COMP348 — Document Processing and the Semantic Web

Week 01 Lecture 2: Text Processing in Python

Diego Mollá

Department of Computer Science
Macquarie University

COMP348 2018H1

Programme

1. A Review of Python
   - Practicalities
   - Basic Python
   - Simple Statistics in NLTK
2. Text Processing with Python
   - Sorting
   - String Handling
   - Text Preprocessing with NLTK
3. Regular Expressions

Reading

- NLTK Chapter 1

Additional Reading

- http://docs.python.org/3/tutorial/index.html
- http://docs.python.org/3/howto/regex.html

Programme

Programme

Why Python

Scripting Language

- Rapid prototyping.
- Platform neutral.

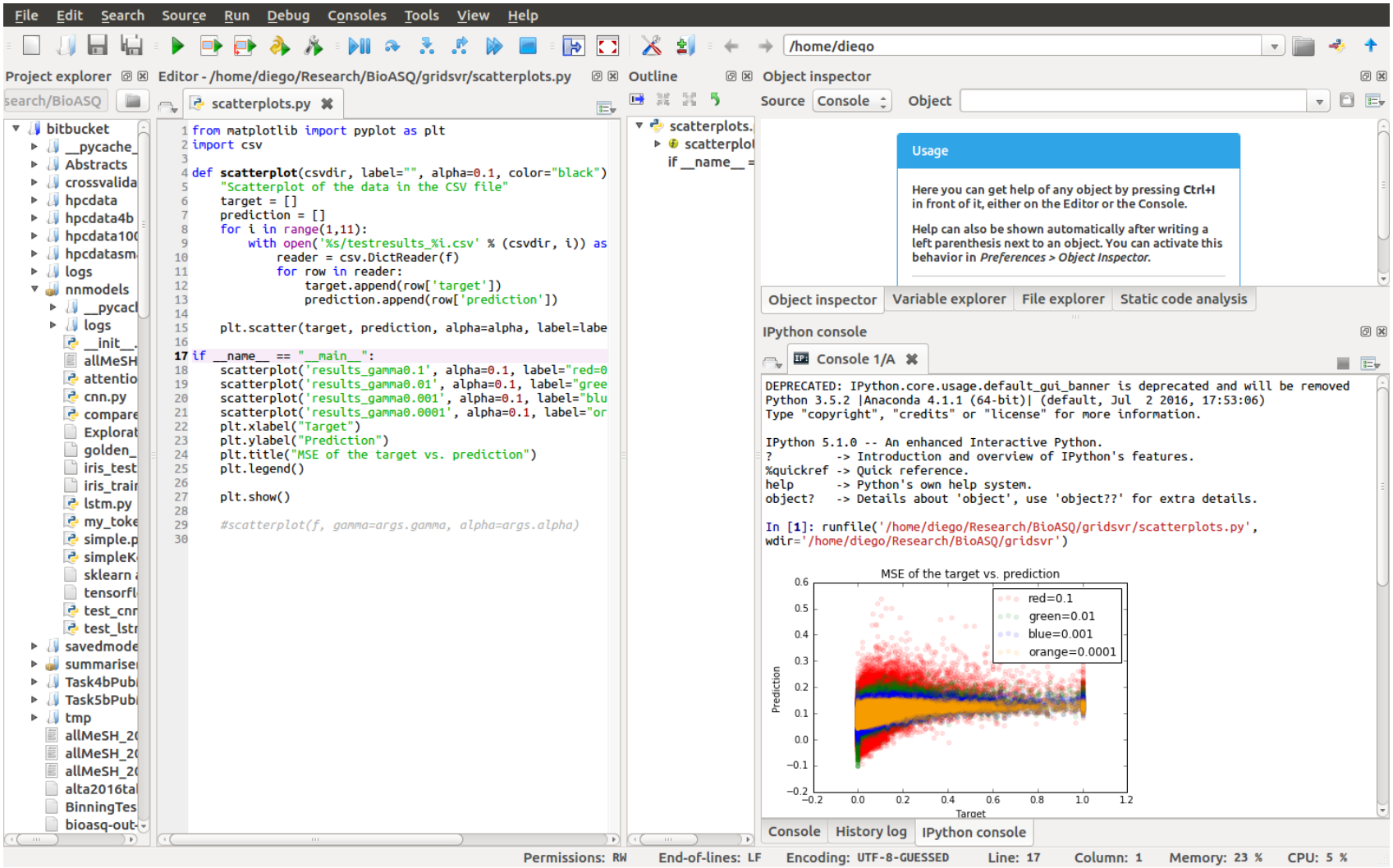
Python

- Even easier prototyping.
  - jupyter notebooks.
- Clean, object oriented.
- Good text manipulation.
- Wide range of libraries.
  - NLTK for NL algorithms.
  - pandas, sklearn, tensorflow for data mining.
  - NumPy and SciPy for scientific computing.
  - matplotlib and pyplot for plotting.

Installing Python

- Official Python at http://www.python.org.
- We will use the Anaconda Python environment from https://www.continuum.io/downloads.
- Current version is 3.x do not use 2.x.
- Windows/Mac/Linux versions.
- Download includes many libraries.
  - cgi.
  - email.
  - HTMLParser.
- Anaconda includes Jupyter notebooks and Spyder, a useful IDE.

Spyder


File Edit Search Source Run Debug Consoles Tools View Help
/home/diego
Project explorer
Editor - /home/diego/Research/BioASQ/gridsvr/scatterplots.py
Outline
Object inspector
scatterplots.py
Source Console Object
bitbucket
nnmodels
from matplotlib import pyplot as plt
import csv
def scatterplot(csvdir, label="", alpha=0.1, color="black")
"Scatterplot of the data in the CSV file"
target = []
prediction = []
for i in range(1,11):
with open('%s/testresults_%i.csv' % (csvdir, i)) as
reader = csv.DictReader(f)
for row in reader:
target.append(row['target'])
prediction.append(row['prediction'])
plt.scatter(target, prediction, alpha=alpha, label=labe
if __name__ == "__main__":
scatterplot('results_gamma0.1', alpha=0.1, label="red=0
scatterplot('results_gamma0.01', alpha=0.1, label="gree
scatterplot('results_gamma0.001', alpha=0.1, label="blu
scatterplot('results_gamma0.0001', alpha=0.1, label="or
plt.xlabel("Target")
plt.ylabel("Prediction")
plt.title("MSE of the target vs. prediction")
plt.legend()
plt.show()
#scatterplot(f, gamma=args.gamma, alpha=args.alpha)
Usage
Here you can get help of any object by pressing Ctrl+I in front of it, either on the Editor or the Console.
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in Preferences > Object Inspector.
Object inspector
Variable explorer
File explorer
Static code analysis
IPython console
Console 1/A
DEPRECATED: IPython.core.usage.default_gui_banner is deprecated and will be removed
Python 3.5.2 |Anaconda 4.1.1 (64-bit)| (default, Jul 2 2016, 17:53:06)
Type "copyright", "credits" or "license" for more information.
IPython 5.1.0 -- An enhanced Interactive Python.
? -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.
In [1]: runfile('/home/diego/Research/BioASQ/gridsvr/scatterplots.py',
wdir='/home/diego/Research/BioASQ/gridsvr')
MSE of the target vs. prediction
red=0.1
green=0.01
blue=0.001
orange=0.0001
Prediction
Target
Console
History log
IPython console
Permissions: RW
End-of-lines: LF
Encoding: UTF-8-GUESSED
Line: 17
Column: 1
Memory: 23 %
CPU: 5 %

Jupyter Notebooks

Scatterplots

localhost:8889/notebooks/Scatterplots.ipynb

jupyter Scatterplots Last Checkpoint: an hour ago (autosaved)

File Edit View Insert Cell Kernel Help Python [Root]

Code CellToolbar

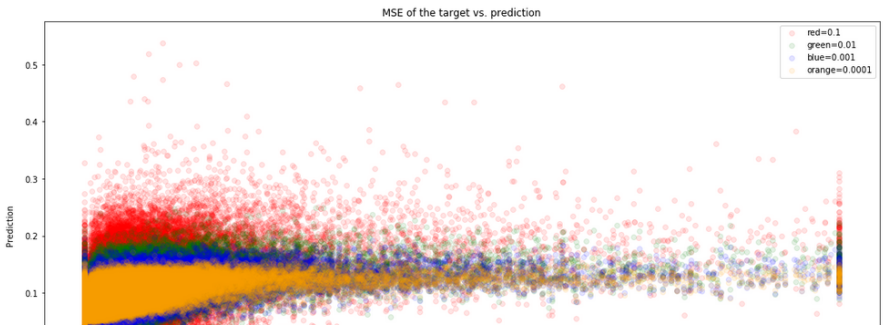
Using full training data

In [3]:
```
def scatterplot(csvdir, label="", alpha=0.1, color="black"):
    "Scatterplot of the data in the CSV file"
    target = []
    prediction = []
    for i in range(1,11):
        with open('%s/testresults_%i.csv' % (csvdir, i)) as f:
            reader = csv.DictReader(f)
            for row in reader:
                target.append(row['target'])
                prediction.append(row['prediction'])

    plt.scatter(target, prediction, alpha=alpha, label=label, color=color)
```

In [21]:
```
plt.rcParams['figure.figsize'] = (18,9)
scatterplot('results_gamma0.1', alpha=0.1, label="red=0.1", color="red")
scatterplot('results_gamma0.01', alpha=0.1, label="green=0.01", color="green")
scatterplot('results_gamma0.001', alpha=0.1, label="blue=0.001", color="blue")
scatterplot('results_gamma0.0001', alpha=0.1, label="orange=0.0001", color="orange")
plt.xlabel("Target")
plt.ylabel("Prediction")
plt.title("MSE of the target vs. prediction")
plt.legend()
```

Out[21]: <matplotlib.legend.Legend at 0x9d31128>



NLTK

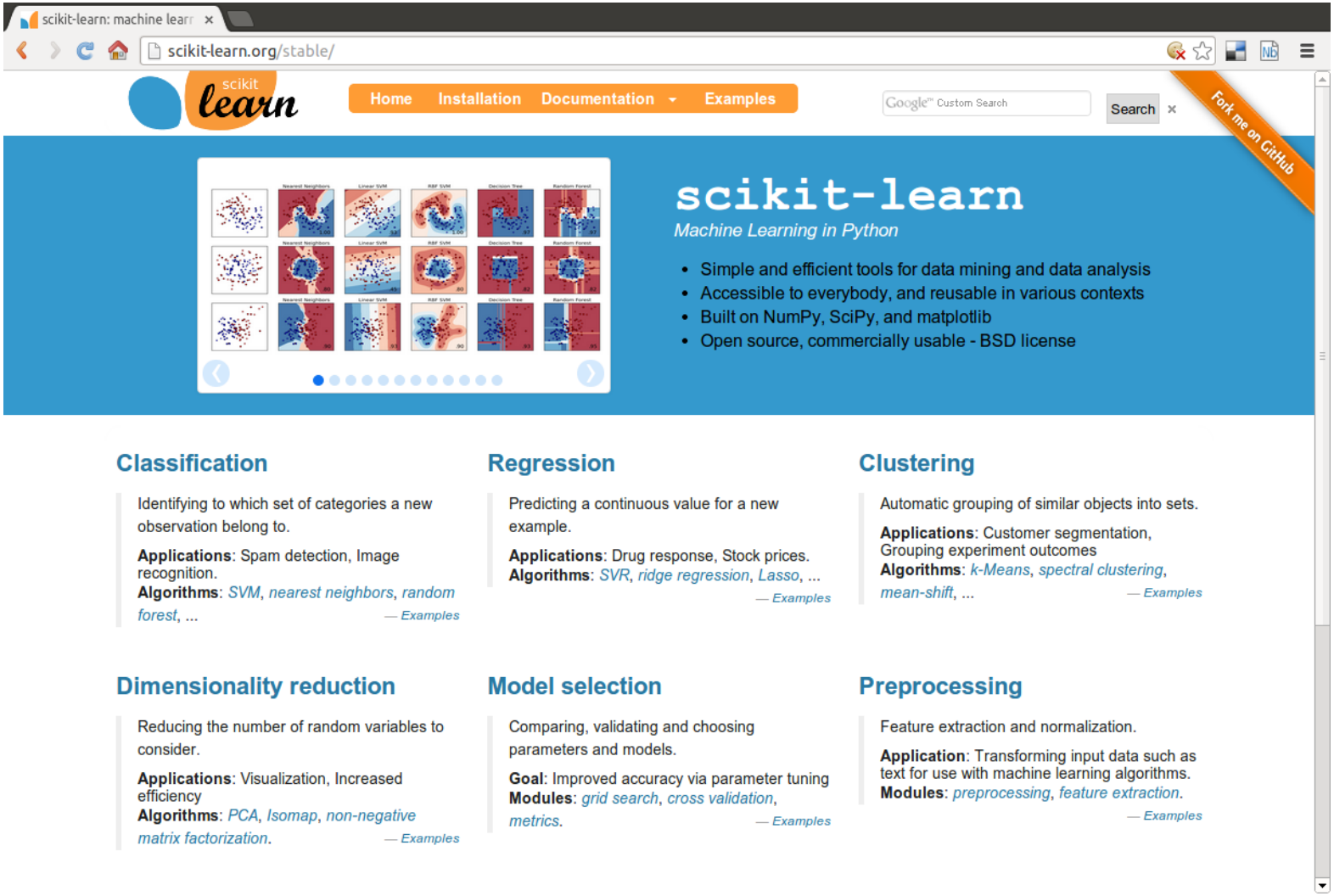
What is NLTK?

- Natural Language Toolkit.
- http://www.nltk.org.
- A collection of Python 3 libraries.

Installing NLTK

- http://www.nltk.org/install.html.
- Pre-installed in Anaconda.

Scikit-learn



Programme

Beginning Python

This and other Python code available as Jupyter notebooks in github: https://github.com/dmollaaliod/comp348.

```
def hello (who):                                    # 1
    """Greet somebody"""                            # 2
    print("Hello " + who + "!")                     # 3

hello("Steve")                                      # 4
hello('World')                                      # 5
people = ['Steve', "Mark", 'Diego']                 # 6
for person in people:                               # 7
    hello(person)                                   # 8
```

Core Data Types

- Strings.
- Numbers (integers, float, complex).
- Lists.
- Tuples (immutable sequences).
- Dictionaries (associative arrays).

Lists I

```
>>> a = ['one', 'two', 3, 'four']
>>> a[0]
'one'
>>> a[-1]
'four'
>>> a[0:3]
['one', 'two', 3]}
>>> len(a)
4
>>> a[1]=2
>>> a
['one', 2, 3, 'four']
>>> a.append('five')
>>> a
['one', 2, 3, 'four', 'five']
```

Lists II

```
>>> top = a.pop()
>>> a
['one', 2, 3, 'four']
>>> top
'five'
```

List Comprehensions

```
>>> a = ['one', 'two', 'three', 'four']
>>> len(a[0])
3
>>> b = [w for w in a if len(w) > 3]
>>> b
['three', 'four']
>>> c = [[1,'one'],[2,'two'],[3,'three']]
>>> d = [w for [n,w] in c]
>>> d
['one', 'two', 'three']
```

Tuples

- Tuples are a sequence data type like lists but are immutable:
  - Once created, elements cannot be added or modified.
- Create tuples as literals using parentheses:
  a = ('one', 'two', 'three')
- Or from another sequence type:
  a = ['one', 'two', 'three']
  b = **tuple**(a)
- Use tuples as fixed length sequences: memory advantages.

Dictionaries

- Associative array datatype (hash).
- Store values under some hash key.
- Key can be any immutable type: string, number, tuple.

```
>>> names = dict()
>>> names['madonna'] = 'Madonna'
>>> names['john'] = ['Dr.', 'John', 'Marshall']
>>> list(names.keys())
['madonna', 'john']
>>> ages = {'steve':41, 'john':22}
>>> 'john' in ages
True
>>> 41 in ages
False
>>> for k in ages:
...     print(k, ages[k])
steve 41
john 22
```

Organising Source Code: Modules

- In Python, a module is a single source file which defines one or more procedures or classes.
- Load a module with the **import** directive.

```
import mymodule
```

- This loads the file mymodule.py and evaluates its contents.
- By default, all procedures are put into the mymodule namespace, accessed with a dotted notation:
  - mymodule.test() — calls the test() procedure defined in mymodule.py

Modules

- Can import names into global namespace.

```
from mymodule import test, doodle
from mymodule import *
```

- The Python distribution comes with many useful modules.

```
from math import *
x = 20 * log(y)
import webbrowser
webbrowser.open('http://www.python.org')
```

Defining Modules

- A module is a source file containing Python code.
  - Usually class/function definitions.
- First non-comment item can be a docstring for the module.

```
# my python module
"""This is a python module to
do something interesting"""

def foo(x):
    'foo the x'
    print('the foo is ' + str(x))
```

Importing NLTK modules

All NLTK modules are under the nltk namespace.

```
>>> import nltk
>>> for id in nltk.corpus.gutenberg.fileids():
...     print(id)
...
austen-emma.txt
austen-persuasion.txt
austen-sense.txt
bible-kjv.txt
blake-poems.txt
bryant-stories.txt
burgess-busterbrown.txt
carroll-alice.txt
chesterton-ball.txt
chesterton-brown.txt
```

Documentation in Python

- Many Python objects have associated documentation strings.
- Good practice is to use these to document your modules, classes and procedures.
- Docstring can be retrieved as the __doc__ attribute of a module/class/procedure name:

```
def hello (who):
    """Greet somebody"""
    print("Hello " + who + "!")

>>> hello.__doc__
'Greet somebody'
```

- The function **help()** uses the docstring to generate interactive help.

Programme

Counting Words I

```
>>> import nltk
>>> emma = nltk.corpus.gutenberg.words('austen-emma.txt')
>>> len(emma)
192427
>>> emma[:10]
['[', 'Emma', 'by', 'Jane', 'Austen', '1816', ']', 'VOLUME',
>>> import collections
>>> emma_counter = collections.Counter(emma)
>>> emma_counter.most_common(10)
[(',', 11454), ('.', 6928), ('to', 5183), ('the', 4844),
('and', 4672), ('of', 4279), ('I', 3178), ('a', 3004),
('was', 2385), ('her', 2381)]
```

Exercises

1. Find the most frequent word with length of at least 7 characters.
2. Find the words that are longer than 7 characters and occur more than 7 times.

Counting Bigrams

A bigram is a sequence of two words.

```
>>> list(nltk.bigrams([1,2,3,4,5,6]))
[(1, 2), (2, 3), (3, 4), (4, 5), (5, 6)]
>>> list(nltk.bigrams(emma))[:3]
[('[', 'Emma'), ('Emma', 'by'), ('by', 'Jane')]
```

Exercises

1. Find the most frequent bigram.
2. Find the most frequent bigram that begins with "the".

Counting Bigrams

A bigram is a sequence of two words.

```
>>> list(nltk.bigrams([1,2,3,4,5,6]))
[(1, 2), (2, 3), (3, 4), (4, 5), (5, 6)]
>>> list(nltk.bigrams(emma))[:3]
[('[', 'Emma'), ('Emma', 'by'), ('by', 'Jane')]
```

Exercises

1. Find the most frequent bigram.
2. Find the most frequent bigram that begins with "the".

Ngrams

- A bigram is an ngram where n is 2.
- A trigram is an ngram where n is 3.

```
>>> list(nltk.ngrams(emma,4))[:5]
[('[', 'Emma', 'by', 'Jane'),
 ('Emma', 'by', 'Jane', 'Austen'),
 ('by', 'Jane', 'Austen', '1816'),
 ('Jane', 'Austen', '1816', ']'),
 ('Austen', '1816', ']', 'VOLUME')]
```

Programme

Programme

Sorting

- The function **sorted()** returns a sorted copy.
- Sequences can be sorted in place with the sort() method.
- Python 3 does not support sorting of lists with mixed contents.

```
>>> foo = [2,5,9,1,11]
>>> sorted(foo)
[1, 2, 5, 9, 11]
>>> foo
[2, 5, 9, 1, 11]
>>> foo.sort()
>>> foo
[1, 2, 5, 9, 11]
>>> foo2 = [2,5,9,1,'a']
>>> sorted(foo2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unorderable types: str() < int()
```

Example

```
>>> l = ['a','abc','b','c','aa','bb','cc']
>>> sorted(l)
['a', 'aa', 'abc', 'b', 'bb', 'c', 'cc']
>>> sorted(l,key=len)
['a', 'b', 'c', 'aa', 'bb', 'cc', 'abc']
>>> sorted(l,key=len,reverse=True)
['abc', 'aa', 'bb', 'cc', 'a', 'b', 'c']
>>> sorted(l,key=lambda x: -len(x))
['abc', 'aa', 'bb', 'cc', 'a', 'b', 'c']
```

Exercises

You're given data of the following form:

```
namedat = dict()
namedat['mc'] = ('Madonna', 45)
namedat['sc'] = ('Steve', 41)
```

1. How would you print a list ordered by name?

```
('Madonna', 45)
('Steve', 41)
```

2. How would you print out a list ordered by age?

```
('Steve', 41)
('Madonna', 45)
```

Programme

Strings in Python

- String is a base datatype.
- Strings are sequences and can use operations like:
  - foo = "A string"
  - **len**(foo)
  - foo[0]
  - foo[0:3]
  - multifoo = """*A multiline* string"""
- In addition, there are some utility functions in the string module.

Some Useful String Functions

```
>>> "my string".capitalize()
'My string'
>>> "my string".upper()
'MY STRING'
>>> "My String".lower()
'my string'
>>> a = "my string with my other text"
>>> a.count('my')
2
>>> a.find('with')
10
>>> a.find('nothing')
-1
```

Split

- split (sep) is a central string operation.
- It splits a string wherever sep occurs (blank space by default).
- It is either a function in the string module or a method of string objects.

```
>>> foo="one :: two :: three"
>>> foo.split()
['one', '::', 'two', '::', 'three']
>>> foo.split('::')
['one ', ' two ', ' three']
>>> import string
>>> string.split("this is a test")
['this', 'is', 'a', 'test']
```

Join

- Join is another useful function/method in the string module.
- It takes a list and joins the elements using some delimiter.

```
>>> text="this is some text to analyse"
>>> words=text.split()
>>> words.sort()
>>> print(",".join(words))
analyse,is,some,text,this,to
>>> print("".join(words))
analyseissometextthisto
```

Replace

```
def censor(text):
    'replace bad words in a text with XXX'
    badwords = ['poo', 'bottom']
    for b in badwords:
        text = text.replace(b, 'XXX')
    return text
```

Programme

NLTK Packaged Tools

Some NLTK tools that are useful for text pre-processing are:

- word_tokenize(text)
- sent_tokenize(text)
- pos_tag(tokens)
- pos_tag_sents(sentences)
- PorterStemmer()

Some popular text preprocessing tools (available in NLTK and other systems) are described in chapter 2 of Maynard et al (2016), "Natural Language Processing for the Semantic Web".

Sentence and Work Tokenisation with NLTK

- NLTK can split text into sentences and words.
  - Sentence segmentation splits text into a list of sentences.
  - Word tokenisation splits text into a list of words (tokens).
- NLTK's default word tokeniser works best after splitting the text into sentences.

```
In [1]: import nltk

In [2]: text = "This is a sentence. This is another sentence.

In [3]: nltk.sent_tokenize(text)
Out[3]: ['This is a sentence.', 'This is another sentence.']

In [4]: for s in nltk.sent_tokenize(text):
   ...:     for w in nltk.word_tokenize(s):
   ...:         print(w)
   ...:     print()
   ...:
```

Part of Speech Tagging

- Often it is useful to know whether a word is a noun, or an adjective, etc. These are called parts of speech.
- NLTK has a part of speech tagger that tags a list of tokens.
- The default list of parts of speech is fairly detailed but we can set a simplified version (called universal by NLTK).

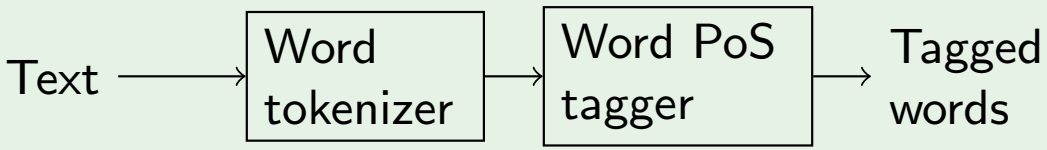
Tag	Meaning	English Examples
ADJ	adjective	new, good, high, special, big, local
ADP	adposition	on, of, at, with, by, into, under
ADV	adverb	really, already, still, early, now
CONJ	conjunction	and, or, but, if, while, although
DET	determiner, article	the, a, some, most, every, no, which
NOUN	noun	year, 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

NLP Pipelines

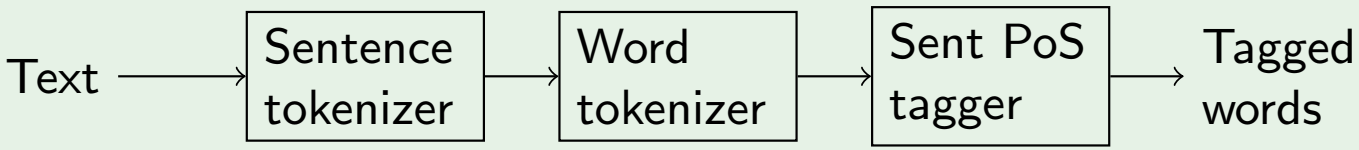
Often, text processing works in pipelines.

- The output of one module is used as the input to the following module.

NLP pipeline for Word PoS tagging



NLP pipeline for Sentence PoS tagging



Note that the above pipelines are different from the sample pipeline in Maynard et al (2016)

Examples in Python

```
In [28]: nltk.pos_tag(["this", "is", "a", "test"])
Out[28]: [('this', 'DT'), ('is', 'VBZ'), ('a', 'DT'), ('test', 'NN')]

In [29]: nltk.pos_tag(["this", "is", "a", "test"], tagset="universal")
Out[29]: [('this', 'DET'), ('is', 'VERB'), ('a', 'DET'), ('test', 'NOUN')]

In [30]: nltk.pos_tag(nltk.word_tokenize("this is a test"), tagset="universal")
Out[30]: [('this', 'DET'), ('is', 'VERB'), ('a', 'DET'), ('test', 'NOUN')]

In [31]: text
Out[31]: 'This is a sentence. This is another sentence.'

In [34]: text_sent_tokens = [nltk.word_tokenize(s) for s in nltk.sent_tokenize(t
    ...: ext)]

In [35]: text_sent_tokens
Out[35]:
[['This', 'is', 'a', 'sentence', '.'],
 ['This', 'is', 'another', 'sentence', '.']]

In [38]: nltk.pos_tag_sents(text_sent_tokens, tagset="universal")
Out[38]:
[[('This', 'DET'),
  ('is', 'VERB'),
  ('a', 'DET'),
  ('sentence', 'NOUN'),
  ('.', '.')],
 [('This', 'DET'),
  ('is', 'VERB'),
```

Stemming

- Often it is useful to remove information such as verb form, or the difference between singular and plural.
- NLTK offers stemming, which removes suffixes.
  - The Porter stemmer is a popular stemmer.
- The remaining stem is not a word but can be used, for example, by search engines (we'll see more of this in another lecture).

Examples of NLTK's Porter Stemmer

```
In [46]: s = nltk.PorterStemmer()

In [47]: s.stem("books")
Out[47]: 'book'

In [48]: s.stem("is")
Out[48]: 'is'

In [50]: s.stem("runs")
Out[50]: 'run'

In [51]: s.stem("running")
Out[51]: 'run'

In [52]: s.stem("run")
Out[52]: 'run'

In [53]: s.stem("goes")
Out[53]: 'goe'
```

Exercises

1. What is the sentence with the largest number of tokens in Austen's "Emma"?
2. What is the most frequent part of speech in Austen's "Emma"?
3. What is the number of distinct stems in Austen's "Emma"?
4. What is the most ambiguous stem in Austen's "Emma"? (meaning, which stem in Austen's "Emma" maps to the largest number of distinct tokens?)

Programme

Regular Expressions

- A RE is a pattern that can be used to match against a string.
- Look at the tutorial http://docs.python.org/3/howto/regex.html.
- REs are quite powerful:
  - patterns can be complicated.
  - you can retrieve bits of the pattern that you match.
- REs implement a simple parser.
  - RE is the 'grammar'.
- But: REs can't solve every text parsing problem.
  - e.g.: can't detect patterns of the form aa ... abb ... b where there is the same number of as and bs.
  - see: http://www.wikipedia.org/wiki/Chomsky_hierarchy

Example

```
import re
def is_there_a_nine (str):
    """return True if there is a 9 in the string"""
    if re.search( '9', str ):
        return True
    else:
        return False

def find_digits (str):
    """return any sequences of digits in str
as a list of strings"""
    return re.findall( '[0-9]+', str )
```

Format of a Regular Expression I

- Most single characters match themselves.
  - except . ^ $ * + ? { [\ | ( )
- [abc] matches any one of a or b or c.
- [a−z] matches any char in the range a to z.
- \ (backslash) gives the next character special meaning.
  - \n — matches a newline.
  - \t — matches a tab.
  - \d — matches a digit.
  - \s — matches any whitespace [\t\n\r\f\v].
  - \w — matches any word character [a−zA−Z0−9_].
  - \W — matches any non-word character.
- e.g. re.split(r'\W', 'Words, words, words.').
- . (period) matches any character except \n.

Format of a Regular Expression II

- (and) group patterns.
- $*$ + ? modify the preceding RE:
  - $*$ means zero or more.
    - [0−9]$*$
    - fo$*$d
  - + means one or more.
    - y(aba)+(daba)+doo
  - ? means zero or one (optional).
    - eggs?
    - one (**or** more)?
- By default, repetition operators are greedy.
  - they try to match as much text as they can.
- Greedy behaviour is cancelled with $*$? and +?.
  - Compare re.findall(r'\(.*\)','a(aa)a(bb)c') vs re.findall(r'\(.*?\)','a(aa)a(bb)c')

Format of a Regular Expression III

- Most general repetition operator is:
  - {n,m} match between n and m times.
    I (really)\{1,3\} like it
- Alternatives:
  - (one|two|three) matches one of these REs.
- ^ matches at the start of lines.
- $ matches at the end of lines.

The Backslash Plague

- Python uses \ as a quote character in strings.
- So the string "foo\d" is really "food".
- To get a real \ in a string need to quote it:
  - "foo\\n"
- To match a literal backslash (eg. match \section).
  - "\\\\section"
- Can use raw strings:
  - r"\\section"
  - r"\w+\s+"

Using Regular Expressions

- Can compile REs for better performance.
  - p = re.**compile**("[0−9]+")
- re module provides various functions for matching REs:
  - match — does the RE match at the start of the string.
  - search — does the match anywhere.
  - split — split using an RE.
  - sub — find matches and replace.
  - subn — as sub but only do so many.
  - findall — find all matches and return them in a list.

Matching Dates

```
def getdate(str):
  """find a date day/month/year in str
return a list of [day, month, year]"""
  p = re.compile(r'([0-9]+)/([0-9]+)/([0-9]+)')
  m = p.search(str)
  if not m == None:
    return [m.group(1), m.group(2), m.group(3)]
```

Modifying Dates

```
def ozifydates(str):
    """map us dates (mm/dd/yy) to Aus.
format (dd/mm/yy)"""
    p = re.compile(r'([0-9]+)/([0-9]+)/([0-9]+)')
    return p.sub(r'\2/\1/\3', str)
```

Modifying Dates

- We'd really like to check the date before replacing it.
- re.sub can take a function instead of a replacement string.
- The function is called with every match and should return the replacement text.
- We could define a function to validate the date and return the Australian version.

Modifying Dates

```
def ozdate(m):
  """return the oz version of matched date"""
  d = m.groups()
  if int(d[0]) > 0 & int(d[0]) <= 12:
    if int(d[1]) > 0 & int(d[1]) <= 31:
      return "/".join([d[1], d[0], d[2]])
  else:
    return "/".join(d)

if __name__ == '__main__':
  import re
  print(re.sub(r'([0-9]+)/([0-9]+)/([0-9]+)',
               ozdate,'9/24/67'))
```

Take-home Messages

- Get to know nltk and the re module
- Read the Regular Expression Howto:
  - http://www.python.org/doc/howto
- We'll use REs a lot for input processing.
- Great for modifying the format of data.

What's Next

Week 2

- Information Retrieval

Additional Reading

- "Introduction to Information Retrieval", http://www-nlp.stanford.edu/IR-book/