Data Collection, Preprocessing & Representation

(Natural Language Processing)

HND Thilini

hnd@ucsc.cmb.ac.lk

DATA COLLECTION

Why data collection?

- Data collection, cleaning and preparation are estimated to take 70% of any data science task
- Quality and diversity of data play a significant role
- Sources of data a disparate (very different one from another)
- Defining clear objectives is essential for guiding the data collection process effectively

Potential Data Sources

- Data sources impact the quality, diversity, and applicability of the collected data
- Identify the kind of data you want to collect? Text? Numerical? Speech?
- Identify reliable data sources for the data you want to collect
  - Look for industry-accepted or professional data sources in your domain
  - Look for highly cited sources
  - Make a list of all possible sources & weigh the pros and cons
    - Services offered by data providers
    - Payment plan, if any
    - Frequency of update
    - Attributes of dataset
- Identify how the data might be collected legally from these sources

Potential Data Sources

- Websites and Online Platforms:
  - Websites, blogs, news articles, social media platforms, discussion forums, online communities.
  - Example: Gathering tweets for sentiment analysis or news articles for summarization.
- Text Corpora and Datasets:
  - Curated datasets created for specific tasks, such as sentiment analysis, machine translation, question answering.
  - Example: IMDb dataset for sentiment analysis, WMT translation datasets.
- Domain-Specific Data:
  - Data from a particular domain, industry, or field (e.g., medical journals, legal documents).
  - Example: Medical research articles for medical NLP tasks.
- User-Generated Content:
  - Content generated by users, such as customer reviews, product feedback, and online comments.
  - Example: Amazon product reviews for sentiment analysis or user opinions.
- Scientific Journals and Publications:
  - Academic and research papers, publications in specific domains.
  - Example: Academic papers for building domain-specific language models.

Potential Data Sources

- Factors to Consider:
  - Relevance: Ensure the data source aligns with the project's objectives and intended tasks.
  - Quality: Choose sources with credible, accurate, and well-written content.
  - Diversity: Opt for sources that provide a diverse range of language styles, topics, and perspectives.
  - Accessibility: Consider data accessibility, licensing, and terms of use.

THE INTERNET IN 2023 EVERY MINUTE

Created by: eDiscovery Today & LTMG

Data Collection through APIs

- What is an API?
  - An API is a set of rules and protocols that allow different software applications to communicate with each other.
  - APIs enable developers to request specific data or perform certain actions from remote servers, usually in a structured format like JSON or XML.

Data Collection through APIs

- Benefits of Using APIs for Data Collection:
  - **Structured Data:** APIs provide structured data in a consistent format, making it easier to extract and use.
  - **Real-Time Data:** Many APIs offer real-time or near-real-time access to the latest information.
  - **Efficiency:** APIs allow you to access specific data points without the need to scrape entire web pages.
  - **Reliability:** Data obtained through APIs is typically more reliable and accurate than web scraping.

Data Collection through APIs

- Example Scenario: Collecting Tweets Using the Twitter API:
  - API: Twitter API (requires an API key and authentication).
  - Steps:
    - Sign up for a Twitter Developer account and create an app to obtain API keys.
    - Use Python libraries like ***tweepy*** to send requests to the Twitter API endpoints.
    - Request tweets based on specific keywords, user profiles, or hashtags.
    - Parse the JSON response and extract relevant tweet data.
    - Store the data in a structured format for analysis.

Data Collection through APIs

- Homework
  - Create a twitter account (if you don't have one)
  - Apply for developer status (today!)
  - Access twitter data though API from your account
    - https://github.com/mikhailklassen/Mining-the-Social-Web-3rd-Edition/blob/master/notebooks/Chapter%201%20-%20Mining%20Twitter.ipynb
  - You can use the following libraries:
    - `Tweepy`, `GetOldTweets3`/`Snscrape`
    - See - http://socialmedia-class.org/twittertutorial.html
  - If you prefer you can do this for Facebook instead (or in addition!)

Data Collection by Web Scraping

- Many valuable sources of text data are available online
  - news articles, social media content, discussion forums, etc.
- Web scraping enables us to gather relevant and up-to-date data for analysis and model training

Data Collection by Web Scraping

- Web Scraping Process:
  - HTTP Requests:
    - Web scraping starts with sending HTTP requests to a website's URL.
    - The website responds with HTML content, which contains the data we want to extract.
  - Parsing HTML:
    - The HTML content is parsed to extract specific elements (e.g., paragraphs, headings) containing the desired text data.
  - Data Extraction:
    - Use CSS selectors or XPath expressions to locate and extract the relevant data from the parsed HTML.
  - Data Transformation:
    - The extracted data might require further preprocessing, such as removing HTML tags, handling special characters, and tokenization.
  - Storing Data:
    - Store the extracted and preprocessed data in a structured format (e.g., CSV, JSON) for further analysis or model training.

Data Collection by Web Scraping

- Tools for Web Scraping
  - **Beautiful Soup:** A Python library for parsing HTML and XML documents. It provides easy methods to navigate and search the parsed content.
  - **Scrapy:** A more advanced Python framework for web scraping. It offers a structured approach to building web scrapers and handling complex websites.
  - **Requests:** A Python library for making HTTP requests, often used in conjunction with Beautiful Soup for simple scraping tasks.
  - **Selenium:** A Python library and tool used for automating web browsers to do a number of tasks including web-scraping to extract useful data and information.

Data Collection by Web Scraping

- Tools for Web Scraping

	Beautiful Soup	Selenium	Scrapy
Performance	Slow for a few tasks	Faster than Beautiful Soup but has its limitations	Pretty fast and has the best performance out of the three
Extensibility	Best suited for small, low-complexity projects & beginners	Best suited for core JavaScript featured website	Best suited for large, complex project. Makes project robust & flexible
Ecosystem	Has a lot of dependencies in the ecosystem	Good ecosystem but can't use proxies	Can use proxies and VPMs and hence suitable for complex projects

Data Collection by Web Scraping

- Example Scenario: News Article Scraping

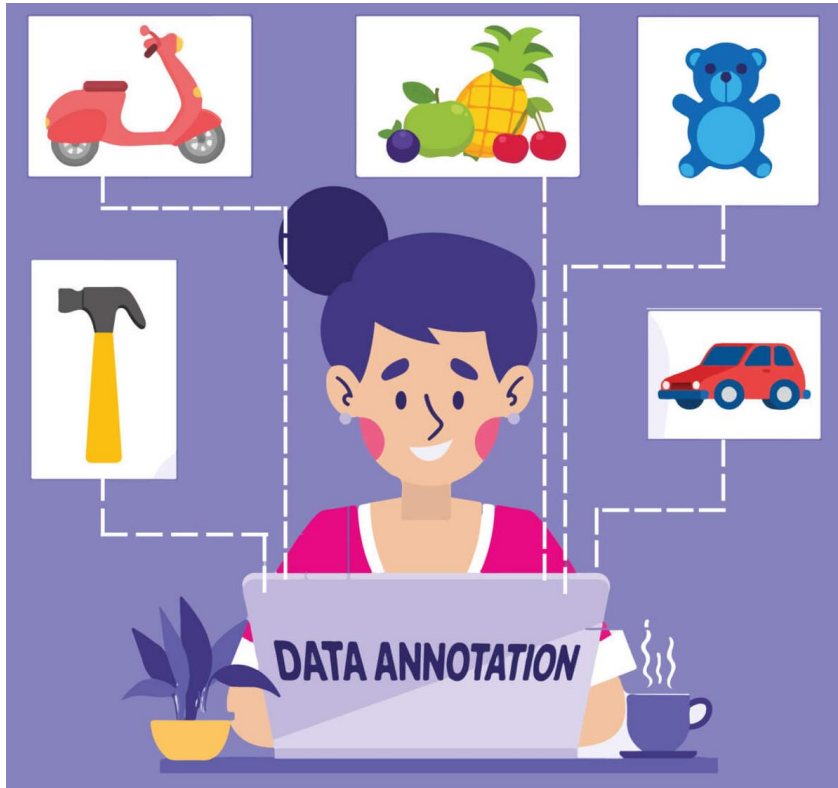
  - **Objective:** Gather recent news articles for a sentiment analysis project.
  - **Steps:**
    - Identify relevant news websites (e.g., major news outlets, specialized news blogs).
    - Send HTTP requests to article pages, retrieve HTML content.
    - Use Beautiful Soup to parse and extract article titles and content.
    - Preprocess extracted text (remove HTML tags, tokenize, etc.).
    - Store the preprocessed data for analysis.

Data Annotation and Labeling

- Involve adding metadata or tags to raw text data, making it suitable for training supervised NLP models.
- High-quality annotations are crucial for building accurate and effective NLP systems

Importance of Annotation:

- Annotated data serves as ground truth for training and evaluating machine learning models.
- Properly labeled data contributes to the success of tasks like sentiment analysis, text classification, and named entity recognition.



Data Annotation and Labeling

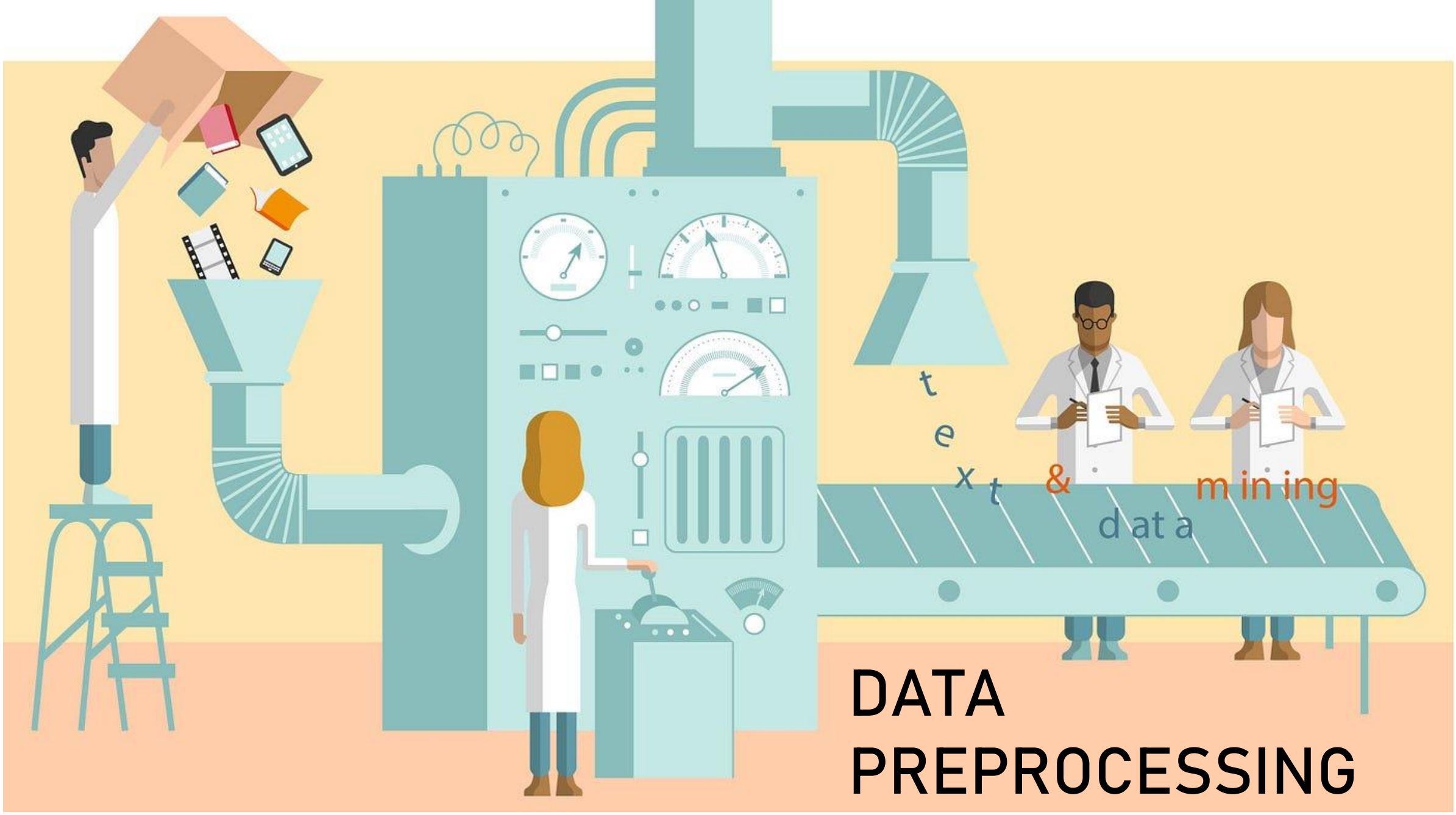
- Annotation Methods:
  - **Manual Annotation:**
    - Human annotators review and label each data point based on predefined guidelines.
    - Ensures high accuracy but can be time-consuming and expensive.
  - **Semi-Automatic Annotation:**
    - Annotators are aided by tools that suggest labels or generate annotations based on certain patterns.
    - Increases efficiency while maintaining human oversight.
  - **Crowdsourcing:**
    - Outsourcing annotation tasks to a large group of workers through platforms like Amazon Mechanical Turk.
    - Cost-effective for large datasets, but quality control is essential.

Data Annotation and Labeling

- Labeling Strategies:
  - **Single-Annotator:** One annotator labels each data point for consistency.
  - **Multiple-Annotator:** Multiple annotators label the same data point to assess inter-annotator agreement.
  - **Majority Voting:** Inconsistent labels are resolved by selecting the most common label among annotators.

Data Annotation and Labeling

- Challenges and Considerations:
  - **Ambiguity:** Some text may have multiple valid interpretations, leading to varying labels.
  - **Subjectivity:** Certain tasks, like sentiment analysis, can be subjective and result in differing annotations.
  - **Bias:** Annotator biases can impact the quality and fairness of labeled data.
  - **Continual Feedback:** Regular communication with annotators helps clarify guidelines and address questions.



DATA PREPROCESSING

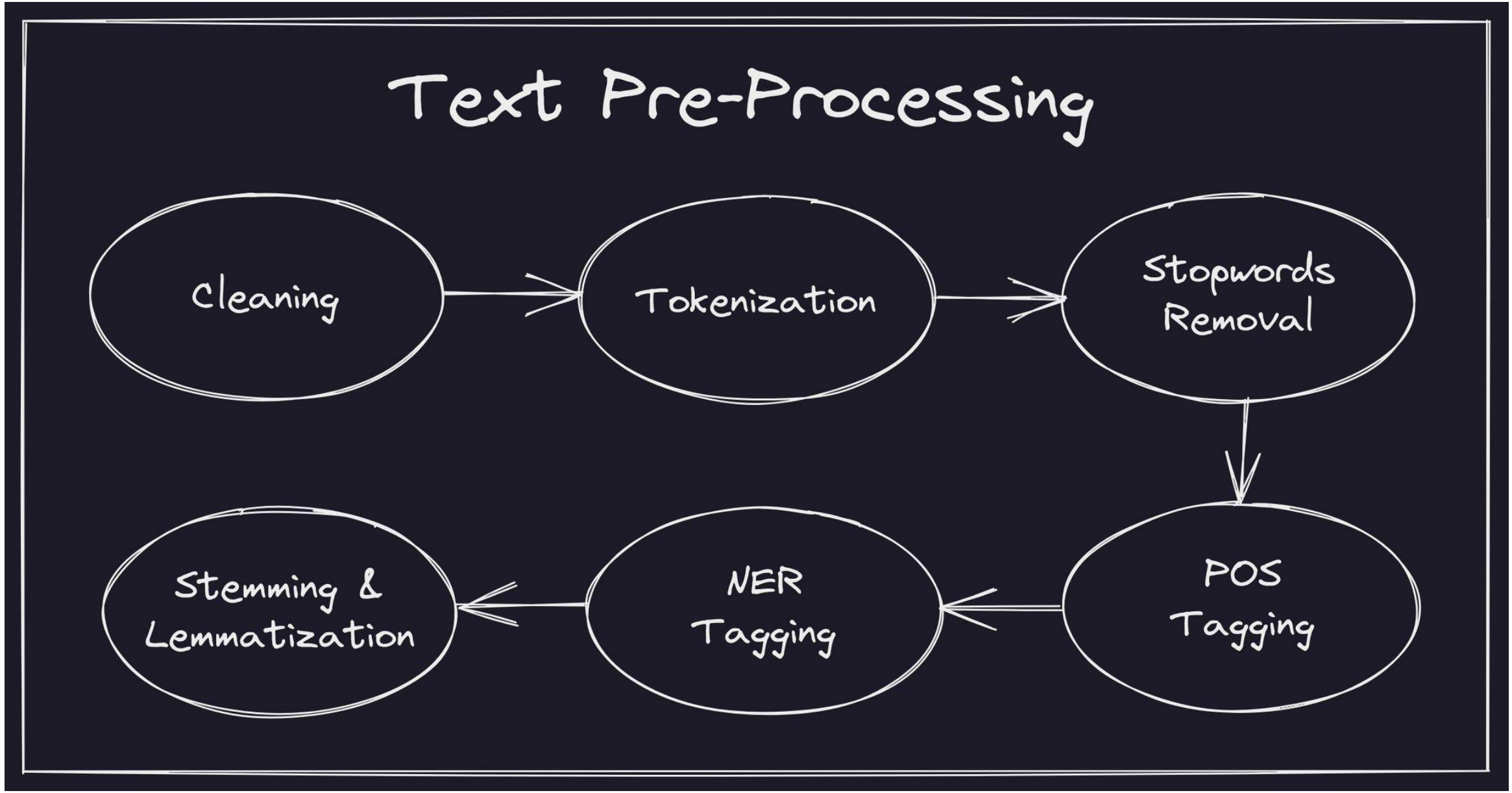
Data Preprocessing

- Definition

  Process of preparing and cleaning raw text data before it can be analyzed by natural language processing (NLP) algorithms or machine learning models.

- Goal

  Transform unstructured text data into a structured and standardized format that can be easily analyzed and understood by computers


Text Pre-Processing
Cleaning
Tokenization
Stopwords Removal
POS Tagging
NER Tagging
Stemming & Lemmatization

Text Preprocessing

- Why Text Preprocessing is required?
  - Cleaning and standardizing the data
  - Reducing the vocabulary size
  - Enabling efficient text representation
  - Improving model accuracy
  - Facilitating text understanding
  - …

Text Preprocessing

- Types of text preprocessing techniques
  - Tokenization
  - Lowercasing
  - Stop word removal
  - Stemming and Lemmatization
  - Removing special characters and digits
  - Removing URLs and email addresses
  - Spell checking and correction
  - …

Tokenization

- Process of breaking up a text into individual words, phrases, symbols or other meaningful elements, called *tokens*
- Tokens are used as the basic building blocks
- Involves identifying the boundaries between words or other meaningful elements in a text
- A text can be split on whitespace or punctuation marks such as spaces, commas, periods, question marks, and exclamation marks

Toenization

- For example, consider the following sentence:

  "John loves playing soccer with his friends.“

- The tokens in this sentence after tokenization might be:
  - John
  - loves
  - Playing
  - soccer
  - with
  - his
  - friends

Lowercasing

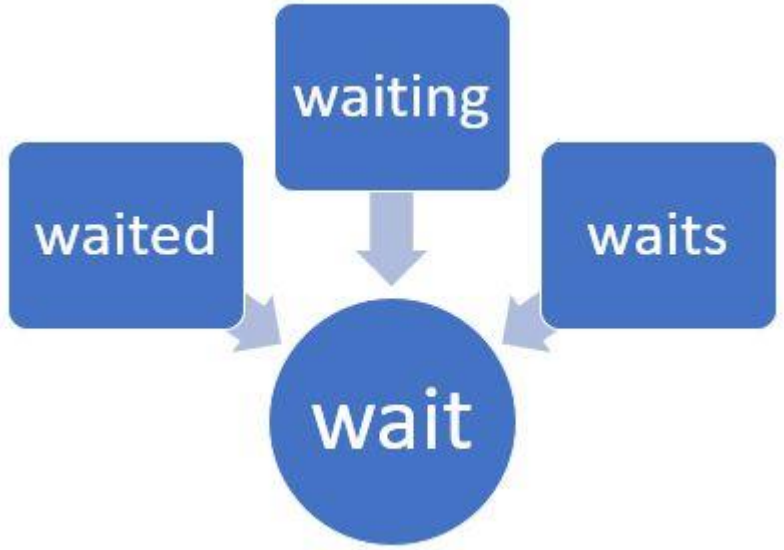
- Lowercasing ALL your text data
- One of the simplest and most effective form of text preprocessing
- Applicable to most text mining and NLP problems
- Useful to ensure that the same word is not treated differently because of its capitalization
- Help in cases where your dataset is not very large
- Significantly helps with consistency of expected output

Lowercasing

Raw	Lowercased
Canada CanadA CANADA	canada
TOMCAT Tomcat toMcat	tomcat

Stemming

- Reduce inflected or derived words to their base form, known as the stem
- Purpose is to simplify the text data and reduce the vocabulary size
- Make easier to process and analyze
- Involves applying a set of rules or algorithms to the words in the text data to identify and remove affixes (prefixes and suffixes)
- Porter stemming, Snowball stemming, Lancaster stemming



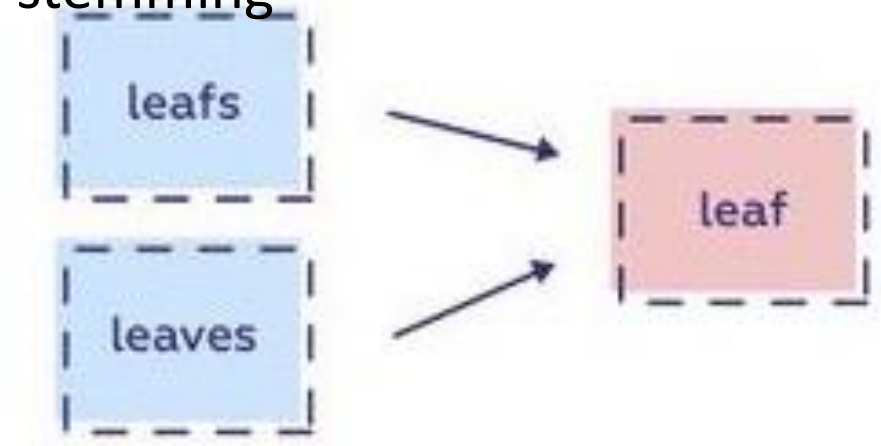
Stemming

	original_word	stemmed_words
0	connect	connect
1	connected	connect
2	connection	connect
3	connections	connect
4	connects	connect

	original_word	stemmed_word
0	trouble	troubl
1	troubled	troubl
2	troubles	troubl
3	troublemsome	troublemsom

Lemmatization

- Very similar to stemming
- Reduce inflected or derived words to their base or dictionary form, known as the lemma
- Considers the context and part of speech of the word in order to produce the correct lemma
- Involves the use of a lexicon or a set of rules
- More accurate technique than stemming
- More computationally expensive and slower than stemming

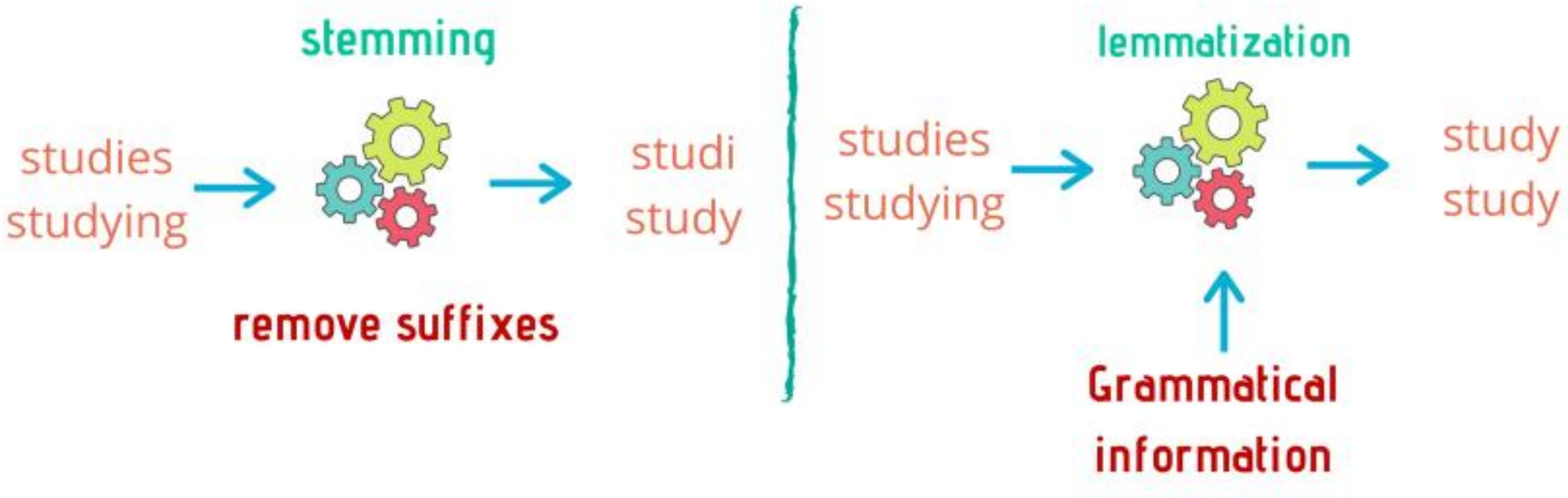


Lemmatization

	original_word	lemmatized_word
0	trouble	trouble
1	troubling	trouble
2	troubled	trouble
3	troubles	trouble

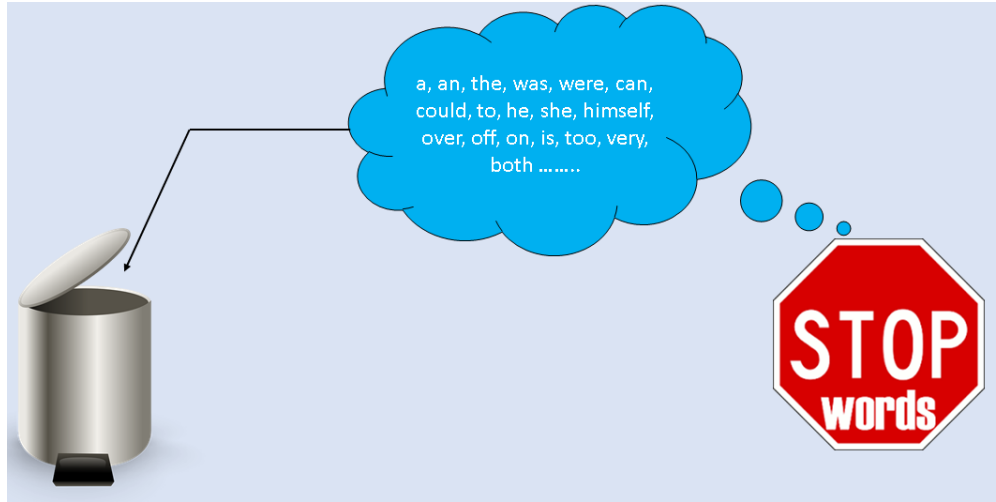
	original_word	lemmatized_word
0	goose	goose
1	geese	goose

Stemming vs Lemmatization



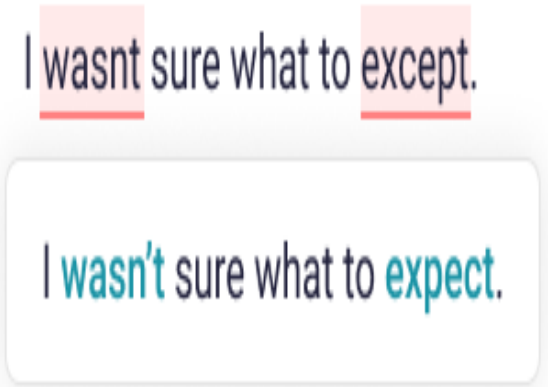
Stop words Removal

- Remove words that are considered irrelevant or redundant in the context of the text data being analyzed
- Stop words are common words that occur frequently in language
- "a", "an", "the", "and", "in", "on", and "at“, etc. - do not carry much meaning on their own and can be safely ignored in many text analysis tasks
- Purpose is to reduce the size of the vocabulary and simplify the text data
- Can be easily implemented using pre-built lists of stop words or by creating custom lists for specific text analysis tasks



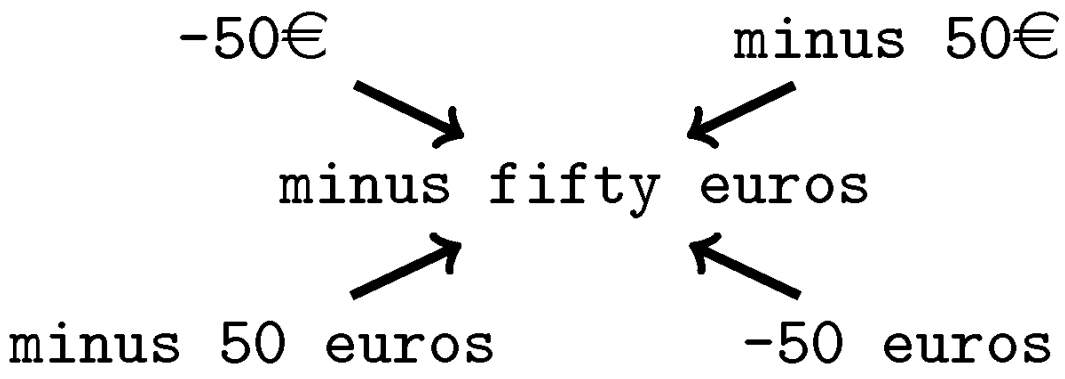
Spell Checking & Correction

- Identify and correct spelling errors in the text data
- Spelling errors can occur due to typos, misspellings, or other mistakes made during the text entry or transcription process - can affect the accuracy and quality of the text analysis results
- Dictionary-based spell checking
- Rule-based spell checking
- Statistical spell checking
- Machine learning-based spell checking
- Mainly useful in applications that require precise language processing, such as sentiment analysis, named entity recognition, and machine translation



Text Normalization

- Transform text data into a standardized format that is easier to process and analyze
- Purpose is to reduce the variability in text data and increase its consistency and accuracy
- Case normalization
- Punctuation removal
- Number normalization
- Symbol and abbreviation expansion
- Accent and diacritic removal …

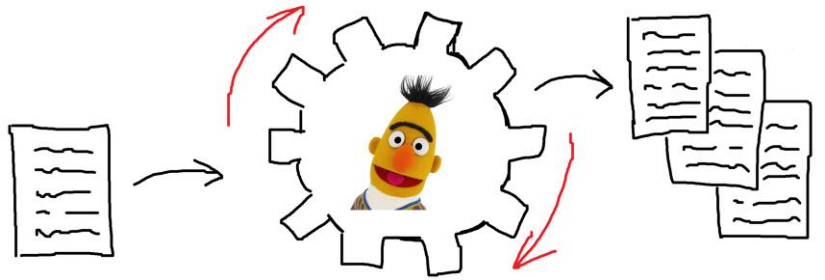


Text Normalization

Raw	Normalized
2moro 2mrrw 2morrow 2mrw tomrw	tomorrow
b4	before
otw	on the way
:) :-) ;-)	smile

Text Enrichment / Augmentation

- Enhance the quality and quantity of text data by adding or generating new content
- Purpose is to increase the diversity and relevance of the text data, and provide additional context and information for text analysis tasks
- **Part-of-speech tagging**: involves labeling each word in the text data with its corresponding part of speech, such as noun, verb, or adjective
- **Named entity recognition**: involves identifying and labeling named entities in the text data, such as people, organizations, and locations
- **Text generation**: involves generating new text data based on the existing text data, using techniques such as language models, recurrent neural networks, and Markov chains



Do you need it all?

- **Must Do:**
  - Noise removal
  - Lowercasing (can be task dependent in some cases)
- **Should Do:**
  - Simple normalization — (e.g. standardize near identical words)
- **Task Dependent:**
  - Advanced normalization (e.g. addressing out-of-vocabulary words)
  - Stop-word removal
  - Stemming / lemmatization
  - Text enrichment / augmentation

General Rule of Thumb

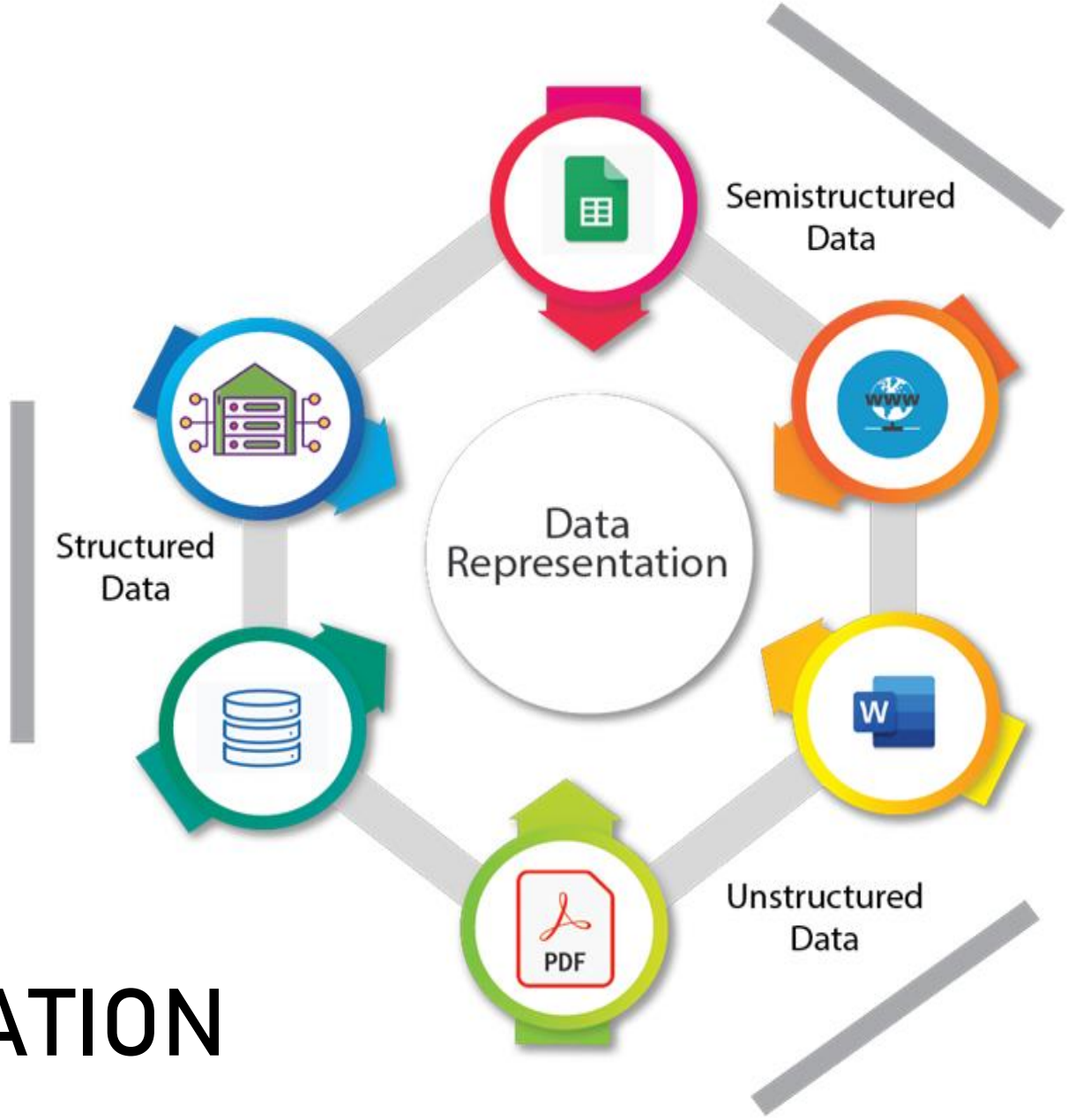
- Not all tasks need the same level of preprocessing

Level of Text Preprocessing Needed

	Domain Specific / Noisy Texts	**General / Well Written Texts**
Lots of data	- Moderate pre-processing - Text enrichment could be helpful	- Light pre-processing - Text enrichment could be helpful, but not critical
Sparse data	- Heavy pre-processing - Text enrichment is important	- Moderate pre-processing - Text enrichment could be helpful

By: Kavita Ganesan

DATA REPRESENTATION

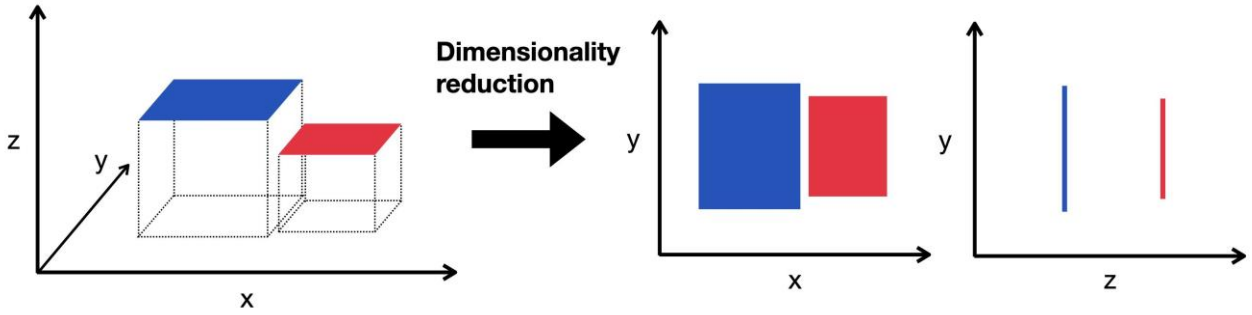


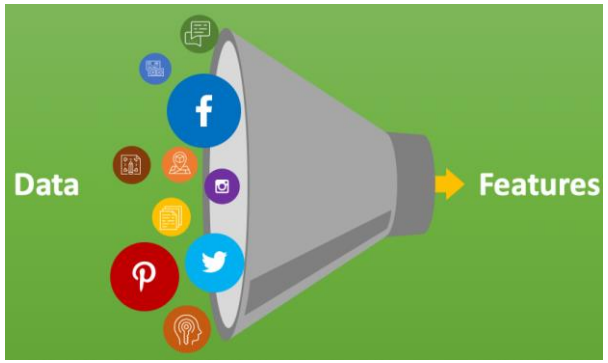
Text Representation

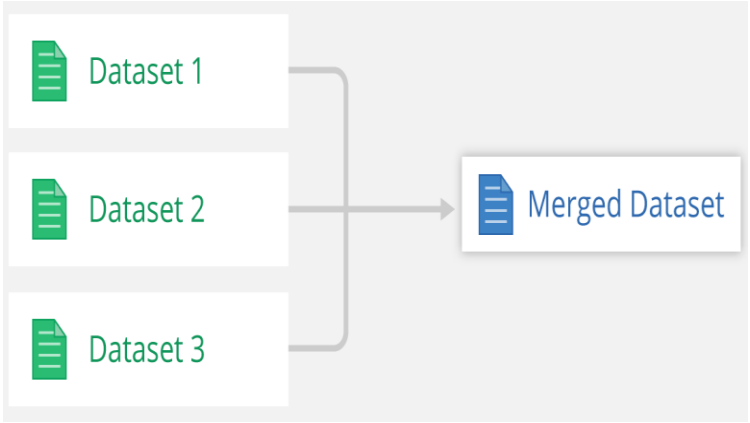
- Process of representing text data in a way that can be easily processed by machines
- Determines how the text data is transformed and processed by machine learning models
- Choice of text representation can significantly impact the performance of these models
- Essential to choose a representation that is appropriate for the task at hand

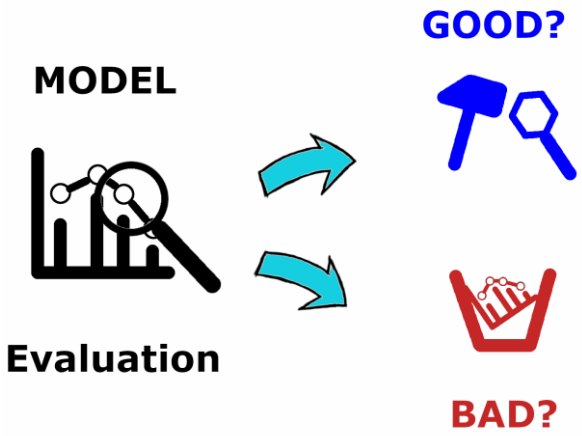
Importance of Text Representation

- Dimensionality reduction
- Feature extraction
- Model performance
- Interpretability
- …







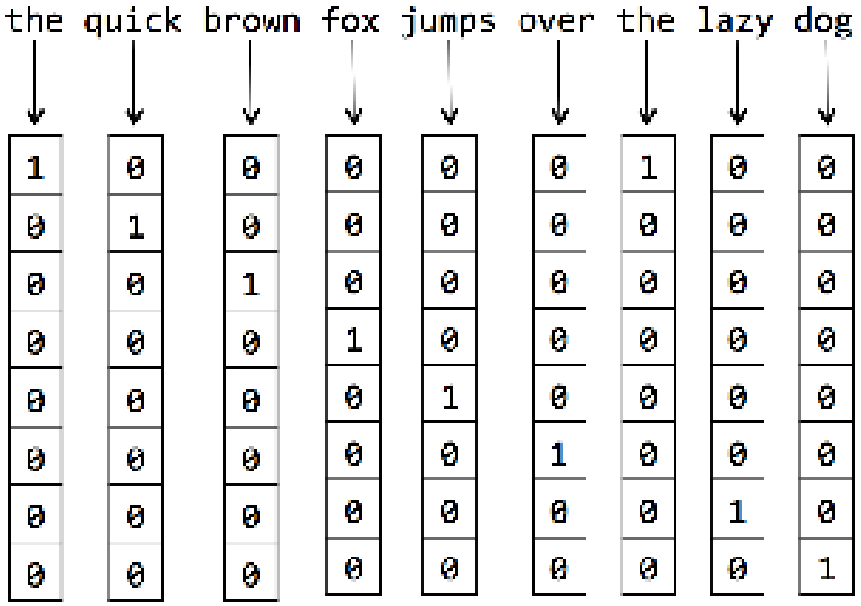


Text Representation Techniques (most common)

- 1-hot encoding
- N-grams
- Bag-of-words
- Vector semantics (tf-idf)
- Distributional semantics (Word2vec, GloVe)
- …

1-hot Encoding

- Each word in a vocabulary is represented as a vector of zeros and ones
- The length of the vector is equal to the size of the vocabulary



- Associate each unique word with an index in this vector.
- To represent a unique word:
  - set the component of the vector to be 1
  - zero out all of the other components

1-hot Encoding

- Is a binary representation method
- It only captures whether a word is present or not in a text, but not the frequency or order of occurrence
- Size of the vector grows linearly with the size of the vocabulary - impractical for large vocabularies
- Useful for many text classification and information retrieval tasks, particularly those that involve small vocabularies and sparse data.

N-gram Models

- Involves breaking down a sequence of words into contiguous sequences of *n* words.
- In other words, an n-gram is a sequence of *n* consecutive words
- N-gram language models estimate the probability of the last word given the previous words
- Useful for tasks, such as text classification, language modeling, and information retrieval.
- Capture local patterns and dependencies between adjacent words in a text.
- Use of n-grams requires careful consideration of the choice of n.
  - Using small values of n can lead to oversimplification and loss of information,
  - Using large values of n can result in sparse and high-dimensional representations, which can be computationally expensive and prone to overfitting

N-gram Models

"The quick brown fox jumps over the lazy dog.“

- Unigrams (n=1): ["The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog."]
- Bigrams (n=2): ["The quick", "quick brown", "brown fox", "fox jumps", "jumps over", "over the", "the lazy", "lazy dog."]
- Trigrams (n=3): ["The quick brown", "quick brown fox", "brown fox jumps", "fox jumps over", "jumps over the", "over the lazy", "the lazy dog."]
- Four-grams (n=4): ["The quick brown fox", "quick brown fox jumps", "brown fox jumps over", "fox jumps over the", "jumps over the lazy", "over the lazy dog."]

Bag of Words

- Represents a piece of text as a bag of its individual words
- Ignore their grammatical structure and word order
- Keep track of their frequency of occurrence
- Typically involves the following steps:
  - Tokenization: The text is first split into individual words or tokens using a tokenizer, which may involve removing stop words and punctuation.
  - Vocabulary creation: A vocabulary of all unique words (tokens) in the text corpus is created, and each word is assigned a unique index.
  - Vectorization: Each text document is represented as a vector of the same length as the vocabulary size. The value of each element in the vector represents the frequency of the corresponding word in the text document.

Bag of Words

"The quick brown fox."
"Jumped over the lazy dog."
"The quick brown fox jumped over the lazy dog.“

- The bag-of-words representation of these sentences might look like this:

  Vocabulary: [The, quick, brown, fox, jumped, over, lazy, dog]

  Sentence 1: [1, 1, 1, 1, 0, 0, 0, 0]
  Sentence 2: [1, 0, 0, 0, 1, 1, 1, 1]
  Sentence 3: [2, 1, 1, 1, 1, 1, 1, 1]

Bag of Words

- Useful for many NLP and IR tasks, such as text classification, document clustering, and information retrieval.
- Limitations:
  - Inability to capture word order and context and the resulting sparsity of the vector representations.

TF-IDF

- Term Frequency-Inverse Document Frequency
- Addresses the limitations of the bag-of-words representation
- Takes into account both the frequency of occurrence of a word in a document (TF) and its inverse frequency in the entire corpus (IDF).

TF-IDF

- The TF-IDF weight for a word w in a document d is calculated as follows:

$$\text{TF-IDF}(w,d) = \text{TF}(w,d) * \text{IDF}(w)$$

- Where,
  - TF(w,d) = Number of times term w appears in document d
  - IDF(w) = log(N / n), where N is the total number of documents in the corpus, and n is the number of documents containing word w.

- The IDF component penalizes words that occur frequently in the corpus and assigns a higher weight to words that are rare but highly relevant to a specific document.
- The logarithmic scaling of the IDF value helps to reduce the impact of very rare or very common words.

TF-IDF

Document 1: "The quick brown fox jumps over the lazy dog."

Document 2: "The brown fox is quick."

Document 3: "The lazy dog is slow."

- The TF-IDF weight for each word in each sentence might look like this:

Word:	The	quick	brown	fox	jumps	over	lazy	dog	is	slow
Sentence 1:	0,	0.176,	0.176,	0.176,	1.099,	1.099,	0.176,	0.176,	0,	0]
Sentence 2:	0,	0.176,	0.176,	0.176,	0,	0,	0,	0,	0.176,	0]
Sentence 3:	0,	0,	0,	0,	0,	0,	0.176,	0.176,	0.176,	1.099]