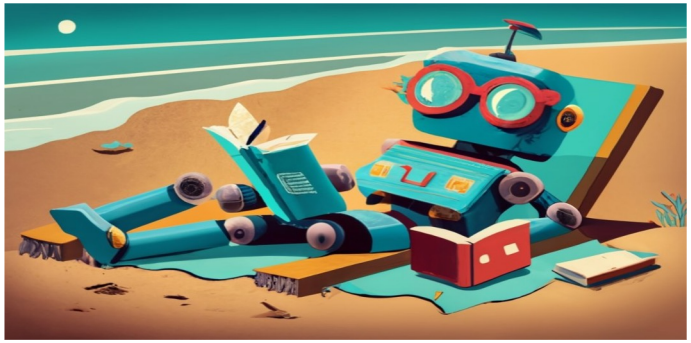
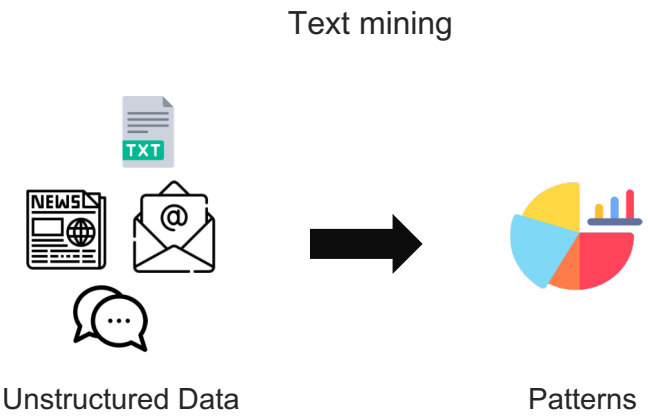
Unit 1.

Natural Language Processing

Machines cannot read text
they can only proccess
numbers

Text mining

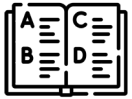




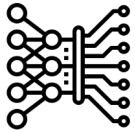
Unstructured Data

Patterns

Modeling

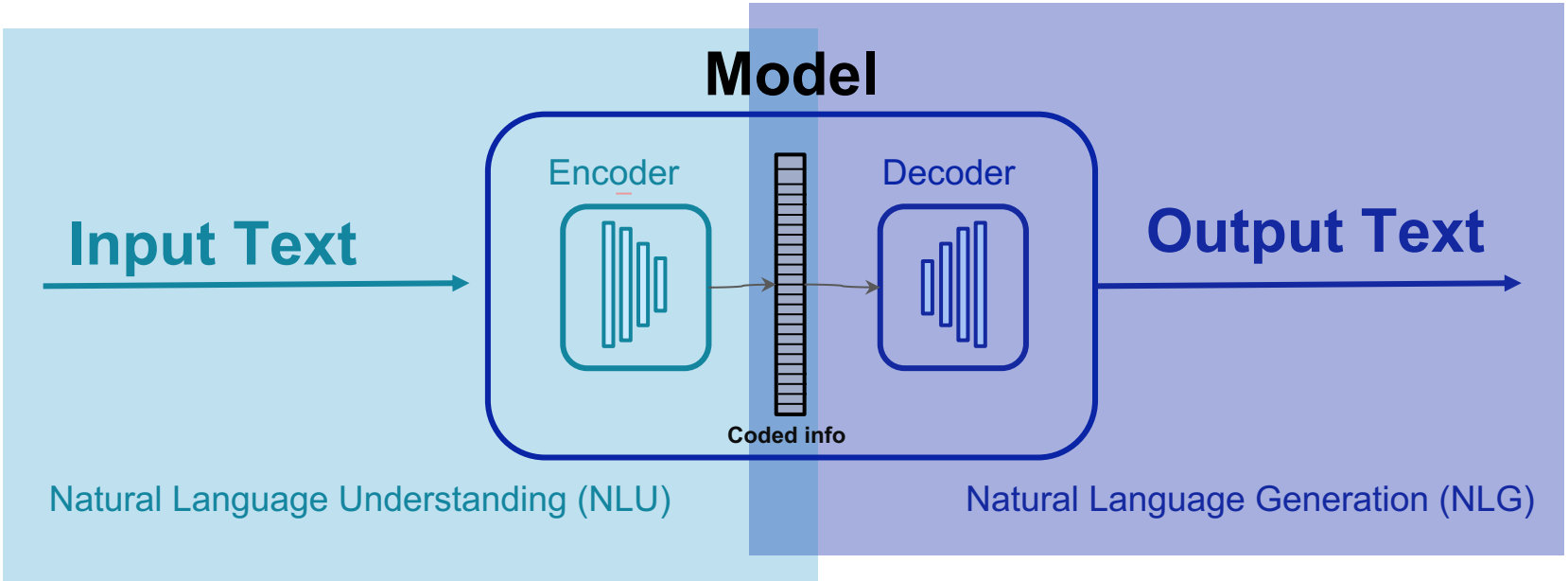


Representative models

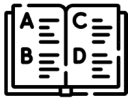


Generative models

A narrow meaning of Natural Language Processing is the processing mechanism used by programs using natural language as input and output.



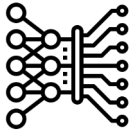
NLP Use cases



Representative models

Retrievals

Text Classification



Generative models

Text translation

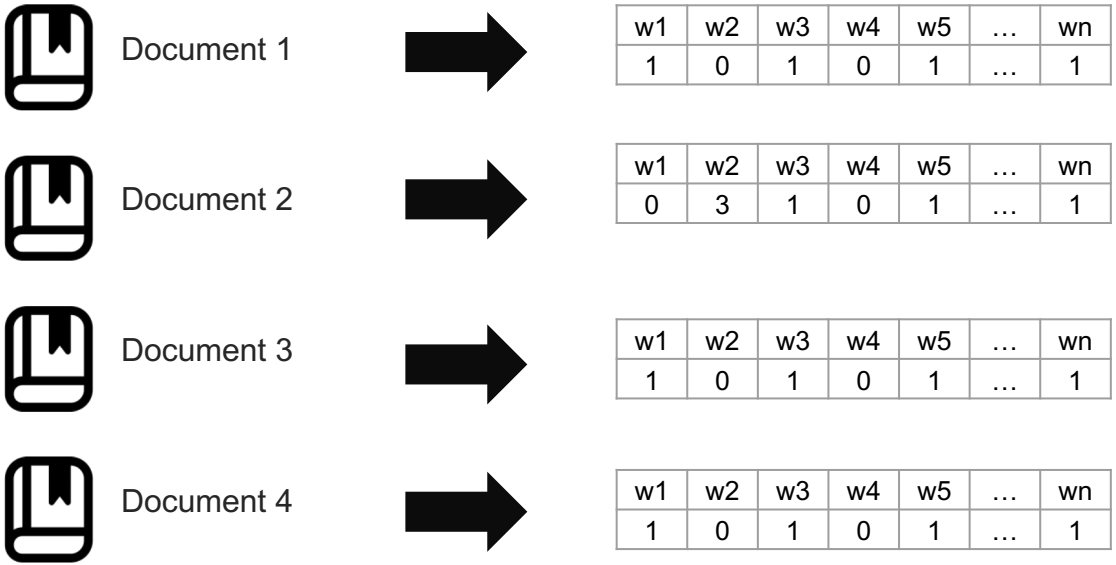
Text generation

Syntactic models

"Why Is The Rum Always Gone?"

Why	is	the	rum	always	gone	Other words
1	1	1	1	1	1	…

Syntactic models



w1	w2	w3	w4	w5	…	wn
1	0	1	0	1	…	1

w1	w2	w3	w4	w5	…	wn
0	3	1	0	1	…	1

w1	w2	w3	w4	w5	…	wn
1	0	1	0	1	…	1

w1	w2	w3	w4	w5	…	wn
1	0	1	0	1	…	1

Syntactic models

Titanic

ship	sea	iceberg	rum	draw	…	wn
80	3	7	0	111	…	1

Pirates of Caribbean

ship	sea	iceberg	rum	draw	…	wn
150	4	0	999	0	…	1

Forrest Gump

ship	sea	iceberg	rum	draw	…	wn
0	0	0	0	0	…	0

Matrix

ship	sea	iceberg	rum	draw	…	wn
10	0	0	0	2	…	0

- Syntactic models – WordClouds

Dr. House episode 15 season 4

Titanic Movie

Embedding Models

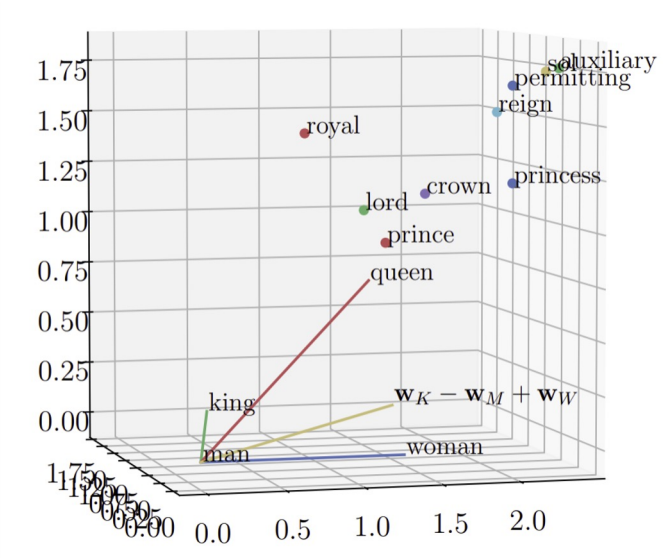
Document 1

d1	d2	d3	d4	d5	…	dn
0	3	1	0	1	…	1

Defined dimensional space

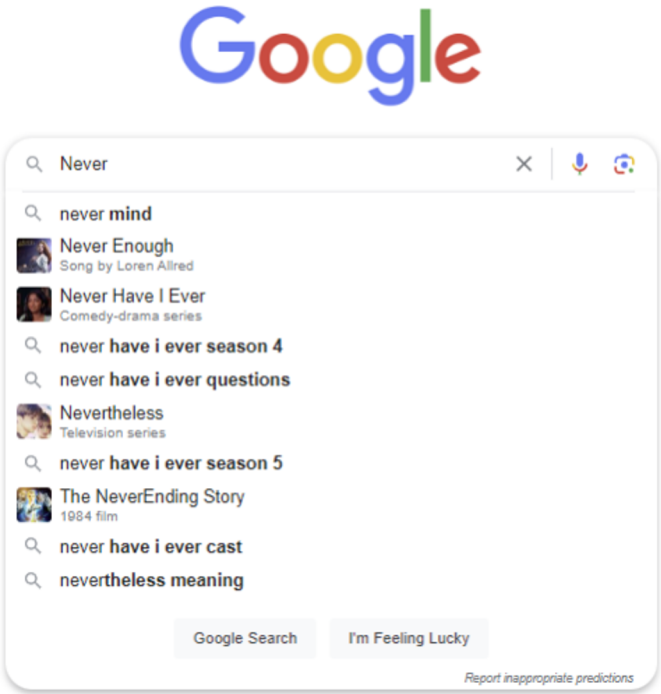
Embedding Models

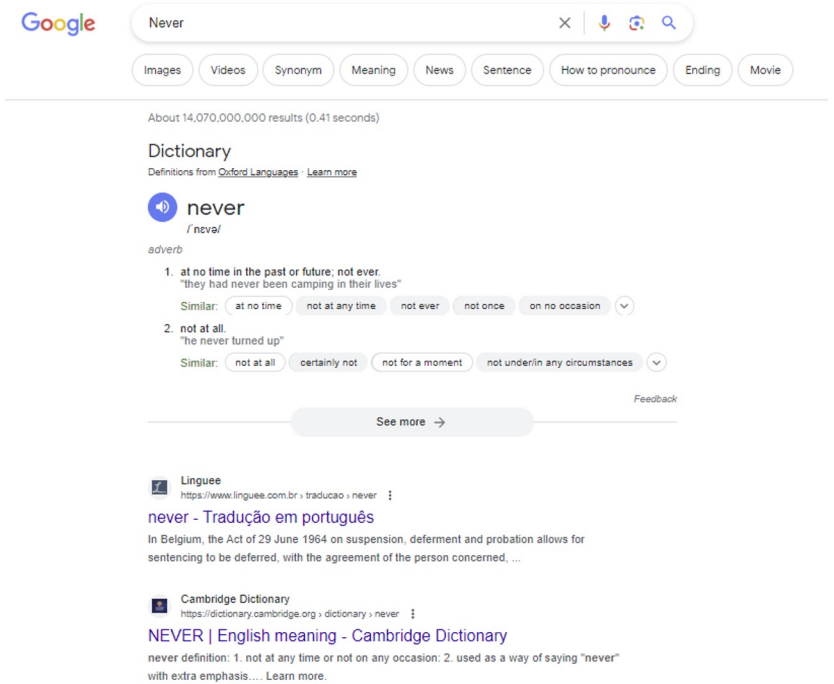
Wk-Wm+Wm



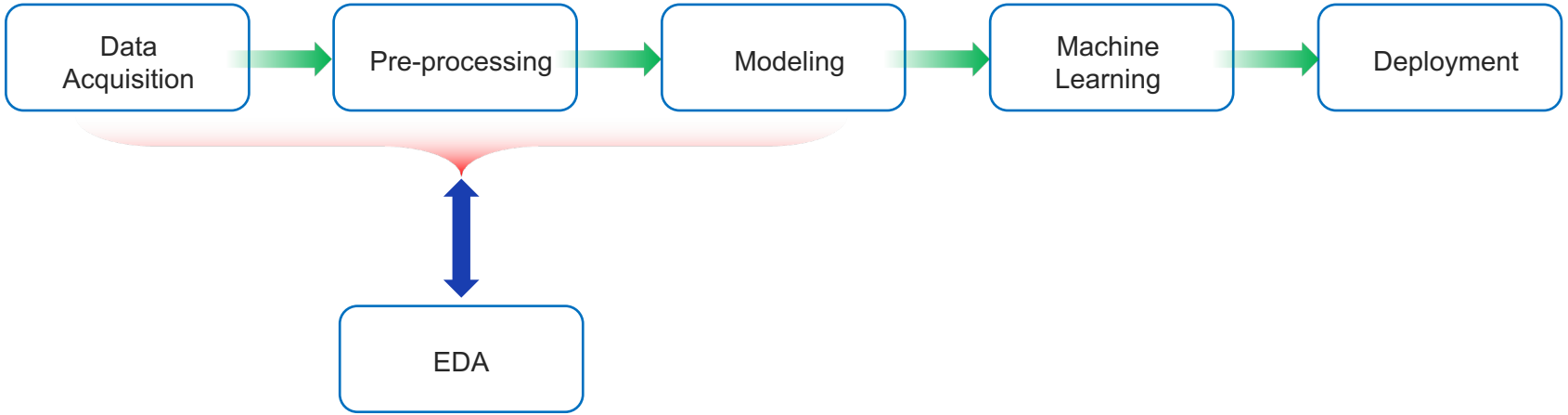
Analogies Explained: Towards Understanding Word Embeddings

Retrievals


Google
Never
never mind
Never Enough
Song by Loren Allred
Never Have I Ever
Comedy-drama series
never have i ever season 4
never have i ever questions
Nevertheless
Television series
never have i ever season 5
The NeverEnding Story
1984 film
never have i ever cast
nevertheless meaning
Google Search
I'm Feeling Lucky
Report inappropriate predictions


Google
Never
Images
Videos
Synonym
Meaning
News
Sentence
How to pronounce
Ending
Movie
About 14,070,000,000 results (0.41 seconds)
Dictionary
Definitions from Oxford Languages · Learn more
never
/ˈnevə/
adverb
1. at no time in the past or future; not ever.
"they had never been camping in their lives"
Similar: at no time | not at any time | not ever | not once | on no occasion
2. not at all.
"he never turned up"
Similar: not at all | certainly not | not for a moment | not under/in any circumstances
Feedback
See more
Linguee
https://www.linguee.com.br › traducao › never
never - Tradução em português
In Belgium, the Act of 29 June 1964 on suspension, deferment and probation allows for sentencing to be deferred, with the agreement of the person concerned, ...
Cambridge Dictionary
https://dictionary.cambridge.org › dictionary › never
NEVER | English meaning - Cambridge Dictionary
never definition: 1. not at any time or not on any occasion: 2. used as a way of saying "never" with extra emphasis.... Learn more.

NLP Workflow

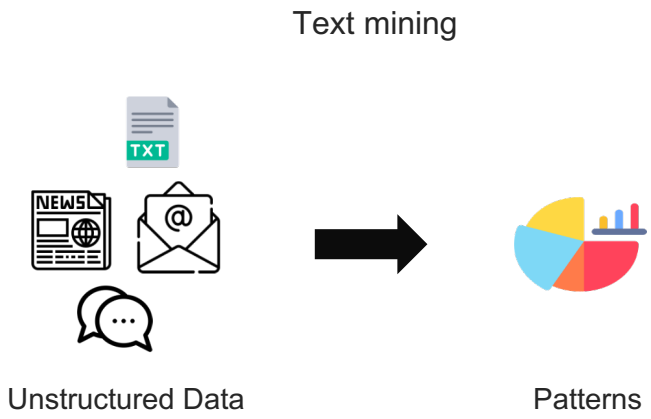


Unit 1.

Natural Language Processing

- 1.1. What is NLP?
- 1.2. Unstructured Data
- 1.3. Language Model
- 1.4. String manipulation and Regex
- 1.5. Tokenkanization

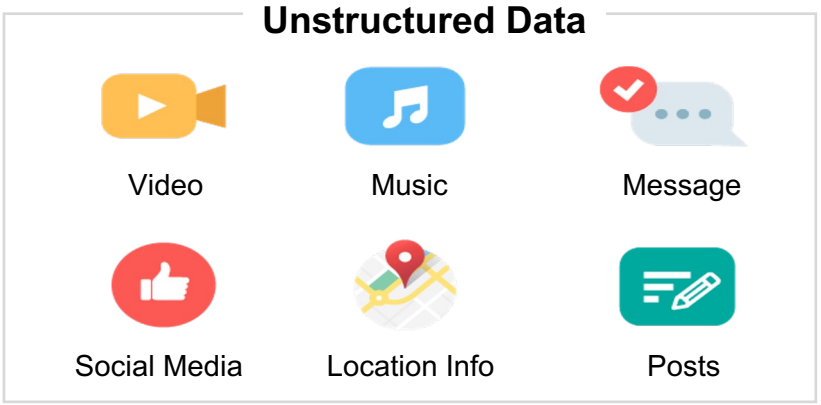
Text Mining



Text Mining

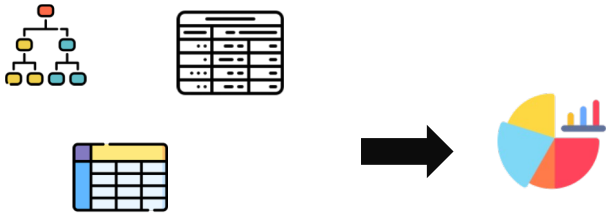
What is unstructured data?

- Data that is not yet structured. It does not have a defined data model (structure).
- Documents, videos, or audio with a large amount of data but with varying structures and forms.
- Books, journals, documents, metadata, health records, audio, video, analog data, images, files, e-mail messages, webpages, and word-processor documents are all composed of unstructured texts.



Data mining vs Text mining

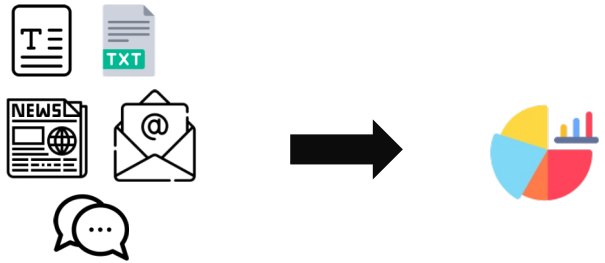
Data mining



Structure Data

Patterns

Text mining



Unstructured Data

Patterns

- Data mining extracts useful and valuable patterns from structured data.
- In contrast, text mining extracts named entities, patterns, or information on word-sentence relationships from unstructured data composed of natural language.

Collecting Data from Different Resources

- Collecting text is a process of establishing the plan for collecting and gathering data suitable for the task's objective and characteristics.
  - Various collecting techniques are used according to data type and features. Main techniques are listed below.

Technique	Features	Data Type
Crawling	- Collects web documents and information on the web, such as social media, news, and web information. - Follows URL link and repetitively collects.	Web document
Scraping	- Collects information from a single website (or document).	Web document
FTP	- Transmits and receives files from internet servers using TCP/IP protocol. - Considers using SFTP for reinforced security. - Considers constructing an exclusive network for linked servers.	File
Open API	- Offers data collecting method with an open API that allows easy access to service, information, and data.	Real-time data
RSS	- XML-based content distribution protocol that allows sharing of up-to-date web-based information.	Content

Unit 2.

Natural Language Processing

- 1.1. What is NLP?
- 1.2. Unstructured Data
- 1.3. Language Model
- 1.4. String manipulation and Regex
- 1.5. Tokenkanization

Language Model

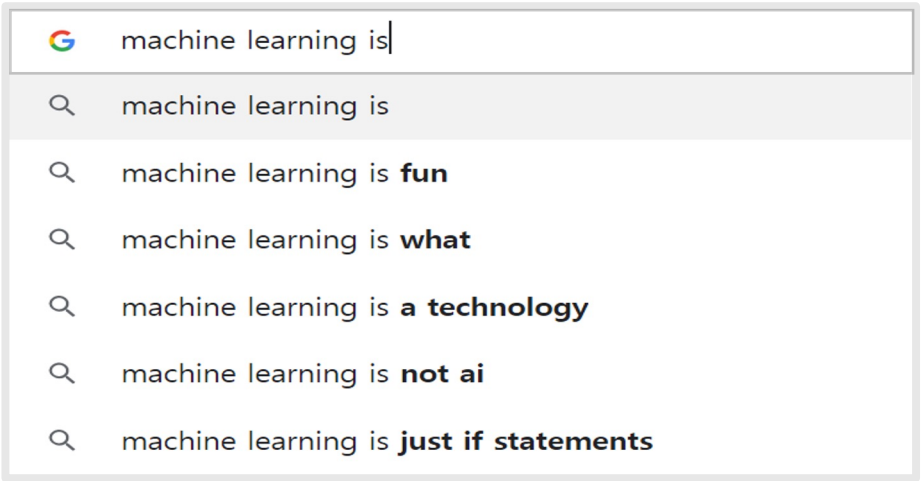
- Language model refers to a model that predicts or generates the next component by assigning the probability to elements of language (letter, word, morpheme, string (sentence), paragraph, etc.).
  - Language model is divided into statistical language model (SLM) and deep learning language model based on artificial neural network. Essentially language models, based on a given word, predicts the next word or combination of words. Such functions can solve numerous natural language processing problems like document generation, machine translation, document summarization, etc.

About language model

- Predicts the probability of a sequence: $P(w\_1,w\_2,w\_3,\ldots,w\_i)$
  Caution: The sub-index of w means the actual time order that cannot be changed.
- Given a sequence of words $\{w\_1,w\_2,w\_3,\ldots,w\_(i-1)\}$ what is the probability of w_i?
  $P(\llbracket w\_i \mid w \rrbracket\_1,w\_2,w\_3,\ldots,w\_(i-1))$?
- Data sparsity is a major problem because most (long) sequences appear very infrequently.
- Practical applications: machine translation, speech recognition, spell correction, autofill, etc.

About language model

Ex In the search engine:



Regular expression

- A joint probability can be expanded as following:

$$P(w_1, w_2, w_3, \ldots, w_m) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2)P(w_4|w_1, w_2, w_3) \cdots P(w_m|w_1, w_2, \ldots, w_{m-1})$$

$$= \prod_{i=1}^{m} P(w_i|w_1, \ldots, w_{i-1})$$

Ex P (three little pigs lived happily)?

⇒

	“little”	“pigs”	“lived”	“happily”

$$P(three\ little\ pigs\ lived\ happily)$$

$$= P(three)P(little|three)P(pigs|three\ little)P(lived|three\ little\ pigs)P(happily|three\ little\ pigs\ lived)$$

n-Gram approximations

- As the sequence grows, the probabilities become harder to estimate due to the data sparsity:

$$P(w_i|w_1, w_2, w_3, \dots, w_{i-1}) = \frac{Count(w_1, w_2, w_3, \dots, w_i)}{Count(w_1, w_2, w_3, \dots, w_{i-1})}$$

- Instead of an exact estimation of probabilities, we can do the so-called n -Gram approximation:

$$P(w_1, w_2, w_3, \dots, w_m) \approx \prod_{i=1}^{m} P(w_i|w_{i-(n-1)}, \dots, w_{i-1})$$

which can be compared with the following exact relation.

$$P(w_1, w_2, w_3, \dots, w_m) = \prod_{i=1}^{m} P(w_i|w_1, \dots, w_{i-1})$$

- => Usually the n above is a small positive number ≅1, 2, 3,…

n-Grams

- Given a text sequence, n-Grams can be constructed by sliding a “moving window” of length = n.

Ex “three little pigs lived happily”

→ $n = 1$, Unigrams = [“three,” “little,” “pigs’, “lived,” “happily”]

→ $n = 2$, Bigrams = [“three little,” “little pigs,” “pigs lived,” “lived happily”]

→ $n = 3$, Trigrams = [“**three little pigs**,” “**little pigs lived**,” “**pigs lived happily**”]

“three little pigs lived happily”

“three little pigs lived happily”

“three little pigs lived happily”

n-Gram approximations

- When n=1, it is the Unigram approximation:

$$P(w_1, w_2, w_3, \ldots, w_m) \approx P(w_1)P(w_2)P(w_3)\cdots P(w_m)$$

- When n=2, it is the Bigram approximation:

$$P(w_1, w_2, w_3, \ldots, w_m) \approx P(w_1)P(w_2|w_1)P(w_3|w_2)\cdots P(w_m|w_{m-1})$$

- When n=3, it is the Trigram approximation:

$$P(w_1, w_2, w_3, \ldots, w_m) \approx P(w_1)P(w_2|w_1)P(w_3|w_2, w_1)P(w_4|w_3, w_2)\cdots P(w_m|w_{m-1}, w_{m-2})$$

Ex Bigram approximation for Sequence = "*three little pigs lived happily*"

$$P(\boldsymbol{Sequence}) \approx P(three)P(little|three)P(pigs|little)P(lived|pigs)P(hapilly|lived)$$

Unit 1.

Natural Language Processing

- 1.1. What is NLP?
- 1.2. Unstructured Data
- 1.3. Language Model
- 1.4. String Manipulation and Regex
- 1.5. Tokenkanization

String Manipulation

- Useful functions and methods for string manipulation

Function	Explanation
x.lstrip()	Remove space on the left.
x.rstrip()	Remove space on the right.
x.strip()	Remove space on both sides.
x.replace(str1, str2)	Replace the substring str1 by str2.
x.count(str)	Number of occurrences of str in x.
x.find(str)	Find the sub-string str. Returns -1 if not found.
x.index(str)	Find the sub-string str. Throws an error if not found.
y.join(str_list)	Concatenate the elements of str_list using y as separator.
x.split(y)	Break up a string using y as separator.
x.upper()	Convert x into uppercase.
x.lower()	Convert x into lowercase.
len(x)	Returns the string length.

Regular Expression

- Metacharacters
  - Characters with special meanings in regular expressions.

. ^ $ * + ? { } [ ] \ | ( )

  - Can be used to construct patterns.
  - More information can be found at https://docs.python.org/3/howto/regex.html.

Metacharacters: [ ]

- Can enclose a set of characters as a match pattern.
- Any character can be enclosed by the [ ].
- For example, “[abc]” matches with any pattern containing “a” or “b” or “c” character.

RegEx	String	Match?	Explanation
“[abc]”	“a”	Yes	“a” is in the string.
“[abc]”	“before”	Yes	“b” is in the string.
“[abc]”	“dude”	No	There is neither “a” nor “b” nor “c” in the string.

Metacharacters: [ ]

- We can use a hyphen "-" to indicate a range of characters.
  - **Ex** "[0-5]" is the same as "[012345]."
  - **Ex** "[0-9]" means the entire set of number digits.
  - **Ex** "[a-d]" is the same as "[abcd]."
  - **Ex** "[a-zA-Z]" means the entire set of alphabet letters, both uppercase and lowercase.

Metacharacters: [^]

- Characters that are not in the enclosed set will be matched.
- " ^ " has to be the first character within the square brackets.

RegEx	String	Match?	Explanation
"[^abc]"	"a"	No	In the string, there is no other character than "a" or "b" or "c."
"[^abc]"	"before"	Yes	There are characters other than "a" or "b" or "c" in the string.
"[^abc]"	"dude"	Yes	There are characters other than "a" or "b" or "c" in the string.

Metacharacters: [ ] and [^]

- There are shorthand expressions as following:

→ “\w” is the same as “[a-zA-Z0-9_].”

→ “\W” is the same as “[^a-zA-Z0-9_].”

→ “\d” is the same as “[0-9].”

→ “\D” is the same as “[^0-9].”

→ “\s” means whitespace character.

→ “\S” means non-whitespace character.

Metacharacters: **Dot .**

- Dot matches with any character.
- “\.” is the dot as a character (not a metacharacter).

RegEx	String	Match?	Explanation
“a.b”	“aab”	Yes	“a” in the middle of the string matches with the dot.
“a.b”	“a0b”	Yes	“0” in the middle of the string matches with the dot.
“a.b”	“abc”	No	There is no character in between “a” and “b.”

Metacharacters: *

- Pattern that repeats the preceding character any number of times (including 0).

RegEx	String	Match?	Explanation
“ca*t”	“ct”	Yes	“a” does not appear.
“ca*t”	“cat”	Yes	“a” appears once.
“ca*t”	“caaat”	Yes	“a” is repeated three times.

Metacharacters: +

- Pattern that repeats the preceding character at least once or more times.

RegEx	String	Match?	Explanation
“ca+t”	“ct”	No	“a” does not appear.
“ca+t”	“cat”	Yes	“a” appears once.
“ca+t”	“caaat”	Yes	“a” is repeated three times.

Metacharacters: ?

- Pattern where the preceding character does not appear or appears just once.

RegEx	String	Match?	Explanation
“ca?t”	“ct”	Yes	“a” does not appear.
“ca?t”	“cat”	Yes	“a” appears once.
“ca?t”	“caat”	No	“a” is repeated twice (more than once).

Metacharacters: {m}

- Pattern where the preceding character is repeated *m* times.

RegEx	String	Match?	Explanation
“ca{2}t”	“ct”	No	“a” does not repeat twice.
“ca{2}t”	“cat”	No	“a” does not repeat twice.
“ca{2}t”	“caat”	Yes	“a” is repeated exactly twice.

Metacharacters: {m, n}

- Pattern where the preceding character is repeated from *m* to *n* times.

RegEx	String	Match?	Explanation
“ca{2,5}t”	“cat”	No	“a” is repeated less than two times.
“ca{2,5}t”	“caat”	Yes	“a” is repeated three times.
“ca{2,5}t”	“caaaaaat”	No	“a” is repeated more than five times.

Metacharacters: ^

- Pattern after the ^ matches with the beginning of a string or text.
- Not the same meaning as the first hat character within the square brackets “[^].”

RegEx	String	Match?	Explanation
“^Life”	“Life is boring”	Yes	“Life” pattern is found at the beginning of the string.
“^Life”	“My Life is boring”	No	“Life” pattern is not found at the beginning of the string.

Metacharacters: $

- Pattern before the $ matches with the end of a string or text.

RegEx	String	Match?	Explanation
“Python$”	“Python is easy”	No	“Python” pattern is not found at the end of the string.
“Python$”	“You need Python”	Yes	“Python” pattern is found at the end of the string.

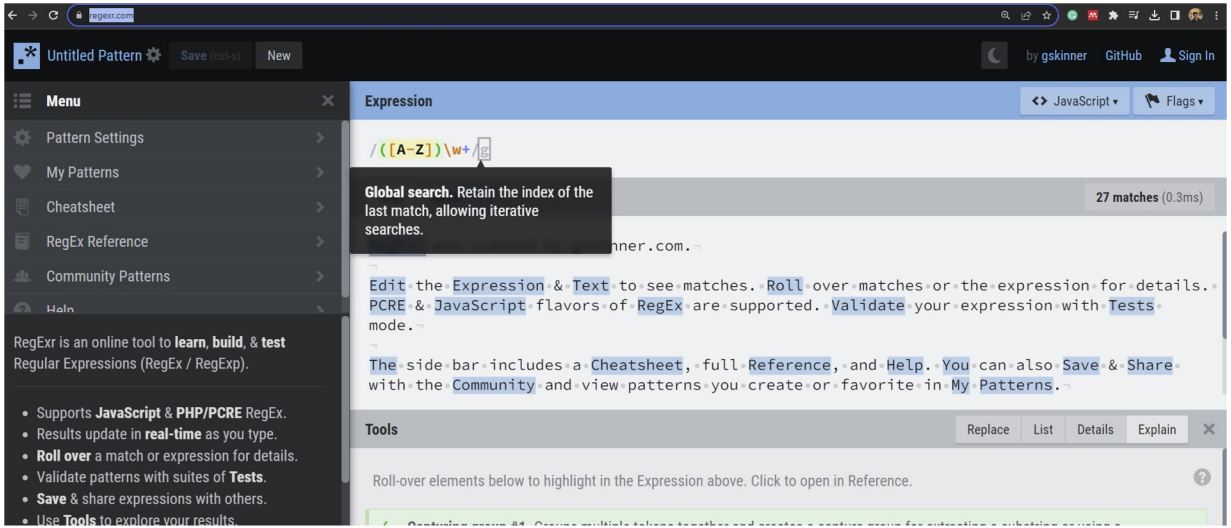
Metacharacters: |

- Used to join patterns by the logical or.
- More than two patterns can be concatenated by the logical or.

RegEx	String	Match?	Explanation
“love|hate”	“I love you”	Yes	“love” pattern found in the string.
“love|hate”	“I hate him”	Yes	“hate” pattern found in the string.
“love|hate”	“I like you”	No	Neither “love” nor “hate” pattern found in the string.

- How to train and test regex:

https://regexr.com/



Matching group patterns

- We can group patterns by enclosing them with ( ).

Ex

In [1]:
```
import re
my_regex  = re.compile( "([0-9]+)[^0-9]+([0-9]+)" )
m = my_regex.search("Anna is 15 years old and John is 12 years old.")
print(m.group(0))
print(m.group(1))
print(m.group(2))
```

```
15 years old and John is 12
15
12
```

- In the example, an equivalent regular expression is: my_regex = re.compile("(\d+)\D+(\d+)").

Unit 2.

Natural Language Processing

- 1.1. What is NLP?
- 1.2. Unstructured Data
- 1.3. Language Model
- 1.4. String Manipulation and Regex
- 1.5. Tokenization

Pre-processing

Pre-processing

- Tokenization: break down the raw text into words or sentences.
- Cleaning:
  - Remove punctuation marks, excess spaces, special characters, etc. (*)
  - Also remove excessively short or infrequent words, etc.
- Normalization:
  - Conversion to lowercase.
  - Remove the stop words.
  - Stemming and Lemmatization.
  - Expansion of abbreviations. (*)
- (*) Regular expressions can be useful.

Tokenization

- Often the first step in the data pre-processing
  - a) Tokenization into sentences:
    - Usually, the sentence boundary is marked by a period, exclamation mark, question mark, etc.
    - But sometimes, a period does not mark the end of a sentence: abbreviations, for example.

      Ex "He got his *M.D.* from the University of Wisconsin."
    - A good sentence tokenizer should recognize these exceptions.
  - b) Tokenization into words:
    - Usually splitting by white space or punctuation mark works.
    - In some cases, there is ambiguity: apostrophe, for example.

      Ex "*Don't* lose hope. *Everything's* possible."

      ⇒ How do we tokenize "*Don't*" and "*Everything's*"? ⇒ It depends on the tokenizer.

Tokenization

- Often the first step in the data pre-processing

I'm the king of the world!

I	am	the	king	of	the	world	titanic	!	…	robot
1	1	1	1	1	1	1	1	1	0	0

Tokenization

- Often the first step in the data pre-processing

Tokens

I'm the king of the world!

I	am	the	king	of	the	world	titanic	!	…	robot
1	1	1	1	1	1	1	1	1	0	0

Tokenization

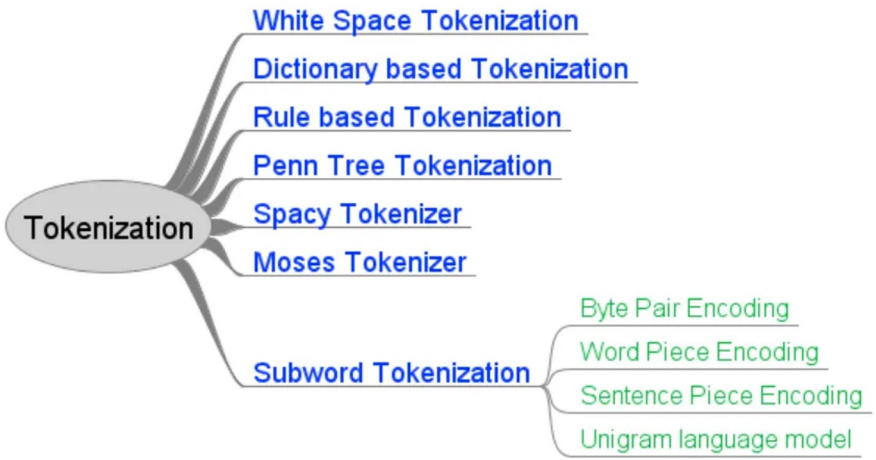
- Often the first step in the data pre-processing

Foundational Tokenization and Text Cleaning
(For syntactic models like word clouds, basic sentiment analysis, and simpler NLP tasks)

- **Basic Tokenization and Handling Simple Cases**: Understanding how to split text by spaces, handle contractions (like "I'm" vs. "I am"), and manage basic cases that ensure clear word boundaries.
- **Punctuation, Special Characters, and Normalization**: Learning to remove or separate punctuation, handle special characters, and apply normalization techniques (like lowercasing, removing accents) for cleaner input, which is crucial for visualizations like word clouds or preliminary sentiment analysis.
- **Multi-Word Expressions and Named Entities**: Recognizing multi-word expressions (e.g., "New York," "machine learning") as single tokens to avoid splitting phrases that carry important meaning.

Tokenization

- Often the first step in the data pre-processing



Tokenization

In [ ]:
```
# NLTK offers tools necessary for tokenization of English corpus.
```

In [1]:
```
from nltk.tokenize import word_tokenize
print(word_tokenize("Don't be fooled by the dark sounding name, Mr. Jone's Orphanage is as cheery as cheery goes for a past
```

```
['Do', "n't", 'be', 'fooled', 'by', 'the', 'dark', 'sounding', 'name', ',', 'Mr.', 'Jone', "'s", 'Orphanage', 'is', 'as',
'cheery', 'as', 'cheery', 'goes', 'for', 'a', 'pastry', 'shop', '.']
```

In [ ]:
```
# word_tokenize separated Don't into Do and n't, but Jone's into Jone and 's
```

In [2]:
```
from nltk.tokenize import WordPunctTokenizer
print(WordPunctTokenizer().tokenize("Don't be fooled by the dark sounding name, Mr. Jone's Orphanage is as cheery as cheery
```

```
['Don', "'", 't', 'be', 'fooled', 'by', 'the', 'dark', 'sounding', 'name', ',', 'Mr', '.', 'Jone', "'", 's', 'Orphanage',
'is', 'as', 'cheery', 'as', 'cheery', 'goes', 'for', 'a', 'pastry', 'shop', '.']
```

In [ ]:
```
# Keras, also as a tokenization tool, supports text_to_word_sequence
```

In [3]:
```
from tensorflow.keras.preprocessing.text import text_to_word_sequence
print(text_to_word_sequence("Don't be fooled by the dark sounding name, Mr. Jone's Orphanage is as cheery as cheery goes fo
```

```
["don't", 'be', 'fooled', 'by', 'the', 'dark', 'sounding', 'name', 'mr', "jone's", 'orphanage', 'is', 'as', 'cheery', 'a
s', 'cheery', 'goes', 'for', 'a', 'pastry', 'shop']
```