文字探勘
(Text Mining)

Python 自然語言處理
(Python for Natural Language Processing)

1082TM03
MBA, BDABI, TKU (E3611) (8480) (Spring 2020)
Mon, 7, 8, 9 (14:10-17:00) (B206)

Chichang Jou
周清江
Associate Professor
cjou@mail.tku.edu.tw

Min-Yuh Day
戴敏育
Associate Professor
myday@mail.tku.edu.tw

Dept. of Information Management, Tamkang University

2020-03-16
<table>
<thead>
<tr>
<th>週次 (Week)</th>
<th>日期 (Date)</th>
<th>內容 (Subject/Topics)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2020/03/02</td>
<td>1 2020/03/02</td>
<td>文字探勘課程介紹 (Course Orientation on Text Mining)</td>
</tr>
<tr>
<td>2 2020/03/09</td>
<td>2 2020/03/09</td>
<td>文字探勘基礎：自然語言處理 (Foundations of Text Mining: Natural Language Processing; NLP)</td>
</tr>
<tr>
<td>3 2020/03/16</td>
<td>3 2020/03/16</td>
<td>Python自然語言處理 (Python for Natural Language Processing)</td>
</tr>
<tr>
<td>4 2020/03/23</td>
<td>4 2020/03/23</td>
<td>處理和理解文本 (Processing and Understanding Text)</td>
</tr>
<tr>
<td>5 2020/03/30</td>
<td>5 2020/03/30</td>
<td>文本表達特徵工程 (Feature Engineering for Text Representation)</td>
</tr>
<tr>
<td>6 2020/04/06</td>
<td>6 2020/04/06</td>
<td>人工智慧文本分析個案研究 I (Case Study on Artificial Intelligence for Text Analytics I)</td>
</tr>
<tr>
<td>週次 (Week)</td>
<td>日期 (Date)</td>
<td>內容 (Subject/Topics)</td>
</tr>
<tr>
<td>------------</td>
<td>------------</td>
<td>----------------------</td>
</tr>
</tbody>
</table>
| 7 2020/04/13 | | 文本分類  
(Text Classification) |
| 8 2020/04/20 | | 文本摘要和主題模型  
(Text Summarization and Topic Models) |
| 9 2020/04/27 | | 期中報告 (Midterm Project Report) |
| 10 2020/05/04 | | 文本相似度和分群  
(Text Similarity and Clustering) |
| 11 2020/05/11 | | 語意分析和命名實體識別  
(Semantic Analysis and Named Entity Recognition; NER) |
| 12 2020/05/18 | | 情感分析  
(Sentiment Analysis) |
課程大綱 (Syllabus)

週次 (Week) 日期 (Date) 內容 (Subject/Topics)
13 2020/05/25 人工智慧文本分析個案研究 II
  (Case Study on Artificial Intelligence for Text Analytics II)
14 2020/06/01 深度學習和通用句子嵌入模型
  (Deep Learning and Universal Sentence-Embedding Models)
15 2020/06/08 問答系統與對話系統
  (Question Answering and Dialogue Systems)
16 2020/06/15 期末報告 I (Final Project Presentation I)
17 2020/06/22 期末報告 II (Final Project Presentation II)
18 2020/06/29 教師彈性補充教學
Outline

• Python for Natural Language Processing
Python for Natural Language Processing
Connect Google Colab in Google Drive
Google Colab
Google Colab

Colaboratory
offered by https://colab.research.google.com
A data analysis tool that combines code, output, and descriptive text into one collaborative document.
Connect Colaboratory to Google Drive
Google Colab
Google Colab
Google Colab
Run Jupyter Notebook
Python3 GPU
Google Colab
Google Colab Python Hello World

print('Hello World')
Natural Language Processing (NLP) and Text Mining

- Raw text
- Sentence Segmentation
- Tokenization
- Part-of-Speech (POS)
- Stop word removal
- Stemming / Lemmatization
- Dependency Parser
- String Metrics & Matching

Source: Nitin Hardeniya (2015), NLTK Essentials, Packt Publishing; Florian Leitner (2015), Text mining - from Bayes rule to dependency parsing
Text Analytics and Natural Language Processing (NLP)

Python for Natural Language Processing

spaCy

- spaCy: Industrial-Strength Natural Language Processing in Python
- Source: https://spacy.io/usage/spacy-101

```python
import spacy
nlp = spacy.load("en_core_web_sm")
doc = nlp("Apple is looking at buying U.K. startup for $1 billion")
for token in doc:
    print(token.text, token.pos_, token.dep_)
```
```python
import spacy

nlp = spacy.load("en_core_web_sm")
doc = nlp("Apple is looking at buying U.K. startup for $1 billion")
import pandas as pd
cols = ["text", "lemma", "POS", "explain", "stopword"]
rows = []
for t in doc:
    row = [t.text, t.lemma_, t.pos_, spacy.explain(t.pos_), t.is_stop]
    rows.append(row)
df = pd.DataFrame(rows, columns=cols)
df
```
import spacy

doc = nlp("Stanford University is located in California. It is a great university."

import pandas as pd

cols = ("text", "lemma", "POS", "explain", "stopword")

rows = []

for t in doc:
    row = [t.text, t.lemma_, t.pos_, spacy.explain(t.pos_), t.is_stop]
    rows.append(row)

df = pd.DataFrame(rows, columns=cols)
```python
[ ]
1 from spacy import displacy
2 displacy.render(doc, style="dep", jupyter=True)
```
```python
[ ]
1 import spacy
2 nlp = spacy.load("en_core_web_sm")
3 text = "Stanford University is located in California. It is a great university."
4 doc = nlp(text)
5 for ent in doc.ents:
6    print(ent.text, ent.label_)
```

- Stanford University ORG
  California GPE

```python
[ ]
1 from spacy import displacy
2 text = "Stanford University is located in California. It is a great university."
3 doc = nlp(text)
4 displacy.render(doc, style="ent", jupyter=True)
```

- Stanford University ORG is located in California GPE. It is a great university.
Keras preprocessing text

```python
# keras.preprocessing.text Tokenizer
from keras.preprocessing.text import Tokenizer
# define 5 documents
docs = ['Well done!', 'Good work', 'Great effort', 'nice work', 'Excellent!']
# create the tokenizer
t = Tokenizer()
# fit the tokenizer on the documents
t.fit_on_texts(docs)
print('docs:', docs)
print('word_counts:', t.word_counts)
print('document_count:', t.document_count)
print('word_index:', t.word_index)
print('word_docs:', t.word_docs)
# integer encode documents
texts_to_matrix = t.texts_to_matrix(docs, mode='count')
print('texts_to_matrix:')
print(texts_to_matrix)
```

Using TensorFlow backend.

docs: ['Well done!', 'Good work', 'Great effort', 'nice work', 'Excellent!']
word_counts: OrderedDict([('well', 1), ('done', 1), ('good', 1), ('work', 2), ('great', 1), ('effort', 1), ('nice', 1), ('excellent', 1)]
document_count: 5
word_index: {'work': 1, 'well': 2, 'done': 3, 'good': 4, 'great': 5, 'effort': 6, 'nice': 7, 'excellent': 8}
word_docs: {'done': 1, 'well': 1, 'work': 2, 'good': 1, 'great': 1, 'effort': 1, 'nice': 1, 'excellent': 1]
texts_to_matrix:
[[0. 0. 1. 1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 1. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1.]]
Text Classification

Source: https://developers.google.com/machine-learning/guides/text-classification/
Text Classification Workflow

• Step 1: Gather Data
• Step 2: Explore Your Data
• Step 2.5: Choose a Model*
• Step 3: Prepare Your Data
• Step 4: Build, Train, and Evaluate Your Model
• Step 5: Tune Hyperparameters
• Step 6: Deploy Your Model

Source: https://developers.google.com/machine-learning/guides/text-classification/
Text Classification Flowchart

Text Classification S/W<1500: N-gram

1. Prepare data
2. N-gram
3. N-gram range
4. Count mode
5. Scoring method
6. Select top_k features [score]
7. Normalization mode
8. Build model
9. SVM
10. MLP
11. GBDT

Text Classification S/W>=1500: Sequence

1. Select top_k features [freq]
2. min(top_1K, 2K, ..., 15K, 20K, 25K, ..., 90K, all)
3. Normalization mode
   - samplewise
   - None
   - featurewise
4. Embeddings
   - S/W < 15K
   - Yes: Fine-tuned pre-trained embedding
   - No: Frozen pre-trained embedding, Embeddings learned from scratch
5. Build model
   - RNN
   - stacked RNN
   - CNN-RNN
   - sepCNN
   - CNN
6. Hyperparameter tuning

Step 2.5: Choose a Model

**Samples/Words < 1500**

150,000/100 = 1500

IMDb review dataset, the samples/words-per-sample ratio is \( \sim 144 \)

Step 2.5: Choose a Model

Samples/Words $< 15,000$

$1,500,000/100 = 15,000$
Step 3: Prepare Your Data

Texts:
T1: 'The mouse ran up the clock'
T2: 'The mouse ran down'

Token Index:
{'the': 1, 'mouse': 2, 'ran': 3, 'up': 4, 'clock': 5, 'down': 6}.

NOTE: 'the' occurs most frequently, so the index value of 1 is assigned to it. Some libraries reserve index 0 for unknown tokens, as is the case here.

Sequence of token indexes:
T1: 'The mouse ran up the clock' =
[1, 2, 3, 4, 1, 5]
T1: 'The mouse ran down' =
[1, 2, 3, 6]
# One-hot encoding

'The mouse ran up the clock' = 

<table>
<thead>
<tr>
<th>Word</th>
<th>Index</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>The</td>
<td>1</td>
<td>[0, 1, 0, 0, 0, 0, 0, 0]</td>
</tr>
<tr>
<td>mouse</td>
<td>2</td>
<td>[0, 0, 1, 0, 0, 0, 0, 0]</td>
</tr>
<tr>
<td>ran</td>
<td>3</td>
<td>[0, 0, 0, 1, 0, 0, 0, 0]</td>
</tr>
<tr>
<td>up</td>
<td>4</td>
<td>[0, 0, 0, 0, 1, 0, 0, 0]</td>
</tr>
<tr>
<td>the</td>
<td>1</td>
<td>[0, 1, 0, 0, 0, 0, 0, 0]</td>
</tr>
<tr>
<td>clock</td>
<td>5</td>
<td>[0, 0, 0, 0, 0, 1, 0, 0]</td>
</tr>
</tbody>
</table>

[0, 1, 2, 3, 4, 5, 6]
Word embeddings

Male-Female

Verb Tense

Country-Capital

Source: https://developers.google.com/machine-learning/guides/text-classification/step-3
Word embeddings

The mouse ran up the clock
  the 1
  mouse 2
  ran 3
  up 4
  clock 5

The mouse ran down
  the 1
  mouse 2
  ran 3
  up 4
  clock 5
  down 6

[1, 2, 3, 4, 1, 5]

Embedding layer (output dim = 4)

[[0.236, -0.141, 0.000, 0.045],
[0.006, 0.652, 0.270, -0.556],
[0.305, 0.569, -0.028, 0.496],
[0.421, 0.195, -0.058, 0.477],
[0.236, -0.141, 0.000, 0.045],
[0.844, -0.001, 0.763, 0.201]]

[[0.236, -0.141, 0.000, 0.045],
[0.006, 0.652, 0.270, -0.556],
[0.305, 0.569, -0.028, 0.496],
[0.466, -0.326, 0.884, 0.007]]

Source: https://developers.google.com/machine-learning/guides/text-classification/step-3
```python
t1 = 'The mouse ran up the clock'
t2 = 'The mouse ran down'
s1 = t1.lower().split(' ')  
s2 = t2.lower().split(' ')  
terms = s1 + s2
sortedset = sorted(set(terms))
print('terms =', terms)
print('sortedset =', sortedset)
```

```python
terms = ['the', 'mouse', 'ran', 'up', 'the', 'clock', 'the', 'mouse', 'ran', 'down']
sortedset = ['clock', 'down', 'mouse', 'ran', 'the', 'up']
```

[Image -1x53 to 717x225]

https://colab.research.google.com/drive/1FEG6DnGvwfUbeo4zJ1zTunjMqf2RkCrT
t1 = 'The mouse ran up the clock'
t2 = 'The mouse ran down'
s1 = t1.lower().split(' ')
s2 = t2.lower().split(' ')
terms = s1 + s2
print(terms)

tfdict = {}
for term in terms:
    if term not in tfdict:
        tfdict[term] = 1
    else:
        tfdict[term] += 1

a = []
for k,v in tfdict.items():
    a.append('{}: {}'.format(k,v))
print(a)

['the', 'mouse', 'ran', 'up', 'the', 'clock', 'the', 'mouse', 'ran', 'down']
['the', 3, 'mouse', 2, 'ran', 2, 'up', 1, 'clock', 1, 'down', 1]
sorted_by_value_reverse = sorted(tfdict.items(), key=lambda kv: kv[1], reverse=True)

sorted_by_value_reverse_dict = dict(sorted_by_value_reverse)

id2word = {id: word for id, word in enumerate(sorted_by_value_reverse_dict)}

word2id = dict([(v, k) for (k, v) in id2word.items()])

https://colab.research.google.com/drive/1FEG6DnGwfmUbeo4zJ1zTunjMqf2RkCrT
sorted_by_value = sorted(tfdict.items(), key=lambda kv: kv[1])
print('sorted_by_value: ', sorted_by_value)

sorted_by_value2 = sorted(tfdict, key=tfdict.get, reverse=True)
print('sorted_by_value2: ', sorted_by_value2)

sorted_by_value_reverse = sorted(tfdict.items(), key=lambda kv: kv[1], reverse=True)
print('sorted_by_value_reverse: ', sorted_by_value_reverse)

sorted_by_value_reverse_dict = dict(sorted_by_value_reverse)
print('sorted_by_value_reverse_dict', sorted_by_value_reverse_dict)

id2word = {id: word for id, word in enumerate(sorted_by_value_reverse_dict)}
print('id2word', id2word)

word2id = dict([(v, k) for (k, v) in id2word.items()])
print('word2id', word2id)

print('len_words:', len(word2id))

sorted_by_key = sorted(tfdict.items(), key=lambda kv: kv[0])
print('sorted_by_key: ', sorted_by_key)

tfstring = '\n'.join(a)
print(tfstring)
tf = tfdict.get('mouse')
print(tf)
from keras.preprocessing.text import Tokenizer

define 5 documents
docs = ['Well done!', 'Good work', 'Great effort', 'nice work', 'Excellent!']
create the tokenizer
t = Tokenizer()
fit the tokenizer on the documents
t.fit_on_texts(docs)

print('docs:', docs)
print('word_counts:', t.word_counts)
print('document_counts:', t.document_counts)
print('word_index:', t.word_index)
print('word_docs:', t.word_docs)

# integer encode documents
texts_to_matrix = t.texts_to_matrix(docs, mode='count')
print('texts_to_matrix:')
print(texts_to_matrix)

docs: ['Well done!', 'Good work', 'Great effort', 'nice work', 'Excellent!']
word_counts: OrderedDict([('well', 1), ('done', 1), ('good', 1), ('work', 2), ('great', 1), ('effort', 1), ('nice', 1), ('excellent', 1)]
word_index: {'work': 1, 'well': 2, 'done': 3, 'good': 4, 'great': 5, 'effort': 6, 'nice': 7, 'excellent': 8}
texts_to_matrix: [[0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0] [0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0] [0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0] [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0] [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]]
from keras.preprocessing.text import Tokenizer

# define 5 documents
docs = ['Well done!', 'Good work', 'Great effort', 'nice work', 'Excellent!']

# create the tokenizer
t = Tokenizer()

# fit the tokenizer on the documents
t.fit_on_texts(docs)

print('docs:', docs)
print('word_counts:', t.word_counts)
print('document_count:', t.document_count)
print('word_index:', t.word_index)
print('word_docs:', t.word_docs)

# integer encode documents

texts_to_matrix = t.texts_to_matrix(docs, mode='count')
print('texts_to_matrix:')
print(texts_to_matrix)
texts_to_matrix =
t.texts_to_matrix(docs, mode='count')

docs: ['Well done!', 'Good work', 'Great effort', 'nice work', 'Excellent!']
word_counts: OrderedDict([('well', 1), ('done', 1), ('good', 1), ('work', 2), ('great', 1), ('effort', 1), ('nice', 1), ('excellent', 1)])
document_count: 5
word_index: {'work': 1, 'well': 2, 'done': 3, 'good': 4, 'great': 5, 'effort': 6, 'nice': 7, 'excellent': 8}
word_docs: {'done': 1, 'well': 1, 'work': 2, 'good': 1, 'great': 1, 'effort': 1, 'nice': 1, 'excellent': 1}
texts_to_matrix:
[[0. 0. 1. 1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 1. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 1. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 1.]]
from keras.preprocessing.text import Tokenizer

# define 5 documents
docs = ['Well done!', 'Good work', 'Great effort', 'nice work', 'Excellent!']

# create the tokenizer
t = Tokenizer()
# fit the tokenizer on the documents
t.fit_on_texts(docs)
print('docs:', docs)
print('word_counts:', t.word_counts)
print('document_count:', t.document_count)
print('word_index:', t.word_index)
print('word_docs:', t.word_docs)

# integer encode documents
texts_to_matrix = t.texts_to_matrix(docs, mode='tfidf')
print('texts_to_matrix:')
print(texts_to_matrix)

texts_to_matrix:
[[0. 0. 1.25276297 1.25276297 0. 0. 0. 0. 0. ]
 [0. 0.98082925 0. 0. 1.25276297 0. 0. 0. 0. ]
 [0. 0. 0. 0. 0. 1.25276297 1.25276297 0. 0. ]
 [0. 0.98082925 0. 0. 0. 0. 0. 1.25276297 0. ]
 [0. 0. 0. 0. 0. 0. 0. 0. 1.25276297]]
NLTK (Natural Language Toolkit)

NLTK 3.0 documentation

NEXT | MODULES | INDEX

Natural Language Toolkit

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

Thanks to a hands-on guide introducing programming fundamentals alongside topics in computational linguistics, plus comprehensive API documentation, NLTK is suitable for linguists, engineers, students, educators, researchers, and industry users alike. NLTK is available for Windows, Mac OS X, and Linux. Best of all, NLTK is a free, open source, community-driven project.

NLTK has been called “a wonderful tool for teaching, and working in, computational linguistics using Python,” and “an amazing library to play with natural language.”

Natural Language Processing with Python provides a practical introduction to programming for language processing. Written by the creators of NLTK, it guides the reader through the fundamentals of writing Python programs, working with corpora, categorizing text, analyzing linguistic structure, and more. The book is being updated for Python 3 and NLTK 3. (The original Python 2 version is still available at http://nltk.org/book_1ed.)

Some simple things you can do with NLTK

Tokenize and tag some text:

```python
>>> import nltk

http://www.nltk.org/```
### Python Jieba

“结巴”中文分词

[GitHub, Inc. [US]](https://github.com/fxsjy/jieba)

[https://github.com/fxsjy/jieba](https://github.com/fxsjy/jieba)

<table>
<thead>
<tr>
<th>Branch: master</th>
<th>New pull request</th>
</tr>
</thead>
</table>

| extra_dict | update to v0.33 | 2 years ago |
| jieba | Bugfix for HMM=False in parallelism. | 6 months ago |
| test | Bugfix for HMM=False in parallelism. | 6 months ago |
| .gitattributes | first commit | 4 years ago |
| .gitignore | update jieba3k | 2 years ago |
| Changelog | version change 0.38 | 3 years ago |
| LICENSE | add a license file | 4 years ago |
| MANIFEST.in | include Changelog & README.md in the distribution package | 4 years ago |
| README.md | Update README.md | 8 months ago |

[https://github.com/fxsjy/jieba](https://github.com/fxsjy/jieba)
import jieba
import jieba.posseg as pseg
sentence = "銀行產業正在改變，金融機構欲挖角科技人才"
words = jieba.cut(sentence)
print(sentence)
print(" ".join(words))
wordspos = pseg.cut(sentence)
result = ''
for word, pos in wordspos:
    print(word + ' (' + pos + ')')
    result = result + ' ' + word + '(' + pos + ')
print(result.strip())
import jieba
words = jieba.cut(sentence)
print(sentence)
print(" ".join(words))
wordpos = pseg.cut(sentence)
result = ''
for word, pos in wordpos:
    print(word + ' (' + pos + ')')
    result = result + ' ' + word + ' (' + pos + ')
print(result.strip())

銀行產業正在改變，金融機構欲挖角科技人才
銀行 產業 正在 改變 ， 金融 機構 欲 挖角 科技人才
銀行 (n)
產業 (n)
正在 (t)
改變 (v)
， (x)
金融 (n)
機構 (n)
欲 (d)
挖角 (n)
科技人才 (n)
銀行(n) 產業(n) 正在(t) 改變(v) ，(x) 金融(n) 機構(n) 欲(d) 挖角(n) 科技人才(n)
```python
import jieba
import jieba.posseg as pseg
sentence = "銀行產業正在改變，金融機構欲挖角科技人才"
words = jieba.cut(sentence)
print(sentence)
print(" ".join(words))
wordspos = pseg.cut(sentence)
result = ""
for word, pos in wordspos:
    print(word + ' (' + pos + ')')
    result = result + ' ' + word + ' (' + pos + ')
print(result.strip())
```

銀行產業正在改變，金融機構欲挖角科技人才
銀行 產業 正在 改變 ， 金融 機構 欲 挖角 科技人才
銀行 (n)
產業 (n)
正在 (t)
改變 (v)
， (x)
金融 (n)
機構 (n)
欲 (d)
挖角 (n)
科技人才 (n)
銀行(n) 產業(n) 正在(t) 改變(v) ，(x) 金融(n) 機構(n) 欲(d) 挖角(n) 科技人才(n)
Python Jieba "结巴”中文分词

• https://github.com/fxsjy/jieba
• jieba.set_dictionary('data/dict.txt.big')
  – #/anaconda/lib/python3.5/site-packages/jieba
  – dict.txt (5.4MB)(349,046)
  – dict.txt.big.txt (8.6MB)(584,429)
  – dict.txt.small.txt (1.6MB)(109,750)
  – dict.tw.txt (4.2MB)(308,431)
• https://github.com/ldkrsi/jieba-zh_TW
  – 結巴中文斷詞台灣繁體版本
TensorFlow NLP Examples

• Basic Text Classification
  (Text Classification) (46 Seconds)

• NMT with Attention
  (20-30 minutes)
Text Classification

IMDB Movie Reviews

https://colab.research.google.com/drive/1x16h1GhHsLIrLYtPCvCHaoO1W-i_gror
Summary

• Python for Natural Language Processing
References

• Ramesh Sharda, Dursun Delen, and Efraim Turban (2017), Business Intelligence, Analytics, and Data Science: A Managerial Perspective, 4th Edition, Pearson.

• Rajesh Arumugam (2018), Hands-On Natural Language Processing with Python: A practical guide to applying deep learning architectures to your NLP applications, Packt.


• Christopher D. Manning and Hinrich Schütze (1999), Foundations of Statistical Natural Language Processing, The MIT Press.


• Nitin Hardeniya (2015), NLTK Essentials, Packt.

• Bing Liu (2009), Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data, Springer.