1

บทนำ

1.1 ไพธอนกับการประมวลผลภาษา

1.2 การประมวลผลคำและข้อความ

1.3 สถิติที่เกี่ยวข้องกับการประมวลผลภาษา

วัตถุประสงค์

- รู้จักประโยชน์ของเทคนิคการเขียนโปรแกรมเบื้องต้นที่เพื่อจัดการกับเอกสารข้อความขนาดใหญ่
- รู้จักเครื่องมือและเทคนิคในภาษาไพธอนเพื่อจัดการกับเอกสารข้อความขนาดใหญ่ และการสกัดคำและวลี
- เข้าใจแนวคิดในการสกัดคำและวลีที่เป็นตัวแทนของรูปแบบและเนื้อหาสำคัญของเอกสารโดยอัตโนมัติ
- รู้จักการใช้คำสั่งเบื้องต้นใน NLTK เพื่อประมวลผลข้อความ และหาค่าสถิติต่างๆ

ในปัจจุบัน มนุษย์ต้องสื่อสารกับคอมพิวเตอร์ด้วยชุดคำสั่งของภาษาโปรแกรมเพื่อสั่งงานให้คอมพิวเตอร์เข้าใจและทำงานได้ตามที่ต้องการ การเขียนคำสั่งต้องเขียนให้ถูกต้องตามกฎเกณฑ์และไวยากรณ์ของภาษาโปรแกรม จึงจะสามารถสั่งให้คอมพิวเตอร์ให้ทำงานได้ แต่การเรียนรู้ภาษาโปรแกรมไม่ใช่เรื่องง่ายนัก จึงเป็นที่มาของการประมวลผลภาษาธรรมชาติ (Natural Language Processing หรือ NLP) ซึ่งเป็นแนวคิดที่ทำให้คอมพิวเตอร์สามารถเข้าใจภาษาที่มนุษย์ใช้สื่อสารกันในชีวิตประจำวัน เช่น ภาษาไทย ภาษาอังกฤษ การที่นักคอมพิวเตอร์เรียกว่าภาษาธรรมชาติ เนื่องจากไม่ต้องการให้เกิดความสับสนกับคำว่าภาษาโปรแกรม การประมวลผลภาษาธรรมชาตินี้จะทำให้มนุษย์สามารถสั่งงานคอมพิวเตอร์ได้โดยใช้ภาษาธรรมชาติ ไม่ว่าจะอยู่ในรูปของเสียง (speech) หรือข้อความ (text) ซึ่งจะทำให้การใช้งานคอมพิวเตอร์ง่ายและสะดวกมากขึ้น ศาสตร์ทางด้านนี้ถือเป็นสาขาหนึ่งของปัญญาประดิษฐ์ (Artificial Intelligence) เนื่องจากเป็นศาสตร์ที่ทำให้คอมพิวเตอร์มีความสามารถคล้ายกับมนุษย์ในแง่การสื่อสารด้วยภาษาที่มนุษย์ใช้

ภาษาธรรมชาติมีความซับซ้อนมากเมื่อเทียบกับภาษาโปรแกรม เนื่องจากภาษาธรรมชาติไม่มีกฎเกณฑ์แน่นอนตายตัว สามารถปรับเปลี่ยนไปตามยุคสมัย มีคำศัพท์ที่ใช้เป็นจำนวนมาก ทำให้ยากต่อการตีความและทำความเข้าใจ ศาสตร์ทางด้านนี้จึงต้องอาศัยความรู้ทั้งทางด้านภาษาศาสตร์ และวิทยาการคอมพิวเตอร์ เพื่อช่วยให้คอมพิวเตอร์เข้าใจภาษาธรรมชาติของมนุษย์ หรือจริงๆ แล้วคอมพิวเตอร์อาจจะไม่จำเป็นต้องเข้าใจภาษามนุษย์ แต่สามารถเปลี่ยนภาษาธรรมชาติเป็นรูปแบบที่คอมพิวเตอร์สามารถนำไปใช้งานหรือสั่งการได้

1.1 ไพธอนกับการประมวลผลทางภาษา

ไพธอนเป็นภาษาที่พัฒนามาจากภาษาซี เป็นภาษาที่สามารถทำงานได้หลายแพลตฟอร์ม เช่น Unix, Linux, Mac, Window ตัวแปลภาษาทำงานแบบ interpreter คือแปลคำสั่งแล้วทำงานทีละบรรทัด โดยมี **I**nteractive **D**eve**L**opment **E**nvironment หรือ IDLE เป็น GUI ที่ใช้งานได้ง่าย ภาษาไพธอนเป็นภาษาที่เหมาะสำหรับงานประมวลผลข้อความ เพราะมีชุดคำสั่งและโครงสร้างข้อมูลที่จัดการกับข้อความได้ง่าย นอกจากนี้ยังมีไลบรารีที่สนับสนุนการประมวลผลภาษาธรรมชาติ อย่างเช่น NLTK อีกด้วย (สามารถ download ได้จาก http://nltk.org)

เมื่อติดตั้ง NLTK เรียบร้อยแล้ว ให้ทำการติดตั้งข้อมูลที่จำเป็นในการเรียนโดยใช้คำสั่ง

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

เลือก book collection จากนั้นทำการทดสอบโดยใช้คำสั่งที่ทำการโหลดหนังสือจากโมดูล book เพื่อแสดงรายชื่อหนังสือที่มีให้ ดังตัวอย่างด้านล่างนี้

```
>>> from nltk.book import *
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
```

```
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
```

ถ้าต้องการทราบชื่อหนังสือแต่ละเล่มก็ให้พิมพ์ชื่อหนังสือหลังเครื่องหมาย prompt

```
>>> text1
<Text: Moby Dick by Herman Melville 1851>
>>> text2
<Text2: Sense and Sensibility by Jane Austen 1811>
```

1.2 การประมวลผลคำและข้อความ

เรามักจะคุ้นเคยกับข้อความ เนื่องจากเราอ่านและเขียนข้อความอยู่เป็นประจำทุกวัน ในการเขียนโปรแกรมเพื่อประมวลผลทางภาษา ข้อความ คือ ข้อมูลดิบ (raw data) ที่เราสนใจนำมาจัดการและวิเคราะห์ได้หลากหลายรูปแบบ ตัวอย่างเช่น ถ้าเราต้องการค้นหาตัวอย่างของการใช้คำๆ หนึ่งที่เราสนใจ โดยอาจจะดูว่าคำนั้นมีการใช้กับเรื่องใดบ้าง ใช้ในบริบทและความหมายใดได้บ้าง เราใช้วิธีการที่เรียกว่า concordance เพื่อแสดงบริบทรอบข้างของคำๆ หนึ่งที่เราสนใจได้ ดังตัวอย่างต่อไปนี้

```
>>> text1.concordance("silently")
Displaying 10 of 10 matches:
ow came a lull in his look , as he silently turned over the leaves of the Book
draw the poles , ye harpooneers !" Silently obeying the order , the three harp
er from the beginning . For me , I silently recalled the mysterious shadows I
sings of his chip of a craft , and silently eyeing the vast blue eye of the se
ped up in him and the slow - match silently burning along towards them ; as he
 sank . Then once more arose , and silently gleamed . It seemed not a whale ;
d back to the vessel ; the rest as silently following . Whatever superstitions
raceful repose of the line , as it silently serpentines about the oarsmen befo
ales of the boats , we swiftly but silently paddled along ; the calm not admit
d at her helm , and for long hours silently guided the way of this fire - ship
cketer , and a whaleman , you will silently worship there . CHAPTER 105 Does t
straight flame , the Parsee passed silently , and bowing over his head towards
, each of the three tall masts was silently burning in that sulphurous air , l
ng from his enchantment , Gardiner silently hurried to the side ; more fell th
ming like a canopy over the fish , silently perched and rocked on this pole ,
```

จากตัวอย่างข้างต้น เป็นการค้นหา concordance ของคำว่า “silently” ในหนังสือ Moby Dick ซึ่งอยู่ใน text1 ของ nltk.book

นอกจากนี้ เราสามารถหาคำที่คล้ายคลึงกันได้ โดยหาจากคำที่มีบริบทรอบข้างเหมือนกัน ดังตัวอย่าง

```
>>> text1.similar("silently")
mildly round steadfastly stood through

>>> text9.similar("silently")
close extraordinary indisputable slight steadily without
```

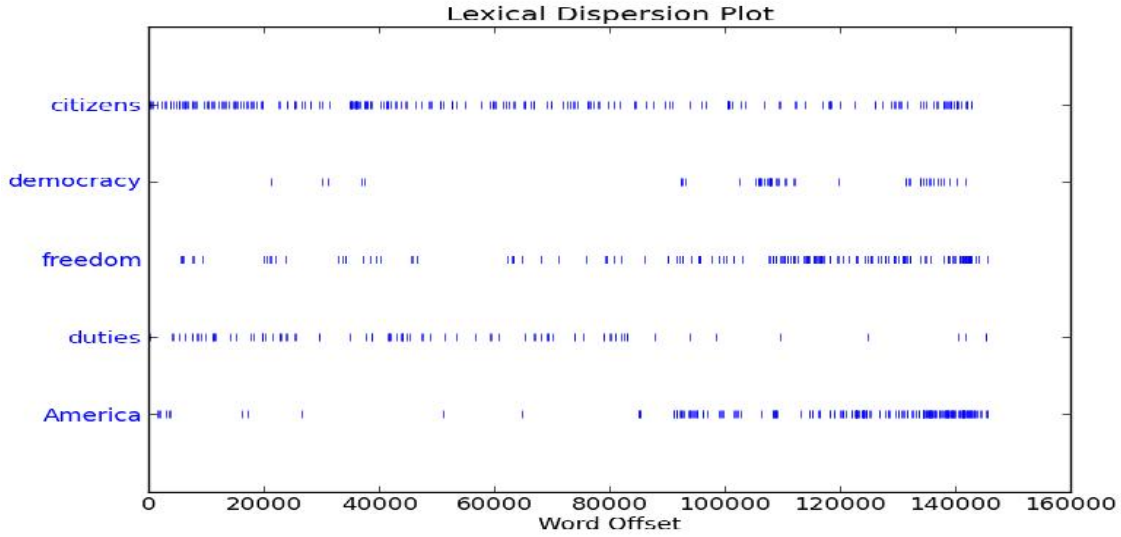
ตัวอย่างด้านล่างนี้ เป็นการค้นหาบริบทที่เหมือนกันของคำทั้งสามคำในลิสต์

```
>>> text1.common_contexts(['silently','mildly','steadfastly'])
and_eyeing
```

นอกจากนี้ เรายังสามารถหาตำแหน่งของคำในเอกสารโดยนำมาแสดงผลในรูปของกราฟ เพื่อดูลักษณะการกระจายตัวในเอกสาร ดังตัวอย่างคำสั่งต่อไปนี้

```
>>> text4.dispersion_plot(["citizens", "democracy", "freedom", "duties", "America"])
```

จากคำสั่งข้างต้น จะได้ผลลัพธ์ดังรูปที่ 1.1 ทำให้เราสามารถวิเคราะห์รูปแบบการใช้คำทั้ง 5 ในลิสต์ตามแต่ละยุคสมัยได้



รูปที่ 1.1 กราฟแสดงตำแหน่งของกลุ่มคำ ["citizens", "democracy", "freedom", "duties", "America"] ในเอกสาร text4 ซึ่งเป็นคลังที่รวบรวมสุนทรพจน์ของประธานาธิบดีของสหรัฐอเมริกาในวันเข้ารับตำแหน่ง

ในส่วนการนับคำศัพท์ เราสามารถใช้คำสั่ง len เพื่อนับจำนวนโทเค็น (token) ที่ปรากฏในลิสต์หรือเอกสารได้ โทเค็นอาจจะเป็นคำหรือเครื่องหมายวรรคตอนก็ได้ ดังตัวอย่างต่อไปนี้

```
>>> saying = ['After', 'all', 'is', 'said', 'and', 'done', 'more', 'is', 'said', 'than', 'done']
>>> len(saying) ❶
11
>>> vocabs = set(saying) ❷
>>> len(vocabs)
8
>>> vocabs
set(['and', 'all', 'said', 'is', 'After', 'done', 'than', 'more'])
>>> tokens = sorted(tokens) ❸
['After', 'all', 'and', 'done', 'is', 'more', 'said', 'than']
```

จากตัวอย่าง คำสั่งที่ 1 เป็นการใช้คำสั่ง len เพื่อนับจำนวนโทเค็นทั้งหมดในลิสต์ saying ถ้าต้องการนับว่ามีคำศัพท์ทั้งหมดเท่าไหร่ จะใช้คำสั่ง set เพื่อกำจัดโทเค็นที่ซ้ำกันออก ดังคำสั่งที่ 2 ถ้าต้องการเรียงลำดับคำศัพท์ตามตัวอักษรและเปลี่ยนจากตัวแปรประเภท set เป็นตัวแปรประเภท list จะใช้คำสั่ง sorted ดังตัวอย่างในคำสั่งที่ 3

เราสามารถคำสั่งข้างต้นมาใช้กับเอกสารได้เช่นกัน ดังตัวอย่าง

```
>>> len(text1)
260819
>>> len(set(text1))
19317
>>> sorted(set(text3))
['!', '"', '(', ')', ',', ',)', '.', '.)', ':', ';', ';)', '?', '?)',
'A', 'Abel', 'Abelmizraim', 'Abidah', 'Abide', 'Abimael', 'Abimelech',
'Abr', 'Abrah', 'Abraham', 'Abram', 'Accad', 'Achbor', 'Adah', ...]
```

จากตัวอย่างข้างต้น จะเห็นว่าจำนวนโทเค็นของ text1 มีทั้งหมด 260819 โทเค็น แต่ถ้าเรานับเฉพาะคำศัพท์ หรือโทเค็นที่ไม่ซ้ำกันจะเหลือเพียง 19317 โทเค็นเท่านั้น ถ้าต้องการทราบสัดส่วนของคำศัพท์ที่ไม่ซ้ำกันต่อโทเค็นทั้งหมด หรือเรียกว่า ค่าความหลากหลายของคำศัพท์ (lexical diversity) จะทำได้โดยใช้คำสั่ง

```
>>> len(set(text1)) / len(text1)
0.07406285585022564
```

ตารางที่ 1.1 แสดงค่าความหลากหลายของคำศัพท์ในคลังข้อมูล Brown แยกตามประเภทของเอกสาร

ประเภทเอกสาร	โทเค็น	คำศัพท์	ความหลากหลายของคำศัพท์
skill and hobbies	82345	11935	0.145
humor	21695	5017	0.231
fiction: science	14470	3233	0.223
press: reportage	100554	14394	0.143
fiction: romance	70022	8452	0.121
religion	39399	6373	0.162

จากตารางที่ 1.1 จะเห็นว่าเอกสารประเภท humor มีค่าความหลากหลายของคำศัพท์สูงที่สุด หมายความว่า เอกสารประเภทนี้มีการใช้คำศัพท์ที่หลากหลายมากกว่าเอกสารประเภทอื่น จะเห็นได้จากจำนวนคำศัพท์ทั้งหมดในเอกสารมีสัดส่วนใกล้เคียงกับจำนวนโทเค็นทั้งหมด ประมาณ 1/4 หรือกล่าวได้ว่า คำศัพท์หนึ่งคำมีการใช้ซ้ำๆ เฉลี่ยแล้ว 4 ครั้งในเอกสาร ส่วนเอกสารประเภท fiction: romance มีค่าความหลากหลายของคำศัพท์ต่ำที่สุด เป็นสัดส่วนประมาณ 1/8 เราจึงทราบว่าเอกสารประเภทนี้มักจะมีการใช้คำศัพท์เดิมซ้ำๆ กันอยู่ คือใช้คำเดิมซ้ำกันเฉลี่ยถึง 8 ครั้ง

เมื่อเราต้องการนับจำนวนครั้งที่โทเค็นใดๆ ปรากฏในเอกสาร จะใช้คำสั่ง count นอกจากนี้ เรายังสามารถหาเปอร์เซ็นของการใช้โทเค็นใดๆ เมื่อเทียบกับจำนวนโทเค็นทั้งหมดในเอกสารได้ ดังตัวอย่าง

```
>>> text3.count("smote")
5
>>> 100 * text4.count('a') / len(text4)
1.4643016433938312
```

เราสามารถเขียนชุดคำสั่งที่ยกตัวอย่างมาแล้วข้างต้นเป็นฟังก์ชันเพื่อสะดวกในการเรียกใช้งานในครั้งถัดไปได้ ดังตัวอย่างต่อไปนี้

```
>>> def lexical_diversity(text):
...     return len(set(text)) / len(text)
...
>>> def percentage(count, total):
...     return 100 * count / total
...
>>> lexical_diversity(text3)
0.06230453042623537
>>> lexical_diversity(text5)
0.13477005109975562
>>> percentage(4, 5)
80.0
>>> percentage(text4.count('a'), len(text4))
1.4643016433938312
>>>
```

1.3 สถิติกับการประมวลผลทางภาษา

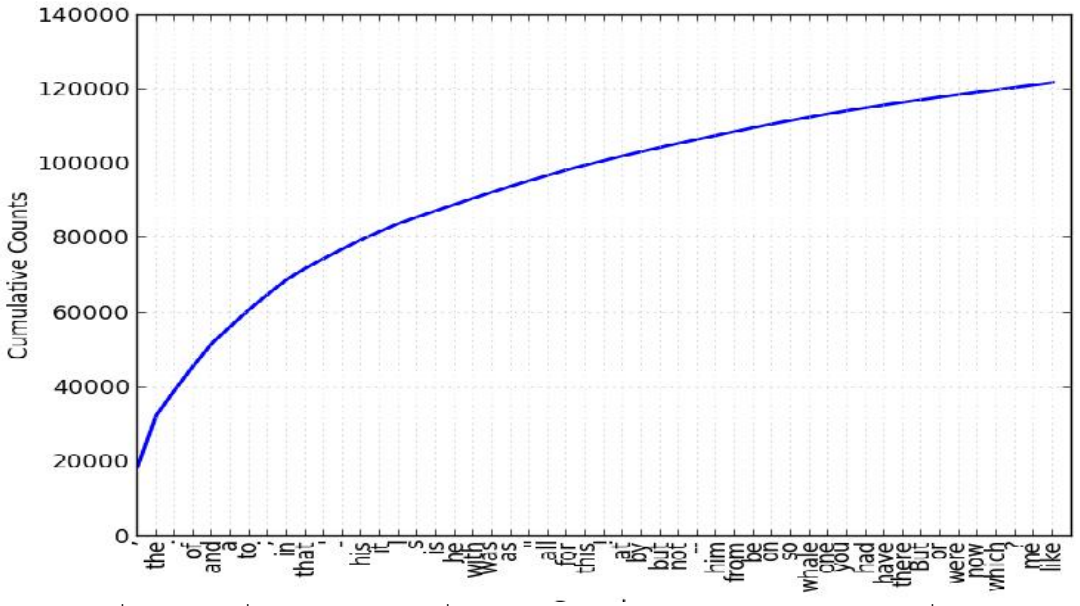
การประมวลผลข้อความมักจะเกี่ยวข้องกับการคำนวณหาค่าทางสถิติต่างๆ สถิติที่มักจะใช้บ่อย ได้แก่ การแจกแจงความถี่ (frequency distribution) เพื่อดูลักษณะการกระจายตัวของโทเค็นบนคำศัพท์ทั้งหมดในเอกสาร ใน NLTK เราสามารถใช้ชุดคำสั่ง FreqDist เพื่อหาการแจกแจงความถี่ของคำศัพท์ในเอกสารได้ ดังตัวอย่าง

```
>>> fdist = FreqDist(text1)
>>> print(fdist)
<FreqDist with 19317 samples and 260819 outcomes>
>>> fdist.most_common(50)
[(',', 18713), ('the', 13721), ('.', 6862), ('of', 6536), ('and', 6024),('a', 4569), ('to', 4542), (';', 4072),
('in', 3916), ('that', 2982),("'", 2684), ('-', 2552), ('his', 2459), ('it', 2209), ('I', 2124), ('s', 1739),
('is', 1695), ('he', 1661), ('with', 1659), ('was', 1632), ('as', 1620), ('"', 1478), ('all', 1462),
('for', 1414), ('this', 1280), ('!', 1269), ('at', 1231), ('by', 1137), ('but', 1113), ('not', 1103),
('--', 1070), ('him', 1058), ('from', 1052), ('be', 1030), ('on', 1005), ('so', 918), ('whale', 906),
('one', 889), ('you', 841), ('had', 767), ('have', 760), ('there', 715), ('But', 705), ('or', 697),
('were', 680), ('now', 646), ('which', 640), ('?', 637), ('me', 627), ('like', 624)]
>>> fdist['whale']
906
>>>fdist.plot(50, cumulative=True)
```

จากตัวอย่าง เป็นการใช้ชุดคำสั่ง FreqDist เพื่อแจกแจงความถี่ของเอกสาร text1 แล้วเก็บค่าไว้ในตัวแปร fdist เมื่อต้องการหาความถี่ของคำศัพท์ใดๆ สามารถอ้างถึงได้โดยใช้ตัวแปร fdist1 แล้วระบุคำที่ต้องการทราบความถี่ในรูปดัชนี ส่วนการหาความถี่ของคำศัพท์ที่พบบ่อยๆ จำนวน n ตัวแรก จะใช้คำสั่ง most_common(n) คำสั่งสุดท้ายในตัวอย่างจะแสดงกราฟความถี่สะสมของคำศัพท์ที่พบบ่อย 50 คำแรก ดังรูปที่ 1.2 ถ้าเราสนใจคำศัพท์ที่ปรากฏเพียงครั้งเดียวในเอกสาร จะใช้คำสั่ง hapaxes() ส่วนตัวอย่างคำสั่งอื่นๆ ใน FreqDist แสดงในตารางที่ 1.2

การสกัดคำหรือวลีสำคัญ (keyword extraction) เป็นงานประมวลผลข้อความที่สามารถใช้การแจกแจงความถี่ เพื่อหาลักษณะของคำสำคัญที่สามารถใช้เป็นตัวแทนของเอกสารได้ จากตัวอย่างจะเห็นได้ว่า คำศัพท์ที่พบได้บ่อยที่สุดในเอกสารไม่ใช่คำที่เหมาะสมสำหรับเป็นตัวแทนของเอกสารได้ และจากกราฟในรูป 1.2 จะพบว่า ความถี่

รวมของคำศัพท์ที่พบบ่อย 50 แรก มีจำนวนเกือบครึ่งหนึ่งของจำนวนโทเค็นทั้งหมดในเอกสาร ถ้าทดลองใช้คำสั่ง hapaxes จะพบว่ามีคำจำนวนมาก (ประมาณ 9000 คำ) ที่ปรากฏเพียงแค่ครั้งเดียวในเอกสาร ซึ่งก็ไม่เหมาะที่จะนำมาเป็นคำสำคัญเช่นกัน



รูปที่ 1.2 ความถี่สะสมของคำศัพท์ที่พบบ่อย 50 คำแรกในเอกสาร text1 (หนังสือชื่อ Moby Dick)

ตารางที่ 1.2 ตัวอย่างฟังก์ชันใน FreqDist

ตัวอย่าง	อธิบายการใช้
fdist = FreqDist(samples)	สร้างตารางแจกแจงความถี่ของเอกสาร samples
fdist['monstrous'] += 1	เพิ่มความถี่ให้กับคำศัพท์
fdist['monstrous']	ความถี่ของคำศัพท์
fdist.freq('monstrous')	ค่าความหลากหลายของคำศัพท์
fdist.N()	จำนวนโทเค็นทั้งหมดใน samples
fdist.most_common(n)	แสดงคำและความถี่ ที่มีความถี่สูงสุด n อันดับแรก
for sample in fdist:	วนลูปเท่ากับจำนวนคำศัพท์ใน samples
fdist.max()	คำศัพท์ที่มีความถี่สูงสุด
fdist.tabulate()	แสดงคำศัพท์เรียงตามความถี่
fdist.plot()	แสดงการแจกแจงความถี่ในรูปกราฟ
fdist.plot(cumulative=True)	แสดงการแจกแจงความถี่สะสมในรูปกราฟ
fdist1 |= fdist2	ปรับปรุงค่าความถี่ในตัวแปร fdist1 โดยรวมกับความถี่ในตัวแปร fdist2

หากเราตั้งสมมติฐานว่า คำที่มีความยาวมากน่าจะเป็นคำสำคัญ เช่น ความยาวของคำมากกว่า 15 ตัวอักษร สามารถเขียนอยู่ในรูปของเซตได้ว่า $\{w \mid w \in V \& P(w)\}$ เมื่อ $P(w)$ คือคุณสมบัติของคำ w ที่จะเป็นจริง ถ้าคำมีความยาวมากกว่า 15 ตัวอักษร ส่วน V คือเซตของคำศัพท์ทั้งหมด เมื่อแปลงเป็นนิพจน์ในภาษาไพธอน จะเขียนอยู่ในรูปของการสร้างลิสต์จากข้อมูลในลิสต์อื่น (list comprehension) ได้ดังนี้ [w for w in V if p(w)] และสามารถแปลงเป็นคำสั่งได้ตัวอย่างต่อไปนี้

```
>> V = set(text1)
>>> long_words = [w for w in V if len(w) > 15]
>>> sorted(long_words)
['CIRCUMNAVIGATION', 'Physiognomically', 'apprehensiveness', 'cannibalistically',
'characteristically', 'circumnavigating', 'circumnavigation', 'circumnavigations',
'comprehensiveness', 'hermaphroditical', 'indiscriminately', 'indispensableness',
'irresistibleness', 'physiognomically', 'preternaturalness', 'responsibilities',
'simultaneousness', 'subterraneousness', 'supernaturalness', 'superstitiousness',
'uncomfortableness', 'uncompromisedness', 'undiscriminating', 'uninterpenetratingly']
```

จะเห็นว่า คำที่มีความยาวมากส่วนใหญ่จะเป็นคำวิเศษณ์ และคำนามที่มีรากศัพท์มาจากคำคุณศัพท์ ซึ่งคำทั้งสองประเภทนี้ ไม่ใช่ลักษณะของคำสำคัญอีกเช่นกัน แต่ด้วยวิธีการนี้ทำให้เราสามารถตัดคำสั้นที่มีความถี่มาก รวมถึงตัดคำยาวที่มีความถี่น้อยออกไปได้ ดังนั้น หากพิจารณาคุณสมบัติทั้งในแง่ความยาวของคำศัพท์ร่วมกับความถี่ ก็อาจจะให้ผลที่ดีขึ้นได้ ดังตัวอย่างคำสั่งต่อไปนี้ที่กำหนดเงื่อนไขว่าเลือกคำที่มีความยาวมากกว่า 7 ตัวอักษร และมีความถี่มากกว่า 7 ครั้ง

```
>>> fdist5 = FreqDist(text5)
>>> sorted(w for w in set(text5) if len(w) > 7 and fdist5[w] > 7)
['#14-19teens', '#talkcity_adults', '((((((((((', '.....', 'Question',
'actually', 'anything', 'computer', 'cute.-ass', 'everyone', 'football',
'innocent', 'listening', 'remember', 'seriously', 'something', 'together',
'tomorrow', 'watching']
```

จากวิธีการข้างต้น จะเห็นว่าเราสามารถเปลี่ยนข้อมูลดิบที่ประกอบด้วยคำจำนวนมาก เป็นสารสนเทศที่มีโอกาสเป็นคำสำคัญที่ใช้เป็นตัวแทนของเอกสารได้ โดยใช้คำสั่งในการประมวลผลไม่กี่คำสั่ง แม้ว่าขั้นตอนนี้จะยังไม่สามารถสกัดคำสำคัญออกมาได้ แต่ก็เพียงพอที่จะนำไปใช้ในขั้นตอนต่อไปได้ไม่ยาก

นอกจากนี้ เราสามารถประยุกต์ชุดคำสั่งการแจกแจงความถี่ ในการวิเคราะห์ค่าทางสถิติต่างๆ ได้ ตัวอย่างเช่น การนับความถี่ของคำที่มีความยาวต่างกัน เพื่อพิจารณาการกระจายตัวของความยาวของคำได้ โดยใช้ FreqDist กับลิสต์ของความยาวของคำทั้งหมดในเอกสาร ดังตัวอย่างต่อไปนี้

```
>>> len_list = [len(w) for w in text1] ❶
[1, 4, 4, 2, 6, 8, 4, 1, 9, 1, 1, 8, 2, 1, 4, 11, 5, 2, 1, 7, 6, 1, 3, 4, 5, 2, ...]
>>> fdist = FreqDist(len_list) ❷
>>> print(fdist) ❸
<FreqDist with 19 samples and 260819 outcomes>
>>> fdist
FreqDist({3: 50223, 1: 47933, 4: 42345, 2: 38513, 5: 26597, 6: 17111, 7: 14399,  8: 9966, 9: 6428,
10: 3528, ...})
>>> fdist.most_common()
[(3, 50223), (1, 47933), (4, 42345), (2, 38513), (5, 26597), (6, 17111), (7, 14399),
(8, 9966), (9, 6428), (10, 3528), (11, 1873), (12, 1053), (13, 567), (14, 177),
(15, 70), (16, 22), (17, 12), (18, 1), (20, 1)]
```

```
>>> fdist.max()
3
>>> fdist[3]
50223
>>> fdist.freq(3)
0.19255882431878046
```

จากตัวอย่าง คำสั่งที่ 1 เป็นการสร้างลิสต์ของความยาวของคำในเอกสาร จากนั้นจึงนำมาแจกแจงความถี่โดยใช้ FreqDist ในคำสั่งที่ 2 เมื่อสั่งพิมพ์ค่า fdist จะพบว่าเอกสาร text1 มีคำที่มี 19 ความยาวที่แตกต่างกัน โดยความยาวที่พบมากสุด คือ ขนาด 3 ตัวอักษร ซึ่งมีจำนวนทั้งหมด 50223 คำ และความหลากหลายของคำศัพท์ที่มีความยาว 3 ตัวอักษรมีค่า 0.19 (50223/260819)

หลักการทางสถิติอีกวิธีการหนึ่งที่เกี่ยวข้องกับงานการประมวลผลข้อความ คือ การหาความน่าจะเป็นของคำที่ปรากฏร่วมกัน n คำ เรียกว่า n-gram เช่น ถ้า n=2 จะเรียกว่า 2-gram หรือ bigram จะเป็นการหาค่าความน่าจะเป็นของคำใดๆ โดยดูจากคำก่อนหน้าเพียงหนึ่งคำ เรานิยมเขียนค่าความน่าจะเป็นนี้ในรูปของ P(w2|w1) ซึ่งหมายถึงความน่าจะเป็นที่คำ w2 จะเกิดตามหลังคำ w1

ในภาษาไพธอนเราสามารถสร้าง bigram จากลิสต์ได้โดยใช้คำสั่ง bigrams() ดังตัวอย่าง

```
>>> list(bigrams(['more', 'is', 'said', 'than', 'done']))
[('more', 'is'), ('is', 'said'), ('said', 'than'), ('than', 'done')]
```

เมื่อเราได้ลิสต์ของ bigram แล้ว เราสามารถนำมาหาคำที่เกิดขึ้นพร้อมกันบ่อยๆ ที่เรียกว่า collocation ได้โดยดูจากค่าความน่าจะเป็นของ bigram ที่มีความถี่สูง ซึ่งขั้นตอนทั้งหมดเราสามารถคำสั่ง collocations() ใน NLTK ที่ใช้ในการหาคำประเภทนี้ได้เลย ดังตัวอย่าง

```
>>> text4.collocations()
United States; fellow citizens; four years; years ago; Federal Government; General Government;
American people; Vice President; Old World; Almighty God; Fellow citizens; Chief Magistrate;
Chief Justice; God bless; every citizen; Indian tribes; public debt; one another; foreign nations;
political parties
>>> text8.collocations()
would like; medium build; social drinker; quiet nights; non smoker; long term; age open; Would like;
easy going; financially secure; fun times; similar interests; Age open; weekends away; poss rship;
well presented; never married; single mum; permanent relationship; slim build
```

collocation มักจะเป็นคำที่จำเป็นต้องใช้ร่วมกันในประโยค และให้ความหมายพิเศษต่างไปจากเดิม คำที่ใช้ร่วมกันนี้ จะใช้คำอื่นที่มีความหมายทำนองเดียวกันแทนที่ไม่ได้ เช่น fast color หมายถึงสีไม่ตก ถ้าเรานำคำที่มีความหมายทำนองเดียวกันมาแทนที่ เช่น stable color หรือ speed color จะกลายเป็นคำที่มีความหมายผิดไปหรือเป็นคำคู่ที่ไม่ใช้ในภาษานั้นเพื่อให้ความหมายแบบเดียวกัน ดังนั้น fast color จึงถือเป็น collocation ถ้าเป็นคำที่ปรากฏร่วมกันแต่ไม่มีความหมายพิเศษก็จะไม่ถือว่าเป็น collocation เช่น the color หรือ red color