width="99%" border="0" cellspacing="0" cellpadding="0" class="body">
  <tr>
    <td valign="bottom" bgcolor="#FF0000"><a href="http://www.imsdb.com" title="The Internet Movie Script D
src="/images/logo_top.gif" border="0"></a></td>
    <td bgcolor="#FF0000">
    <center>
        <font color="#FFFFFF"><h1>The Internet Movie Script Database (IMSDb)</h1></font>
</center>
  <tr>
    <td background="/images/reel.gif" height="13" colspan="2"><a href="http://www.imsdb.com" title="The Int
Database"><img src="/images/logo_middle.gif" border="0"></a></td>
  <tr>
    <td width="170" valign="top" class="smalltxt"> <a href="http://www.imsdb.com" title="The Internet Movie
src="/images/logo_bottom.gif" width="170" border="0"></a>
      <br>
    <center><span class="smalltxt">The web's largest <br>movie script resource!</span></center>
    </td>
    <td>

<div id="leaderboard"></div>
```

Crime and Punishment

La La Land

HTML

- Read BBC News story called *Blondes to die out in 200 years*
- To get text out of HTML, use **BeautifulSoup** at http://www.crummy.com/software/BeautifulSoup/
- Find the start and end indexes of the content

Your turn: La La Land at http://www.imsdb.com/scripts/La-La-Land.html)

```
>>> from nltk import word_tokenize
>>> url = "http://news.bbc.co.uk/2/hi/health/2284783.stm"
>>> html = request.urlopen(url).read().decode('utf8')
>>> from bs4 import BeautifulSoup
>>> raw = BeautifulSoup(html, 'lxml').get_text()
>>> tokens = word_tokenize(raw)
>>> tokens
['BBC', 'NEWS', '|', 'Health', '|', 'Blondes', "'to", 'die', 'out', ...]

>>> tokens = tokens[110:390]
>>> text = nltk.Text(tokens)
>>> text.concordance('gene')
Displaying 5 of 5 matches:
hey say too few people now carry the gene for blondes to
last beyond the next blonde hair is caused by a recessive
gene . In order for a child to have blond have blonde hair , it
must have the gene on both sides of the family in the gere is
a disadvantage of having that gene or by chance . They do
n't disappear des would disappear is if having the gene was
a disadvantage and I do not thin
```

Read from pdf

- PyPDF2 module
- Back to the future script: http://www.imsdb.com/scripts/Back-to-the-Future.pdf

```
from PyPDF2 import PdfFileReader
import os
import nltk
def text_extractor(path):
    with open(path, 'rb') as f:
        pdf = PdfFileReader(f)
        # get the first page
        page = pdf.getPage(1)
        print(page)
        print('Page type: {}'.format(str(type(page))))
        text = page.extractText()
        raw = text
        print(len(text), type(raw))
        tokens = nltk.word_tokenize(raw)
        print(type(tokens), len(tokens))
        text = nltk.Text(tokens)
        print("text length is:",len(text))
        print(raw)
        print(raw.find('Method'))
        print(text)

if __name__ == '__main__':
    path = 'US9152209.pdf'
    #print(os.listdir('.'))
    text_extractor(path)
```

Word file

- https://automatetheboringstuff.com/chapter13/
- pip install python-docx

```
>>> import docx
>>> doc = docx.Document('Synset.docx')
>>> for i in doc.paragraphs:
>>>     print(i.text)

>>> doc = docx.Document()
>>> doc.add_heading('Header 0', 0)
>>> doc.add_heading('Header 1', 1)
>>> doc.add_heading('Header 2', 2)
>>> doc.add_heading('Header 3', 3)
>>> doc.add_heading('Header 4', 4)
>>> doc.save('headings.docx')
```

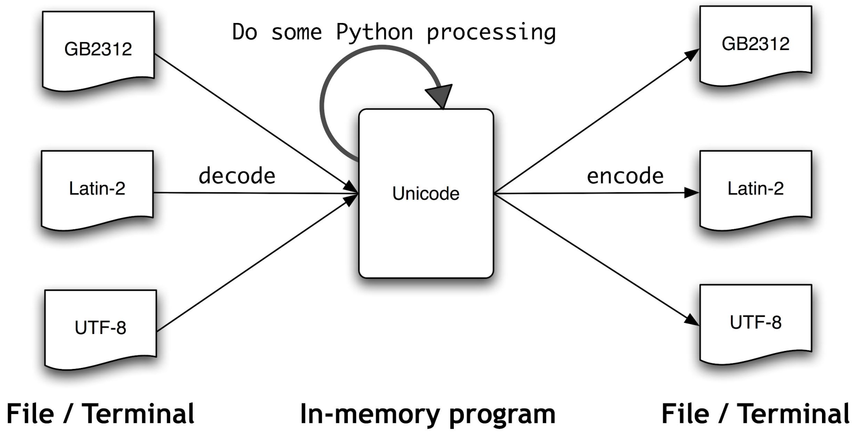
More Formats

- Local files: 표준 라이브러리의 open, read 사용

  -
    ```
    >>> with open('document.txt', 'r' ) as f:
    …       raw = f.read()
    ```
- RSS feeds: feedparser 사용

Unicode

\u 를 앞에 붙여서 표현



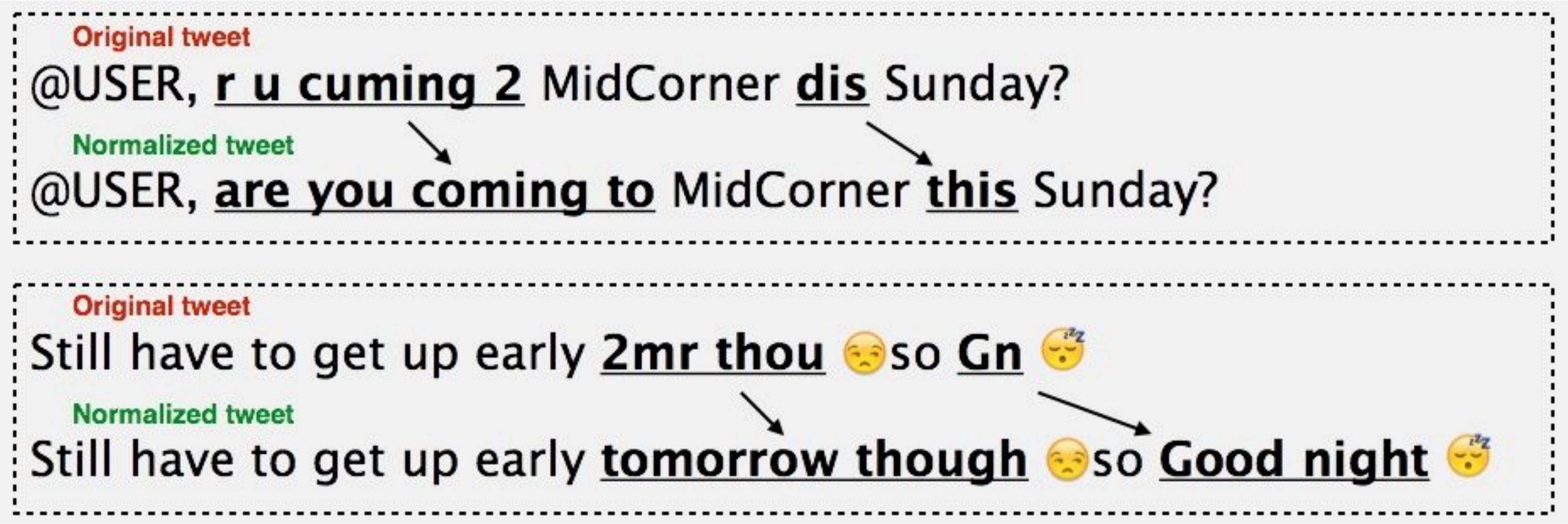
```
path = nltk.data.find('corpora/unicode_samples/polish-lat2.txt')

f = open(path, encoding='latin2')
for line in f:
    line = line.strip()
    print(type(line),line)
    print(line.encode('unicode_escape'))

print(ord('ń'))
nacute = '\u0144'
print(nacute)
print(nacute.encode('utf8'))
print('\xf3')
print('\u00f3')
print('\xc5\x84')
```

Normalizing Text

Converting text to a standard form

1. Segmenting/tokenizing words from running text
2. Normalizing word formats
3. Segmenting sentences in running text

Noisy Text: Tweet



Original tweet

@USER, **r u cuming 2** MidCorner **dis** Sunday?

Normalized tweet

@USER, **are you coming to** MidCorner **this** Sunday?

Original tweet

Still have to get up early **2mr thou** 😒so **Gn** 😴

Normalized tweet

Still have to get up early **tomorrow though** 😒so **Good night** 😴

https://noisy-text.github.io/2015/

Text Normalization Challenges

- Word limit: punctuations (.,’-!), running text, compound nouns
- Contractions: I’ve,
- Numbers: thirty two, 03/01/2001
- Abbreviations: U.S.A., NYC, NASA
- Varied Verb forms: sang, sung, sings, eat, ate, eaten, eats
- Miss-spelling: tokenize, tokenise, sooooo much
- Slang
- Synonyms: car, sedan, truck, SUV
- Named Entity: Anne, Java, Seoul, Bentz, Samsung, Twitter
- Emoticons (^^, :), hashtags (#nlproc), url, email address
- Out-of-Vocabulary (OOV)

Regular Expressions

Source: SLP Ch.2

https://web.stanford.edu/~jurafsky/slp3/slides/2_TextProc.pptx

NLTK 3. Processing Raw Text

Regular Expressions

- A formal language for specifying text strings
- Many linguistic processing tasks involve pattern matching
- How can we search for any of these?
  - woodchuck
  - woodchucks
  - Woodchuck
  - Woodchucks

Regular Expressions: Disjunctions

- Letters inside square brackets []

Pattern	Matches
[wW]oodchuck	Woodchuck, woodchuck
[1234567890]	Any digit

- Ranges [A-Z]

Pattern	Matches	
[A-Z]	An uppercase letter	Drenched Blossoms
[a-z]	A lowercase letter	my beans were impatient
[0-9]	A single digit	Chapter 1: Down the Rabbit Hole

Regular Expressions: Disjunctions

- Woodchucks is another name for groundhog! ⇒ groundhog|woodchuck
- The pipe | for disjunction

Pattern	Matches
yours|mine	yours, mine
a|b|c	= [abc]
[gG]roundhog|[Ww]oodchuck	

Regular Expressions: Negation in disjunction

- Negations [^Ss]
  - Carat means negation only when first in []

Pattern	Matches	
[^A-Z]	Not an uppercase letter	Oyfn pripetchik
[^Ss]	Neither ‘S’ nor ‘s’	I have no exquisite reason”

Regular Expressions: ?   *   +   .

Pattern	Matches	
colou?r	Optional previous char	color colour
oo*h!	0 or more of previous char	oh! ooh! oooh! ooooh!
o+h!	1 or more of previous char	oh! ooh! oooh! ooooh!
beg.n	1 char	begin begun begun beg3n

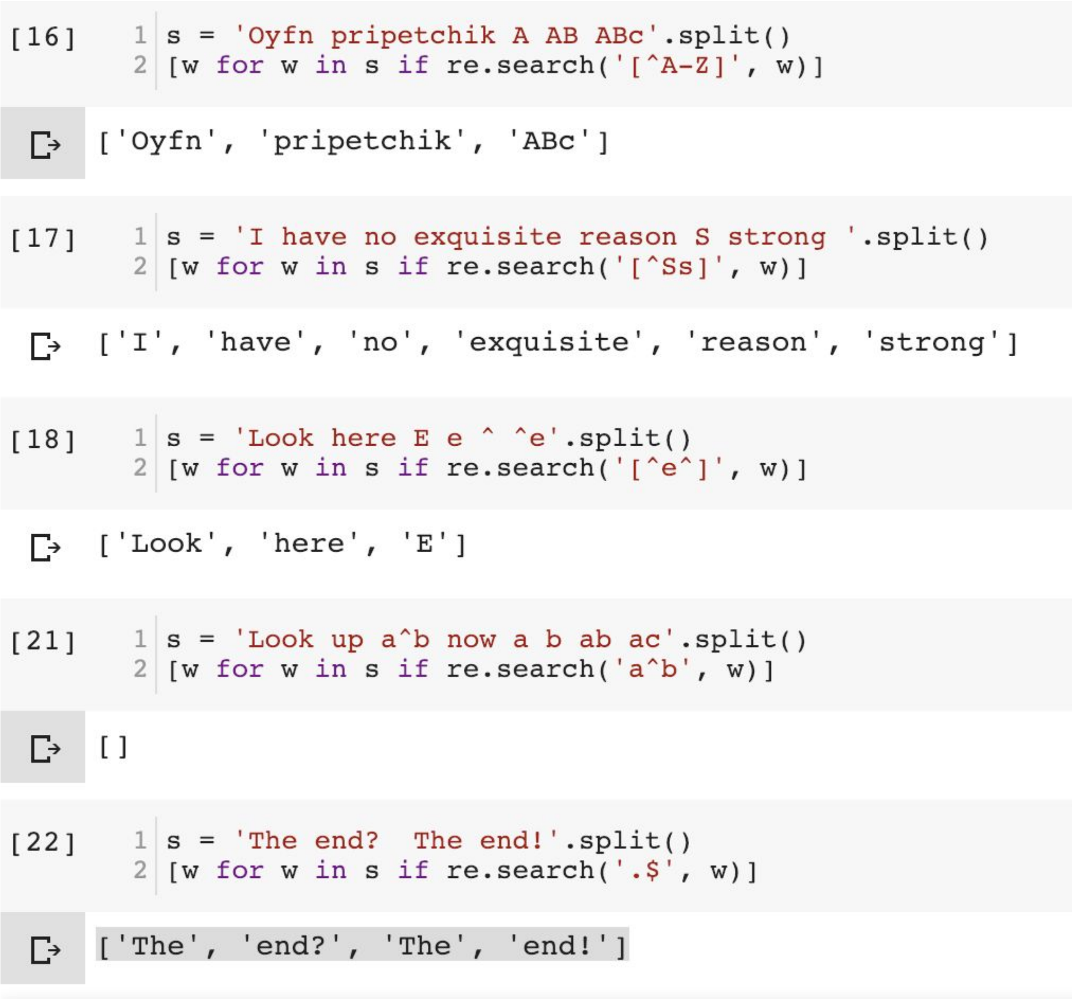
Regular Expressions: Anchors ^ $

Pattern	Matches	
^[A-Z]	at the beginning	Palo Alto
^[^A-Za-z]		1 “Hello”
\.$	. at the end	The end.
.$	any char at the end	The end? The end!

Regular Expressions: NLTK

```
>>> import re
>>> [w for w in wordlist if re.search('ed$', w)] # endswith('ed')
['abaissed', 'abandoned', 'abased', 'abashed', 'abatised', 'abed', 'aborted', ...]
>>> [w for w in wordlist if re.search('^..j..t..$', w)]
['abjectly', 'adjuster', 'dejected', 'dejectly', 'injector', 'majestic', ...]

>>> [w for w in wordlist if re.search('^[ghi][mno][jlk][def]$', w)]
['gold', 'golf', 'hold', 'hole']
```

- Wikipedia: https://en.wikipedia.org/wiki/Regular_expression
- Python: https://docs.python.org/3/library/re.html
- 파이썬 코딩 도장: https://dojang.io/mod/page/view.php?id=1141
- Python Howto: https://docs.python.org/3/howto/regex.html



[16]

```
s = 'Oyfn pripetchik A AB ABc'.split()
[w for w in s if re.search('[^A-Z]', w)]
```

```
['Oyfn', 'pripetchik', 'ABc']
```

[17]

```
s = 'I have no exquisite reason S strong '.split()
[w for w in s if re.search('[^Ss]', w)]
```

```
['I', 'have', 'no', 'exquisite', 'reason', 'strong']
```

[18]

```
s = 'Look here E e ^ ^e'.split()
[w for w in s if re.search('[^e^]', w)]
```

```
['Look', 'here', 'E']
```

[21]

```
s = 'Look up a^b now a b ab ac'.split()
[w for w in s if re.search('a^b', w)]
```

```
[]
```

[22]

```
s = 'The end?  The end!'.split()
[w for w in s if re.search('.$', w)]
```

```
['The', 'end?', 'The', 'end!']
```

Example

Find all instances of the word “the” in a text.

- **the**: Misses capitalized examples
- **[tT]he**: Incorrectly returns o**the**r or **the**ology
- **[^a-zA-Z][tT]he[^a-zA-Z]**

Regular Expression Meta-Characters

Operator	Behavior
.	Wildcard, matches any character
^abc	Matches some pattern *abc* at the start of a string
abc$	Matches some pattern *abc* at the end of a string
[abc]	Matches one of a set of characters
[A-Z0-9]	Matches one of a range of characters
ed|ing|s	Matches one of the specified strings (disjunction)
*	Zero or more of previous item, e.g. a*, [a-z]* (also known as *Kleene Closure*)
+	One or more of previous item, e.g. a+, [a-z]+
?	Zero or one of the previous item (i.e. optional), e.g. a?, [a-z]?
{n}	Exactly *n* repeats where n is a non-negative integer
{n,}	At least *n* repeats
{,n}	No more than *n* repeats
{m,n}	At least *m* and no more than *n* repeats
a(b|c)+	Parentheses that indicate the scope of the operators

Regular Expression symbols

Symbol	Function
\b	Word boundary (zero width)
\d	Any decimal digit (equivalent to [0-9])
\D	Any non-digit character (equivalent to [^0-9])
\s	Any whitespace character (equivalent to [\t\n\r\f\v])
\S	Any non-whitespace character (equivalent to [^ \t\n\r\f\v])
\w	Any alphanumeric character (equivalent to [a-zA-Z0-9_])
\W	Any non-alphanumeric character (equivalent to [^a-zA-Z0-9_])
\t	The tab character
\n	The newline character

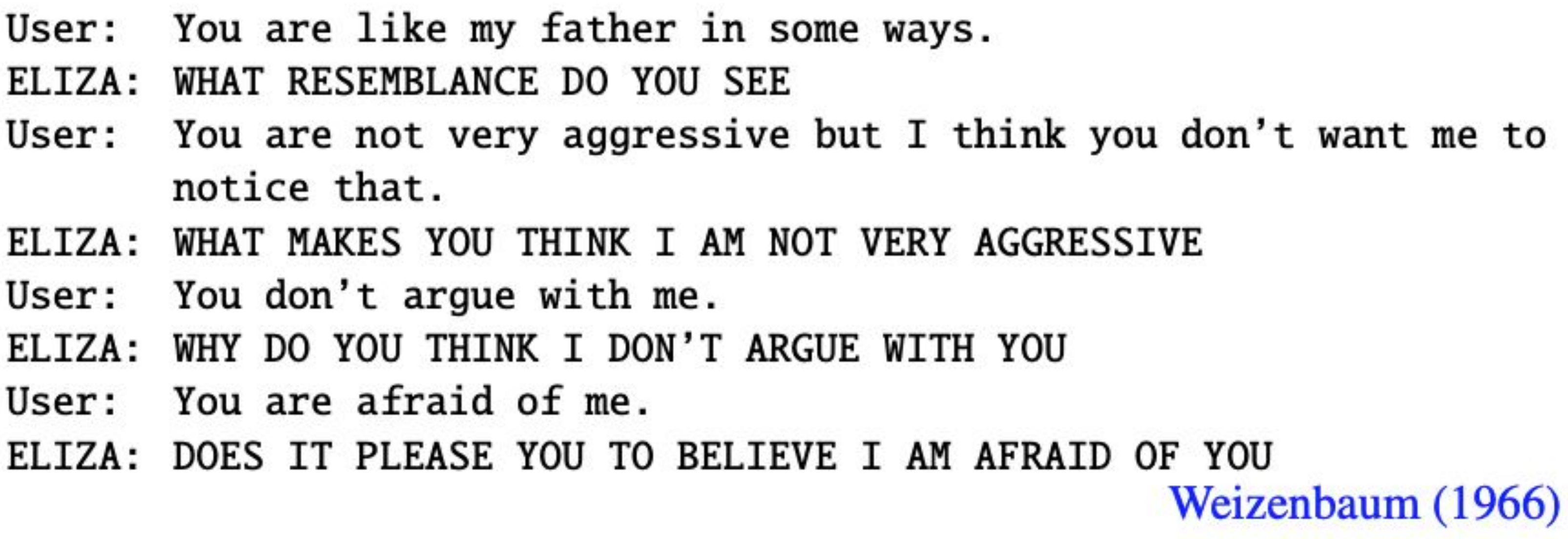
Applications of Regular Expressions: Chatbot

NLTK Chatbot demo

https://www.nltk.org/api/nltk.chat.html

- nltk.chat.eliza.demo()
- nltk.chat.iesha.demo(): tongue-in-cheek take on the average teen anime junky that frequents YahooMessenger or MSNM. All spelling mistakes and flawed grammar are intentional
- nltk.chat.suntsu.demo(): Quoted from Sun Tsu’s The Art of War Translated by LIONEL GILES, M.A. 1910
- nltk.chat.zen.demo()

ELIZA (1966, MIT)



User: You are like my father in some ways.

ELIZA: WHAT RESEMBLANCE DO YOU SEE

User: You are not very aggressive but I think you don't want me to notice that.

ELIZA: WHAT MAKES YOU THINK I AM NOT VERY AGGRESSIVE

User: You don't argue with me.

ELIZA: WHY DO YOU THINK I DON'T ARGUE WITH YOU

User: You are afraid of me.

ELIZA: DOES IT PLEASE YOU TO BELIEVE I AM AFRAID OF YOU

Weizenbaum (1966)

ELIZA

- Source code at

http://www.nltk.org/_modules/nltk/chat/eliza.html

```
from nltk.chat.util import Chat, reflections

pairs = (
  (r'I need (.*)',
  ( "Why do you need %1?",
    "Would it really help you to get %1?",
    "Are you sure you need %1?")),

  (r'Why don\'t you (.*)',
  ( "Do you really think I don't %1?",
    "Perhaps eventually I will %1.",
    "Do you really want me to %1?"))]

eliza_chatbot = Chat(pairs, reflections)

def eliza_chat():
    print("Therapist\n---------")
    print("Talk to the program")
    print('='*72)
    print("Hello.  How are you feeling today?")

    eliza_chatbot.converse()
```

Chatbot using NLTK

```
'''# nltk.chat.util
reflections = {
  "i am"       : "you test",
  "i was"      : "you were",
  "i"          : "you",
  "i'm"        : "you are",
  "i'd"        : "you would",
  "i've"       : "you have",
  "i'll"       : "you will",
  "my"         : "your",
  "you are"    : "I am",
  "you were"   : "I was",
  "you've"     : "I have",
  "you'll"     : "I will",
  "your"       : "my",
  "yours"      : "mine",
  "you"        : "me",
  "me"         : "you"
}'''
```

```
from nltk.chat.util import Chat, reflections
pairs = [
    [
        r'hi',
        ['hello', 'kamusta', 'mabuhay',]
    ],
    [
        r'(.*) (hungry|sleepy)',
        [  "%1 %2"   ]
    ],

    [
        r"My name is (.*)",
        ['hello %1',]
    ]
]

print("Hi how can I help you today?")
chat = Chat(pairs, reflections)
chat.converse()
```

https://medium.com/@allanmeriales/a-simple-chatbot-using-nltk-chat-640456dcdf72

Regular Expressions: Summary

- Regular expressions play a surprisingly large role
  - Sophisticated sequences of regular expressions are often the first model for any text processing text
- For many hard tasks, we use machine learning classifiers
- But regular expressions are used as features in the classifiers
  - Can be very useful in capturing generalizations

Word Normalization

Tokenization: segmenting running text into words
Normalization: putting words/tokens in a standard format

Word Normalization

- Normalize the text to lowercase so that the distinction between *The* and *the* is ignored
    - set(w.**lower**() for w in text)
- We can strip off any affixes, a task known as **stemming**. A further step is to make sure that the resulting form is a known word in a dictionary, a task known as **lemmatization**

```
>>> raw = """DENNIS: Listen, strange women lying in ponds distributing swords
... is no basis for a system of government.  Supreme executive power derives from
... a mandate from the masses, not from some farcical aquatic ceremony."""
>>> tokens = word_tokenize(raw)

[result]
['DENNIS', ':', 'Listen', ',', 'strange', 'women', 'lying', 'in', 'ponds',
'distributing', 'swords', '...', 'is', 'no', 'basis', 'for', 'a', 'system', 'of',
'government', '.', 'Supreme', 'executive', 'power', 'derives', 'from',
'a', 'mandate', 'from', 'the', 'masses', ',', 'not', 'from', 'some',
'farcical', 'aquatic', 'ceremony', '.']
```

Case folding

- Applications like IR: reduce all letters to lower cases
  - Since users tend to use lower case
- Possible exception: upper case in mid-sentence?
  - e.g., General Motors
  - Fed vs. fed
  - SAIL vs. sail
- For sentiment analysis, MT, Information extraction
  - Case is helpful (*US* versus *us* is important)

Removal of Stopwords

- Stopwords are **high-frequency** words that we sometimes want to filter out of a document

Tokens: ['DENNIS', ':', 'Listen', ',', 'strange', 'women', 'lying', **'in'**, 'ponds', 'distributing', 'swords', '...', **'is'**, **'no'**, 'basis', **'for'**, **'a'**, 'system', **'of'**, 'government', '.', 'Supreme', 'executive', 'power', 'derives', **'from'**, **'a'**, 'mandate', **'from'**, **'the'**, 'masses', ',', **'not'**, **'from'**, **'some'**, 'farcical', 'aquatic', 'ceremony', '.']

Tokens without stopwords: ['DENNIS', ':', 'Listen', ',', 'strange', 'women', 'lying', 'ponds', 'distributing', 'swords', '...', 'basis', 'system', 'government', '.', 'Supreme', 'executive', 'power', 'derives', 'mandate', 'masses', ',', 'farcical', 'aquatic', 'ceremony', '.']

```
>>> from nltk.corpus import stopwords
>>> stopwords.words('english')
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',
'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself',
'she', 'her', 'hers', 'herself', 'it', 'its', 'itself', 'they', 'them', 'their',
'theirs', 'themselves']

>>> raw = """DENNIS: Listen, strange women lying in ponds distributing swords
... is no basis for a system of government.  Supreme executive power derives from
... a mandate from the masses, not from some farcical aquatic ceremony."""
>>> tokens = word_tokenize(raw)
>>> tokens = [w for w in tokens if w not in stopwords.words(‘english’) ]
['DENNIS', ':', 'Listen', ',', 'strange', 'women', 'lying', 'ponds',
'distributing', 'swords', '...', 'basis', 'system', 'government', '.', 'Supreme',
'executive', 'power', 'derives', 'mandate', 'masses', ',', 'farcical', 'aquatic',
'ceremony', '.']
```

Correcting Repeating Characters

- Remove repeated characters

```
[old]:My [new]:My
[old]:schooool [new]:schoool
[old]:schoool [new]:school
[old]:realllllyyy [new]:realllllyy
[old]:realllllyy [new]:realllllly
[old]:realllllly [new]:realllly
[old]:realllly [new]:reallly
[old]:reallly [new]:really
[old]:amaaazingggg [new]:amaaazinggg
[old]:amaaazinggg [new]:amaaazingg
[old]:amaaazingg [new]:amaaazing
[old]:amaaazing [new]:amaazing
[old]:amaazing [new]:amazing
['My', 'school', 'is', 'really', 'amazing']
```

```
from nltk.corpus import wordnet
sample_sentence = 'My schooool is realllllyyy amaaazingggg'
sample_sentence_tokens = nltk.word_tokenize(sample_sentence)

def remove_repeated_characters(tokens):
    repeat_pattern = re.compile(r'(\w*)(\w)\2(\w*)') # characters occur twice
    match_substitution = r'\1\2\3'
    def replace(old_word):
        if wordnet.synsets(old_word): # check if old_word is valid
            return old_word
        new_word = repeat_pattern.sub(match_substitution, old_word)
        print('[old]:'+old_word, '[new]:'+ new_word)
        return replace(new_word) if new_word != old_word else new_word

    correct_tokens = [replace(word) for word in tokens]
    return correct_tokens
print (remove_repeated_characters(sample_sentence_tokens))

[Results]
['My', 'school', 'is', 'really', 'amazing']
```

Stemming

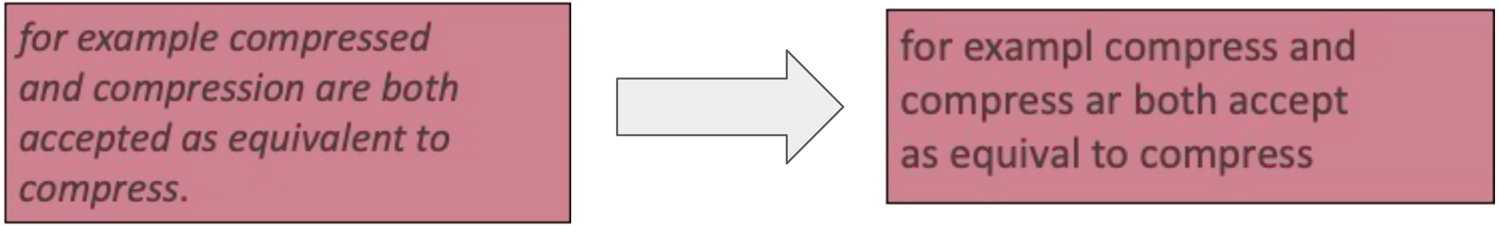
- Reduce terms to their stems in information retrieval
- Stemming is crude chopping of affixes
    - language dependent
    - e.g., *automat**e(s)***, *automat**ic***, *automat**ion*** all reduced to ***automat***.

adding “additional” meanings of various kinds

affixes

stem

the central morpheme of the word, supplying the main meaning



for example compressed and compression are both accepted as equivalent to compress.

for exampl compress and compress ar both accept as equival to compress

Regular Expressions for Stemming

- We want to ignore word endings, and just deal with word **stems**
- re.**findall**() finds all (non-overlapping) matches of the given regular expression

```
def stem(word):
    regexp = r'^(.*?)(ing|ly|ed|ious|ies|ive|es|s|ment)?$'
    stem, suffix = re.findall(regexp, word)[0]
    return stem

>>> raw = """DENNIS: Listen, strange women lying in ponds
distributing swords... is no basis for a system of government.
Supreme executive power derives from
... a mandate from the masses, not from some farcical
aquatic ceremony."""
>>> tokens = word_tokenize(raw)
>>> [stem(t) for t in tokens]
['DENNIS', ':', 'Listen', ',', 'strange', 'women', 'ly', 'in', 'pond',
'distribut', 'sword', 'i', 'no', 'basi', 'for', 'a', 'system', 'of', 'govern',
'.', 'Supreme', 'execut', 'power', 'deriv', 'from', 'a', 'mandate',
'from', 'the', 'mass', ',', 'not', 'from', 'some', 'farcical', 'aquatic',
'ceremony', '.']
```

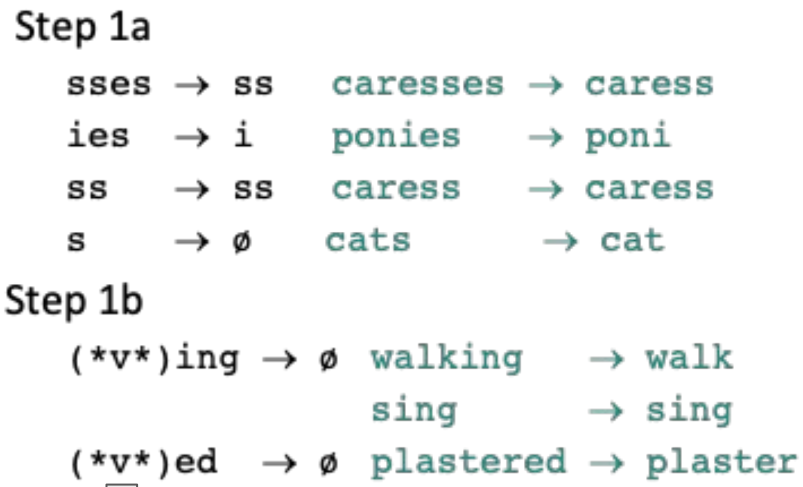
Stemmers

- Stripp affixes
- NLTK includes several off-the-shelf stemmers
- **Porter** stemmer correctly handles the word lying (mapping it to lie), while the Lancaster stemmer does not.

https://tartarus.org/martin/PorterStemmer/

```
>>> porter = nltk.PorterStemmer()
>>> porter.stem(‘presumably’)
u’presum’
>>> lancaster = nltk.LancasterStemmer()
>>> [porter.stem(t) for t in tokens]
['DENNI', ':', 'Listen', ',', 'strang', 'women', 'lie', 'in', 'pond',
'distribut', 'sword', 'is', 'no', 'basi', 'for', 'a', 'system', 'of',
'govern', '.', 'Suprem', 'execut', 'power', 'deriv', 'from', 'a',
'mandat', 'from', 'the', 'mass', ',', 'not', 'from', 'some',
'farcic', 'aquat', 'ceremoni', '.']
>>> [lancaster.stem(t) for t in tokens]
['den', ':', 'list', ',', 'strange', 'wom', 'lying', 'in', 'pond',
'distribut', 'sword', 'is', 'no', 'bas', 'for', 'a', 'system', 'of',
'govern', '.', 'suprem', 'execut', 'pow', 'der', 'from', 'a',
'mand', 'from', 'the', 'mass', ',', 'not',
'from', 'som', 'farc', 'aqu', 'ceremony', '.']
```

Porter's algorithm: The most common English stemmer



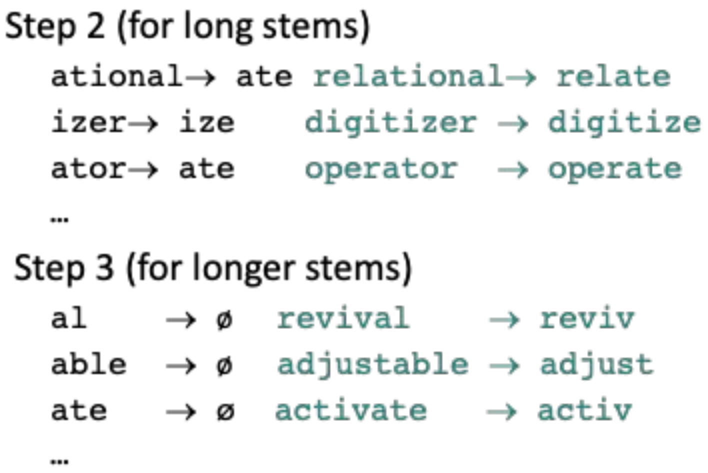
Step 1a

sses → ss	caresses → caress
ies → i	ponies → poni
ss → ss	caress → caress
s → ø	cats → cat

Step 1b

(*v*)ing → ø	walking → walk
	sing → sing
(*v*)ed → ø	plastered → plaster

if stem contains vowel



Step 2 (for long stems)

ational→ ate	relational→ relate
izer→ ize	digitizer → digitize
ator→ ate	operator → operate

…

Step 3 (for longer stems)

al → ø	revival → reviv
able → ø	adjustable → adjust
ate → ø	activate → activ

…

Lemmatization

- Reduce inflections or variant forms to base form which are found in dictionary
    - am, are, is → be
    - car, cars, car's, cars' → car
    - *the boy's cars are different colors → the boy car be different color*

Lemmatization

- WordNet lemmatizer only removes affixes if **the resulting word is in its dictionary**
- This makes the lemmatizer **slower** than stemmers
    - women -> woman
- Note that the lemmatize method default pos argument is “n” which means noun for *is*
- You need to specify the pos for the word as v

```
>>> wnl = nltk.WordNetLemmatizer()
>>> [wnl.lemmatize(t) for t in tokens]
['DENNIS', ':', 'Listen', ',', 'strange', 'woman', 'lying', 'in',
'pond','distributing', 'sword', 'is', 'no', 'basis', 'for', 'a',
'system', 'of', 'government', '.', 'Supreme', 'executive',
'power', 'derives', 'from', 'a', 'mandate', 'from', 'the',
'mass', ',', 'not', 'from', 'some', 'farcical',
'aquatic', 'ceremony', '.']


>>> wordnet_lemmatizer.lemmatize(‘is’)
‘is’
>>> wordnet_lemmatizer.lemmatize(‘is’, pos=’v’)
’be’
```

Out-Of-Vocabulary words

- These normalization methods help deal with the problem of unknown words (OOV), words that a system has not seen before
- A solution to this problem is to use a different kind of tokenization in which most tokens are words, but some tokens are frequent word parts like *-er*, so that an unseen word can be represented by combining the parts: Byte-Pair Encoding

Segmentation

- Tokenization is an instance of a more general problem of segmentation

Sentence Segmentation

- **sent_tokenize** uses an instance of PunktSentenceTokenizer (Kiss & Strunk, 2006)
- PunktSentenceTokenizer instance has been **trained on** 17 European languages
- So it knows what punctuation and characters mark the end of a sentence and the beginning of a new sentence
- the quoted speech contains several sentences, and these have been split into individual strings
- Sentence segmentation is difficult
    - e.g., U.S.A.

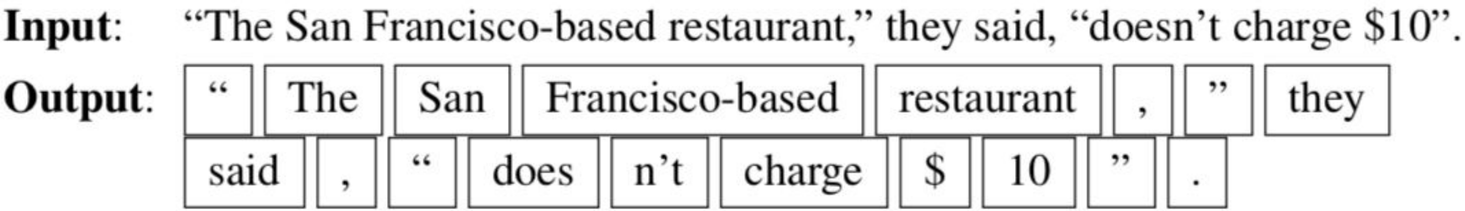
```
>>> text = nltk.corpus.gutenberg.raw('chesterton-thursday.txt')
>>> sents = nltk.sent_tokenize(text)
>>> print(sents[79:89])
['"Nonsense!"', 'said Gregory, who was very rational when anyone
else\n attempted paradox.', '"Why do all the clerks and navvies in
the\n railway trains look so sad and tired, so very sad and tired?',
'I will\n tell you.', 'It is because they know that the train is going
right.', 'It\n is because they know that whatever place they have
taken a ticket\n for that place they will reach.', 'It is because after
they have\n passed Sloane Square they know that the next
station must be\n Victoria, and nothing but Victoria.', 'Oh, their
wild rapture!', 'oh,\n their eyes like stars and their souls again in
Eden, if the next\n station were unaccountably Baker Street!"', '"It
is you who are unpoetical," replied the poet Syme.']
```

Word Tokenization

- **nltk.tokenize.word_tokenize()** splits text into words
- word_tokenize is a wrapper function that calls tokenize by the **TreebankWordTokenizer**, here is the code in NLTK:

```
# Standard word tokenizer.
_word_tokenize = TreebankWordTokenizer().tokenize
def word_tokenize(text):
    return _word_tokenize(text)
```

```
>>> from nltk.tokenize import word_tokenize
>>> word_tokenize(‘Hello World.’)
[‘Hello’, ‘World’, ‘.’]
>>> word_tokenize(“this’s a test”)
[‘this’, “‘s”, ‘a’, ‘test’]

>>> from nltk.tokenize import TreebankWordTokenizer
>>> tokenizer = TreebankWordTokenizer()
>>> tokenizer.tokenize(“this’s a test”)
[‘this’, “‘s”, ‘a’, ‘test’]
```



Input: “The San Francisco-based restaurant,” they said, “doesn’t charge $10”.

Output: “ | The | San | Francisco-based | restaurant | , | ” | they | said | , | “ | does | n’t | charge | $ | 10 | ” | .

https://textminingonline.com/dive-into-nltk-part-ii-sentence-tokenize-and-word-tokenize

Word Tokenizer: others

- **PunktTokenizer** splits on punctuation, but keeps it with the word
- **WordPunctTokenizer** splits all punctuations into separate tokens

```
>>> from nltk.tokenize import PunktWordTokenizer
>>> punkt_word_tokenizer = PunktWordTokenizer()
>>> punkt_word_tokenizer.tokenize(“this’s a test”)
[‘this’, “‘s”, ‘a’, ‘test’]


>>> from nltk.tokenize import WordPunctTokenizer
>>> word_punct_tokenizer = WordPunctTokenizer()
>>> word_punct_tokenizer.tokenize(“This’s a test”)
[‘This’, “‘”, ‘s’, ‘a’, ‘test’]
```

Word Tokenizer: **TweetTokenizer**

- Twitter-aware tokenizer is flexible and easy to adapt to new domains and tasks
- https://www.nltk.org/api/nltk.tokenize.html

```
>>> from nltk.tokenize import TweetTokenizer
>>> tknzr = TweetTokenizer()
>>> s0 = "This is a cooool #dummysmiley: :-) :-P <3 and some arrows < > -> <--"
>>> tknzr.tokenize(s0)
['This', 'is', 'a', 'cooool', '#dummysmiley', ':', ':-)', ':-P', '<3', 'and', 'some', 'arrows', '<', '>', '->', '<--']

>>> tknzr = TweetTokenizer(strip_handles=True, reduce_len=True)
>>> s1 = '@remy: This is waaaaayyyy too much for you!!!!!!'
>>> tknzr.tokenize(s1)
[':', 'This', 'is', 'waaayyy', 'too', 'much', 'for', 'you', '!', '!', '!']
```

Data Sources

- Gutenberg
    - 저작권이 만료된 소설, plain text
    - 죄와 벌 (http://www.gutenberg.org/files/2554/2554-0.txt)
- IMSDB
    - 영화 스크립트 저장 사이트, html
    - Lala Land: http://www.imsdb.com/scripts/La-La-Land.html
- Public datasets
    - Kaggle: https://www.kaggle.com/
    - UCI machine learning repository: https://archive.ics.uci.edu/ml/index.php
    - Reddit: https://www.reddit.com/r/datasets/
    - Github that contains public datasets: https://github.com/awesomedata/awesome-public-datasets
    - 채팅 데이터: Stanford https://nlp.stanford.edu/blog/a-new-multi-turn-multi-domain-task-oriented-dialogue-dataset/
    - Google advanced search https://www.google.com/advanced_search

Summary

- Text as a list of words
- Regular expression is a powerful tool for pattern-matching
    - **concatenation** of symbols, **disjunction** of symbols ([], |, and .), **counters** (*, +, and {n,m}), **anchors** (ˆ, $) and **precedence** operators ((,))
- Basic text normalization
    - **Word segmentation** and **normalization** generally done by cascades of simple regular expressions substitutions or finite automata
    - **Sentence segmentation**
    - **Stemming** strips off affixes (E.g., Porter algorithm)
    - **Lemmatization** maps the various forms of a word to its canonical form