บทที่ 7
การประมวลผลภาษาธรรมชาติขั้นพื้นฐาน

Contents

ข้อมูลจัดเป็นทรัพยากรที่มีค่ามหาศาล ในปัจจุบันเกือบทุกบริษัท ทุกองค์กรทั้งภาครัฐและเอกชน ต่างเก็บข้อมูลต่าง ๆ ที่เกี่ยวกับการดำเนินธุรกิจ หรือบริหารงานทุกประเภท เช่น เมื่อเราเข้าร้านสะดวกซื้อ หรือห้างสรรพสินค้า พนักงานมักจะถามหาหมายเลขสมาชิก หรือเบอร์โทรศัพท์ เพื่อเก็บข้อมูลการซื้อของลูกค้าตลอดระยะเวลาที่ยังเป็นลูกค้าอยู่ สิ่งที่บริษัทต้องการได้คือข้อมูลของลูกค้า ถึงแม้ว่าจะต้องแลกมากับการให้ส่วนลด หรือให้ลูกค้าแลกแต้มเพื่อได้ของสมมนาคุณต่าง ๆ ข้อมูลเหล่านี้ทำให้บริษัทได้ศึกษาพฤติกรรมของลูกค้า ทำให้เลือกสินค้ามาขายได้ถูกใจลูกค้ามากขึ้น ได้ออกกิจกรรมส่งเสริมการขายได้ตรงใจลูกค้ามากขึ้น และทำให้สามารถแบ่งลูกค้าออกมาได้เป็นกลุ่ม (เช่น กลุ่มที่เป็นลูกค้าใหม่ กลุ่มที่ซื้อไม่บ่อยแต่ซื้อเยอะ กลุ่มที่ซื้อสม่ำเสมอ เป็นต้น) เพื่อสามารถให้บริการลูกค้าได้ดีขึ้น ให้ลูกค้ามีความพึงพอใจมากขึ้น และเพิ่มยอดขายได้มากขึ้น

ข้อมูลอีกประเภทที่กำลังเป็นที่นิยมมากขึ้น คือข้อมูลตัวอักษร (text data) ซึ่งได้มาจากแพลตฟอร์มสื่อสังคมออนไลน์ แพลตฟอร์มการซื้อขายออนไลน์ หรือการทำสำรวจความคิดเห็นที่มีคำถามปลายเปิด ข้อมูลเหล่านี้มีหลากมิติกว่าข้อมูลที่เป็นเชิงปริมาณ หรือข้อมูลที่เป็นตัวเลข เนื่องจากผู้ที่ให้ข้อมูลสามารถแสดงความเห็นได้อย่างอิสระ ทำให้ได้คำตอบที่หลากหลาย เมื่อนำมาวิเคราะห์ทำให้เกิดความรู้เชิงประจักษ์ที่เอื้อต่อการดำเนินการต่อ (actionable insight) ที่สามารถนำไปปรับใช้กับองค์กรหรือธุรกิจได้ ตัวอย่างเช่น บริษัทสามารถดึงข้อมูลรีวิวความเห็นของลูกค้าที่ได้ซื้อสินค้าจากแพลตฟอร์มการซื้อขายออนไลน์ที่บริษัทไปเปิดร้านไว้ แล้วนำข้อมูลนี้ไปวิเคราะห์ว่าลูกค้าชอบอะไรหรือไม่ชอบอะไรเกี่ยวกับผลิตภัณฑ์ของเรา หรือลูกค้าชอบอะไรเกี่ยวกับผลิตภัณฑ์ที่ทำโดยบริษัทคู่แข่ง ทำให้เกิดความรู้เชิงประจักษ์ที่สามารถนำไปพัฒนาผลิตภัณฑ์ให้ตอบโจทย์ของผู้บริโภคได้ดีขึ้น และพัฒนากระบวนการการสั่งซื้อของและส่งของให้ลูกค้าให้มีประสิทธิภาพมากขึ้น

การใช้เทคโนโลยีในการวิเคราะห์ข้อมูล เพื่อสกัดความรู้เชิงประจักษ์ที่ก่อให้เกิดประโยชน์และมูลค่าทางธุรกิจ เรียกว่า วิทยาการข้อมูล (data science) เป็นการผสมผสานระหว่างศาสตร์และความรู้การบริหารธุรกิจ ซึ่งทำให้เราเข้าใจกลไกในการประกอบธุรกิจและเข้าใจลูกค้าช่วยให้มีผลประกอบการที่ดี สถิติ ซึ่งทำให้เราสามารถวิเคราะห์และสรุปหาแพตเทิร์นในข้อมูลที่มีขนาดใหญ่ และวิทยาการคอมพิวเตอร์ (computer science) ซึ่งทำให้เราสามารถเขียนโปรแกรมที่สามารถใช้แบบจำลองที่ซับซ้อนหรือจัดการกับข้อมูลที่มีขนาดใหญ่และโครงสร้างซับซ้อนได้ แต่เมื่อเราต้องวิเคราะห์ข้อมูลตัวอักษร หรือข้อมูลที่เป็นภาษาธรรมชาติ (natural language data) ซึ่งไม่สามารถนำมาหาค่าเฉลี่ย หรือบวกลบคูณหารอย่างที่ทำกับข้อมูลเป็นเชิงปริมาณ เราจึงต้องใช้เทคนิควิธีการประมวลผลภาษาธรรมชาติ (natural language processing) ซึ่งมีแบบจำลองในการทำความเข้าใจภาษาเพื่อการวิเคราะห์ข้อมูลเหล่านี้ เพราะฉะนั้นการประมวลผลภาษาธรรมชาติ คือ เทคนิควิธีที่ใช้ในการประมวลผลและทำความเข้าใจข้อมูลตัวอักษร แบบอิงแบบจำลองทางภาษา เมื่อนำมาประกอบกับการวิเคราะห์ข้อมูลเพื่อพัฒนาธุรกิจ เราจะเรียกว่าการวิเคราะห์ข้อความ (text analytics)

คำว่า “ภาษาธรรมชาติ” ใช้เพื่ออ้างอิงถึงภาษาที่มนุษย์ใช้ในการสื่อสารกัน ซึ่งรวมไปถึงทั้งภาษาพูดและภาษาเขียนที่พัฒนาขึ้นอย่างธรรมชาติในสังคมมนุษย์ ไม่ว่าจะเป็นภาษาไทย อังกฤษ จีน หรือภาษาอื่น ๆ การใช้คำว่า “ภาษาธรรมชาติ” เพื่อแยกแยะจาก “ภาษาคอมพิวเตอร์” ซึ่งเป็นภาษาที่ถูกสร้างขึ้นมาเพื่อการเขียนโปรแกรมและการสื่อสารกับเครื่องคอมพิวเตอร์ เช่น ภาษาไพทอน หรือ ภาษา

จาวา (Java) ซึ่งมีโครงสร้างและกฎเกณฑ์ที่เข้มงวดกว่ามาก ด้วยเหตุนี้ เทคนิคที่ถูกพัฒนามาเพื่อการประมวลผลและทำความเข้าใจภาษาที่มนุษย์ใช้จึงเรียกว่า “ประมวลผลภาษาธรรมชาติ” หรือ NLP เทคนิคเหล่านี้ออกแบบมาเพื่อให้เครื่องคอมพิวเตอร์สามารถเข้าใจและประมวลผลข้อมูลที่เป็นภาษาธรรมชาติได้อย่างมีประสิทธิภาพ เป้าหมายคือการให้คอมพิวเตอร์สามารถ “เข้าใจ” ภาษามนุษย์ได้ใกล้เคียงกับวิธีที่มนุษย์เข้าใจภาษาของกันและกัน เพื่อใช้ประโยชน์ในงานต่าง ๆ ที่เกี่ยวกับภาษา

นอกจากนั้นการประมวลผลภาษาธรรมชาติ เป็นเทคโนโลยีที่เป็นกระดูกสันหลังของแอปพลิเคชันที่ทำหน้าที่ทางภาษาโดยอัตโนมัติได้ ตัวอย่างเช่น Google Translate เป็นแอปพลิเคชันทำหน้าที่แปลภาษาโดยอาศัยแบบจำลองทางภาษาที่เข้าใจทั้งภาษาต้นทางที่ต้องการแปลและภาษาปลายทาง หรือแอปพลิเคชัน ChatGPT ที่สามารถทำหน้าที่ทางภาษาได้อย่างหลากหลาย ไม่ว่าจะเป็นการสรุปข่าว การแต่งนิยาย การปรับแก้ภาษาให้สละสลวยไร้ข้อผิดพลาด การตอบคำถามที่เป็นปลายเปิด การให้คำปรึกษาเรื่องต่าง ๆ หรือแอปพลิเคชัน Google Search เองที่สามารถทำความเข้าใจสิ่งที่ผู้ใช้ต้องการค้นหา โดยพิจารณาจากคำค้นที่ผู้ใช้พิมพ์เข้ามาในกล่อง และทำความเข้าใจเว็บไซต์ทุกเว็บไซต์ และเลือกมาเฉพาะเว็บไซต์ที่ตอบสนองโจทย์ความต้องการทางข้อมูลของผู้ใช้ตามที่ได้ระบุมาในคำค้น

สรุปคือการประยุกต์ใช้ NLP สามารถนำไปใช้ประโยชน์ได้อย่างน้อย 2 ทาง คือ

1. เครื่องมือวิเคราะห์ข้อความที่สกัดความรู้เชิงประจักษ์
2. เทคโนโลยีเบื้องหลังของแอปพลิเคชันที่ทำหน้าที่ทางภาษาโดยอัตโนมัติ

หลักการของการประมวลผลภาษาธรรมชาติ

ข้อมูลตัวอักษรมักจะเก็บอยู่ในรูปของสตริง หรือเก็บอยู่ในโครงสร้างข้อมูลที่เก็บสตริงอยู่ เช่น ลิสต์ของสตริง ข้อมูลเมื่อรวบรวมมาอยู่ในชุดเดียวกัน เราเรียกว่าชุดข้อมูล (dataset) เช่น ชุดข้อมูลทวิตเตอร์ที่เก็บมาจากแฮชแท็กหนึ่งจากช่วงเวลาหนึ่ง ชุดข้อมูลข่าวต่างประเทศจากหนังสือพิมพ์ไทยออนไลน์จากช่วงเวลาหนึ่ง เป็นต้น ชุดข้อมูลชุดหนึ่งประกอบด้วย ข้อมูลหลาย ๆ แถว (row) หรือเรียกอีกอย่างหนึ่งได้ว่า ระเบียน หรือเรคคอร์ด (record) เช่น ชุดข้อมูลทวิตเตอร์มีข้อมูลอยู่ 50,000 แถว ซึ่งก็คือ 50,000 ทวีต หรือชุดข้อมูลข่าวมีข้อมูลอยู่ 10,000 แถว ซึ่งก็คือ 10,000 บทความ

ข้อมูลแต่ละแถวที่อยู่ในชุดข้อมูลเป็นเพียงสตริง ซึ่งตัวสตริงเองนั้นไม่ได้มีความหมายอะไรในตัวมันเอง เป็นเพียงรูปแบบการเก็บข้อมูลในรูปแบบดิจิทัลที่นำตัวอักษรมาร้อยเรียงกัน เราจึงเรียกข้อมูลตัวอักษรว่า ข้อมูลแบบไม่มีโครงสร้าง (unstructured data) การที่จะทำให้เครื่องคอมพิวเตอร์สามารถเข้าใจความหมายได้จำเป็นต้องใช้ทฤษฎีทางด้านภาษาศาสตร์เข้ามาช่วยทำให้สตริงมีโครงสร้างมากขึ้น

ภาษาศาสตร์ เป็นศาสตร์ที่วิเคราะห์ภาษาออกเป็นโครงสร้างย่อย ๆ เช่น ประโยค กลุ่มคำ คำ พยางค์ เสียงพยัญชนะ เสียงสระ หน่วยคำ เพื่อโยงโครงสร้างต่าง ๆ เข้ากับลักษณะทางภาษาทุกด้าน การประยุกต์ใช้ NLP อาศัยการวิเคราะห์ส่วนย่อย ๆ ของภาษา และโครงสร้างของภาษากับความหมาย เช่น ถ้าหากเราต้องการทราบว่าข้อมูลทวิตเตอร์ที่ติดแฮชแท็กชื่อสินค้าของบริษัทเรา พูดถึงสินค้าเราในแง่บวกหรือลบ โปรแกรมอาจจะต้องตรวจหาว่า

- คำใดบ้างที่ใช้ในการพูดถึงสินค้าของเรา หรือสินค้าของคู่แข่ง
- คำใดบ้าง และกลุ่มคำใดบ้างที่ใช้ในการสื่อความหมายในแง่บวก แง่ลบ
- คำใดบ้างที่ใช้เพื่อบ่งบอกว่าข้อความนั้นไม่ได้มีความคิดเห็นแฝงอยู่ แต่อาจจะเป็นการให้ข้อมูลอย่างเป็นกลางเท่านั้น
- ลักษณะประโยคแบบใดที่แสดงให้เห็นถึงน้ำเสียงแบบประชดประชัน หรือล้อเล่น
- การรีทวีตตอบโต้กันระหว่างผู้ใช้บนแพลตฟอร์ม แสดงถึงความคิดเห็นของลูกค้าต่อสินค้าของเราอย่างไร
- ลักษณะทางภาษาใดบ้าง ทำให้เราทราบถึงอายุ เพศ ถิ่นที่อยู่ของลูกค้าได้

การใช้ไลบรารีในภาษาไพทอน

NLP ในปัจจุบันมักจะอาศัยไลบรารีของภาษาไพทอน เนื่องจากภาษาไพทอนมีไลบรารีที่หลากหลายและมีประสิทธิภาพสูงสำหรับการทำงานด้าน NLP ไลบรารีในที่นี้หมายถึงชุดของโมดูลหรือฟังก์ชันที่ถูกจัดเก็บและจัดระเบียบไว้เพื่อใช้งานร่วมกัน ซึ่งช่วยให้นักพัฒนาสามารถเรียกใช้ฟังก์ชันที่ต้องการได้อย่างง่ายดายโดยไม่ต้องเขียนโค้ดเหล่านั้นตั้งแต่ต้น ทำให้ประหยัดเวลาและลดความซับซ้อนในการพัฒนาโปรแกรม การใช้ไลบรารีให้ความสะดวกในหลายด้าน เช่น การเข้าถึงฟังก์ชันที่ซับซ้อนได้ง่าย การลดระยะเวลาในการพัฒนาโปรแกรม และการใช้งานโค้ดที่ได้รับการทดสอบแล้วจากชุมชนผู้พัฒนา ซึ่งช่วยลดความเสี่ยงในการเกิดจุดบกพร่อง (bug) และปัญหาความปลอดภัย

ในบทนี้ จะกล่าวถึงวิธีการติดตั้งไลบรารีลงในเครื่องผ่านเครื่องมือการจัดการแพ็กเกจ เช่น pip ซึ่งเป็นเครื่องมือมาตรฐานของภาษาไพทอนในการติดตั้งและจัดการไลบรารี จากนั้นจะอธิบายวิธีการใช้งานไลบรารีเหล่านั้นในโปรแกรมไพทอน เพื่อให้ผู้อ่านสามารถนำไปประยุกต์ใช้ในโปรเจกต์ NLP ของตนเองได้ นอกจากนี้ยังจะแนะนำไลบรารีที่ใช้บ่อยในการทำงานด้าน NLP ขั้นพื้นฐาน เช่น NLTK, และ pythainlp ซึ่งแต่ละไลบรารีมีคุณสมบัติเฉพาะตัวที่เหมาะสมกับงาน NLP ต่าง ๆ ตั้งแต่การแยกคำ การแท็กชนิดของคำ ไปจนถึงการสร้างแบบจำลองภาษาที่ซับซ้อน

การติดตั้งไลบรารี

ไลบรารีของภาษาไพทอนถูกจัดเก็บในที่เก็บรวมชื่อว่า PyPI (Python Package Index) ซึ่งเป็นระบบที่เก็บไลบรารีและโมดูลของภาษาไพทอนที่พัฒนาและแชร์โดยชุมชนผู้พัฒนาทั่วโลก การติดตั้งไลบรารีผ่านเครื่องมือจัดการแพ็กเกจเช่น pip หรือ conda จะมีการสื่อสารกับที่เก็บรวมนี้เพื่อดาวน์โหลดและติดตั้งแพ็กเกจที่ต้องการลงในระบบของผู้ใช้ ในกระบวนการนี้ อาจมีขั้นตอนหลังบ้านที่รวมถึงการตรวจสอบการขึ้นต่อกันของไลบรารี (dependencies)

ไลบรารีที่พบใน PyPI มักจะเป็นแบบโอเพนซอร์ส (open-source) หมายความว่าโค้ดของไลบรารีเหล่านี้เปิดเผยให้สาธารณะสามารถเข้าถึง ใช้งาน แก้ไข และแชร์ต่อได้ เป็นส่วนหนึ่งของวัฒนธรรมการพัฒนาซอฟต์แวร์แบบร่วมมือ ซึ่งส่งเสริมการเรียนรู้ร่วมกันและการนำไปใช้ประโยชน์อย่างกว้างขวาง การเป็นโอเพนซอร์สทำให้ไลบรารีเหล่านี้ได้รับการตรวจสอบ ทดสอบ และพัฒนาอย่างต่อเนื่องจากชุมชนผู้ใช้และผู้พัฒนาทั่วโลก นอกจากนี้ยังช่วยเพิ่มความโปร่งใสและความน่าเชื่อถือของไลบรารีเนื่องจากผู้ใช้สามารถตรวจสอบและทำความเข้าใจการทำงานภายในของไลบรารีเหล่านั้นได้ โอเพนซอร์สยังเป็นแรงบันดาลใจและเป็นฐานสำหรับนวัตกรรมใหม่ ๆ เนื่องจากผู้พัฒนาสามารถนำโค้ดที่มีอยู่มาปรับปรุงหรือรวมเข้ากับโปรเจกต์ของตนเองได้โดยไม่ต้องสร้างขึ้นจากศูนย์ เพิ่มความเร็วในการพัฒนาและลดต้นทุนในการวิเคราะห์ข้อมูล หรือสร้างซอฟต์แวร์ใหม่ ๆ

สภาพแวดล้อม (environment) ในบริบทของการพัฒนาซอฟต์แวร์ หมายถึง สภาพแวดล้อมการทำงานที่เราได้ติดตั้งโปรแกรมต่าง ๆ เอาไว้ที่เราจะใช้ในการรันโปรแกรมต่าง ๆ สำหรับในกรณีทั่วไป เรามักจะมีสภาพแวดล้อมเดียว และติดตั้งซอฟต์แวร์ทั้งหมดลงไปในสภาพแวดล้อมเดียวกัน แต่การแยกสภาพแวดล้อมเป็นสิ่งจำเป็นเมื่อทำงานกับโปรเจกต์หลาย ๆ โปรเจกต์ที่อาจต้องการเวอร์ชันของไลบรารีที่แตกต่างกัน ถ้าเราติดตั้งไพทอนผ่าน miniconda หรือ anaconda จะมีการแยกสภาพแวดล้อมออกมาต่างหากให้อยู่แล้ว และติดตั้งตัวแปลภาษาไพทอน (Python interpreter) และไลบรารีอื่น ๆ ลงไปในสภาพแวดล้อมนั้น ซึ่งมักจะถูกตั้งชื่อไว้ว่า base หากเราติดตั้งไลบรารีไว้ในสภาพแวดล้อม base แต่ว่าไปใช้ตัวแปลภาษาไพทอนที่อยู่อีกในสภาพแวดล้อมหนึ่ง เราจะไม่สามารถใช้ไลบรารีนั้นได้ เพราะว่าอยู่คนละสภาพแวดล้อมกัน

ติดตั้งไลบรารีโดยใช้ `pip`

`pip install` เป็นคำสั่งในเทอร์มินัลหรือคอมมานด์ไลน์ (ไม่ใช่คำสั่งภาษาไพทอน) ที่ใช้สำหรับการติดตั้งแพ็กเกจจาก PyPI โดยตรง คำสั่งนี้เป็นวิธีที่ง่ายที่สุดในการเพิ่มไลบรารีเข้าไปในสภาพแวดล้อมการพัฒนาภาษาไพทอนของคุณ เช่น `pip install pythainlp` จะดำเนินการดาวน์โหลดและติดตั้งไลบรารี pythainlp เราสามารถตรวจสอบว่าไลบรารีถูกติดตั้งแล้วหรือไม่โดยใช้คำสั่ง `pip list` ซึ่งจะแสดงรายการแพ็กเกจทั้งหมดที่ติดตั้งอยู่ในสภาพแวดล้อมนั้น

ถ้าหากใช้ไพทอนผ่าน jupyter notebook หรือ Google Colab ให้ใช้เครื่องหมาย `!` เพื่อบ่งบอกว่าเราจะใช้คำสั่งในระบบคอมมานด์ไลน์ตามด้วย `pip install ชื่อไลบรารี` ถ้าหากใช้ผ่านเทอร์มินัล เช่น โปรแกรม Terminal ของระบบปฏิบัติการ MacOS หรือ โปรแกรม Anaconda Prompt ของระบบปฏิบัติการ Windows ก็สามารถพิมพ์ `pip install ชื่อไลบรารี` ได้เลย ทั้งสองกรณีเราจะได้ เอาท์พุตดังตัวอย่างข้างล่าง

```
Collecting pythainlp
  Downloading pythainlp-5.0.1-py3-none-any.whl (17.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 17.9/17.9 MB 30.9 MB/s eta 0:00:00
Requirement already satisfied: requests>=2.22.0 in /usr/local/lib/python3.10/dist-packages (from pytha
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from req
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from req
Installing collected packages: pythainlp
Successfully installed pythainlp-5.0.1
```

คำสั่งนี้จะเริ่มจากการดาวน์โหลดโค้ดของไลบรารี pythainlp และอ่านว่าจะต้องติดตั้งไลบรารีอะไรอื่นก่อนหรือไม่ เพราะว่าไลบรารี pythainlp อาจจะพึ่งพิงไลบรารีอื่น ๆ อีก เราเรียกว่าไลบรารีเหล่านี้ว่าสิ่งพึ่งพิง (dependencies) ในตัวอย่างข้างบนไลบรารี pythainlp

พึ่งพิงไลบรารีชื่อว่า requests charset-normalizer idna urllib3 และ certifi เวอร์ชันตามที่ pythainlp เวอร์ชันนี้กำหนดไว้ หลังจากนั้นจึงติดตั้ง pythainlp ลงสภาพแวดล้อมที่เรารันคำสั่ง `pip install` เมื่อเสร็จสิ้นแล้วจะแสดงข้อความว่า `Successfully installed pythainlp-5.0.1` หมายความว่าไลบรารี pythainlp เวอร์ชั่น 5.0.1 ได้ถูกติดตั้งเรียบร้อยแล้ว และเมื่อติดตั้งเสร็จแล้ว ไม่จำเป็นต้องติดตั้งอีกครั้งเมื่อต้องการเรียกใช้ เพราะว่าโค้ดทั้งหมดของไลบรารีนี้ได้ติดตั้งอยู่ในสภาพแวดล้อมนี้ที่อยู่ในเครื่องของเราแล้ว

เราสามารถลองรันคำสั่ง `import pythainlp` เพื่อทดสอบอีกครั้งได้ ถ้าหากไม่มีข้อผิดพลาดขึ้น แสดงว่าไลบรารี pythainlp ถูกติดตั้งเรียบร้อยแล้ว แต่ถ้าหากโปรแกรมแจ้งขึ้นมาว่า `ModuleNotFoundError: No module named 'pythainlp'` หมายความว่าไลบรารี pythainlp ไม่ได้ถูกติดตั้ง หรือติดตั้งไม่สำเร็จ

คำสั่ง `pip list` เป็นคำสั่งในระบบจัดการแพ็กเกจที่ใช้สำหรับแสดงรายการของแพ็กเกจที่ได้ติดตั้งไว้ในสภาพแวดล้อมการพัฒนา ณ ขณะนั้น เราสามารถใช้คำสั่งนี้เพื่อตรวจสอบแพ็กเกจต่าง ๆ ที่ได้รับการติดตั้งแล้ว เวอร์ชันของแพ็กเกจ รวมทั้งเพื่อยืนยันว่าแพ็กเกจที่ต้องการใช้งานได้ถูกติดตั้งเรียบร้อยหรือไม่ การใช้งานคำสั่งนี้ทำได้โดยพิมพ์ `pip list ชื่อไลบรารี` และกด Enter จากนั้นระบบจะแสดงรายชื่อแพ็กเกจที่ติดตั้งอยู่พร้อมกับเวอร์ชันของแต่ละแพ็กเกจออกมา

ติดตั้งไลบรารีโดยใช้ `conda`

`conda install` เป็นอีกหนึ่ง คำสั่งในเทอร์มินัลหรือคอมมานด์ไลน์ ซึ่งใช้ในการติดตั้งไลบรารีผ่านเครื่องมือจัดการไลบรารี คำสั่ง `conda` เป็นส่วนหนึ่งของ Anaconda และ Miniconda เพราะฉะนั้นถ้าเราไม่ได้ติดตั้งภาษาไพทอนผ่าน Anaconda หรือ Miniconda เราจะไม่มีคำสั่ง `conda` เช่น หากเราใช้งานไพทอนผ่าน Google Colab เราก็จะไม่มีคำสั่ง `conda` เพราะว่า Google Colab สร้างสภาพแวดล้อมในการเขียนโค้ดเป็นของมันเอง และไม่ได้ติดตั้งโปรแกรมผ่าน Anaconda คำสั่ง `conda` มีประโยชน์ในการจัดการสภาพแวดล้อมการพัฒนาและไลบรารีสำหรับภาษาไพทอน คำสั่งนี้มีวิธีการใช้งานที่คล้ายกับ `pip` แต่มีคุณสมบัติเพิ่มเติม เช่น สามารถสร้างและจัดการสภาพแวดล้อมการพัฒนาแยกต่างหากได้ การตรวจสอบว่าการติดตั้งด้วย `conda install` เสร็จสมบูรณ์แล้วหรือไม่ สามารถทำได้โดยใช้คำสั่ง `conda list` ซึ่งจะแสดงรายการแพ็กเกจที่ติดตั้งในสภาพแวดล้อม Conda นั้น ๆ

อาร์กิวเมนต์โดยปริยาย (default argument)

ไลบรารีทุกไลบรารีมักจะมีเอกสารประกอบการใช้งาน (documentation) ที่อธิบายฟังก์ชันและคลาสต่าง ๆ ที่มีให้ใช้งานในไลบรารีทั้งหมด อีกทั้งรวบรวมตัวอย่างการใช้งานที่ทำให้เข้าใจวิธีการใช้งานได้เร็วยิ่งขึ้น ซึ่งช่วยให้เราสามารถใช้ไลบรารีได้อย่างสะดวกสบายและเต็มศักยภาพ สามารถปรับใช้และผสานรวมเข้ากับโค้ดของตัวเองได้ง่ายขึ้น

ฟังก์ชันในไลบรารีมักจะมีอาร์กิวเมนต์จำนวนมาก ข้อดีของการออกแบบนี้คือมันช่วยให้ฟังก์ชันนั้นมีความยืดหยุ่นและสามารถปรับแต่งได้หลายอย่าง เพื่อตอบสนองความต้องการที่แตกต่างกันของผู้ใช้ อย่างไรก็ตามการเรียกใช้ฟังก์ชันที่มีอาร์กิวเมนต์จำนวนมากอาจทำให้เกิดความซับซ้อนและความยากลำบากในการทำความเข้าใจว่าแต่ละอาร์กิวเมนต์ทำงานอย่างไร อาร์กิวเมนต์โดยปริยาย (default argument) จึงเข้ามาเป็นสิ่งจำเป็น

อาร์กิวเมนต์โดยปริยาย ช่วยลดความจำเป็นในการระบุอาร์กิวเมนต์ทุกครั้งที่เรียกใช้ฟังก์ชัน ข้อดีหลัก ๆ คือการลดความซับซ้อนในการใช้งานฟังก์ชันแต่ยังคงความยืดหยุ่นของการใช้ฟังก์ชัน เพราะว่าผู้ใช้ไม่จำเป็นต้องระบุอาร์กิวเมนต์ทั้งหมดจึงจะใช้งานได้ ทำให้ผู้ใช้สามารถปรับแต่งการใช้ฟังก์ชันเฉพาะเจาะจงกับสถานการณ์การใช้นั้น ตัวอย่างเช่น ในไลบรารีการวาด (plot) กราฟ ฟังก์ชันสำหรับการวาดเส้นอาจมีอาร์กิวเมนต์สำหรับสี ความหนาของเส้น และลักษณะของเส้น (เช่น เส้นประหรือเส้นต่อเนื่อง) โดยค่าโดยปริยายสำหรับสีอาจเป็นสีดำ ความหนาของเส้นเป็น 1 หน่วย และเป็นเส้นต่อเนื่อง ถ้าผู้ใช้ไม่ได้ระบุอาร์กิวเมนต์เหล่านี้ ฟังก์ชันจะใช้ค่าโดยปริยายเหล่านี้เป็นค่าเริ่มต้น ผู้ใช้ไม่จำเป็นต้องระบุค่าเหล่านี้เมื่อใช้ฟังก์ชันเลย ผู้ใช้ระบุค่าอาร์กิวเมนต์เมื่อต้องการปรับแต่งการใช้งานฟังก์ชันให้ตรงกับความต้องการของตนเองเท่านั้น

เรากำหนดอาร์กิวเมนต์โดยปริยายได้โดยการใช้ `=` ตอนที่ประกาศฟังก์ชัน ตัวอย่างเช่น

```
def calc_volume(width, length, height=10):
    return width * length * height
```

ฟังก์ชันนี้กำหนดว่าค่าของพารามิเตอร์ `height` เป็น 10 ถ้าผู้ใช้ไม่ได้ระบุค่า `height` เพราะฉะนั้นเมื่อเรียกใช้ฟังก์ชัน `calc_volume` เราไม่จำเป็นต้องใส่พารามิเตอร์ครบทั้ง 3 ตัว เช่น

```
calc_volume(5, 3, 2)  # คืนค่า 5 * 3 * 2
calc_volume(5, 3) # คืนค่า 5 * 3 * 10
```

การเรียกฟังก์ชันตามปกติดังตัวอย่างข้างต้น เราเรียกว่าการใช้อาร์กิวเมนต์แบบเรียงตำแหน่ง (positional argument) เพราะว่าค่าของอาร์กิวเมนต์จะถูกใส่ตามลำดับที่เรียงอยู่บนฟังก์ชัน ซึ่งเป็นรูปแบบการเรียกใช้ฟังก์ชันที่เราใช้กันโดยส่วนใหญ่ แต่ถ้าเราต้องการเปลี่ยนค่าของอาร์กิวเมนต์ที่ไม่ใช่ตามลำดับที่เรียงอยู่บนฟังก์ชัน เราสามารถเรียกฟังก์ชันและระบุชื่ออาร์กิวเมนต์ที่ต้องการเปลี่ยนค่าโดยตรงได้ วิธีการนี้เรียกว่าการใช้อาร์กิวเมนต์แบบตั้งชื่อ (named argument หรือ keyword argument) ได้ ตัวอย่างเช่น

```
calc_volume(length=3, width=5, height=2)
```

การใช้อาร์กิวเมนต์แบบตั้งชื่อ ช่วยให้เราสามารถใส่ค่าอาร์กิวเมนต์ที่ต้องการเปลี่ยนได้โดยไม่ต้องสนใจลำดับที่ประกาศเรียงอยู่บนส่วนหัวของฟังก์ชัน

ถ้าหากเราต้องการใช้อาร์กิวเมนต์แบบเรียงตำแหน่งเพียงอย่างเดียว เราต้องระบุอาร์กิวเมนต์ให้ครบทุกตัว และค่าจะถูกใส่ตามลำดับที่เรียงอยู่บนฟังก์ชัน ถ้าเราไม่ใส่ค่าอาร์กิวเมนต์ให้ครบ โปรแกรมจะแจ้งข้อผิดพลาดขึ้นว่าไม่ได้ใส่ค่าอาร์กิวเมนต์ครบทุกตัว เว้นเสียแต่ว่าอาร์กิวเมนต์นั้นมีค่าโดยปริยาย และถ้าหากต้องการใช้อาร์กิวเมนต์แบบตั้งชื่อผสมกับอาร์กิวเมนต์แบบเรียงตำแหน่ง จะต้องใช้ตามหลังอาร์กิวเมนต์แบบเรียงตำแหน่งเท่านั้น ตัวอย่างเช่น

คำสั่ง	ถูกต้องหรือไม่	สาเหตุ
`calc_volume(5)`	❌	ไม่ได้ เพราะใส่ค่าไม่ครบ โปรแกรมไม่ทราบว่าค่าของอาร์กิวเมนต์ `length` จะต้องเป็นค่าอะไร
`calc_volume(5, 3, height=2)`	✅	จะผสมก็ได้ แต่ต้องใช้อาร์กิวเมนต์แบบตั้งชื่อหลังสุด
`calc_volume(length=2, 5, 10)`	❌	อาร์กิวเมนต์แบบตั้งชื่อต้องมาทีหลังสุด
`calc_volume(length=2, width=5, 10)`	❌	อาร์กิวเมนต์แบบตั้งชื่อต้องมาทีหลังสุด

ตัวอย่างฟังก์ชันที่มีการใช้อาร์กิวเมนต์โดยปริยาย 1

ขอยกตัวอย่างจากเอกสารประกอบการใช้งานของเมท็อด `str.split` ซึ่งเป็นเมท็อดที่เราเคยใช้กันมาแล้วในบทก่อน ๆ `str.split` เป็นเมท็อดที่ใช้สำหรับการตัดข้อความให้เป็นลิสต์ โดยใช้ตัวแบ่งที่เรากำหนด ส่วนหัวของฟังก์ชันจากเอกสารประกอบการใช้งานของไพทอนมีดังนี้

```
str.split(sep=None, maxsplit=-1)
```

เราเห็นได้ว่าเมท็อดนี้มีพารามิเตอร์ 2 ตัว ได้แก่

- `sep` ซึ่งมีอาร์กิวเมนต์โดยปริยายเป็น `None`
- `maxsplit` ซึ่งมีอาร์กิวเมนต์โดยปริยายเป็น `-1`

เพราะฉะนั้นเวลาเรียกใช้เมท็อด `str.split` เราสามารถใช้ได้โดยไม่ต้องระบุอาร์กิวเมนต์ใด ๆ และผู้เขียนฟังก์ชันคิดมาแล้วว่าอาร์กิวเมนต์โดยปริยายเป็นค่าที่เหมาะสมในกรณีทั่วไป แต่ว่ายังคงให้ความยืดหยุ่นกับผู้ใช้ในการระบุอาร์กิวเมนต์เพื่อปรับแต่งการใช้งานได้

ตัวอย่างฟังก์ชันที่มีการใช้อาร์กิวเมนต์โดยปริยาย 2

ตัวอย่างนี้มาจากฟังก์ชัน `nltk.tokenize.sent_tokenize` ในไลบรารี nltk ซึ่งใช้สำหรับการตัดประโยคจากข้อความ ส่วนหัวของฟังก์ชันนี้และวิธีการใช้งานตามที่ปรากฏในเอกสารประกอบการใช้งานของ nltk มีดังนี้

```
nltk.tokenize.sent_tokenize(text, language='english')

Return a sentence-tokenized copy of text, using NLTK's recommended sentence tokenizer (currently Punkt

Parameters
text — text to split into sentences
language — the model name in the Punkt corpus
```

ฟังก์ชันนี้มีพารามิเตอร์ 2 ตัว ได้แก่

- `text` ซึ่งเป็นพารามิเตอร์ที่ต้องระบุเสมอ
- `language` ซึ่งมีอาร์กิวเมนต์โดยปริยายเป็น `'english'`

เมื่อเราใช้ฟังก์ชันนี้ เราไม่จำเป็นต้องระบุว่ากำลังตัดประโยคภาษาอะไร อาร์กิวเมนต์โดยปริยายคือภาษาอังกฤษ แต่ว่าเราสามารถเปลี่ยนค่านี้ได้เมื่อต้องการปรับใช้กับภาษาอื่น ๆ ที่มีชื่ออยู่ใน Punkt corpus ได้

ตัวอย่างการใช้งานของฟังก์ชันนี้

```
import nltk

nltk.tokenize.sent_tokenize() # ไม่ได้ เพราะว่าไม่ได้ระบุค่าของอาร์กิวเมนต์ text ซึ่งไม่มีค่าโดยปริยาย

nltk.tokenize.sent_tokenize("The U.S. have dots. Mr. Robert met Dr. Evil in the lab.")  # ได้

nltk.tokenize.sent_tokenize("The U.S. have dots. Mr. Robert met Dr. Evil in the lab.",
    language='english') # ได้

nltk.tokenize.sent_tokenize("The U.S. have dots. Mr. Robert met Dr. Evil in the lab.",
    language='french')  # ได้ แต่ว่าผลอาจจะออกมาไม่ถูกต้อง
```

การทำความสะอาดข้อมูล (data cleaning)

ขั้นตอนแรกของการประมวลผลข้อมูล คือ การทำความสะอาดข้อมูล ข้อมูลที่เราได้รับมามักจะไม่สะอาด มีอักขระที่ไม่ต้องการปนอยู่ ข้อมูลอาจจะยังมีความผิดปกติอยู่ ถ้าหากเราไม่ทำความสะอาดอย่างระมัดระวังให้มีความเพี้ยนน้อยลง ให้เหลือเฉพาะส่วนที่เราต้องการวิเคราะห์ อาจจะเกิดอุปสรรคในการวิเคราะห์ข้อมูล ทำให้วิเคราะห์คลาดเคลื่อน เชื่อถือผลการวิเคราะห์ได้ยาก การทำความสะอาดข้อมูลไม่ได้มีสูตรสำเร็จตายตัว ผู้วิเคราะห์ข้อมูลต้องปรับกระบวนการให้เข้ากับจุดประสงค์ของการวิเคราะห์ ดังกรณีตัวอย่างต่อไปนี้

ตัวอย่างการทำความสะอาดข้อมูล 1

> RT @MatichonOnline: "บิ๊กตู่"ลั่นรบ.ทำอะไรยึดกม.ไม่ใช่ติดคุกแล้วหนี เล่นมุกพรรคร่วม "พลังปชป.ภูมิใจไทย" https://t.co/9nmOBJnhrq via @มติชนอ...

ตัวอย่างข้างบนนี้มีข้อมูลหลายส่วนที่ไม่ใช่ข้อความที่เราต้องการวิเคราะห์ ได้แก่

- RT @MatichonOnline ซึ่งหมายถึงการรีทวิตข่าวจาก MatichonOnline
- ลิงก์ https://t.co/9nmOBJnhrq ซึ่งเป็นลิงก์ที่เชื่อมไปยังเว็บไซต์ที่นำเสนอข่าวฉบับเต็มอยู่
- via @มติชนอ... ซึ่งเป็นการบอกว่าลิงก์ที่นำไปสู่เว็บไซนต์ของมติชน แต่ว่าชื่อบัญชีก็ยังมีข้อผิดพลาด

เราจึงจำเป็นต้องเขียนโค้ดเพื่อใช้นิพจน์ปรกติ หรือเรกเอกซ์ในการสกัดเอาข้อมูลส่วนที่เราไม่ต้องการออกไป ให้เหลือเพียงแค่

> "บิ๊กตู่"ลั่นรบ.ทำอะไรยึดกม.ไม่ใช่ติดคุกแล้วหนี เล่นมุกพรรคร่วม "พลังปชป.ภูมิใจไทย"

ถ้าเราเก็บข้อมูลที่ยังไม่ได้ทำความสะอาดใส่ตัวแปรชื่อว่า `tweet` เราสามารถเขียนโค้ดในการทำความสะอาดได้ดังนี้

```
import re
tweet = 'RT @MatichonOnline: “บิ๊กตู่”ลั่นรบ.ทำอะไรยึดกม.ไม่ใช่ติดคุกแล้วหนี เล่นมุกพรรคร่วม “พลังปชป.ภูมิใจไทย” https://t
# Remove RT @username
tweet = re.sub(r'RT @\w+: ', '', tweet)
# Remove URL that begins with http
tweet = re.sub(r'https?://\S+', '', tweet)
# Remove via @username
tweet = re.sub(r' via @\S+', '', tweet)
```

ซึ่งเราอาจจะรวมเป็นฟังก์ชันที่ทำให้เราใช้งานกับทวีตอื่น ๆ ได้ด้วย ดังนี้

```
def clean_tweet(tweet):
    # Remove RT @username
    tweet = re.sub(r'RT @\w+: ', '', tweet)
    # Remove URL that begins with http
    tweet = re.sub(r'https?://\S+', '', tweet)
    # Remove via @username
    tweet = re.sub(r' via @\S+', '', tweet)
    return tweet
```

ตัวอย่างการทำความสะอาดข้อมูล 2

> บกพร่องโดยสุจริต VS อยู่-ไม่-เป็น | ขยี้คดีโกง | 10 พ.ย. 62 | (3/3) https://t.co/atUF6PrXdx via @YouTube

ตัวอย่างข้างต้นนี้มีข้อมูลที่เราไม่ต้องการอยู่ด้วยหลายส่วน ได้แก่

- (3/3) ซึ่งหมายถึง ทวีตนี้เป็นทวีตที่ 3 ในชุดทวีตทั้งหมด 3 ทวีต
- ลิงก์ https://t.co/atUF6PrXdx ซึ่งเป็นลิงก์ที่เชื่อมไปยังวีดีโอที่อยู่บน YouTube
- via @YouTube ซึ่งเป็นการบอกว่าลิงก์ที่นำไปสู่วีดีโอนั้นอยู่บนแพลตฟอร์ม YouTube

ในกรณีนี้จะเห็นว่าข้อมูลที่เราได้มาไม่สมบูรณ์ เนื่องจากเป็นทวีตเป็นทวีตต่อเนื่องจากสองทวีตก่อนหน้า ผู้วิเคราะห์อาจจะเลือกไม่วิเคราะห์ทวีตที่มีความต่อเนื่องกันในลักษณะนี้ หรือไม่เช่นนั้นต้องเตรียมข้อมูลให้เรียงลำดับตามการทวีตและเชื่อมทวีตเข้าไว้ด้วยกัน เช่น สมมติว่าเรามีข้อมูลลิสต์ของทวีต เราสามารถเขียนโค้ดเพื่อรวมทวีตที่มีการต่อเนื่องกัน เอาไว้ด้วยกันได้ดังนี้

```
def merge_tweet_in_sequence(tweet_list_time_sorted):
    new_list = []
    index = 0
    while (index < len(tweet_list_time_sorted)):
        # if tweet contains (1/x), merge this tweet with the next x tweets
        patt = re.compile(r'\(1/(\d)\)')
        tweet = tweet_list_time_sorted[index]
        match = patt.search(tweet)
        if match:
            num_tweets = int(match.group(1))
            merged_tweet = clean_tweet(tweet)
            for i in range(1, num_tweets):
                merged_tweet += clean_tweet(tweet_list_time_sorted[index + i])
            new_list.append(merged_tweet)
            index += num_tweets
        else:
            new_list.append(tweet_list_time_sorted[index])
            index += 1
    return merged_tweet
```

ตัวอย่างการทำความสะอาดข้อมูล 3

> ขอให้นึกถึงการท่องเที่ยวภายภาคหน้าด้วยคะ

ตัวอย่างข้างต้นนี้มีการสะกดผิดสองจุด ได้แก่

- สะกด *ขอไห้* แทนที่จะเป็น *ขอให้*
- สะกด *คะ* แทนที่จะเป็น *ค่ะ*

โดยทั่วไปแล้วเรามักจะไม่แก้การสะกดผิด เพราะส่วนใหญ่แล้วเรามักจะวิเคราะห์ข้อมูลที่มีขนาดค่อนข้างใหญ่ ใหญ่เกินที่จะให้มนุษย์วิเคราะห์เองด้วยมือได้ทันท่วงที วิธีทางสถิติหรือการใช้โมเดลทางภาษาอาศัยการจับแพตเทิร์นต่าง ๆ ที่อยู่ในข้อมูล การสะกดผิดตามสถิติแล้วมักจะเกิดขึ้นไม่มาก เมื่อเทียบกับส่วนของข้อมูลที่สะกดถูกตามหลักพจนานุกรม หรือตามความนิยมในช่วงเวลานั้น คำที่สะกดผิดมักจะไม่ปรากฏออกมาในแพตเทิร์นที่วิเคราะห์ได้ เนื่องจากปริมาณของคำที่สะกดผิดน้อยกว่าคำที่สะกดถูกมาก ๆ ดังนั้นส่วนใหญ่แล้วมักจะไม่ต้องกังวลว่าการวิเคราะห์จะผิดเพี้ยนเนื่องจากมีคำที่สะกดผิดอยู่

อีกเหตุผลหนึ่งที่เรามักจะตัดสินใจไม่แก้ไขคำที่สะกดผิดก่อนวิเคราะห์ข้อมูล คือ เราไม่มีโปรแกรมที่สามารถแก้ไขการสะกดผิดได้อย่างแม่นยำพอ เครื่องตรวจตัวสะกด (spellchecker) ที่มีอยู่ในปัจจุบันแม้มีความแม่นยำระดับหนึ่ง แต่ก็มีโอกาสที่จะแก้ไขส่วนที่ผิดให้ผิดไปอีกแบบหนึ่งอยู่บ้าง หรือเปลี่ยนส่วนที่ถูกอยู่แล้วให้เป็นผิด ในกรณีที่ผลการวิเคราะห์ออกมาแปลกหรือผิดเพี้ยนจากที่สิ่งที่เราคาดการณ์ไว้มากเกินไป กลับกลายเป็นว่าผู้วิเคราะห์ต้องทำการตรวจสอบขั้นตอนการตรวจสะกดเพิ่มไปอีกขั้นตอนหนึ่ง ทำให้หาจุดที่ทำให้วิเคราะห์ผิดเพี้ยนยากขึ้นอีกขั้นตอนหนึ่ง

ตัวอย่างการทำความสะอาดข้อมูล 4

> ชาล็อตพาร์ทแบบนี้จิ้งเว่อ น้องเก่งมาก มืออาชีพสุดๆ ขนาดพี่ในกองส่งเสียงชมไม่หยุด สวยมาก ดีมากกกกกก #ENGLOTshootingTvcWinkwhite @itscharlotty (ที่มา: Twitter @vanitcheryl วันที่ 2 มีนาคม 2567)

ในโลกโซเชียลเรามักจะพบภาษาไม่ได้เป็นไปตามมาตรฐาน ในตัวอย่างนี้เราพบลักษณะของภาษาโซเชียลหลายจุด ได้แก่

- การใช้ไทยคำอังกฤษคำ เช่น *พาร์ท*
- การใช้คำแสลง เช่น *จิ้ง เว่อ*
- การสะกดแบบไม่มาตรฐาน เช่น *มากกกกกก*

ตามหลักภาษาศาสตร์แล้ว ภาษาที่เป็นมาตรฐานเป็นภาษาที่มีคนกลุ่มใดกลุ่มหนึ่งในสังคมเป็นคนกำหนดมาว่าเป็นมาตรฐาน คนกลุ่มนั้นมักจะเป็นกลุ่มคนที่มีสถานะทางสังคมและเศรษฐกิจสูง คนกลุ่มนี้มักจะกระทำการใด ๆ ให้สังคมยอมรับว่ารูปแบบภาษาที่ตนตั้งขึ้นมานั้นเป็นภาษาสำหรับคนที่ได้รับการยอมรับนับถือจากสังคมด้วยวิธีต่าง ๆ ในมุมมองของนักภาษาศาสตร์เห็นว่าเป็นการปะทะสังสรรค์ระหว่างปัจจัยทางสังคม และรูปแบบของภาษา การศึกษาภาษาในบริบทของสังคมและวัฒนธรรม เรียกว่า ภาษาศาสตร์สังคม (sociolinguistics) จากแง่มุมนี้ภาษาเป็นเพียงเครื่องมือที่ใช้ในการสื่อสาร และทุกภาษาต่างมีแบบแผนของตัวมันเอง การตีค่าว่ารูปแบบภาษาใดดีกว่าอีกรูปแบบภาษาหนึ่งนั้น ล้วนแต่เป็นสิ่งที่ถูกสร้างขึ้นโดยสังคม (social construct)

หากเราต้องการวิเคราะห์ข้อมูลที่มาจากโลกโซเชียลมักจะไม่มีความจำเป็นใด ๆ ที่จะต้องแก้ภาษาให้เป็นมาตรฐาน เนื่องจากภาษาในโลกโซเชียลปรากฏเป็นแพตเทิร์นของมันเอง วิธีทางสถิติและแบบจำลองทางภาษาอาศัยการหาแพตเทิร์นที่เกิดขึ้นซ้ำ ๆ อยู่แล้ว เช่น คำว่า จิ้ง ปรากฏอยู่ถึง 6.7 ล้านครั้ง และ คำว่า *เว่อ* ปรากฏอยู่ถึง 18.7 ล้านครั้ง เมื่อลองค้นหาคำเหล่านี้บน Google (วันที่ 2 มีนาคม 2567) ซึ่งแสดงให้เห็นว่าสองคำนี้กลายเป็นส่วนหนึ่งของภาษาแล้ว สังคมบนโลกอินเตอร์เน็ตยอมรับและนำไปใช้ต่ออย่างแพร่หลายในขณะนั้น

ส่วนคำว่า *มากกกกกก* เราอาจจะทำความสะอาดให้เหลือ ก เพียงตัวเดียว เพราะคนแต่ละคนอาจจะใช้จำนวนตัว ก ไม่เท่ากัน ทำให้เครื่องมือตรวจจับได้ยากว่าใช้คำว่า *มาก* ไปแล้วกี่ครั้ง กระบวนการนี้เราเรียกว่าการเปลี่ยนให้เป็นมาตรฐาน (normalization) สำหรับภาษาไทยเรามักจะใช้กฎง่าย ๆ ในการเปลี่ยนให้เป็นมาตรฐาน โดยการตรวจจับว่ามีตัวอักษรเดียวกัน ตั้งแต่สามตัวขึ้นไปหรือไม่ ถ้ามีให้ทำให้เหลือตัวเดียว ซึ่งสามารถทำโดยใช้เรกเอกซ์ดังตัวอย่างโค้ดดังนี้

```
import re
def normalize(tweet):
    # ถ้าเจอ [ก-์] สามตัวขึ้นไป ทำให้เหลือตัวเดียว
    tweet = re.sub(r'([ก-์])\1{2,}', r'\1', tweet)
    return tweet
```

คำอธิบายเรกเอกซ์ที่ใช้ในโค้ดด้านบนคือ

- `([ก-์])` หมายถึง ตัวอักษรไทยที่อยู่ในช่วง ก ถึง ์ และเก็บไว้ในกลุ่มที่ 1 (เครื่องหมายวงเล็บคู่แรก)
- `\1{2,}` หมายถึง อ้างกลับถึงตัวอักษรที่อยู่ในกลุ่มที่หนึ่ง (`\1`) ตรวจว่าเจอตั้งแต่ 2 ตัวขึ้นไปหรือไม่ (`{2,}`)
- `r'\1'` หมายถึง ตัวอักษรที่ตรวจพบและเก็บอยู่ในกลุ่มที่ 1

ตัวอย่างการใช้

```
normalize('คิดถึงงงงงมากกกกก') # คิดถึงมาก
```

ขั้นตอนสุดท้ายของการทำความสะอาดข้อมูลคือ การกำจัดตัวซ้ำ (deduplication) โดยทั่วไปแล้วข้อมูลแต่ละชิ้นมักจะไม่ซ้ำกัน ชุดข้อมูลทีมาจากทวิตเตอร์อาจจะมีซ้ำกันบ้าง เมื่อผู้ใช้รีทวีตผู้ใช้อีกคนหนึ่ง เมื่อนำข้อความ *RT @* ตามด้วยชื่อผู้ใช้ออกไปแล้ว ก็จะเหลือเพียงข้อความที่เหมือนกับเจ้าของทวีต ถ้าในชุดข้อมูลมีทวีตที่ถูกรีทวีตบ่อย ๆ ก่อให้เกิดแถวที่มีข้อมูลซ้ำ ๆ กันมากมาย ทำให้ค่าสถิติของคำถูกบิดเบือนไป เพราะฉะนั้นเราจึงจำเป็นต้องนำข้อมูลที่ซ้ำออกไป วิธีการที่ง่ายที่สุดคือใช้เซตในการเก็บข้อมูลที่เคยเจอแล้ว ดังนี้

```
def deduplicate(tweet_list):
    seen = set()
    deduplicated_tweet_list = []
    for tweet in tweet_list:
        if tweet not in seen:
            deduplicated_tweet_list.append(tweet)
            seen.add(tweet)
    return deduplicated_tweet_list
```

เมื่อรวมโค้ดทั้งหมดเข้าด้วยกัน จะได้ฟังก์ชันที่ทำความสะอาดข้อมูลทวีตได้ดังนี้

```
cleaned_normalized_tweets = []
for tweet in tweet_list:
    tweet = clean_tweet(tweet)
    tweet = normalize(tweet)
    cleaned_normalized_tweets.append(tweet)
dataset = deduplicate(cleaned_normalized_tweets)
```

หรืออาจจะรวมกันเป็นฟังก์ชันเดียวกันได้ดังนี้

```
def clean_normalize_tweet(tweet):
    tweet = clean_tweet(tweet)
    tweet = normalize(tweet)
    return tweet

merged_tweet_list = merge_tweet_in_sequence(tweet_list)
cleaned_normalized_tweets = [clean_normalize_tweet(tweet) for tweet in merged_tweet_list]
dataset = deduplicate(cleaned_normalized_tweets)
```

ทั้งนี้เราจะเห็นว่าเราต้องใช้ฟังก์ชัน 4 ฟังก์ชันในการประมวลผลทำความสะอาดข้อมูล `merge_tweet_in_sequence`, `clean_tweet`, `normalize`, และ `deduplicate`
ซึ่งการแยกออกมาเป็น 4 ฟังก์ชันนั้นมีข้อดีคือทำให้เราปรับแก้เป็นส่วน ๆ ไปได้ ถ้าหากว่าเราต้องการปรับวิธีการทำความสะอาดข้อมูลให้นำเอาแฮชแท็กออกไปด้วย เราสามารถเพิ่มขั้นตอนนี้เข้าไปใน `clean_tweet` หรือเขียนฟังก์ชัน `remove_hashtag` และเรียกใช้ในฟังก์ชัน `clean_tweet` อีกชั้นหนึ่งก็ได้ ขึ้นอยู่กับการตัดสินใจของผู้ที่เขียนโปรแกรมว่าแบบใดเข้าใจง่ายกว่า

การแปลงเป็นโทเค็น (tokenization)

ขั้นตอนต่อจากการทำความสะอาดข้อมูล คือ การแปลงเป็นโทเค็น (tokenization) กระบวนการนี้เป็นกระบวนการพิเศษสำหรับการเตรียมข้อมูลจากชุดข้อมูลที่เป็นตัวอักษร โทเค็น คือ หน่วยที่เล็กที่สุดที่ใช้ในการวิเคราะห์ หน่วยที่เล็กที่สุดที่ใช้ในการวิเคราะห์ข้อมูลตัวอักษรคืออะไร โดยทั่วไปแล้วผู้วิเคราะห์ข้อมูลสามารถกำหนดได้ว่าอยากให้โทเค็นเป็นอะไร ส่วนมากเราจะอ้างอิงหลักการการวิเคราะห์ภาษาจากภาษาศาสตร์ คือใช้คำเป็นโทเค็นในการวิเคราะห์ความหมายของประโยค เพราะฉะนั้นการวิเคราะห์ความหมายของข้อความมักจะต้องอาศัยการเปลี่ยนสตริงให้เป็นลิสต์ของคำ เรียกว่า การตัดคำ (word segmentation) บางครั้งเราเรียกกระบวนการแปลงให้เป็นโทเค็น

ว่าการตัดคำ ถึงแม้ที่จริงแล้วการแปลงให้เป็นโทเค็นเป็นมโนทัศน์ที่กว้างกว่าการตัดคำ เพราะเราสามารถกำหนดให้โทเค็นเป็นอะไรก็ได้ ไม่จำเป็นต้องเป็นคำเท่านั้น

คำ ในเชิงภาษาศาสตร์ คำ คือ หน่วยที่เล็กที่สุดของภาษาที่ยังคงสื่อความหมายได้ด้วยตัวเอง ตัวอย่างเช่น

ข้อความ	เป็นคำหรือไม่	เหตุผล
สวัสดีครับ	ไม่ใช่	เป็นกลุ่มคำที่มีสองคำอยู่ *สวัสดี* และ *ครับ* ต่างเป็นคำที่สื่อความหมายด้วยตัวเอง
ตัดหญ้า	ไม่ใช่	เป็นกลุ่มคำที่มีสองคำอยู่ *ตัด* และ *หญ้า* ต่างเป็นคำที่สื่อความหมายด้วยตัวเอง ความหมายของ *ตัดหญ้า* มาจากการรวมความหมายของคำสองคำ
ตัดใจ	ใช่	เป็นคำที่สื่อความหมายด้วยตัวเอง และไม่ใช่การประกอบความหมายของ *ตัด* และ *ใจ*
คิดถึง	ใช่	เป็นคำเดียวกันที่สื่อความหมายด้วยตัวเอง และไม่ใช่การประกอบความหมายของ *คิด* และ *ถึง*
เดินทาง	ใช่	เป็นคำเดียวกันที่สื่อความหมายด้วยตัวเอง และไม่ใช่การประกอบความหมายของ *เดิน* และ *ทาง*

จากตัวอย่างข้างต้น จะเห็นได้ว่าการตัดคำต้องอาศัยเกณฑ์ทางความหมาย เพื่อตัดสินว่าสตริงที่มีสตริงย่อยเป็นคำ (เช่น *ตัด*/*ใจ* หรือ *ตัด*/*หญ้า*) เป็นกลุ่มคำ หรือคำเดี่ยว หากว่าความหมายไม่ได้ต่างจากการนำคำย่อยมารวมกันให้จัดว่าเป็นกลุ่มคำ ตัวอย่างเช่น คำว่า *ตัดหญ้า* มีความหมายจากคำกริยา *ตัด* รวมกับคำนามที่เป็นกรรม *หญ้า* มารวมกัน ดังนั้นเราจึงจัดเป็นคำสองคำมาอยู่ใกล้กันเป็นกลุ่มคำ ในขณะที่คำว่า *ตัดใจ* มีความหมายว่า เลิกคิด และไม่ได้มาจากการนำคำย่อย *ตัด* และ *ใจ* มารวมกัน ดังนั้นเราจึงจัดเป็นคำเดี่ยว [Aroonmanakun, 2007]

คำถามชวนคิด

จากกรณีให้ลองคิดว่ากรณีใดบ้างที่เป็นคำ กรณีใดบ้างที่เป็นกลุ่มคำ

- *ตู้เย็น*
- *ตะกร้าผ้า*
- *ขวดแก้ว*
- *หมอฟัน*

เฉลย

- *ตู้เย็น* เป็นคำเดี่ยว เพราะว่าความหมายคือ เครื่องใช้ไฟฟ้าที่ทำความเย็น ซึ่งห่างจากความหมายของคำย่อย *ตู้* และ *เย็น*
- *ตะกร้าผ้า* เป็นกลุ่มคำ เพราะว่าความหมายคือ ตะกร้าที่ใช้ใส่ผ้า ซึ่งมาจากการนำคำย่อย *ตะกร้า* และ *ผ้า* มารวมกัน
- *ขวดแก้ว* เป็นกลุ่มคำ เพราะว่าความหมายคือ ขวดที่ทำจากแก้ว ซึ่งมาจากการนำคำย่อย *ขวด* และ *แก้ว* มารวมกัน
- *หมอฟัน* เป็นกลุ่มคำ เพราะว่าความหมายคือ ผู้เชี่ยวชาญที่ทำงานเฉพาะทางที่เกี่ยวกับฟัน ซึ่งมาจากการนำคำย่อย *หมอ* และ *ฟัน* มารวมกัน สามารถใช้หลักคิดนี้ได้กับ *หมอตา หมอผี หมอกระดูก*

ในบางกรณีการพิจารณาว่าอะไรจัดเป็นคำ อะไรจัดเป็นกลุ่มคำ เป็นเรื่องที่ไม่ชัดเจนสักทีเดียว อาจจะทำให้เกิดการถกเถียงว่าความหมายโดยรวมเกิดจากการนำความหมายของสองคำย่อยมารวมกันหรือไม่ ในกรณีดังกล่าว เรามักจะตัดคำให้มีจำนวนคำมากที่สุด เพราะถ้าหากหลักการวิเคราะห์เปลี่ยนไปเรายังสามารถนำคำที่ตัดไปแล้วมารวมกันใหม่ได้

การตัดคำโดยอัตโนมัติสามารถทำด้วย 3 วิธีใหญ่ ๆ ได้แก่

1. การตัดคำแบบอิงกฎเกณฑ์ (rule-based word segmentation)
2. การตัดคำแบบอิงคลังศัพท์ (lexicon-based word segmentation)
3. การตัดคำแบบอิงการเรียนรู้ด้วยเครื่อง (machine-learning-based word segmentation)

การตัดคำแบบอิงกฎเกณฑ์

บางภาษาเราสามารถตั้งกฎเกณฑ์ผ่านการเขียนเรกเอกซ์เพื่อตัดคำได้ เช่น ภาษาอังกฤษ ภาษาเยอรมัน ภาษาอิตาเลียน และภาษาอื่น ๆ ที่ใช้ตัวอักษรลาติน กฎเกณฑ์ที่ตั้งอาจจะเป็นเรกเอกซ์เพื่อบอกว่าจะแพตเทิร์นไหนเป็นตัวแบ่งบ้าง เช่น เราอาจจะใช้เครื่องหมายวรรคตอนและช่องว่างเป็นตัวแบ่ง ซึ่งตรงกับเรกเอกซ์ `r'\s+'` เพราะว่า `\s` หมายถึงตัวอักษรที่เป็น whitespace ได้แก่ ช่องว่าง แท็บ และการขึ้นบรรทัดใหม่ และเราเผื่อว่ามีการใช้แท็บหลาย ๆ ครั้ง หรือขึ้นบรรทัดใหม่หลาย ๆ ครั้ง เมื่อเราอ่านเอาข้อมูลมาจากไฟล์ที่มีหลายบรรทัด ยกตัวอย่างเช่น

```
import re
text = "Got a long list of ex-lovers, they'll tell you I'm insane (Yeah) "
tokens = re.split(r'\s+', text)
print(tokens)
```

ได้ผลลัพธ์เป็น

```
['Got', 'a', 'long', 'list', 'of', 'ex-lovers,', "they'll", 'tell', 'you', "I'm", 'insane', '(Yeah)',
```

เราสังเกตเห็นได้ว่าโทเค็นที่ถูกต้องมีทั้งหมด 8 ตัว และผิด 4 ตัว ได้แก่ *ex-lovers, they'll I'm* และ *(Yeah)* เพราะว่าเครื่องหมายวรรคตอนไปติดอยู่กับคำ ใช้ `re.split` เป็นวิธีง่ายเหมาะกับการวิเคราะห์ข้อมูลเบื้องต้น ไม่ต้องการความละเอียดมาก แต่ว่าถูกต้องประมาณ 70% เท่านั้นสำหรับภาษาอังกฤษ

อีกวิธีที่นิยมในการตัดคำ คือ การเขียนเรกเอกซ์เพื่อบอกแพตเทิร์นของโทเค็น แทนที่จะเขียนเรกเอกซ์เพื่อบอกแพตเทิร์นของตัวแบ่ง สำหรับภาษาที่ใช้ตัวอักษรลาติน หรือภาษาอื่น ๆ ที่มีการใช้ตัวแบ่งระหว่างคำชัดเจน เรามักจะใช้เรกเอกซ์ `\w+|[^\w\s]+` ซึ่งประกอบด้วยสองแพตเทิร์นย่อย ได้แก่

- `\w+` เราต้องการโทเค็นที่เป็น ตัวเลขหรือตัวอักษร a-z (ตัวลาติน) รวมถึงตัวอักษรที่มีเครื่องหมายแสดงการออกเสียงที่อยู่บนหรือล่างตัวอักษร (diacritics) เช่น é หรือ ä หรือตัวอักษรจากระบบการเขียนอื่น ๆ ที่จัดว่าเป็นตัวหนังสือ ไม่ใช่เครื่องหมายวรรคตอน
- `[^\w\s]+` เราต้องการโทเค็นที่เป็น เครื่องหมายวรรคตอนล้วนหรือเครื่องหมายอื่น ๆ ที่ไม่ใช่ตัวเลขหรือตัวอักษร โทเค็นนี้จะมีแต่เครื่องหมายวรรคตอนไม่มีตัวอักษรปนอยู่ เพื่อที่จะแยก `ex-lovers` เป็น `['ex', '-', 'lovers']` และ `they'll` เป็น `['they', "'", 'll']` และ `(Yeah)` เป็น `['(', 'Yeah', ')']`

ตัวอย่างการใช้

```
import re
text = "Got a long list of ex-lovers, they'll tell you I'm insane (Yeah) "
tokenizing_pattern = r'\w+|[^\w\s]+'
tokens = re.findall(tokenizing_pattern, text)
print(tokens)
```

ผลลัพธ์เป็น

```
['Got', 'a', 'long', 'list', 'of', 'ex', '-', 'lovers', ',', 'they', "'", 'll', 'tell', 'you', 'I', "'
```

เราสังเกตเห็นได้ว่าโทเค็นที่ถูกต้องมีทั้งหมด 17 ตัว และผิด 4 ตัว ได้แก่ *'ll* และ *'m* ซึ่งผลลัพธ์ที่ได้จากการใช้ `re.findall` มีความถูกต้องมากกว่าการใช้ `re.split` แต่ก็ยังไม่ถูกต้อง 100% สำหรับภาษาอังกฤษ และยังมีแพตเทิร์นอื่นอีกไม่สามารถเขียนเรกเอกซ์มาตัดได้ง่าย ๆ เช่น

- ตัวเลขที่เป็นจุดทศนิยม
- 's ที่ใช้แสดงความเป็นเจ้าของ เช่น *John's car*
- อักษรย่อ เช่น *U.S.A.*

การตัดคำโดยอาศัยกฎเกณฑ์และเรกเอกซ์มีข้อดีคือ เป็นวิธีที่สะดวก และรันได้อย่างรวดเร็ว เพราะมีความซับซ้อนน้อย เหมาะสำหรับการวิเคราะห์ข้อมูลเบื้องต้น ที่ไม่ได้จำเป็นต้องสนใจเรื่องคำที่มีขีดคั่น รูปย่อ หรือตัวเลข และสามารถประยุกต์ใช้ได้กับภาษาหลัก ๆ ในโลกได้

หลายหลายภาษา เช่น ภาษาอังกฤษ ภาษาสเปน ภาษาเยอรมัน ภาษารัสเซีย ภาษาอาหรับ ภาษาฮีบรู แต่ข้อเสียของวิธีนี้ คือ ไม่สามารถใช้กับภาษาที่ไม่มีการใช้ช่องว่างในการแบ่งคำ เช่น ภาษาไทย ภาษาจีน ภาษาญี่ปุ่น ภาษาเกาหลี

การตัดคำแบบอิงคลังศัพท์

การตัดคำแบบอิงคลังศัพท์ เป็นวิธีการตัดคำที่มีประสิทธิภาพสูงวิธีหนึ่ง และมีหลักการคล้ายคลึงกับการเขียนเรกเอกซ์เพื่อบ่งบอกว่าโทเค็นควรจะหน้าตาเป็นอย่างไร แต่ว่าเราจะระบุออกมาทั้งหมดเลยว่าอะไรบ้างที่ควรจะเป็นคำ (โทเค็น) ได้ เพราะฉะนั้นเราจะต้องมีลิสต์ของสตริงของภาษาที่เราต้องการจะตัดให้เป็นคำ เราเรียกลิสต์ของสตริงนั้นว่าคลังศัพท์ (lexicon) และใช้อัลกอริทึมในการไล่หาจากซ้ายไปขวาจนเจอคำที่ปรากฏอยู่ในคลังศัพท์ เช่น *ไม่ต้องกลัวช้ำกับใคร* กลายเป็น *ไม่|ต้อง|กลัว|ช้ำ|กับ|ใคร* ได้เพราะเราลองไล่จากซ้ายไปขวาจนเจอคำว่า *ไม่* และไม่มีคำไหนเลยที่ขึ้นต้นด้วย *ไม่ต* จึงตัดคำว่า *ไม่* ออกมาได้ จากนั้นจึงไล่ต่อจนเจอคำว่า *ต้อง* และไม่มีคำไหนในคลังศัพท์เลยที่ขึ้นต้นด้วย *ต้องก* เราจึงตัด *ต้อง* ออกมา และทำเช่นเดียวกันแบบนี้ไปเรื่อย ๆ จนจบสตริง แล้วจะเห็นได้ว่าวิธีนี้ทำให้ทุกโทเค็นเป็นคำที่อยู่ในคลังศัพท์

คลังศัพท์จะได้มาจากพจนานุกรมอิเล็กทรอนิกส์ซึ่งมักมีคำศัพท์ที่พบเห็นบ่อย ๆ และผู้จัดทำพจนานุกรมได้ตรวจสอบเป็นที่เรียบร้อยแล้ว แต่อย่างไรก็ตามเราสามารถพบข้อมูลที่มีคำที่ไม่ได้อยู่ในคลังศัพท์ ไม่ว่าจะเป็นชื่อเฉพาะที่เป็นภาษาไทย และภาษาต่างประเทศ คำที่สะกดไม่เป็นมาตรฐาน (เช่น *อัลไล ทามมาย เส็ด*) คำที่สะกดผิดโดยไม่ได้ตั้งใจ สำหรับกรณีเหล่านี้เราใช้อัลกอริทึม maximal matching ในการรับมือ ซึ่งวิธีการทำงานของอัลกอริทึมและโครงสร้างข้อมูลที่ต้องใช้มีรายละเอียดค่อนข้างซับซ้อน และอยู่นอกเหนือขอบเขตของเนื้อหาของหนังสือเล่มนี้ เรามักจะใช้ไลบรารีที่มีประสิทธิภาพสูงโดยไม่ต้องเขียนโค้ดใช้อัลกอริทึมนี้ด้วยตัวเอง ซึ่งจะสาธิตวิธีการใช้ในบทต่อไป

การตัดคำแบบอิงคลังศัพท์มีข้อดีหลายประการ การตัดคำด้วยวิธีนี้มีความแม่นยำค่อนข้างสูง ประมาณ 70%-80% ขึ้นอยู่กับข้อมูลที่ใช้ในการทดสอบ และรันได้เร็ว [Chormai *et al.*, 2020] นอกจากนั้นแล้วยังสามารถปรับแต่งให้เข้ากับข้อมูลได้ง่ายโดยการเปลี่ยนหรือเพิ่มคำศัพท์เข้าไปในคลังคำศัพท์ที่ใช้ในการตัดคำ เช่น สมมติว่าเราต้องการวิเคราะห์ข้อมูลที่มาจากสื่อสังคมออนไลน์ เราสามารถเพิ่มประสิทธิภาพโดยการเพิ่มคำศัพท์ที่เป็นคำแสลงนี่เป็นที่นิยมในช่วงเวลานั้น ถ้าหากว่าเราต้องการวิเคราะห์ข้อมูลที่เกี่ยวกับสินค้าต่างประเทศ เราสามารถเพิ่มชื่อแบรนด์ ชื่อรุ่นสินค้า เข้าไปในคลังศัพท์ ทำให้เครื่องทราบว่าคำเหล่านี้เป็นคำที่ต้องตรวจจับให้ได้ คลังศัพท์มักจะอยู่ในรูปของไฟล์ที่มีคำอยู่ในนั้น ทำให้ผู้ที่ไม่มีพื้นฐานด้านการเขียนโค้ดก็สามารถปรับแต่งการตัดคำได้ และที่สำคัญที่สุดคือ การตัดคำด้วยวิธีนี้สามารถใช้ได้กับภาษาทุกภาษาที่ไม่มีการใช้ช่องว่างในการแบ่งคำ เราสามารถพบเห็นวิธีนี้ในการวิเคราะห์ข้อมูลภาษา หรือการสร้างแอปพลิเคชันเกี่ยวกับภาษาที่ต้องประมวลผลภาษาไทย ภาษาจีน ภาษาญี่ปุ่น ภาษาเกาหลี ส่วนภาษาอื่น ๆ ที่มีการใช้ตัวลาติน หรือระบบการเขียนที่มีช่องว่างระหว่างคำไม่จำเป็นต้องใช้การตัดคำด้วยวิธีนี้

การตัดคำแบบอิงคลังศัพท์มีข้อเสีย คือ ผลลัพธ์มักจะผิดพลาดเมื่อข้อมูลมีชื่อเฉพาะและคำศัพท์ภาษาต่างประเทศที่ทับศัพท์เป็นภาษาไทย ชื่อเฉพาะมักจะไม่อยู่ในคลังศัพท์ และแทบจะเป็นไปไม่ได้เลยที่จะเพิ่มคำศัพท์เข้าสู่คลังศัพท์ให้เป็นปัจจุบันตลอดเวลา เช่น ชื่อของคนไทยหลายชื่อเป็นชื่อที่ใหม่ตามสมัยนิยม และอาจจะไม่เหมือนใครเลย หรือชื่อวงดนตรี ชื่อแอปพลิเคชัน ชื่อห้างสรรพสินค้า ชื่อสถานที่ ก็มักจะเป็นชื่อใหม่ ๆ ไม่ซ้ำใคร ทำให้ปรับคลังศัพท์ได้ยาก ไม่เหมือนกับศัพท์ทางการแพทย์ หรือศัพท์เทคนิคอื่น ๆ ที่เรามักจะสามารถหาแหล่งความรู้ เช่น หนังสือตำรา ที่รวบรวมคำศัพท์เฉพาะเหล่านี้ได้อยู่แล้ว และไม่เปลี่ยนแปลงเร็วเหมือนชื่อเฉพาะ ส่วนคำศัพท์ภาษาต่างประเทศที่ทับศัพท์เป็นภาษาไทยก็สามารถรับมือได้ยาก เนื่องจากโลกสมัยปัจจุบันมีการเชื่อมต่อกับชาติอื่น ๆ มากมาก ในหนึ่งภาษาอาจจะมีคำศัพท์จากภาษาอื่น ๆ เราไม่สามารถแปลงคลังศัพท์ทุกภาษาในโลกมาทับศัพท์เป็นภาษาไทยได้ครบ ที่สำคัญกว่านั้นคือ การทับศัพท์ตามเกณฑ์ราชบัณฑิตยสภานั้นใช้สำหรับเอกสารทางราชการหรือตำราทางวิชาการเพื่อให้เป็นระบบเดียวกันป้องกันการสื่อสารคลาดเคลื่อน เช่น การทับศัพท์ชื่อสถานที่ในการทำแผนที่ แต่สำหรับประชาชนทั่วไปแล้วสามารถทับศัพท์ได้อย่างอิสระจะไม่ยึดตามเกณฑ์ราชบัณฑิตยสภาก็ย่อมได้ อาทิ

- คำว่า *application* คนทั่วไปอาจจะทับศัพท์ได้หลากหลายแบบ ได้แก่ *แอปพลิเคชัน แอพพลิเคชั่น แอปปลิเคชั่น*
- คำว่า *graphics* คนทั่วไปอาจจะทับศัพท์เป็น *กราฟฟิก กราฟฟิค กราฟิค*
- คำว่า *clinic* คนทั่วไปอาจจะทับศัพท์เป็น *คลินิก คลินิค คลีนิค*

ผู้ที่วิเคราะห์มักจะไม่สามารถไปบอกผู้ให้ข้อมูลให้เขียนให้ถูกต้องตามหลัก ตามเกณฑ์ที่ต้องการ เราต้องวิเคราะห์ข้อมูลภาษาที่ใช้กันจริง ๆ ไม่ใช่ภาษาที่ตรงตามเกณฑ์ราชบัณฑิต ฯ หรือมาตรฐานอื่น ๆ ดังนั้นถ้าเราพบว่าข้อมูลที่เราได้มาอาจจะมีชื่อเฉพาะ คำศัพท์เทคนิค หรือคำทับศัพท์ภาษาต่างประเทศที่ไม่ได้สะกดด้วยเกณฑ์เดียวกัน เราจำเป็นต้องเลือกใช้วิธีการตัดคำแบบอื่น

การตัดคำแบบอิงการเรียนรู้ของเครื่อง

การตัดคำแบบอิงการเรียนรู้ของเครื่อง คือ การตัดคำโดยการเรียนรู้จากข้อมูล ไม่มีการกำหนดแพตเทิร์นหรือคลังศัพท์โดยตรง การตัดคำวิธีนี้อาศัยแบบจำลองหรือโมเดล (model) ที่ถูกฝึกขึ้นมาจากชุดข้อมูลการตัดคำ อัลกอริทึมและเทคนิควิธีในการสร้างแบบจำลองโดยการ

เรียนรู้จากชุดข้อมูล เรียกว่า การเรียนรู้ของเครื่อง (machine learning) ซึ่งเป็นเทคนิคที่สำคัญที่สุดในการสร้างปัญญาประดิษฐ์ (artificial intelligence) แบบจำลองจะทำการเรียนรู้หาแพตเทิร์นของตัวอักษรที่มักจะมาประกอบเป็นคำจากข้อมูลที่มีตัวอย่างการตัดคำที่ถูกต้อง กระบวนการนี้เรียกว่ากระบวนการฝึกแบบจำลอง (training) หลังจากที่โมเดลฝึกเสร็จเรียบร้อยแล้ว เราจึงนำโมเดลที่ได้ไปใช้ตัดคำจากข้อมูลที่ไม่เคยเห็นมาก่อน โมเดลประเภทนี้สามารถตรวจหาคำมักจะพบในคลังศัพท์ อีกทั้งยังสามารถขยายผลไปตรวจจับแพตเทิร์นของคำที่อาจจะไม่ได้ปรากฏมาก่อนในชุดข้อมูล ไม่ว่าจะเป็นชื่อเฉพาะ หรือคำทับศัพท์ เนื่องจากทั้งชื่อเฉพาะ และคำทับศัพท์ ต่างก็มีแพตเทิร์นของมันเองที่ยากต่อการเขียนเรกเอกซ์ออกมาให้ครบถ้วน

ในการใช้งานจริงเรามักจะไม่สร้างโมเดลการตัดคำด้วยตัวเอง เรามักจะใช้ไลบรารีที่มีการควบรวมโมเดลที่ถูกฝึกมาก่อนหน้านี้แล้ว และนำมาใช้โดยไม่มีการปรับแต่ง นับเป็นวิธีที่ตัดคำได้แม่นยำที่สุด โดยมักจะได้ความแม่นยำประมาณ 85% ขึ้นอยู่กับชุดข้อมูลที่ใช้ในการทดสอบ [Chormai *et al.*, 2020, Limkonchotiwat *et al.*, 2020, Limkonchotiwat *et al.*, 2021] ดังนั้นการตัดคำแบบอิงการเรียนรู้ของเครื่องเป็นวิธีที่เหมาะสมที่สุดสำหรับการวิเคราะห์ข้อมูลตัวอักษรภาษาที่ไม่มีการใช้ช่องว่างในการแบ่งคำ

การตัดคำแบบอิงการเรียนรู้ของเครื่องมีข้อเสียเรื่องความเร็ว หากใช้เครื่องคอมพิวเตอร์ทั่วไปมักจะใช้เวลาในการประมวลผลนานกว่าการตัดคำโดยใช้คลังศัพท์ถึง 5-10 เท่า โมเดลใช้เวลาประมาณ 10-20 วินาทีต่อการตัด 1 ล้านคำ [Chormai *et al.*, 2020] ถ้าหากเราต้องการวิเคราะห์ข้อมูลที่เราเก็บมาเสร็จเรียบร้อยแล้วและขนาดไม่ได้ใหญ่เกินไป การตัดคำประเภทนี้ยังคงเป็นวิธีที่เหมาะสมอยู่ เพราะเราสามารถแบ่งข้อมูลไปประมวลด้วยเครื่องคอมพิวเตอร์หลาย ๆ เครื่อง หรือใช้เครื่องที่มีกำลังในการคำนวณสูง ๆ แต่ถ้าหากเราต้องประมวลข้อมูลตามเวลาจริง (real-time) กล่าวคือเมื่อได้รับข้อมูลมาขณะนั้นแล้วต้องประมวลผลให้เสร็จในขณะนั้น การตัดคำด้วยวิธีนี้อาจจะช้าเกินไป นอกจากนั้นแล้ววิธีนี้เป็นวิธีที่ปรับแต่งได้ยาก เพราะต้องเข้าใจวิธีการใช้การเรียนรู้ของเครื่องเพื่อฝึกโมเดลใหม่ หรือฝึกเพิ่มเติมจากโมเดลเดิม อีกทั้งยังต้องใช้ทรัพยากรในการสร้างชุดข้อมูลเพื่อฝึกโมเดลอีกด้วย

สรุปเรื่องการแปลงให้เป็นโทเค็น

การตัดคำทั้งสามวิธีดังกล่าวยังคงเป็นวิธีที่นิยมและมีประสิทธิภาพในการตัดคำ การเลือกใช้ตัวตัดคำต้องคำนึงถึงภาษาของข้อมูล และลักษณะของภาษาในชุดข้อมูล เราสามารถสรุปข้อดีและข้อเสียของตัดคำทั้งสามวิธีได้ดังนี้

วิธีการตัดคำ	ส่วนประกอบที่สำคัญ	ข้อดี	ข้อเสีย
ตัดคำแบบอิงกฎเกณฑ์	กฎเกณฑ์ในรูปเรกเอกซ์	เรียบง่าย และแม่นยำ	ใช้ได้กับเฉพาะภาษาที่ใช้ช่องว่างในการแบ่งคำ
ตัดคำแบบอิงคลังศัพท์	คลังศัพท์	แม่นยำ เร็ว และสามารถปรับแต่งได้	มักเกิดข้อผิดพลาดหากข้อมูลมีคำที่ไม่อยู่ในคลังศัพท์
ตัดคำแบบอิงการเรียนรู้ของเครื่อง	โมเดลที่ถูกฝึกแล้ว	แม่นยำที่สุดสำหรับภาษาที่ไม่ใช้ช่องว่างในการแบ่งคำ	ช้า ปรับแต่งได้ยาก

ไลบรารีที่ใช้ในการตัดคำ

การตัดคำ หรือการแปลงให้เป็นโทเค็นเป็นกระบวนการพื้นฐานที่มักจะต้องทำเป็นอันดับแรก กลุ่มนักพัฒนาโปรแกรมจึงได้สร้างไลบรารีในการประมวลผลภาษาธรรมชาติมาหลายตัวซึ่งมีฟังก์ชันในการตัดคำ ไลบรารีที่สามารถตัดคำได้มีหลายตัว แต่ว่าปัจจุบันยังไม่มีไลบรารีตัวใดเลยที่สามารถตัดคำได้ทุกภาษา เพราะฉะนั้นวิธีการเลือกใช้ไลบรารีในการตัดคำนั้นขึ้นอยู่กับภาษาที่ต้องการตัดคำ เราจะพูดถึงไลบรารี 3 ตัวที่น่าสนใจ ได้แก่ NLTK และ pythainlp

การตัดคำภาษาอังกฤษ และภาษาที่ใช้ช่องว่างในการแบ่งคำ

การตัดคำภาษาอังกฤษมักจะใช้ชุดของเรกเอกซ์ซึ่งที่จริงแล้วสามารถประยุกต์ใช้ในการตัดคำของภาษาอื่น ๆ ที่ใช้ช่องว่างเป็นตัวแบ่งคำเป็นส่วนใหญ่

- ภาษาตระกูลอินโดยูโรเปียนที่ใช้ตัวอักษรละติน เช่น ภาษาฝรั่งเศส ภาษาเยอรมัน ภาษาสเปน
- ภาษาที่ใช้ตัวอักษรซิริลลิก (มักจะเป็นภาษาตระกูลสลาวิก) เช่น ภาษารัสเซีย ภาษาเบลารุส ภาษาเซอร์เบีย

- ภาษาตระกูลเซมิติก เช่น ภาษาอาหรับ ภาษาเปอร์เซีย

ตัดคำด้วยไลบรารี NLTK

ไลบรารี NLTK (Natural Language Toolkit) เป็นหนึ่งในไลบรารีที่ใช้กันอย่างแพร่หลายในการประมวลผลภาษาธรรมชาติด้วยภาษาไพทอน ไลบรารีนี้มีเครื่องมือและโมดูลที่หลากหลายสำหรับการวิเคราะห์และประมวลผลข้อความ นอกเหนือจากโมดูลการตัดคำที่ถูกใช้บ่อยที่สุดแล้ว ยังมีโมดูลที่มีโครงสร้างข้อมูลที่รองรับโครงสร้างต้นไม้ของประโยค โมดูลการตัดประโยค โมดูลการวิเคราะห์โครงสร้างประโยค และ โมดูลในการดาวน์โหลดคลังข้อมูลอีกด้วย [Bird, 2006]

วิธีการตัดคำโดยใช้ nltk ค่อนข้างตรงไปตรงมา โดยการเรียกใช้งานโมดูล `nltk.tokenize` และเรียกใช้ฟังก์ชัน `word_tokenize` โดยใส่ข้อความที่ต้องการตัดคำเข้าไป โดยฟังก์ชันนี้จะคืนค่าเป็นลิสต์ของคำที่ถูกตัดออกมา โดยตัดด้วยชุดของเรกเอกซ์ดังตัวอย่างในโค้ดที่

```
import nltk
from nltk.tokenize import word_tokenize

text = "Hello, how are you?"
tokens = word_tokenize(text, preserve_line=True)
print(tokens)

arabic_text = "مرحباً بكم في عالم البرمجة اللغوية."
arabic_tokens = word_tokenize(arabic_text, preserve_line=True)
print(arabic_tokens)
```

ผลลัพธ์ที่ได้จะเป็นลิสต์ของคำที่ถูกตัดออกมา ดังนี้

```
['Hello', ',', 'how', 'are', 'you', '?']
['اللغوية' ,'البرمجة' ,'عالم' ,'في' ,'بكم' ,'مرحباً', '.']
```

ข้อสังเกตเมื่อเรียก `word_tokenize` คือการตั้ง `preserve_line=True` ซึ่งจะทำให้ฟังก์ชันการตัดคำไม่ทำการตัดประโยคก่อนที่จะเริ่มตัดคำ เพราะโดยปกติแล้วฟังก์ชันนี้จะทำแบ่งให้สตริงเป็นประโยคย่อย ๆ ก่อนเพื่อทำให้การตัดคำแม่นยำขึ้น แต่ว่าการตัดประโยคจำเป็นต้องแบบจำลองที่ต้องดาวน์โหลดเพิ่มเติม ซึ่ง nltk ไม่ได้มีแบบจำลองในการตัดประโยคทุกภาษา และทำให้การตัดคำช้าลงเล็กน้อย วิธีการดาวน์โหลดแบบจำลองในการตัดประโยคให้ใช้คำสั่ง `nltk.download('punkt')` ก่อนการใช้งาน ซึ่งคล้ายกับการติดตั้งโปรแกรม เราดาวน์โหลดลงเครื่องเพียงครั้งเดียวก็สามารถดึงมาใช้ได้ตลอดไป ดังโค้ดต่อไปนี้

```
import nltk
nltk.download('punkt')
text = "Hello, how are you?"
tokens = word_tokenize(text)
print(tokens)
```

ข้อสังเกตคือ เราไม่ต้องตั้งค่า `preserve_line= False` เพราะว่า `False` เป็นค่าโดยปริยายของอาร์กิวเมนต์นี้

ถ้าหากเราต้องการตัดคำภาษาอื่น ๆ โดยมีการตัดประโยคก่อนด้วย เราต้องระบุภาษาที่ต้องการตัดคำด้วย ดังโค้ดต่อไปนี้

```
import nltk
nltk.download('punkt') # หากยังไม่เคยดาวน์โหลดมาก่อน
russian_text = "Здравствуй, мир!"
russian_tokens = word_tokenize(russian_text, language='russian')
print(russian_tokens)
```

ผลลัพธ์ที่ได้จะเป็นลิสต์ของคำที่ถูกตัดออกมา ดังนี้

```
['Здравствуй', ',', 'мир', '!']
```

ทั้งนี้รายละเอียดวิธีการใช้อาจมีการเปลี่ยนแปลงได้เรื่อย ๆ เนื่องจากเป็นไลบรารีแบบโอเพนซอร์ส และมีการพัฒนาอยู่ตลอดเวลา ดังนั้นควรตรวจสอบคู่มือการใช้ หรือเปิดดูโค้ดของไลบรารีเพื่อตรวจสอบวิธีการใช้งานที่ถูกต้องที่สุด เช่น ในเวอร์ชันปัจจุบัน (2024) โค้ดของฟังก์ชันนี้ คือ

```
# มาจาก https://www.nltk.org/_modules/nltk/tokenize.html#word_tokenize
def word_tokenize(text, language="english", preserve_line=False):
    """
    Return a tokenized copy of *text*,
    using NLTK's recommended word tokenizer
    (currently an improved :class:`.TreebankWordTokenizer`
    along with :class:`.PunktSentenceTokenizer`
    for the specified language).

    :param text: text to split into words
    :type text: str
    :param language: the model name in the Punkt corpus
    :type language: str
    :param preserve_line: A flag to decide whether to sentence tokenize the text or not.
    :type preserve_line: bool
    """
    sentences = [text] if preserve_line else sent_tokenize(text, language)
    return [
        token for sent in sentences for token in _treebank_word_tokenizer.tokenize(sent)
    ]
```

จากโค้ดการตัดคำของ เราสังเกตได้ว่าหากเราต้องตัดคำภาษาอังกฤษ เราไม่ต้องระบุอาร์กิวเมนต์อะไรอื่นนอกจาก `text` แต่ถ้าเราต้องการตัดคำภาษาอื่น ๆ เราต้องระบุภาษาที่ต้องการตัดด้วยอาร์กิวเมนต์ `language` และถ้าเราต้องการให้ฟังก์ชันไม่ตัดประโยคก่อน เราต้องตั้งค่า `preserve_line=True` ด้วย

การตัดคำภาษาไทย และภาษาอื่น ๆ ที่ไม่ใช้ช่องว่างในการแบ่งคำ

มีเพียงไม่กี่ภาษาในโลกที่ใช้ระบบการเขียนที่ไม่ใช้ช่องว่างเป็นตัวแบ่งคำ ภาษาเหล่านี้ต้องใช้คลังศัพท์ หรือแบบจำลองในการตัดคำ ที่ต้องใช้ทรัพยากรเพื่อสร้างขึ้นมาเฉพาะเจาะจงกับแต่ละภาษาโดยเฉพาะ เพราะฉะนั้นมีเพียงภาษาไม่กี่ภาษาเท่านั้นที่มีความสำคัญทางเศรษฐกิจพอที่จะได้รับความสนใจจากนักวิจัย ในการพัฒนาเครื่องมือในการตัดคำโดยอัตโนมัติ เช่น

- ภาษาไทย
- ภาษาจีน
- ภาษาญี่ปุ่น
- ภาษาเกาหลี
- ภาษาลาว
- ภาษาพม่า
- ภาษาฮินดี
- ภาษาเวียดนาม

ภาษาทั้งหมดข้างต้นนี้ล้วนแต่ระบบการเขียนเฉพาะของภาษานั้น ยกเว้นภาษาเวียดนามที่ใช้ตัวละติน แต่ว่าใช้ช่องว่างในการแบ่งพยางค์ ไม่ได้ใช้ช่องว่างในการแบ่งคำ

ไลบรารี pythainlp เป็นไลบรารีที่ใช้ในการประมวลผลภาษาธรรมชาติภาษาไทยโดยเฉพาะ
ไลบรารีนี้ถูกพัฒนาโดยกลุ่มนักพัฒนาไทย และมีการพัฒนาอย่างต่อเนื่อง ไลบรารีนี้มีฟังก์ชันในการตัดคำ และประมวลผลภาษาไทยอื่น ๆ มากมาย แต่ฟังก์ชันที่ถูกใช้มากที่สุดคือการตัดคำ เนื่องจากใช้ง่าย และเป็นขั้นตอนที่สำคัญที่สุดขั้นตอนหนึ่งของการวิเคราะห์ข้อมูลภาษาไทย วิธีการใช้งานไลบรารีนี้ในการตัดคำ ดังตัวอย่างโค้ดต่อไปนี้

```
from pythainlp.tokenize import word_tokenize

text = "สวัสดีครับ สบายดีไหมครับ"
tokens = word_tokenize(text)
print(tokens)
```

ผลลัพธ์ที่ได้จะเป็นลิสต์ของคำที่ถูกตัดออกมา ดังนี้

```
['สวัสดี', 'ครับ', ' ', 'สบาย', 'ดี', 'ไหม', 'ครับ']
```

การตัดประโยค

หากข้อมูลที่ได้มามีความยาวมากกว่า 1 ประโยค ในบางครั้งเราจำเป็นต้องแบ่งข้อความ 1 หน่วยให้ออกมาเป็นประโยค ก่อนที่จะนำไปประมวลผลและวิเคราะห์ต่อ ซึ่งกระบวนการตัดประโยคและการตัดคำเป็นสองกระบวนการที่เกี่ยวพันกัน ตัวอย่างเช่น การตัดคำและการตัดประโยคสำหรับภาษาอังกฤษต้องอาศัยการตรวจหาคำย่อ เช่น *U.S.A. Mr. etc.* เพราะจุดสามารถใช้ในภาษาอังกฤษเพื่อแสดงคำย่อเช่นเดียวกับการแสดงจุดสิ้นสุดของประโยค ในกรณีที่จุดแสดงคำย่อ จุดมักจะถูกพิจารณาว่าเป็นส่วนหนึ่งของโทเค็นคำย่อ ในขณะที่จุดที่อยู่ท้ายประโยคมักจะถูกพิจารณาว่าเป็นโทเค็นโดด ๆ การแยกโทเค็นของคำย่อมีความซับซ้อนมากขึ้นเมื่อคำย่อเกิดขึ้นที่ท้ายประโยค และจุดนั้นแสดงทั้งคำย่อและขอบเขตประโยค เช่น

> Mr. Smith will arrive in the U.S. at 4 p.m. Make sure to remind Dr. Rutherford to be on time.

สามารถถูกตัดออกมาเป็นสองประโยคดังนี้

1. *Mr.|Smith|will|arrive|in|the|U.S.|at|4|p.m.*
2. *Make|sure|to|remind|Dr.|Rutherford|to|be|on time|.*

เราสังเกตว่าจุดเป็นส่วนหนึ่งของโทเค็น *Mr. U.S. p.m.* และ *Dr.* เพราะว่าเป็นจุดที่แสดงคำย่อ ประโยคแรกจึงไม่มีโทเค็นที่เป็นจุดเดี่ยว ๆ เนื่องจากอักขรวิธีของภาษาอังกฤษไม่จะเขียนจุดซ้ำ หากคำสุดท้ายลงท้ายด้วยจุดอยู่แล้ว ส่วนประโยคที่ 2 มีจุดที่อยู่ท้ายประโยค แยกออกมาเป็นโทเค็นเดี่ยว ๆ เนื่องจากเป็นจุดที่แสดงจุดสิ้นสุดของประโยค

ประโยคในภาษาเขียนส่วนใหญ่มักคั่นด้วยเครื่องหมายวรรคตอน แต่กฎการใช้เครื่องหมายวรรคตอนไม่ได้ถูกกำหนดไว้อย่างชัดเจนเสมอไป แม้แต่ภาษาที่มีการกำหนดกฎการแบ่งประโยคไว้ชัดเจน ผู้ใช้ภาษาอาจจะไม่ได้ยึดถือกฎเหล่านั้นเสมอไป ขึ้นอยู่กับที่มาของแหล่งข้อความและประเภทของข้อความ หากเป็นข้อความที่มาจากการเขียนพูดคุยกันผ่านโลกออนไลน์อาจจะไม่ได้ยึดถือกฎการใช้เครื่องหมายวรรคตอน และการแบ่งประโยคตามที่อักขรวิธีกำหนด แต่ว่าประกาศราชการและหนังสือมักจะยึดถือกฎการใช้เครื่องหมายวรรคตอนอย่างเคร่งครัดกว่า ไม่เหมือนกับการใช้ช่องว่างในการแบ่งคำของภาษาอังกฤษ ซึ่งคนมักจะยึดถือกันอย่างไม่ค่อยมีข้อยกเว้นเท่าใด

นอกจากนั้นแล้ว ภาษาต่าง ๆ มักจะคั่นประโยคด้วยเครื่องหมายวรรคตอนที่แตกต่างกัน การตัดประโยคที่ถูกต้องสำหรับภาษาหนึ่ง ๆ จึงต้องอาศัยความเข้าใจในการใช้อักขระเครื่องหมายวรรคตอนต่าง ๆ ในภาษานั้น ในภาษาส่วนใหญ่ โจทย์การตัดประโยคมักจะถูกลดทอนกลายเป็นโจทย์การแก้ความกำกวมของอักขระเครื่องหมายวรรคตอน (punctuation disambiguation) กล่าวคือตรวจหาเครื่องหมายวรรคตอนทั้งหมดอาจจะเป็นเครื่องหมายวรรคตอนแบ่งขอบเขตประโยค (sentence boundary punctuation) ได้ ซึ่งโจทย์นี้มีวิธีการแก้แตกต่างกันไป ขึ้นอยู่กับแต่ละภาษา [Palmer, 2000]

การตัดประโยคภาษาอังกฤษ และภาษาอื่น ๆ ที่มีการใช้เครื่องหมายวรรคตอนแบ่งขอบเขตประโยค

การตัดประโยคในภาษาอังกฤษมักจะใช้เครื่องหมายวรรคตอน ซึ่งเป็นเครื่องหมายที่ใช้ในการแบ่งประโยค โดยทั่วไปแล้วเครื่องหมายวรรคตอนประกอบด้วยเครื่องหมายต่าง ๆ ที่ใช้ในการแบ่งประโยค ซึ่งมีเครื่องหมายวรรคตอนที่ใช้บ่อยที่สุด 5 ตัว คือ เครื่องหมายมหัพภาค หรือจุด (.) เครื่องหมายทวิภาค หรือโคลอน (:) เครื่องหมายจุด 3 จุด (...) เครื่องหมายปรัศนีย์หรือคำถาม (?) และเครื่องหมายอัศเจรีย์ หรือเครื่องหมายตกใจ (!) แต่ว่าเครื่องหมายวรรคตอนเหล่านี้ยังมีความกำกวมอยู่ กล่าวคือเครื่องหมายวรรคตอนเหล่านี้ไม่ได้เป็นตัวบ่งบอกขอบเขตเสมอไป จึงจำเป็นต้องใช้แบบจำลองเข้ามาช่วยในการแก้ความกำกวมของเครื่องหมายเหล่านี้ว่าเป็นตัวแบ่งขอบเขตประโยคหรือไม่ แบบจำลองที่นิยมใช้มากที่สุดประเภทหนึ่ง คือแบบจำลองที่เรียนรู้จากคลังข้อมูลขนาดใหญ่โดยใช้การเรียนรู้โดยไม่อาศัยผู้สอน (unsupervised learning) กล่าวคือ แบบจำลองเรียนรู้การตัดประโยคโดยไม่ต้องใช้ชุดข้อมูลที่มีการตัดประโยคด้วยมือเรียบร้อยแล้วเป็นส่วนหนึ่งของการเรียน แบบจำลองเรียนรู้จากข้อมูลที่เป็นข้อความเพียงอย่างเดียว ไม่ใช่ชุดข้อมูลที่การปิดป้ายกำกับเป็นพิเศษ

แบบจำลองนี้เรียกว่าระบบพุงคท์ (Punkt) ซึ่งภาษาเยอรมันแปลว่าจุด [Kiss and Strunk, 2006] ระบบนี้ใช้การคำนวณค่าสถิติของคำที่อยู่รอบ ๆ เครื่องหมายจุด โดยเริ่มจากการตรวจหาคำย่อซึ่งมีจุดอยู่ เพราะจุดที่อยู่ในคำย่อที่มีจุดมักจะไม่ใช่ตัวแบ่งประโยค แต่ว่าก็ไม่แน่นอนเสมอไป ระบบจึงมีการใช้ค่าสถิติที่นำมาเป็นตัวช่วยในการตัดสินใจ (heuristic) เพิ่มอีก 3 ตัว (รายละเอียดและสูตรการคำนวณค่าสถิติเหล่านี้อยู่นอกเหนือขอบเขตของหนังสือเล่มนี้) ได้แก่

1. ตัวช่วยในการตัดสินใจที่คำนวณจากอักขระวิธี (orthographic heuristic) ของคำที่อยู่รอบ ๆ จุด ซึ่งมาจากข้อสังเกตที่สำคัญ คือ หากคำที่ตามหลังจุดขึ้นต้นถูกเขียนด้วยตัวพิมพ์ใหญ่บ่อยกว่าตัวพิมพ์เล็ก คำนั้นมักจะเป็นคำเริ่มต้นประโยค
2. ตัวช่วยในการตัดสินใจที่คำนวณจากการปรากฏร่วมจำเพาะของคำ (collocation heuristic) ซึ่งมาจากข้อสังเกตที่สำคัญ คือ หากคำที่อยู่ข้างหน้าและคำที่อยู่ข้างหลังจุดมีการปรากฏร่วมจำเพาะ (collocation) สูง จุดนั้นมักจะไม่ใช่ตัวแบ่งขอบเขตของประโยค
3. ตัวช่วยในการตัดสินใจที่คำนวณจากการขึ้นต้นประโยคบ่อย ๆ (frequent sentence starter heuristic) ซึ่งมาจากข้อสังเกตที่สำคัญ คือ จุดที่เกิดหลังคำที่ไม่ใช่คำย่อ ชื่อย่อ หรือตัวเลข มักจะเป็นตัวแบ่งขอบเขตประโยค

ระบบพุงคท์เป็นระบบการตัดประโยคที่ความแม่นยำสูงถึง 97% - 99% สำหรับภาษาโปรตุเกส ดัทช์ อังกฤษ เอสโตเนียน ฝรั่งเศส เยอรมัน อิตาเลียน นอร์เวย์ สเปน สวีเดน และตุรกี โดยแต่ละภาษามีระบบพุงคท์ของตัวเอง เนื่องจากต้องคำนวณค่าสถิติต่าง ๆ ข้างต้นจากคลังข้อมูลของภาษานั้น ๆ

การตัดประโยคภาษาไทย และภาษาจีน

ภาษาที่ไม่สามารถใช้วิธีการคำนวณค่าตัวช่วยในการตัดสินใจมักจะเป็นภาษาที่ไม่ได้ใช้เครื่องหมายวรรคตอนในการแบ่งประโยคในความถี่ที่สูง เช่น ภาษาไทย และภาษาจีน ถึงแม้ภาษาไทยมีการใช้เครื่องหมายวรรคตอนอื่นในการแบ่งประโยคอยู่บ้าง เช่น เครื่องหมายอัศเจรีย์ และเครื่องหมายคำถาม แต่ว่ามักไม่ค่อยพบในการแบ่งประโยคภาษาไทยเท่าใดนัก ในส่วนของภาษาจีน ข้อความบางประเภทมีการใช้จุดในการแบ่งประโยคชัดเจน เช่น หนังสือพิมพ์ แต่ว่ามีข้อความจากแหล่งอื่น ๆ อีกมากที่ไม่ใช้จุดในการแบ่งประโยค เช่น สื่อสังคมออนไลน์ [Xue and Yang, 2011] การแบ่งประโยคของภาษาไทย (โดยทั่วไป) และภาษาจีน (เฉพาะบางแหล่ง) ต้องอาศัยบริบทและโครงสร้างย่อย ๆ ของประโยค [Aroonmanakun, 2007] มากกว่าที่จะอาศัยเครื่องหมายวรรคตอน ด้วยเหตุนี้ทั้งสองภาษานี้ต้องใช้ระบบตัดคำที่อาศัยการเรียนรู้แบบมีผู้สอน (supervised learning) ซึ่งการเรียนรู้ด้วยเครื่องแบบใช้ชุดข้อมูลที่มีการตัดประโยคด้วยมือเรียบร้อยแล้วเป็นส่วนหนึ่งของการเรียนรู้การตัดประโยค ด้วยเหตุผลนี้เองการสร้างระบบการตัดประโยคของภาษาเหล่านี้จำเป็นต้องมีการจ้างทีมงานในการกำกับข้อมูล (annotation) เพื่อสร้างคลังข้อมูลภาษา ทีมงานนักกำกับข้อมูล (annotator) จะต้องมีการเตรียมข้อมูลและตัดประโยคด้วยมือ ตามคำนิยามของประโยคของภาษานั้น ๆ เป็นกระบวนการที่ใช้เวลาและทรัพยากรค่อนข้างมาก

เมื่อได้ชุดข้อมูลที่เหมาะสมแล้ว แบบจำลองที่ใช้ในการตัดประโยคแบบใหม่ ๆ มักจะใช้แบบจำลองแบบอิงฟีเจอร์ (feature-based model) แบบจำลองแบบการเรียนรู้เชิงลึก (deep learning) ในการเรียนรู้จากคลังข้อมูลที่มีกำกับข้อมูลเรียบร้อยแล้วทั้งภาษาไทย [Saetia *et al.*, 2019, Sirirattanajakarin *et al.*, 2020] และภาษาจีน [Srinivasan and Dyer, 2021] แบบจำลองที่ได้รับการฝึกฝนเรียบร้อยแล้วมักจะถูกนำมาเผยแพร่ในลักษณะของไลบรารีให้คนทั่วไปสามารถใช้โดยไม่เสียค่าใช้จ่าย และมีการพัฒนาอย่างต่อเนื่องโดยกลุ่มของนักพัฒนาโอเพนซอร์ส

ไลบรารีที่ใช้ในการตัดประโยค

ปัจจุบันมีอยู่ไลบรารีภาษาไพทอนตัวเดียวที่ใช้ในการตัดประโยคภาษาไทย นั่นคือ pythainlp ซึ่งเป็นไลบรารีที่เรานิยมใช้ในการตัดคำด้วย แบบจำลองที่ pythainlp ใช้คือแบบจำลองที่อาศัยฟีเจอร์ชื่อว่า CRFCut ซึ่งถูกพัฒนาขึ้นเพื่อตัดประโยคเพื่อนำสร้างคลังคู่ประโยคสำหรับการพัฒนาระบบการแปลด้วยเครื่อง [Lowphansirikul *et al.*, 2022] ถึงแม้ว่าตอนนี้ได้มีระบบอื่น ๆ ที่ผลดีกว่า CRFCut แล้ว CRFCut มีอัตราความแม่นยำอยู่ประมาณ 0.62 ในขณะที่แบบจำลองแบบการเรียนรู้เชิงลึกมีอัตราความแม่นยำอยู่ที่ประมาณ 0.69 [Yuenyong and Sornlertlamvanich, 2022] ซึ่งความแม่นยำที่สามารถคาดหวังได้จริงนั้นยังขึ้นอยู่กับปัจจัยอื่น ๆ เช่น ทำการประเมินประสิทธิภาพอย่างไร แหล่งข้อมูลที่ใช้ในการฝึกฝน และแหล่งข้อมูลการประเมินความแม่นยำมีความคล้ายคลึงกับแหล่งข้อมูลที่ใช้ในการฝึกฝนเพียงใด ผู้เขียนยังคงเห็นว่าระบบการตัดประโยคภาษาไทยยังไม่แม่นยำพอที่จะนำไปใช้ได้จริง โจทย์นี้เป็นยังคงเป็นโจทย์เปิดที่นักวิจัยยังคำต้องศึกษาและพัฒนาต่อไป

การตัดประโยคภาษาไทยด้วย pythainlp สามารถทำได้โดยใช้ฟังก์ชัน `sent_tokenize` ดังนี้

```
from pythainlp.tokenize import sent_tokenize

text = "ภาษาศาสตร์ คือ การศึกษาเกี่ยวกับภาษาโดยใช้แนวคิด ทฤษฎีและวิธีการวิจัยที่เป็นวิทยาศาสตร์ เพื่อให้เข้าใจธรรมชาติหรือระบบข
```

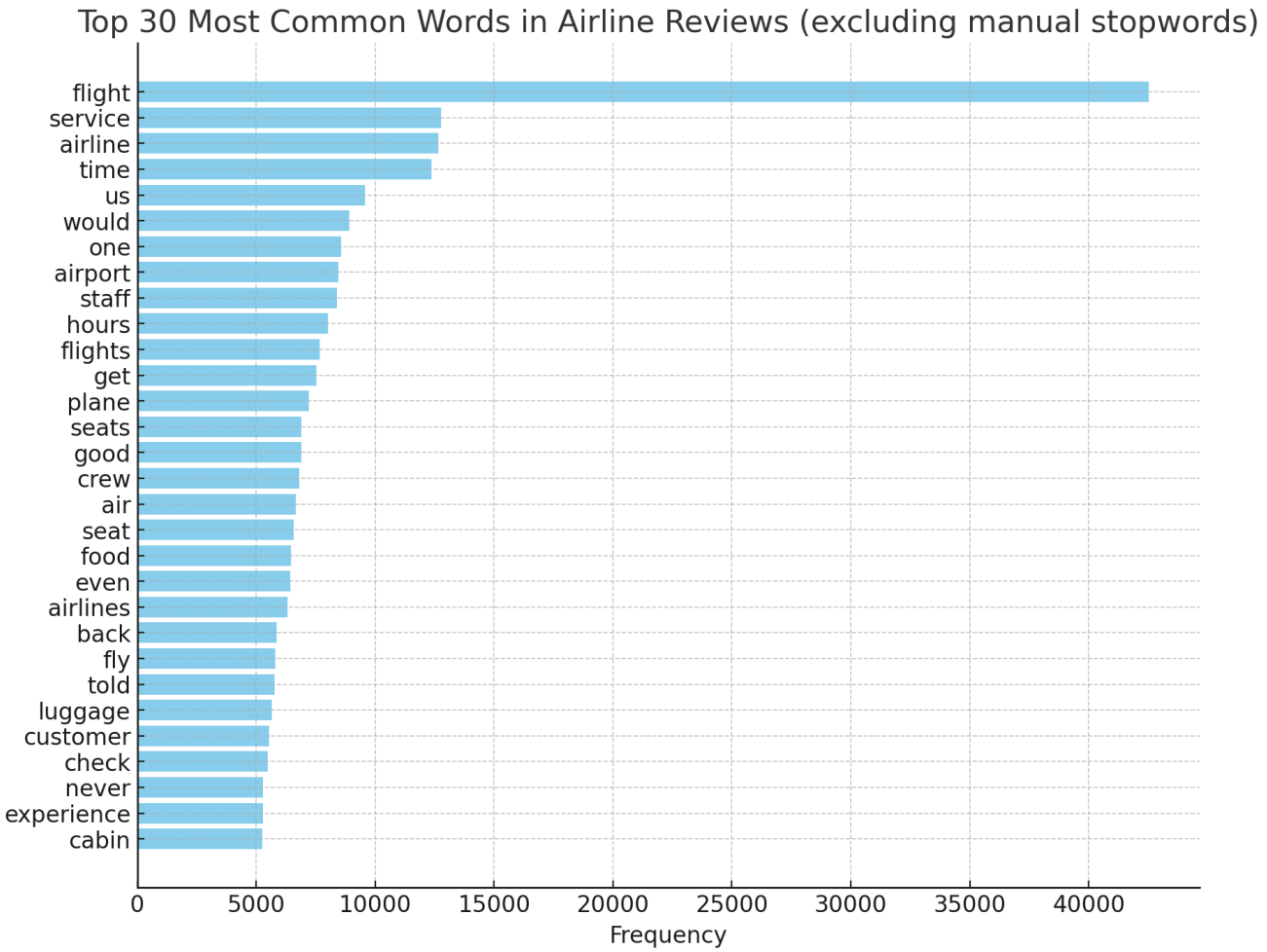
```
sentences = sent_tokenize(text)
print(sentences)
```

ผลลัพธ์ที่ได้จะเป็นลิสต์ของประโยคที่ถูกตัดออกมา ดังนี้

```
['ภาษาศาสตร์ คือ การศึกษาเกี่ยวกับภาษาโดยใช้แนวคิด ทฤษฎีและวิธีการวิจัยที่เป็นวิทยาศาสตร์ เพื่อให้เข้าใจธรรมชาติหรือระบบของภาษา
```

การวิเคราะห์ความถี่ของคำ (word frequency analysis)

เมื่อข้อมูลได้รับการทำความสะอาดและการแบ่งคำอย่างเหมาะสมแล้ว การวิเคราะห์ความถี่ของคำ เป็นการวิเคราะห์ขั้นพื้นฐานที่ช่วยให้เราเข้าใจเนื้อหาของชุดของเอกสารที่มีขนาดใหญ่ได้ ความถี่ของคำในการวิเคราะห์ข้อความ หมายถึง การนับจำนวนครั้งที่คำปรากฏในชุดข้อมูล การวิเคราะห์นี้เป็นพื้นฐานของหลายเทคนิคในการประมวลผลภาษาธรรมชาติ และมีความสำคัญในการเข้าใจลักษณะสำคัญของข้อมูลตัวอักษร ความถี่ของคำช่วยให้เราเห็นภาพรวมของหัวข้อหรือเนื้อหาที่ถูกพูดถึงบ่อยครั้งในชุดของเอกสารที่มีขนาดใหญ่ เพื่อให้เห็นภาพชัดขึ้น ลองพิจารณา ภาพที่ 15



ภาพที่ 15 แผนภูมิแท่งแสดงความถี่ของคำจากชุดข้อมูล 30 อันดับแรก

เราพอจะเดาได้ว่าชุดข้อมูลที่นำมาวิเคราะห์นั้นเกี่ยวกับสายการบิน เพราะพบคำว่า *flight service airline* ด้วยความถี่รวมกว่า 50,000 ครั้ง รวมถึงพบคำอื่น ๆ ที่เกี่ยวกับสายการบินอยู่ใน 30 อันดับแรก เช่น *airport plane seats crew luggage*

การวิเคราะห์ความถี่ของคำจากข้อมูลที่ได้มาจากลูกค้าช่วยให้เราเข้าใจได้ว่าลูกค้าพูดถึงสินค้าหรือบริการของเราด้วยคำใดบ่อยครั้งที่สุด ซึ่งสามารถชี้วัดได้ถึงปัจจัยที่ลูกค้าพอใจหรือไม่พอใจ หรือปัจจัยใดบ้างที่ลูกค้าให้ความสนใจ การวิเคราะห์ข้อมูลรีวิว หรือข้อมูลบนสื่อสังคมออนไลน์ในลักษณะนี้ช่วยให้ธุรกิจสามารถปรับปรุงสินค้าหรือบริการของตนเองให้ตอบสนองความต้องการของลูกค้าได้ดียิ่งขึ้น

ในทำนองเดียวกันเราสามารถใช้การวิเคราะห์ความถี่ของคำในลักษณะนี้เพื่อเปรียบเทียบเนื้อหาของชุดข้อมูลหลาย ๆ ชุดได้อีกด้วย เช่น หากเราต้องการวิเคราะห์เนื้อหา และความแตกต่างของหนังสือพิมพ์จาก 2 สำนักพิมพ์ที่อาจจะมีความเห็นต่างกัน เราสามารถคำนวณและเปรียบเทียบความถี่ของคำที่พบจากข้อมูลหนังสือพิมพ์จาก 2 สำนักพิมพ์นี้

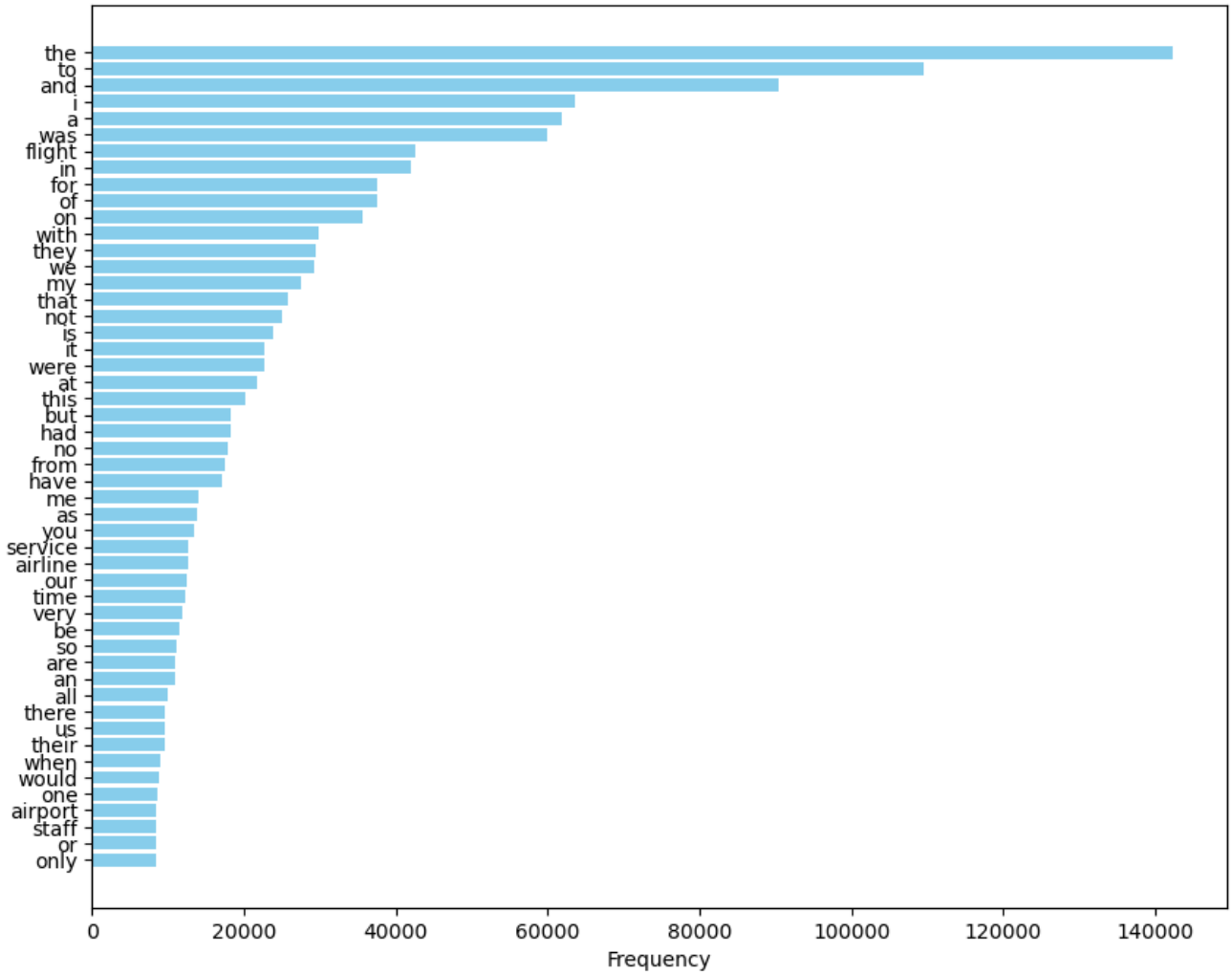
การประยุกต์ใช้การวิเคราะห์ความถี่ของคำที่พบเห็นได้บ่อยที่สุด และเป็นที่นิยมมากขึ้นในขณะนี้ คือ ระบบการฟังเสียงสื่อสังคมออนไลน์ (social listening) ซึ่งเป็นระบบการติดตามและวิเคราะห์ข้อมูลจากสื่อสังคมออนไลน์และอินเตอร์เน็ตเพื่อเข้าใจถึงสิ่งที่กลุ่มเป้าหมายหรือผู้บริโภคกำลังพูดถึงแบรนด์ สินค้า บริการ หรือประเด็นที่เกี่ยวข้อง โดยเฉพาะอย่างยิ่งการตอบสนองต่อแคมเพนการตลาดหรือข่าวสารต่าง ๆ วิธีการนี้ช่วยให้องค์กรสามารถเก็บรวบรวมและวิเคราะห์ข้อมูลจากโซเชียลมีเดียเพื่อเข้าใจและตอบสนองต่อความต้องการและความคาดหวังของผู้บริโภคได้ดียิ่งขึ้น

นอกจากนั้นแล้วการวิเคราะห์ความถี่ยังช่วยเน้นคำที่มีความสำคัญและสามารถเป็นตัวชี้วัดเชิงลึกเกี่ยวกับความรู้สึกและความคิดเห็นของผู้เขียนข้อความ ในหลาย ๆ กรณีการวิเคราะห์ความถี่ของคำเป็นขั้นตอนแรกที่นำไปสู่การวิเคราะห์ที่ซับซ้อนมากขึ้น เช่น การวิเคราะห์อารมณ์และความรู้สึก (sentiment analysis) หรือการสร้างโมเดลทางสถิติเพื่อทำนายพฤติกรรมของผู้ใช้หรือลูกค้า

วิธีการวิเคราะห์ความถี่ของคำ

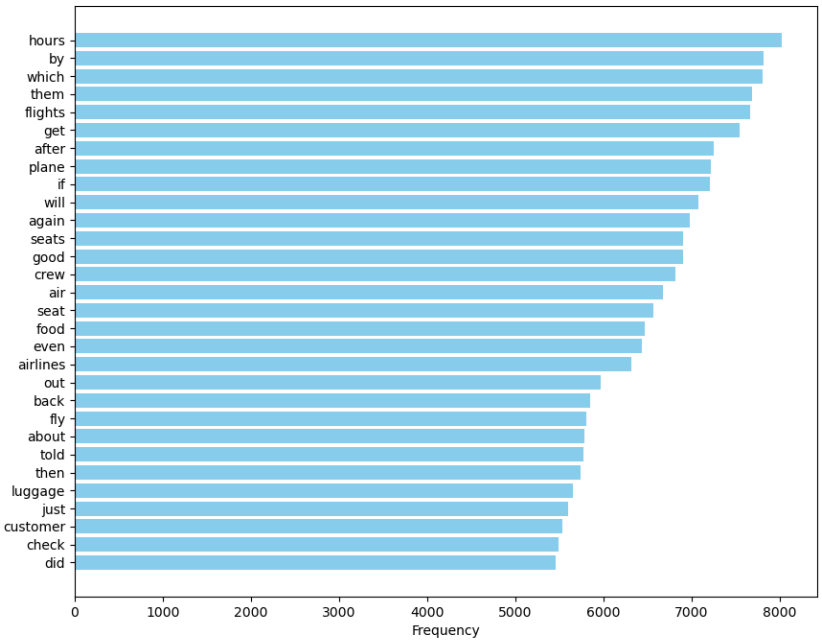
การคำนวณความถี่ของคำเป็นกระบวนการที่ไม่ซับซ้อน แต่ต้องอาศัยความละเอียดในการทำความสะอาดข้อมูลเพื่อให้ได้ผลลัพธ์ที่แม่นยำ ขั้นตอนแรกในการคำนวณความถี่ของคำคือการเตรียมข้อมูลข้อความให้พร้อมสำหรับการวิเคราะห์ การเตรียมข้อมูลอาจรวมถึงการทำความสะอาดข้อมูล เช่น การลบอักขระพิเศษ แฮชแท็ก URL วันที่ หรือคำอธิบายข้อมูล (metadata) อื่น ๆ ตามที่ได้อธิบายไปในบทนี้ เมื่อข้อมูลได้รับการเตรียมพร้อมแล้ว ขั้นตอนถัดไปคือการนับจำนวนครั้งที่แต่ละคำปรากฏในชุดข้อมูล การนับนี้สามารถทำได้โดยการเขียนโปรแกรม เพื่อตัดคำ และนับว่าแต่ละคำปรากฏอยู่ในข้อมูลทั้งหมดกี่ครั้ง ผลลัพธ์จะเป็นรายการของคำพร้อมกับจำนวนครั้งที่พบในข้อมูล ซึ่งเรียกว่า “ความถี่” ของคำนั้น ๆ ความถี่ของคำสามารถนำมาใช้ในการวิเคราะห์เพื่อเห็นแนวโน้มหรือลักษณะเด่นของข้อมูล

เรามักจะพบว่าการวิเคราะห์ความถี่ของคำมักจะไม่ได้ผลออกมาดี ตีความได้ หากเราไม่ทำการกรองคำหยุด (stopword) ออกไปด้วย คำหยุด คือ คำที่มีความถี่สูงแต่ไม่มีความหมายในบริบทของการวิเคราะห์ ตัวอย่างของคำหยุด ได้แก่ *และ ที่ ใน the to for was* เป็นต้น ถ้าหากเราไม่การกรองคำหยุดออกจากชุดข้อมูล ผลการวิเคราะห์จะไม่มีความหมายอะไรเลย ตามที่เห็นในแผนภูมิ ภาพที่ 16 ซึ่งเราไม่สามารถทราบได้เลยว่าชุดข้อมูลที่วิเคราะห์มีเนื้อหาเกี่ยวกับอะไรบ้าง เพราะว่าคำที่ไม่มีนัยสำคัญมักปรากฏอยู่ในอันดับต้น ๆ กลบคำอื่น ๆ ที่มีสาระสำคัญเชิงเนื้อหา ซึ่งทำให้การวิเคราะห์ข้อมูลไม่มีประสิทธิผล

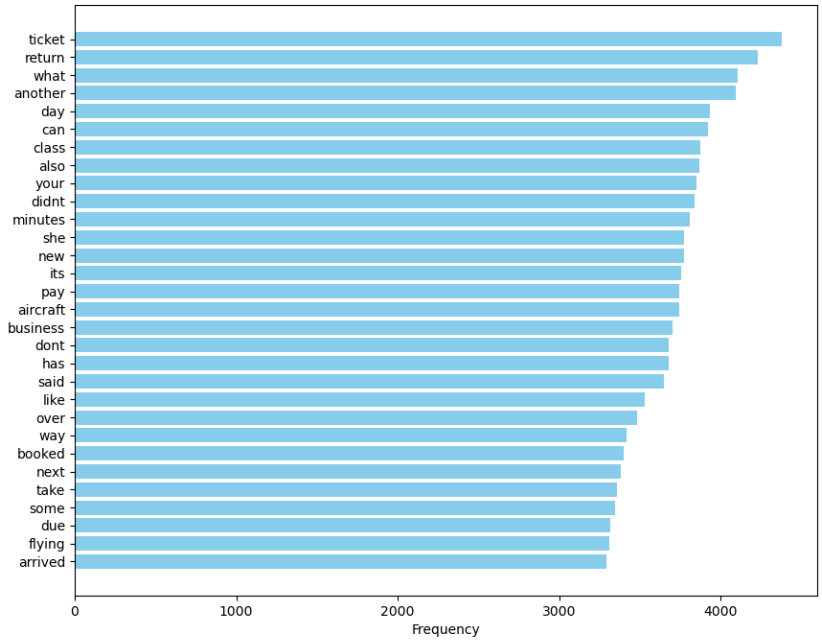


ภาพที่ 16 แผนภูมิแท่งแสดงความถี่ของคำที่อยู่ในชุดข้อมูลรีวิวสายการบิน โดยไม่ได้กรองคำหยุดออก

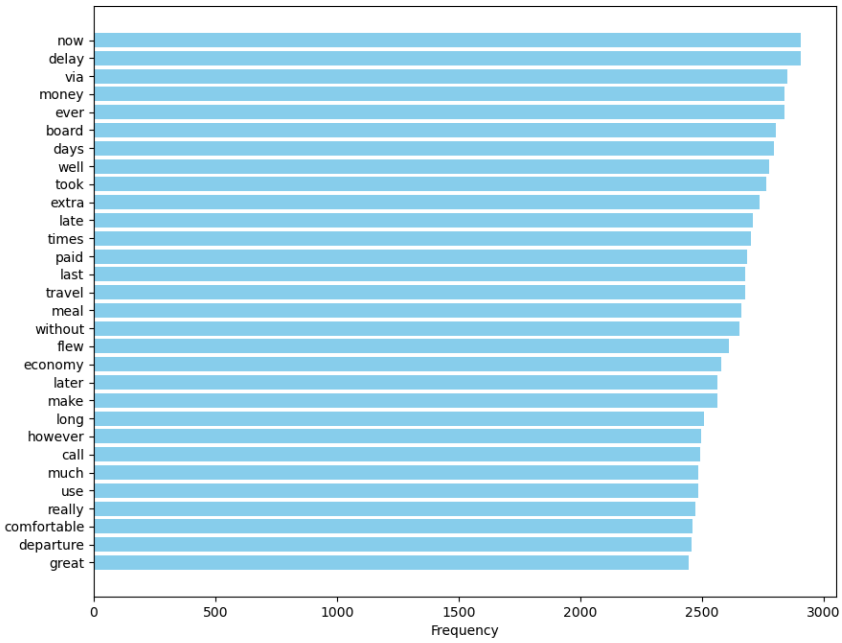
อีกประการหนึ่งที่สังเกตได้จากภาพที่ 16 คือ คำที่ความถี่ลำดับต้น ๆ จะมีความถี่แตกต่างกันมาก ๆ และคำที่ความถี่ลำดับต่อ ๆ มาจะมีความถี่แตกต่างกันไม่มากนัก เป็นลักษณะที่ปรากฏอยู่เสมอในทุกชุดข้อมูลภาษาทุกภาษา การกระจายของคำในลักษณะนี้ เรียกว่าการกระจายตัวตามกฎของซิปฟ์ (Zipf's Law) เพราะฉะนั้นเราสามารถกรองคำหยุดได้ง่าย ๆ ด้วยการกรองเอาคำที่ความถี่ลำดับแรก ๆ ที่ความแตกต่างของความถี่ยังต่างกันอยู่มาก ๆ ออกไป โดยอาจจะลองตัดเอาคำที่ความถี่ลำดับ 1 - 100 ออกไป แล้วทำการวิเคราะห์ความถี่ของคำอีกครั้ง ถ้าหากผลยังออกมาไม่ชัดเจนให้ตัดเอาคำที่ความถี่ลำดับ 1 - 150 ออกไป แล้วทำการวิเคราะห์ความถี่ของคำอีกครั้ง แล้วทำไปเรื่อย ๆ จนกว่าจะได้ผลลัพธ์ที่ดูดี



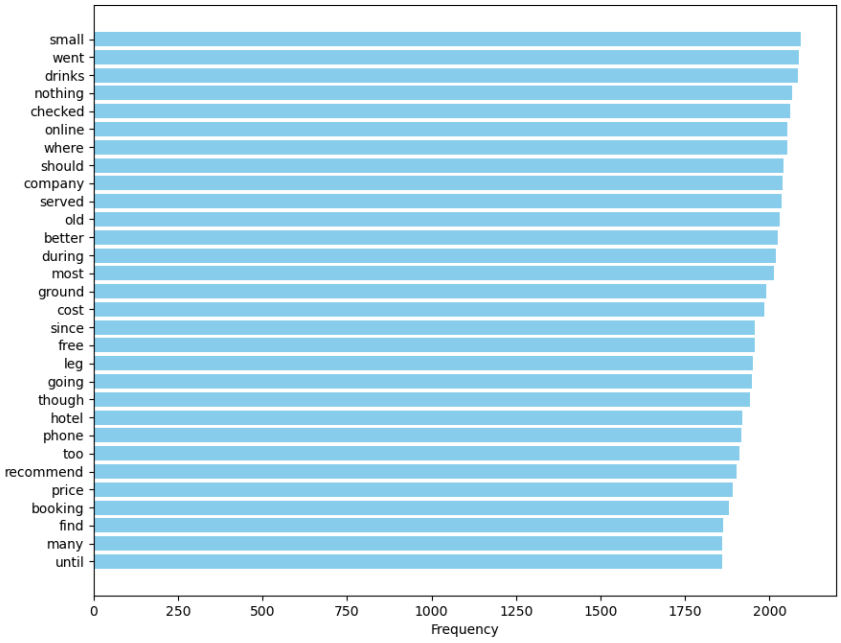
ภาพที่ 17 แผนภูมิแท่งแสดงความถี่ของคำที่อยู่ในชุดข้อมูลรีวิวสายการบิน โดยกรองคำที่มีความถี่สูงสุด 50 อันดับแรกออก



ภาพที่ 18 แผนภูมิแท่งแสดงความถี่ของคำที่อยู่ในชุดข้อมูลรีวิวสายการบิน โดยกรองคำที่มีความถี่สูงสุด 100 อันดับแรกออก



ภาพที่ 19 แผนภูมิแท่งแสดงความถี่ของคำที่อยู่ในชุดข้อมูลรีวิวสายการบิน โดยกรองคำที่มีความถี่สูงสุด 150 อันดับแรกออก



ภาพที่ 20 แผนภูมิแท่งแสดงความถี่ของคำที่อยู่ในชุดข้อมูลรีวิวสายการบิน โดยกรองคำที่มีความถี่สูงสุด 200 อันดับแรกออก

การกำหนดคำหยุดและการใช้คำหยุดจากไลบรารี

นอกจากการกรองคำหยุดด้วยการอาศัยความถี่ของคำแล้ว เรายังสามารถกำหนดรายการคำหยุดเองได้ ซึ่งวิธีนี้มีความยืดหยุ่นและปรับแต่งได้ง่ายกว่า คำหยุดที่กำหนดเองนี้มักจะเป็นคำที่ใช้บ่อยในภาษาแต่ไม่มีความหมายหรือความสำคัญเมื่อทำการวิเคราะห์ข้อความ ซึ่งมักจะเป็นคำในหมวดที่มีหน้าที่หลักในการเชื่อมโยงคำทั้งหมดในประโยคเข้าด้วยกัน เพื่อให้เป็นไปตามหลักไวยากรณ์ หรือสื่อถึงความเชื่อมโยงระหว่างประโยค หรือเชื่อมโยงกับสถานการณ์ที่พูด แต่ว่าไม่เพิ่มความหมายหลักในเชิงเนื้อหา คำในหมวดเหล่านี้ ได้แก่

- กริยาช่วย (auxiliary verb) ยกตัวอย่างเช่น กริยาช่วยอย่าง *am* หรือ *will จะ คง* มักใช้เพื่อช่วยให้กริยาหลักมีความหมายครบถ้วน
- คำบุพบท (preposition) เช่น *ใน* หรือ *เพื่อ* ใช้แสดงความสัมพันธ์ของนามกับส่วนอื่นของประโยค
- คำนำหน้านาม (determiner) เช่น *นี้ นั้น my the those* ใช้ในการชี้เฉพาะนามที่กล่าวถึง
- คำเชื่อม (conjunction) เช่น *และ หรือ อย่างก็ตาม* ใช้เพื่อเชื่อมประโยคหรือคำให้เกิดความสัมพันธ์อย่างใดอย่างหนึ่ง
- คำสรรพนาม (pronoun) เช่น *คุณ he she what อะไร อย่างไร*
- คำอนุภาค (particles) เช่น *สิ นะ ครับ* ใช้เพื่อให้สอดรับกับเจตนาในการพูด และบริบททางสังคมของผู้พูด

อีกตัวเลือกหนึ่งในการกรองคำหยุด คือการใช้คำหยุดจากไลบรารีต่าง ๆ ซึ่งรวมเอาคำหยุดเอาไว้ให้แล้ว เช่น เราสามารถดึงคำหยุดจากไลบรารีของภาษาไทยได้จากไลบรารี pythainlp ดังนี้

```
import pythainlp.corpus
stopwords = set(pythainlp.corpus.thai_stopwords())
```

เราสามารถดึงคำหยุดภาษาอังกฤษจากไลบรารี nltk ได้ดังนี้

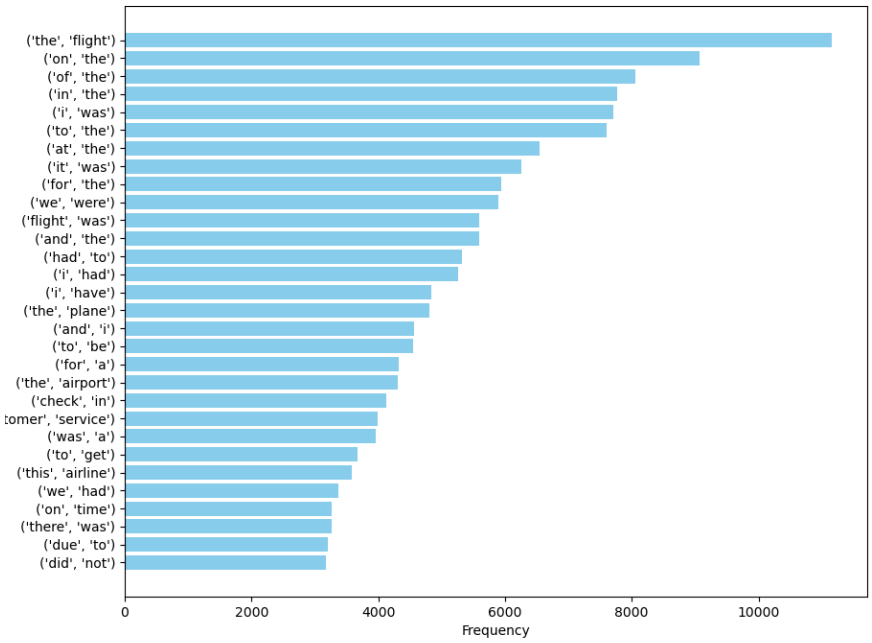
```
from nltk.corpus import stopwords
stopwords = set(stopwords.words('english'))
```

การวิเคราะห์ความถี่ของไบแกรม

การวิเคราะห์ความถี่ของคำไม่จำเป็นต้องจำกัดอยู่เพียงแค่การนับคำเท่านั้น แต่ยังสามารถขยายไปถึงการวิเคราะห์ความสัมพันธ์ระหว่างคำ บางครั้งคำเดี่ยวอาจไม่สามารถสื่อความหมายอย่างชัดเจนหรือครบถ้วน เช่น คำว่า “แห้ง” อาจจะทำให้เราไม่ทราบว่าอะไรแห้ง ดีหรือไม่ดี หรือ คำว่า “ขาย” อาจจะไม่บ่งบอกถึงพฤติกรรมการขายได้ว่าขายอะไร ขายอย่างไร แต่เมื่อพิจารณาคำที่ปรากฏติดต่อกัน เช่น “ผิวแห้ง” หรือ “เทขาย” ความหมายที่สื่อออกมาจะชัดเจนและมีความเฉพาะเจาะจงมากขึ้น การรวมคำสองคำเข้าด้วยกันเพื่อสร้างความหมายที่เฉพาะเจาะจงนี้ เรียกว่า “ไบแกรม” (Bigram) ซึ่งเป็นเทคนิคง่าย ๆ ในการวิเคราะห์คลังข้อมูลจากความถี่ได้ลึกซึ้งมากขึ้น ไบแกรม คือ การรวมคำสองคำที่ปรากฏต่อเนื่องกันในข้อความ โดยไม่คำนึงว่าคำสองคำนั้นจะอยู่ในนามวลี หรือกริยาวลีเดียวกันหรือไม่ ตัวอย่าง เช่น

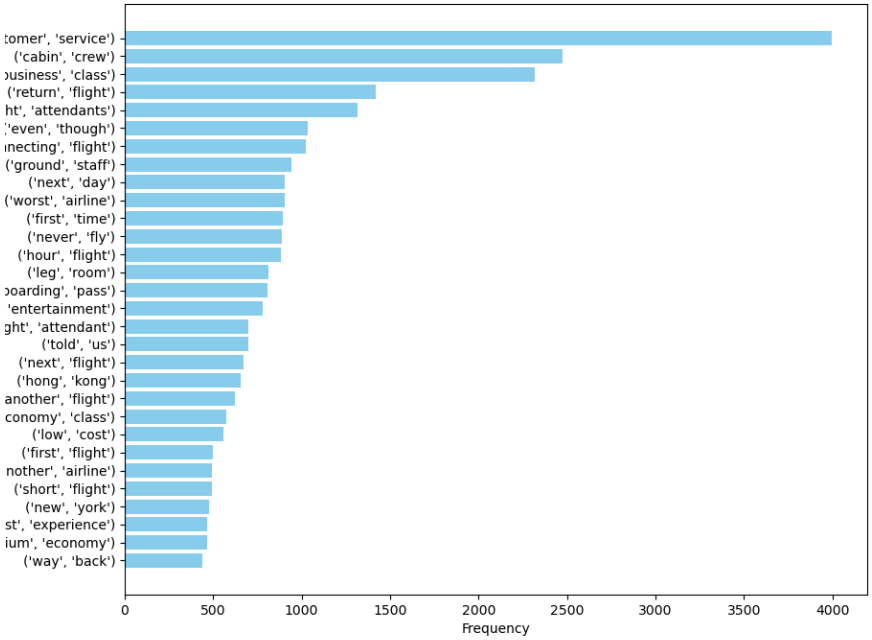
- ประโยค *ครีม|สูตร|นี้|เหมาะ|กับ|คน|ผิว|แห้ง* มีไบแกรมทั้งหมดดังนี้ *ครีมสูตร สูตรนี้ นี้เหมาะ เหมาะกับ กับคน คนผิว ผิวแห้ง*
- ประโยค *ต่าง|ชาติ|เท|ขาย|หุ้น|อย่าง|ต่อเนื่อง” มีไบแกรมทั้งหมดดังนี้ *ต่างชาติ ชาติเท เทขาย ขายหุ้น หุ้นอย่าง อย่างต่อเนื่อง*

หลังจากที่เราได้ดึงไบแกรมออกมาจากข้อความในคลังข้อมูลทั้งหมดแล้ว ขั้นตอนถัดไปคือการนับจำนวนครั้งที่ไบแกรมปรากฏขึ้นในข้อมูลของเรา เพื่อหาไบแกรมที่มีความถี่สูงสุด ในทำนองเดียวกับการวิเคราะห์ความถี่ของคำ



ภาพที่ 21 แผนภูมิแท่งแสดงความถี่ของไบแกรมที่อยู่ในชุดข้อมูลรีวิวสายการบิน

อย่างไรก็ตาม เพื่อให้การวิเคราะห์มีประสิทธิภาพมากขึ้น เราอาจต้องพิจารณากรองเอาไบแกรมที่ประกอบด้วยคำหยุดอย่างน้อย 1 คำออกจากการวิเคราะห์ การกรองคำหยุดออกจากไบแกรมสามารถช่วยให้เก็บไว้เพียงไบแกรมที่มีความหมาย เช่น ไบแกรม *ผิวแห้ง* หรือ *เทขาย* มีประโยชน์กว่า สื่อความหมายได้มากกว่า ไบแกรมที่มีคำหยุด เช่น *กับคน* หรือ *นี้เหมาะ* ด้วยวิธีนี้ เราจึงสามารถลดภาระในการวิเคราะห์และเพิ่มความสามารถในการเข้าใจเนื้อหาของคลังข้อมูล



ภาพที่ 22 แผนภูมิแท่งแสดงความถี่ของไบแกรมที่อยู่ในชุดข้อมูลรีวิวสายการบิน โดยที่กรองเอาไบแกรมที่ประกอบด้วยคำหยุดออก

จากตัวอย่างการวิเคราะห์ความถี่ของไบแกรมใน `numref{airline-review-bigram-no-stopwords}` จะเห็นว่าผู้ที่เขียนรีวิวมักจะพูดถึงการบริการลูกค้า เนื่องจากไบแกรม *customer service* เป็นไบแกรมที่ปรากฏบ่อยที่สุด และผู้ที่เขียนรีวิวพูดถึงตั๋วชั้นธุรกิจบ่อยกว่าตั๋ว

ชั้นประหยัด ถึงแม้ว่าผู้โดยสารชั้นประหยัดมักจะมีมากกว่าผู้โดยสารชั้นธุรกิจ ซึ่งอาจจะตีความได้ว่าประสบการณ์การนั่งชั้นธุรกิจมีประเด็นให้พูดถึงมากกว่า หรือเป็นประสบการณ์ที่ผู้โดยสารตั้งความหวังไว้สูง อีกประเด็นที่ผู้ที่เขียนรีวิวให้ความสำคัญคือ พนักงานผู้ให้บริการ (ไบแกรม *cabin crew flight attendant*, *ground staff*) ซึ่งพบเห็นได้เป็นอันดับแรก ๆ ส่วนเรื่องอื่น ๆ ที่ผู้ที่เขียนรีวิวให้ความสำคัญเป็นอันดับรอง ๆ ลงไป คือเรื่อง *leg room* และ *inflight entertainment*

เมื่อผู้วิเคราะห์ได้ภาพรวมแล้วว่ารีวิวที่ได้รับมาพูดถึงอะไรบ้าง อาจจะเริ่มวิเคราะห์ให้ลึกขึ้นโดยการเลือกเฉพาะรีวิวที่พูดถึงไบแกรมที่เราสนใจ เช่น อาจจะเลือกเฉพาะรีวิวที่พูดถึงไบแกรม *cabin crew* หรือ *flight attendant* หรือ *leg room* เพื่อวิเคราะห์ออกมาเป็นรายประเด็นว่าผู้โดยสารมีความคิดเห็นอย่างไรเกี่ยวกับประเด็นเหล่านี้บ้าง

การสร้างเมฆคำ

การสร้างเมฆคำ (Word Cloud) เป็นเทคนิคที่ใช้ในการแสดงภาพรวมของข้อมูลข้อความ โดยจะแสดงคำที่ปรากฏในข้อมูลข้อความเป็นภาพกราฟิกที่คำที่มีความถี่สูงจะปรากฏให้เห็นด้วยขนาดที่ใหญ่กว่าคำที่มีความถี่น้อย การใช้เมฆคำช่วยให้ผู้วิเคราะห์หรือผู้อ่านสามารถจับตาได้ง่ายว่าคำไหนถูกพูดถึงบ่อยครั้งในชุดข้อมูลข้อความที่กำลังศึกษาอยู่ ซึ่งส่งผลให้สามารถประเมินความสำคัญหรือความนิยมของหัวข้อหรือแนวคิดต่าง ๆ ได้อย่างรวดเร็ว

การสร้างเมฆคำมีส่วนช่วยเสริมการวิเคราะห์ความถี่ของคำในหลาย ๆ ด้าน ข้อดีหลัก ๆ คือ การทำให้ข้อมูลดูน่าดึงดูด สะดุดตา เมื่อนำไปเป็นส่วนหนึ่งของการนำเสนอ รวมถึงทำให้ผู้วิเคราะห์และผู้ชมสามารถมองเห็นคำที่มีความสำคัญหรือถูกพูดถึงบ่อยในชุดข้อมูลได้ง่ายดาย โดยไม่ต้องดูตัวเลขความถี่โดยตรง การแสดงคำที่มีขนาดใหญ่ขึ้นตามความถี่ของคำนั้น ๆ ช่วยให้ผู้ชมเข้าใจได้ว่าคำไหนที่มีความสำคัญหรือเป็นที่สนใจมากในหัวข้อหรือชุดข้อมูลนั้น

ขั้นตอนการทำความสะอาดข้อมูลสำหรับการสร้างเมฆคำ เหมือนกับการวิเคราะห์ความถี่ของคำ เมื่อเราได้คำหรือไบแกรมที่มีความถี่สูงสุดแล้ว เราสามารถใช้ซอฟต์แวร์หรือไลบรารีที่ช่วยสร้างเมฆคำขึ้นมาได้ ซึ่งเราอาจจะปรับแต่งสี ขนาดฟอนต์สูงสุด ต่ำสุด จนกว่าได้เมฆคำที่สวยงามตามความต้องการของเรา ตัวอย่างเมฆคำที่สร้างจากไบแกรมที่มีความถี่สูงสุดในชุดข้อมูลรีวิวสายการบิน แสดงใน ภาพที่ 23

ภาพที่ 23 เมฆคำแสดงไบแกรมที่พบบ่อยอันดับแรก ๆ ในชุดข้อมูลรีวิวสายการบิน

ตัวอย่างโค้ดที่ใช้ในการสร้างเมฆคำภาษาอังกฤษ

ผู้เขียนแนะนำให้ใช้ไลบรารีชื่อว่า `wordcloud` ในการสร้างเมฆคำ ในการสร้างเมฆคำเราจำเป็นต้องกำหนดลักษณะต่าง ๆ ของเมฆคำ เช่น ขนาดของภาพ ขนาดของฟอนต์สูงสุด และสีพื้นหลัง ผู้เขียนแนะนำว่าให้ตั้งค่าสีพื้นหลังเป็นสีขาว และสีตัวอักษรเป็นสีดำ รวมถึงตั้งค่าให้คำทุกคำอยู่ในแนวนอน เพื่อให้คำทุกคำอ่านง่ายเท่ากัน และเน้นการใช้ขนาดของตัวอักษรเป็นตัวบ่งบอกความสำคัญของแต่ละคำ

เท่านั้น ไลบรารีนี้ไม่ได้ทำทุกอย่างให้อย่างสำเร็จรูปจากสตริงดิบ เราต้องเตรียมข้อมูล ทำความสะอาดข้อมูล และวิเคราะห์ความถี่ของคำให้เรียบร้อย จากนั้นจึงป้อนข้อมูลของคำที่ต้องการทำเป็นเมฆคำให้กับไลบรารี

ตัวอย่างโค้ดดังนี้ สมมติว่าเรามี `filtered_biggram_counts` ซึ่งเป็น `Counter` ที่เก็บจำนวนครั้งที่ไบแกรมปรากฏในข้อความทั้งหมด และเราต้องการสร้างเมฆคำจากไบแกรมที่พบบ่อยที่สุด 40 อันดับแรก

```
from wordcloud import WordCloud
wordcloud = WordCloud(
    width = 1600,
    height = 800,
    max_font_size = 200,
    prefer_horizontal = 1,
    background_color = 'white',
    color_func = lambda *args, **kwargs: "black",
    random_state=44
)

filtered_bigram_counts = dict(filtered_bigram_counts.most_common(40))

swc = wordcloud.generate_from_frequencies(filtered_bigram_counts)
plt.figure(figsize=(16, 8))
plt.imshow(swc, interpolation='bilinear')
plt.axis('off')
plt.savefig('../img/airline-review-bigram-wordcloud.png')
plt.show()
```

โค้ดข้างต้นเป็นการสร้างและแสดงผลเมฆคำจากข้อความที่ให้มา โดยใช้ไลบรารี `wordcloud` ประกอบด้วยขั้นตอนต่าง ๆ ดังนี้

1. นำเข้าโมดูล WordCloud จากไลบรารี wordcloud ถ้าหากได้ `ImportError` ให้ลอง `pip install wordcloud` เพื่อลงไลบรารีให้เรียบร้อย
2. สร้างวัตถุ WordCloud พร้อมกำหนดค่าต่าง ๆ เช่น ขนาดของเมฆคำ ขนาดฟอนต์สูงสุด ทิศทางของข้อความ สีพื้นหลัง ฟอนต์ที่ใช้ ฯลฯ ให้สังเกตว่าเราจะตั้งค่า `random_state` ด้วย เนื่องจากการสร้างเมฆคำจะมีการสุ่มตำแหน่งในการวางคำแต่ละคำ ถ้าหากเราไม่ตั้งค่า `random_state` เราจะไม่สามารถรันโค้ดอีกหนึ่งครั้งเพื่อสร้างเมฆคำที่วางคำออกมาแล้วเหมือนเดิมได้ ถ้าหากตำแหน่งการวางคำต่าง ๆ ไม่สวยงามตามที่เราชอบ เราสามารถเปลี่ยนค่า `random_state` เป็นค่าอะไรก็ได้
3. กรองไบแกรมที่ไม่มีคำหยุด เนื่องจากคำหยุดเป็นอุปสรรคในการวิเคราะห์ความถี่ของคำและไบแกรม
4. นับจำนวนครั้งที่แต่ละไบแกรมคู่คำปรากฏ และเลือกเพียง 40 ไบแกรมที่พบบ่อยที่สุด และเก็บใส่ดิกชันนารีที่คีย์คือคำ และแวลูคือจำนวนครั้งที่ปรากฏ โดยสามารถแปลงจากลิสต์ของทูเปิลซึ่งเป็นผลลัพธ์จาก `most_common` ได้
5. วาดเมฆคำจากไบแกรมที่เลือกไว้ พร้อมกับนำมาแสดงผล
6. บันทึกภาพเมฆคำลงไฟล์ เพื่อนำไปใช้ร่วมกับการนำเสนอได้

ข้อจำกัดของการวิเคราะห์ความถี่

การวิเคราะห์ความถี่ของคำและไบแกรมในชุดข้อมูลมีข้อจำกัดที่สำคัญอยู่บางประการ ที่ทำให้การวิเคราะห์ไม่สมบูรณ์ ประการแรก คือ เราไม่ได้พิจารณาการใช้คำในบริบทของทั้งประโยค หรือกลุ่มคำที่ประกอบด้วยคำมากกว่า 2 คำ เช่น *บริการได้ดี* กับ *บริการไม่เต็มใจ* อาจถูกนับเป็นคำว่า *บริการ* เท่านั้น โดยไม่สามารถแยกแยะความหมายบวกหรือลบได้ เว้นแต่จะมีการอ่านและวิเคราะห์เพิ่มเติม

ประการที่สอง คือ ประเด็นที่มีความถี่น้อยอาจถูกมองข้าม การวิเคราะห์ความถี่อาจทำให้ประเด็นที่ไม่ถูกพูดถึงบ่อยครั้งเป็นจำนวนน้อยหรือมีความสำคัญในบริบทที่แตกต่างออกไปไม่ได้รับความสนใจ เนื่องจากคำที่เกี่ยวข้องกับประเด็นเหล่านี้อาจมีความถี่น้อย เช่น ประเด็นเรื่องความล่าช้าของเที่ยวบิน อาจถูกพูดถึงในหลายรูปแบบ เช่น “เครื่องไม่ตรงเวลา”, “ความล่าช้า”, “ดีเลย์”, “สาย” ซึ่งทำให้การวิเคราะห์ความถี่แต่ละคำแยกกันไม่สามารถจับภาพประเด็นนี้ได้อย่างชัดเจน

การแก้ไขข้อจำกัดเหล่านี้ต้องอาศัยการวิเคราะห์ที่ลึกซึ้งยิ่งขึ้น เช่น การวิเคราะห์ความรู้สึก (sentiment analysis) เพื่อระบุความหมายบวกหรือลบของคำ และการรวบรวมคำที่มีความหมายใกล้เคียงกันเข้าด้วยกันเพื่อวิเคราะห์ประเด็น ซึ่งมีความจำเป็นต้องใช้เครื่องมือการประมวลผลภาษาธรรมชาติขั้นสูง ซึ่งนอกเหนือจากขอบเขตของหนังสือเล่มนี้ อย่างไรก็ตามการวิเคราะห์ความถี่ของคำในชุดข้อมูลก็ยังเป็นวิธีการวิเคราะห์คลังข้อมูลที่ถึงแม้จะเรียบง่าย แต่ว่ามีประสิทธิภาพดีในระดับที่ทำให้เราทำความเข้าใจเนื้อหาของชุดข้อมูลในภาพรวมได้ และยังเป็นวิธีเบื้องต้นที่ผู้วิเคราะห์ข้อมูลนิยมใช้เป็นด่านแรกในการวิเคราะห์ข้อมูลภาษา

สรุป

การประมวลผลภาษาธรรมชาติ มีประโยชน์อย่างมากในการวิเคราะห์ข้อมูลภาษา เพื่อให้เข้าใจความหมายและสาระสำคัญของข้อความต่าง ๆ ในการวิเคราะห์ข้อมูลภาษาเหล่านี้ เรามักใช้ไลบรารีภาษาไพทอน ซึ่งมีเครื่องมือต่าง ๆ ที่สนับสนุนการทำงานด้านการประมวลผลภาษาธรรมชาติได้เป็นอย่างดี ขั้นตอนแรกของการวิเคราะห์คือการทำความสะอาดข้อมูล เพื่อลบส่วนที่ไม่จำเป็นออกจากข้อมูลดิบ และคำนึงถึงแหล่งที่มาของข้อมูลเพื่อให้การวิเคราะห์มีความเชื่อถือได้ การตัดคำในภาษาต่าง ๆ จำเป็นต้องใช้เทคนิคที่เหมาะสมกับภาษานั้น ๆ โดยเทคนิคเหล่านี้อาจแตกต่างกันไปในแต่ละภาษา นอกจากนี้ การวิเคราะห์ความถี่ของคำและการใช้เมฆคำเป็นวิธีที่ง่ายและสะดวกสำหรับการเริ่มต้นทำความเข้าใจข้อมูลภาษาขนาดใหญ่ วิธีการเหล่านี้ช่วยให้เราสามารถมองเห็นภาพรวมและทิศทางของข้อมูลได้เป็นอย่างดี

อ้างอิง

[1] (1,2) Wirote Aroonmanakun. Thoughts on word and sentence segmentation in thai. In *Proceedings of the Seventh Symposium on Natural language Processing, Pattaya, Thailand, December 13–15*, 85–90. 2007.

[2] Steven Bird. Nltk: the natural language toolkit. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, 69–72. 2006.

[3] (1,2,3) Pattarawat Chormai, Ponrawee Prasertsom, Jin Cheevaprawatdomrong, and Attapol Rutherford. Syllable-based neural Thai word segmentation. In Donia Scott, Nuria Bel, and Chengqing Zong, editors, *Proceedings of the 28th International Conference on Computational Linguistics*, 4619–4637. Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics. URL: https://aclanthology.org/2020.coling-main.407, doi:10.18653/v1/2020.coling-main.407.

[4] Tibor Kiss and Jan Strunk. Unsupervised multilingual sentence boundary detection. *Computational Linguistics*, 32(4):485–525, 2006. URL: https://aclanthology.org/J06-4003, doi:10.1162/coli.2006.32.4.485.

[5] Peerat Limkonchotiwat, Wannaphong Phatthiyaphaibun, Raheem Sarwar, Ekapol Chuangsuwanich, and Sarana Nutanong. Domain adaptation of Thai word segmentation models using stacked ensemble. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 3841–3847. Online, November 2020. Association for Computational Linguistics. URL: https://aclanthology.org/2020.emnlp-main.315, doi:10.18653/v1/2020.emnlp-main.315.

[6] Peerat Limkonchotiwat, Wannaphong Phatthiyaphaibun, Raheem Sarwar, Ekapol Chuangsuwanich, and Sarana Nutanong. Handling cross- and out-of-domain samples in Thai word segmentation. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, 1003–1016. Online, August 2021. Association for Computational Linguistics. URL: https://aclanthology.org/2021.findings-acl.86, doi:10.18653/v1/2021.findings-acl.86.

[7] Lalita Lowphansirikul, Charin Polpanumas, Attapol T Rutherford, and Sarana Nutanong. A large english–thai parallel corpus from the web and machine-generated text. *Language Resources and Evaluation*, 56(2):477–499, 2022.

[8] David D Palmer. Tokenisation and sentence segmentation. *Handbook of natural language processing*, pages 11–35, 2000.

[9] Chanatip Saetia, Ekapol Chuangsuwanich, Tawunrat Chalothorn, and Peerapon Vateekul. Semi-supervised thai sentence segmentation using local and distant word representations. *arXiv preprint arXiv:1908.01294*, 2019.

[10] Sorratat Sirirattanajakarin, Duangjai Jitkongchuen, and Peerasak Intarapaiboon. Boydcut: bidirectional lstm-cnn model for thai sentence segmenter. In *2020 1st International Conference on Big Data Analytics and Practices (IBDAP)*, 1–4. IEEE, 2020.

[11] Srivatsan Srinivasan and Chris Dyer. Better chinese sentence segmentation with reinforcement learning. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, 293–302. 2021.

[12] Nianwen Xue and Yaqin Yang. Chinese sentence segmentation as comma classification. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 631–635. 2011.

[13] Sumeth Yuenyong and Virach Sornlertlamvanich. Transentcut-transformer based thai sentence segmentation. *Songklanakarin Journal of Science and Technology*, 44(3):852–860, 2022.