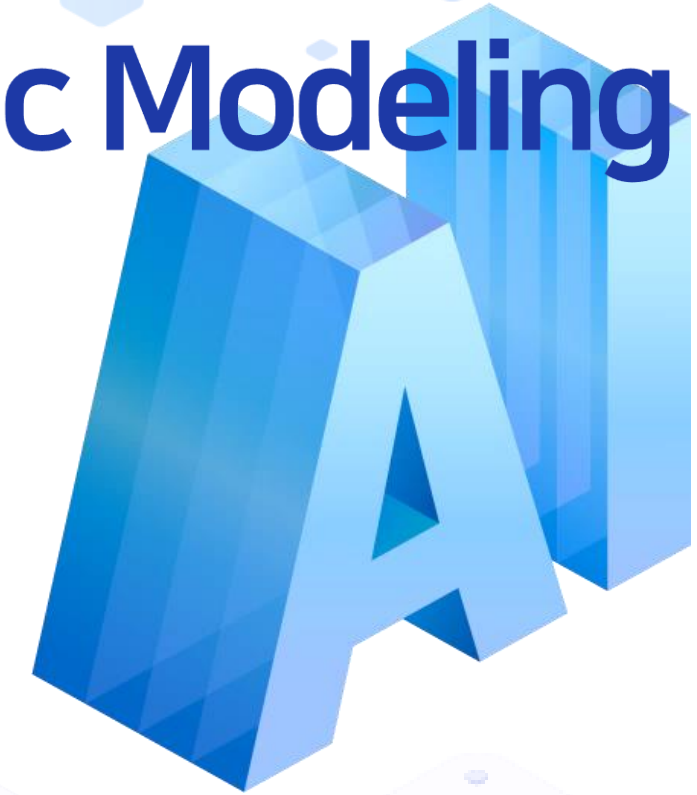
ReviewMind

Introduction to NLP & Topic Modeling



ReviewMind
CEO - Sunny Kim

2021.09.03.

ReviewMind

Objective

- Lesson 1: Introduction to NLP
- Lesson 2: NLP Basics – text preprocessing
- Lesson 3: Advanced NLP – word vectors
- Lesson 4: Topic Modeling

ReviewMind

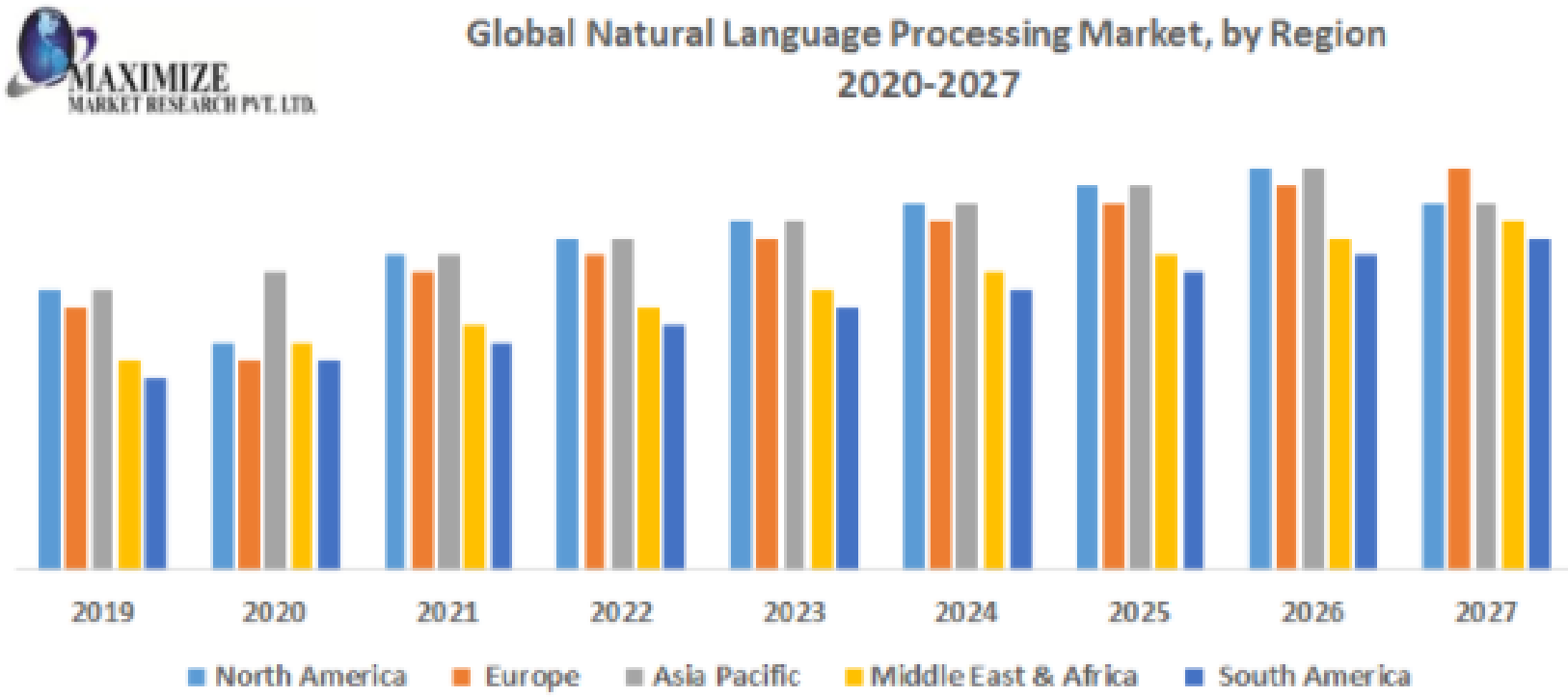
Lesson 1

Introduction to NLP

ReviewMind

Why NLP?

- Industry needs for unstructured text documents are getting higher
- According to industry lore, up to 80% of the information owned by large organizations is in the form of 'unstructured' text documents



※Source - https://www.maximizemarketresearch.com/market-report/global-natural-language-processing-market/29407/

ReviewMind

Natural Language Processing (NLP)

- **Natural language processing(NLP)** is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data.

  - wikipedia

※Source - Natural language processing - Wikipedia

ReviewMind

Text Analytics

- According to industry lore, up to 80% of the information owned by large organizations is in the form of ‘unstructured’ text documents
- Some processes are inherently textual (litigation support), others generate text as a byproduct (help desk)
- How do we get at and make use of that information?

ReviewMind

NLP Research Fields

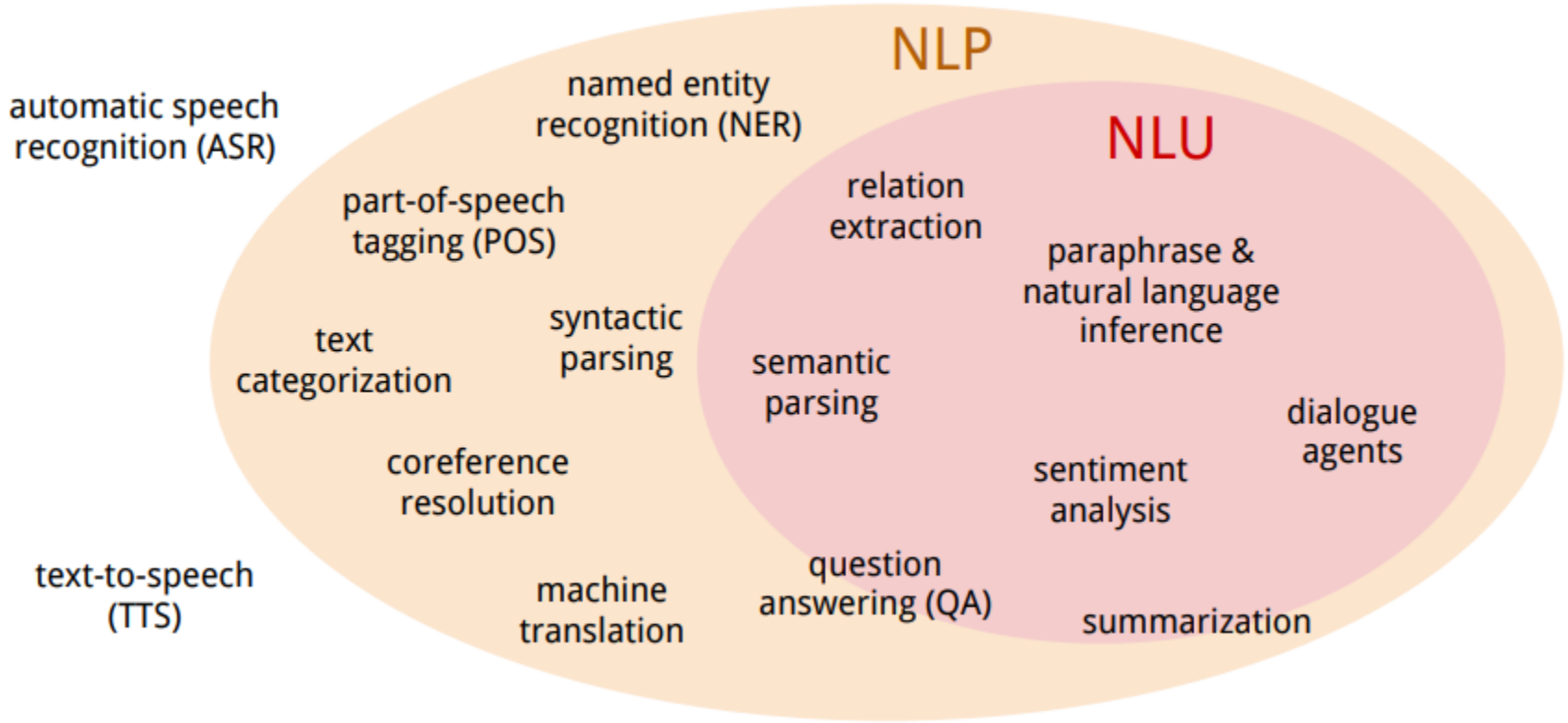
- It has been overlappingly studied in the different departments, and it tends to be unified in practical fields
  - Computational linguistics in linguistics department
  - Natural language processing in computer science
  - Speech recognition in electrical engineering
  - Computational psycholinguistics in psychology

※Source - Natural language processing - Wikipedia

ReviewMind

Terminology: NLU vs. NLP vs. ASR

- In more practical term, NLP relies on machine learning to derive meaning from human languages by analysis of the text semantics and syntax



※Source – https://nlp.stanford.edu/~wcmac/papers/20140716-UNLU.pdf

ReviewMind

Applications of NLP

- There can be a lot of NLP-based solutions for different real-world business scenarios

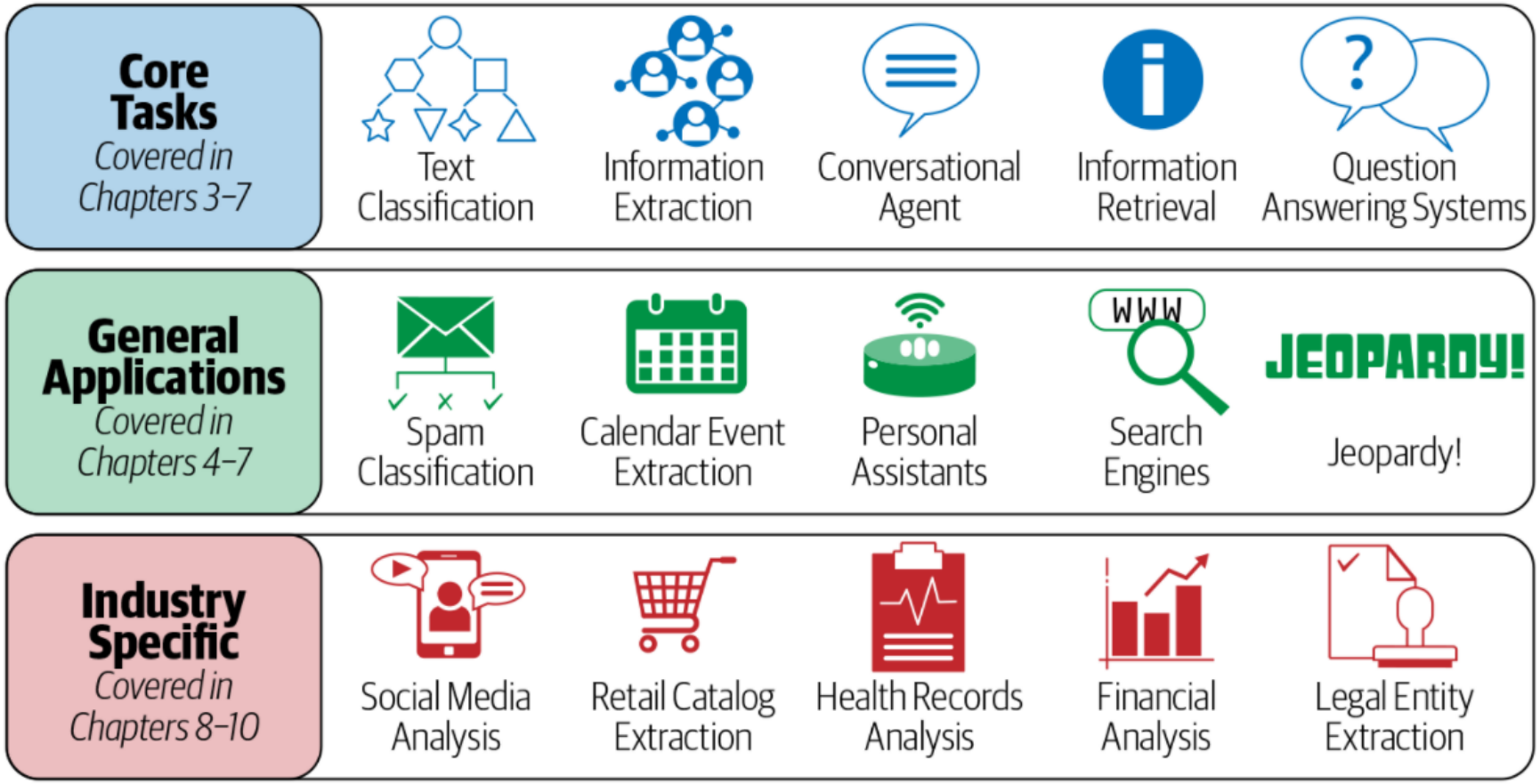


Figure 1-1. NLP tasks and applications

※Source – https://www.oreilly.com/library/view/practical-natural-language/9781492054047/ch01.html

ReviewMind

Applications of NLP

- A depiction of NLP tasks based on their relative difficulty in terms of developing comprehensive solutions

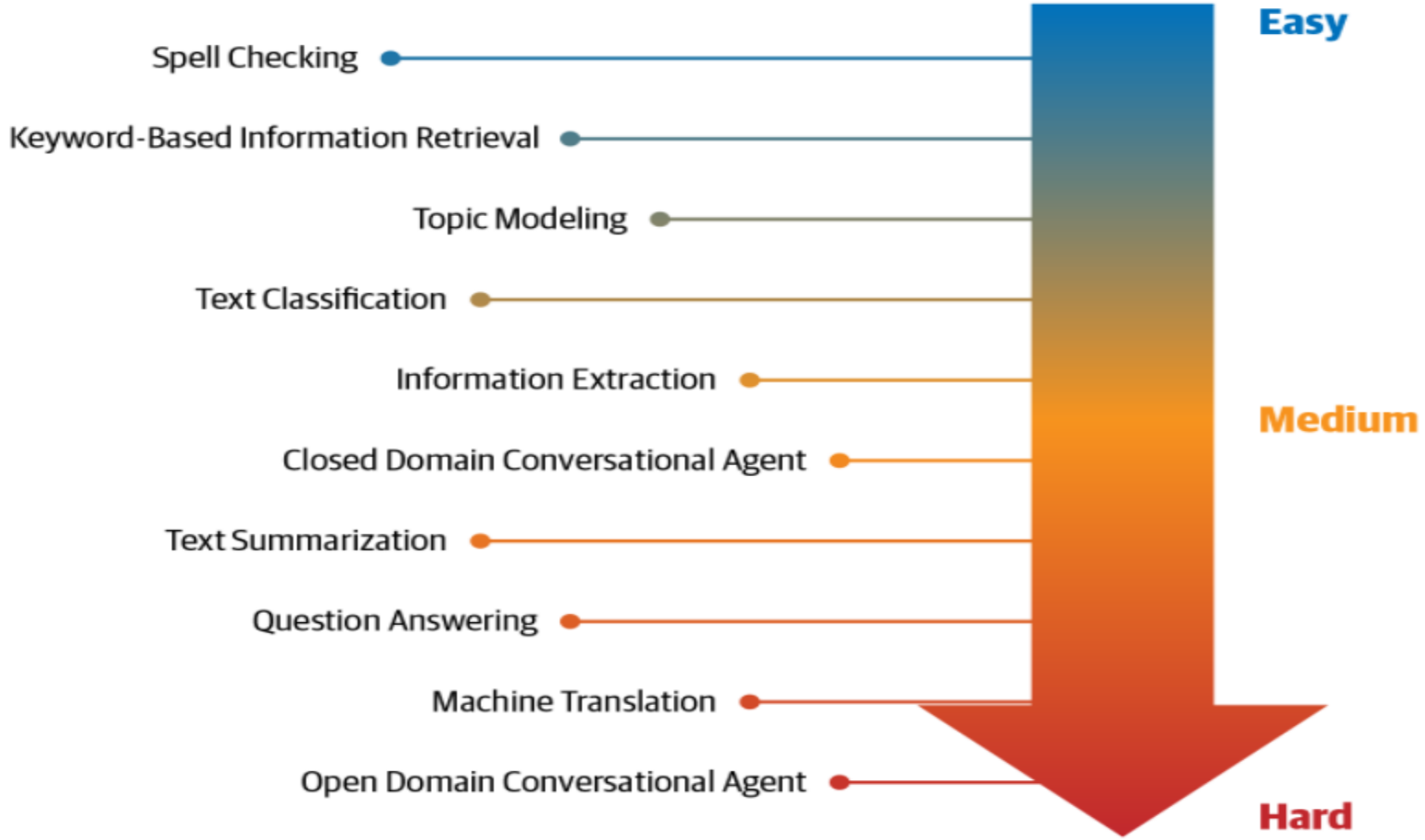


Figure 1-2. NLP tasks organized according to their relative difficulty

※Source – https://www.oreilly.com/library/view/practical-natural-language/9781492054047/ch01.html

ReviewMind

Lesson 2

NLP Basics

- Text Preprocessing

ReviewMind

NLP Basics: Pre-processing I

- Text data preprocessing usually involves a sequential combination of various processing steps depending on the problem domain and target model context

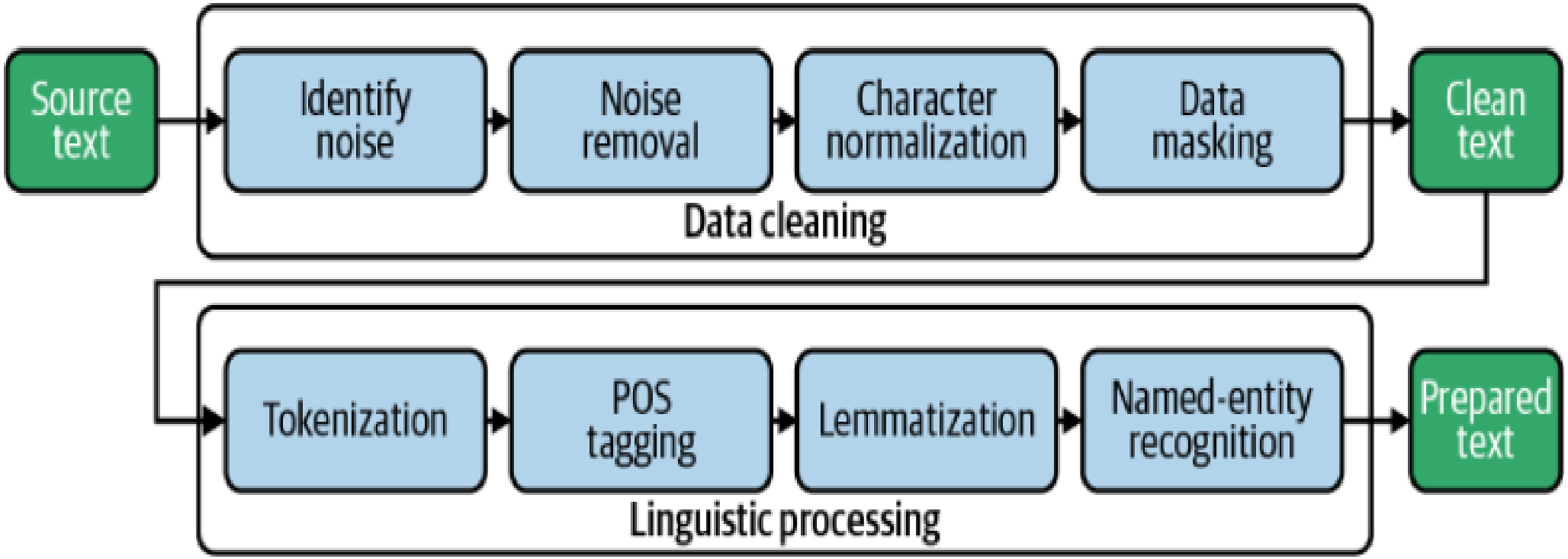


Figure 4-1. A pipeline with typical preprocessing steps for textual data

※Source – https://www.oreilly.com/library/view/blueprints-for-text/9781492074076/ch04.html

ReviewMind

NLP Basics: Pre-processing II

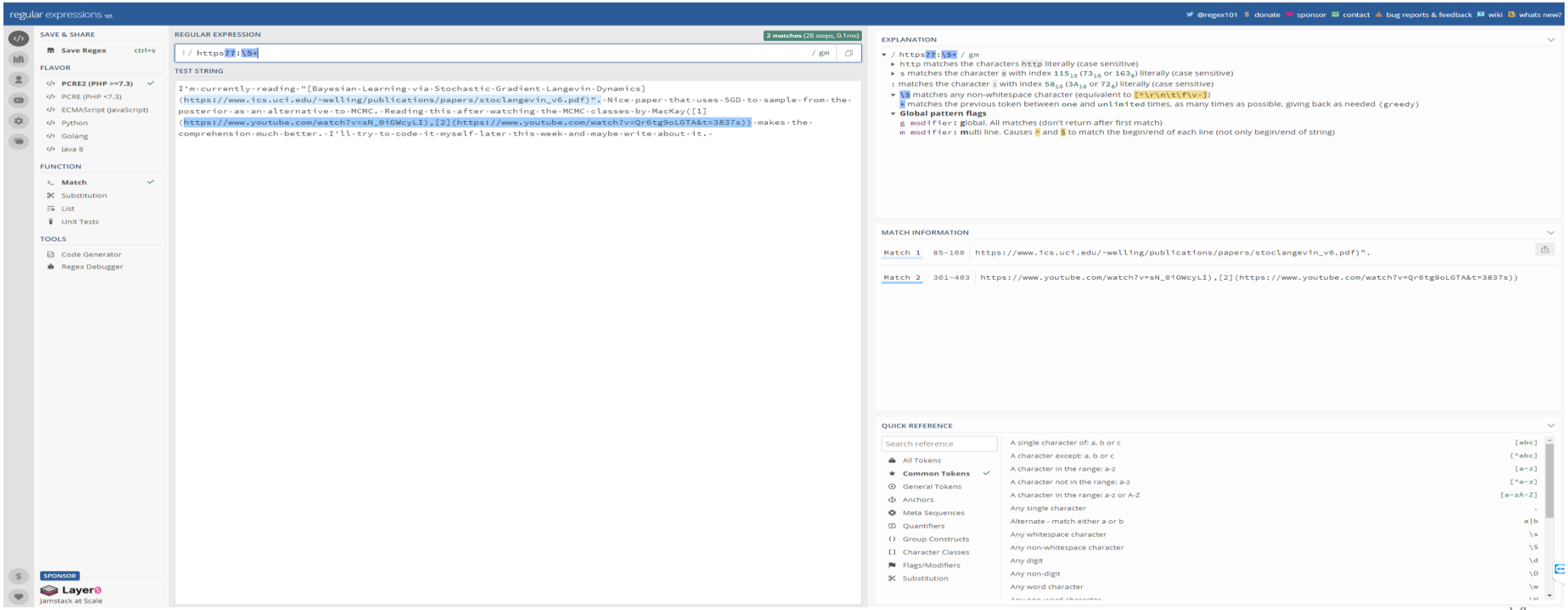
- Tokenization – dividing a continuous text into distinct units such as sentences, phrases, words
- Stemming – cutting the inflected words into their root form
- Lemmatization – reducing the inflected forms of a word into a single form for easy analysis
- Punctuation – removing all the punctuation which can be considered as a source of noise (ex. , . / ; ! …)
- Stop words – removing all the words which can be considered as a source of noise in our text data
  (ex. can be vary on the analysis objectives - prepositonal words such as "an", "the", etc.)
- Part of Speech (PoS) Tagging – a process of marking up a word in a text as corresponding to a particular part of speech based on both its definition and its context (ex. Noun, verb, adjective, etc)

※Source – https://www.kdnuggets.com/2019/07/nlp-vs-nlu-understanding-language-processing.html

ReviewMind

NLP Tools - Regular Expression I

- Regex is a sequence of characters mainly used to find and replace patterns in a string or document
  A very useful tool for cleaning and wrangling of text data. Recommend to utilize a regex tester



※Source - The Handbook of Computational Linguistics and Natural Language Processing, Clark et al., 2010

ReviewMind

NLP Tools - Regular Expression II

- Regex is a sequence of characters mainly used to find and replace patterns in a string or document
  A very useful tool for cleaning and wrangling of text data. Recommend to utilize a regex tester

Operators	Description
.	Matches with any single character except newline '\n'.
?	match 0 or 1 occurrence of the pattern to its left
+	1 or more occurrences of the pattern to its left
*	0 or more occurrences of the pattern to its left
\w	Matches with a alphanumeric character whereas \W (upper case W) matches non alphanumeric character.
\d	Matches with digits [0-9] and /D (upper case D) matches with non-digits.
\s	Matches with a single white space character (space, newline, return, tab, form) and \S (upper case S) matches any non-white space character.
\b	boundary between word and non-word and /B is opposite of /b
[..]	Matches any single character in a square bracket and [^..] matches any single character not in square bracket
\	It is used for special meaning characters like \. to match a period or \+ for plus sign.
^ and $	^ and $ match the start or end of the string respectively
{n,m}	Matches at least n and at most m occurrences of preceding expression if we write it as {,m} then it will return at least any minimum occurrence to max m preceding expression.
a| b	Matches either a or b
( )	Groups regular expressions and returns matched text
\t, \n, \r	Matches tab, newline, return

For more details on meta characters "(", ")","|" and others details , you can refer this link (https://docs.python.org/2/library/re.html).

* Import re

1. re.match()
2. re.search()
3. re.findall()
4. re.split()
5. re.sub()
6. re.compile()

※Source - https://www.analyticsvidhya.com/blog/2015/06/regular-expression-python

ReviewMind

NLP Tools - Regular Expression III

- Regex example 1 – find all punctuation with regex

```
def findAllPunct(df):
    df['punct'] = [re.findall(r'[^\w\s,]', x) for x in df['review_text']]
    return df

df_food = findAllPunct(test_food[:5])
```

	review_index	review_rank	review_text	punct
0	n0000000	5	뭉제 얼마나 오랫동안 고민고민후 구매했는지 몰라요~ ㅎㅎ\n4살아이 제방에서 독립시...	[~, ~, ~, ., ., ., ., ., ., ., ., ., ., ~, (, /, ...
1	n0000001	4	아직 써보지는 않았는데 아쉬운점이 좀 있네요\n여름에 더위많이타는 돌쟁이아들용으로 ...	[^, ^, ., ., ., ., ., ., ., ., ., ., -, ., ., ...

ReviewMind

NLP Tools - Regular Expression IV

- Regex example 2 – find erratic emoticons Korean often use with regex

```python
def erratic_expression(df):
    n_punctuation_list4 = []

    for x in df["review_text"]:
        #e = re.findall(r'\^\^', x)
        e = re.findall(r'\^.\^', x)

        if e != []:
            n_punctuation_list4.append(e)

    print(len(n_punctuation_list4))
    print(n_punctuation_list4[:3])
```

```
In [68]: erratic_expression(df_fashion)
         erratic_expression(df_food)
         erratic_expression(df_interior)
```

```
2820
[['^^^'], ['^-^'], ['^^^']]
3117
[['^~^'], ['^_^'], ['^—^']]
1816
[['^-^', '^-^'], ['^ ^'], ['^.^']]
```

ReviewMind

NLP Tools – SpaCy vs. NLTK

- SpaCy is an open source natural language processing library designed to effectively handle NLP tasks with the most efficient implementation of common algorithms
- For many NLP tasks, Spacy only has one implementation method, choosing the most efficient algorithm currently available. This means you have no option to choose
- NLTK is a representative and traditional open source in NLP, initially released in 2001 which is older than SpaCy
- NLTK provides many functionalities, but include less efficient implementations

ReviewMind

NLP Tools – SpaCy vs. NLTK

	⊕ PROS	⊖ CONS
Natural Language ToolKit 	+ The most well-known and full NLP library + Many third-party extensions + Plenty of approaches to each NLP task + Fast sentence tokenization + Supports the largest number of languages compared to other libraries	– Complicated to learn and use – Quite slow – In sentence tokenization, NLTK only splits text by sentences, without analyzing the semantic structure – Processes strings which is not very typical for object-oriented language Python – Doesn't provide neural network models – No integrated word vectors
spaCy	+ The fastest NLP framework + Easy to learn and use because it has one single highly optimized tool for each task + Processes objects; more object-oriented, comparing to other libs + Uses neural networks for training some models + Provides built-in word vectors + Active support and development	– Lacks flexibility, comparing to NLTK – Sentence tokenization is slower than in NLTK – Doesn't support many languages. There are models only for 7 languages and "multi-language" models

※Source – A Comparison of NLTK and SpaCy Frameworks (Bobriakov, 2018)

ReviewMind

Tokenization I

- Tokenization – dividing a continuous text into distinct units such as sentences, phrases, words

```
#Import the English model of spacy
nlp = spacy.load("en_core_web_sm")
df['doc'] = list(nlp.pipe(df['body']))
```

In [25]:
```
df[:2]
```

Out[25]:

	author	body	score	doc
0	rrmuller	I'm currently reading "[Bayesian Learning via ...	8	(I, 'm, currently, reading, ", [, Bayesian, Le...
1	probablyuntrue	Are these not stickied anymore?	7	(Are, these, not, stickied, anymore, ?)

ReviewMind

Tokenization II

- Tokenization – dividing a continuous text into distinct units such as sentences, phrases, words

```
from nltk.tokenize import word_tokenize

df['token'] = [word_tokenize(text) for text in df['body']]
```

In [16]:
```
df[:2]
```

Out[16]:

	author	body	score	token
0	rrmuller	I'm currently reading "[Bayesian Learning via ...	8	[I, 'm, currently, reading, ``, [, Bayesian, L...
1	probablyuntrue	Are these not stickied anymore?	7	[Are, these, not, stickied, anymore, ?]

ReviewMind

Stemming

- Stemming – cutting the inflected words into their root form

```
from nltk.stem.snowball import SnowballStemmer
from nltk.stem.porter import *
from nltk.stem import WordNetLemmatizer

def stemming(token_list):
    p_stemmer = PorterStemmer()
    return [p_stemmer.stem(word) for word in token_list]

df['stem'] = df['token'].apply(stemming)
```

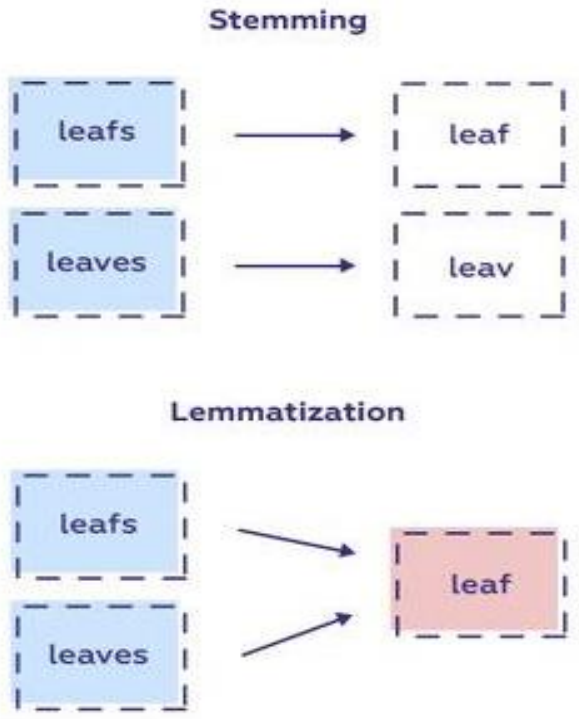
```
df[:3]
```

	token	stem
0	[I, 'm, currently, reading, ``, [, Bayesian, L...	[I, 'm, current, read, ``, [, bayesian, learn,...
1	[Are, these, not, stickied, anymore, ?]	[are, these, not, sticki, anymor, ?]
2	[I, 'm, currently, reading, [, Entity, Embeddi...	[I, 'm, current, read, [, entiti, embed, of, c...

ReviewMind

Lemmatization I

- Lemmatization – reducing the inflected forms of a word into a single form for easy analysis



※Source – https://medium.com/mlearning-ai/nlp-tokenization-stemming-lemmatization-and-part-of-speech-tagging-9088ac068768

ReviewMind

Lemmatization II

- Lemmatization – reducing the inflected forms of a word into a single form for easy analysis

```
def lemmatizer(token_list):
    wordnet_lemmatizer = WordNetLemmatizer()
    return [wordnet_lemmatizer.lemmatize(word) for word in token_list]

df['lemma'] = df['stem'].apply(lemmatizer)
```

```
df[10:12]
```

	token	stem	lemma
10	[Sometimes, it, just, takes, the, mods, a, day...	[sometim, it, just, take, the, mod, a, day, or...	[sometim, it, just, take, the, mod, a, day, or...
11	[I, 'm, reading, the, same, thing, ,, please, ...	[I, 'm, read, the, same, thing, ,, pleas, shar...	[I, 'm, read, the, same, thing, ,, plea, share...

ReviewMind

Stopwords I

- Stop words – removing all the words which can be considered as a source of noise in our text data (ex. can be vary on the analysis objectives - prepositonal words such as “an”, “the”, etc.)

```
#Filter stopword
def remove_stopword(token_list):
    nltk_stop_words = set(stopwords.words("english"))
    return [word for word in token_list if word not in nltk_stop_words]

df['rm_stop'] = df['lemma'].apply(remove_stopword)
```

```
df[1:4]
```

	token	stem	lemma	rm_stop
1	[Are, these, not, stickied, anymore, ?]	[are, these, not, sticki, anymor, ?]	[are, these, not, sticki, anymor, ?]	[sticki, anymor, ?]
2	[I, 'm, currently, reading, [, Entity, Embeddi...	[I, 'm, current, read, [, entiti, embed, of, c...	[I, 'm, current, read, [, entiti, embed, of, c...	[I, 'm, current, read, [, entiti, embed, categ...
3	[Currently, reading, [, Causal, inference, for...	[current, read, [, causal, infer, for, recomme...	[current, read, [, causal, infer, for, recomme...	[current, read, [, causal, infer, recommend,]...

ReviewMind

Stopwords II

- Stop words – removing all the words which can be considered as a source of noise in our text data
  (ex. can be vary on the analysis objectives - prepositonal words such as “an”, “the”, etc.)

```
nltk_stop_words = set(stopwords.words("english"))
print(nltk_stop_words)
```

```
{"isn't", 'a', 'her', 'don', 'were', 'there', 'by', 'who', "should've",
'weren', 'above', 'doing', 'yourselves', 'your', 'hers', 'have', 'eac
h', 'what', "couldn't", 'didn', 'couldn', "don't", "you've", 'any', 'do
esn', 'not', 'has', 'hasn', 'his', "shouldn't", 're', 'hadn', 'he', 'wa
sn', "aren't", 'most', "wouldn't", 'yours', 'i', 'until', 'ma', 'would
n', 'aren', 'on', "haven't", 'an', 'whom', 'are', "mightn't", 'this',
"hadn't", 'at', "you'll", "she's", 'up', 'can', 'when', 'o', 'over', 'w
on', 'with', "it's", 'for', 'was', 'about', 'off', 'haven', 'those', 'i
sn', 'below', 'themselves', 'myself', 'after', 'down', 'so', 'just', 'n
ow', 'more', 'its', 'been', 'ain', 'few', "shan't", 'than', 't', 've',
'under', 'own', 'should', 'having', 'd', 'before', 'him', 'himself', 'd
oes', 'll', 'all', 'the', 'nor', 'their', 'needn', 'you', 'being', 'her
self', 'against', 'between', 'my', 'them', 'how', 'and', 'while', 'di
d', 'm', "you'd", 'me', 'through', 'our', 'shan', 'such', 'theirs', 'du
ring', 'in', 'to', "wasn't", "weren't", 'because', 'both', 'very', "nee
dn't", 'why', 'shouldn', "you're", 'is', 'be', 'or', 'into', 'no', 'w
e', 'ours', 'some', 'too', 'am', 'she', 'that', 'other', 'do', 'as', 't
hen', 'further', 'only', "won't", 'these', 'will', 'where', 'mustn', 'y
ourself', "that'll", 'same', 'y', "doesn't", 'ourselves', 'mightn', 'on
ce', "didn't", 'had', 'itself', 'out', 'if', 'they', 'it', 'which', 'bu
t', 'of', 'again', 'here', "mustn't", "hasn't", 's', 'from'}
```

ReviewMind

Punctuation

- Punctuation – removing all the punctuation which can be considered as a source of noise (ex. , . / ; ! ···)

```
#Removing Punctuation
def remove_punct(token_list):
    punctuations="?:!.,;\[,\],\("
    return [word for word in token_list if word not in punctuations]

df['rm_punct'] = df['rm_stop'].apply(remove_punct)
```

```
df.head()
```

	token	stem	lemma	rm_stop	rm_punct
0	[I, 'm, currently, reading, ``, [, Bayesian, L...	[I, 'm, current, read, ``, [, bayesian, learn,...	[I, 'm, current, read, ``, [, bayesian, learn,...	[I, 'm, current, read, ``, [, bayesian, learn,...	[I, 'm, current, read, ``, bayesian, learn, vi...
1	[Are, these, not, stickied, anymore, ?]	[are, these, not, sticki, anymor, ?]	[are, these, not, sticki, anymor, ?]	[sticki, anymor, ?]	[sticki, anymor]
2	[I, 'm, currently, reading, [, Entity, Embeddi...	[I, 'm, current, read, [, entiti, embed, of, c...	[I, 'm, current, read, [, entiti, embed, of, c...	[I, 'm, current, read, [, entiti, embed, categ...	[I, 'm, current, read, entiti, embed, categor,...

ReviewMind

Part of Speech (POS) Tagging

- Part of Speech (PoS) Tagging – a process of marking up a word in a text as corresponding to a particular part of speech based on both its definition and its context (ex. Noun, verb, adjective, etc)

```
# Part of Speech Tagging
nlp = spacy.load("en_core_web_sm")
df['doc'] = list(nlp.pipe(df['body']))
pos_test = [(token, token.pos_) for token in df['doc'][0]]
print(pos_test)
```

```
[(I, 'PRON'), ('m, 'AUX'), (currently, 'ADV'), (reading, 'VERB'), ('',
'PUNCT'), ([, 'PUNCT'), (Bayesian, 'PROPN'), (Learning, 'PROPN'), (via,
'ADP'), (Stochastic, 'PROPN'), (Gradient, 'PROPN'), (Langevin, 'PROP
N'), (Dynamics](https://www.ics.uci.edu/~welling, 'NOUN'), (/, 'SYM'),
(publications, 'NOUN'), (/, 'SYM'), (papers, 'NOUN'), (/, 'SYM'), (stoc
langevin_v6.pdf, 'NOUN'), (), 'PUNCT'), ('', 'PUNCT'), (., 'PUNCT'), (Ni
ce, 'ADJ'), (paper, 'NOUN'), (that, 'DET'), (uses, 'VERB'), (SGD, 'PROP
N'), (to, 'PART'), (sample, 'VERB'), (from, 'ADP'), (the, 'DET'), (post
erior, 'NOUN'), (as, 'ADP'), (an, 'DET'), (alternative, 'NOUN'), (to,
'ADP'), (MCMC, 'PROPN'), (., 'PUNCT'), (Reading, 'VERB'), (this, 'DE
T'), (after, 'ADP'), (watching, 'VERB'), (the, 'DET'), (MCMC, 'PROPN'),
(classes, 'NOUN'), (by, 'ADP'), (MacKay([1](https://www.youtube.com, 'N
UM'), (/, 'SYM'), (watch?v, 'PROPN'), (=, 'PUNCT'), (sN_OiGWcyLI).[2](h
ttps://www.youtube.com, 'X'), (/, 'SYM'), (watch?v, 'NOUN'), (=, 'SY
M'), (Qr6tg9oLGTA&t=3837s, 'PROPN'), (), 'PUNCT'), (), 'PUNCT'), (make
s, 'VERB'), (the, 'DET'), (comprehension, 'NOUN'), (much, 'ADV'), (bett
er, 'ADV'), (., 'PUNCT'), (I, 'PRON'), ('ll, 'AUX'), (try, 'VERB'), (t
o, 'PART'), (code, 'VERB'), (it, 'PRON'), (myself, 'PRON'), (later, 'AD
V'), (this, 'DET'), (week, 'NOUN'), (and, 'CCONJ'), (maybe, 'ADV'), (wr
ite, 'VERB'), (about, 'ADP'), (it, 'PRON'), (., 'PUNCT'), ( , 'SPACE')]
```

ReviewMind

Lesson 3

Advanced NLP - Word vectors

ReviewMind

Terminology – Corpus

- Corpus is a language resource of a large and structured set of texts/documents (nowadays usually electronically stored and processed)

※Source – https://en.wikipedia.org/wiki/Text_corpus

ReviewMind

Word Representation

- The vast majority of rule-based and statistical NLP work regards words as atomic symbols – Manning
  : which means every unique word in the corpus is considered as a feature

- Bag of Words (One-hot encoding)
  : In the vector space of all BOW, the presence of a word in a document is encoded as 1 other than as 0

- Term Frequency (TF) Representation (= Count Vectorization)
  : In the same vector, the frequency of a word in a document is encoded

$$TF(\boldsymbol{term}) = \frac{Number\ of\ times\ \boldsymbol{term}\ appears\ in\ a\ document}{Total\ number\ of\ items\ in\ the\ document}$$

- Term Frequency–Inversed Document Frequency (TF-IDF) Representation
  : This method tried to reflect the importance of the word in the corpus based on how common a word is in the corpus

$$IDF(\boldsymbol{term}) = \log\left(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ \boldsymbol{term}\ in\ it}\right)$$

$$TFIDF(\boldsymbol{term}) = TF(\boldsymbol{term}) * \mathrm{IDF}(\boldsymbol{term})$$

※Source – https://www.analyticsvidhya.com/blog/2021/06/part-5-step-by-step-guide-to-master-nlp-text-vectorization-approaches/

ReviewMind

TF Vectorization I

- TF vector example

	text
0	Eddard Stark is a king in the north.
1	A king but one king : kings are everywhere.
2	Hodor was different : he was not a king .
3	But the North could not change without him.

	king	was	the	not	But	him	one	north	kings	is	in	he	Eddard	everywhere	different	could	change	but	are	Stark	North	Hodor	without
0	1	0	1	0	0	0	0	1	0	1	1	0	1	0	0	0	0	0	0	1	0	0	0
1	2	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0	1	1	0	0	0	0
2	1	2	0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	0
3	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	0	1

※Source – https://knowledge.dataiku.com/latest/courses/nlp-visual/text-handling/text-handling-summary.html

ReviewMind

TF Vectorization II - sklearn

- TF Vectorization example

```
corpus = ["This is a brown house. This house is big.",
          "This is a small house. This house has 1 bedroom. ",
          "This dog is brown. This dog likes to play.",
          "The dog is in the bedroom."]
from sklearn.feature_extraction.text import CountVectorizer
vect = CountVectorizer()
X = vect.fit_transform(corpus)
X.toarray()
```

```
array([[0, 1, 1, 0, 0, 2, 0, 2, 0, 0, 0, 0, 2, 0],
       [1, 0, 0, 0, 1, 2, 0, 1, 0, 0, 1, 0, 2, 0],
       [0, 0, 1, 2, 0, 0, 0, 1, 1, 1, 0, 0, 2, 1],
       [1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 2, 0, 0]], dtype=int64)
```

ReviewMind

TF-IDF Vectorization I

- TF-IDF vector example

	text	tf	idf
0	Eddard Stark is a king in the north.	1	3
1	A king but one king : kings are everywhere.	2	3
2	Hodor was different : he was not a king .	1	3
3	But the North could not change without him.	0	3

	king	was	the	not	a		he	one	north	kings	is	in	him	everywhere	A	different	could	change	but	are	Stark	North	Hodor	Eddard
0	0.333333	0.0	0.5	0.0	0.5	0.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0
1	0.666667	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0
2	0.333333	2.0	0.0	0.5	0.5	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
3	0.000000	0.0	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0

※Source – https://knowledge.dataiku.com/latest/courses/nlp-visual/text-handling/text-handling-summary.html

ReviewMind

TF-IDF Vectorization II - sklearn

- TF-IDF Vectorization example

```
corpus = ["This is a brown house. This house is big.",
          "This is a small house. This house has 1 bedroom. ",
          "This dog is brown. This dog likes to play.",
          "The dog is in the bedroom."]
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(corpus)
X.toarray()
```

```
array([[0.        , 0.38272682, 0.30174622, 0.        , 0.        ,
        0.60349245, 0.        , 0.39944547, 0.        , 0.        ,
        0.        , 0.        , 0.48857915, 0.        ],
       [0.29778054, 0.        , 0.        , 0.        , 0.37769685,
        0.59556108, 0.        , 0.19709789, 0.        , 0.        ,
        0.37769685, 0.        , 0.48215802, 0.        ],
       [0.        , 0.        , 0.27857283, 0.55714567, 0.        ,
        0.        , 0.        , 0.18438451, 0.35333432, 0.35333432,
        0.        , 0.        , 0.45105744, 0.35333432],
       [0.30887228, 0.        , 0.        , 0.30887228, 0.        ,
        0.        , 0.39176533, 0.2044394 , 0.        , 0.        ,
        0.        , 0.78353065, 0.        , 0.        ]])
```

ReviewMind

Word Embedding

- Traditional word representation methods have its problem that there is no natural notion of similarity between words, so that the word embedding seeks the distributional similarity-based representation by learning the relational meaning with neighbor words of a word using neural network
- Word2Vec, Glove, FastText, etc. are all in the word embedding methodology group

※Source – Stanford university lecture – CS 224N NLP

ReviewMind

Word Embedding – Learning

- The basic idea of learning neural network word embeddings

We define a model that aims to predict between a center word w_t and context words in terms of word vectors

$$p(\text{context} \mid w_t) = \ldots$$

which has a loss function, e.g.,

$$J = 1 - p(w_{-t} \mid w_t)$$

We look at many positions t in a big language corpus

We keep adjusting the vector representations of words to minimize this loss

Sta
Uni

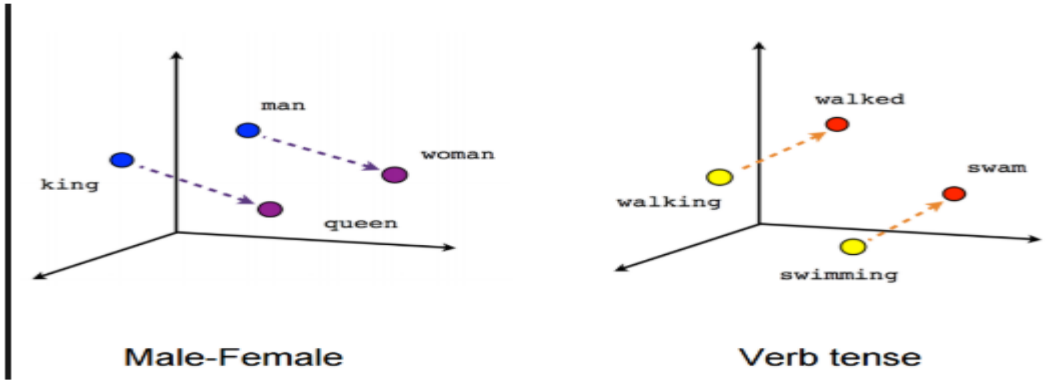
※Source – Stanford university lecture – CS 224N NLP

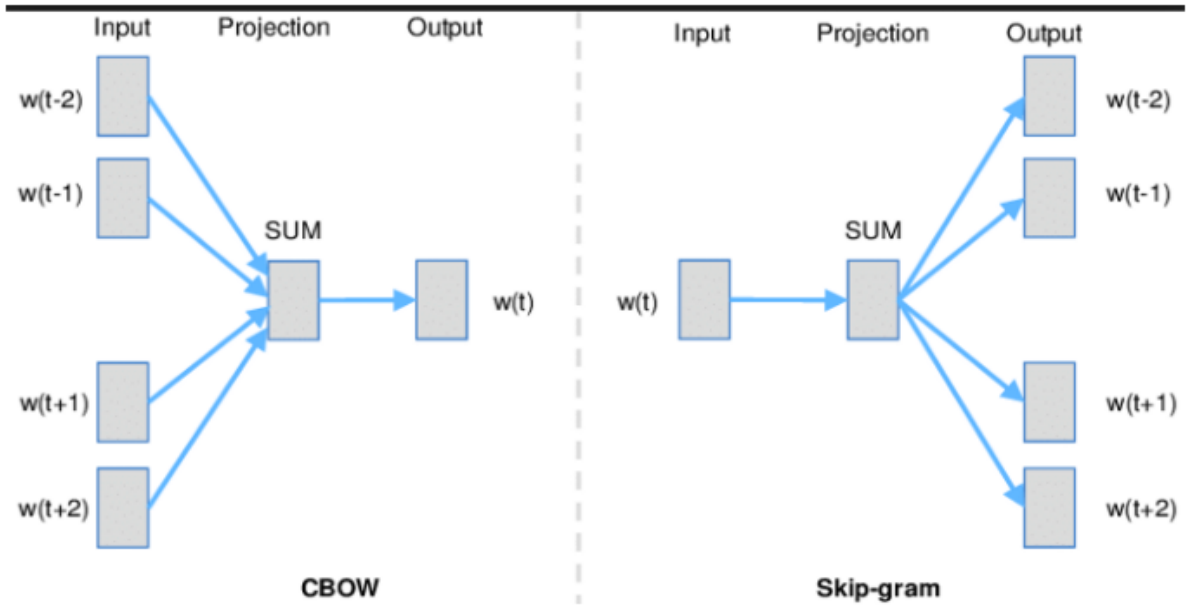
ReviewMind

Word2Vec I

- Word2Vec starts with one representation of all words in the corpus and train a NN (with 1 hidden layer) on a very large corpus of data (such as Google News corpus)

- Two algorithms
  - Skip-grams: predict context words given target (position independent)
  - Continuous Bag of Words (CBOW): predict target from bag-of-words context





※Source – https://medium.com/voice-tech-podcast/nlp-pipeline-101-with-basic-code-example-feature-extraction-ea9894ed8daf

ReviewMind

Word2Vec II - gensim

- Word2Vec example

```
import gensim
from gensim.models import KeyedVectors

# Load vectors directly from the file
model = KeyedVectors.load_word2vec_format('./Pre_trained_vectors/GoogleNews-vectors-
negative300.bin.gz', binary=True)

# Access vectors for specific words with a keyed lookup:
vector = model['easy']
# see the shape of the vector (300,)
vector.shape

print(vector)
print(vector.shape)
```

```
 3.54003906e-02  1.28906250e-01 -2.85644531e-02 -3.80859375e-01
-1.03515625e-01 -6.93359375e-02  1.39648438e-01  1.19140625e-01
 2.27050781e-02  9.61914062e-02 -8.64257812e-02  1.39648438e-01
 3.20312500e-01 -1.84570312e-01  8.59375000e-02 -3.51562500e-02
-6.59179688e-02 -1.38671875e-01 -2.10937500e-01 -7.32421875e-02
-2.71484375e-01  1.83593750e-01 -9.81445312e-02  6.07910156e-02
-7.91015625e-02  1.75781250e-01 -1.17187500e-01  1.53320312e-01
 8.25195312e-02 -4.54101562e-02 -2.14843750e-01  2.64892578e-02
 1.31835938e-01 -1.59912109e-02 -2.16796875e-01  1.01562500e-01]
(300,)
```

ReviewMind

Word2Vec III - gensim

- Word2Vec example

```
# Some predefined functions that show content related information for given words
print(model.most_similar(positive=['woman', 'queen'], negative=['man']))
print(model.similarity('woman', 'queen'))
```

```
[('princess', 0.6431564688682556), ('queens', 0.6387215256690979), ('very
_pampered_McElhatton', 0.5774043798446655), ('Queen_Consort', 0.550426661
9682312), ('Queen', 0.5450494289398193), ('princesses', 0.542153954505920
4), ('duchess', 0.5339502096176147), ('empress', 0.526210845167542), ('m
onarch', 0.521640300750732 4), ('Princess', 0.52029603719711 3)]
0.3161814
```

ReviewMind

FastText - gensim

- An extended technique of Word2Vec proposed by Facebook in 2016 by feeding the several n-grams (sub-words) instead of individual words into the Neural Network (ex, apple – app, ppl, and ple - ignoring the starting and ending of boundaries of words)

```
# Word2vec Model.
WVmodel = gensim.models.Word2Vec(min_count=1, vector_size=50)
WVmodel.build_vocab(sentences)
WVmodel.train(sentences, total_examples=len(sentences), epochs=30)
# ----------------------------------
# FastTest Model.
FTmodel = gensim.models.fasttext.FastText(min_count=1, vector_size=50)
FTmodel.build_vocab(sentences)
FTmodel.train(sentences, total_examples=len(sentences), epochs=30)
print(f"Word2vec similarity: {WVmodel.wv.similarity('boy', 'boys'):.4f}")
print(f"FastText similarity: {FTmodel.wv.similarity('boy', 'boys'):.4f}")
```

```
Word2vec similarity: 0.4560
FastText similarity: 0.8681
```

※Source – https://arxiv.org/abs/1607.04606, **Enriching Word Vectors with Subword Information, Bojanowski et al., 2017**

ReviewMind

Lesson 4

Topic Modeling

ReviewMind

Topic Models I

- A technique to extract topics or concepts based the features from text documents which allows for us to analyze large volumes of text by clustering documents into topics – A unsupervised learning

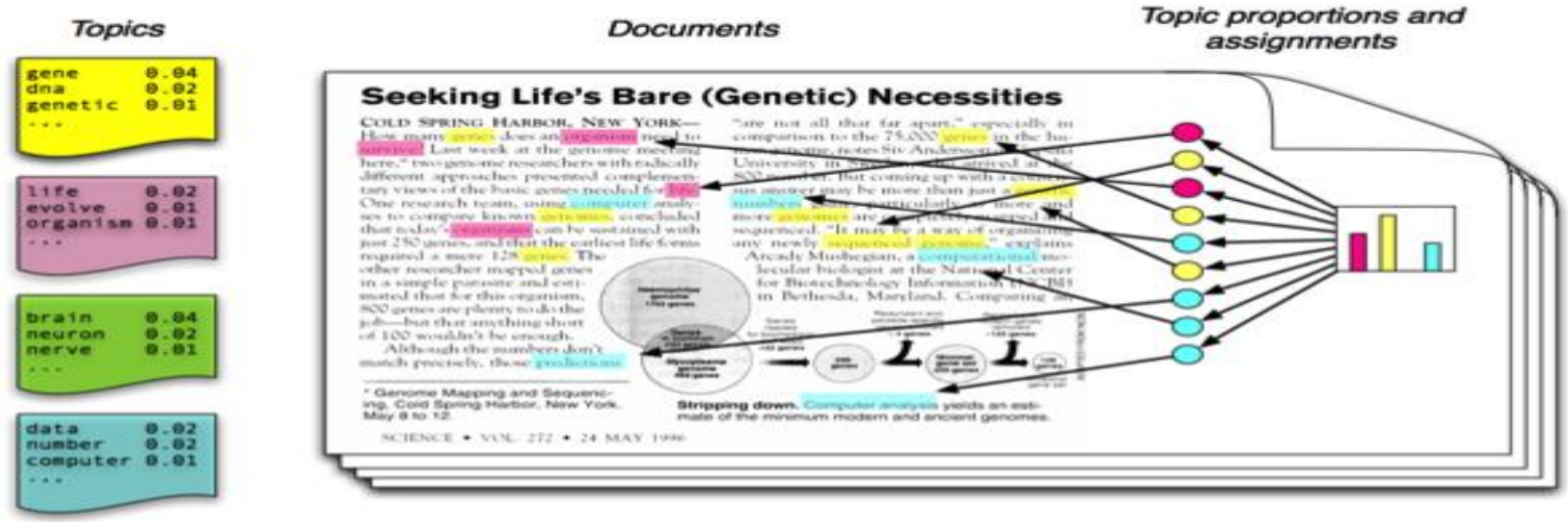


Figure source: Blei, D. M. (2012). Probabilistic topic models. *Communications of the ACM*, *55*(4), 77-84.

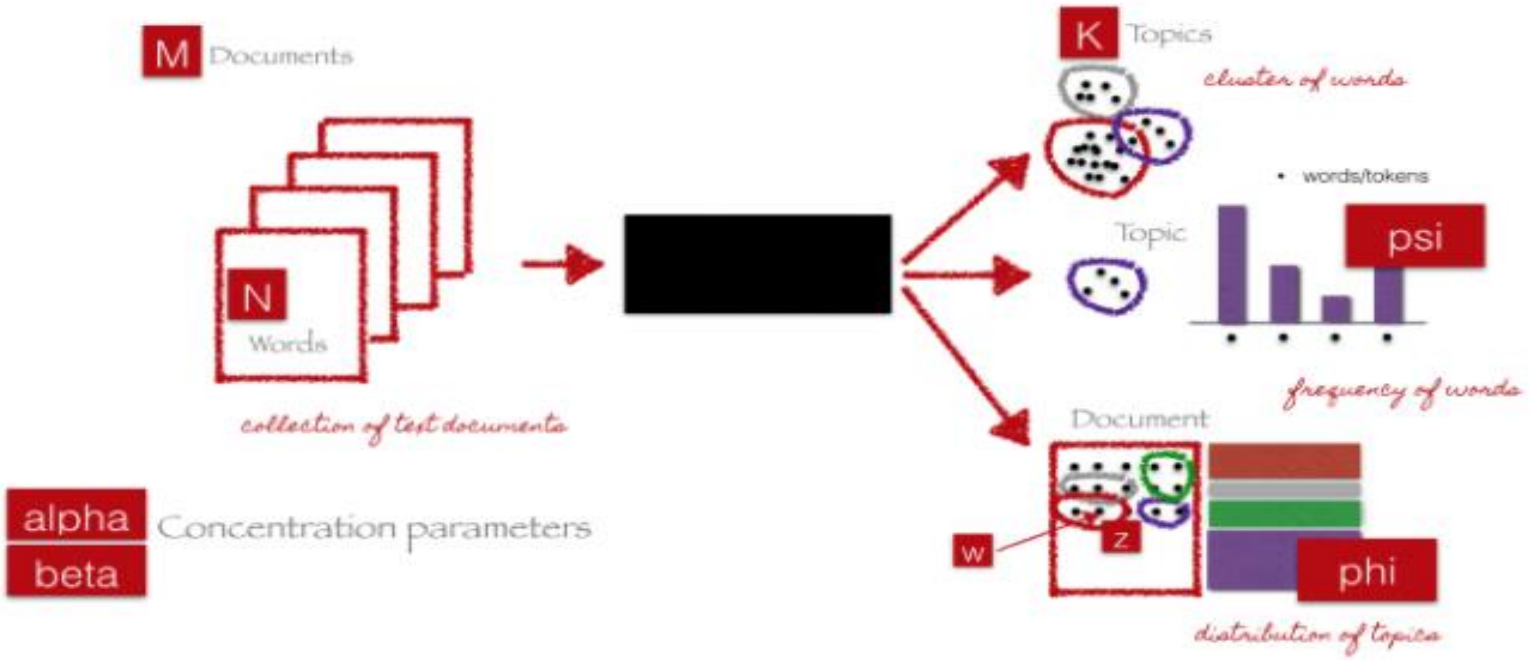
- Each document is a random mixture of corpus-wide topics
- Each word is drawn from one of those topics

※Source – https://towardsdatascience.com/understanding-feature-engineering-part-3-traditional-methods-for-text-data-f6f7d70acd41

ReviewMind

Latent Dirichlet Analysis (LDA)

- LDA is the most commonly using method for topic modeling which is a statistical method with probability distribution



End-to-end LDA framework (courtesy of C. Doig, Introduction to Topic Modeling in Python)

※Source – https://towardsdatascience.com/understanding-feature-engineering-part-3-traditional-methods-for-text-data-f6f7d70acd41

ReviewMind

Latent Dirichlet Analysis (LDA)

- LDA is the most commonly using method for topic modeling which is a statistical method with probability distribution

1. Initialize the necessary parameters.
2. For each document, randomly initialize each word to one of the K topics.
3. Start an iterative process as follows and repeat it several times.
4. For each document D:
   a. For each word W in document:
      - For each topic T:
        - Compute $P(T|D)$, which is proportion of words in D assigned to topic T.
        - Compute $P(W|T)$, which is proportion of assignments to topic T over all documents having the word W.
      - Reassign word W with topic T with probability $P(T|D) \times P(W|T)$ considering all other words and their topic assignments.

※Source – https://towardsdatascience.com/understanding-feature-engineering-part-3-traditional-methods-for-text-data-f6f7d70acd41

ReviewMind

Latent Dirichlet Analysis (LDA)

- Gensim provides different types of built-in LDA functions such as plain LDA, hierarchical LDA, mallot's LDA, etc.

```
# Build LDA model
lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus, id2word=id2word, num_topics=20,
                                            random_state=100, update_every=1, chunksize=100, passes=10,
                                            alpha='auto', per_word_topics=True)
# Print the Keyword in the 10 topics
pprint(lda_model.print_topics())
doc_lda = lda_model[corpus]
```

ReviewMind

Latent Dirichlet Analysis (LDA)

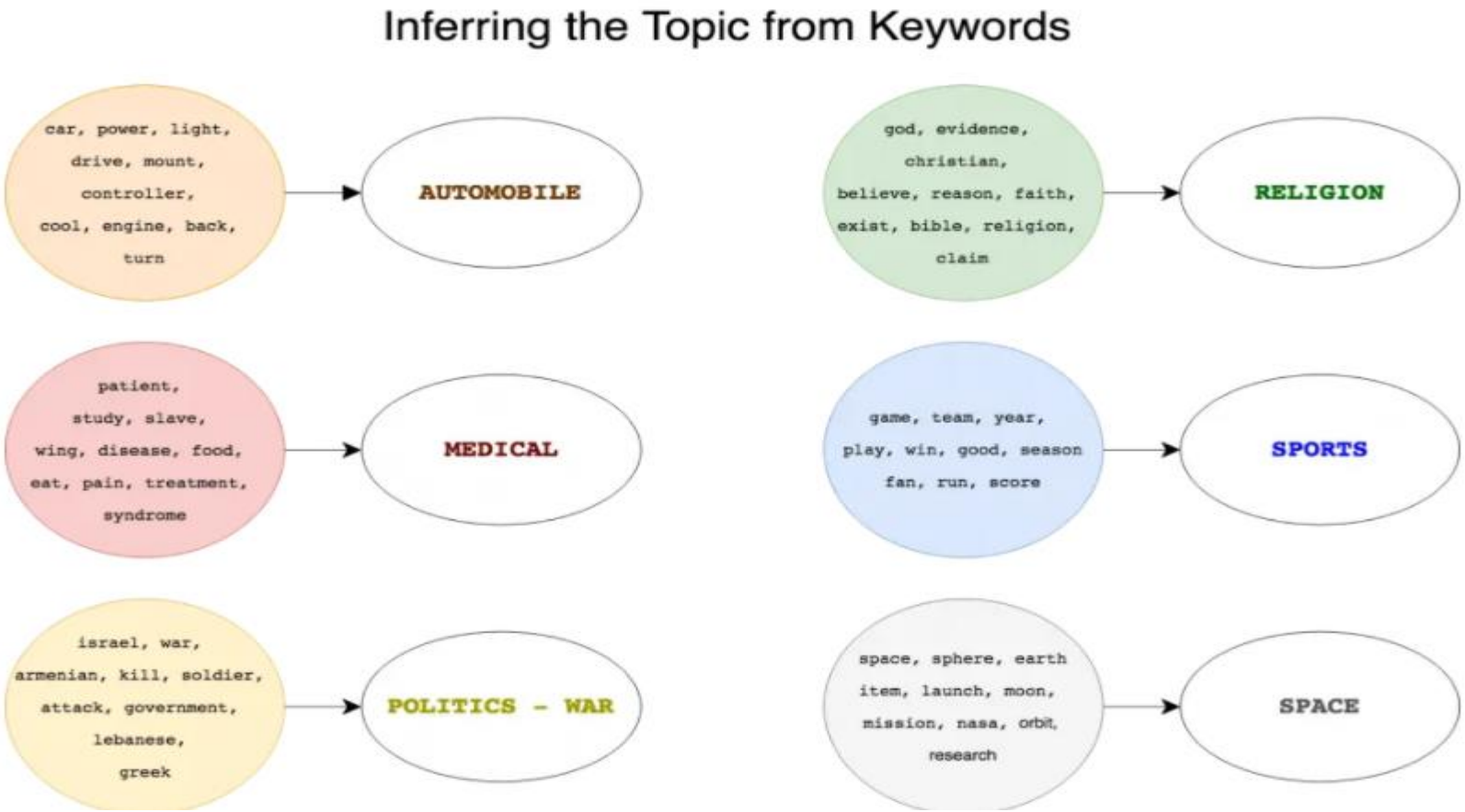
- Gensim provides different types of built-in LDA functions such as plain LDA, hierarchical LDA, mallot's LDA, etc.

```
[(0,
  '0.065*"data" + 0.034*"enable" + 0.011*"microsoft" + 0.003*"efficiently" + '
  '0.000*"textual" + 0.000*"slave" + 0.000*"jumper" + 0.000*"cp_ut" + '
  '0.000*"master_slave" + 0.000*"latch"'),
 (1,
  '0.065*"scsi" + 0.064*"mb" + 0.058*"ide" + 0.054*"headache" + '
  '0.044*"gateway" + 0.029*"water" + 0.028*"oil" + 0.025*"nuclear" + '
  '0.023*"heat" + 0.022*"cylinder"'),
 (2,
  '0.046*"gun" + 0.029*"whole" + 0.024*"bike" + 0.020*"black" + 0.019*"draw" + '
  '0.019*"carry" + 0.017*"white" + 0.015*"police" + 0.015*"ride" + '
  '0.014*"safety"'),
 (3,
  '0.055*"year" + 0.049*"team" + 0.048*"game" + 0.035*"play" + 0.033*"win" + '
  '0.016*"season" + 0.015*"fan" + 0.015*"last" + 0.015*"hit" + 0.014*"first"'),
 (4,
  '0.053*"reality" + 0.049*"distribution_na" + 0.036*"concept" + 0.033*"boy" + '
  '0.030*"poor" + 0.030*"parent" + 0.029*"door" + 0.028*"assumption" + '
  '0.025*"benefit" + 0.022*"blood"'),
 (5,
  '0.039*"number" + 0.032*"list" + 0.023*"include" + 0.023*"copy" + '
```

ReviewMind

Latent Dirichlet Analysis (LDA)

- Gensim provides different types of built-in LDA functions such as plain LDA, hierarchical LDA, mallot's LDA, etc.



※Source – Gensim Topic Modeling - A Guide to Building Best LDA models (machinelearningplus.com)

ReviewMind

Latent Dirichlet Analysis (LDA)

- Gensim provides different types of built-in LDA functions such as plain LDA, hierarchical LDA, mallot's LDA, etc.

```
# Compute Perplexity
print('\nPerplexity: ', lda_model.log_perplexity(corpus))  # a measure of how good the
model is. lower the better.

# Compute Coherence Score
coherence_model_lda = CoherenceModel(model=lda_model, texts=data_lemmatized,
dictionary=id2word, coherence='c_v')
coherence_lda = coherence_model_lda.get_coherence()
print('\nCoherence Score: ', coherence_lda)
```

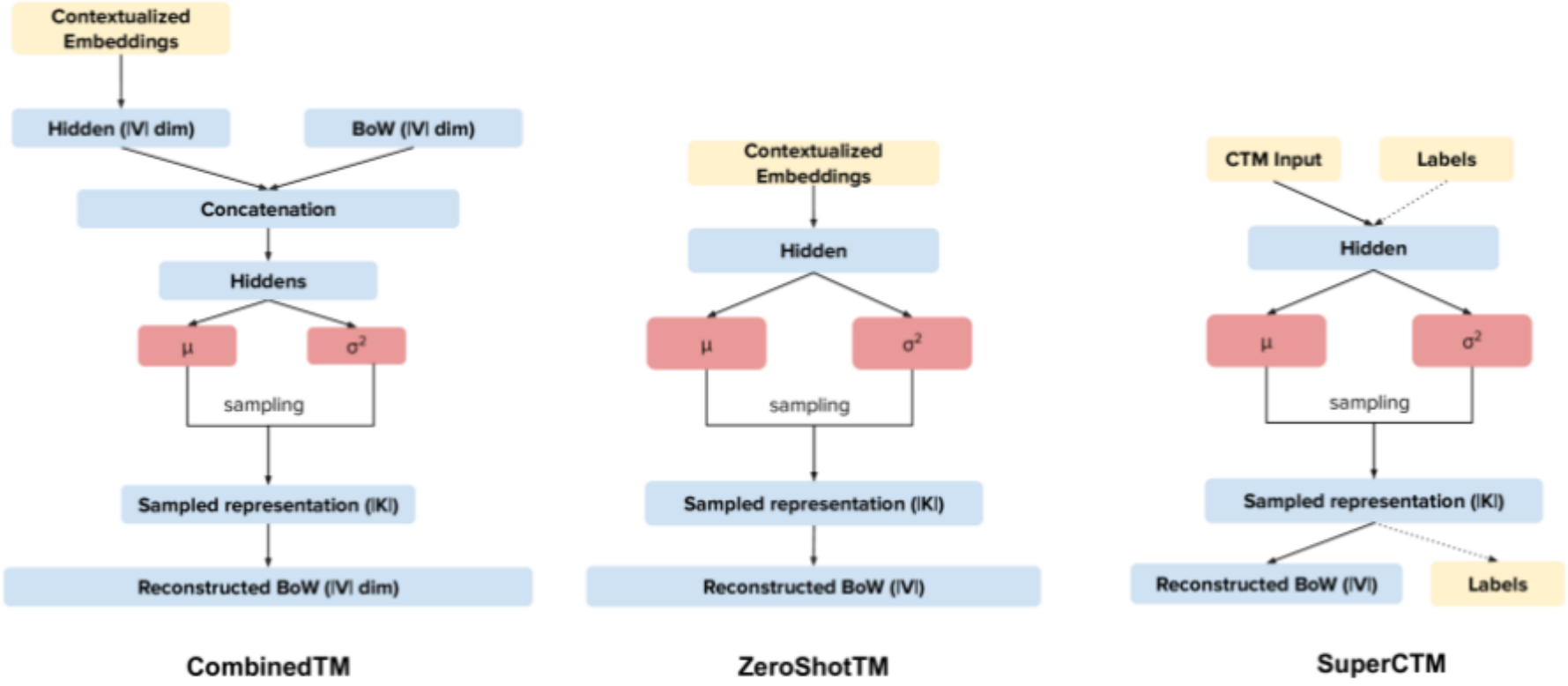
```
Perplexity:  -11.871131132507523

Coherence Score:  0.5066312434890633
```

ReviewMind

Contextualized Topic Models

- Contextualized Topic Models are a family of topic models that use pre-trained representations of language (ex, BERT) to support topic modeling. This approach trains an encoding neural network to map pre-trained contextualized word embeddings to latent representations

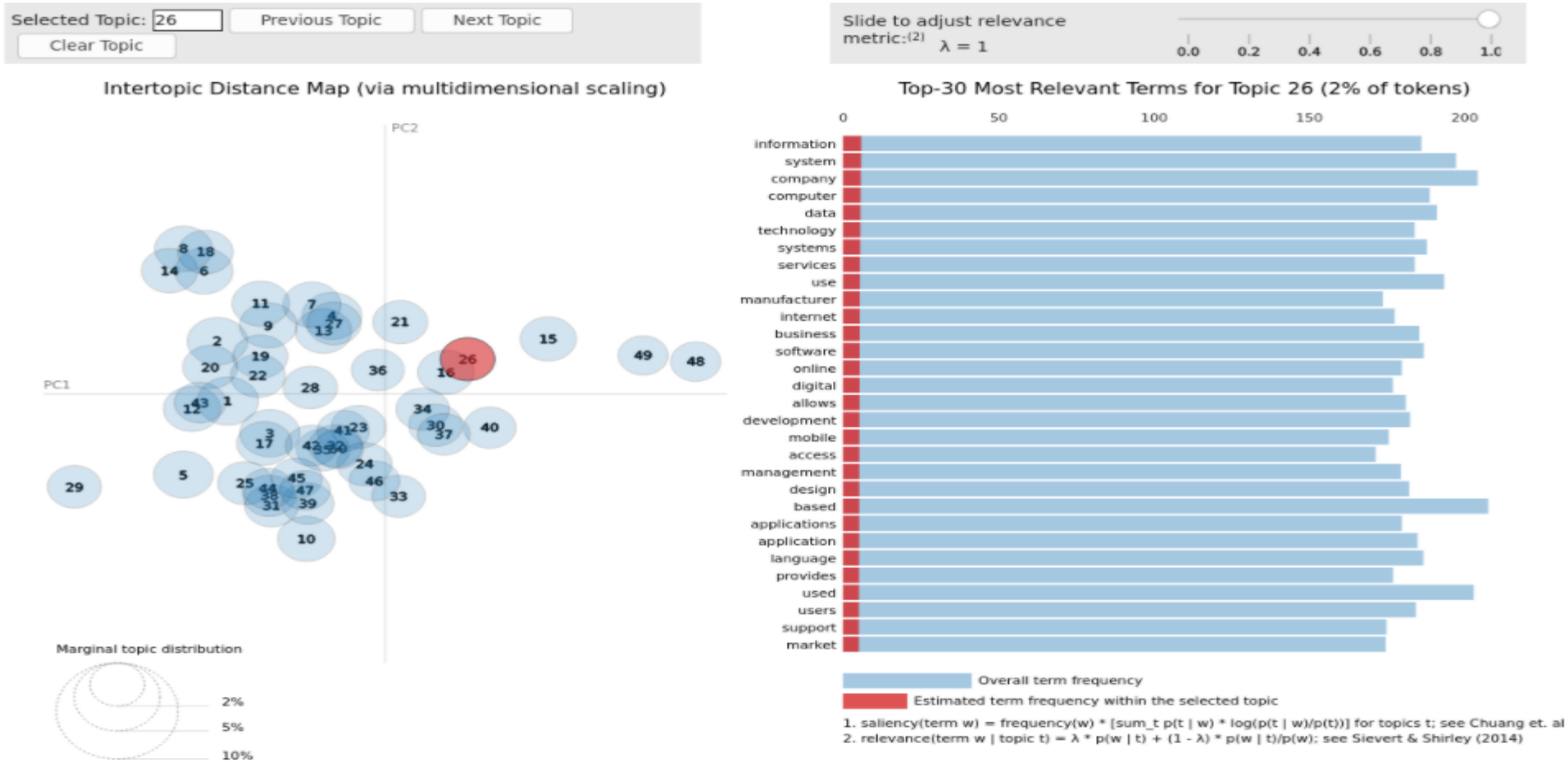


※Source – GitHub - MilaNLProc/contextualized-topic-models: A python package to run contextualized topic modeling. CTMs combine contextualized embeddings (e.g., BERT) with topic models to get coherent topics. Published at EACL and ACL 2021

ReviewMind

Contextualized Topic Models

- It's also support PyLDA visualization like traditional LDA!



※Source – GitHub - MilaNLProc/contextualized-topic-models: A python package to run contextualized topic modeling. CTMs combine contextualized embeddings (e.g., BERT) with topic models to get coherent topics. Published at EACL and ACL 2021

ReviewMind

Thank you!

Email:
sunnykim@reviewmind.com