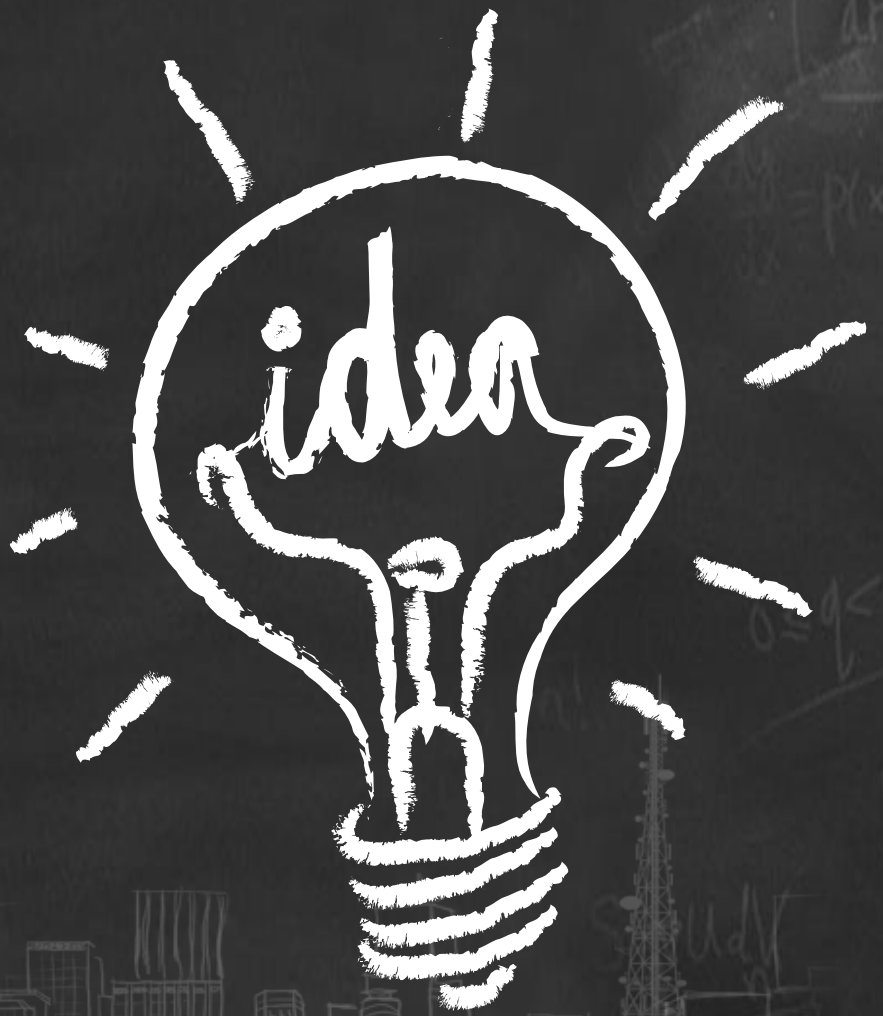
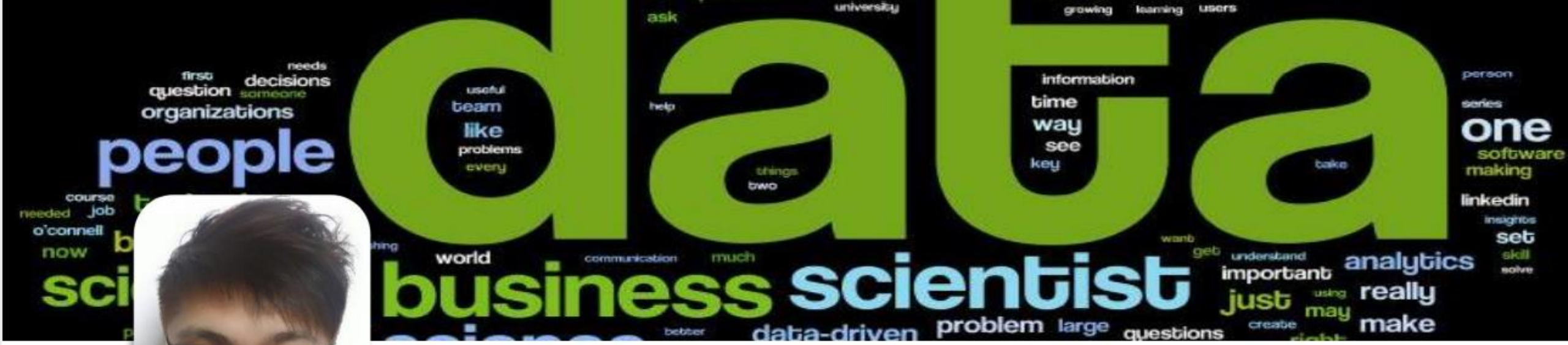
NLP自然語言基礎課程

Fundamental Course in Natural Language Processing (NLP)

Dr. Jia-Wei Chang

jwchang@nutc.edu.tw





Work Experience

- 2018/8 ~ Now
  Associate Professor
  National Taichung University of Science and Technology
- 2018/2 ~ Now
  Adjunct Assistant Professor
  National Cheng Kung University
- 2015/8 ~ 2017/11
  Project Manager & Data Scientist
  NEXCOM International Co., Ltd.

About Me

- [Since Aug. 2021] Founder, IET Oncampus x Taichung Tech
- [Since Jan. 2019] Young Professionals Chair, IET Taipei Local Network.
- [Since Dec. 2017] Consultant, NEXCOM Industry 4.0 Center.
- [Jan. 2017] Ph.D. degree, National Cheng Kung University.

Research Topics

a) Natural Language Processing
   - ✓ Natural Language Understanding
   - ✓ Chatbot
   - ✓ Text Summarization / Classification

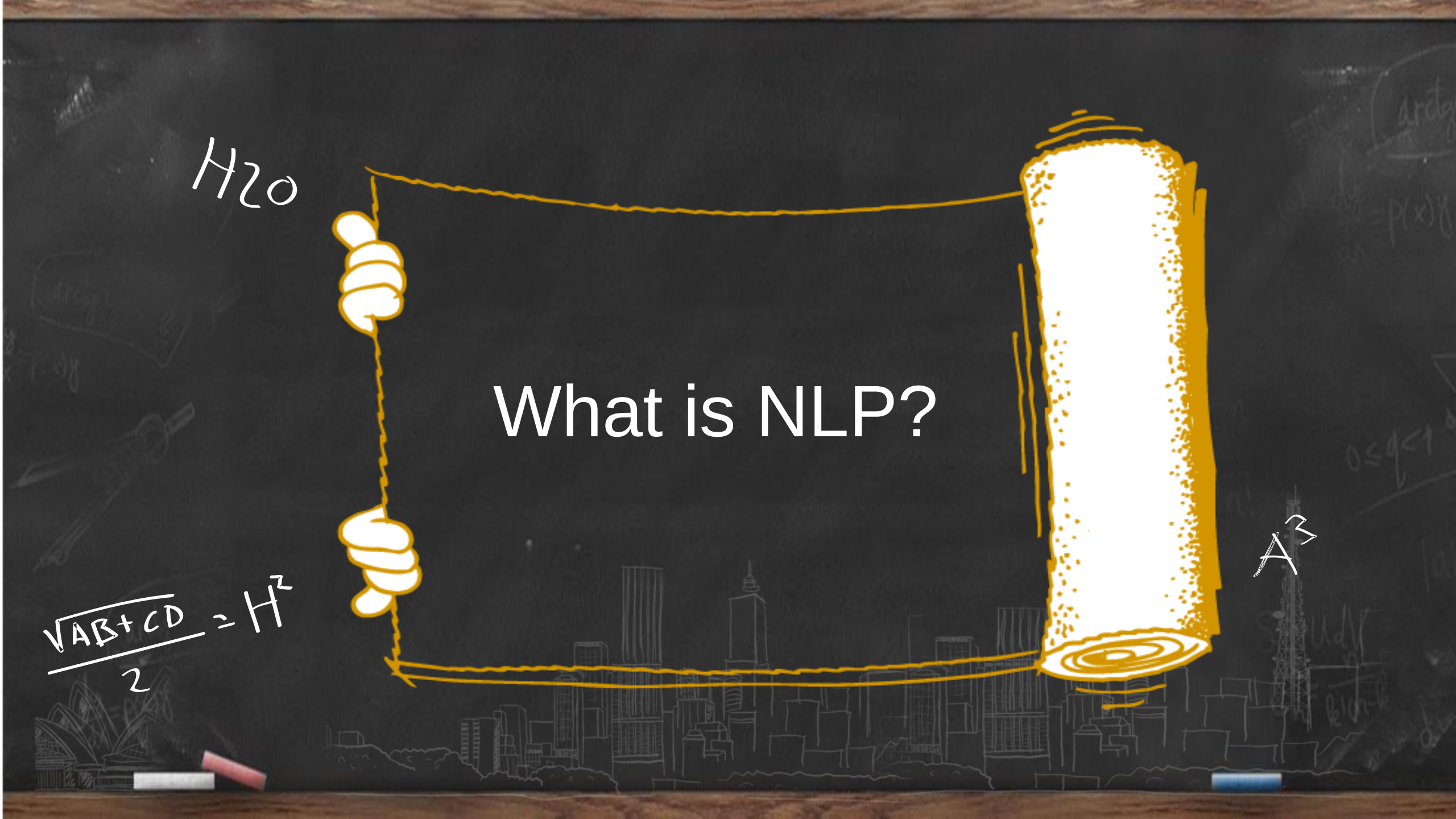
b) Deep Learning

c) Data Mining

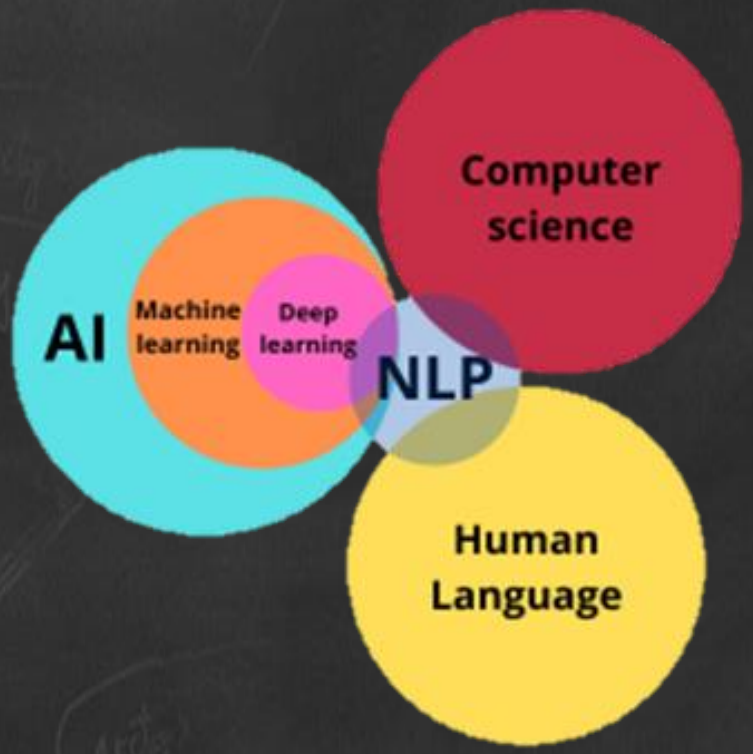
d) Internet of Things
   - ✓ Smart Speaker

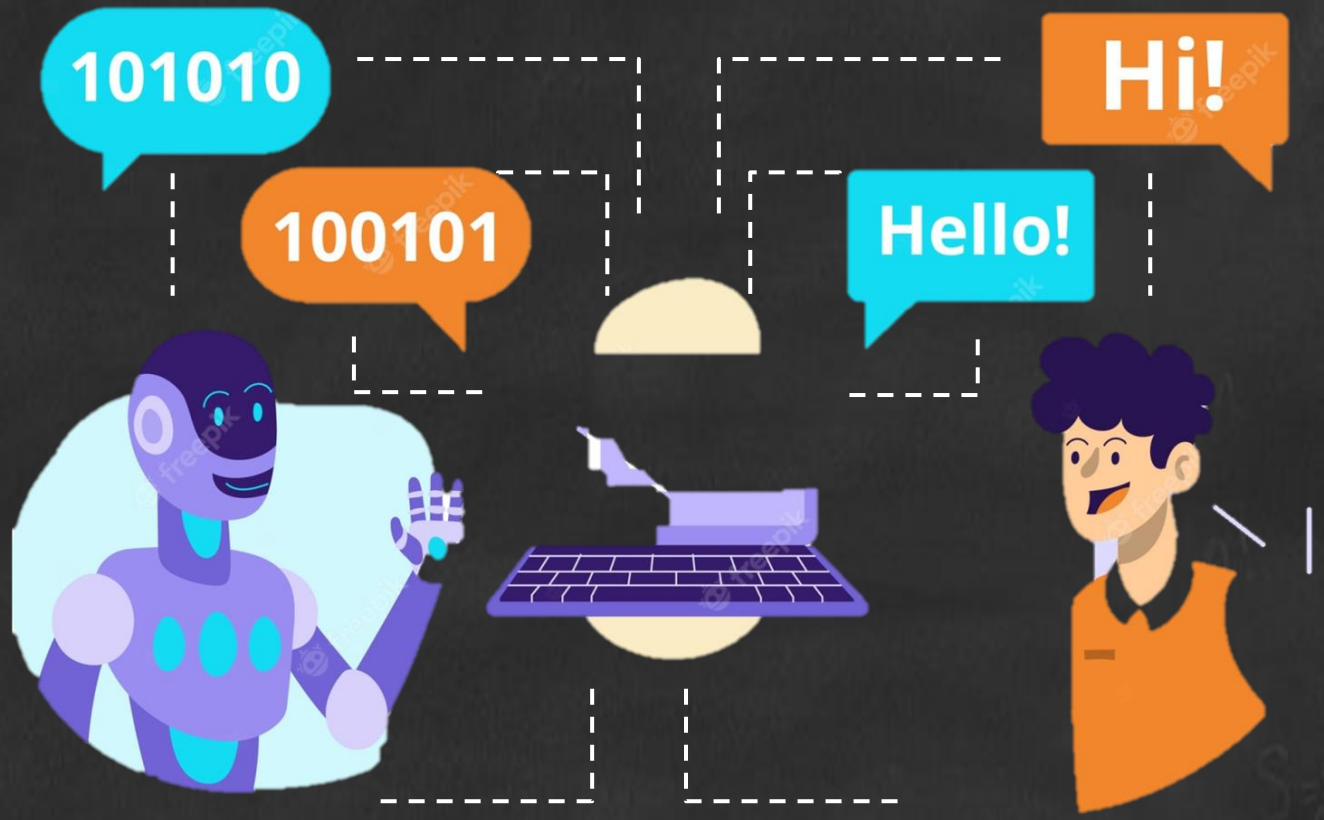
目錄

Contents


What is NLP?

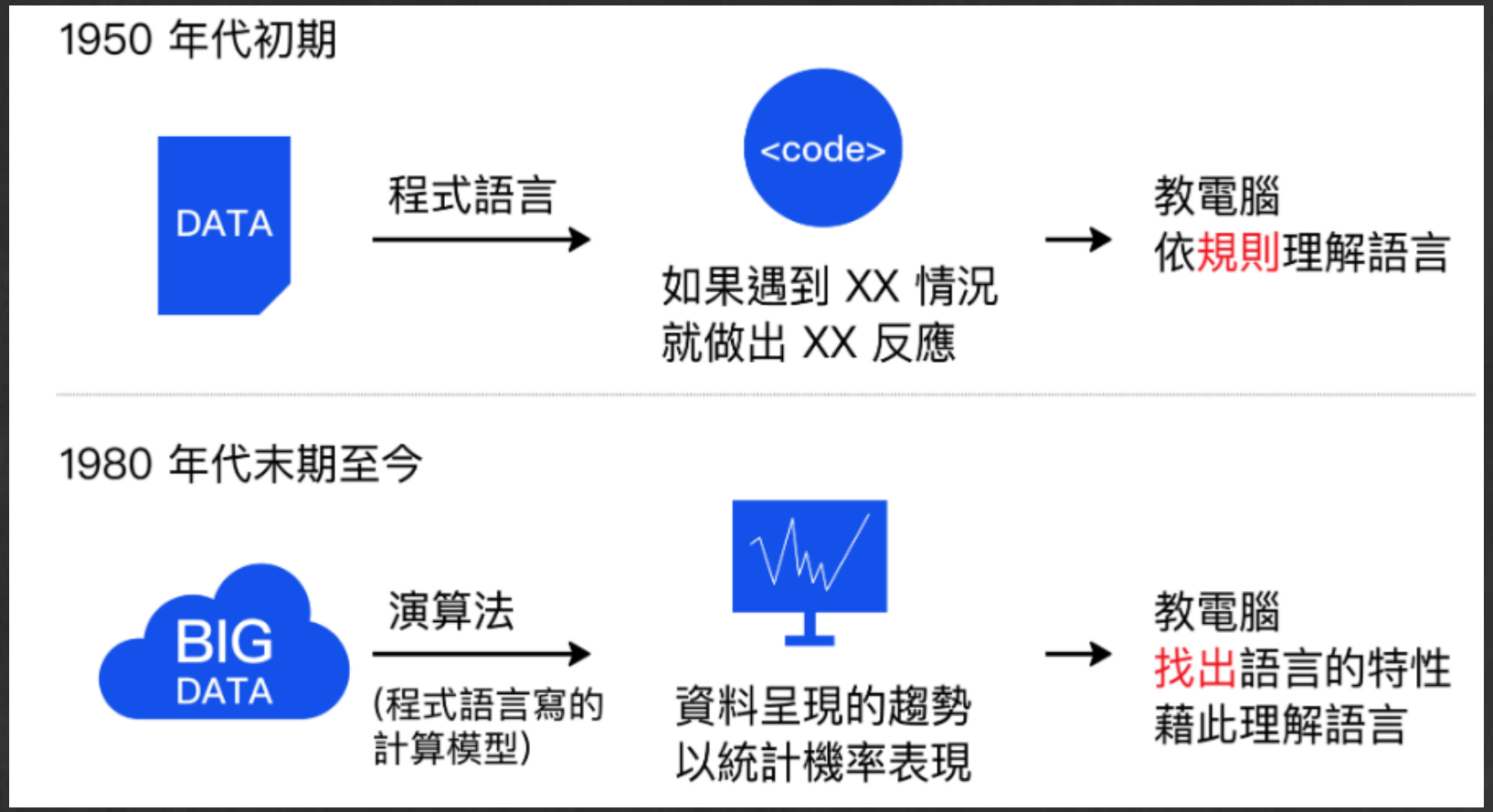
What is NLP?


AI
Machine learning
Deep learning
Computer science
NLP
Human Language


101010
100101
Hi!
Hello!

What is NLP?

自然語言處理的演變



What is NLP?

如何教電腦學語言?

01

斷詞

將句子拆成一個個詞

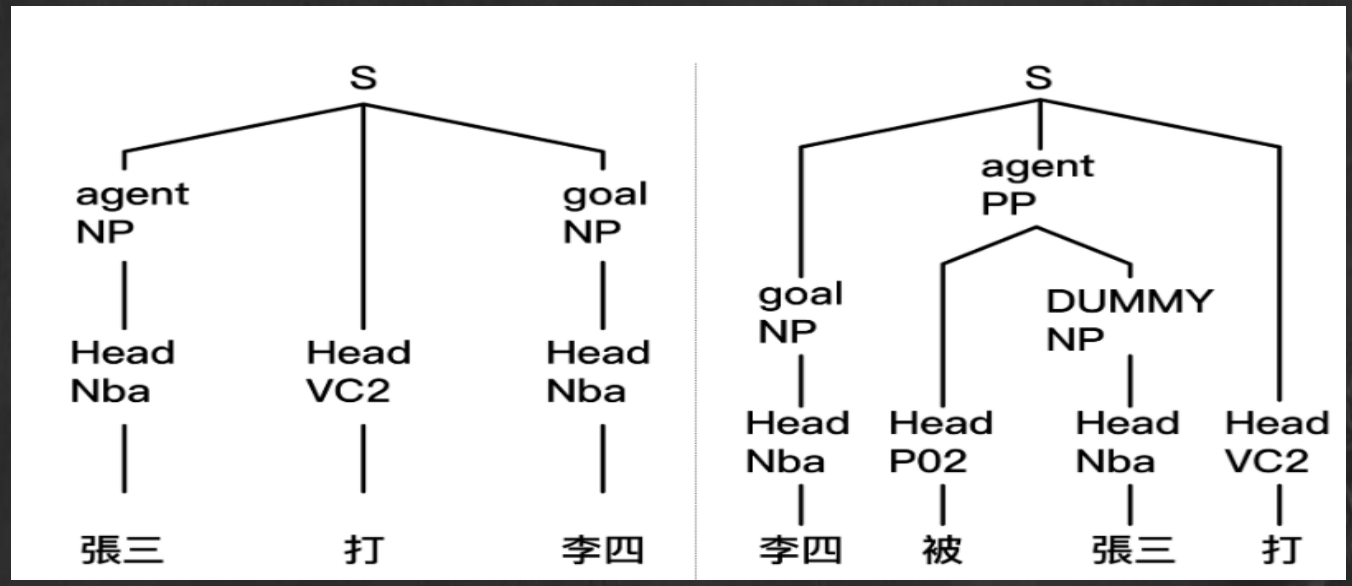
努力才能成功 ⟶ 努力 才 能 成功

他的領導才能很突出 ⟶ 他 的 領導 才能 很 突出

02

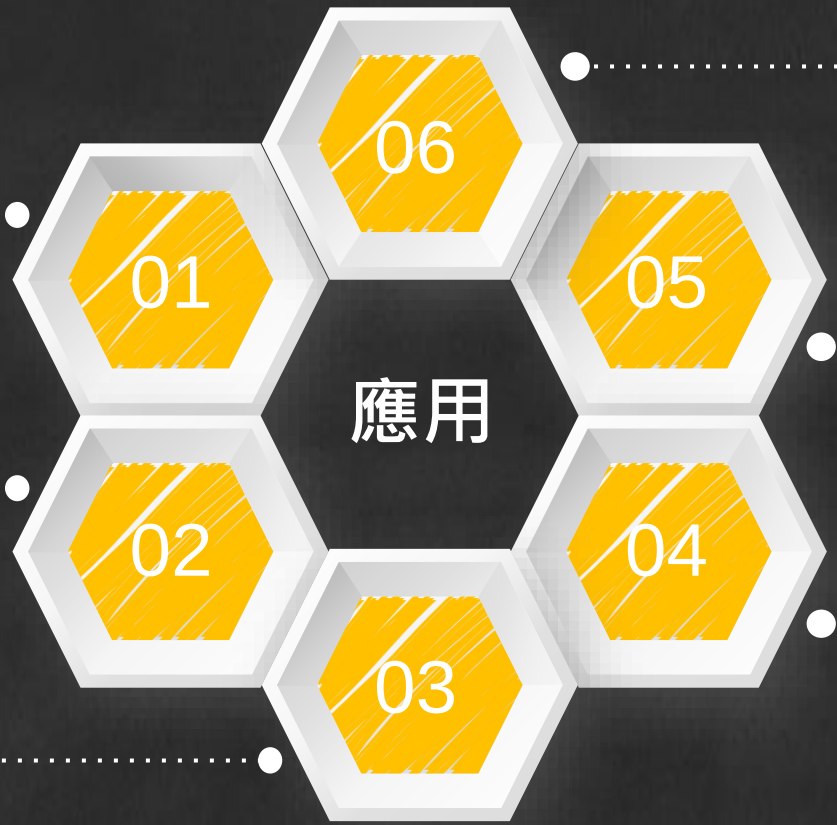
分析句子

使電腦學會理解句子的意思



What is NLP?

常見的NLP應用



聊天機器人

情感分析

文本分類

摘要擷取

機器翻譯

命名實體識別NER


H2O
Challenges and Difficulties in Chinese NLP
$\frac{\sqrt{AB+CD}}{2} = H^2$
A^3

Challenges and Difficulties in Chinese NLP

中文和英文的差別

字符和詞語結構

英文:單詞間以空格分開，詞幹和詞綴也較為明確。

中文:漢字間沒有明確的分隔符號，且詞彙包含多個字符。

01

02

語法結構

英文:語法結構簡單，時態和語氣使用較少。

中文:語法結構靈活，可以有多種結構組成句子，語氣及時態也影響句子。

詞彙多樣性

英文:單詞多義性較少。

中文:相同漢字有不同的意義。

03

04

數據量和資源

英文:全球廣泛使用，資源較多。

中文:使用較少，資源部及英文，但也有不少研究與資源可以使用。

Challenges and Difficulties in Chinese NLP

困難與挑戰

01 分詞和詞義消歧

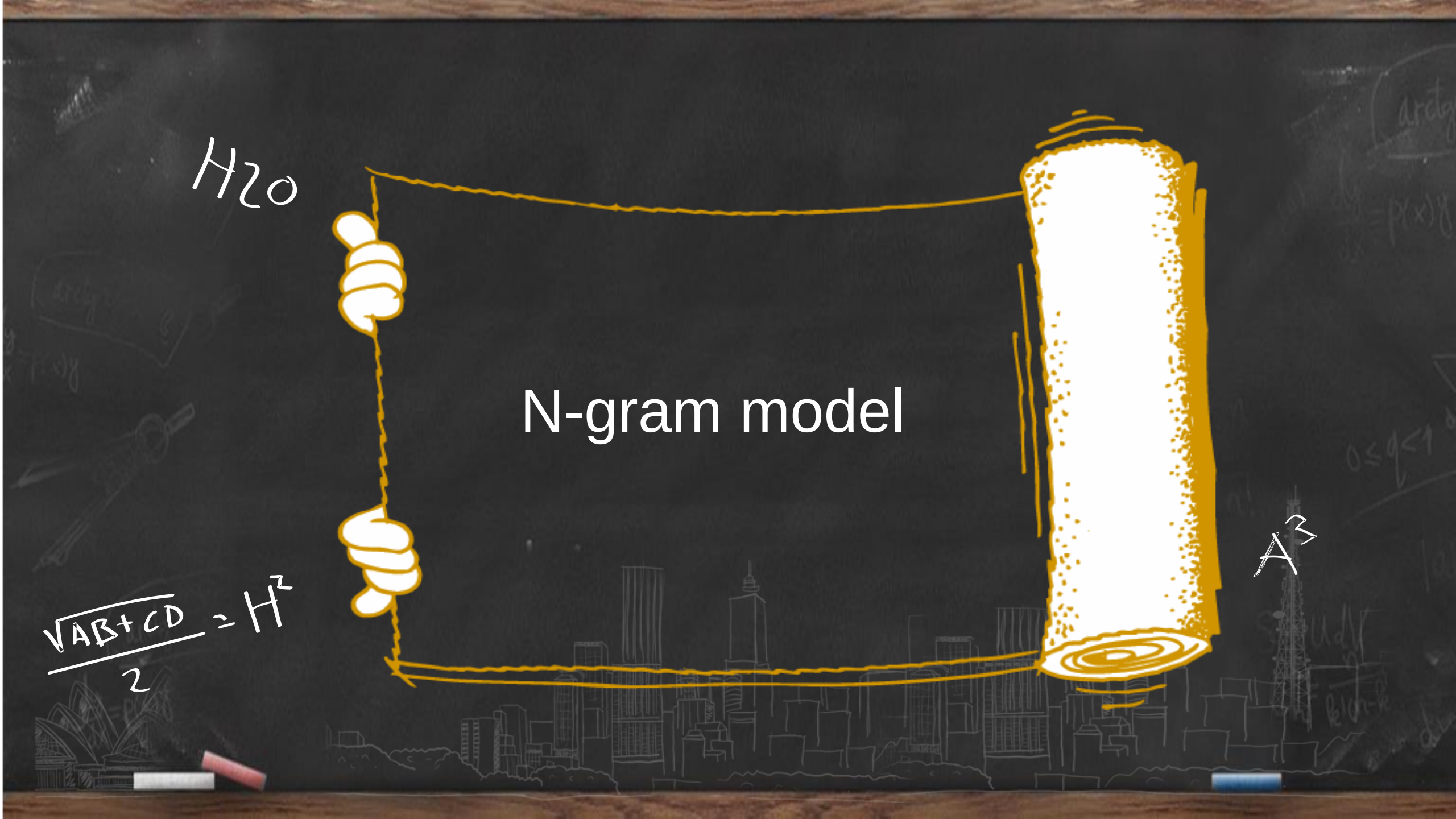
02 多義性

03 語言變體

04 語法結構

05 詞彙多樣性

06 專業領域術語


H2O
N-gram model
√AB+CD / 2 = H²
A³

N-gram model

原理介紹

- N-gram 模型是一種基於統計機率的自然語言處理模型，用於對文本進行建模和預測。它基於一個簡單的假設，即在一個句子或文本中，下一個詞的出現只與前面的 N-1 個詞有關，與整個文本的上下文無關。
- N-gram 模型將文本拆分為一系列的 N 個詞的序列，這些序列被稱為 N-gram。
- 假設N=2，文本為“ChatGPT is a language model”，則會被拆分成:(ChatGPT, is) (is,a) (a,language) (language,model)

N-gram model

原理介紹

unigram	bigram	trigram
你是誰啊	你是誰啊	你是誰啊
[你, 是, 誰, 啊]	[你是, 是誰, 誰啊]	[你是誰, 是誰啊]

N-gram model

原理介紹

- Bigram(2-gram):

$$P(w_n|w_{n-1}) = \frac{Count(w_{n-1}w_n)}{Count(w_{n-1})}$$

- Trigram(3-gram):

$$P(w_n|w_{n-2}, w_{n-1}) = \frac{Count(w_{n-2}w_{n-1}w_n)}{Count(w_{n-2}w_{n-1})}$$

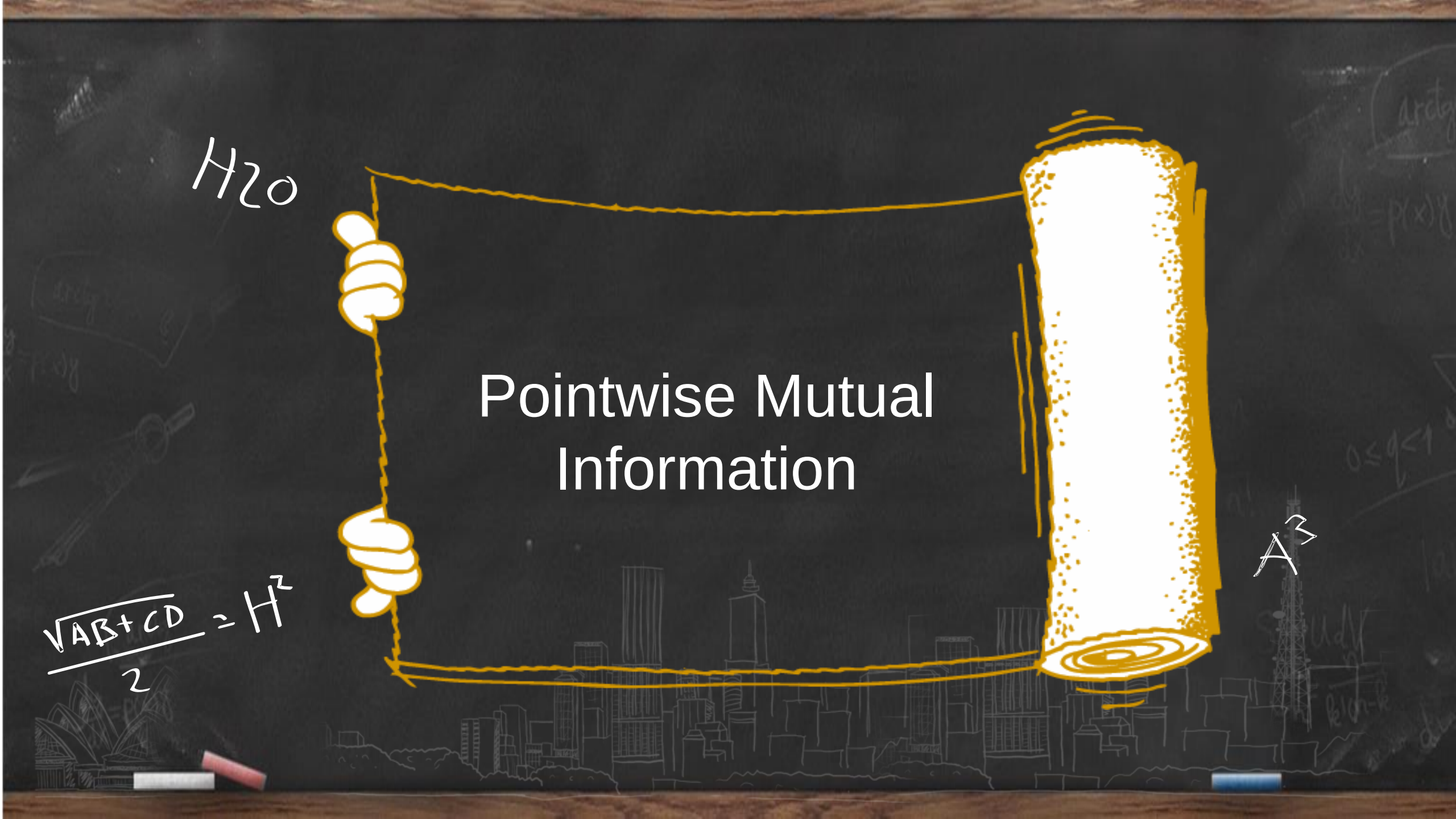
N-gram model

實際應用 pip install scikit-learn

```
# 載入相關套件
from sklearn.feature_extraction.text import CountVectorizer  # 從sklearn套件中載入CountVectorizer

# 語料庫
text = ["orange banana apple grape",
        "banana apple apple",
        "grape",
        "orange apple"]

# N-gram 模型建構
ngram_vectorizer = CountVectorizer(ngram_range=(2, 2), decode_error="ignore")  # 創建一個2-gram的CountVectorizer

# 文本轉化為向量表示並顯示結果
x1 = ngram_vectorizer.fit_transform(text)  # 將文本轉換成2-gram的向量表示
print("print(x1.toarray())運行結果：")
print(x1.toarray())  # 印出轉換後的向量表示
print("print(ngram_vectorizer.vocabulary_)運行結果：")
print(ngram_vectorizer.vocabulary_)  # 印出2-gram的詞彙表
```

```
print(x1.toarray())運行結果：
[[0 1 1 0 1]
 [1 0 1 0 0]
 [0 0 0 0 0]
 [0 0 0 1 0]]
print(ngram_vectorizer.vocabulary_)運行結果：
{'orange banana': 4, 'banana apple': 2, 'apple grape': 1, 'apple apple': 0, 'orange apple': 3}
```

補充：https://practicaldatascience.co.uk/machine-learning/how-to-use-count-vectorization-for-n-gram-analysis


Pointwise Mutual Information

Pointwise Mutual Information

原理介紹

- Pointwise Mutual Information(PMI)是一種用於衡量兩個事件之間相互關聯性的統計量。對於兩個詞w_i和w_j，它們之間的PMI計算如下：
- $PMI(w_i, w_j) = \log(\frac{P(w_i \cap w_j)}{P(w_i)P(w_j)})$

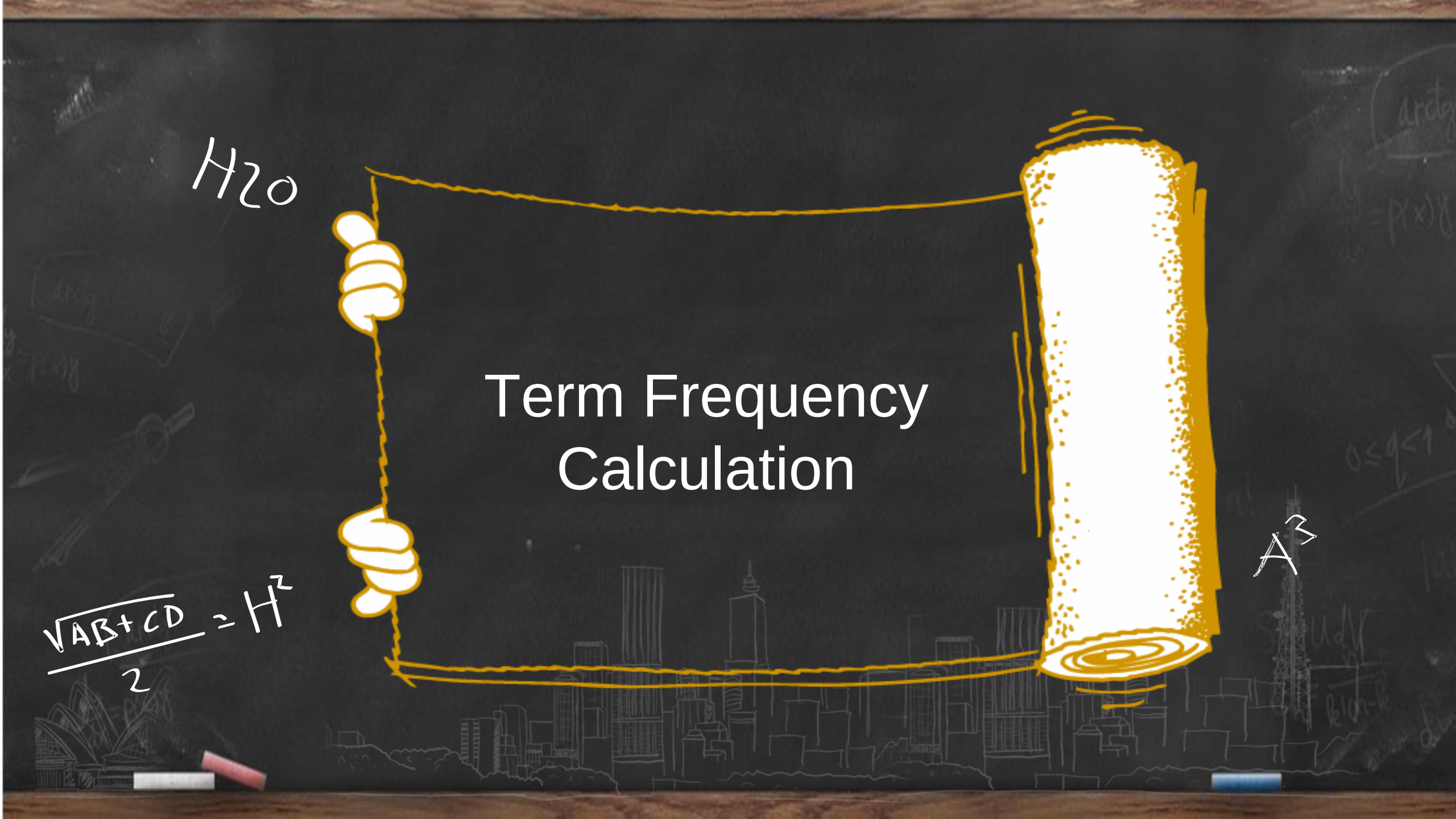
Pointwise Mutual Information

實際應用 pip install nltk

```python
# 載入相關套件
import nltk  # 載入nltk套件
from nltk.collocations import BigramCollocationFinder  # 從nltk套件中載入BigramCollocationFinder

# 文本數據集
corpus = [
    'i like apples and bananas',
    'bananas are yellow',
    'apples are red',
    # 更多文檔...
]

# 處理文本數據，將每個文檔拆分成單詞
tokenized_corpus = [nltk.word_tokenize(doc) for doc in corpus]  # 對每份文檔進行斷詞處理

# 創建BigramCollocationFinder對象
finder = BigramCollocationFinder.from_documents(tokenized_corpus)  # 使用斷詞後的文本創建BigramCollocationFinder對象

# 計算PMI值
bigram_measures = nltk.collocations.BigramAssocMeasures()  # 創建BigramAssocMeasures對象
pmi_scores = finder.score_ngrams(bigram_measures.pmi)  # 計算bigram的PMI值

# 打印前N個PMI值最高的bigram
N = 8
print("運行結果：")
for bigram, pmi in pmi_scores[:N]:  # 印出前N個PMI值最高的bigram
    print(f"PMI({bigram}): {pmi:.2f}")  # 印出bigram及其對應的PMI值
```

運行結果：

```
PMI(('i', 'like')): 3.46
PMI(('and', 'bananas')): 2.46
PMI(('apples', 'and')): 2.46
PMI(('are', 'red')): 2.46
PMI(('are', 'yellow')): 2.46
PMI(('like', 'apples')): 2.46
PMI(('apples', 'are')): 1.46
PMI(('bananas', 'are')): 1.46
```


H2O
Term Frequency
Calculation
√AB+CD / 2 = H²
A³

Term Frequency Calculation

TF-IDF（Term Frequency-Inverse Document Frequency）

- TF-IDF是一種用於評估一個詞彙在一份文檔中的重要性的統計方法。
  - TF（Term Frequency）詞頻是指一個詞在一份文檔中出現的次數。通常情況下，一個詞在一份文檔中出現的次數越多，它在該文檔中的重要性越高。

$$TF(t,d) = \frac{n_{t,d}}{\sum_{w \in d} n_{w,d}}$$

  - IDF（Inverse Document Frequency）逆文檔頻率衡量了一個詞在整個文檔集合中的重要性。如果一個詞在很多文檔中都出現，它可能對區分這些文檔的重要性就不高。

$$IDF(t,D) = \log\left(\frac{N}{|\{d \in D : t \in d\}|}\right)$$

Term Frequency Calculation

TF-IDF（Term Frequency-Inverse Document Frequency）

- TF-IDF將TF和IDF進行結合，得到一個詞彙在文檔中的綜合重要性分數。高詞頻（TF）且在整個文檔集合中很少出現（IDF高）的詞將具有較高的TF-IDF分數。

$$TF - IDF(t, d, D) = TF(t, d)\ X\ IDF(t, D)$$

Term Frequency Calculation

實際應用

```
# 載入相關套件
from sklearn.feature_extraction.text import TfidfVectorizer # 從sklearn套件中載入TfidfVectorizer
import pandas as pd # 載入pandas套件，用於資料處理

# 語料庫
corpus = [
    "This is the first document.",
    "This document is the second document.",
    "And this is the third one.",
    "Is this the first document?",
]

# 計算TF-IDF值
vectorizer = TfidfVectorizer() # 初始化TfidfVectorizer
X = vectorizer.fit_transform(corpus) # 對語料庫進行TF-IDF計算，得到特徵矩陣X

# 將特徵矩陣轉換成DataFrame
data = {'word': vectorizer.get_feature_names_out(), # 取得特徵詞列表
        'tfidf': X.toarray().sum(axis=0).tolist()} # 計算每個詞的TF-IDF值
df = pd.DataFrame(data) # 創建DataFrame

# 根據TF-IDF值降序排序
df_sorted = df.sort_values(by='tfidf', ascending=False) # 根據tfidf列降序排序

# 輸出運行結果
print("運行結果：")
print(df_sorted) # 印出排序後的DataFrame
```

運行結果：

	word	tfidf
0	and	0.511849
1	document	1.627206
2	first	1.160572
3	is	1.316363
4	one	0.511849
5	second	0.538648
6	the	1.316363
7	third	0.511849
8	this	1.316363


H2O
Data
preprocessing
√AB+CD / 2 = H²
A³

Data preprocessing

資料準備

- Data preprocessing and cleaning
  - 預處理資料可以減少雜訊並處理缺失值
  - 斷字、斷詞
- Relevance analysis(feature selection)
  - 刪除不相關或多餘的屬性
  - 移除stop words,擷取有用資訊(TF-IDF)
- Data transformation
  - Generalize and/or normalize data
  - 轉成向量(Vector representation)

Data preprocessing

文字分割

- 斷詞/斷句
- 英文的斷詞:
  - 可以直接利用空格來分割單詞
  - EX:I love machine learning. → [I, love, machine, learning]
- 中文的斷詞:
  - 通過有意義的單詞來分割單詞，而不是直接利用字來分割
  - EX:我喜歡機器學習→ [我, 喜歡, 機器, 學習]

Data preprocessing

Dear 小明,
這是目前公司的最新技術，利用apples和pens的特性可以讓產能最佳化……

Dear, 小明, 這是, 目前, 公司, 的, 最新, 技術, 利用, apples, 和, pens, 的, 特性, 可以, 讓, 產能, 最佳化, …….

Data preprocessing

中文常用斷詞套件

- Jieba
  - Jieba是一個Python中文斷詞套件，廣泛用於中文NLP的領域。它具有高效、穩定且容易使用的特點。
- 特點：
  - 支援基於字典的準確斷詞和基於統計的全模式、精確模式、搜索引擎模式。
  - 支援繁體中文和簡體中文的斷詞。
  - 可以進行自定義詞典的添加，以滿足特定領域的需求。
  - 提供了詞性標註功能。

Data preprocessing

實際應用 pip install jieba

```python
import jieba

# 全模式
seg_list = jieba.cut("我来到北京清华大学", cut_all=True)
print("Full Mode: " + "/ ".join(seg_list))
# 精确模式
seg_list = jieba.cut("我来到北京清华大学", cut_all=False)
print("Default Mode: " + "/ ".join(seg_list))

# 默认是精确模式
seg_list = jieba.cut("他来到了网易杭研大厦")
print(", ".join(seg_list))

 # 搜索引擎模式
seg_list = jieba.cut_for_search("小明硕士毕业于中国科学院计算所，后在日本京都大学深造")
print(", ".join(seg_list))
```

```
Full Mode: 我/ 来到/ 北京/ 清华/ 清华大学/ 华大/ 大学
Default Mode: 我/ 来到/ 北京/ 清华大学
他, 来到, 了, 网易, 杭研, 大厦
小明, 硕士, 毕业, 于, 中国, 科学, 学院, 科学院, 中国科学院, 计算, 计算所, ，, 后, 在, 日本, 京都, 大学, 日本京都大学, 深造
```

Data preprocessing

中文常用斷詞套件

- CKIP
  - CKIP是由中研院資訊所推出的中文斷詞系統，包括斷詞、詞性標註、命名實體識別等功能，以及相應的Python套件。
- 特點：
  - 提供了高準確度的繁體中文斷詞、詞性標註、命名實體識別等功能。
  - 提供線上實作系統，連結: https://ckip.iis.sinica.edu.tw/service/transformers/

Data preprocessing

實際應用 pip install –U ckip-transformers

```
# -*- coding: utf-8 -*- # 設定編碼方式為UTF-8
# 載入相關套件
from ckip_transformers import __version__  # 載入ckip_transformers套件的版本信息
from ckip_transformers.nlp import CkipWordSegmenter, CkipPosTagger, CkipNerChunker  # 從ckip_transformers.nlp中載入相應模組
# 初始化 drivers
print("Initializing drivers ... WS")  # 印出初始化Word Segmenter的訊息
ws_driver = CkipWordSegmenter(model="albert-base")  # 初始化Word Segmenter
print("Initializing drivers ... POS")  # 印出初始化Pos Tagger的訊息
pos_driver = CkipPosTagger(model="albert-base")  # 初始化Pos Tagger
def clean(sentence_ws, sentence_pos):  # 定義一個名為clean的函數，用於保留主要詞彙
  short_with_pos = []  # 初始化一個空串列short_with_pos，用於儲存含有詞性的短詞
  short_sentence = []  # 初始化一個空串列short_sentence，用於儲存短詞
  stop_pos = set(['Nep', 'Nh', 'Nb']) # 定義一個包含特定詞性的集合stop_pos
  # 逐詞進行處理
  for word_ws, word_pos in zip(sentence_ws, sentence_pos):
    # 只留名詞和動詞
    is_N_or_V = word_pos.startswith("V") or word_pos.startswith("N")  # 判斷是否為名詞或動詞
    # 去掉名詞裡的某些詞性
    is_not_stop_pos = word_pos not in stop_pos  # 判斷是否為特定詞性
    # 只剩一個字的詞也不留
    is_not_one_charactor = not (len(word_ws) == 1)  # 判斷是否只有一個字母
    # 符合條件的詞加入串列
    if is_N_or_V and is_not_stop_pos and is_not_one_charactor:
      short_with_pos.append(f"{word_ws}({word_pos})")  # 將短詞和詞性以特定格式加入short_with_pos串列
      short_sentence.append(f"{word_ws}")  # 將短詞加入short_sentence串列
  return (" ".join(short_sentence), " ".join(short_with_pos))  # 返回處理後的短詞和含詞性的短詞
def main():
    #語料庫
    text = [
        '經過多年激烈戰事，複製人大戰即將結束。絕地議會派歐比王將導致戰亂的主謀者繩之以法；不料，西斯勢力已悄悄深入銀河系，勢力漸大的議長白卜庭用黑暗勢力的力量，誘惑天行者安納金轉變成黑武士達斯維達，幫助他達成心願建立銀河帝國，剷除絕地武士…【星際大戰】系列電影最後一塊拼圖，喬治盧卡斯不僅要解開黑武士的影壇跨世紀謎團，更要著手打造影史最大星際戰爭。',
    ]
    ws = ws_driver(text)  # 進行斷詞
    pos = pos_driver(ws)  # 進行詞性標注
    print()  # 印出空行，增加可讀性
    for sentence, sentence_ws, sentence_pos in zip(text, ws, pos):  # 逐句進行處理
        print("原文：")
        print(sentence)  # 印出原文
        (short, res) = clean(sentence_ws, sentence_pos)  # 呼叫clean函數進行處理
        print("斷詞後：")
        print(short)  # 印出斷詞後的短詞
        print("斷詞後+詞性標注：")
        print(res)  # 印出含詞性的短詞
        print('='*50)  # 印出分隔線
if __name__ == "__main__":
    main()  # 呼叫main函數執行主程序
```

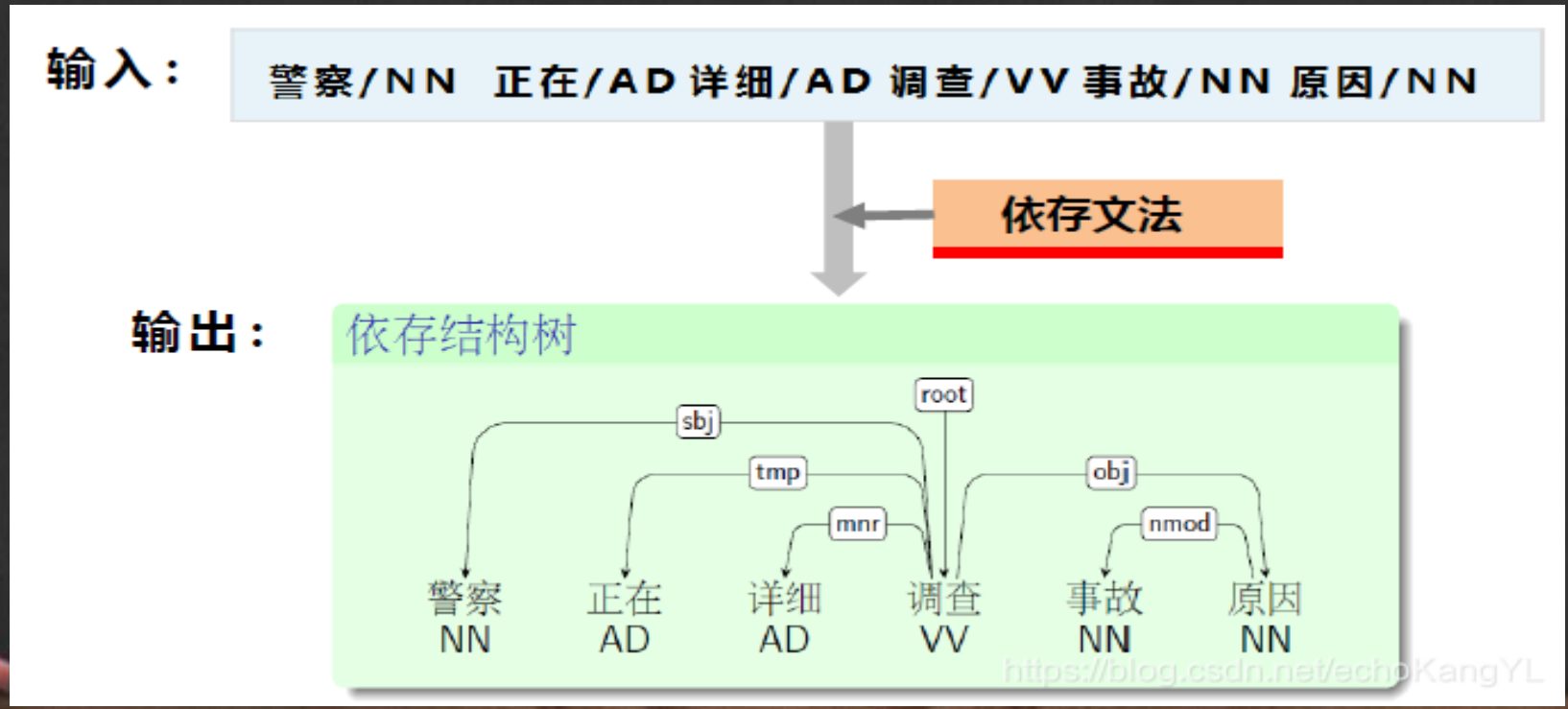
```
原文：
經過多年激烈戰事，複製人大戰即將結束。絕地議會派歐比王將導致戰亂的主謀者繩之以法；不料，西斯勢力已悄悄深入銀河系，勢力漸大的議長白卜庭用黑暗勢力的力量，誘惑天行者安納金轉變成黑武士達斯維達，幫助他達成心願建立銀河帝國，剷除絕地武士…【星際大戰】系列電影最後一塊拼圖，喬治盧卡斯不僅要解開黑武士的影壇跨世紀謎團，更要著手打造影史最大星際戰爭。
斷詞後：
經過 激烈 戰事 複製人 大戰 結束 絕地 議會 派歐比王將 導致 戰亂 主謀 繩之以法 勢力 深入 銀河系 勢力 議長 黑暗 勢力 力量 誘惑 天行 轉變成 黑武士 幫助 達成 心願 建立 銀河 帝國 剷除 絕地 武士 星際 大戰 系列 電影 最後 拼圖 解開 黑武士 影壇 世紀 謎團 著手 打造 影史 星際 戰爭
斷詞後+詞性標注：
經過(VCL) 激烈(VH) 戰事(Na) 複製人(Na) 大戰(Na) 結束(VHC) 絕地(VJ) 議會(Nc) 派歐比王將(VF) 導致(VL) 戰亂(Na) 主謀(Na) 繩之以法(VB) 勢力(Na) 深入(VCL) 銀河系(Nc) 勢力(Na) 議長(Na) 黑暗(Na) 勢力(Na) 力量(Na) 誘惑(VC) 天行(Na) 轉變成(VG) 黑武士(VH) 幫助(VC) 達成(VC) 心願(Na) 建立(VC) 銀河(Na) 帝國(Na) 剷除(VC) 絕地(Na) 武士(Na) 星際(Nc) 大戰(Na) 系列(Na) 電影(Na) 最後(Nd) 拼圖(Na) 解開(VC) 黑武士(VH) 影壇(Nc) 世紀(Na) 謎團(Na) 著手(VF) 打造(VC) 影史(Na) 星際(Nc) 戰爭(Na)
```


H2O
Syntax Tree
√AB+CD / 2 = H²
A³

Syntax Tree

原理介紹

- 語法樹（Syntax Tree），也稱為句法樹（Parse Tree）或結構樹（Parse Tree），它是一種用於表示句子結構的樹狀結構，用於分析句子的語法結構和詞彙之間的關係。



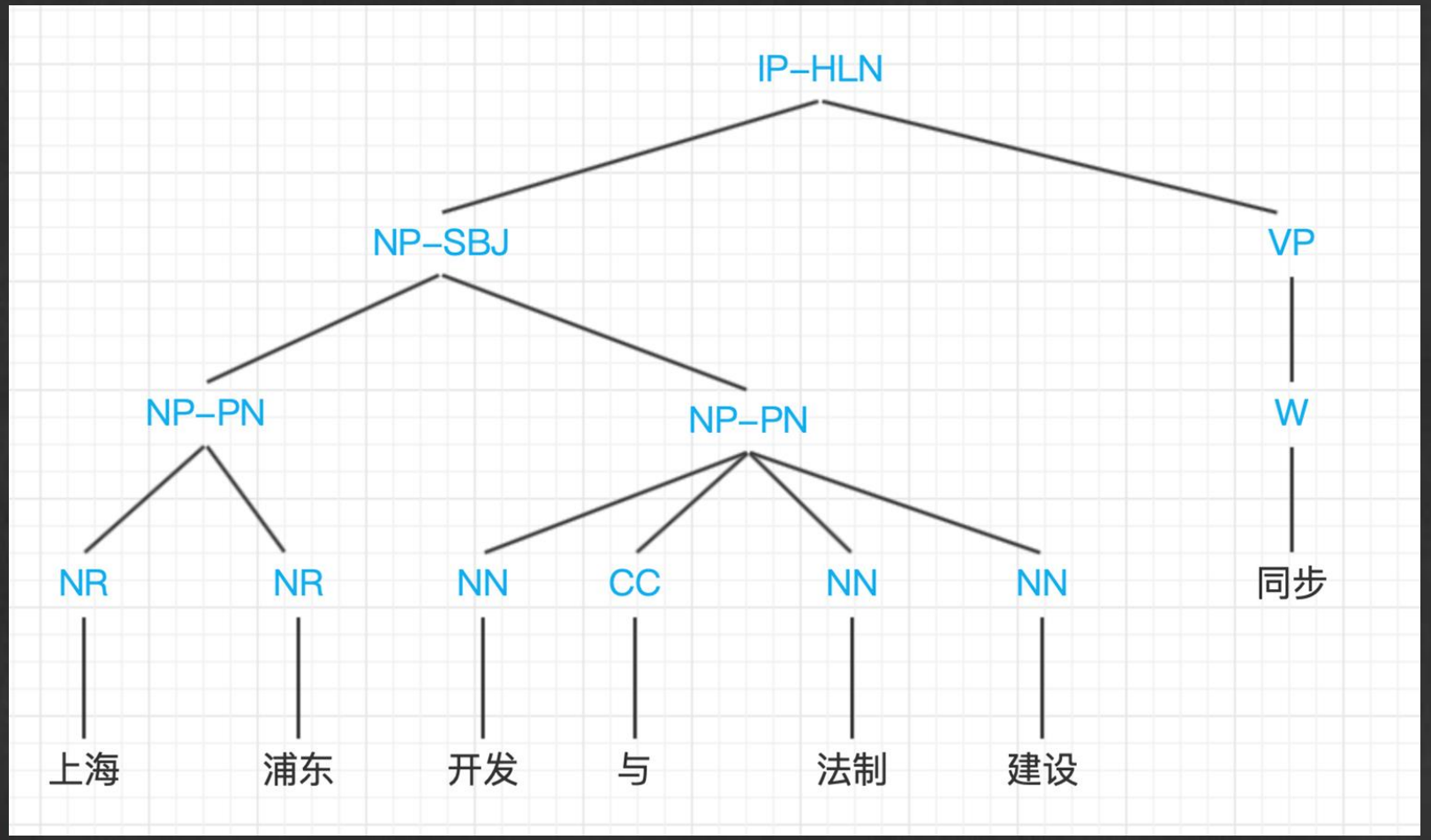
Syntax Tree

原理介紹

- 結構和組成成分：
  - 節點（Node）：每個節點代表了句子中的一個詞或一個組合的詞。這些節點包括詞彙節點和非詞彙節點。
  - 邊（Edge）：邊表示節點之間的關係，通常用於連接不同節點。
  - 根節點（Root Node）：整棵樹的最頂層節點，它代表了整個句子。
  - 子節點（Child Node）：一個節點下面連接的較低層級的節點稱為子節點。
  - 父節點（Parent Node）：一個節點連接到上面的節點稱為父節點。

Syntax Tree

原理介紹



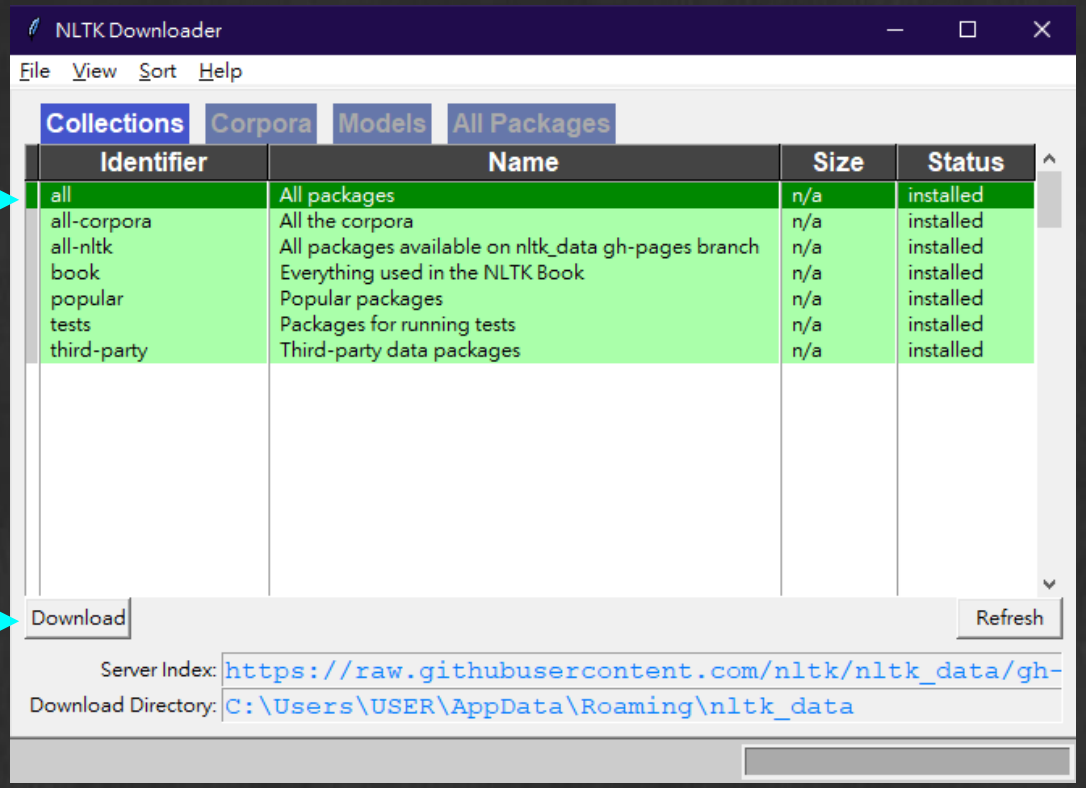
Syntax Tree

實際應用

```
import nltk
nltk.download()
```

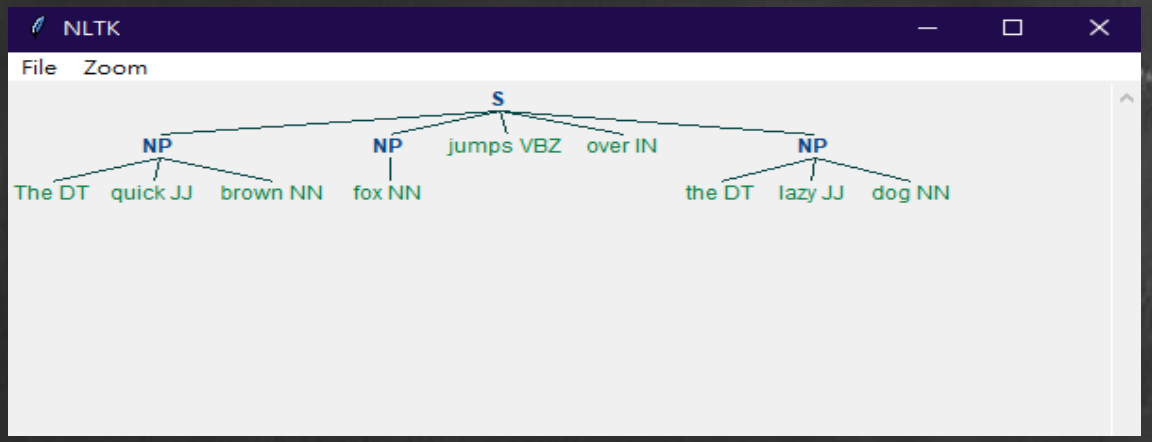
1.選擇all

2.按下 Download



Syntax Tree

實際應用




```python
import nltk  # 載入nltk套件
from nltk import pos_tag  # 從nltk中載入pos_tag函數，用於詞性標註
from nltk import RegexpParser  # 從nltk中載入RegexpParser，用於正則表達式句法分析

# 定義文本
text = "The quick brown fox jumps over the lazy dog"  # 定義一段文本

# 斷詞&詞性標註
tokens = nltk.word_tokenize(text)  # 使用nltk進行斷詞
tagged_tokens = pos_tag(tokens)  # 使用pos_tag進行詞性標註

# 定義文法
grammar = "NP: {<DT>?<JJ>?<NN>}"  # 定義一個名為grammar的文法，用於句法分析

# 初始化句法分析器
chunk_parser = RegexpParser(grammar)  # 使用定義的文法初始化句法分析器

# 進行句法分析
parse_tree = chunk_parser.parse(tagged_tokens)  # 使用句法分析器進行句法分析

# 繪製句法樹
parse_tree.draw()  # 繪製句法樹
```


H2O
Named Entity Recognition
√AB+CD / 2 = H²
A³

Named Entity Recognition

原理介紹

- 命名實體識別（Named Entity Recognition, NER）的目標是從文本中識別和區分出具有特定意義的實體，這些實體可以是名字、地點、組織、日期、時間、數字等。

- NER 的主要目標：
  - 識別命名實體：在文本中找到並識別出具有特定意義的實體，例如人名、地名、組織名等。
  - 區分實體類型：將識別出的實體進行分類，例如將人名歸為人物、地名歸為地點等。
  - 消除歧義：在一個句子中，相同的詞可能指代不同的實體，NER 的目標之一是解決這種歧義。
  - 提供上下文：對於識別出的實體，NER 常常會提供相關的上下文信息，以幫助理解這個實體在文本中的角色和意義。
  - 支援信息擷取：NER 的結果可以用於從文本中提取出特定信息，例如識別出的人名可以用於建立聯繫人列表。

Named Entity Recognition

實際應用

```
# -*- coding: utf-8 -*- # 設定編碼方式為UTF-8

# 載入相關套件
from ckip_transformers.nlp import CkipWordSegmenter, CkipNerChunker  # 從ckip_transformers.nlp中載入相應模組
# 初始化 drivers
print("Initializing drivers ... WS")  # 印出初始化Word Segmenter的訊息
ws_driver = CkipWordSegmenter(model="albert-base")  # 初始化Word Segmenter
print("Initializing drivers ... NER")  # 印出初始化NER Chunker的訊息
ner_driver = CkipNerChunker(model="albert-base")  # 初始化NER Chunker
print("Initializing drivers ... all done")  # 印出初始化完成的訊息
print()  # 印出空行，增加可讀性
def clean(sentence_ws): #定義一個名為clean的函數，用於清洗斷詞結果
  short_sentence = []  # 初始化一個空串列short_sentence，用於儲存短詞
  for word_ws in sentence_ws:  # 逐詞進行處理
    # 只剩一個字的詞也不留
    is_not_one_charactor = not (len(word_ws) == 1)  # 判斷是否只有一個字母
    # 符合條件的詞加入串列
    if is_not_one_charactor:
      short_sentence.append(f"{word_ws}")  # 將短詞加入short_sentence串列
  return (" ".join(short_sentence))  # 返回處理後的短詞
def main():
    text = [
        '經過多年激烈戰事，複製人大戰即將結束。絕地議會派歐比王將導致戰亂的主謀者繩之以法；不料，西斯勢力已悄悄深入銀河系，勢力漸大的議長白卜庭用黑暗勢力的力量，誘惑
行者安納金轉變成黑武士達斯維達，幫助他達成心願建立銀河帝國，剷除絕地武士…【星際大戰】系列電影最後一塊拼圖，喬治盧卡斯不僅要解開黑武士的影壇跨世紀謎團，更要著手打造影史
最大星際戰爭。',
    ]
    ws = ws_driver(text)  # 進行斷詞
    ner = ner_driver(text)  # 進行NER
    print()  # 印出空行，增加可讀性
    print('=====')  # 印出分隔線
    for sentence, sentence_ws, sentence_ner in zip(text, ws, ner):  # 逐句進行處理
        print("原文：")
        print(sentence)  # 印出原文
        short = clean(sentence_ws)  # 呼叫clean函數進行處理
        print("斷詞後：")
        print(short)  # 印出斷詞後的短詞
        print("NER：")
        print(sentence_ner)  # 印出NER結果
        print('=====')  # 印出分隔線
if __name__ == "__main__":
    main()  # 呼叫main函數執行主程序
```

Named Entity Recognition

執行結果

```
原文：
經過多年激烈戰事，複製人大戰即將結束。絕地議會派歐比王將導致戰亂的主謀者繩之以法；不料，西斯勢力已悄悄深入銀河系，勢力漸大的議長白卜庭用黑暗勢力的力量，誘惑天行者安納金轉變成黑武士達斯維達，幫助他達成心願建立銀河帝國，剷除絕地武士…【星際大戰】系列電影最後一塊拼圖，喬治盧卡斯不值要解開黑武士的影壇跨世紀謎團，更要著手打造影史最大星際戰爭。
斷詞後：
經過 激烈 戰事 複製人 大戰 即將 結束 絕地 議會 派歐比王將 導致 戰亂 主謀 繩之以法 不料 西斯 勢力 悄悄 深入 銀河系 勢力 議長 白卜庭 黑暗 勢力 力量 誘惑 天行 安納金 轉變成 黑武士 達斯維達 幫助 達成 心願 建立 銀河 帝國 剷除 絕地 武士 星際 大戰 系列 電影 最後 拼圖 喬治盧卡斯 不值 解開 黑武士 影壇 世紀 謎團 著手 打造 影史 星際 戰爭
NER：
[NerToken(word='多年', ner='DATE', idx=(2, 4)), NerToken(word='歐比王', ner='PERSON', idx=(24, 27)), NerToken(word='西斯', ner='PERSON', idx=(44, 46)), NerToken(word='銀河系', ner='ORG', idx=(53, 56)), NerToken(word='白卜庭',
 ner='PERSON', idx=(64, 67)), NerToken(word='安納金', ner='PERSON', idx=(81, 84)), NerToken(word='銀河帝國', ner='GPE', idx=(104, 108)), NerToken(word='喬治盧卡斯', ner='PERSON', idx=(133, 138))]
```

Named Entity Recognition

CKIP-NER標記介紹

Name	Description
CARDINAL	數字
DATE	日期
EVENT	事件
FAC	設施
GPE	行政區
LANGUAGE	語言
LAW	法律
LOC	地理區
MONEY	金錢
NORP	民族、宗教、政治團體
ORDINAL	序數
ORG	組織
PERCENT	百分比率
PERSON	人物
PRODUCT	產品
QUANTITY	數量
TIME	時間
WORK_OF_ART	作品


感謝你的耐心聆聽