SAMSUNG

Samsung Innovation Campus

Artificial Intelligence Course

Together for Tomorrow!
Enabling People
Education for Future Generations

Chapter 7.

Natural Language Processing and Language Models for Text Mining

Artificial Intelligence Course

Chapter Description

Chapter objectives

- Process input text from sources of various text formats in order to extract high quality information.
- Structure language and derive patterns by natural language processing. and evaluate and analyze these results to utilize them for real world applications.

Chapter contents

- Unit 1. Text Mining
- Unit 2. Text Preprocessing
- Unit 3. Language Model
- Unit 4. Natural Language Processing with Keras

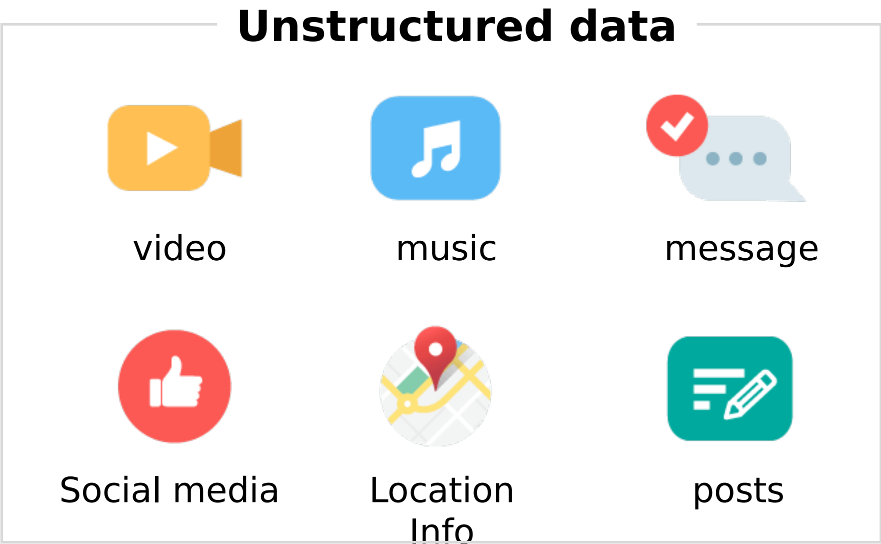
Unit 1.

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 audios that have a large amount of data but with varying structures and forms.
- Books, journals, documents, metadata, health records, audio, video, analog data, images, files, and also e-mail messages, webpages, word-processor documents are all composed of unstructured texts.



1.1. What is Text Mining?

Analyzing unstructured data

- Manually enter tags into metadata to structure texts.
- Skilled data structuring based on text-mining uses a method that creates a tag so that a word in the text and a part in the speech correspond.
- Uses software that builds structures processable by machines.
- Analysis that enables semantic deduction from texts, syntax, and other small or large patterns. This analysis uses algorithms that inspect all internal structure of human communication in word unit by forming them into linguistic, auditory, visual structure.

Finding pattern in unstructured data

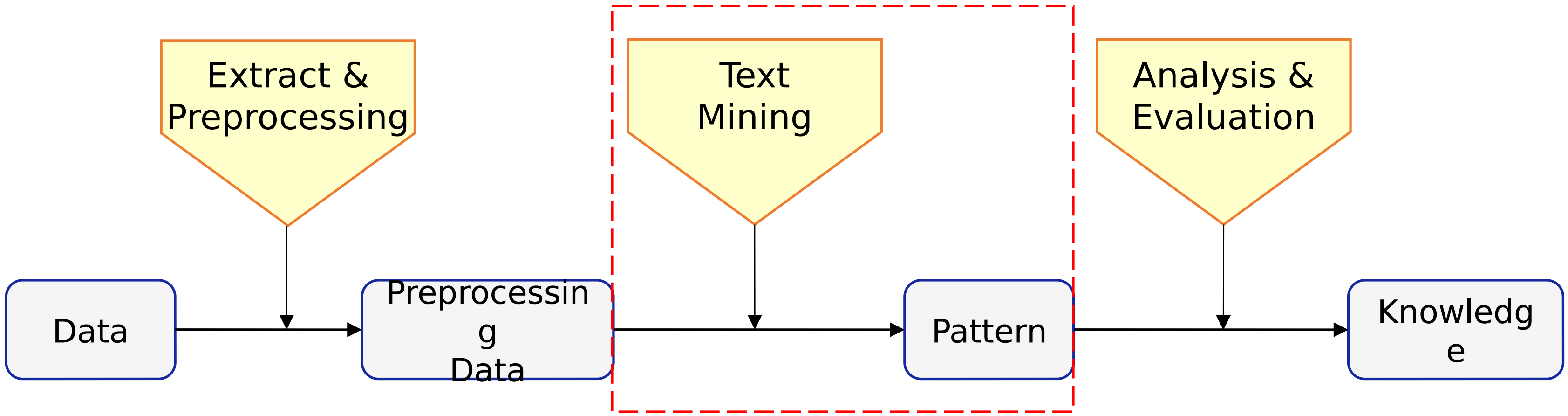
Finding pattern in unstructured data

Use data mining, text analysis, non-standard language analysis

1.1. What is Text Mining?

What is Text Mining??

- Text mining or text analytics is technology that extracts useful information from unstructured text data
- To be more specific, it is finding practical patterns from large amount of document data by applying mechanical algorithm and statistical techniques.



1.1. What is Text Mining?

Text mining vs data mining

- 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

1.1. What is Text Mining?

Text Mining Application Example in Real World

- **Used for marketing:** Corporations collect and analyze posts on twitter that mention their brand names or messages with the customers.
  - They examine if users mention certain topics in certain time; if users have positive or negative connotations; how keywords change within time; if customers' keywords changed before and after a campaign; if a promotion generated a word of mouth; if a certain group of customers react; and etc.
  - With this information, corporations can closely monitor marketing activities and control reputation management and build competitive strategies against competing brands based on feedback monitoring.
- **Supporting data for various industries:** It can support data from fields such as politics, environment and medicine, and business areas such as manufacturing, facilities, and marketing.
- **Factories predicting breakdown**: Factories can predict facilities breakdown from documents recorded by facilities maintenance workers.
- **Checking product reviews**: Customers have access to information of a product's performance or issues from product analysis or reviews.
- **News analysis**: It can show popular issues within time flow and discover experts of a certain field by analyzing topics of news reports or speech statements.

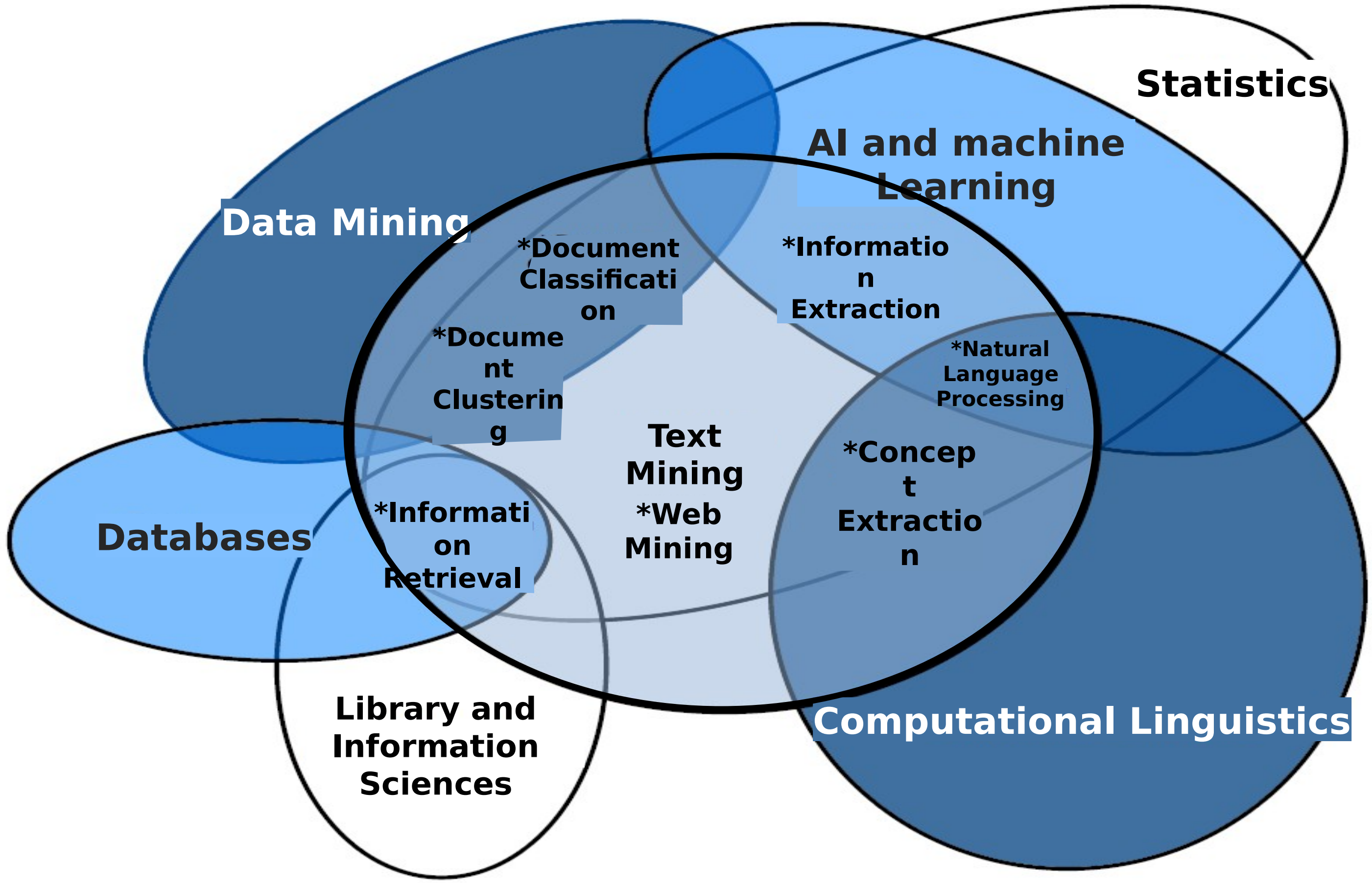
Text mining vs Natural Language Processing (NLP) (1/3)

- **NLP is a field of AI that handles communication.** It enables machines to generate and analyze human language (generate and understand natural language)
  - NLP can process many types of voice including slangs, dialects and grammatical errors. Machine learning constructs the foundation for this methodology. Applications of NPL are search engine, AI chatbots, grammar correction apps etc.
- On the other hand, **text mining is a sub-category of data mining science.** Data mining science includes data search, data mining and machine learning methods.
- More than 80% of organizations worldwide utilize textual information. **NLP recognizes text and voice while text mining evaluates the quality of text.**

Text mining vs Natural Language Processing (NLP) (2/3)

- Different tools are used for text mining and NLP.
  - In order to construct high-quality NLP system, you must be proficient in neural network, deep learning and NLTK.
  - Text mining system is a technique like Levenshtein Distance, Cosine similarity or Feature Hashing.
  - You must be familiar with text processing programming language and statistical models like Perl or Python.
- **NLP offers understanding** of explained emotions and grammatical structure, and detect intent behind a text.
  - This assists fluent translation of text to another language.
- Meanwhile, **text mining discovers relationship** among words in the text.
  - It analyzes frequency of word use and patterns.

Text Mining vs Natural Language Processing (NLP) (3/3)



Text mining algorithms and their topics (1/3)

- Text mining handles topics listed on the right table by applying various algorithms listed on the left table. (1/2)

Algorithm	Area
Naïve bayes	Document classification
Conditional random fields	Information extraction
Hidden Markov models	Information extraction
K-means	Clustering
Singular value decomposition (SVD)	Document classification
Logistic regression	Document classification
Decision trees	Document classification

Topics	Practice Area
Keyword search	Search and information retrieval
Inverted index	Search and information retrieval
Document clustering	Document Clustering
Document similarity	Document Clustering
Feature selection	Document classification
Sentiment analysis	Document classification Web mining
Dimensionality reduction	Document classification
eDiscovery	Document classification

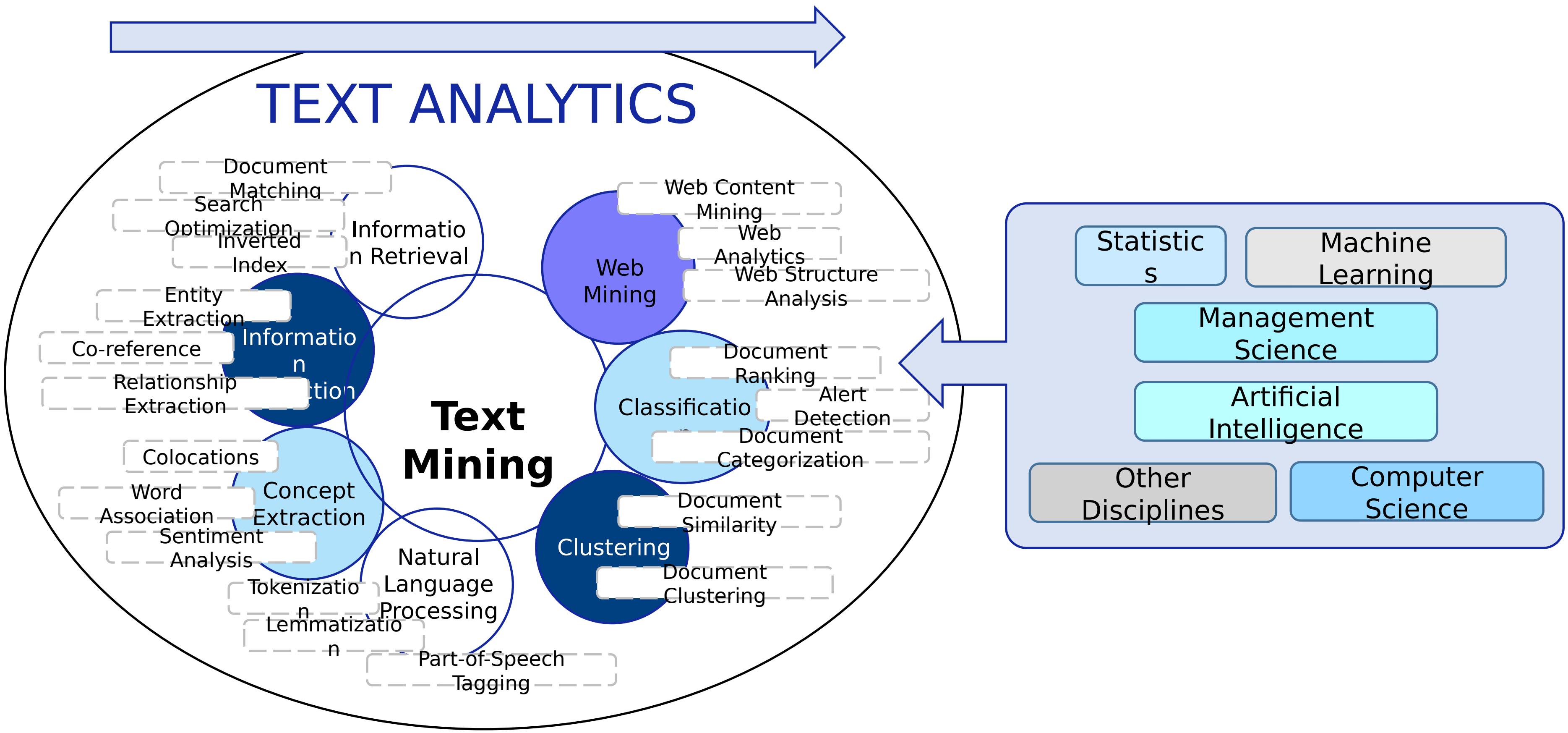
Text mining algorithms and their topics (2/3)

- Text mining handles topics listed on the right table by applying various algorithms listed on the left table.

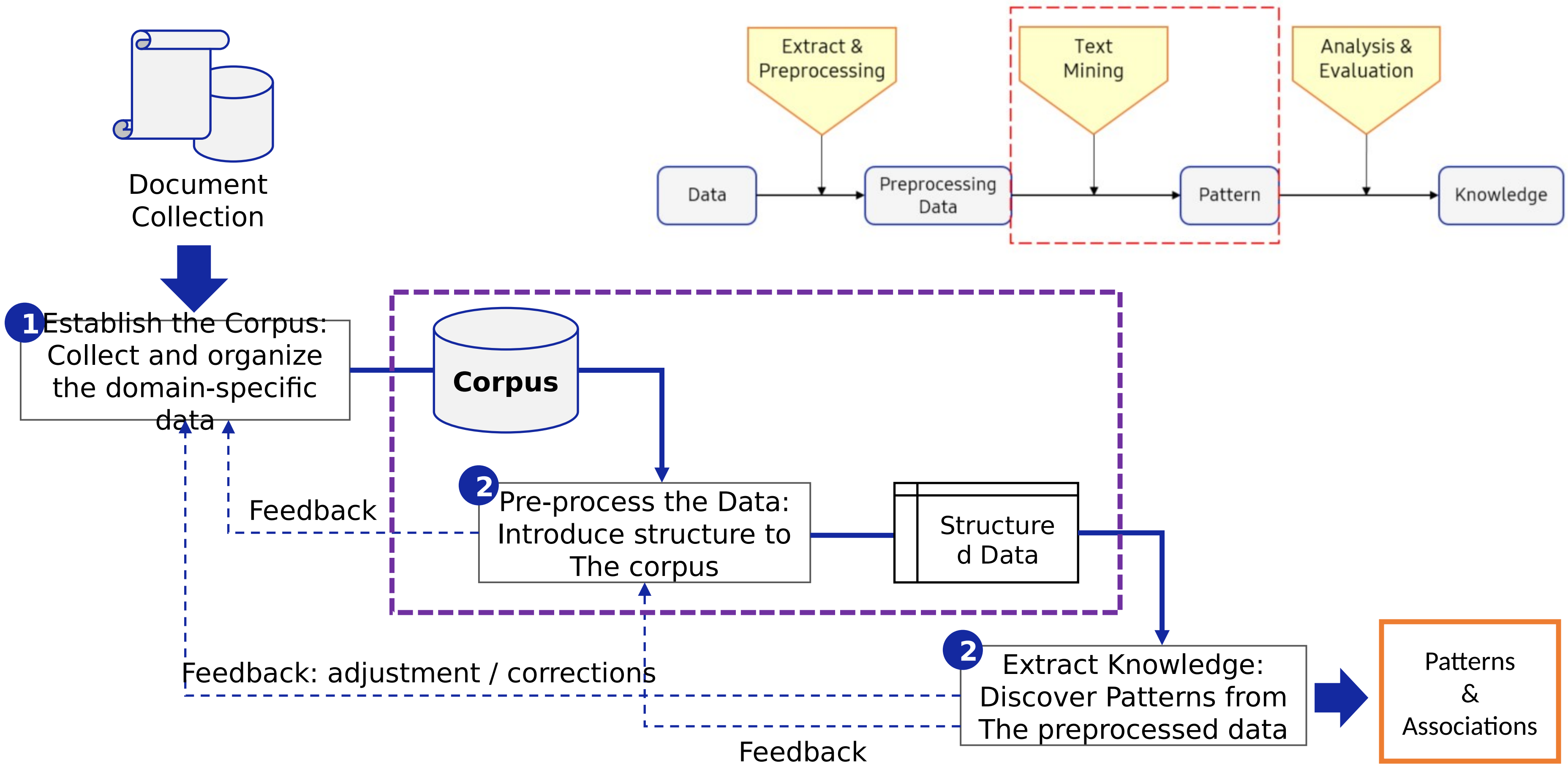
Algorithm	Area
Neural network	Document classification
Support vector machines	Document classification
MARSplines	Document classification
Link analysis	Concept extraction
k-nearest neighbors	Document classification
Word clustering	Concept extraction
Regression	classification

Topic	Practice Area
Web crawling	Web mining
Link analytics	Web mining
Entity extraction	Information extraction
Link extraction	Information extraction
Part of speech tagging	Natural language processing
Tokenization	Natural language processing
Question answering	Natural language processing Search and information retrieval
Topic modeling	Concept extraction
Synonym identification	Concept extraction

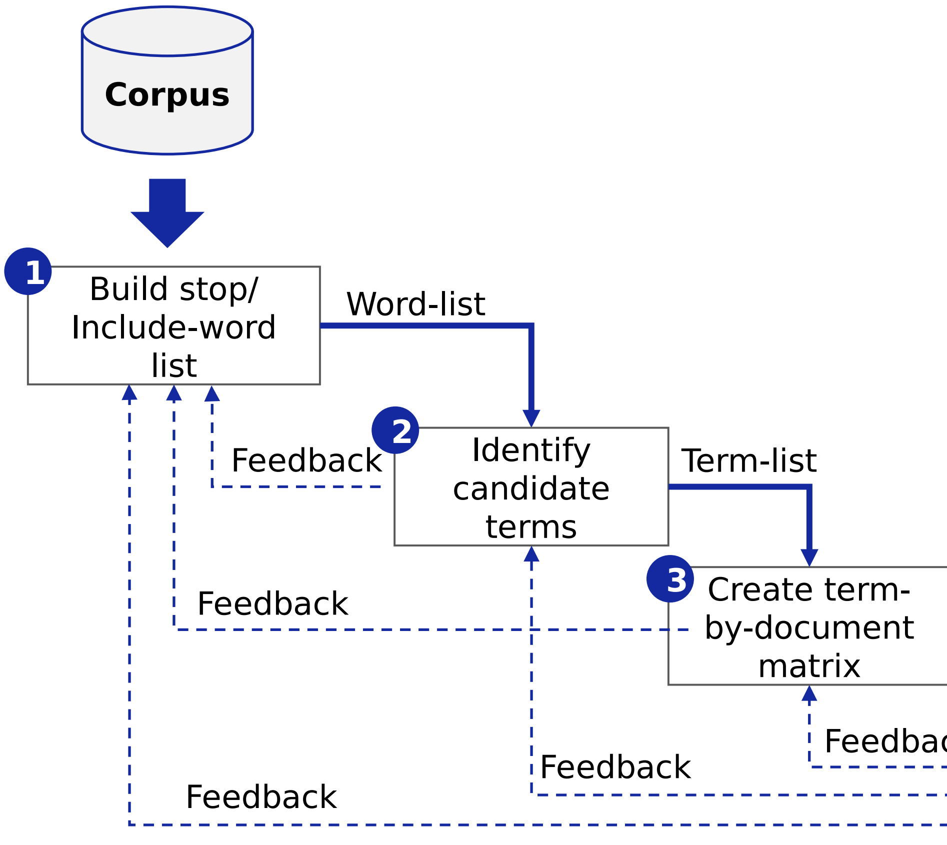
Text mining algorithms and their topics (3/3)

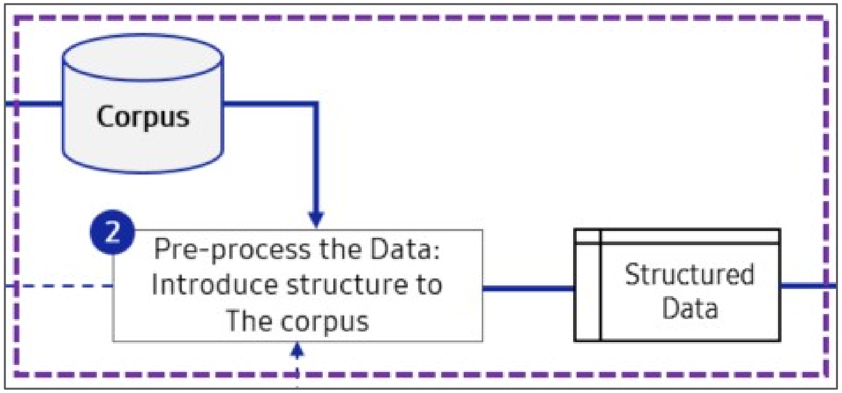


Basic Procedure for Text Mining



Basic Procedure for Text Mining





Unit 1.

Text Mining

Collecting Data from Different Resources

- Collecting text is a process of establishing the plan for collecting data and gathering data suitable for the task's objective and characteristics.
  - Collecting text is an important process that determines the quality of text analysis services.
  - You make a detailed plan after reviewing time period, cost, possibility of personal information infringement, and inclusion of data categories meeting the objective. According to the plan, you perform pre-test and proceed with data collection from different resources.

Collecting Data from Different Resources

Collecting text is a process of establishing the plan for collecting data 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 collect repetitively	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 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 up-to-date web-based information	Content

Text Data from Websites

Download a webpage with the Requests library

- This is OK when the exact URL is known.
- If log-in is required, use the Selenium library instead.
- HTML content without parsing.

Ex

In [1]:

```
1 import requests as rq
2 res = rq.get("https://en.wikipedia.org/wiki/Machine_learning")
3 res.status_code
4 print(res.text)
```

Line 1-3

- If the status code is 200, then OK.

Parsing HTML with the BeautifulSoup4 library:

- The downloaded HTML should be parsed in order to access the desired content.

Ex

In [1]:
```
1 import requests, bs4
2 res = reqquests.get("https://en.wikipedia.org/wiki/Machine_learning")
3 soup = bs4.BeautifulSoup(res.text, 'html.parser')
4 x=soup.find_all('p')
5 text =' '
6 for i in range(len(x)):
7 text += x[i].text.strip() +'\n'
8 print(text)
```

Line 1-3

- Returns a BeautifulSoup object.

Parsing HTML with the BeautifulSoup4 library:

- The downloaded HTML should be parsed in order to access the desired content.

Ex

In [1]:

```
1 import requests, bs4
2 res = reqquests.get("https://en.wikipedia.org/wiki/Machine_learning")
3 soup = bs4.BeautifulSoup(res.text, 'html.parser')
4 x=soup.find_all('p')
5 text =' '
6 for i in range(len(x)):
7 text += x[i].text.strip() +'\n'
8 print(text)
```

Line 1-4

- Get all the paragraphs.

Parsing HTML with the BeautifulSoup4 library:

- The downloaded HTML should be parsed in order to access the desired content.

Ex

In [1]:

```
1 import requests, bs4
2 res = reqquests.get("https://en.wikipedia.org/wiki/Machine_learning")
3 soup = bs4.BeautifulSoup(res.text, 'html.parser')
4 x=soup.find_all('p')
5 text =' '
6 for i in range(len(x)):
7 text += x[i].text.strip() +'\n'
8 print(text)
```

Line 1-7

- Join all the text contents.

Parsing HTML with the BeautifulSoup4 library:

- The downloaded HTML should be parsed in order to access the desired content.

Ex The example from the previous slide produces an output as shown below.

> Machine learning (ML) is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of artificial intelligence. Machine learning algorithms build a mathematical model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task.[1][2]:2 Machine learning algorithms are used in a wide variety of applications, such as email filtering and computer vision, where it is difficult or infeasible to develop a conventional algorithm for effectively performing the task.
>
> Machine learning is closely related to computational statistics, which focuses on making predictions using computers. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a field of study within machine learning, and focuses on exploratory data analysis through unsupervised learning.[3][4] In its application across business problems, machine learning is also referred to as predictive analytics.

Text Data From Twitter

About Twitter:

- Social networking, news and microblogging.
- Users post and interact through short messages known as tweets.
- Used for spreading news and ideas, promotion, building relationship, etc.
- Twitter API allows access to the features of Twitter without having to go through the website.
- Source of text data.

1.2. Data Collection

Reading Tweets from Twitter:

- Apply for access following the steps below:

1) Create a developer account and then log in.

2) Go to https://apps.twitter.com/ and press the Create an app button.

3) Fill out the form and then press the Create button.

4) Click on the Keys and tokens tab.

   The “Consumer API key” and “Consumer API secret key” are already available.

5) In order to generate the “Access token” and “Access token secret”, click on the Create my access token button.

Reading Tweets from Twitter:

Ex Using the Tweepy library to fetch tweets:

In [1]:
```
1  import tweepy
2  from tweepy import OAuthHandler
3
4  my_consumer_key = "----------------------"
5  my_consumer_secret = "----------------------"
6  my_access_token = "----------------------"
7  my_access_secret = "----------------------"
8
9  # Initialize the token.
10 Auth = OAuthHandler(my_consumer_key, my_consumer_secret)
11 Auth.set_access_token(my_access_token, my_access_secret)
12
13 api = tweepy.API(auth, timeout = 10)            # Timeout after 10 seconds.
14 for status in tweepy.Cursor(api.search, q = my_keyword + " -filter:retweets",
15                             lang="en", result_type="recent", geocode=my_location).items(n_tweets):
16 my_tweets.append(status.text)
```

Line 1-4 ~ 1-7

- Replace with your own context

Reading Tweets from Twitter:

Ex A simple output:

```
Eric McCormack backtracks call to 'blacklist' Hollywood Trump donors https://t.co/r64ILsB5Mp
https://t.co/5gCWP9aStI

If you don't understand people's pulse, Certainly politics is not for you.

Trump and Modi both are elected becaus… https://t.co/PQR7L4JJsz

@JohnBerman have you heard Trump's idea on how to get rid of Dorian/hurricanes in general? I hear when he
suggested… https://t.co/Sde1ieROWO

"Dare We Dream of the End of the G.O.P.?" by BY MICHELLE GOLDBERG https://t.co/xf9ggrWEPc

Afghanistan President Ashraf Ghani to visit Washington to meet Donald Trump | 2019-09-06 https://t.co/dNJsyIbLI2

Times of Middle East: Ideal storm: Media pound Trump about ‘Sharpie-gate’ hurricane map
https://t.co/Hwv6p6nwpP @DaPathanGuy @TimesofMEast

Times of Middle East: Trump’s Hurricane Strategy Tops This Week’s Internet Information Ro
https://t.co/QX0yrghNDO @DaPathanGuy @TimesofMEast

Times of Middle East: Google’s Hellish 3 A long time, Trump’s Tariff Hold off, and Add https://t.co/M9xKV2ARIc
@DaPathanGuy @TimesofMEast

Times of Middle East: Trump Aims to Privatize FANNIE and FREDDIE… https://t.co/OuBJp6STQ0 @DaPathanGuy
@TimesofMEast

Trump shows old map with Alabama in Dorian’s forecast path https://t.co/Ihkq7yuax6

Times of Middle East: Trump Administration Expedites Obstacle to California on https://t.co/h03hooqRzQ
@DaPathanGuy @TimesofMEast
```

Unit 1.

Text Mining

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.

Useful functions and methods for string manipulation:

- Used for making string patterns.
- Useful for recognizing and processing complex string patterns   pre-processing of text data.
- More powerful than the combinations of the usual string functions or methods.
- Supported by many languages including Python.

Regular Expression

Metacharacters:

- Characters with special meanings in the 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 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 white space character.
  - → "\S" means non-white space 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 for 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.

Matching group patterns:

- We can group patterns by enclosing 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+)")

Matching group patterns:

- We can group patterns by enclosing with ( ).

Ex “Hide the telephone number”

In [1]:
```
1 import requests as rq
2 res = rq.get("https://en.wikipedia.org/wiki/Machine_learning")
3 res.status_code
4 print(res.text)
```

Line 1-3

- If the status code is 200, then OK.

1.3 String Manipulation

Matching group patterns:

- We can group patterns by enclosing with ( ).

Ex "Hide the telephone number"

In [1]:
```
1 import requests as rq
2 res = rq.get("https://en.wikipedia.org/wiki/Machine_learning")
3 res.status_code
4 print(res.text)
```

Line 1-4

- Not easily intelligible yet.

Matching group patterns:

- We can group patterns by enclosing with ( ).

Ex “Extract only the phone number”

In [2]:
```
my_regex = re.compile("(\D+)((\d+)\D+(\d+)\D+(\d+))")
m = my_regex.search("John 010-1234-5678")
print("Phone number : " + m.group(2))
```

Unit 1.

Text Mining

Natural Language Processing

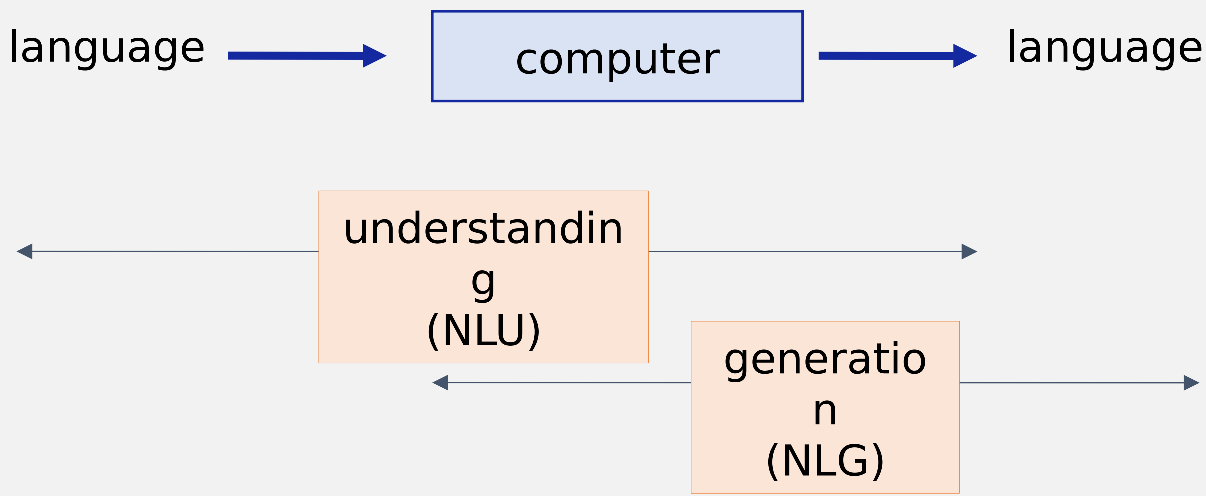
What the Natural Language Processing (NLP) is:

Natural language refers to language people naturally use in their daily lives.
Natural Language Processing (NLP) is an academic field that enables computer to understand and generate natural language.

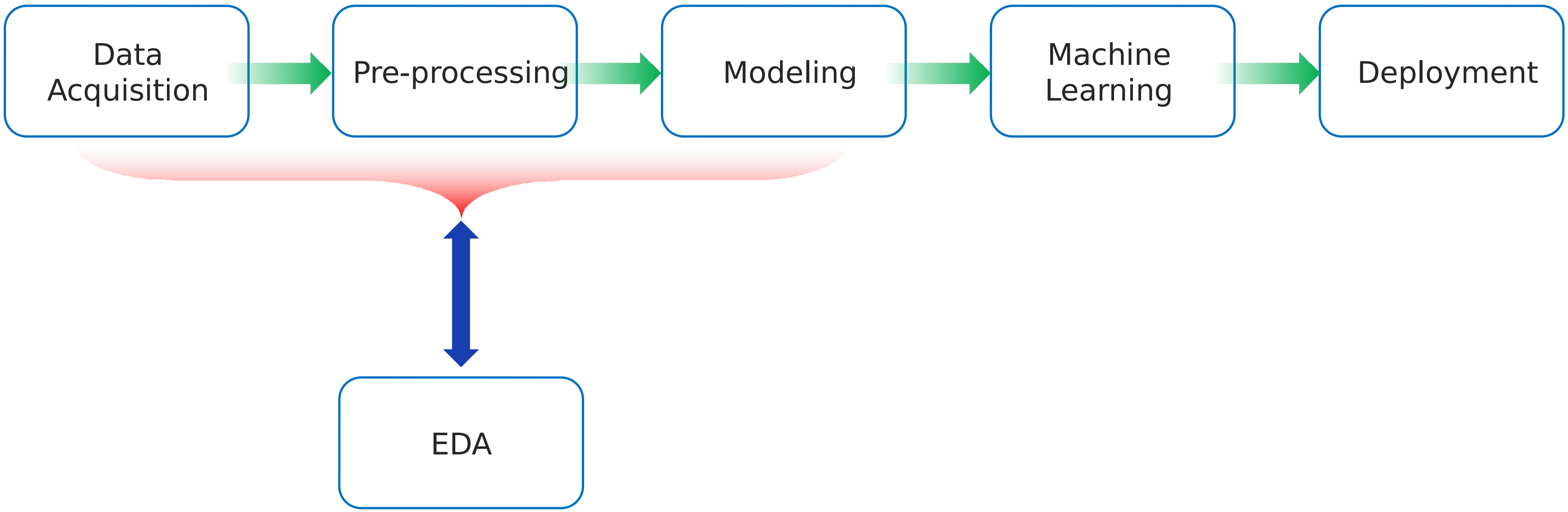
- Extracts features from text data in order to classify, summarize, cluster and do sentiment analysis.
- Intersection of linguistics and AI
- Based on statistical models
- Different from the way humans understand the language
- Requires transformation into a structured model

1.4. Natural Language Processing (NLP)

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



NLLP Workflow:



Difficulties of Natural Language Processing

- NLP is receiving much attention and is widely used, but the procedure is complex.
- The performance of machine translation has improved, but machine translation still often produces awkward translation results.
- NLP is complex because the input data is a not numerical value but human language. The input of human language makes the data processing extremely complex and uncertain.
  - Even the same words have possibility of various interpretations depending on context. This is called linguistic ambiguity.
  - Like idioms that take a different meaning once various words assemble, there is always an exception to how a phrase or words or morphemes construct a sentence.
  - Since language is flexible and open to expansion, modeling language always entails uncertainty. Also, as time goes by, new words are created, and some words become unused.

Research paradigm of NLP: (1) Rule-based approach

- The rule-based approach defines beforehand the grammatical rules of the language, and processes natural language based on those rules.

  - It decides the part of speech (POS) for a given word based on linguistic phenomena rather than statistical methods.
  - In the sentence below, it grasps the meaning of the sentence with the first verb, and figures out the object of the instruction and its subject matter with ‘to’ or ‘that.’

    “Send a message to Michael that I will be late for meeting.”

- The problem with the rule-based system is that it is impossible to establish the rules prior.
- Currently, it is only used within combination of other methods because grammatical rules cannot be entirely neglected in language processing.

Research paradigm of NLP: (2) Statistics-based approach

- The statistics-based approach decides the part of speech (POS) by computing lexical probabilities and contextual probabilities within reference to a huge volume of dictionaries to eliminate ambiguity of POS.
  - Lexical probability is probability that a certain POS applies to a word. This can be expressed as P (POS | word) in mathematical form.
  - Contextual possibility is possibility that a certain POS of a word will show with the POS of the next word. This, mathematically is P (POS | POS)
  - Then it labels the POS that produces highest result of the multiplication of linguistic probability and contextual probability as the most appropriate for words of semantic ambiguity
- There has been much progress as computers analyze sentences much faster with improvement in performance, but human intervention is still necessary.
- Such issues are being tackled by deep learning techniques nowadays.

1.4. Natural Language Processing (NLP)

Research paradigm of NLP: (3) Deep Learning-based approach

- While statistical analysis is based on peripheral analysis such as frequency of word appearance, Deep Learning approach enables in-depth analysis based on composite connection among data.
  - NLP models that understand sentence or overall context of sentence could have been created after deep learning based NLP was initiated.
  - If you create an artificial neural network component that connects with all parts of a sentence, this variable, after learning, contains information of the whole sentence. The accuracy is constantly increasing.

Coding Exercise #0501~0508

Follow practice steps on 'ex_0501~8.ipynb' file

Unit 1.

Text Mining

Sequential Data

Data of temporal property is widely used around the world. (1/2)

- For example, there are stock prices, human voice, or ECG signals.
- These data have sequences. You must utilize this feature and temporal information in order to obtain high performance from sequential data.

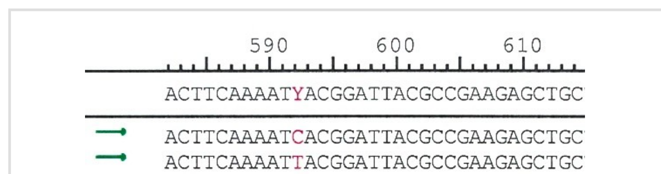
(a) ECG signal

(b) Stock price

(c) Vocal signal

Climbing down did I see, the flower I couldn’t see on the way up

(d) sentence



(e) DNA sequence

1.5. Sequential Data

Data of temporal property is widely used around the world. (2/2)

- Temporal data is dynamic because they change overtime, and it generally has variable length.
  - Recurrent neural network, which you will study later, is a learning model that effectively processes such temporal data.
- Lately, it can process extremely long patterns that occur in daily lives. For example, machine translator can now translate sentences of more than 30 words, while only 10 words was the maximum in the past.
- To translate a long sentence, it is needed to understand context between two words that are far apart. This is called long-term dependency.
  - Since standard RNN cannot fully process long-term dependency, LSTM, which supplemented selective memory function to standard RNN, is widely used. Selective memory is the capacity to discern memory for long-term and short-term.

Unit 1.

Text Mining

Corpus

- Corpus:
  - Refers to a set of text data subject to analysis.

    a) Raw corpus: text data stored in a data base.

    b) Tagged corpus: text data where words and phrases have been labeled according to a model.

Unit 2.

Text Preprocessing

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 the 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 your hope, *everything's* possible."

    How do we tokenize "*don't*" and "*everything's*"? It depends on the tokenizer.

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']
```

Unit 2.

Text Preprocessing

Stop Words

Stop words and keywords:

- Stop words are commonly used words that do not contain semantic information:
  - Ex Definite and indefinite articles: “the”, “a”, “an”.
  - Ex Prepositions: “on”, “with”, “into”, “upon”, etc.
- Stop words are removed during the “data normalization” process.
- Keywords are selected among the available words after removing the stop words.
  - Often the most frequent words can be selected as keywords.
  - Sometimes we should also take into account the purpose and context.

2.2. Stop Words

Stop words and keywords:

- In the NLTK library, there are 179 English stop words:

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

Stop words and keywords:

In [9]:
```
from nltk.corpus import stopwords
stopwords.words('english')[:10]
```

Out[9]:
```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]
```

In [11]:
```
# NLTK defines words such as 'i', 'me', 'my' as stop words
```

In [10]:
```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

example = "Family is not an important thing. It's everything."
stop_words = set(stopwords.words('english'))

word_tokens = word_tokenize(example)

result = []
for w in word_tokens:
    if w not in stop_words:
        result.append(w)

print(word_tokens)
print(result)
```

```
['Family', 'is', 'not', 'an', 'important', 'thing', '.', 'It', "'s", 'everything', '.']
['Family', 'important', 'thing', '.', 'It', "'s", 'everything', '.']
```

Unit 2.

Text Preprocessing

Lemmatization and Stemming

Lemmatization :

```
In [12]: from nltk.stem import WordNetLemmatizer
         n=WordNetLemmatizer()
         words=['policy', 'doing', 'organization', 'have', 'going', 'love', 'lives', 'fly', 'dies', 'watched', 'has', 'starting']
         print([n.lemmatize(w) for w in words])

         ['policy', 'doing', 'organization', 'have', 'going', 'love', 'life', 'fly', 'dy', 'watched', 'ha', 'starting']

In [13]: n.lemmatize('dies', 'v')
Out[13]: 'die'

In [14]: n.lemmatize('watched', 'v')
Out[14]: 'watch'

In [15]: n.lemmatize('has', 'v')
Out[15]: 'have'
```

Stemming:

In [16]:
```python
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
s = PorterStemmer()
text="This was not the map we found in Billy Bones's chest, but an accurate copy, complete in all things--names and heights
words=word_tokenize(text)
print(words)
```

```
['This', 'was', 'not', 'the', 'map', 'we', 'found', 'in', 'Billy', 'Bones', "'s", 'chest', ',', 'but', 'an', 'accurate',
'copy', ',', 'complete', 'in', 'all', 'things', '--', 'names', 'and', 'heights', 'and', 'soundings', '--', 'with', 'the',
'single', 'exception', 'of', 'the', 'red', 'crosses', 'and', 'the', 'written', 'notes', '.']
```

In [17]:
```python
print([s.stem(w) for w in words])
```

```
['thi', 'wa', 'not', 'the', 'map', 'we', 'found', 'in', 'billi', 'bone', "'s", 'chest', ',', 'but', 'an', 'accur', 'copi',
',', 'complet', 'in', 'all', 'thing', '--', 'name', 'and', 'height', 'and', 'sound', '--', 'with', 'the', 'singl', 'excep
t', 'of', 'the', 'red', 'cross', 'and', 'the', 'written', 'note', '.']
```

Stemming:

```
In [18]: #Stemmization of porter algorithm has rules as below.
         #ALIZE → AL
         #ANCE →Delete
         #ICAL → IC
```

```
In [19]: words=['formalize', 'allowance', 'electricical']
         print([s.stem(w) for w in words])

         ['formal', 'allow', 'electric']
```

```
In [20]: from nltk.stem import PorterStemmer
         s=PorterStemmer()
         words=['policy', 'doing', 'organization', 'have', 'going', 'love', 'lives', 'fly', 'dies', 'watched', 'has', 'starting']
         print([s.stem(w) for w in words])

         ['polici', 'do', 'organ', 'have', 'go', 'love', 'live', 'fli', 'die', 'watch', 'ha', 'start']
```

```
In [21]: from nltk.stem import LancasterStemmer
         l=LancasterStemmer()
         words=['policy', 'doing', 'organization', 'have', 'going', 'love', 'lives', 'fly', 'dies', 'watched', 'has', 'starting']
         print([l.stem(w) for w in words])

         ['policy', 'doing', 'org', 'hav', 'going', 'lov', 'liv', 'fly', 'die', 'watch', 'has', 'start']
```

Unit 2.

Text Preprocessing

POS Tagging

Part of Speech (POS) tag set in English:

Tag Set	Description	Example
CC	Coordinating conjunction	
CD	Cardinal digit	
DT	Determiner	
EX	Existential ‘there’	there is
FW	Foreign word	
IN	Preposition/subordinating conjunction	
JJ	Adjective	Big
JJR	Adjective, comparative	Bigger
JJS	Adjective, superlative	Biggest
LS	List marker	1)
MD	Modal	could, will
NN	Noun, singular	Desk
NNS	Noun, plural	Desks
NNP	Proper noun, singular	Harrison
NNPS	Proper noun, plural	Americans
PDT	Predeterminer	‘all’ the kids
POS	Possessive ending	parent's

Part of Speech (POS) tag set in English:

Tag Set	Description	Example
PRP	Personal pronoun	I, he, she
PRP$	Possessive pronoun	my, his, hers
RB	Adverb	very, silently
RBR	Adverb, comparative	better
RBS	Adverb, superlative	best
RP	Particle	give up
TO	To	go 'to' the store
UH	Interjection	errrrrrrrm
VB	Verb, base form	take
VBD	Verb, past tense	took
VBG	Verb, gerund/present participle	taking
VBN	Verb, past participle	taken
VBP	Verb, sing. Present, non-3d	take
VBZ	Verb, 3rd person sing. Present	takes
WDT	Wh-determiner	which
WP	Wh-pronoun	who, what
WP$	Possessive wh-pronoun	whose
WRB	Wh-abverb	where, when

Part of Speech (POS) tag set in English:

```
In [22]: # Test sentence.
         my_sentence = "The Colosseum was built by the emperor Vespassian"

In [24]: import nltk

In [25]: # Simple pre-processing.
         my_words = nltk.word_tokenize(my_sentence)
         for i in range(len(my_words)):
             my_words[i] = my_words[i].lower()
         my_words

Out[25]: ['the', 'colosseum', 'was', 'built', 'by', 'the', 'emperor', 'vespassian']

In [26]: # POS tagging.
         # OUTPUT: A list of tuples.
         my_words_tagged = nltk.pos_tag(my_words)
         my_words_tagged

Out[26]: [('the', 'DT'),
          ('colosseum', 'NN'),
          ('was', 'VBD'),
          ('built', 'VBN'),
          ('by', 'IN'),
          ('the', 'DT'),
          ('emperor', 'NN'),
          ('vespassian', 'NN')]
```

NLTK Library

NLTK (Natural Language Toolkit):

- One of the most used Python libraries for the NLP.

  a) Tokenization: sent_tokenize, word_tokenize, etc.

  b) Stop words: 179 in total. The list of stop words needs to be downloaded.

  ```
  nltk.download('stopwords')
  ```

  c) Stemming: PorterStemmer, LancasterStemmer, SnowballStemmer, etc.

  d) Lemmatization: WordNetLemmatizer.

  e) POS tagging: Uses the “Penn Treebank tag set”.

  f) Lexical resource: WordNet, SentiWordNet, etc. Useful for sentiment analysis.

Visualization

- Word Cloud:
  - Visualization of the keywords where the size is related to the frequency.

Word Cloud:

- We can create a word cloud following the steps below:

  1) Tokenize by words.

  2) Apply cleaning and normalization.

  3) Make a frequency table of the words.

  4) Sort by frequency and keep only the top keywords. The number of keywords is adjustable.

  5) Customize the arguments of the WordCloud() function.

Unit 2.

Text Preprocessing

Integer Encoding

Exercise

```
In [1]: import nltk
        nltk.download('punkt')

[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\emcast\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!

Out[1]: True

In [2]: from nltk.tokenize import sent_tokenize
        from nltk.tokenize import word_tokenize
        from nltk.corpus import stopwords

In [3]: text = "A barber is a person. a barber is a good person. a barber is a huge person. he Knew A Secret!. The Secret He Kept i

In [4]: text = sent_tokenize(text)
        print(text)

['A barber is a person.', 'a barber is a good person.', 'a barber is a huge person.', 'he Knew A Secret!.', 'The Secret He Kept is huge secret.', 'Huge secret.', 'His barber kept his word.', 'a barber kept his word.', 'His barber kept his secret.', 'But keeping and keeping such a huge secret to himself was driving the barber crazy.', 'the barber went up a huge mountain.']
```

Integer Encoding

In [4]:
```python
# Cleaning and word tokenization
vocab = {} # Python's dictionary datatype
sentences = []
stop_words = set(stopwords.words('english'))

for i in text:
    sentence = word_tokenize(i) # executes word tokenization
    result = []

    for word in sentence:
        word = word.lower() # Reduce the number of words by lowercasing all words
        if word not in stop_words: # Remove stop words in case of word tokenization.
            if len(word) > 2: # Remove additional words if the length of the word is lower or equal to 2
                result.append(word)
                if word not in vocab:
                    vocab[word] = 0
                vocab[word] += 1
    sentences.append(result)
print(sentences)
```

```
[['barber', 'person'], ['barber', 'good', 'person'], ['barber', 'huge', 'person'], ['knew', 'secret'], ['secret', 'kept',
'huge', 'secret'], ['huge', 'secret'], ['barber', 'kept', 'word'], ['barber', 'kept', 'word'], ['barber', 'kept', 'secre
t'], ['keeping', 'keeping', 'huge', 'secret', 'driving', 'barber', 'crazy'], ['barber', 'went', 'huge', 'mountain']]
```

In [5]:
```python
print(vocab)
```

```
{'barber': 8, 'person': 3, 'good': 1, 'huge': 5, 'knew': 1, 'secret': 6, 'kept': 4, 'word': 2, 'keeping': 2, 'driving': 1,
'crazy': 1, 'went': 1, 'mountain': 1}
```

In [6]:
```python
print(vocab["barber"])
```

```
8
```

Align in the order of frequency

```
In [7]: vocab_sorted = sorted(vocab.items(), key = lambda x:x[1], reverse = True)
        print(vocab_sorted)

[('barber', 8), ('secret', 6), ('huge', 5), ('kept', 4), ('person', 3), ('word', 2), ('keeping', 2), ('good', 1), ('knew', 1), ('driving', 1), ('crazy', 1), ('went', 1), ('mountain', 1)]
```

Assign low integer index for words of high frequency

```
In [8]: word_to_index = {}
        i=0
        for (word, frequency) in vocab_sorted :
            if frequency > 1 : # Low frequency words are deleted. (It was learned in the Cleaning section.)
                i=i+1
                word_to_index[word] = i
        print(word_to_index)

        {'barber': 1, 'secret': 2, 'huge': 3, 'kept': 4, 'person': 5, 'word': 6, 'keeping': 7}
```

Use top 5 words

In [9]:
```
vocab_size = 5
words_frequency = [w for w,c in word_to_index.items() if c >= vocab_size + 1] #Remove words of index higher than 5
for w in words_frequency:
    del word_to_index[w] #Remove index information from the corresponding word
print(word_to_index)
```

```
{'barber': 1, 'secret': 2, 'huge': 3, 'kept': 4, 'person': 5}
```

Changing each word from sentences stored after tokenization to an integer by using word_to_index

```
In [11]: word_to_index['OOV'] = len(word_to_index) + 1
```

```
In [12]: encoded = []
         for s in sentences:
             temp = []
             for w in s:
                 try:
                     temp.append(word_to_index[w])
                 except KeyError:
                     temp.append(word_to_index['OOV'])
             encoded.append(temp)
         print(encoded)
```

```
[[1, 5], [1, 6, 5], [1, 3, 5], [6, 2], [2, 4, 3, 2], [3, 2], [1, 4, 6], [1, 4, 6], [1, 4, 2], [6, 6, 3, 2, 6, 1, 6], [1,
6, 3, 6]]
```

Using counter

```
In [13]: from collections import Counter
```

```
In [14]: print(sentences)
```

```
[['barber', 'person'], ['barber', 'good', 'person'], ['barber', 'huge', 'person'], ['knew', 'secret'], ['secret', 'kept', 'huge', 'secret'], ['huge', 'secret'], ['barber', 'kept', 'word'], ['barber', 'kept', 'word'], ['barber', 'kept', 'secret'], ['keeping', 'keeping', 'huge', 'secret', 'driving', 'barber', 'crazy'], ['barber', 'went', 'huge', 'mountain']]
```

```
In [15]: words = sum(sentences, [])
# Above task can be executed by words=np.hstack(sentences)
print(words)
```

```
['barber', 'person', 'barber', 'good', 'person', 'barber', 'huge', 'person', 'knew', 'secret', 'secret', 'kept', 'huge', 'secret', 'huge', 'secret', 'barber', 'kept', 'word', 'barber', 'kept', 'word', 'barber', 'kept', 'secret', 'keeping', 'keeping', 'huge', 'secret', 'driving', 'barber', 'crazy', 'barber', 'went', 'huge', 'mountain']
```

```
In [16]: vocab = Counter(words) #You can easily count all frequency of the words by using Python's Counter module.
print(vocab)
```

```
Counter({'barber': 8, 'secret': 6, 'huge': 5, 'kept': 4, 'person': 3, 'word': 2, 'keeping': 2, 'good': 1, 'knew': 1, 'driving': 1, 'crazy': 1, 'went': 1, 'mountain': 1})
```

```
In [17]: print(vocab["barber"]) #Print the frequency of the word 'barber'
```

```
8
```

Assign low integer index for words of high frequency

```
In [18]: vocab_size = 5
         vocab = vocab.most_common(vocab_size)# Save top 5 words with the highest frequency
         vocab

Out[18]: [('barber', 8), ('secret', 6), ('huge', 5), ('kept', 4), ('person', 3)]

In [19]: word_to_index = {}
         i = 0
         for (word, frequency) in vocab :
             i = i+1
             word_to_index[word] = i
         print(word_to_index)

         {'barber': 1, 'secret': 2, 'huge': 3, 'kept': 4, 'person': 5}
```

Use NLTK’s FreqDist

In [20]:
```
from nltk import FreqDist
import numpy as np
```

In [21]:
```
# Remove sentence section with np.hstack and use it as input. Ex) 'barber', 'person', 'barber', 'good'
vocab = FreqDist(np.hstack(sentences))
```

In [22]:
```
print(vocab["barber"])
```

```
8
```

In [23]:
```
vocab_size = 5
vocab = vocab.most_common(vocab_size) # Save top 5 words with the highest frequency
vocab
```

Out[23]:
```
[('barber', 8), ('secret', 6), ('huge', 5), ('kept', 4), ('person', 3)]
```

In [24]:
```
word_to_index = {word[0] : index + 1 for index, word in enumerate(vocab)}
print(word_to_index)
```

```
{'barber': 1, 'secret': 2, 'huge': 3, 'kept': 4, 'person': 5}
```

2.5. Integer Encoding

Use Enumerate

In [25]:
```
test=['a', 'b', 'c', 'd', 'e']
for index, value in enumerate(test):# Assign index starting from 0 in the order of input
  print("value : {}, index: {}".format(value, index))
```

```
value : a, index: 0
value : b, index: 1
value : c, index: 2
value : d, index: 3
value : e, index: 4
```

Use Keras

In [27]:
```
from tensorflow.keras.preprocessing.text import Tokenizer
```

In [28]:
```
sentences=[['barber', 'person'], ['barber', 'good', 'person'], ['barber', 'huge', 'person'], ['knew', 'secret'], ['secret',
```

In [29]:
```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(sentences)
# fit_on_texts(), with corpus as input,  generates vocabulary based on word frequency
```

In [30]:
```
print(tokenizer.word_index)
```

```
{'barber': 1, 'secret': 2, 'huge': 3, 'kept': 4, 'person': 5, 'word': 6, 'keeping': 7, 'good': 8, 'knew': 9, 'driving': 1
0, 'crazy': 11, 'went': 12, 'mountain': 13}
```

In [34]:
```
print(tokenizer.word_counts)
```

```
OrderedDict([('barber', 8), ('person', 3), ('good', 1), ('huge', 5), ('knew', 1), ('secret', 6), ('kept', 4), ('word', 2),
('keeping', 2), ('driving', 1), ('crazy', 1), ('went', 1), ('mountain', 1)])
```

In [39]:
```
print(tokenizer.texts_to_sequences(sentences))
```

```
[[1, 5], [1, 5], [1, 3, 5], [2], [2, 4, 3, 2], [3, 2], [1, 4], [1, 4], [1, 4, 2], [3, 2, 1], [1, 3]]
```

Coding Exercise

Follow practice steps on 'Integer Encoding.ipynb' file

Unit 2.

Text Preprocessing

Padding

Padding using NumPy

In [1]:
```
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
```

In [2]:
```
sentences = [['barber', 'person'], ['barber', 'good', 'person'], ['barber', 'huge', 'person'],
             ['knew', 'secret'], ['secret', 'kept', 'huge', 'secret'], ['huge', 'secret'],
             ['barber', 'kept', 'word'], ['barber', 'kept', 'word'], ['barber', 'kept', 'secret'],
             ['keeping', 'keeping', 'huge', 'secret', 'driving', 'barber', 'crazy'],
             ['barber', 'went', 'huge', 'mountain']]
```

In [3]:
```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(sentences) # fit_on_texts(), with corpus as input, generates vocabulary based on word freque
```

In [4]:
```
encoded = tokenizer.texts_to_sequences(sentences)
print(encoded)
```

```
[[1, 5], [1, 8, 5], [1, 3, 5], [9, 2], [2, 4, 3, 2], [3, 2], [1, 4, 6], [1, 4, 6], [1, 4, 2], [7, 7, 3, 2, 10, 1, 11], [1,
12, 3, 13]]
```

In [5]:
```
max_len = max(len(item) for item in encoded)
print(max_len)
```

```
7
```

Padding using NumPy

In [6]:
```
for item in encoded: # For each sentence
    while len(item) < max_len: # If smaller than max_len
        item.append(0)

padded_np = np.array(encoded)
padded_np
```

Out[6]:
```
array([[ 1,  5,  0,  0,  0,  0,  0],
       [ 1,  8,  5,  0,  0,  0,  0],
       [ 1,  3,  5,  0,  0,  0,  0],
       [ 9,  2,  0,  0,  0,  0,  0],
       [ 2,  4,  3,  2,  0,  0,  0],
       [ 3,  2,  0,  0,  0,  0,  0],
       [ 1,  4,  6,  0,  0,  0,  0],
       [ 1,  4,  6,  0,  0,  0,  0],
       [ 1,  4,  2,  0,  0,  0,  0],
       [ 7,  7,  3,  2, 10,  1, 11],
       [ 1, 12,  3, 13,  0,  0,  0]])
```

2.6. Padding

Padding using Keras preprocessing tool

In [8]:
```
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

In [10]:
```
encoded = tokenizer.texts_to_sequences(sentences)
print(encoded)
```

```
[[1, 5], [1, 8, 5], [1, 3, 5], [9, 2], [2, 4, 3, 2], [3, 2], [1, 4, 6], [1, 4, 6], [1, 4, 2], [7, 7, 3, 2, 10, 1, 11], [1,
12, 3, 13]]
```

In [11]:
```
padded = pad_sequences(encoded)
padded
```

Out[11]:
```
array([[ 0,  0,  0,  0,  0,  1,  5],
       [ 0,  0,  0,  0,  1,  8,  5],
       [ 0,  0,  0,  0,  1,  3,  5],
       [ 0,  0,  0,  0,  0,  9,  2],
       [ 0,  0,  0,  2,  4,  3,  2],
       [ 0,  0,  0,  0,  0,  3,  2],
       [ 0,  0,  0,  0,  1,  4,  6],
       [ 0,  0,  0,  0,  1,  4,  6],
       [ 0,  0,  0,  0,  1,  4,  2],
       [ 7,  7,  3,  2, 10,  1, 11],
       [ 0,  0,  0,  1, 12,  3, 13]])
```

Coding Exercise

Follow practice steps on 'Padding.ipynb' file

Unit 2.

Text Preprocessing

One-Hot Encoding

One-hot encoding using Keras preprocessing tool

```
In [1]: text='I want to go to lunch with me. The lunch menu is hamburgers. Hamburgers are the best'
```

```
In [2]: from tensorflow.keras.preprocessing.text import Tokenizer
        from tensorflow.keras.utils import to_categorical
```

```
In [3]: t = Tokenizer()
        t.fit_on_texts([text])
        print(t.word_index) # Print encoding result for each word

{'to': 1, 'lunch': 2, 'the': 3, 'hamburgers': 4, 'i': 5, 'want': 6, 'go': 7, 'with': 8, 'me': 9, 'menu': 10, 'is': 11, 'ar
e': 12, 'best': 13}
```

```
In [4]: encoded=t.texts_to_sequences([text])
        print(encoded)

[[5, 6, 1, 7, 1, 2, 8, 9, 3, 2, 10, 11, 4, 4, 12, 3, 13]]
```

One-hot encoding using Keras preprocessing tool

In [5]:
```
one_hot = to_categorical(encoded)
print(one_hot)
```

```
[[[0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
  [0. 1. 0. 0. 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. 1. 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. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
  [0. 0. 0. 1. 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. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
  [0. 0. 0. 0. 1. 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. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
  [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]]]
```

Coding Exercise

Follow practice steps on 'One-hot Encoding.ipynb' file

Unit 3.

Language Model

Language Model

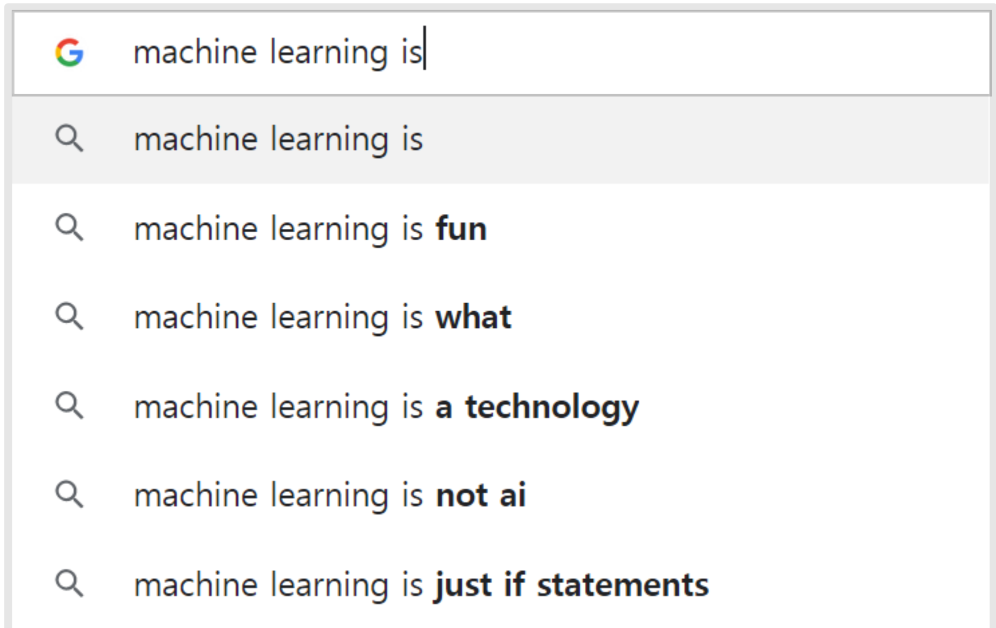
- Language model refers to model that predict or generate the next component by assigning probability to elements of language (letter, word, morpheme, string (sentence), paragraph etc.).
  - Language model is divided into statistical language model (SLM) and deep learning language model based on artificial neural network. Essentially language models, based on a given word, predicts the next word or combination of words, and such function can solve numerous natural language processing problems such as 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?
  $P(〖w\_i \mid w〗\_1, w\_2, w\_3, \ldots, w\_(i-1))$?
- 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:

Ex P (three little pigs lived happily)?

“”	“little”	“pigs”	“lived”	“happily”

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”

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

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

→ , 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:

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

$$P(w_i \vee w¿¿1, w_2, w_3, ..., w_{i-1}) = \frac{Count(w_1, w_2, w_3, ..., w_i)}{Count(w_1, w_2, w_3, ..., w_{i-1})}¿$$

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

which can be compared with the following exact relation.

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

3.1. Language Model

n-Gram approximations:

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

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

$$P\left(w\text{¿¿}1, w_2, w_3, \ldots, w_m\right) \approx P\left(w_1\right) P\left(w_2 \vee w_1\right) P\left(w_3 \vee w_2\right) \cdots P\left(w_m \vee w_{m-1}\right)\text{¿}$$

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

$$P\left(w\text{¿¿}1, w_2, w_3, \ldots, w_m\right) \approx P\left(w_1\right) P\left(w_2 \vee w_1\right) P\left(w_3 \vee w_2, w_1\right) P\left(w_4 \vee w_3, w_2\right) \cdots P\left(w_m \vee w_{m-1}, w_{m-2}\right)\text{¿}$$

Ex Bigram approximation for **Sequence** = “”

Coding Exercise #0509

Follow practice steps on 'ex_0509.ipynb' file

Unit 3.

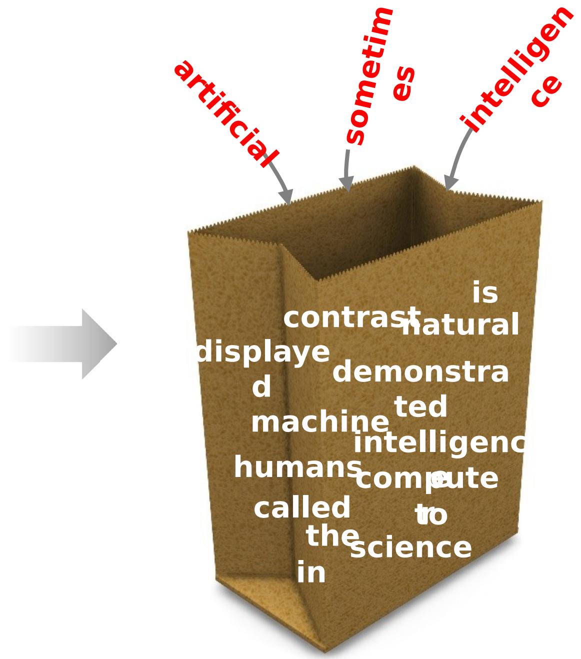
Language Model

Representation Model

Bag-of-Words (BOW) model:

- A document is represented by a collections of its words.
- Word ordering and grammar are ignored.
- Only the word frequencies matter.

Ex "In computer science, artificial intelligence, sometimes called machine intelligence, is intelligence demonstrated by machines, in contrast to the natural intelligence displayed by humans."



Bag-of-Words (BOW) model:

Ex “It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness.”

after removing the stop words such as “it”, “the”, and “of”, the BOW can be expressed as an

age	best	foolish-ness	times	was	wisdom	worst
2	1	1	2	1	1	1

3.2. Representation Model

Document-Term Matrix (DTM) and Term-Document Matrix (TDM):

- Documents expressed as BOW are the rows of DTM.
- Documents expressed as BOW are the columns of TDM.

	Feature #1	Feature #2	Feature #3	Feature #4	…
Document #1	1	0	1	0	0
Document #2	0	0	2	0	0
Document #3	0	1	0	0	1
Document #4	0	0	0	1	0
⋮	0	0	1	0	0

DTM

	Document #1	Document #2	Document #3	Document #4	…
Feature #1	1	0	0	0	0
Feature #2	0	0	1	0	0
Feature #3	1	2	0	0	1
Feature #4	0	0	0	1	0
⋮	0	0	1	0	0

TDM

Document-Term Matrix (DTM) and Term-Document Matrix (TDM):

Ex Let's suppose the following pre-processed documents.

Document #1: "learning intelligence machine learning statistics"

Document #1: "machine classification learning performance"

Document #1: "machine classification, machine learning, machine performance"

DTM =

	learning	intelligence	machine	statistics	classification	performance
Doc. #1	2	1	1	1	0	0
Doc. #2	1	0	1	0	1	1
Doc. #3	1	0	3	0	1	1

Document-Term Matrix (DTM) and Term-Document Matrix (TDM):

Ex Let’s suppose the following pre-processed documents.

Document #1: “learning intelligence machine learning statistics”

Document #1: “machine classification learning performance”

Document #1: “machine classification, machine learning, machine performance”

TDM =

	Doc. #1	Doc. #2	Doc. #3
learning	2	1	1
intelligence	1	0	0
machine	1	1	3
statistics	1	0	0
classification	0	1	1
performance	0	1	1

Term Frequency (TF):

- Indicates the relative importance of each word (term) within a document.
- A frequently occurring word within a short document would have a large TF value.

$$TF(word, document) = \frac{Frequency\ of\ the\ word\ within\ the\ document}{The\ document\ length}$$

- TF has to be calculated per word and per document.

Team Frequency (TF):

Ex Let's suppose the following pre-processed documents.

Document #1: "learning intelligence machine learning statistics"
length = 5

Document #1: "machine classification learning performance"
length = 4

Document #1: "machine classification, machine learning, machine performance"
length = 6

TF =

	Doc. #1	Doc. #2	Doc. #3
learning	2/5 = 0.4	1/4 = 0.25	1/6 = 0.17
intelligence	1/5 = 0.2	0	0
machine	1/5 = 0.2	1/4 = 0.25	3/6 = 0.5
statistics	1/5 = 0.2	0	0
classification	0	1/4 = 0.25	1/6 = 0.17
performance	0	1/4 = 0.25	1/6 = 0.17

Document Frequency (DF) and Inverse Document Frequency (IDF):

- DF: the number of documents where a particular word appears.
- IDF: a measure of rarity and information carried by a particular word.

$$IDF(word) = Log\left(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ that\ include\ the\ word}\right)$$

- IDF is a property of the corpus and has to be calculate per word only.

Document Frequency (DF) and Inverse Document Frequency (IDF):

Ex Let's suppose the following pre-processed documents.

Document #1: "learning intelligence machine learning statistics"

Document #1: "machine classification learning performance"

Document #1: "machine classification, machine learning, machine performance"

IDF =

	DF	IDF
learning	3	Log(3/3) = 0
intelligence	1	Log(3/1) = 0.48
machine	3	Log(3/3) = 0
statistics	1	Log(3/1) = 0.48
classification	2	Log(3/2) = 0.18
performance	2	Log(3/2) = 0.18

TF IDF representation:

Ex Let's suppose the following pre-processed documents.

Document #1: "learning intelligence machine learning statistics"

Document #1: "machine classification learning performance"

Document #1: "machine classification, machine learning, machine performance"

TF IDF =

	Doc. #1	Doc. #2	Doc. #3
learning	0.4	0.25	0.17
intelligence	0.2	0	0
machine	0.2	0.25	0.5
statistics	0.2	0	0
classification	0	0.25	0.17
performance	0	0.25	0.17

	IDF
learning	0
intelligence	0.48
machine	0
statistics	0.48
classification	0.18
performance	0.18

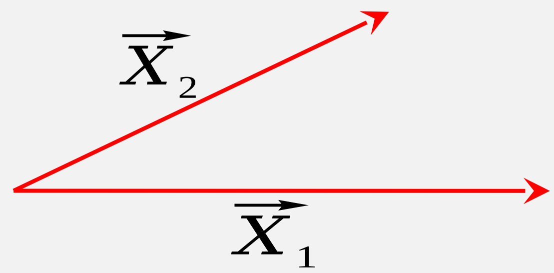
=

	Doc. #1	Doc. #2	Doc. #3
learning	0	0	0
intelligence	0.095	0	0
machine	0	0	0
statistics	0.095	0	0
classification	0	0.044	0.03
performance	0	0.044	0.03

3.2. Representation Model

Cosine similarity:

- Documents are vectors. The similarity between two documents can be quantified.



$$\text{Cosine similarity is } Cos(\theta) = \frac{\vec{X}_1 \cdot \vec{X}_2}{|\vec{X}_1||\vec{X}_2|}$$

Coding Exercise #0510

Follow practice steps on 'ex_0510.ipynb' file

Unit 3.

Language Model

Naïve Bayes Classifier

Naïve Bayes classifier using the BOW model:

- For convenience, let's suppose that there are two document types A and B. The document types can be, for example, A= "spam" and B = "no spam".
- Let's apply the BOW model: the bags A and B contain the tokenized words.
- Applying the Bayes' theorem, we have:

Caution: The sub-index of w serves only a labeling purpose. Here, the words are not ordered.

Naïve Bayes classifier using the BOW model:

- Prediction based on the comparison between $P(A|w_1,w_2,w_3,\ldots)$ and $P(B|w_1,w_2,w_3,\ldots)$.
- For the comparison, only the relative difference matters. Question: Which probability is higher?
- For the comparison, we do not need the common denominator $P(w_1,w_2,w_3,\ldots)$.

- In the BOW model, the words occur independently from each other.
  Thus, we can expand in the following way.

Naïve Bayes classifier using the BOW model:

- Instead of comparing the probabilities, we can compare the logarithms of probabilities.
- Applying the $Log()$ on both sides of the equal sign, we have:

- If we balance the training set such that the number of type A = number of type B, then. So, we can also drop these terms in the comparison.

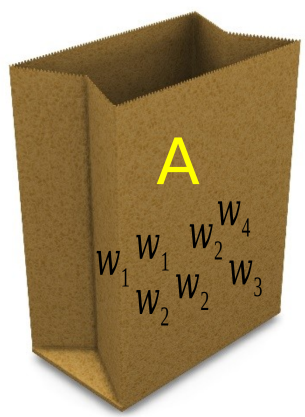
Naïve Bayes classifier using the BOW model:

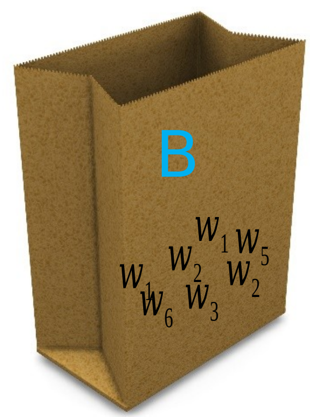
- Training step:

  1) For each word in the bag A, calculate the probabilities $P(w\_i|A)$ and their logarithms $Log(P(w\_i|A))$.

  2) For each word in the bag B, calculate the probabilities $P(w\_i|B)$ and their logarithms $Log(P(w\_i|B))$.

  3) Save for later use the logarithmic probabilities calculated in the steps 1) and 2).





Naïve Bayes classifier using the BOW model:

- Prediction step:

1) Given a test document made up of words w'_1, w'_2, w'_3, … add their logarithmic probabilities:

2) If $LogProbA > LogProbB$: then the test document is predicted as type A.
If $LogProbA < LogProbB$: then the test document is predicted as type B.

Classification Analysis

Classification analysis using the TF IDF model:

- In the TF IDF model:

  → document observation.

  → word () explanatory variable ().

- If the data is labeled (response), we can do predictive analysis with the classification algorithms such as logistic regression, KNN, decision tree, Random Forest, etc.

Obs. #1					
Obs. #2					
Obs. #3					

Doc. #1					
Doc. #2					
Doc. #3					

Coding Exercise #0511

Follow practice steps on 'ex_0511.ipynb' file

Unit 3.

Language Model

Latent Semantic Analysis (LSA)

About the LSA:

- Extracts common topics from a set of documents.

With the TF IDF matrix, carry out the SVD and extract the principal components.

- Let's suppose that a TF IDF matrix $\boldsymbol{M}$ has the following size:

$Size(\boldsymbol{M}) = m \times n$

m= Number of documents.

n= Number of features.

- Each "topic vector" is a principal component of $\boldsymbol{M}$.
- By sorting the singular values, we can extract the most salient topics.

LSA can be used for:

- Clustering of documents.
- Studying the relationship between the documents.
- Labeling of documents for a search engine.

Singular value decomposition (SVD):

- A matrix $\boldsymbol{M}$ is decomposed as $\boldsymbol{M}=\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\boldsymbol{t}}$.

  where,

  $Size(\boldsymbol{M})=m\times n$

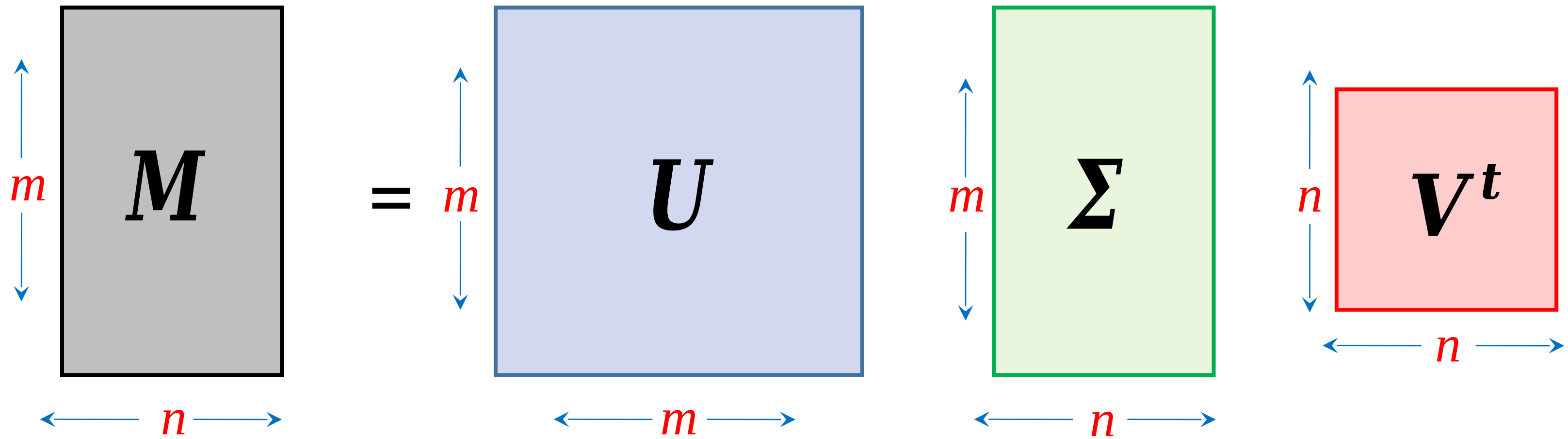
  $Size(\boldsymbol{U})=m\times m$

  $Size(\boldsymbol{\Sigma})=m\times n$

  $Size(\boldsymbol{V})=n\times n$

Singular Value Decomposition (SVD):

- A matrix M is decomposed as $M=U\Sigma V^t$.



Singular value decomposition (SVD):

- A matrix $\boldsymbol{M}$ is decomposed as $\boldsymbol{M}=\boldsymbol{U}$ 〖$\boldsymbol{\Sigma V}$〗 ^$\boldsymbol{t}$.
- Here, $\boldsymbol{\Sigma}$ contains the singular values as diagonal elements.
- These singular values are ordered from the largest to the smallest: σ_1>σ_2>…>σ_m

$$\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 & 0 \\ 0 & \sigma_2 & & 0 & 0 \\ \vdots & & \ddots & \vdots & \\ 0 & 0 & \cdots & \sigma_m & 0 \end{bmatrix}$$

Singular value decomposition (SVD):

- A matrix $\boldsymbol{M}$ is decomposed as $\boldsymbol{M}=\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{t}$.
- The columns of $\boldsymbol{U}$ are the “left singular vectors”.
- The columns of $\boldsymbol{V}$ are the “right singular vectors”.

- => Between a set of the left and right singular vectors and their singular value we have:

$$\boldsymbol{M}\boldsymbol{v}_{\boldsymbol{i}}=\sigma_{i}\boldsymbol{u}_{\boldsymbol{i}}$$

- => Between any two singular vectors, we have the following orthogonality condition:

Truncated SVD and topic vector:

- If r = number of topics, a dimension is reduced as following:

  $Size(\boldsymbol{M}) = m \times n$

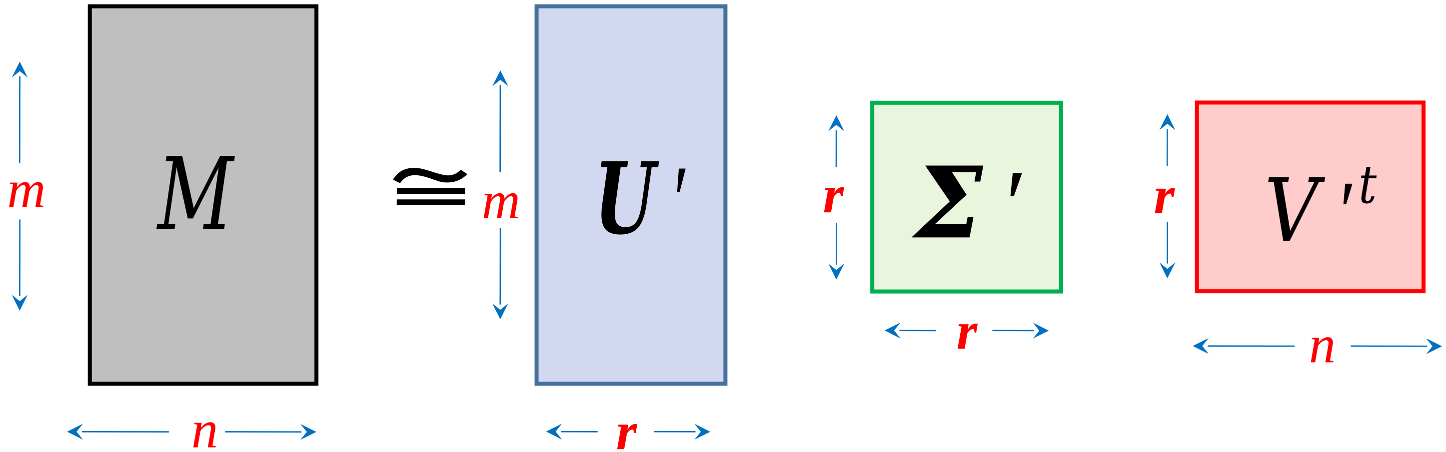
  $Size(\boldsymbol{U}) = m \times m$ → reduced to $m \times r$.

  $Size(\boldsymbol{\Sigma}) = m \times n$ → reduced to $r \times r$.

  $Size(\boldsymbol{V})\ r = n \times n$ → reduced to $n \times r$.

- We do not need all the possible topics. We only need a few “important” topics.

- Truncated SVD and topic vector:
  - If r = number of topics, the dimensional reduction as following is carried out:



  - The columns of V' are the topic vectors.

Coding Exercise #0512

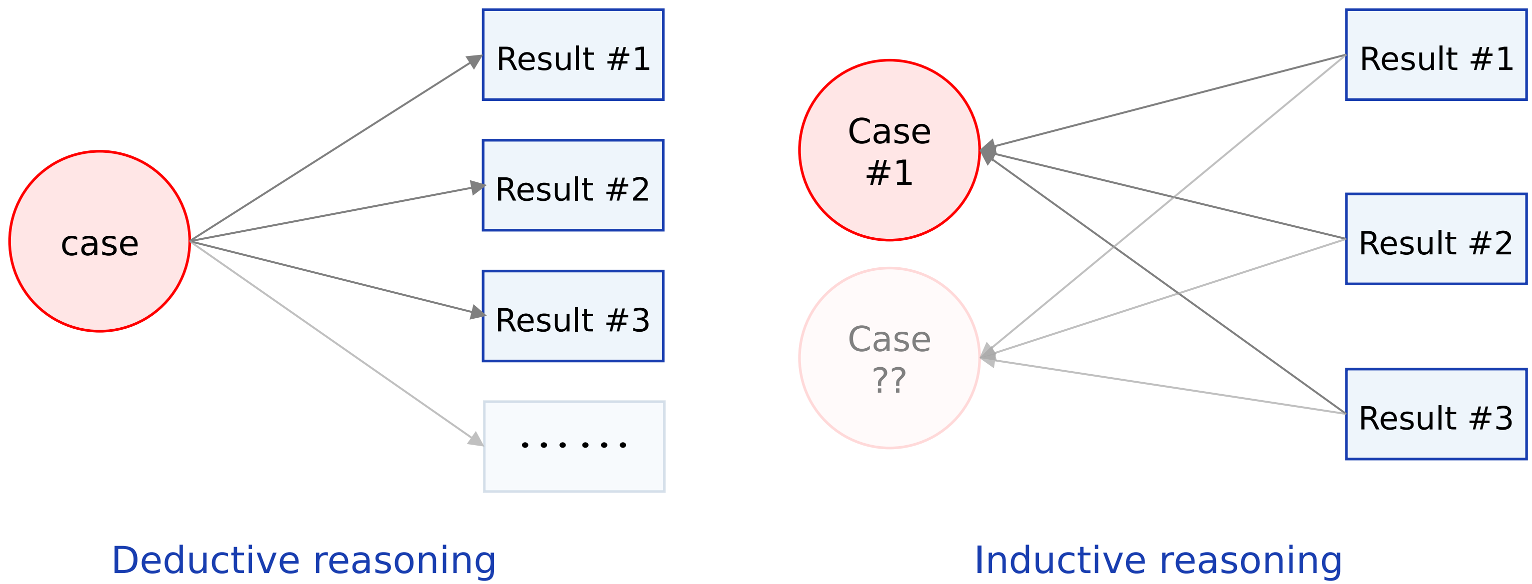
Follow practice steps on 'ex_0512.ipynb' file

Latent Semantic Analysis (LDA)

About the LDA:

- Developed by D. Blei, A. Ng and M. I. Jordan.
- One of the most representative topic modeling algorithms.
- Achieves clustering by calculating the topic distributions.
- Based on a Bayesian model.
- More information can be found at:
  http://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf
  https://ai.stanford.edu/~ang/papers/nips01-lda.pdf

Deductive reasoning vs Inductive reasoning:

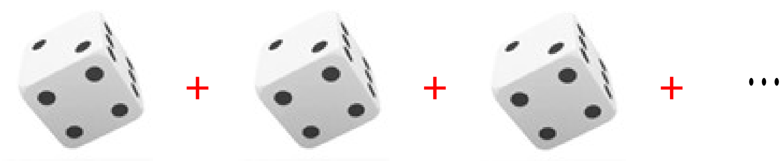


- In an inductive reasoning, the results are observed and the causes are hidden.

LDA algorithm:

- The accumulated counts of random experiments with more than two possible discrete outcomes can be described by a Multinomial distribution.

Ex Rolling a dice n times, the count probability of the sides follows a Multinomial distribution.



- In Bayes' theorem, the Dirichlet is "prior conjugate" of the Multinomial distribution.

Multinomial posterior

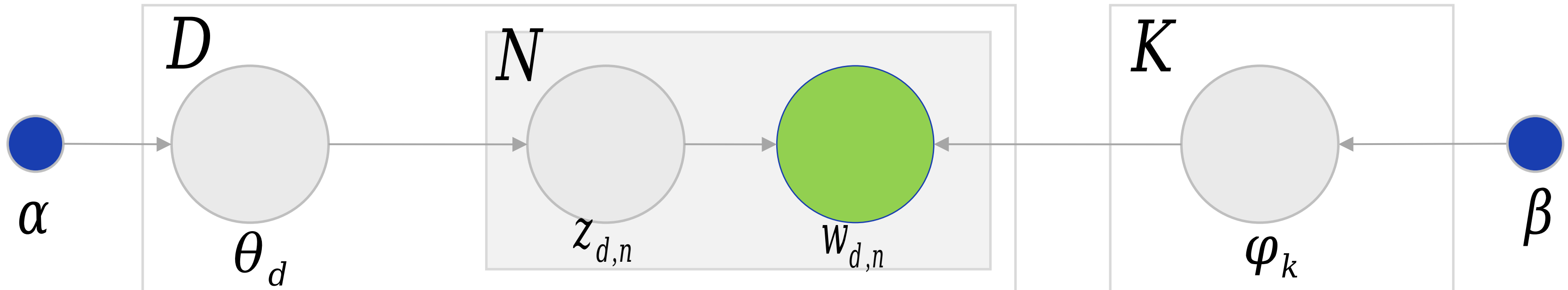
$$\boxed{P(X|Y)} = \frac{P(Y|X)\boxed{P(X)}}{P(Y)}$$

Dirichlet prior

LDA algorithm:

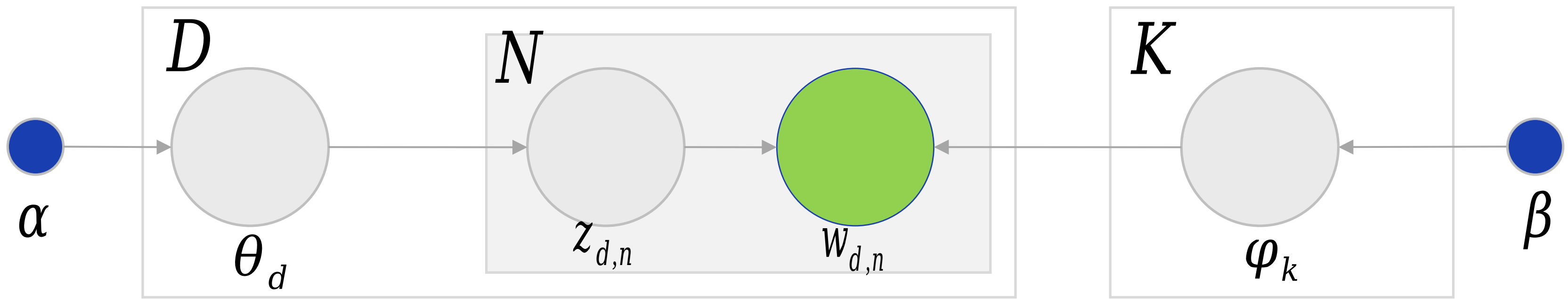
- In LDA we assume that:
  - Document distribution follows a Multinomial with the Dirichlet prior conjugate.
  - Word distribution follows a Multinomial with the Dirichlet prior conjugate.
- What we can actually observe are the documents that contain words. The topics remain hidden.
- We also assume that the documents are “bags-of-words” where the ordering does not matter.
- By an inductive reasoning (Bayes’ theorem), LDA extracts the distribution of the hidden topics.

LDA algorithm:

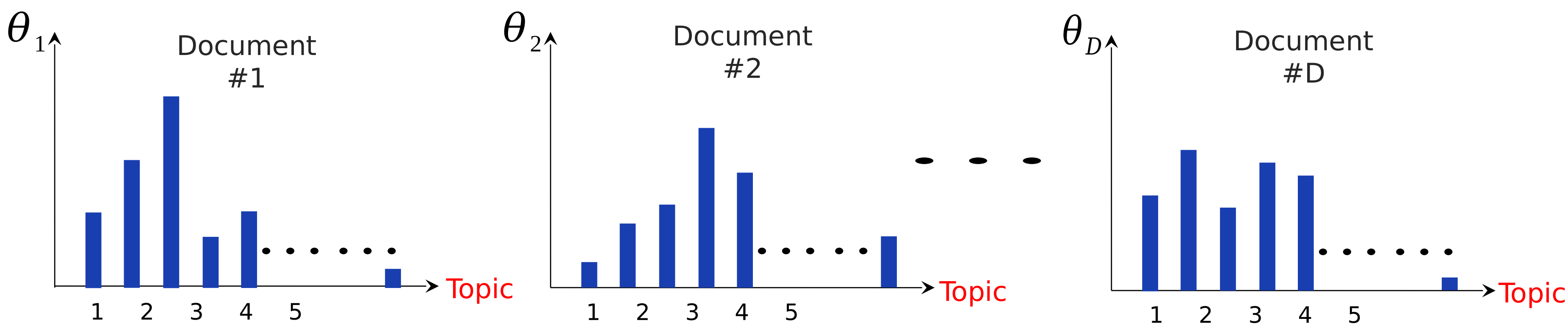


- Illustrated in a "plate notation".
- Rectangles mean repetitive steps.
- We assume that there are in total:
  - D documents such that d=1,…,D
  - N words such that n=1,…,N
  - K topics such that k=1,…,K
- Also, we assume that the number of different words (features) is F.

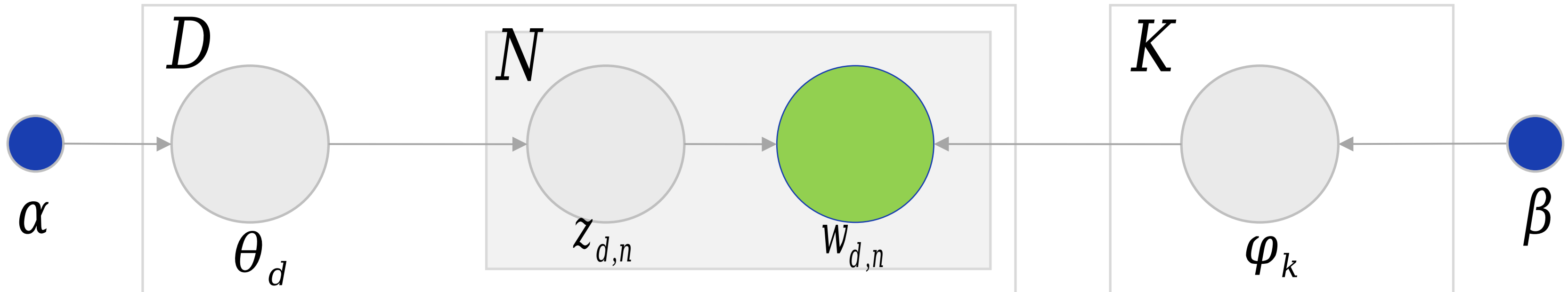
LDA algorithm:



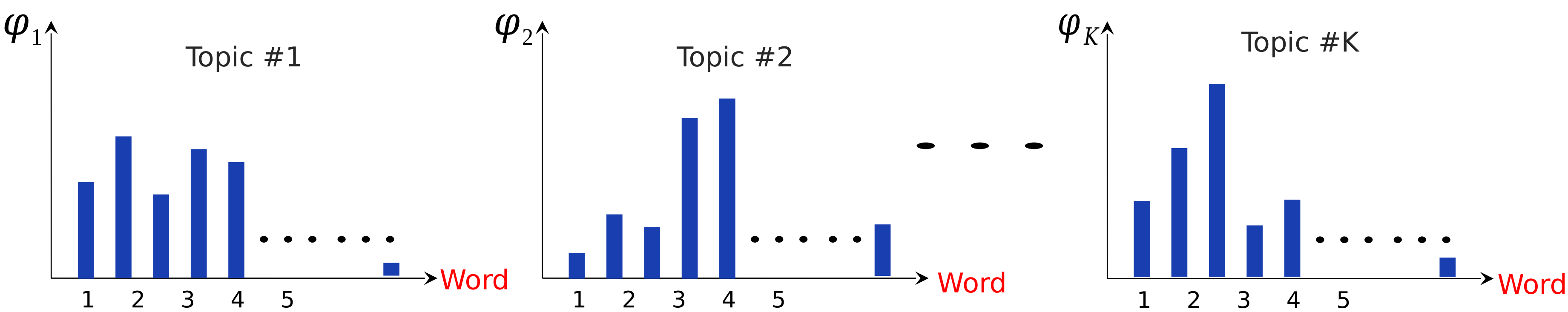
- α is the parameter of Dirichlet prior for the per-document topic distribution θd (with d=1,…,D).



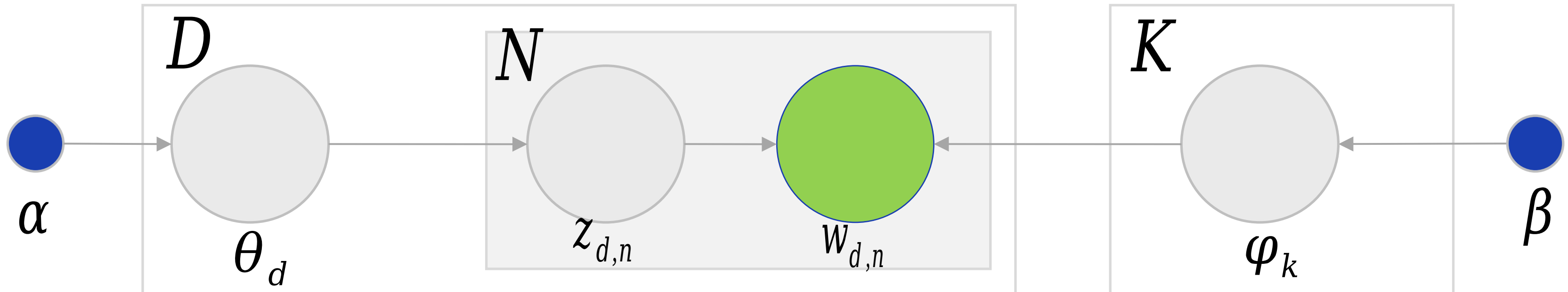
- LDA algorithm:



- β is the parameter of Dirichlet prior for the per-topic word distribution φk (with k=1,…,K).



LDA algorithm:



- $z_{d,n}$ is the topic for the n-th word in the d-th document. It is an integer from the range [1,2,…,K].
- $w_{d,n}$ is the n-th word in the d-th document. It is an integer from the range [1,2,…,F].

3.4. Topic Modeling

Generative process:

- Corresponds to a deductive reasoning.
- Suppose that we know the model.
- Documents that are word collections (bag-of-words) can be generated as following.

  K = number of topics, D= number of documents, N= number of words.

A per-document topic distribution is chosen with Dirichlet() prior.
,

→

A per-topic word distribution is chosen with Dirichlet() prior.
,

→

The topics are randomly chosen according to and the words are randomly chosen according to .

→

The words that have been generated form the documents (bag-of-words).

Inference of the topics:

- Corresponds to an inductive reasoning.
- We do not know the model nor the topics.
- We would like to infer the topics from the dataset (documents, that is, word collections).
- The generative model was based on the joint probability $P(W,Z,\theta,\varphi;\alpha,\beta)$.
- Steps for the inference:

  1) Marginalize out the θ and φ. Get

  2) Using the dataset (W), get the probability $P(Z|W;\alpha,\beta)$.

  3) Infer about the α and β.

Coding Exercise #0513

Follow practice steps on 'ex_0513.ipynb' file

Unit 4.

Natural Language Processing with Keras

4.1. Natural Language Processing with Keras

Natural Language Processing with Keras

One-hot-encoding vs word embedding:

- We notice obvious problems with the one-hot-encoding representation.
- To improve, the “word embedding” is introduced which is a distributed representation method.

One-Hot-Encoding	Embedding
The dimension of the vector space is large. The dimension is as large as the vocabulary size.	The dimension of the vector space is limited.
Vectors are sparse; they are mostly filled with 0s that carry no information.	Vectors are dense. Every vector element carries some information.
No semantic relationship among the vectors. The vectors are orthogonal to each other.	Semantic relationship among the vectors.

- There are also “paragraph embedding” and “document embedding” representations.
- We will call “dense vector” or “embedding vector” interchangeably.

One-hot-encoding vs word embedding:

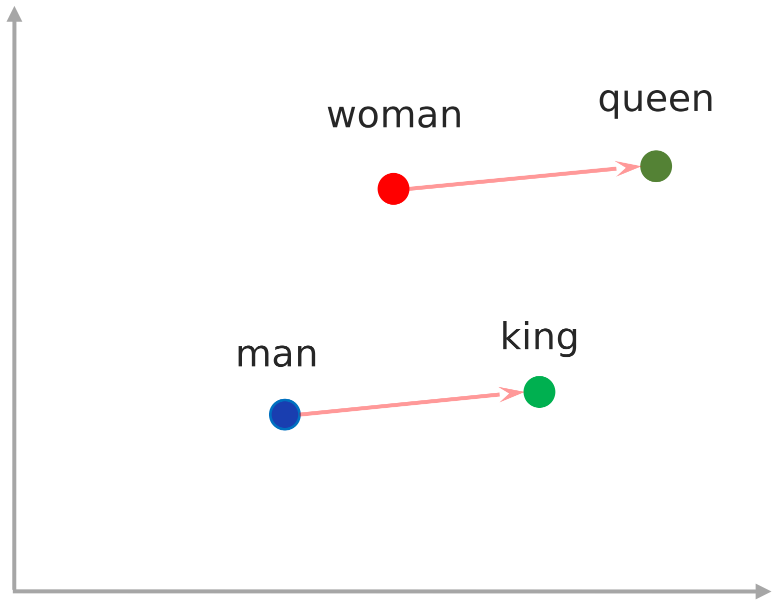
Ex Given a sentence "I eat an apple every morning", let's suppose that the words are indexed as:

I	: 3
Eat	: 0
An	: 2
Apple	: 1
Every	: 4
Morning	: 5

The words would have the following one-hot-encoding representations:

I	: [0 0 0 1 0 0]
Eat	: [1 0 0 0 0 0]
An	: [0 0 1 0 0 0]
Apple	: [0 1 0 0 0 0]
Every	: [0 0 0 0 1 0]
Morning	: [0 0 0 0 0 1]

Word embedding (Word2Vec):



- Among the dense vectors, relationships such as following are established:

$$queen - woman = king - man$$

Word embedding (Word2Vec):

- Use CBOW (Continuous Bag of Words) and/or Skip-Gram to build the embedding vectors.
  1). Build a predictive model based on the Softmax regression (multi-class logistic regression).
  2). We assume one-hot-encoded input and output vectors.
  3). Extract the embedding vectors from the trained weight matrices.

Word embedding (Word2Vec):

- CBOW: Using the context words, predict the (missing) center word.

Training Sentence	Center Word	Context Words
I eat an apple every morning	I	eat
I eat an apple every morning	eat	I, an
I eat an apple every morning	an	eat, apple
I eat an apple every morning	apple	an, every
I eat an apple every morning	every	apple, morning
I eat an apple every morning	morning	every

- We assumed a “sliding window” over the training sentence.

Word embedding (Word2Vec):

- CBOW: Using the context words, predict the (missing) center word.

Ex) Let’s suppose that the vector dimension of the one-hot-encoded words = 6.

Let’s also suppose that we would like to find dense vectors of dimension = 3.

We will consider two context words (one from the left and another from the right).

So, we have a situation as the following:

I eat an apple every morning

↓

I eat an ___?___ every morning

Word embedding (Word2Vec):

- CBOW: Using the context words, predict the (missing) center word.

Ex) (continues from the previous page)

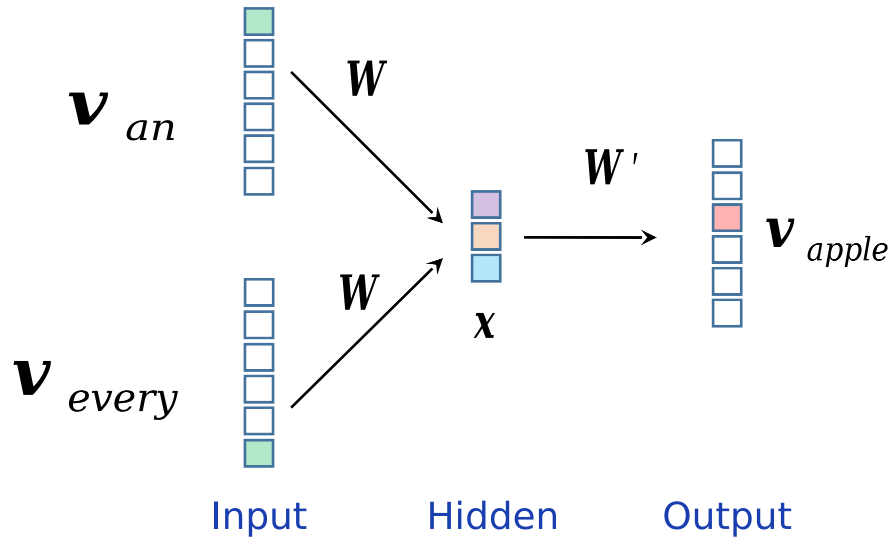
Then, we build a Softmax regression model.

For the vector inputs of “an” and “every”,

we would like to train the weights $\boldsymbol{W}$

and $\boldsymbol{W'}$ such that the predicted output is

the vector “apple”.



One-hot-encoded words : =[1 0 0 0 0 0] , =[0 0 0 0 0 1] , =[0 0 1 0 0 0]

Word embedding (Word2Vec):

- CBOW: Using the context words, predict the (missing) center word.

Ex) (continues from the previous page)

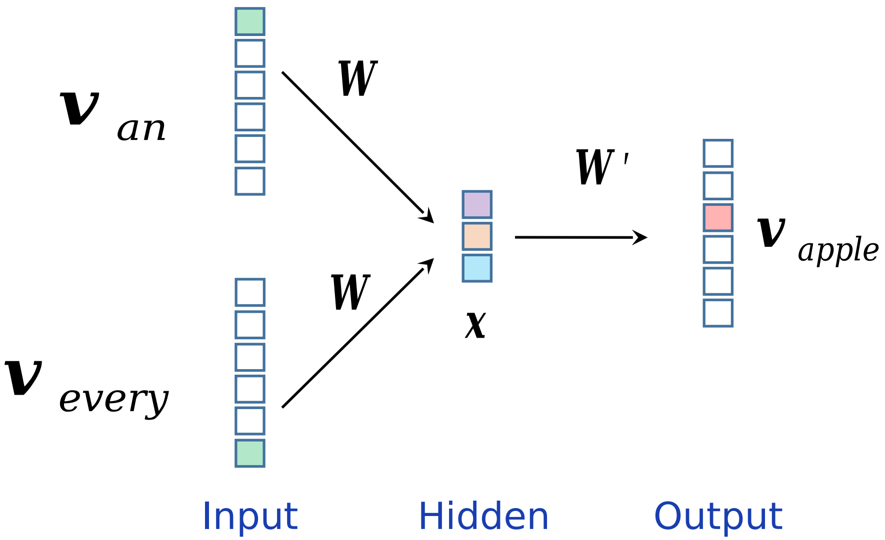
We have the following sizes:

Size of the matrix W=3×6

Size of the matrix W'=6×3

Dimension of the vector x= 3.

Dimension of the input and output = 6.



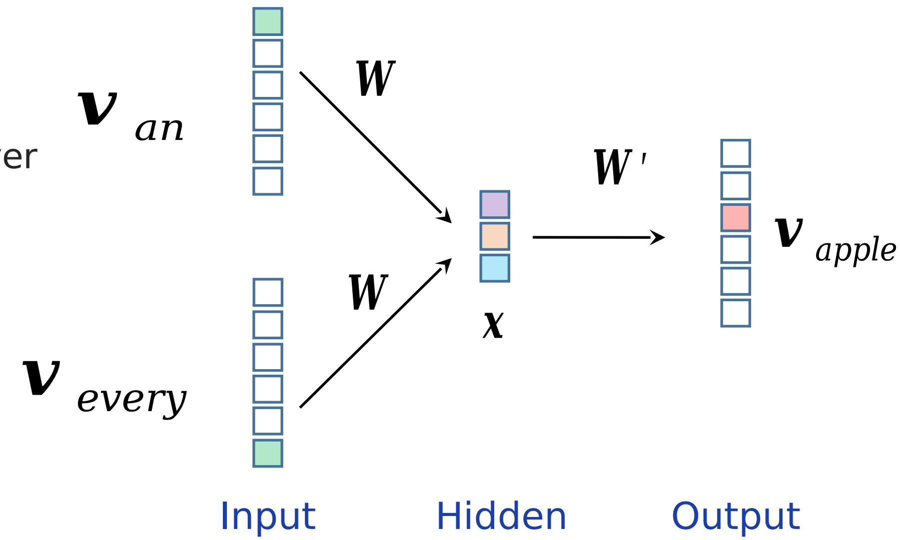
Word embedding (Word2Vec):

- CBOW: Using the context words, predict the (missing) center word.

Ex). (continues from the previous page)

We propagate forward from the input layer

to the hidden layer (a single node):



= Average for the hidden node.

Word embedding (Word2Vec):

- CBOW: Using the context words, predict the (missing) center word.

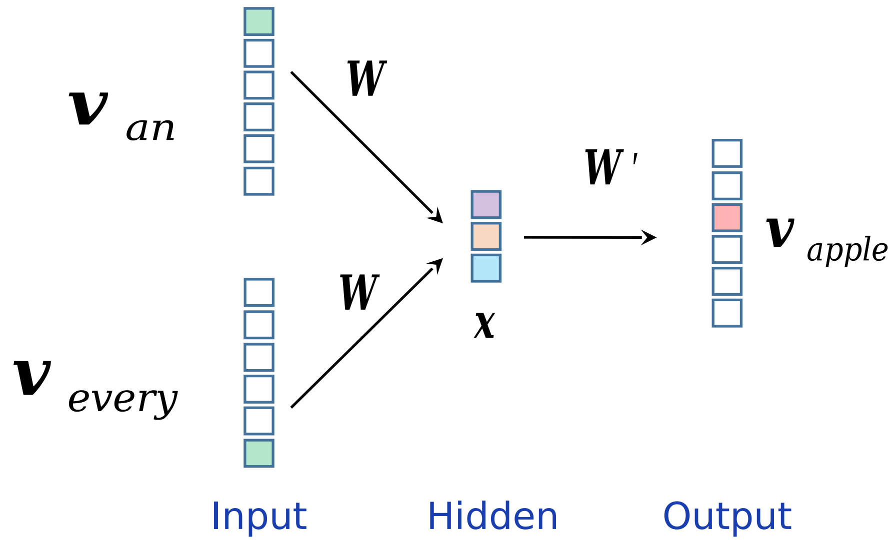
Ex. (continues from the previous page)

We propagate forward to the output layer:

We should train the weights W and W'

argmax() and argmax()

is minimized.



Word embedding (Word2Vec):

- CBOW: Using the context words, predict the (missing) center word.

Ex). (continues from the previous page)

Now, let’s interpret the result.

a). When we propagate from the input layer to the hidden layer (by matrix multiplication),

the one-hot-encoded input vectors and are picking the columns 0 and 5

of $\boldsymbol{W}$ and projecting them to the hidden layer with the target dimension = 3.

b). So, the dense vectors for “an”, “every” are the columns 0 and 5 of the trained $\boldsymbol{W}$.

c). Analogously, we can extract dense vectors from the rows of the trained $\boldsymbol{W'}$.

Word embedding (Word2Vec):

- Skip-Gram: Using a center word, predict the (missing) context words.

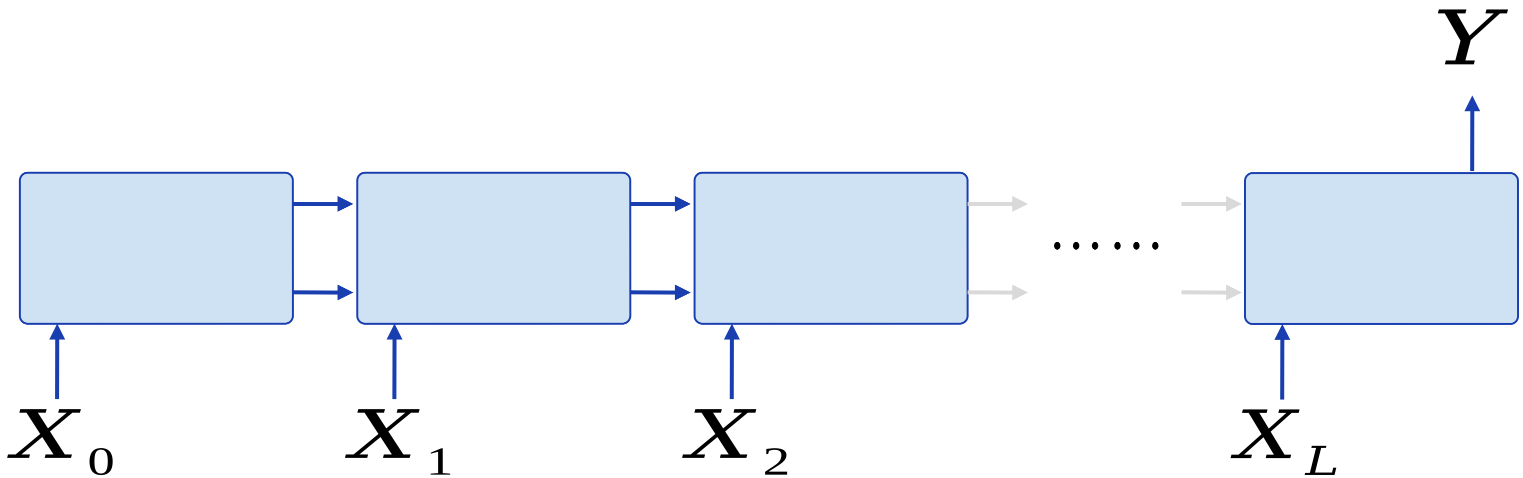
I eat an apple every morning

I eat ? apple ? morning

- Similar to the CBOW, here also we train a Softmax regression to predict the missing words.
- We extract the dense vectors (embedding vectors) from the trained weight matrices.

LSTM network for document classification:

- “Sequence in and Vector out” model.
- Embedding representation of the words.



LSTM network for document classification: a code example.

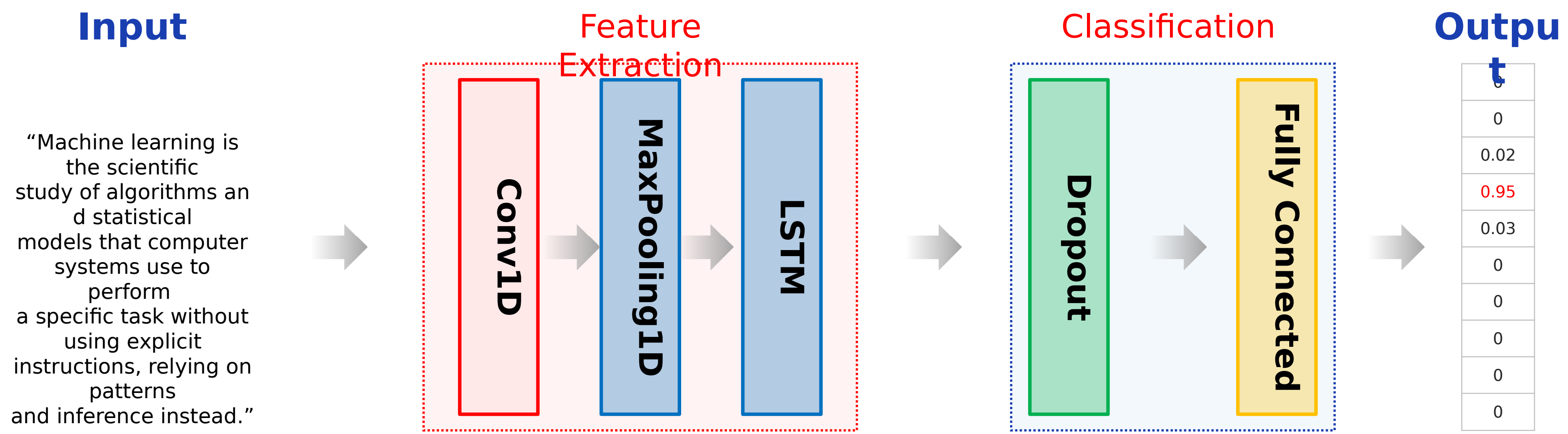
```
# Import the necessary classes.
from keras.models import Sequential                                          # We will use the
Sequential API.
from keras.layers import Dense, LSTM, Embedding
```

```
# Build a model by adding the layers.
my_model = Sequential()
my_model.add(Embedding(n_words,n_input))
my_model.add(LSTM(units=n_neurons, return_sequences=False, input_shape=(None, n_input),
activation='tanh'))
my_model.add(Dense(1, activation='sigmoid'))
```

- In LSTM(), we should set *return_sequences=False* for a “Sequence in and Vector out” model.

Deep learning model for document classification:

- 1D convolution + 1D max pooling + LSTM for the feature extraction.
- Localized sequence patterns picked up by the 1D convolution.



Deep learning model for document classification: a code example.

```
# Import the necessary classes.
from keras.models import Sequential                                          # We
will use the Sequential API.
from keras.layers import Dense, LSTM, Embedding, Conv1D, MaxPool1D, Dropout

# Build a model by adding the layers.
my_model = Sequential()
my_model.add(Embedding(n_words, n_input))                    # Embedding layer.
my_model.add(Conv1D(filters=n_filters, kernel_size = k_size,
strides=stride_size,padding='valid',activation='relu'))
my_model.add(MaxPool1D(pool_size = 2))
my_model.add(LSTM(units=n_neurons, return_sequences=False, input_shape=(None, n_input),
activation='tanh'))
my_model.add(Dropout(rate=hold_prob))
my_model.add(Dense(1, activation='sigmoid'))
```

Coding Exercise #0514

Follow practice steps on 'ex_0514.ipynb' file

Coding Exercise #0515

Follow practice steps on 'ex_0515.ipynb' file

Coding Exercise #0516

Follow practice steps on 'ex_0516.ipynb' file

Coding Exercise #0517

Follow practice steps on 'ex_0517.ipynb' file

End of Document

SAMSUNG

Together for Tomorrow!
Enabling People

Education for Future Generations