自然語言處理實作 (2003-)

張俊盛 jason@nlplab.cc

助教：郭俊豪 isaac@nlplab.cc

許瑋芩 winnie@nlplab.cc

課程網站：https://lms.nthu.edu.tw/course.php?courseID=38752

2019 0917 每星期四 15:30 到作業做出來 資電館 323 iMac 教室

課程說明

- Syllabus
- 介紹自然語言處理研究所需的基本程式技巧，以及基礎研究問題
- 程式語言：Python
- 工具：Unix

進行方式

- 每週一個實作題目
- 由老師先講解背景與題目 (約30~60分鐘)，剩下的時間開始實作
- 過程中同學可以互相討論，或詢問老師及助教

評分方式

- Assignments
  - 做完題目將結果demo給助教看，再將程式上傳至iLMS，助教會根據結果的正確性及完成時間打分數
- Term Project
  - 作業成績優秀者，可以提出規劃，審查後執行
  - 其他同學，繼續做題目

歷屆助教

- 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 (兩次) 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

Why Python?

- 乾淨直覺的排版
- 動態遞迴的資料結構
- 不拘泥受限於一種典範
  - 程序
  - 函數
  - 物件
- AI, NLP 研究的首選

執行 Python 的方式

- 互動式： python (指令列、IDLE)
- 批次式： python <filename>
- 指令直接式：python -c "<text>"
- 先批次再互動：python -i <filename>
- 瀏覽器執行 (CGI)
- 混合程式、解說、輸出 ipython notebook

用例子學 Python— 1行2行3 行很簡單

- 一行：呼叫內建函數，分解字串
- 兩行：使用模組、定義函數、呼叫函數
- 3 行：定義函數，回傳字串的所有分解方式
- 4 行：讀單字次數檔案，定義單字機率函數
- 5 行：遞迴函數的函數—用@裝扮、加速
- 6 行：讀入檔案庫，計算詞的次數、詞頻
- 7 行：產生拼字錯誤的各種修正候選答案

1 行：呼叫內建函數，分解字串

```
$ 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 行：模組、定義＋呼叫函數

```
$ 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.']
```

3 行：定義函數，回傳字串分解

```
>>> 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 行：讀檔案，建立「詞典」

```
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 行：遞迴、裝飾、記憶、加速

```
@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 行：NLP 的 Hello World—wc

```
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)]
```

7 行：拼字錯誤各種修正候選答案

```
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)
```

7 行

```
>>> 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']
```

7 行

```
>>> 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 行的 Python 程式

```
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 行：測試一下

```
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 行的 Python 程式

```
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 行：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))
```

本次作業

- 擴充 http://norvig.com/spell- correct.html 程式
- 讀入一句，處理原有錯誤，以及下列錯誤：
  - Fusion errors (e.g. “taketo” → “take to”)
  - Multi-token errors (e.g. “mor efun” → “more fun”)
  - Fusion errors (e.g. “with out” → “without”)

參考文獻 (可下載)

- How to think like a Computer Scientist - introductory programming book that comes in Python and Java version. by Downey, Elkner, and Meyers
- Dive Into Python - free Python book for experienced programmers. By Mark Pilgrim
- Thinking In Python - for intermediate Python programmers. By Bruce Eckel
- Python Text Processing with NLTK 2.0 Cookbook. By Jacob Perkins