Class 02: 自然語言處理

林義隆
義守大學 資訊工程系 副教授

產學合作計畫

義守大學醫學核心專題研究計畫補助辦法

108 年 08 月 21 日行政會議通過（全文），108 年 08 月 30 日校長核定公告

第 一 條　為鼓勵專任教師積極與醫學相關領域進行多元合作研究，提升醫學院研究能量、跨領域及與新興科技領域結合等合作發展，特訂定「義守大學醫學核心專題研究計畫補助辦法」（以下簡稱本辦法）。

第 二 條　為發展本校醫學特色研究，本辦法補助計畫（以下簡稱專題計畫）區分為以下二類型：

一、醫學核心個別型計畫。

二、醫學核心整合型計畫。

第 三 條　醫學核心個別型計畫為與醫學相關之跨領域計畫，主持人應由本校分屬醫學院及非醫學院之二位專任教師共同擔任，並推派一位負責計畫聯絡及經費核銷工作。

第 四 條　醫學核心整合型計畫為與醫學相關之跨領域整合計畫，其中應包含三個以上之子計畫，每一子計畫執行期間為二至三年。

子計畫主持人應為本校專任教師，其中至少一個子計畫主持人應為本校醫學院專任教師。

第 五 條　各類型專題計畫提出時間，於每年公告時程向研究發展處學術發展組（以下簡稱學發組）提出申請，每位教師每學年於各類型計畫以申請一件為限。

第 六 條　申請之專題計畫，不得重覆申請同一學年度本校校內專題研究計畫、校內專題產學合作計畫、與義大醫療財團法人研究合

NLP

- Word Segmentation
- Part of Speech Tagging
- Stemming
- Named Entity Extraction
- Parsing
- Text Categorization

課程大綱

- 本課程與自然語言處理之關聯
- 斷辭斷字
- 去停用字
- 詞向量表示
- 詞袋模型

課程重點

- 句子分割
- 去停用字
- 文字編碼
- 詞彙
- 使用Python字典模型建立
  - Index to word
  - Word ti index

Python 語法

- 套件: NLTK/Scikit-Learn/Keras
- set/dictionary/tuple/list 語法

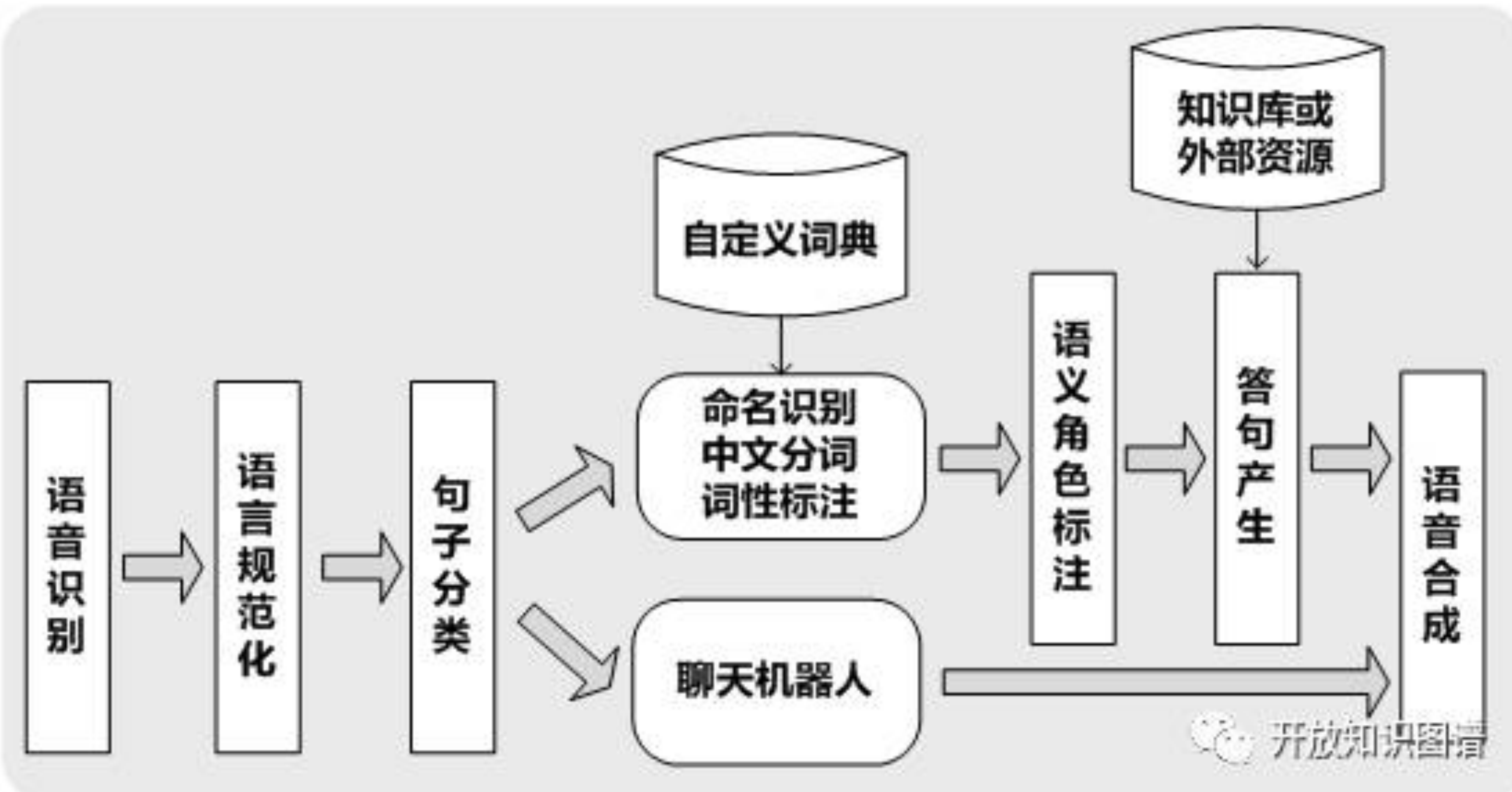
自然語言處理

http://www.gdhcfunds.com/index.php?c=article&id=338

自然語言處理

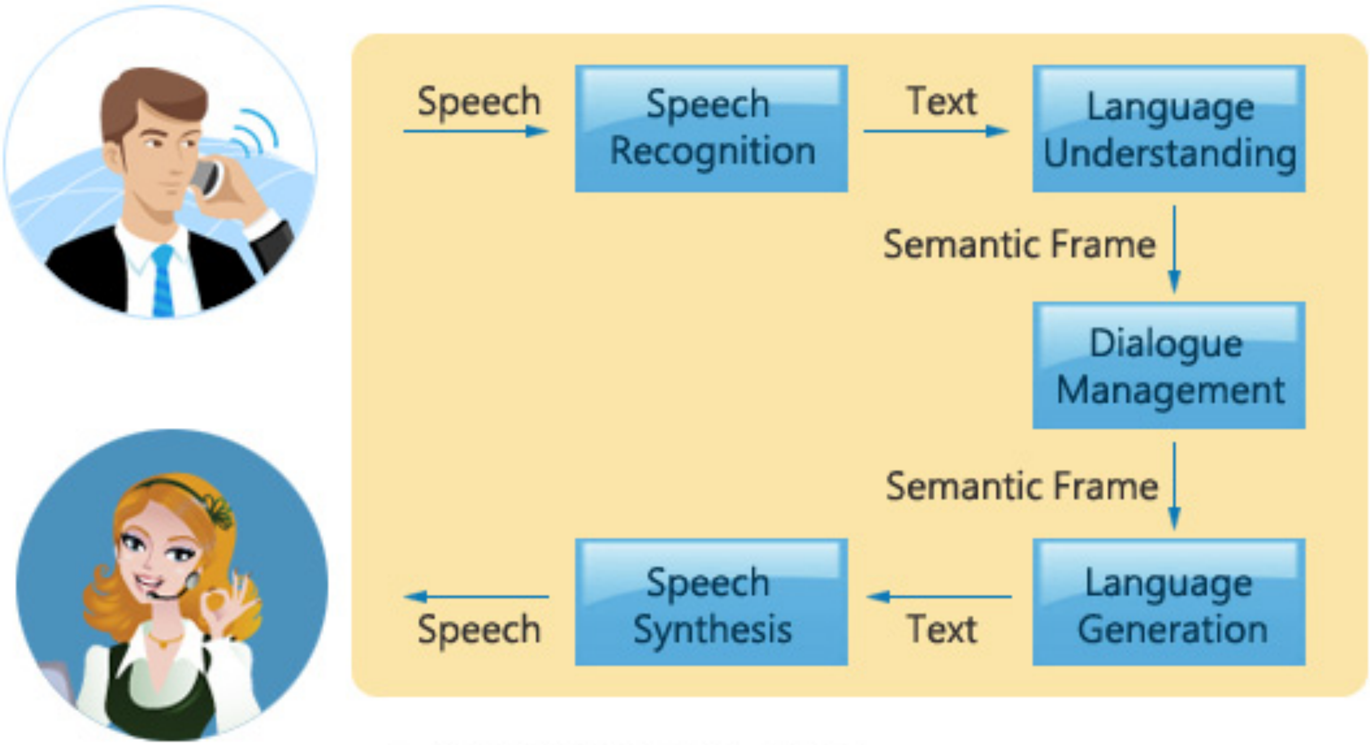
資料來源| 中研院: 研之有物

對話系統



语音识别
语言规范化
句子分类
自定义词典
命名识别
中文分词
词性标注
聊天机器人
语义角色标注
知识库或
外部资源
答句产生
语音合成
开放知识图谱

對話系統



口語對話系統架構示意圖

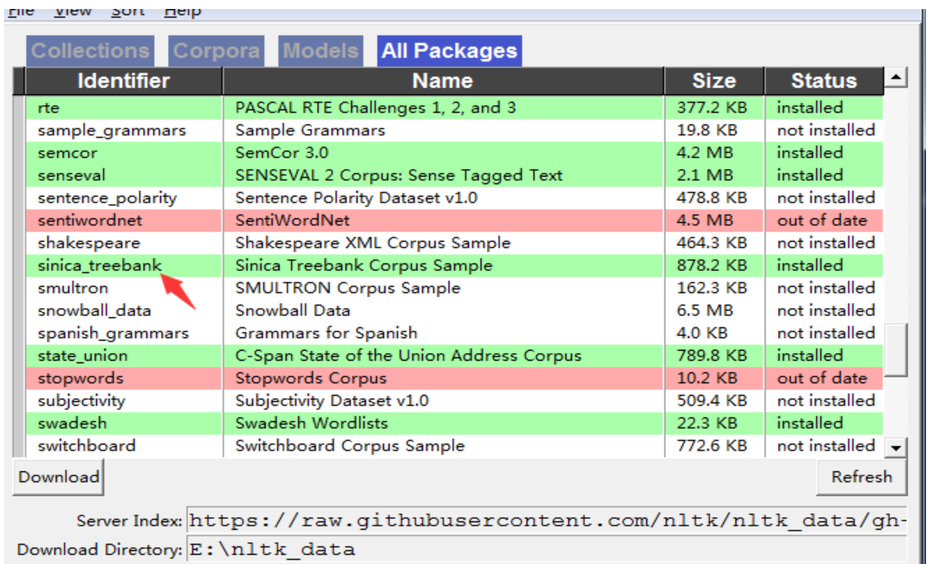
http://atc.ccl.itri.org.tw/interaction/conversation.php

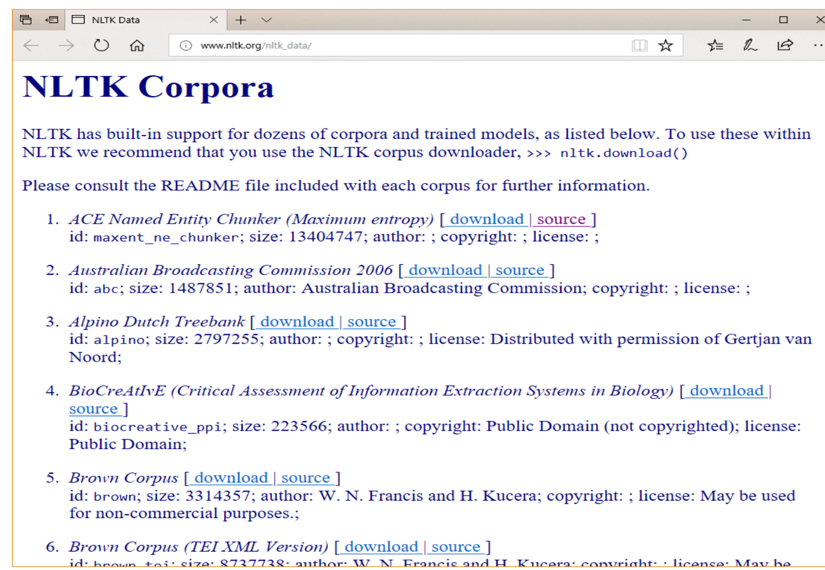
NLTK

import nltk

nltk.download()

python -m nltk.downloader all

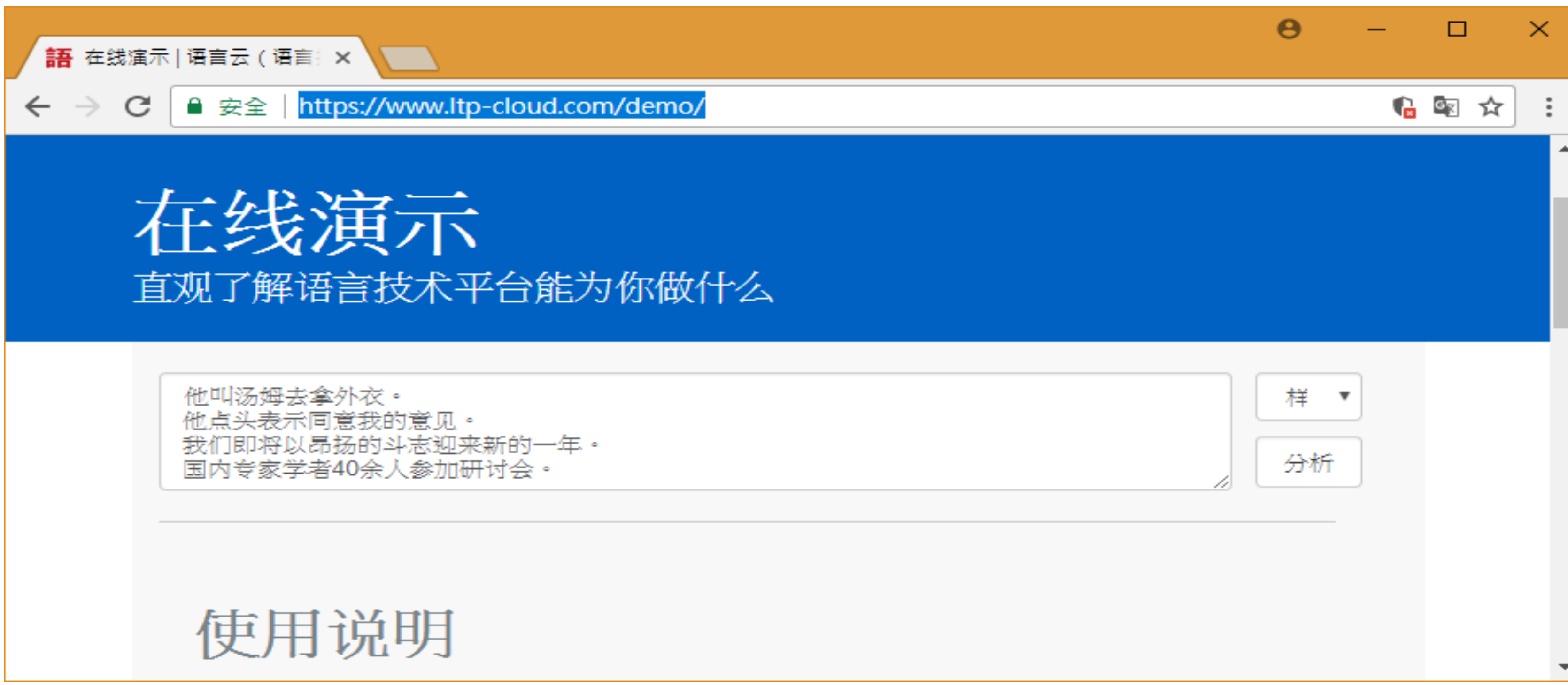




http://www.nltk.org/ http://www.nltk.org/nltk_data/

LTP

- Language Technology Platform (LTP)
- pip install pyltp

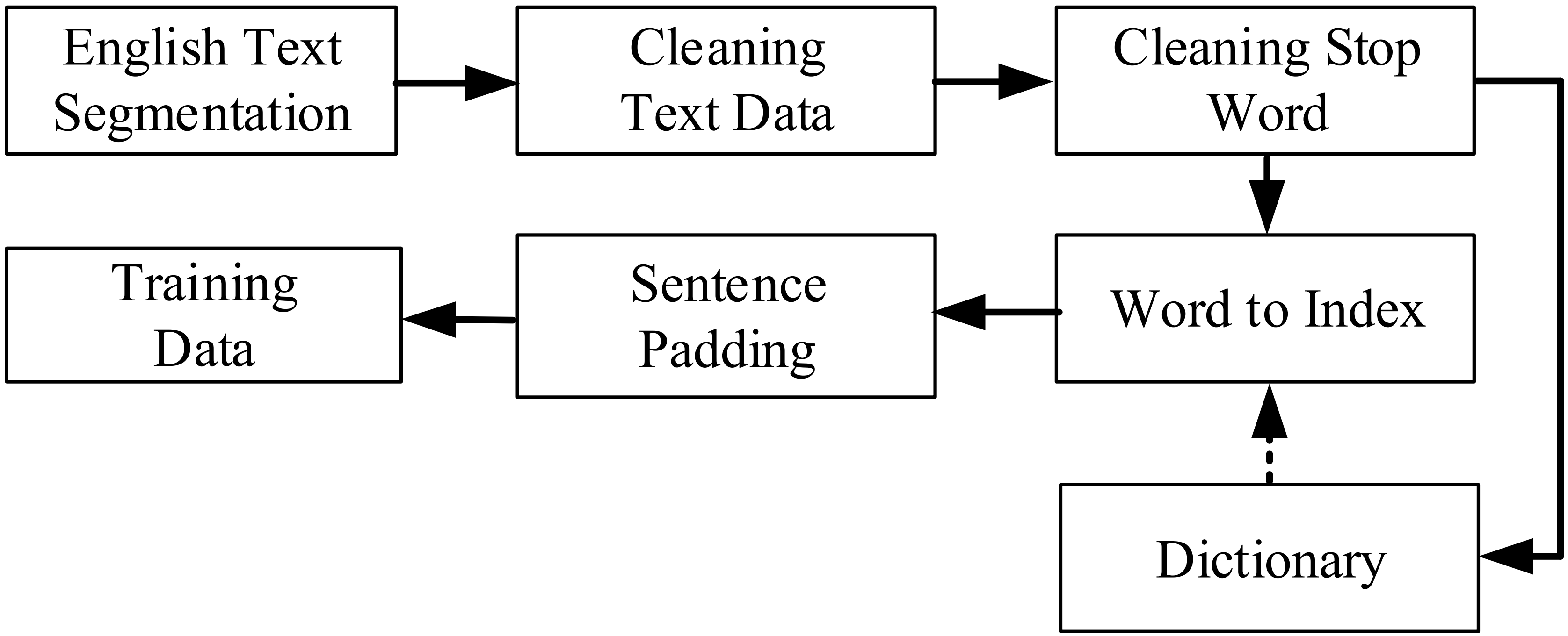


语言技术平台云　https://www.ltp-cloud.com/demo/

CKIP

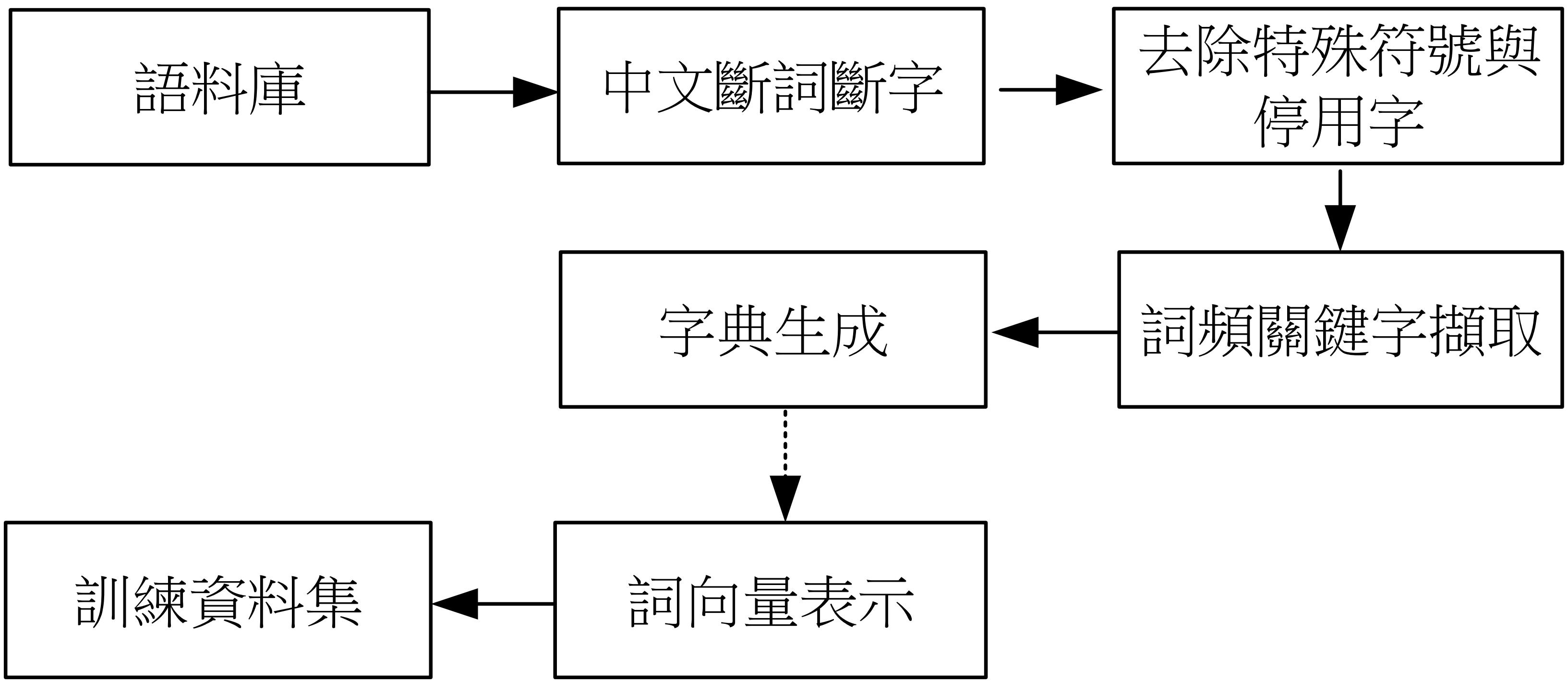
http://ckipsvr.iis.sinica.edu.tw/

英文前處理



English Text Segmentation
Cleaning Text Data
Cleaning Stop Word
Word to Index
Dictionary
Sentence Padding
Training Data

中文前處理

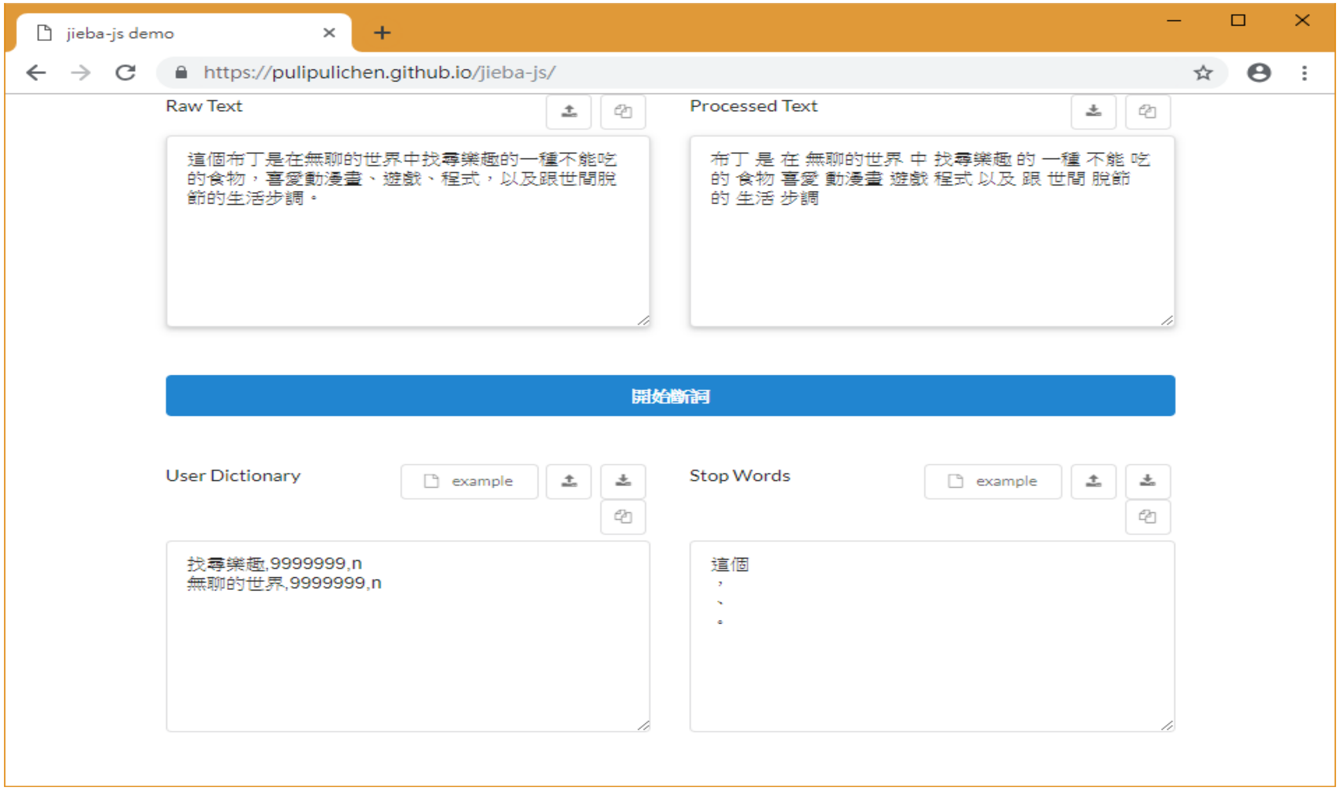


中文分詞器

- KIP: 台灣中研院研發的一款斷詞器，不過並未對外公布技術節。
- HanLP: 這是一個開源的分詞器(java),我在這篇Hanlp自然語言處理器有使用範例
- Ansj: 這也是一個開源的中文分詞器(java)
- jieba: Python 的中文分詞器

中文分詞器

- Jieba Package



https://pulipulichen.github.io/jieba-js/

Jieba 中文斷詞器

- Raw Text
- Processed Text
- User Dictionary (user_dict.txt)
- Stop Words (stop_words.txt)

基於Jieba的詞性標註

```
In [9]: import jieba
        import jieba.posseg as psg
        s='皮膚科在哪裡?'
        cut = jieba.cut(s)
        print(','.join(cut))
        print([(x.word,x.flag) for x in psg.cut(s)])

皮膚科,在,哪裡,?
[('皮膚', 'n'), ('科', 'n'), ('在', 'p'), ('哪裡', 'r'), ('?', 'x')]

In [10]: s='陳大義醫師在哪裡?'
         cut = jieba.cut(s)
         print(','.join(cut))

陳,大義醫師,在,哪裡,?

In [11]: s='心臟科在哪裡?'
         cut = jieba.cut(s)
         print(','.join(cut))
         print([(x.word,x.flag) for x in psg.cut(s)])

心臟科,在,哪裡,?
[('心臟科', 'n'), ('在', 'p'), ('哪裡', 'r'), ('?', 'x')]

In [12]: cut = jieba.cut(s)
         print([(x.word,x.flag) for x in psg.cut(s)])
         s='腦外科怎麼走?'
         cut = jieba.cut(s)
         print(','.join(cut))

[('心臟科', 'n'), ('在', 'p'), ('哪裡', 'r'), ('?', 'x')]
腦,外科,怎,麼,走,?

In [14]: print([(x.word,x.flag) for x in psg.cut(s)])
         s='腦外科在哪裡?'
         cut = jieba.cut(s)
         print(','.join(cut))

[('腦', 'zg'), ('外科', 'n'), ('在', 'p'), ('哪裡', 'r'), ('?', 'x')]
腦,外科,在,哪裡,?

In [15]: print([(x.word,x.flag) for x in psg.cut(s)])

[('腦', 'zg'), ('外科', 'n'), ('在', 'p'), ('哪裡', 'r'), ('?', 'x')]
```

問題討論

- 使用Python建立字典

基於Jieba的詞性標註

In [7]:
```
import jieba
import jieba.posseg as psg
s='周兆明老師在哪裡'
cut = jieba.cut(s)
print(','.join(cut))
print([(x.word,x.flag) for x in psg.cut(s)])
```

```
周兆明,老師,在,哪裡
[('周兆明', 'nr'), ('老師', 'n'), ('在', 'p'), ('哪裡', 'r')]
```

In [8]:
```
s='我要找王小明老師'
cut = jieba.cut(s)
print(','.join(cut))
print([(x.word,x.flag) for x in psg.cut(s)])
```

```
我要,找,王小明,老師
[('我', 'r'), ('要', 'v'), ('找', 'v'), ('王小明', 'nr'), ('老師', 'n')]
```

In [9]:
```
s='丁釩在哪裡'
cut = jieba.cut(s)
print(','.join(cut))
print([(x.word,x.flag) for x in psg.cut(s)])
```

```
丁,釩,在,哪裡
[('丁釩', 'nr'), ('在', 'p'), ('哪裡', 'r')]
```

In [10]:
```
s='我要找丁釩'
cut = jieba.cut(s)
print(','.join(cut))
print([(x.word,x.flag) for x in psg.cut(s)])
```

```
我要,找,丁,釩
[('我', 'r'), ('要', 'v'), ('找', 'v'), ('丁', 'nr'), ('釩', 'n')]
```

詞袋模型簡介

- One-Hot Encoding
- 字母形式/字串形式轉換成數字形式
- Token (字符)
- Vocabulary (詞彙)
  - 整個文件中的字詞(word)
- 文件的特徵
  - 特徵向量
  - 字詞與出現的次數

字詞(word)轉特徵向量

- One Hard Encoding
- 字詞出現的次數(Count)建立詞袋
  - 單元模型(1-gram model或unigram model)
- 詞頻-反向文件詞頻(TF-IDF)
- 詞特徵向量

One-Hot Encoding

- 使用 keras 的 Tokenizer 類別

```
#OneHotEncoder
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder()
encoder.fit([
    ['A', 'C', 'B', 'L'],
    ['B', 'D', 'F', 'D'],
    ['C', 'D', 'C', 'L'],
    ['B', 'C', 'E', 'D']
])
encoded_vector = encoder.transform([['C', 'D', 'F', 'D']]).toarray()
print("\n Encoded vector =", encoded_vector)
```

```
 Encoded vector = [[0. 0. 1. 0. 1. 0. 0. 0. 1. 1. 0.]]
```

次數詞袋範例說明

- 使用scikit-learn中的 CountVectorizer 類別

```
from sklearn.feature_extraction.text import CountVectorizer
# list of text documents
text = ["The quick brown fox jumped over the lazy dog."]
# create the transform
vectorizer = CountVectorizer()
# tokenize and build vocab
vectorizer.fit(text)
# summarize
print(vectorizer.get_feature_names())
print(vectorizer.vocabulary_)
# encode document
vector = vectorizer.transform(text)
# summarize encoded vector
print(vector.shape)
print(type(vector))
print(vector.toarray())
```

```
['brown', 'dog', 'fox', 'jumped', 'lazy', 'over', 'quick', 'the']
{'the': 7, 'quick': 6, 'brown': 0, 'fox': 2, 'jumped': 3, 'over': 5, 'lazy': 4, 'dog': 1}
(1, 8)
<class 'scipy.sparse.csr.csr_matrix'>
[[1 1 1 1 1 1 1 2]]
```

TF-IDF 詞袋

- 使用 scikit-learn中的 TfidfTransformer

When we are analyzing text data, we often encounter words that occur across multiple documents from both classes. Those frequently occurring words typically don't contain useful or discriminatory information. In this subsection, we will learn about a useful technique called term frequency-inverse document frequency (tf-idf) that can be used to downweight those frequently occurring words in the feature vectors. The tf-idf can be de ned as the product of the term frequency and the inverse document frequency:

$$\text{tf-idf}(t, d) = \text{tf}\ (\text{t,d}) \times \text{idf}(t, d)$$

Here the tf(t, d) is the term frequency that we introduced in the previous section, and the inverse document frequency *idf(t, d)* can be calculated as:

$$\text{idf}(t, d) = \log \frac{n_d}{1 + \text{df}(d, t)},$$

where n_d is the total number of documents, and *df(d, t)* is the number of documents *d* that contain the term *t*. Note that adding the constant 1 to the denominator is optional and serves the purpose of assigning a non-zero value to terms that occur in all training samples; the log is used to ensure that low document frequencies are not given too much weight.

https://github.com/rasbt/python-machine-learning-book-2nd-edition/blob/master/code/ch08/ch08.ipynb

TF-IDF 詞袋

- 使用 scikit-learn中的 TfidfTransformer

However, if we'd manually calculated the tf-idfs of the individual terms in our feature vectors, we'd have noticed that the `TfidfTransformer` calculates the tf-idfs slightly differently compared to the standard textbook equations that we de ned earlier. The equations for the idf and tf-idf that were implemented in scikit-learn are:

$$\text{idf}(t,\ d) = log\frac{1 + n_d}{1 + \text{df}(d,\ t)}$$

The tf-idf equation that was implemented in scikit-learn is as follows:

$$\text{tf-idf}(t,\ d) = \text{tf}(t,\ d) \times (\text{idf}(t,\ d) + 1)$$

While it is also more typical to normalize the raw term frequencies before calculating the tf-idfs, the `TfidfTransformer` normalizes the tf-idfs directly.

By default (`norm='l2'`), scikit-learn's TfidfTransformer applies the L2-normalization, which returns a vector of length 1 by dividing an un-normalized feature vector *v* by its L2-norm:

$$v_{\text{norm}} = \frac{v}{\|v\|_2} = \frac{v}{\sqrt{v_1^2 + v_2^2 + \ldots + v_n^2}} = \frac{v}{\left(\sum_{i=1}^{n} v_i^2\right)^{\frac{1}{2}}}$$

To make sure that we understand how TfidfTransformer works, let us walk through an example and calculate the tf-idf of the word is in the 3rd document.

https://github.com/rasbt/python-machine-learning-book-2nd-edition/blob/master/code/ch08/ch08.ipynb

TF-IDF 詞袋範例說明

```python
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
# list of text documents
text = ["The quick brown fox jumped over the lazy dog.", "The dog.", "The fox"]
# create the transform
vectorizer = TfidfVectorizer()
# tokenize and build vocab
vectorizer.fit(text)
np.set_printoptions(precision=5)
# summarize
print(vectorizer.vocabulary_)
print(vectorizer.get_feature_names())
print(vectorizer.idf_)
vector = vectorizer.transform(text)
#summarize encoded vector
print(vector.shape)
print(vector.toarray())
```

```
{'the': 7, 'quick': 6, 'brown': 0, 'fox': 2, 'jumped': 3, 'over': 5, 'lazy': 4, 'dog': 1}
['brown', 'dog', 'fox', 'jumped', 'lazy', 'over', 'quick', 'the']
[1.69315 1.28768 1.28768 1.69315 1.69315 1.69315 1.69315 1.     ]
(3, 8)
[[0.36389 0.27675 0.27675 0.36389 0.36389 0.36389 0.36389 0.42983]
 [0.      0.78981 0.      0.      0.      0.      0.      0.61336]
 [0.      0.      0.78981 0.      0.      0.      0.      0.61336]]
```

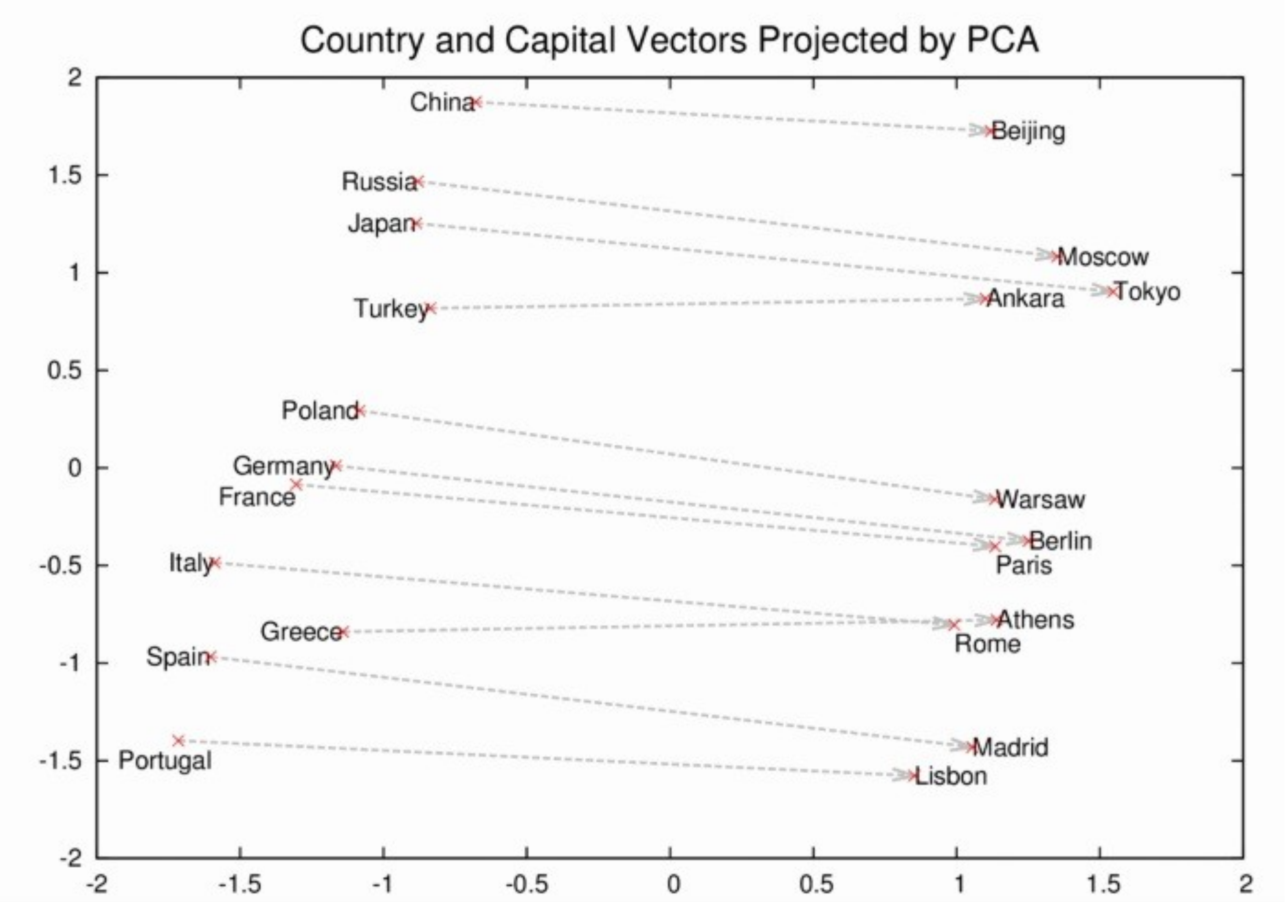
the

Word Embedding

- Google's **word2vec** [Mikolov et al. 2013]
  - COBW
  - Skip-gram
- **GloVe** [Levy et al. 2014]

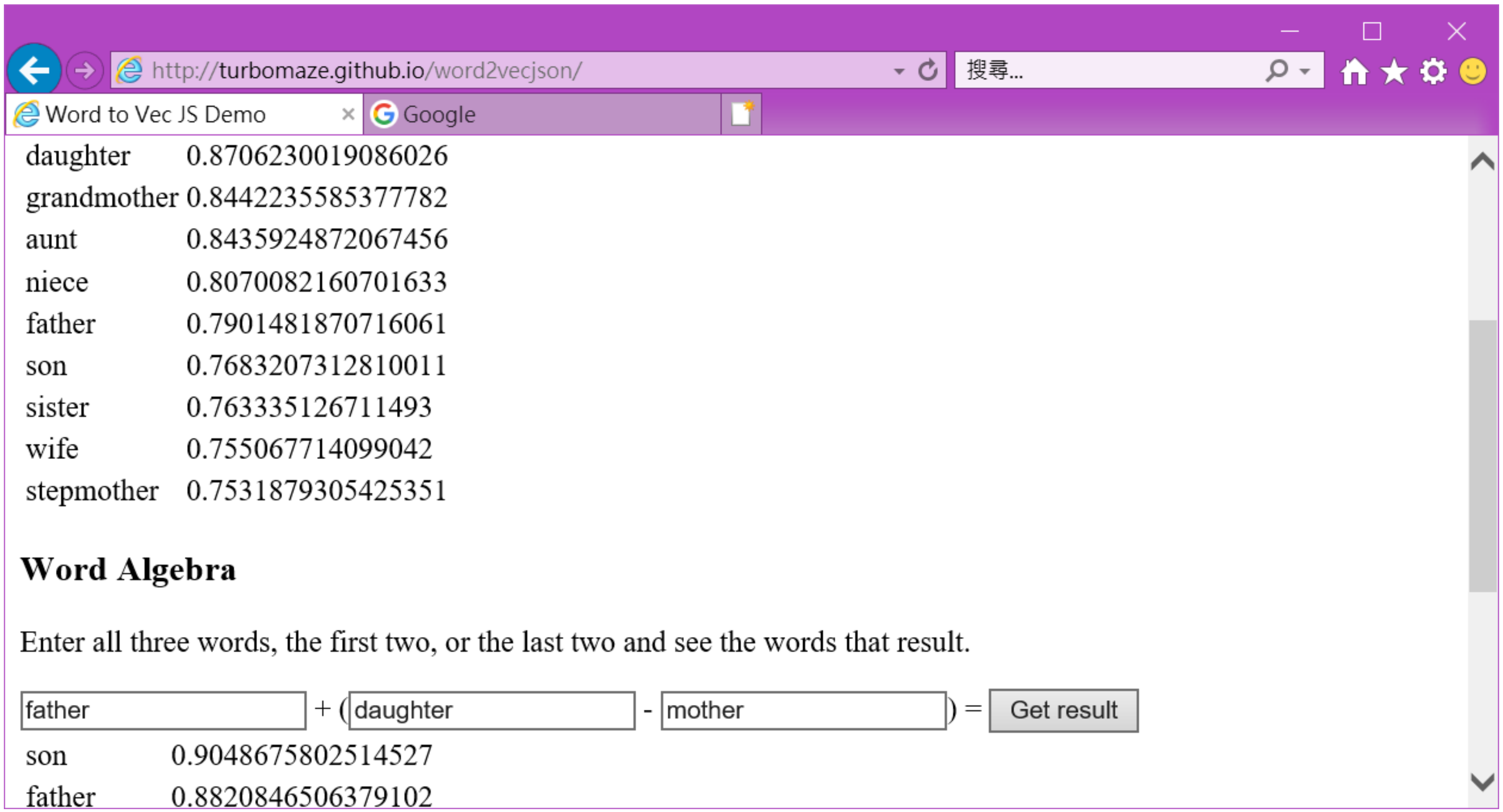
https://code.google.com/archive/p/word2vec/
https://nlp.stanford.edu/projects/glove/

Word to Vector



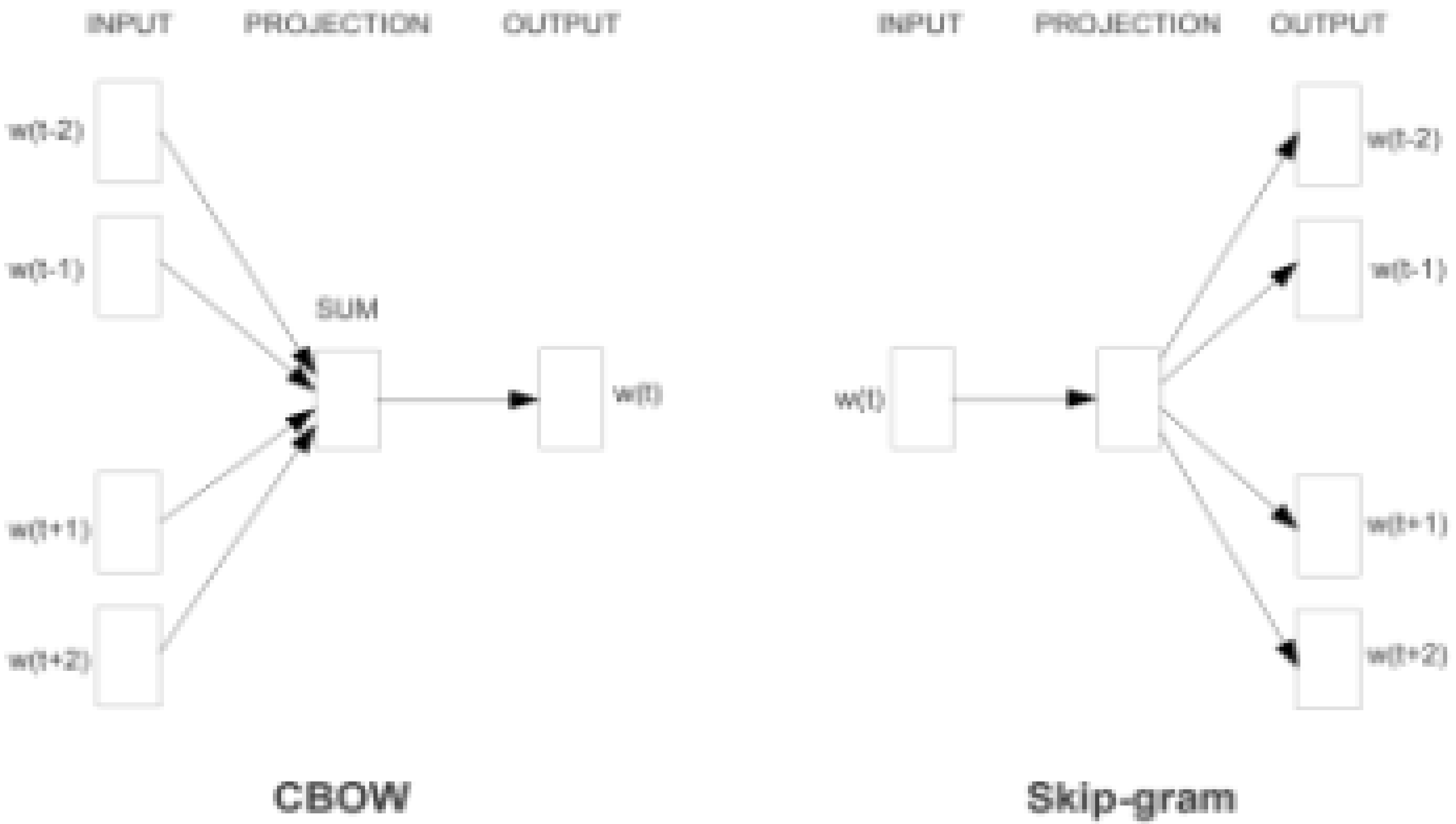
Google Open Source Blog

Word to Vec JS Demo



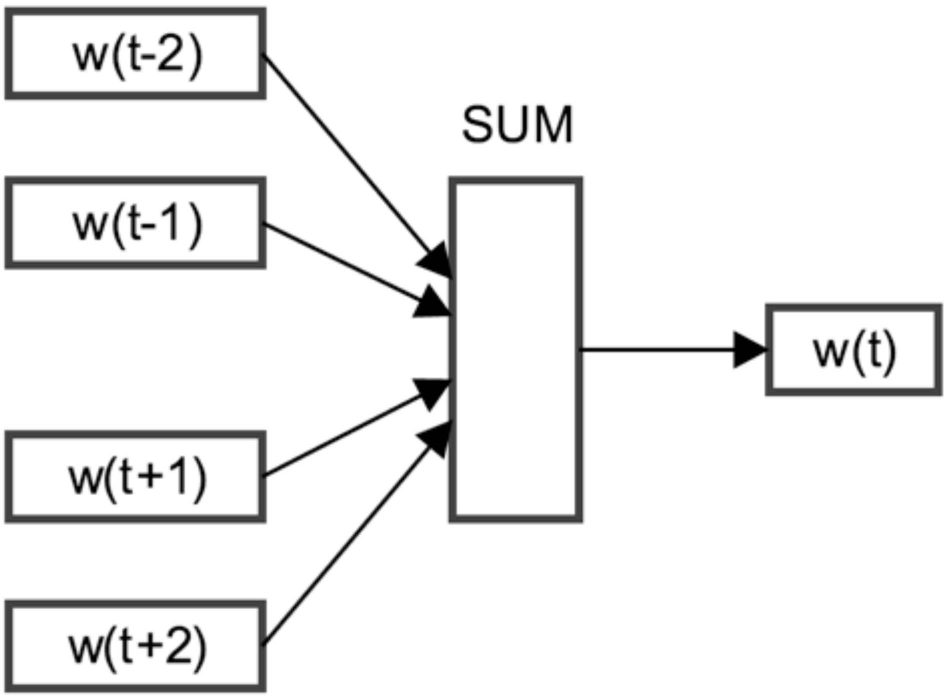
http://turbomaze.github.io/word2vecjson/

BOW & Skip-Gram Models



Bag of Words Model

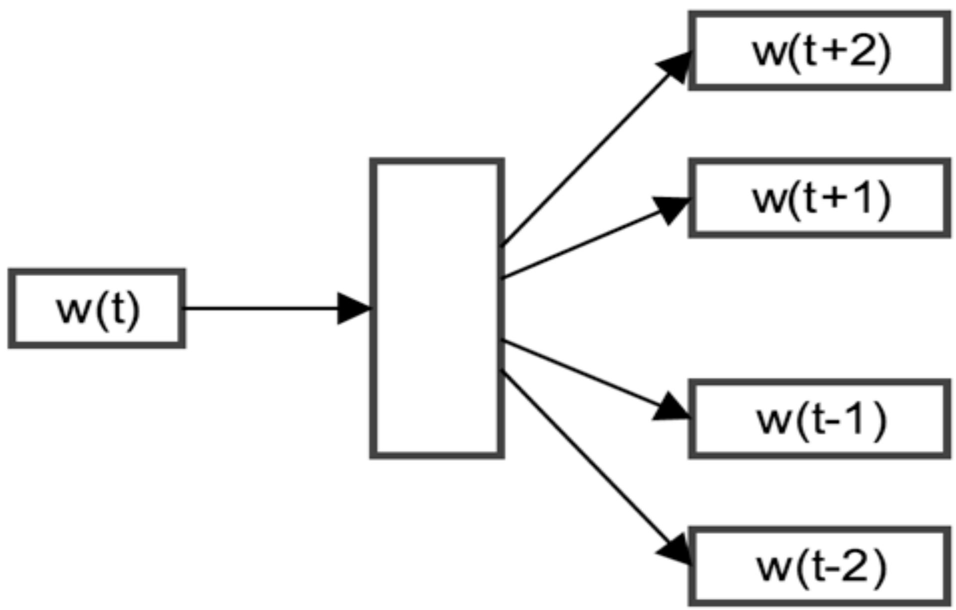
- Predict the surrounding words, based on the current word.



Mikolov et. al. 2013. Efficient Estimation of Word Representations in Vector Space.

Skip-Gram Model

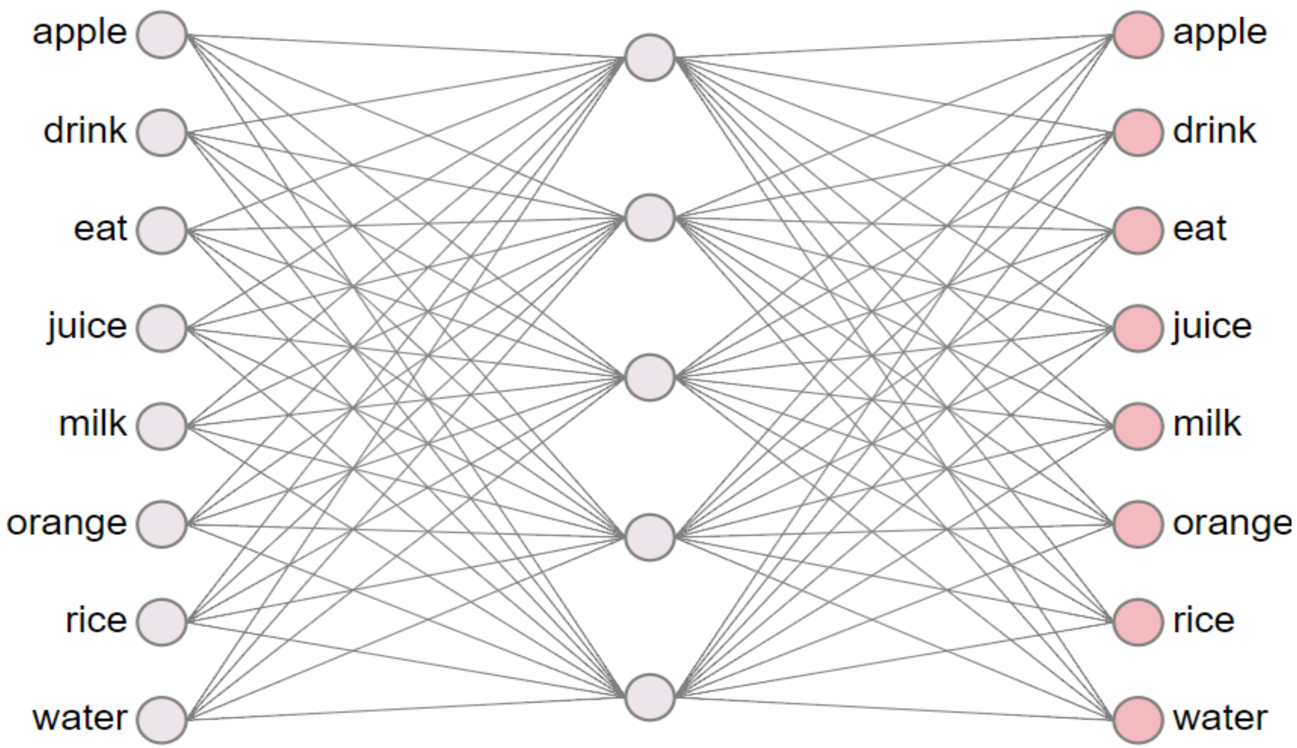
- Predict the surrounding words, based on the current word.



Mikolov et. al. 2013. Efficient Estimation of Word Representations in Vector Space.

Word to Vector

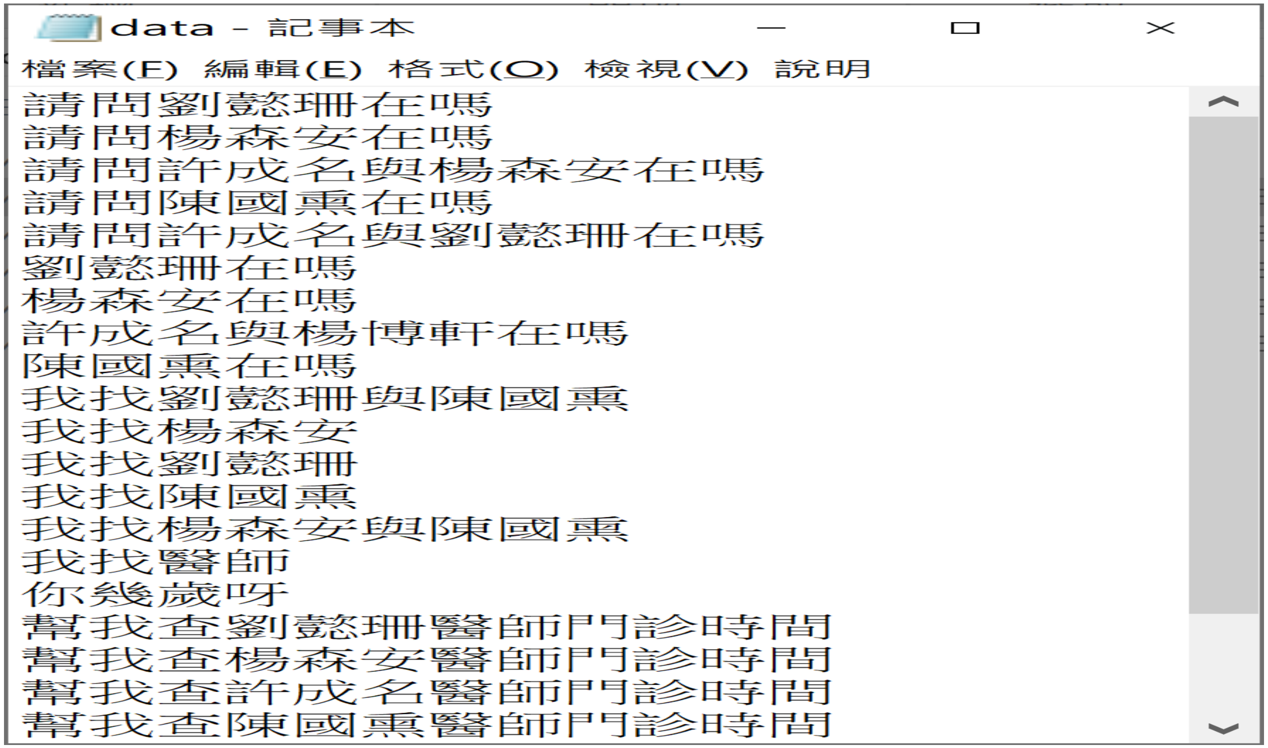
- **WEVI**



https://ronxin.github.io/wevi/

作業

- 判斷文件中醫師有幾位



data - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) 說明

請問劉懿珊在嗎
請問楊森安在嗎
請問許成名與楊森安在嗎
請問陳國熏在嗎
請問許成名與劉懿珊在嗎
劉懿珊在嗎
楊森安在嗎
許成名與楊博軒在嗎
陳國熏在嗎
我找劉懿珊與陳國熏
我找楊森安
我找劉懿珊
我找陳國熏
我找楊森安與陳國熏
我找醫師
你幾歲呀
幫我查劉懿珊醫師門診時間
幫我查楊森安醫師門診時間
幫我查許成名醫師門診時間
幫我查陳國熏醫師門診時間

作業

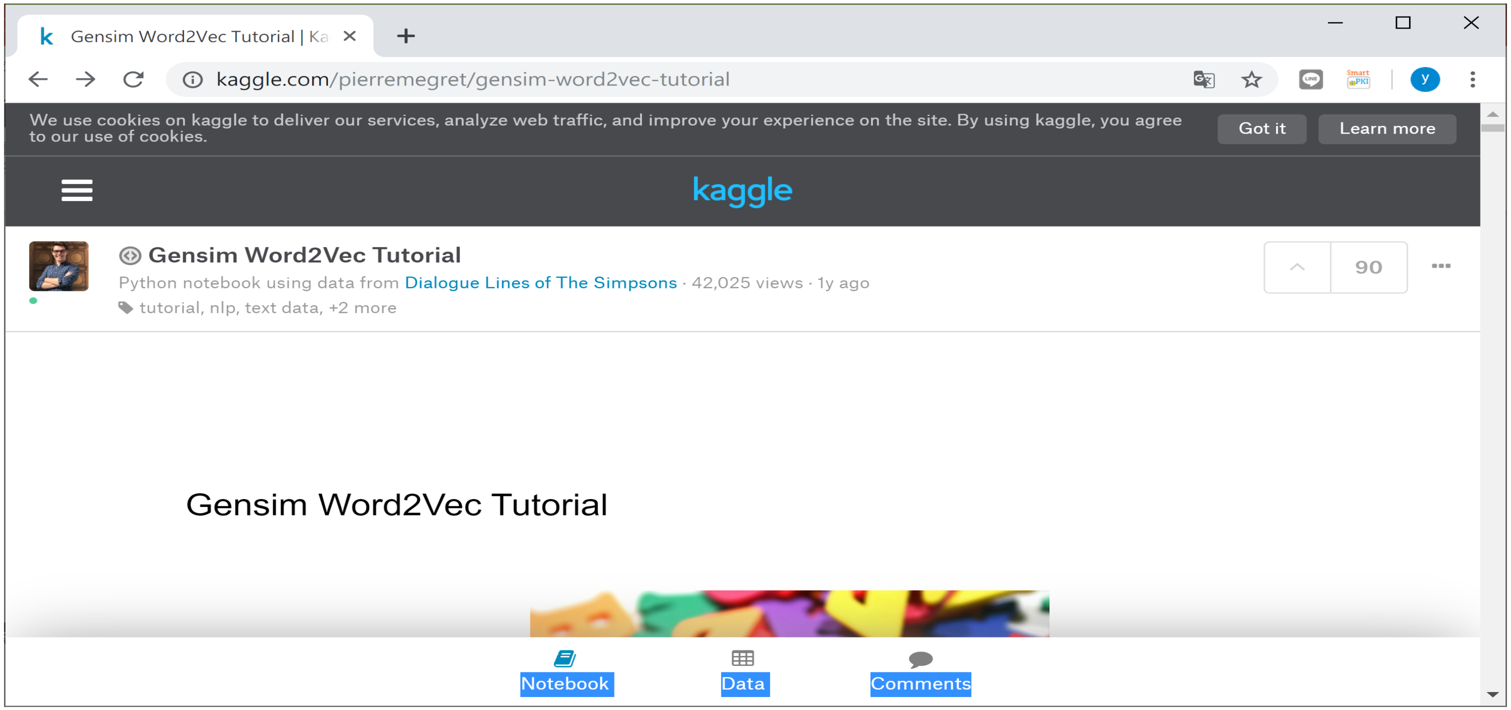
- n -gram model
  - CountVectorizer(ngram_range=(2, 4))
- 解釋說明結果

作業

- 使用 The 在計數與TF-IDF 詞袋差別
  - 詞彙總長度 vs 出現次數
  - 詞彙總長度 vs ?
- 為何取 TFIDF 要取 log 函數
- 說明 The 在三個文件中都有出
  - 文件一: 0.42983
  - 文件二: 0.61336
  - 文件三: 0.61336
- 計算文件一的TF-IDF

作業

- 實作練習



https://www.kaggle.com/pierremegret/gensim-word2vec-tutorial