CHULA ENGINEERING
Foundation toward Innovation
COMPUTER

Dictionary-based Tokenization

2110572: NLP SYS

Assoc. Prof. Peerapon Vateekul, Ph.D.

Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University

peerapon.v@chula.ac.th

Credits to: TA.Pluem, TA.Knight, and all TA alumni

Based on Aj.Ekapol's slide in 2017

Outlines

The need for segmentation

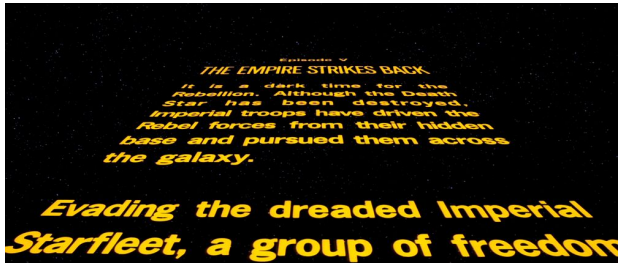
Thai Tokenization

1) Longest Matching

2) Maximal Matching

The need for segmentation

- Text as a stream of characters



- We need a way to understand the meaning of text
  - Break into words (assign meaning to word) ← Word Tokenization
  - Break into sentences (put word meanings back to sentence meaning)

Tokenization - Thai

- Thai has no space between words
- Thai has no clear sentence boundaries

- เริ่มจากชนวนของสงครามแรกสุด สมาพันธ์การค้า ทำการปิดล้อม-รุกรานดาวนาบู ส่งผลต่อมาขยายเป็นสงครามโคลนอันมีฝ่ายแบ่งแยกดินแดนเป็นผู้ชักใยสงคราม หมายโค่นล้มฝ่ายสาธารณรัฐ ขนานไปกับเรื่องราวพัฒนาการของของเด็กน้อยคนหนึ่งผู้มีพลังสถิตแรงมาก ชาวดาวทะเลทรายทาทูอินนาม "อนาคิน สกายวอล์คเกอร์" ผู้ถูกคาดการณ์ว่าคือผู้ถูกเลือกในตำนานของเจได หลังจากสงครามยุทธการดาวนาบู อนาคิน ก็ได้รับฝึกฝนในวิถีเจได โดย อ.เจได "โอบีวัน" แต่ เมื่ออนาคินโตเป็นหนุ่มก็ดันแหกกฎเจไดโดยแอบมีความสัมพันธ์ชู้สาวลับๆกับ ราชินี "อมิดาลา" แห่งดาวนาบู จนเธอตั้งครรภ์ลูกแฝด ... และอนาคินก็ค่อยๆถูกกลืนเข้าสู่ด้านมืดของพลังจนกลายเป็นซิธลอร์ด ได้ฉายา "ดาร์ธ เวเดอร์" ภายใต้การโน้มน้าวชี้นำของ ซิธลอร์ดลึกลับนาม "ดาร์ธ ซีเดียส" ซึ่งเผยในตอนท้ายว่า ซีเดียส ไม่ใช่ใครที่ไหน แต่คือท่าน "พัลพาทีน" สมุหนายกผู้นำสูงสุดของฝ่ายสาธารณรัฐเสียเอง และแท้จริงก็ยังเป็นผู้นำลับชักใยฝ่ายแบ่งแยกดินแดนด้วยอีกต่างหาก ... สงครามจบลงที่ฝ่ายสาธารณรัฐพ่ายแพ้ล่มสลาย อมิดาลาก็ตายหลังคลอดลูกแฝด ทั้งเหล่าอัศวินเจไดก็ถูกฆ่ากวาดล้างสิ้นแบบไม่ทันตั้งตัว เหลือแต่ อ.โอบีวัน กับ อ.โยดา ต้องลี้ภัยหลบหนีซ่อนตัว โดยลูกแฝดของอนาคินได้ถูกส่งไปสู่ที่หลบซ่อนลับเช่นกัน ... และแล้ว พัลพาทีน ก็กินรวบทั้งกระดาน เปลี่ยนการปกครองจาก ระบอบสาธารณรัฐเดิมไปเป็น ระบอบเผด็จการจักรวรรดิซิธแทน ตั้งตนเป็นจักรพรรดิปกครองแกแลคซีทั้งปวง โดยมี ดาร์ธ เวเดอร์ เป็นขุนพลซิธลอร์ดเคียงข้างนับแต่นั้นมา

http://www.starwars.siligon.com/swcong.html

Tokenization - Thai

Social media text

#สตอรี่ของโม 😊🐙 #Days23ofMobile 🌈 ...23 วันแล้วนะเจ้าโม พี่กำลังคิดถึงหนูอยู่เลย แหนะ เมื่อวานหายเลยนะ 🙈 พี่ก็ว่าหายไปไหนแอบหนีไปเล่นลองบอร์ดนี่เอง เล่นระวังๆนะพี่เป็นห่วง เดวล้มแถมไม่มีตะหลามคอยเล่นเป็นเพื่อนอีก 😆 พี่จะบอกว่า "หนูอย่าลืมลงชื่อเลือกตั้งนะ" เดี๋ยวพวกพี่ขำกันไม่ออกนะ 5555 ฝากไว้กันลืม ยิ่งเด๋อๆอยู่ อีกอย่างๆหนูต้องกลับมานะพวกเรารอหนูอยู่ 😊 พี่นี่เตรียมรอโหวตให้หนูเต็มที่เล้ย 😋 ไปเรียนเป็นไงบ้างยังเหงาอยู่มั้ย แต่พี่ว่าคงไม่แล้วหละมั้ง วันนี้ขึ้นสเตจอีกแล้วสินะ เหนื่อยมั้ยคะ แต่คงสนุกมากกว่าอยู่แล้วเนอะ 😁 แค่ได้เห็นรอยยิ้มของหนูพี่ก็สบายใจและ แต่เอ๊ะ เมื่อวานใครบ่นอยากกลับบ้านอีกแล้วนะ อดดู the toy เลยอ่ะดิ้ 😆 หว่ายๆๆๆ น่าสงสาร 555 โอกาสหน้ายังมีนะ ตั้งใจทำงานนะคะ เจ้าหลามแฟนหนูก็อด ไม่ได้อดคนเดียวซะหน่อยนะ 555 🙊 ดูมีความสุขจังน้าเจ้าตัวเล็ก 😊💗 คิดถึงนะ เดี๋ยวก็ได้ 2shot กันแล้ว พี่โคตรตื่นเต้นเลยตื่นเต้นกว่าไปจับมือเยอะ 🙈 คิดท่าไม่ออกเล้ย มีท่าไหนแนะนำมั้ย ช่วยพี่หน่อยยยยย 😅... #Mobilebnk48 #ตู้เพลงโมบิล #ชาวเหรียญหยอดตู้ #MOTA09

Tokenization - Thai

- Many word boundaries depends on the context (meaning)

  ตา กลม vs ตาก ลม

- Even amongst Thais the definition of word boundary is unclear
  - Needs a consensus when designing a corpus
  - Sometimes depends on the application
    - Linguist vs machine learning concerns

  คณะกรรมการตรวจสอบถนน

Dictionary-based vs Machine-learning-based

- 1) Dictionary-based
  - Longest matching
  - Maximal matching
- 2) Machine-learning-based

Dictionary-based word segmentation

- Perform by scanning a string and matching each substring against words from a **dictionary**. (No dataset needed, just prepare a dictionary!)
- However, there is ambiguity in matching. (There are many many ways to match.)

ป้ายกลับรถ

- So, **matching methods** are developed:
  - 1. Longest matching
  - 2. Maximal matching

Meknavin, Surapant, Paisarn Charoenpornsawat, and Boonserm Kijsirikul. "Feature-based Thai word segmentation." Proceedings of Natural Language Processing Pacific Rim Symposium. Vol. 97. 1997.

1) Longest Matching

- Scan a sentence from left to right
- Keep finding a word from the starting point **until no word matches (longest)**, then move to the next point
- Backtrack if current segmentation leads to an un-segmentable chunk

- ป้ายกลับรถ — Start scanning with “ป” as the starting point
- ป้ายกลับรถ — Keep scanning …
- ป้าย/กลับรถ — No more words start with “ป้าย”, move to the next point
- …
- ป้าย/กลับ/รถ

2) Maximal Matching

- Generate all possible segmentations
- Select the segmentations with **the fewest words**

Haruechaiyasak, Choochart, Sarawoot Kongyoung, and Matthew Dailey. "A comparative study on thai word segmentation approaches." Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, 2008. ECTI-CON 2008. 5th International Conference on. Vol. 1. IEEE, 2008.

What if?

- What if there are more than one segmentation with the fewest words?
- Other heuristics are applied, for example
  - Language model score

> Longest Matching คุณ | อากร | กช

> Language Modeling คุณ | อา | กรกช

Maximal matching

- Maximal matching can be done using dynamic programming.
- Let d(i,j) be the function that returns the number of the fewest words possible, with the last word starting with i^{th} character (row) and ending with j^{th} character (column). It can be defined as:

$$d(i,j) = \begin{cases} 1 & \text{if i=1 and c[1, j] is in the dictionary.} \\ 1 + min_{k=1\ldots i-1} d(k, i-1) & \text{if c[i..j] is in the dictionary.} \\ \infty, & \text{otherwise.} \end{cases}$$

when c[i..j] is a string of words in the sentence (assume it is started at index 1) and the base case is d(1,1) = 1.

Maximal matching: Initialization (1st row)

- Maximal matching can be done using dynamic programming.
- Let d(i,j) be the function that returns the number of the fewest words possible, with the last word starting with i^{th} character (row) and ending with j^{th} character (column). It can be defined as:

1st row and it is a word

$$d(i,j) = \begin{cases} 1 & \text{if i=1 and c[1, j] is in the dictionary.} \\ 1 + min_{k=1...i-1} d(k, i-1) & \text{if c[i..j] is in the dictionary.} \\ \infty, & \text{otherwise.} \end{cases}$$

when c[i..j] is a string of words in the sentence (assume it is started at index 1) and the base case is d(1,1) = 1.

Maximal matching: Find a word in dictionary

- Maximal matching can be done using dynamic programming.
- Let d(i,j) be the function that returns the number of the fewest words possible, with the last word starting with i^{th} character (row) and ending with j^{th} character (column). It can be defined as:

$$d(i,j) = \begin{cases} 1 & \text{if i=1 and c[1, j] is in the dictionary.} \\ 1 + min_{k=1...i-1} d(k, i-1) & \text{if c[i..j] is in the dictionary.} \\ \infty, & \text{otherwise.} \end{cases}$$

If last word is a word

when c[i..j] is a string of words in the sentence (assume it is started at index 1) and the base case is d(1,1) = 1.

Maximal matching: Check all possible segmentations before the final word

- Maximal matching can be done using dynamic programming.
- Let d(i,j) be the function that returns the number of the fewest words possible, with the last word starting with i^{th} character (row) and ending with j^{th} character (column). It can be defined as:

$$d(i,j) = \begin{cases} 1 & \text{if i=1 and c[1, j] is in the dictionary.} \\ 1 + min_{k=1...i-1} d(k, i-1) & \text{if c[i..j] is in the dictionary.} \\ \infty, & \text{otherwise} \end{cases}$$

Check all possible segmentations before the final word
Check the whole column in the previous row

when c[i..j] is a string of words in the sentence (assume it is started at index 1) and the base case is d(1,1) = 1.

Maximal matching: The final word

- Maximal matching can be done using dynamic programming.
- Let d(i,j) be the function that returns the number of the fewest words possible, with the last word starting with i^{th} character (row) and ending with j^{th} character (column). It can be defined as:

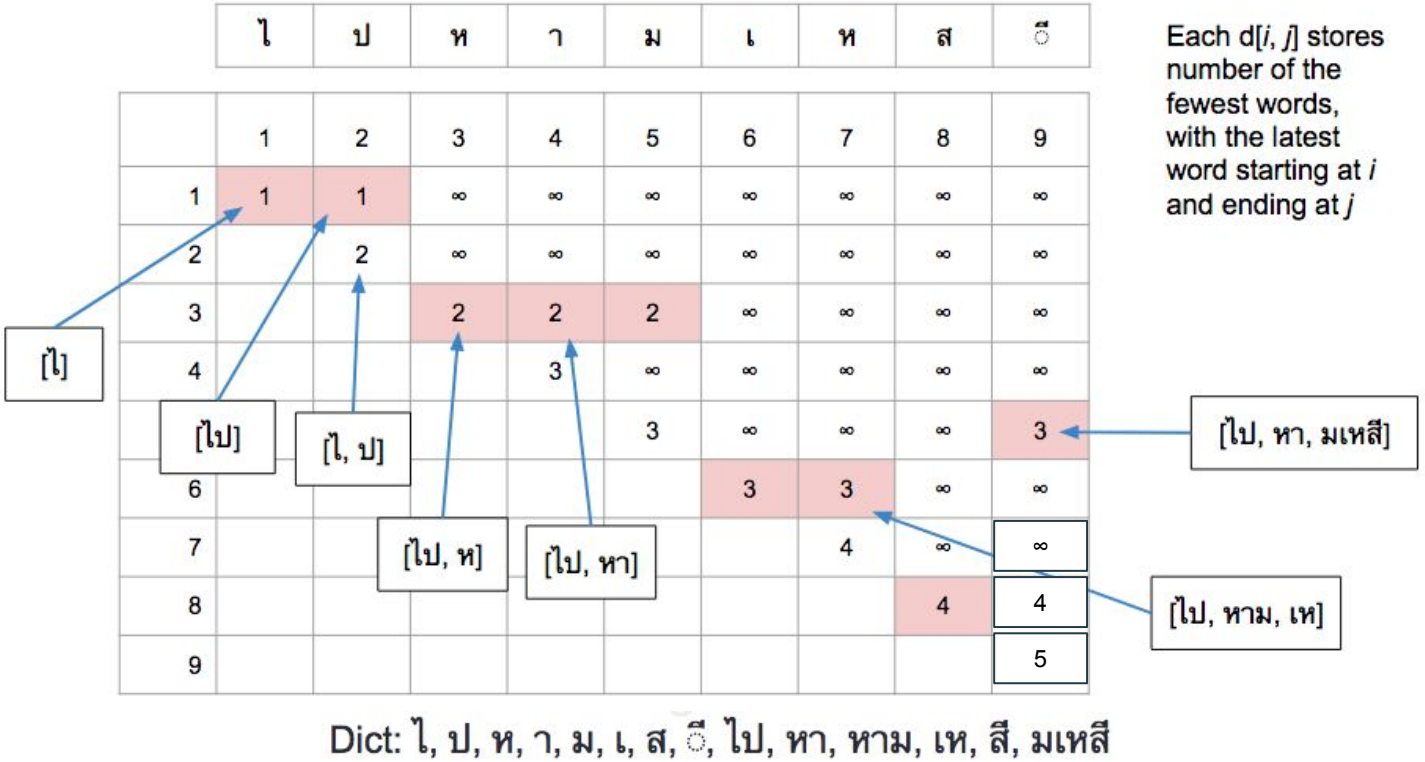
$$d(i,j) = \begin{cases} 1 & \text{if i=1 and c[1, j] is in the dictionary.} \\ 1 + min_{k=1\ldots i-1} d(k, i-1) & \text{if c[i..j] is in the dictionary.} \\ \infty, & \text{otherwise.} \end{cases}$$

The final word

when c[i..j] is a string of words in the sentence (assume it is started at index 1) and the base case is d(1,1) = 1.

Example (1)

$$d(i,j) = \begin{cases} 1 & \text{if i=1 and c[1, j] is in the dictionary.} \\ 1 + min_{k=1\ldots i-1} d(k, i-1) & \text{if c[i..j] is in the dictionary.} \\ \infty, & \text{otherwise.} \end{cases}$$



Example (2)

$$d(i,j) = \begin{cases} 1 & \text{if i=1 and c[1, j] is in the dictionary.} \\ 1 + min_{k=1\ldots i-1} d(k, i-1) & \text{if c[i..j] is in the dictionary.} \\ \infty, & \text{otherwise.} \end{cases}$$

d[1,3] [ไปห] not in dictionary

	ไ	ป	ห	า	ม	เ	ห	ส	ี
	1	2	3	4	5	6	7	8	9
1	1	1	∞	∞	∞	∞	∞	∞	∞
2		2	∞	∞	∞	∞	∞	∞	∞
3			2	2	2	∞	∞	∞	∞
4				3	∞	∞	∞	∞	∞
					3	∞	∞	∞	3
6						3	3	∞	∞
7							4	∞	∞
8								4	4
9									5

Each d[*i*, *j*] stores number of the fewest words, with the latest word starting at *i* and ending at *j*

[ไ]

[ไป]

[ไ, ป]

[ไป, ห]

[ไป, หา]

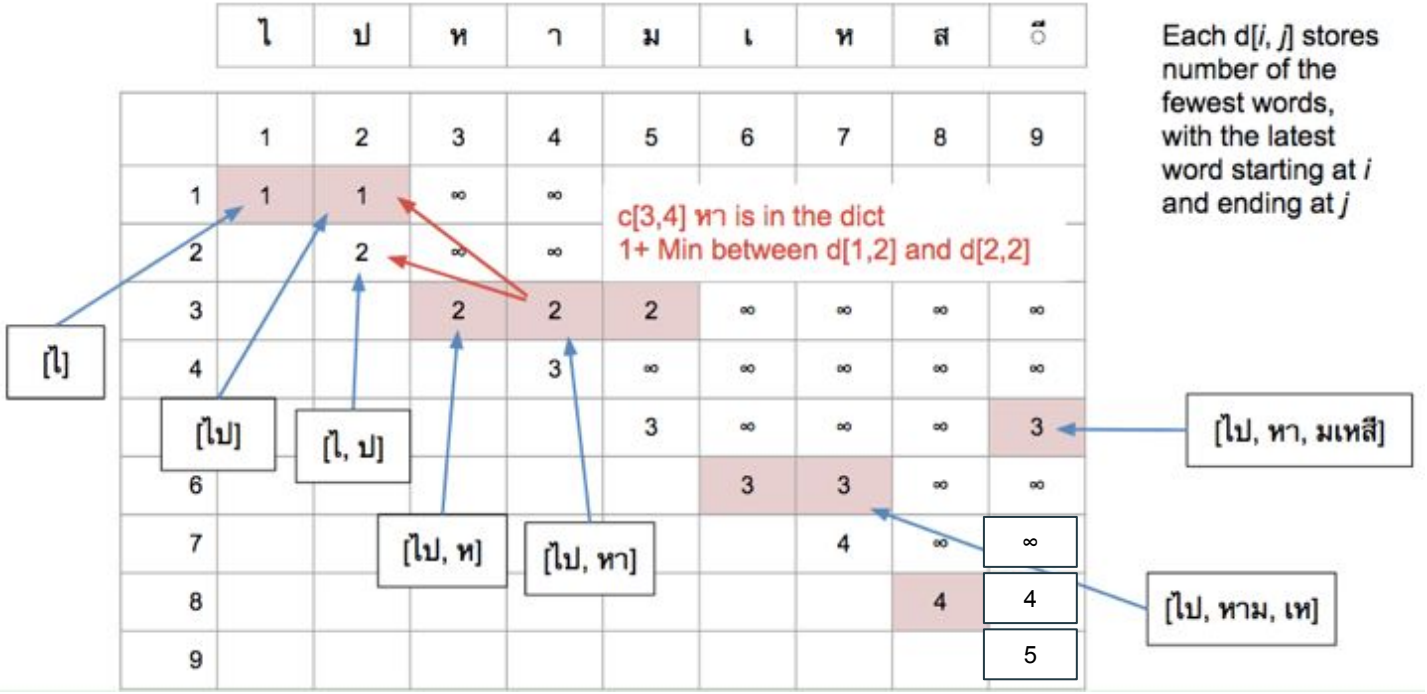
[ไป, หา, มเหสี]

[ไป, หาม, เห]

Dict: ไ, ป, ห, า, ม, เ, ส, ี, ไป, หา, หาม, เห, สี, มเหสี

Example (3)

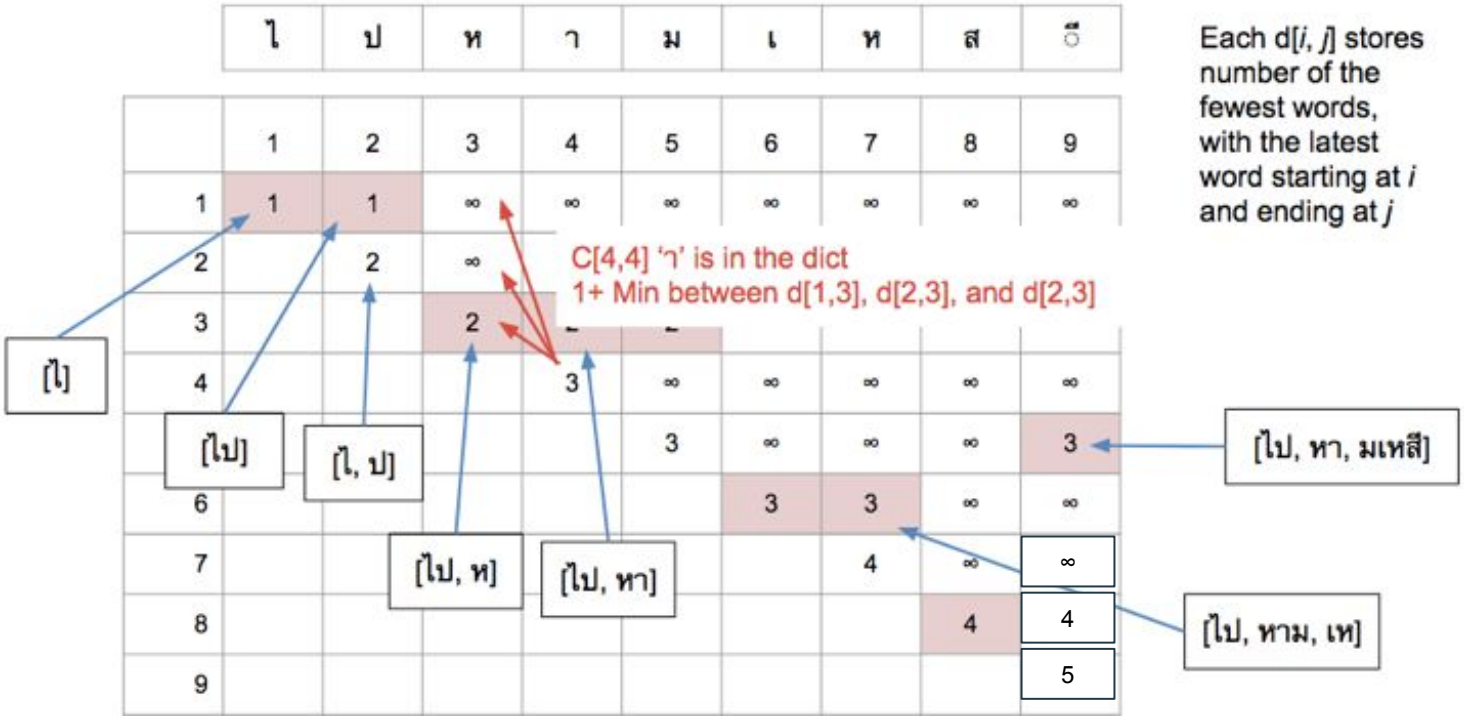
$$d(i,j) = \begin{cases} 1 & \text{if i=1 and c[1, j] is in the dictionary.} \\ 1 + min_{k=1...i-1} d(k, i-1) & \text{if c[i..j] is in the dictionary.} \\ \infty, & \text{otherwise.} \end{cases}$$



Dict: ไ, ป, ห, า, ม, เ, ส, ี, ไป, หา, หาม, เห, สี, มเหสี

Example (4)

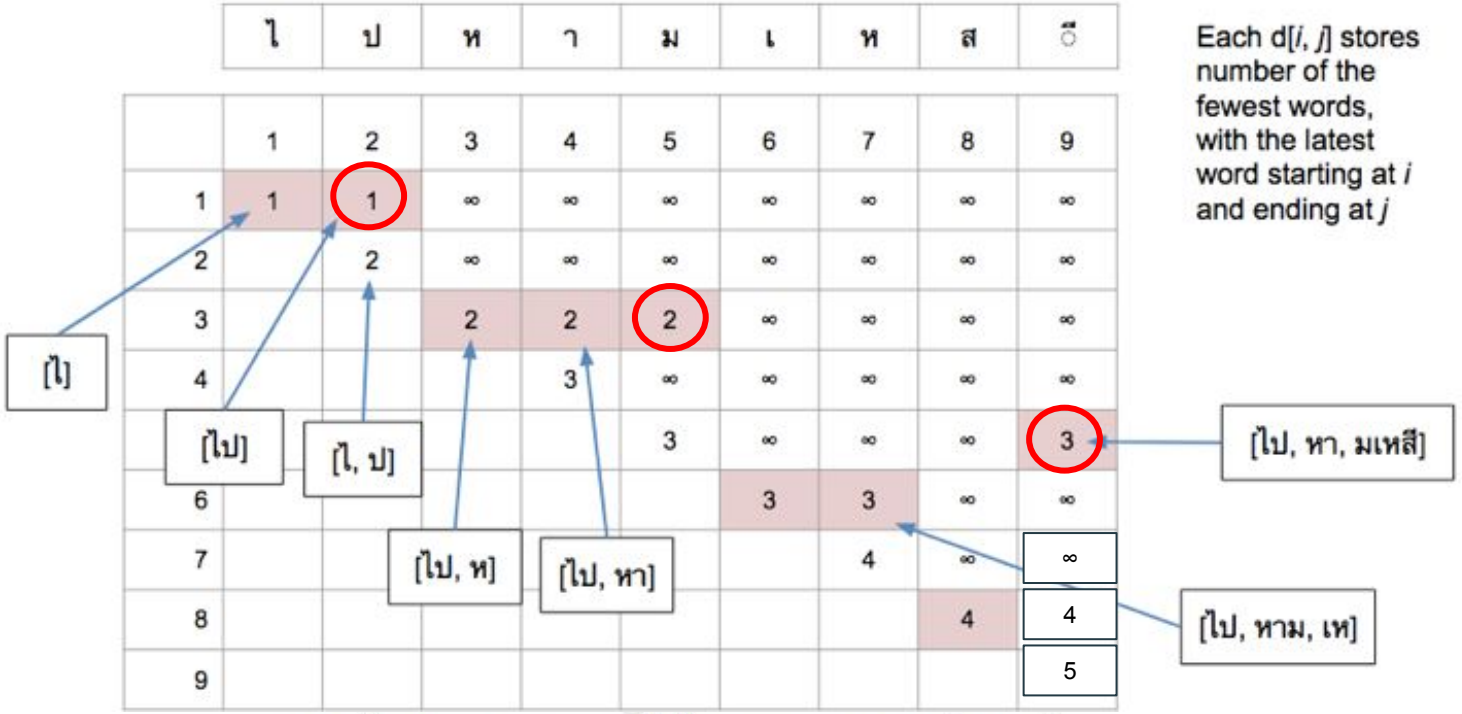
$$d(i,j) = \begin{cases} 1 & \text{if i=1 and c[1, j] is in the dictionary.} \\ 1 + min_{k=1...i-1} d(k, i-1) & \text{if c[i..j] is in the dictionary.} \\ \infty, & \text{otherwise.} \end{cases}$$



Dict: ไ, ป, ห, า, ม, เ, ส, ี, ไป, หา, หาม, เห, สี, มเหสี

Example (5): Backtracking

$$d(i,j) = \begin{cases} 1 & \text{if i=1 and c[1, j] is in the dictionary.} \\ 1 + min_{k=1\ldots i-1} d(k, i-1) & \text{if c[i..j] is in the dictionary.} \\ \infty, & \text{otherwise.} \end{cases}$$



Dict: ไ, ป, ห, า, ม, เ, ส, ี, ไป, หา, หาม, เห, สี, มเหสี

pythainlp.tokenize.word_tokenize(*text: str, custom_dict: ~pythainlp.util.trie.Trie = <pythainlp.util.trie.Trie object>, engine: str = 'newmm', keep_whitespace: bool = True, join_broken_num: bool = True*) → List[str] [source]

Word tokenizer.

Tokenizes running text into words (list of strings).

Parameters:
- **text** (*str*) – text to be tokenized
- **engine** (*str*) – name of the tokenizer to be used
- **custom_dict** (*pythainlp.util.Trie*) – dictionary trie (some engine may not support)
- **keep_whitespace** (*bool*) – True to keep whitespace, a common mark for end of phrase in Thai. Otherwise, whitespace is omitted.
- **join_broken_num** (*bool*) – True to rejoin formatted numeric that could be wrongly separated. Otherwise, formatted numeric could be wrongly separated.

Returns: list of words

Return type: List[str]

Options for engine

- *attacut* - wrapper for AttaCut., learning-based approach
- *deepcut* - wrapper for DeepCut, learning-based approach
- *icu* - wrapper for a word tokenizer in PyICU., from ICU (International Components for Unicode), dictionary-based
- *longest* - dictionary-based, longest matching
- *mm* - "multi-cut", dictionary-based, maximum matching
- *nercut* - dictionary-based, maximal matching, constrained by Thai Character Cluster (TCC) boundaries, combining tokens that are parts of the same named-entity
- *newmm* (default) - "new multi-cut", dictionary-based, maximum matching, constrained by Thai Character Cluster (TCC) boundaries with improved TCC rules that are used in newmm.
- *newmm-safe* - newmm, with a mechanism to avoid long processing time for text with continuously ambiguous breaking points
- *nlpo3* - wrapper for a word tokenizer in nlpO3., adaptation of newmm in Rust (2.5x faster)
- *oskut* - wrapper for OSKut., Out-of-domain StacKed cut for Word Segmentation
- *sefr_cut* - wrapper for SEFR CUT., Stacked Ensemble Filter and Refine for Word Segmentation
- *tltk* - wrapper for TLTK.,

  maximum collocation approach

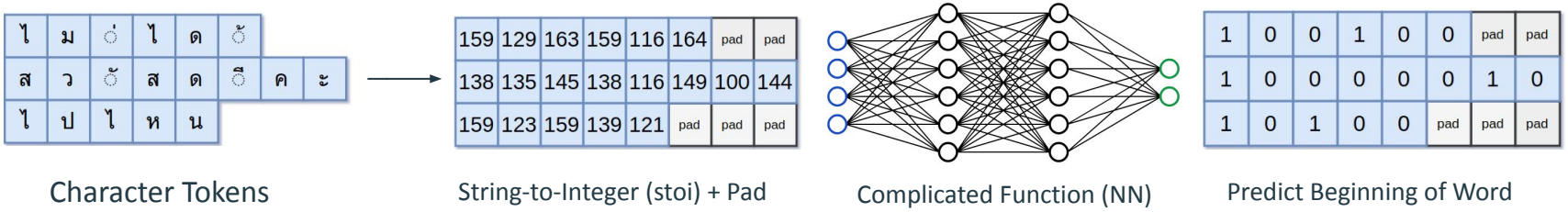
Note:
- The **custom_dict** parameter only works for *deepcut*, *longest*, *newmm*, and *newmm-safe* engines.

Example:

https://pythainlp.org/docs/5.0/api/tokenize.html

How to improve the tokenizer? NN Predict 1/0 for each character

Use neural nets to tokenize?



Character Tokens

String-to-Integer (stoi) + Pad

Complicated Function (NN)

Predict Beginning of Word

Reference: http://web.archive.org/web/20181005101442/https://sertiscorp.com/thai-word-segmentation-with-bi-directional_rnn/

NN-based Tokenizers Performance

		Others			Ours	
Last Updated: 29/08/2019		**PyThaiNLP newmm**	**Sertis Bi-GRU**	**DeepCut**	**AttaCut-C**	**AttaCut-SC**
BEST Validation Set						
Character-Level	precision	0.94±0.11	0.95±0.10	0.99±0.05	0.97±0.07	0.98±0.05
	recall	0.83±0.09	0.99±0.02	0.99±0.03	0.98±0.04	0.99±0.03
	f1	0.88±0.08	0.97±0.07	**0.99±0.04**	0.98±0.05	0.99±0.04
Word-Level	precision	0.73±0.16	0.91±0.14	0.97±0.07	0.94±0.10	0.96±0.08
	recall	0.65±0.16	0.94±0.10	0.97±0.07	0.94±0.09	0.97±0.08
	f1	0.68±0.15	0.93±0.12	**0.97±0.07**	0.94±0.10	**0.97±0.08**
BEST Test Set						
Character-Level	precision	0.91±0.15	0.92±0.11	0.96±0.08	0.94±0.10	0.95±0.09
	recall	0.85±0.09	0.98±0.04	0.98±0.04	0.98±0.04	0.98±0.04
	f1	0.86±0.11	0.95±0.08	**0.97±0.06**	0.96±0.07	0.96±0.07
Word-Level	precision	0.70±0.19	0.85±0.18	0.92±0.14	0.88±0.17	0.91±0.15
	recall	0.64±0.18	0.90±0.14	0.93±0.12	0.91±0.14	0.92±0.13
	f1	0.67±0.19	0.87±0.16	**0.93±0.13**	0.89±0.16	0.91±0.14

https://github.com/PyThaiNLP/attacut