

# Textual Data Preparation for Analytics

CPE 393: Text Analytics

***Dr. Sansiri Tarnpradab***

*Department of Computer Engineering  
King Mongkut's University of Technology Thonburi*

*Intro*

*Pattern  
Matching*

*Text  
Visualization*

*Web Scraping*

*Text  
Preparation*

*Text Feature  
Representation*

*Text  
Classification*

*Text  
Clustering*

*Topic  
Modeling*

*Extractive  
Summarization*

*Abstractive  
Summarization*

*???*



# Outline

- Challenge from textual Data as-is
- Textual data preparation
  - Cleaning
  - Pre-processing

# Nature of Textual Data

- Sequence of characters (tokens)
- Unstructured
- Constantly evolving
- Challenges are from:
  - Informal settings
    - e.g. social media, SMS, etc.
    - Brief and informal
  - Automatic speech recognition
  - Optical character recognition

Original tweet

Still have to get up early 2mr thou 😊 so Gn 😴

Normalized tweet

Still have to get up early tomorrow though 😊 so Good night 😴

# Noises in Textual Data



More examples:

- Social media post

Hellooooooooooooo

- Fat finger error
- HTML tags
- Frequent words (e.g. stopwords)

```
> stopwords("english")
[1] "i"          "me"          "my"          "myself"      "we"
[6] "our"        "ours"        "ourselves"   "you"         "your"
[11] "yours"      "yourself"    "yourselves"  "he"          "him"
[16] "his"        "himself"     "she"         "her"         "hers"
[21] "herself"    "it"          "its"         "itself"      "they"
[26] "them"       "their"       "theirs"      "themselves"  "what"
[31] "which"     "who"         "whom"        "this"        "that"
[36] "these"     "those"       "am"          "is"          "are"
[41] "was"       "were"        "be"          "been"        "being"
[46] "have"      "has"         "had"         "having"      "do"
```



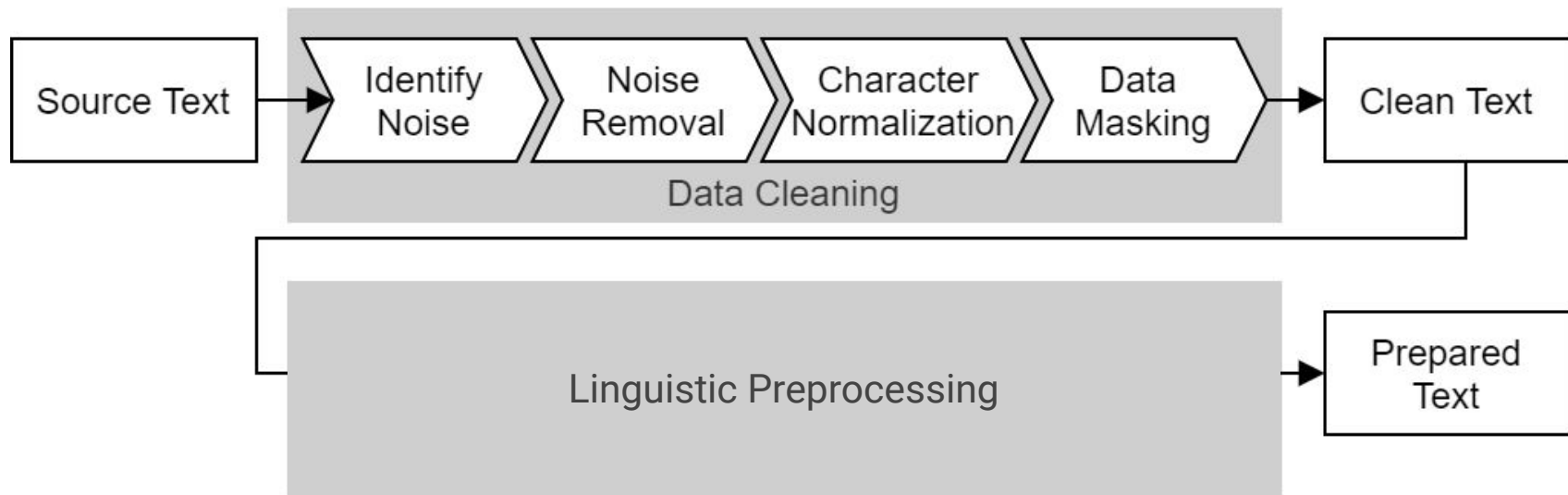
# *Pre-processing*

*(Revisiting Lecture 3)*

## Common Techniques:

- Tokenization
- Stopword removal
- Lemmatization (or stemming)
- Special characters

# Textual Data Preprocessing *Pipeline*



Ref: <https://github.com/blueprints-for-text-analytics-python/blueprints-text/tree/master/ch04>



# *Text Cleaning*

- Noise removal
- Character normalization
- Data masking

# Handling *Noises*

- Addressing stopwords
- Removing HTML Tags, Special Characters
- Handling Capitalization, Punctuation
- Eliminating Non-Alphabetic Characters

Let's try with short sentence first.

✓  
0s

```
[10] # Example text
      text = "This is an example sentence with some stopwords."

      # Tokenize the text
      words = word_tokenize(text)

      # Get the English stopwords from NLTK
      stop_words = set(stopwords.words('english'))

      # Remove stopwords
      filtered_words = [word for word in words if word.lower() not in stop_words]

      # Print the original and filtered words
      print("Original Words:", words)
      print("\nFiltered Words:", filtered_words)
```

Original Words: ['This', 'is', 'an', 'example', 'sentence', 'with', 'some', 'stopwords', '.']

Filtered Words: ['example', 'sentence', 'stopwords', '.']

## Remove stopwords

```
✓ [15] # Example HTML text with tags and formatting
0s    html_text = """
        <!DOCTYPE html>
        <html>
        <head>
        <title>Sample HTML</title>
        </head>
        <body>
        <h1>Welcome to my website</h1>
        <p>This is a <b>sample</b> paragraph with <i>formatting</i>.</p>
        <p>Here's a <a href="https://example.com">link</a> to another page.</p>
        </body>
        </html>
        """
```

```
✓ [16] from bs4 import BeautifulSoup
0s    import re

    def clean_html_tags(text):
        # Initialize BeautifulSoup with the HTML text
        soup = BeautifulSoup(text, 'html.parser')

        # Remove all HTML tags
        clean_text = soup.get_text(separator=' ')

        # Remove extra whitespace
        clean_text = re.sub(r'\s+', ' ', clean_text).strip()

        return clean_text
```

## Removing HTML Tags

```
[24] import re

def clean_punctuation(text):
    # Define a regular expression pattern to match punctuation and special characters
    # Matches any character that is not a word character (\w), space (\s), or underscore (_)
    punctuation_pattern = re.compile(r'^\w\s|_|_')

    # Replace punctuation and special characters with an empty string
    cleaned_text = re.sub(punctuation_pattern, '', text)

    return cleaned_text

# Example text with punctuation and special characters
text_with_punctuation = "Hello! How are you doing? I'm doing fine, thank you!"

# Clean punctuation and special characters
cleaned_text = clean_punctuation(text_with_punctuation)
print(cleaned_text)
```

Hello How are you doing Im doing fine thank you

## Clean special characters

```

import html

def clean(text):
    # convert html escapes like & to characters.
    text = html.unescape(text)
    # tags like <tab>
    text = re.sub(r'<[^>]*>', ' ', text)
    # markdown URLs like [Some text](https://....)
    text = re.sub(r'\([^\(\)]*\)\([^\(\)]*\)', r'\1', text)
    # text or code in brackets like [0]
    text = re.sub(r'\[[^\[\]]*\]', ' ', text)
    # standalone sequences of specials, matches &# but not #cool
    text = re.sub(r'(?:^(|s)[&#<>{}\\[\\]|\\:-]{1,})(?:|s|$)', ' ', text)
    # standalone sequences of hyphens like --- or ==
    text = re.sub(r'(?:^(|s)[\\-+=]{2,})(?:|s|$)', ' ', text)
    # sequences of white spaces
    text = re.sub(r'\\s+', ' ', text)
    return text.strip()

```

## Remove noise with RegEx

# Character

## Normalization

The process of transforming characters in text data to a standardized form

- Unicode
- Lowercase/Uppercase
- Diacritics, accents, or other variations

### DIACRITICS

´	(é)	acute accent	˘	(ü)	breve
`	(è)	grave accent	ˇ	(č)	haček
ˆ	(ô)	circumflex	¨	(naïve)	diaeresis
~	(ñ)	tilde	¨	(glögg)	umlaut
—	(ō)	macron	¸	(ç)	cedilla

```
[25] import unicodedata

def normalize_characters(text):
    # Normalize text by decomposing Unicode characters and then removing combining characters
    normalized_text = unicodedata.normalize('NFKD', text).encode('ascii', 'ignore').decode('utf-8')

    # Convert text to lowercase
    normalized_text = normalized_text.lower()

    return normalized_text

# Example text with accented characters
text_with_accented_characters = "Café au Lait"

# Normalize characters
normalized_text = normalize_characters(text_with_accented_characters)
print(normalized_text)
```

cafe au lait

## Character normalization



```
text = "The café “Saint-Raphaël” is loca-\nted on Côte d’Azur.”  
text
```

```
import textacy  
import textacy.preprocessing as tprep  
  
if textacy.__version__ < '0.11':  
    # as in book  
    def normalize(text):  
        text = tprep.normalize_hyphenated_words(text)  
        text = tprep.normalize_quotation_marks(text)  
        text = tprep.normalize_unicode(text)  
        text = tprep.remove_accents(text)  
        return text  
  
else:  
    # adjusted to textacy 0.11  
    def normalize(text):  
        text = tprep.normalize.hyphenated_words(text)  
        text = tprep.normalize.quotation_marks(text)  
        text = tprep.normalize.unicode(text)  
        text = tprep.remove.accents(text)  
        return text  
  
print(normalize(text))
```

The cafe "Saint-Raphael" is located on Cote d'Azur.

## Character normalization

Function	Description
<code>normalize_hyphenated_words</code>	Reassembles words that were separated by a line break
<code>normalize_quotation_marks</code>	Replace all kind of fancy quotation marks with an ASCII equivalent
<code>normalize_Unicode</code>	Unifies different codes of accented characters in Unicode
<code>remove_accents</code>	Replaces accented characters with ASCII, if possible, or drops them
<code>replace_urls</code>	Similar for URLs like <code>https://xyz.com</code>
<code>replace_emails</code>	Replace emails with <code>_EMAIL_</code>
<code>replace_hashtags</code>	Similar for tags like <code>#sunshine</code>
<code>replace_numbers</code>	Similar for numbers like <code>12345</code>
<code>replace_phone_numbers</code>	Similar for telephone number <code>+1 800 456-6553</code>
<code>replace_user_handles</code>	Similar for user handles like <code>@pete</code>
<code>replace_emojis</code>	Replaces smiley etc. with <code>_EMOJI_</code>

## Preprocessing functions of Textacy

# Data *Masking*

A process of replacing sensitive information with generic placeholders to protect privacy.

## Un-Masked Data

name: Jane Smith

ssn: 555-55-0123

credit card: 0012-3456-7891-2345

address: 12529 Oak Rd. AZ

date of birth: 03-29-1964

e-mail: j.smith@mail.com

## Masked Data

name: [NAME]

ssn: [SSN]

credit card: [CARD]

address: [ADDR]

date of birth: [DOB]

e-mail: [EMAIL]

```
[36] import re

def mask_email_addresses(text):
    # Define a regular expression pattern to match email addresses
    email_pattern = re.compile(r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b')

    # Replace email addresses with a generic placeholder
    masked_text = re.sub(email_pattern, '[EMAIL]', text)

    return masked_text

def mask_phone_numbers(text):
    # Define a regular expression pattern to match phone numbers
    phone_pattern = re.compile(r'(\d{3})-(\d{3})-(\d{4})')

    # Replace phone numbers with a generic placeholder
    masked_text = re.sub(phone_pattern, '[PHONE]', text)

    return masked_text

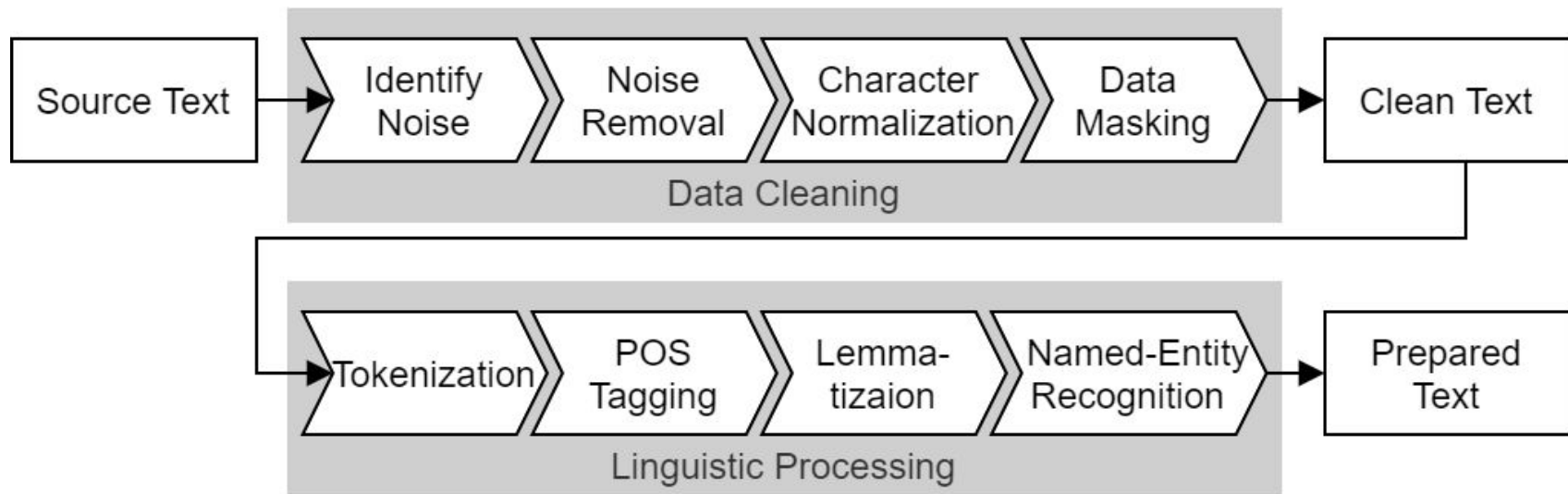
# Example text with email addresses and phone numbers
text_with_sensitive_info = "Please contact me at john.doe@example.com or 123-456-7890. Thank you!"

# Mask sensitive information
masked_text = mask_email_addresses(text_with_sensitive_info)
masked_text = mask_phone_numbers(masked_text)
print(masked_text)
```

Please contact me at [EMAIL] or [PHONE]. Thank you!

## Data masking

# Textual Data Preprocessing Pipeline



Ref: <https://github.com/blueprints-for-text-analytics-python/blueprints-text/tree/master/ch04>

# *Linguistic (Text)*

## *Preprocessing*

- Tokenization
- Lemmatization
- POS tagging
- Name-Entity Recognition (NER)

# POS Tagging



Ref:

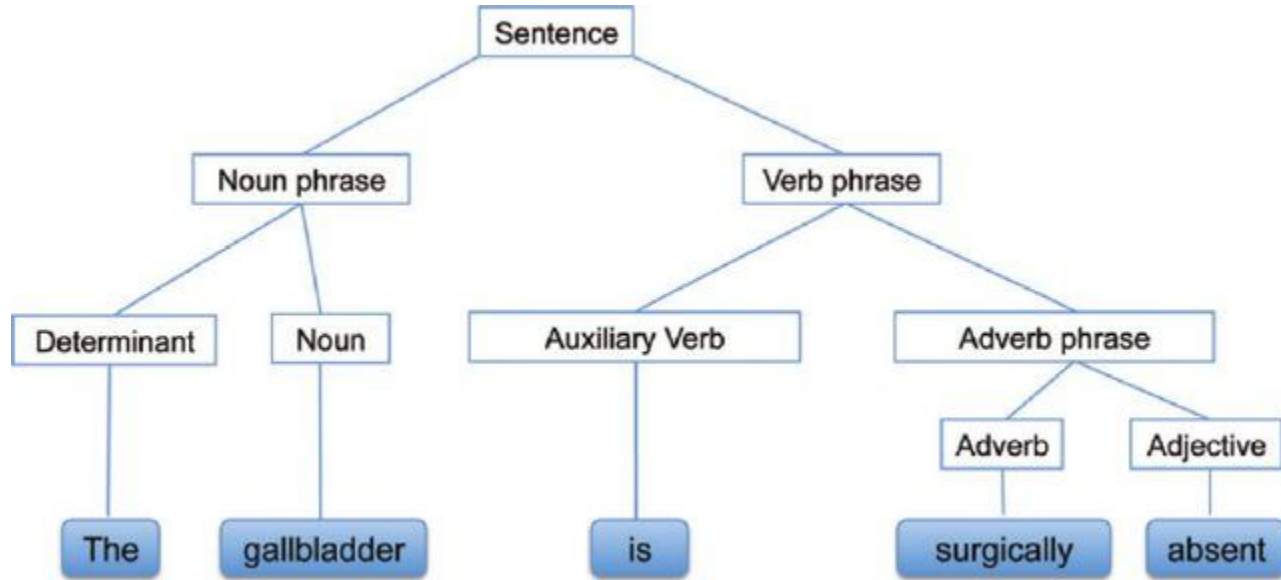
<https://dataingovernment.blog.gov.uk/2019/06/14/natural-language-processing-in-government/>

The process of assigning grammatical categories.

- Noun
- Verb
- Adjective
- Adverb
- etc.

based on its definition and its context within the sentence.

Useful for NLP tasks such as syntax parsing, text analysis, and information extraction.



## Syntactic analysis of the sentence



### 3 List of tags with corresponding part of speech

This section contains a list of tags in alphabetical order and the parts of speech corresponding to them.

1.	CC	Coordinating conjunction			
2.	CD	Cardinal number			
3.	DT	Determiner			
4.	EX	Existential <i>there</i>			
5.	FW	Foreign word	30.	VBN	Verb, past participle
6.	IN	Preposition or subordinating conjunction	31.	VBP	Verb, non-3rd person singular present
7.	JJ	Adjective	32.	VBZ	Verb, 3rd person singular present
8.	JJR	Adjective, comparative	33.	WDT	Wh-determiner
9.	JJS	Adjective, superlative	34.	WP	Wh-pronoun
10.	LS	List item marker	35.	WP\$	Possessive wh-pronoun
11.	MD	Modal	36.	WRB	Wh-adverb
12.	NN	Noun, singular or mass			
13.	NNS	Noun, plural			
14.	NNP	Proper noun, singular			
15.	NNPS	Proper noun, plural			
16.	PDT	Predeterminer			
17.	POS	Possessive ending			
18.	PRP	Personal pronoun			
19.	PRP\$	Possessive pronoun			
20.	RB	Adverb			
21.	RBR	Adverb, comparative			
22.	RBS	Adverb, superlative			
23.	RP	Particle			
24.	SYM	Symbol			
25.	TO	<i>to</i>			
26.	UH	Interjection			
27.	VB	Verb, base form			
28.	VBD	Verb, past tense			
29.	VBG	Verb, gerund or present participle			

List of tags (Ref: <https://www.cis.upenn.edu/~bies/manuals/tagguide.pdf>)

```
[51] import spacy

# Load the English language model
nlp = spacy.load("en_core_web_sm")

def pos_tagging(text):
    # Process the text using spaCy
    doc = nlp(text)

    # Extract POS tags for each token in the text
    pos_tags = [(token.text, token.pos_) for token in doc]

    return pos_tags

# Example text
text = "I love eating pizza with my friends"

# Perform POS tagging
pos_tags = pos_tagging(text)
print(pos_tags)
```

('I', 'PRON')

('love', 'VERB')

('eating', 'VERB')

('pizza', 'NOUN')

('with', 'ADP')

('my', 'PRON')

('friends', 'NOUN')

## Part-of-speech tagging

# *Name-Entity Recognition*

The process of identifying and classifying named entities in text into predefined categories

- Person
- Organization
- Location
- Date
- Quantity
- etc.

The purpose of NER task is to:

- Extract and classify entities mentioned in text data
- Understand the context and extract structured information from unstructured text

contentSkip to site indexPoliticsSubscribeLog InSubscribeLog InToday's **PaperAdvertisementSupported** **ORG** byF.B.I. Agent **Peter Strzok** **PERSON** ,  
**Who Criticized Trump** **PERSON** in Texts, Is FiredImagePeter Strzok, a top **F.B.I.** **GPE** counterintelligence agent who was taken off the special counsel  
 investigation after his disparaging texts about President **Trump** **PERSON** were uncovered, was fired. **CreditT.J. Kirkpatrick** **PERSON** for **The New York**  
**TimesBy Adam Goldman** **ORG** and **Michael S. SchmidtAug** **PERSON** . **13** **CARDINAL** , **2018WASHINGTON** **CARDINAL** — **Peter Strzok**  
**PERSON** , the **F.B.I.** **GPE** senior counterintelligence agent who disparaged President **Trump** **PERSON** in inflammatory text messages and helped  
 oversee the **Hillary Clinton** **PERSON** email and **Russia** **GPE** investigations, has been fired for violating bureau policies, Mr. **Strzok** **PERSON** 's lawyer  
 said **Monday** **DATE** .Mr. Trump and his allies seized on the texts — exchanged during the **2016** **DATE** campaign with a former **F.B.I.** **GPE** lawyer,  
**Lisa Page** — in **PERSON** assailing the **Russia** **GPE** investigation as an illegitimate “witch hunt.” Mr. **Strzok** **PERSON** , who rose over **20 years**  
**DATE** at the **F.B.I.** **GPE** to become one of its most experienced counterintelligence agents, was a key figure in **the early months** **DATE** of the  
 inquiry.Along with writing the texts, Mr. **Strzok** **PERSON** was accused of sending a highly sensitive search warrant to his personal email account.The  
**F.B.I.** **GPE** had been under immense political pressure by Mr. **Trump** **PERSON** to dismiss Mr. **Strzok** **PERSON** , who was removed **last summer**  
**DATE** from the staff of the special counsel, **Robert S. Mueller III** **PERSON** . The president has repeatedly denounced Mr. **Strzok** **PERSON** in posts on

## NER example

```
▶ import spacy

# Load the English language model
nlp = spacy.load("en_core_web_sm")

def ner(text):
    # Process the text using spaCy
    doc = nlp(text)

    # Extract named entities from the processed text
    entities = [(entity.text, entity.label_) for entity in doc.ents]

    return entities

# Example text
text = "KMUTT is located in Bangkok, Thailand."

# Perform NER
named_entities = ner(text)
print(named_entities)
```

```
[('KMUTT', 'ORG'), ('Bangkok', 'GPE'), ('Thailand', 'GPE')]
```

## NER application

# Conclusion

- Challenge from textual Data as-is
- Text Cleaning
  - Noise removal
  - Character normalization
  - Data masking
- Text Preparation
  - Tokenization
  - Lemmatization
  - POS tagging
  - Name-Entity Recognition (NER)

# Lab

(30 mins)

- Pick a forum from one of the websites below:
  - Stackoverflow  
<https://stackoverflow.com/>
  - Ubuntu forum  
<https://ubuntuforums.org/>
- Select ONE question and ONE answer from that forum (*medium to long in length*)
- Apply what we have learned today:
  - Clean (at least 2 techniques)
  - Preprocess (at least 2 techniques)

# Q & A