Natural Language Processing Lab (2003-)

Jason S. Chang 張俊盛 jason@nlplab.cc

TA：Joanna Wu 吳婞慈 joannawu@nlplab.cc

LK Huang 黃麟凱 hlk@nlplab.cc

Course Website：https://eeclass.nthu.edu.tw/course/info/4137

2021 0916 – 2022 0113 Thur 15:30 Online

Course Description

- Purpose: how to program for doing NLP research
- Tools: Python, Unix
- Syllabus is available online

Course Format

- One task every week mimicking previous work, or presenting potential research opportunity
- Class starts with a 30-60 minute brief of a problem, datasets, method, and steps
- Write a Python program to solve the problem and produce some results
- You are encouraged to discuss with classmates
- Raise your hand (metaphorically) and ask a TA

How do we give grades

- Show a TA your finished work
- Upload your work for the record
- Grades are determined by how accurately, completely, and when you finish your work
- Term Project (during the final 4 weeks)
  - Submit a brief proposal based on one of weekly task
  - You may choose not to do term project
  - Consultation sessions available by appointment

Previous TAs have contribute a lot …

- 吳鑑城 發表 ACL 2010 論文
- 粘子弈 Trend Micro
- 張至 CMU，創期末專題，發表在 ACL 2012
- 高定慧 Yahoo!奇摩，CoNLL 2014改錯世界亞軍
- 顏孜羲 Pinkoi, 發表 NA ACL 2015
- 張竟 Google，發表在 NAACL2015
- 劉郁蘭 京都大學實習，競逐於NTCIR
- 陳志杰 投稿長勝軍：COLING, IJCNLP, PACLIC
- 韓文彬 日本 NII實習 ACL 2019
- 蔡仲庭 台積電，投稿 ACL 2020

郭俊豪 許瑋芩 蝦皮 台積電

Why using Python?

- Neat code layout (by design)
- Lisp-like Dynamic and recursive data structure
- Support many programming paradigm
  - Imperative (procedural)
  - Functional Programming
  - Object-oriented Programming
- Programming Language of Choice for AI, NLP

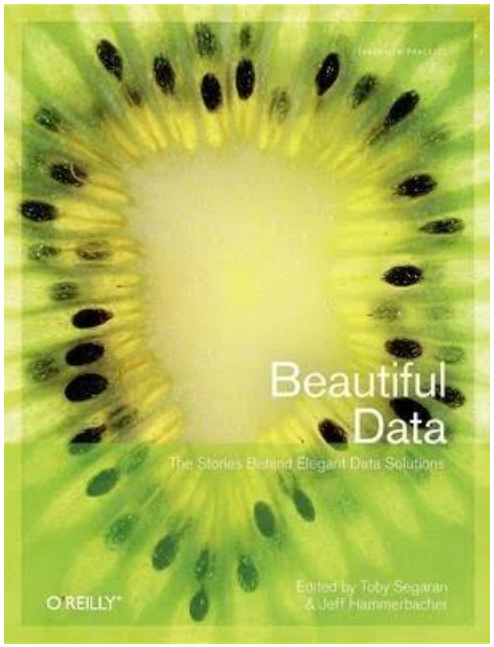
How do you run **Python**

- Interative : python (command line, IDLE)
- Batch: (under Unix) python <your program>
- Command direct: python -c "<code>"
- Batch/interative: python -i < your program>
- In a browser (CGI)
- In a notebook mixing code, comment, output

Good idea: source-rich environment

- Peter Norvig's Python code
- Famous 21-line spell checker https://norvig.com/spell-correct.html
- and word segmenter https://norvig.com/ngrams/ch14.pdf

```
import re
from collections import Counter

def words(text): return re.findall(r'\w+', text.lower())

WORDS = Counter(words(open('big.txt').read()))

def P(word, N=sum(WORDS.values())):
    "Probability of `word`."
    return WORDS[word] / N

def correction(word):
    "Most probable spelling correction for word."
    return max(candidates(word), key=P)

def candidates(word):
    "Generate possible spelling corrections for word."
    return (known([word]) or known(edits1(word)) or known(edits2(word)) or [word])

def known(words):
    "The subset of `words` that appear in the dictionary of WORDS."
    return set(w for w in words if w in WORDS)

def edits1(word):
    "All edits that are one edit away from `word`."
    letters    = 'abcdefghijklmnopqrstuvwxyz'
    splits     = [(word[:i], word[i:])    for i in range(len(word) + 1)]
    deletes    = [L + R[1:]               for L, R in splits if R]
    transposes = [L + R[1] + R[0] + R[2:] for L, R in splits if len(R)>1]
    replaces   = [L + c + R[1:]           for L, R in splits if R for c in letters]
    inserts    = [L + c + R               for L, R in splits for c in letters]
    return set(deletes + transposes + replaces + inserts)

def edits2(word):
    "All edits that are two edits away from `word`."
    return (e2 for e1 in edits1(word) for e2 in edits1(e1))
```



Learning (brushing up your) Python as easy as one, two, three

- 1: call a built-in function to split a string into words
- 2: import a module and calling functions
- 3: define a function returning ways of splitting a word
- 4: Read, count, estimate probabilities of words in a file
- 5: function of a function using (functional) decorator
- 6: Read, count, estimate probabilities of words in a file
- 7: generate candidates for correcting spelling errors

1: call a built-in function to split a string into words

```
$ Python
>>> 'Colorless green ideas sleep furiously.'.split()
['Colorless', 'green', 'ideas', 'sleep', 'furiously.']

Same as using the built-in print function:
>>> print('Colorless green ideas sleep furiously.'.split())
['Colorless', 'green', 'ideas', 'sleep', 'furiously.']
```

2: Modules and functions

```
import re
def words(text): return re.findall(r'\w+', text.lower())


from collections import Counter
WORDS = Counter(words(open('big.txt').read()))
```

3: function to return all the ways of splitting a string

```
>>> def splits(text, L=10):
>>>     return [(text[:i+1], text[i+1:])
>>>             for i in range(min(len(text), L))]

Run the function:
$ python -i my.py
>>> from pprint import pprint
>>> pprint(splits('colorlessgreenideassleepfuriously.'))
[('c', 'olorlessgreenideassleepfuriously.'),
 ('co', 'lorlessgreenideassleepfuriously.'),
 ('col', 'orlessgreenideassleepfuriously.'),
 ('colo', 'rlessgreenideassleepfuriously.'),
 ('color', 'lessgreenideassleepfuriously.'),
 ('colorl', 'essgreenideassleepfuriously.'),
 ('colorle', 'ssgreenideassleepfuriously.'),
 ('colorles', 'sgreenideassleepfuriously.'),
 ('colorless', 'greenideassleepfuriously.'),
 ('colorlessg', 'reenideassleepfuriously.')]
```

4 : Read, count, estimate probabilities of words in a file

```
N = 1024908267229 ## Size of Google Web 1T Dataset
word_count = [ line.split('\t') for line in open('count_1w.txt', 'r') ]
Pdist = dict( [ (word, float(count)/N) for word, count in word_count ] )

def Pw(word): return Pdist[word] if word in Pdist else 10./10**len(word)/N

Run the function:
>>> pprint [ (w, Pw(w)) for w in words('Colorless green ideas sleep
furiously.') ]
[('colorless', 5.0e-07),
 ('green', 0.00011),
 ('ideas', 6.6e-05),
 ('sleep', 2.9e-05),
 ('furiously', 4.4e-07)
 ('.', 9.76e-13) ]
>>> print( map(Pw, words('Colorless green ideas sleep furiously.') ))
[ 5.0e-07, 0.00011, 6.6e-05, 2.9e-05, 4.4e-07, 9.76e-13 ]
```

5: function of a function using (functional) decorator

```
@memoize
def segment(text):
    if not text: return []
    candidates = ([first]+segment(rem) for first,rem in splits(text))
    return max(candidates, key=lambda x: product(P(w) for w in x))

Run the function:
>>> print(segment('colorlessgreenideassleepfuriously.'))
['colorless', 'green', 'ideas', 'sleep', 'furiously', '.']
>>> print(' '.join(segment('colorlessgreenideassleepfuriously.')))
'colorless green ideas sleep furiously .'
```

```
class memoize:
    def __init__(self, fn):
        self.function = fn
        self.memodict = {}

    def __call__(self, *args):
        if args not in self.memodict:
                self.memodict[args] = self.function(*args)
        return self.memodict[args]
```

6: Read, count, estimate probabilities of words in a file

```
import re, collections

def words(text):
    return re.findall(r'\w+', text.lower())

word_count =
collections.Counter(words(open('big.txt').read()))

def P(word, N = sum(word_count.values())):
    return word_count[word]/N

$ python -i 6.py
>>> pprint( map(P, words('speling spelling speeling')))
[('speling', 0.0), ('spelling', 3.59e-06), ('speeling', 0.0)]
```

```
                                                    0.0)]
```

7: generate candidates for correcting spelling errors

```
letters    = 'abcdefghijklmnopqrstuvwxyz'


def edits1(word):
    splits     = [(word[:i], word[i:])    for i in range(len(word) + 1)]
    deletes    = [L + R[1:]               for L, R in splits if R]
    transposes = [L + R[1] + R[0] + R[2:] for L, R in splits if len(R)>1]
    replaces   = [L + c + R[1:]           for L, R in splits if R for c in letters]
    inserts    = [L + c + R               for L, R in splits for c in letters]
    return set(deletes + transposes + replaces + inserts)
```

```
>>> pprint(  [(L, c, R) for L, R in splits for c in 'l'] )
[('', 'l', 'speling'),
 ('s', 'l', 'peling'),
 ('sp', 'l', 'eling'),
 ('spe', 'l', 'ling'),
 ('spel', 'l', 'ing'),
 ('speli', 'l', 'ng'),
 ('spelin', 'l', 'g'),
 ('speling', 'l', '')]
>>> pprint(  [L + c + R  for L, R in splits for c in 'l'] )
['lspeling',
 'slpeling',
 'spleling',
 'spelling',
 'spelling',
 'spelilng',
 'spelinlg',
 'spelingl']
```

```
>>> pprint( list(edits1('speling')) )
['spelinx', 'spebling', 'spelinf' ... ]
>>> pprint( list(map(lambda x: (x, P(x)), list(edits1('speling')))) )
[('spjling', 0.0),
 ('bspeling', 0.0),
 ('spelint', 0.0), ...
 ('spelling', 3.5e-6), ...

>>> print( list(filter(lambda x: P(x) != 0.0, edits1('speling'))) )
['spelling']
>>> print( max(edits1('speling'), key=P) )
spelling
```

8 Lines

```
def correction(WORD):

(1) if P(WORD) > 0: return WORD
(2) Generate candidates C1 with one WORD away from word
(3) If there exists a candidate x in C1, P(x) > 0:
        return argmax(x) P(x) for x in C1
(4) Generate candidates C2: one edit away from any c in C1
(5) If there exists a candidate x in C2, P(x) > 0:
        return argmax P(x) for x in C2
```

```
def correction(word):
    return max(candidates(word), key=P)
def candidates(word):
    return (known([word]) or known(edits1(word)) or known(edits2(word)) or
[word])
def known(words):
    return set(w for w in words if w in WORDS)
def edits2(word):
    return (e2 for e1 in edits1(word) for e2 in edits1(e1))
```

```
$ python -i 8.py
>>> print('speling -->', correction('speling'))
speling --> spelling
```

9 Lines

```
def unit_tests():
    assert correction('speling') == 'spelling'              # insert
    assert correction('korrectud') == 'corrected'           # replace 2
    assert Counter(words('This is a test. 123; A TEST this is.')) == (
           Counter({'123': 1, 'a': 2, 'is': 2, 'test': 2, 'this': 2}))
    assert P('quintessential') == 0
    assert 0.07 < P('the') < 0.08
    return 'unit_tests pass'

>>> ...
```

10 Lines

```
def spelltest(tests): # Run correction(wrong) on (right, wrong) pairs
    good, unknown = 0, 0
    for right, wrong in tests:
        w = correction(wrong)
        if w == right:  good += 1
        else:                unknown += (right not in WORDS)
    n = len(tests)
    print('{:.0%} of {} correct ({:.0%} unknown) '\
                .format(good / n, n, unknown / n))

if __name__ == '__main__':
    spelltest(Testset(open('spell-testset1.txt')))

$ python -i 10.py
>>> spelltest(Testset(open('spell-testset1.txt')))
...
...
```

21 Lines spell.py by Peter Norvig

```
import re
from collections import Counter

def words(text): return re.findall(r'\w+', text.lower())

WORDS = Counter(words(open('big.txt').read()))

def P(word, N=sum(WORDS.values())):
    "Probability of `word`."
    return WORDS[word] / N

def correction(word):
    "Most probable spelling correction for word."
    return max(candidates(word), key=P)

def candidates(word):
    "Generate possible spelling corrections for word."
    return (known([word]) or known(edits1(word)) or known(edits2(word)) or [word])

def known(words):
    "The subset of `words` that appear in the dictionary of WORDS."
    return set(w for w in words if w in WORDS)

def edits1(word):
    "All edits that are one edit away from `word`."
    letters    = 'abcdefghijklmnopqrstuvwxyz'
    splits     = [(word[:i], word[i:])    for i in range(len(word) + 1)]
    deletes    = [L + R[1:]               for L, R in splits if R]
    transposes = [L + R[1] + R[0] + R[2:] for L, R in splits if len(R)>1]
    replaces   = [L + c + R[1:]           for L, R in splits if R for c in letters]
    inserts    = [L + c + R               for L, R in splits for c in letters]
    return set(deletes + transposes + replaces + inserts)

def edits2(word):
    "All edits that are two edits away from `word`."
    return (e2 for e1 in edits1(word) for e2 in edits1(e1))
```

Task for this week

- Expand http://norvig.com/spell-correct.html
- Read a sentence and hand additional error types
  - Fusion errors (e.g. “taketo” → “take to”)
  - Multi-token errors (e.g. “mor efun” → “more fun”)
  - Fusion errors (e.g. “with out” → “without”)
- Example input:
  - We tookto the street with out wering shooes.

References (downloadable books)

- Natural Language Processing with Python. Steven Bird, Ewan Klein, and Edward Loper. https://www.nltk.org/book/
- Deep Learning with Python (keras), Second Edition. François Chollet
- Think Python, Allen B. Downey. https://greenteapress.com/wp/think-python-2e/
- Dive Into Python - free Python book for experienced programmers. By Mark Pilgrim
- Thinking In Python - for intermediate Python programmers. By Bruce Eckel