

# **Applied Text Mining, from Foundations to Advanced**

*Ayoub Bagheri*



# ChatGPT



Quiz me on  
world capitals



Message to  
comfort a friend



Activities to make  
friends in new city



Pick outfit to look  
good on camera

# Foundational Models

Answer this riddle: If eleven plus two equals one, what does nine plus five equal?

2019 AI ⓘ  
GPT-2 ~\$4.6k

✗ Incorrect

The answer is nine plus five.

2020 AI ⓘ  
GPT-3 ~\$690k

✗ Incorrect

The riddle is a play on words.  
The answer is 'four'.

2022 AI ⓘ  
GPT-3.5

✗ Incorrect

💬 Therefore, the answer to  
the riddle is that "nine plus  
five" equals 5.

2023 AI ⓘ  
GPT-4 ~\$50m

✓ Correct

💬 If we start at 11 o'clock  
and add two hours, we get 1  
o'clock.

So, if we start at 9 o'clock and  
add five hours, we get 2  
o'clock. ...

# ChatGPT

how many words are there in your answer?

This answer contains **8 words**:

"This answer contains 8 words: [...]"



why 8 words?

# ChatGPT

how many words are there in your answer?

This answer contains 8 words:

"This answer contains 8 words: [...]"



Great question! Let's break it down.

The sentence:

| "This answer contains 8 words: [...]"

can be counted word by word:

1. This
2. answer
3. contains
4. 8
5. words
6. (the colon counts as punctuation, not a word)
7. [...] — This is an ellipsis (three dots) in brackets, typically standing for omitted content, not a literal word or group of words.

So technically, the full sentence only has **5 actual words before the colon**, and the part after the colon (**[ ... ]**) is not a word.

Therefore, a more accurate version would be:

| "This answer contains 5 words."

And that is a 5-word sentence.

Apologies for the earlier confusion — I got caught in the recursion! Want to try a meta version that actually works?

# Today

- Introduction
- Examples
- Applications
- Regular expressions

# Introduction

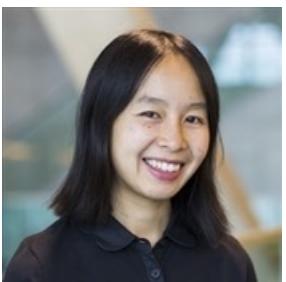


**Ayoub Bagheri**

Associate Professor

NLP, Machine learning

[a.bagheri@uu.nl](mailto:a.bagheri@uu.nl)



**Dong Nguyen**

Associate Professor

NLP, Computational sociolinguistics

[d.p.nguyen@uu.nl](mailto:d.p.nguyen@uu.nl)



**Pablo Mosteiro**

Assistant Professor

NLP, Applied data science

[p.j.mosteiroromero@uu.nl](mailto:p.j.mosteiroromero@uu.nl)

# Introduction

**Participants!**

# Course website

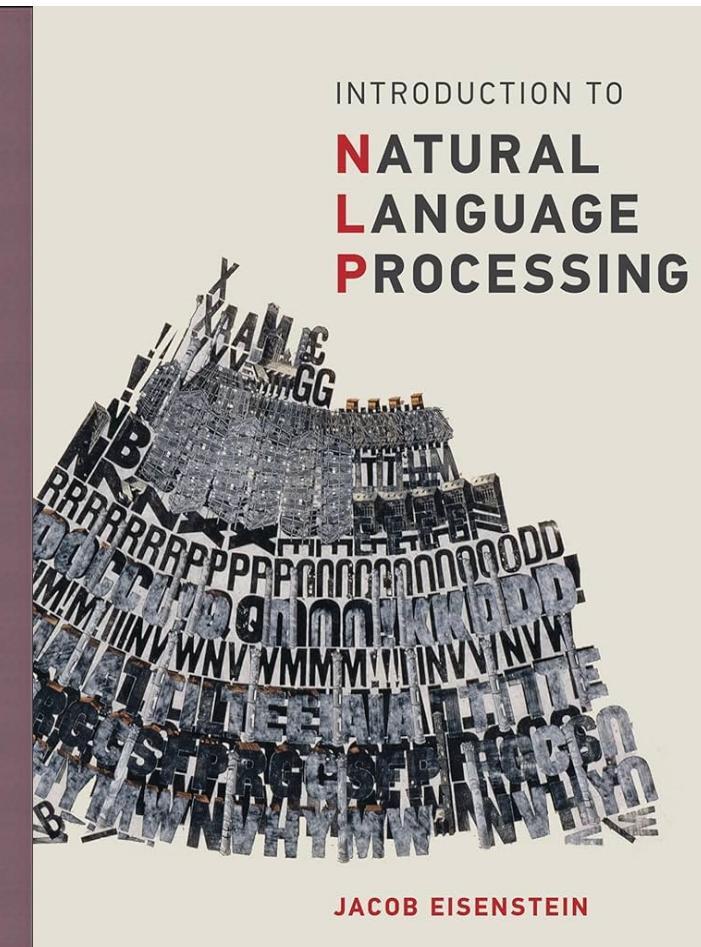
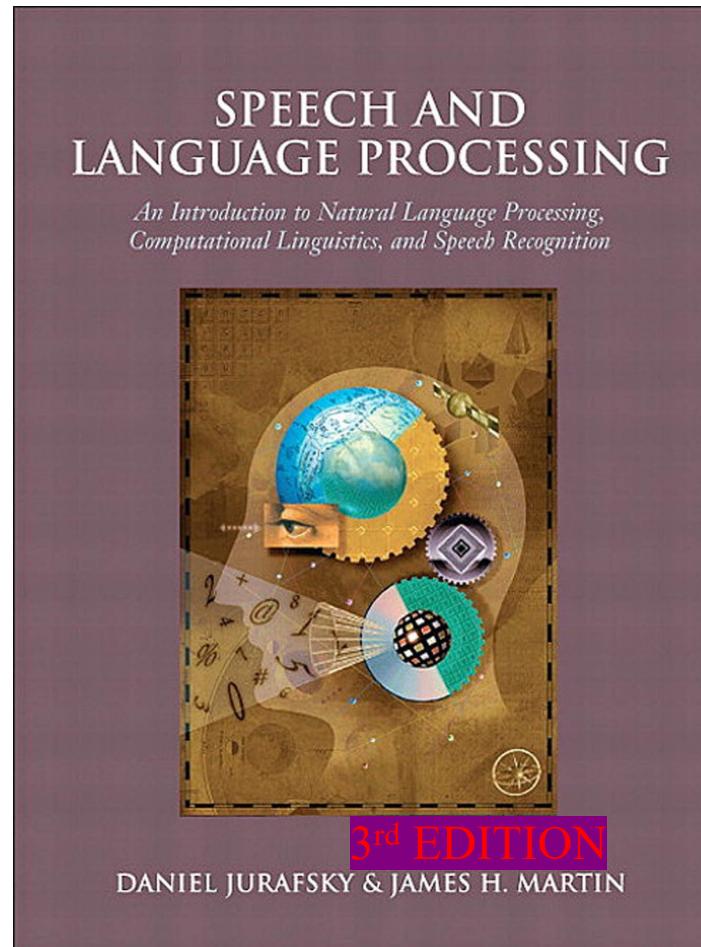
[https://ayoubbagheri.nl/applied\\_tm/](https://ayoubbagheri.nl/applied_tm/)

# Course proceedings

Daily schedule		
Start time	End time	Type
09:00	10:30	Lecture
		<b>Break</b>
10:50	12:00	Practical
12:00	12:30	Discussion
<u><a href="#">Lunch at Vening Meinesz building (A)</a></u>		
14:00	15:20	Lecture
		<b>Break</b>
15:30	16:30	Practical
16:30	17:00	Discussion

# Readings about text mining

- Jurafsky & Martin (2025). *Speech and Language Processing*.  
<https://web.stanford.edu/~jurafsky/slp3/>
- Eisenstein (2019). *Natural Language Processing*.  
<https://github.com/jacobeisenstein/gt-nlp-class/blob/master/notes/eisenstein-nlp-notes.pdf>



**Read the materials online!**  
**We adapt the course as we go!**

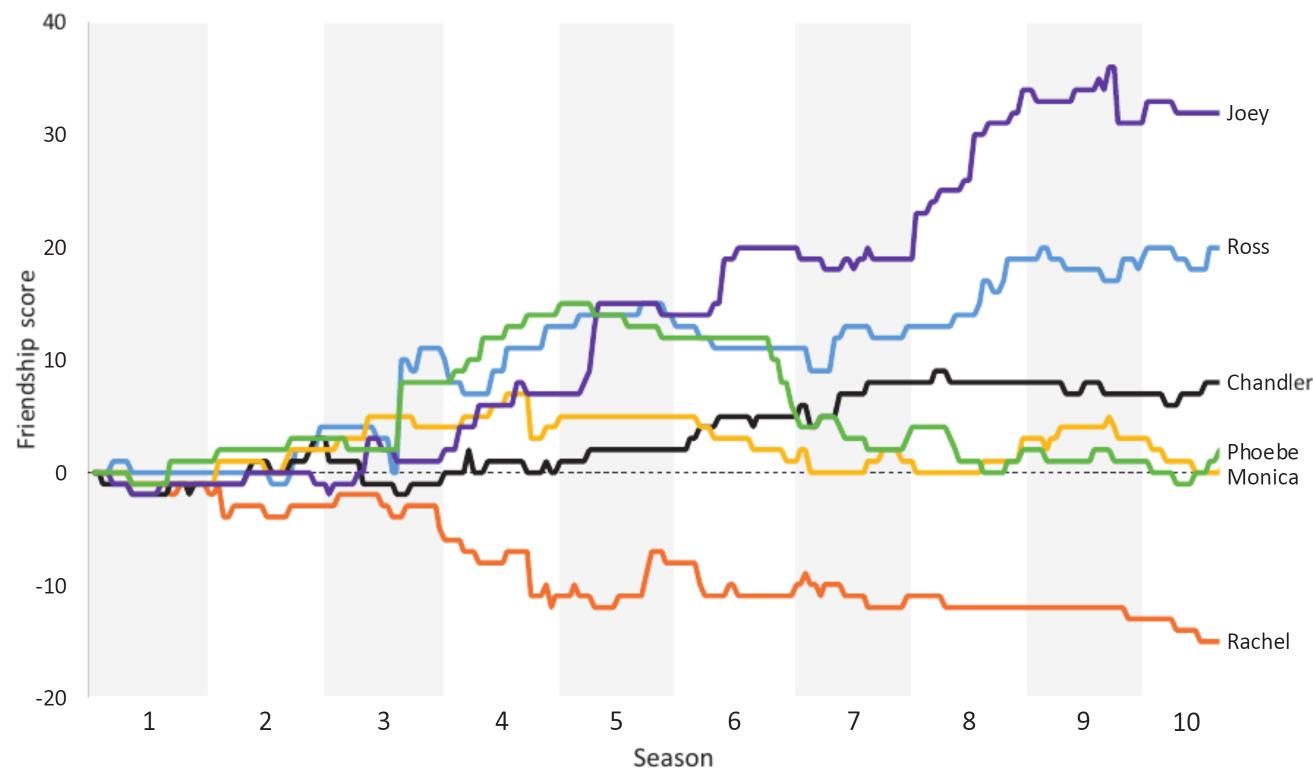
# Text Mining

# Text mining: definition 1

- “The discovery by computer of **new, previously unknown information**, by automatically extracting information from different written resources”, Hearst (1999).

# Example 1: Who was the best Friend?

<https://rss.onlinelibrary.wiley.com/doi/epdf/10.1111/1740-9713.01574>



# Text mining: definition 2

- Text mining is about looking for **patterns in text**, in a similar way that data mining can be loosely described as looking for patterns in data.

# Example 2: Did a poet with donkey ears write the oldest anthem in the world?

<https://dh2017.adho.org/abstracts/079/079.pdf>

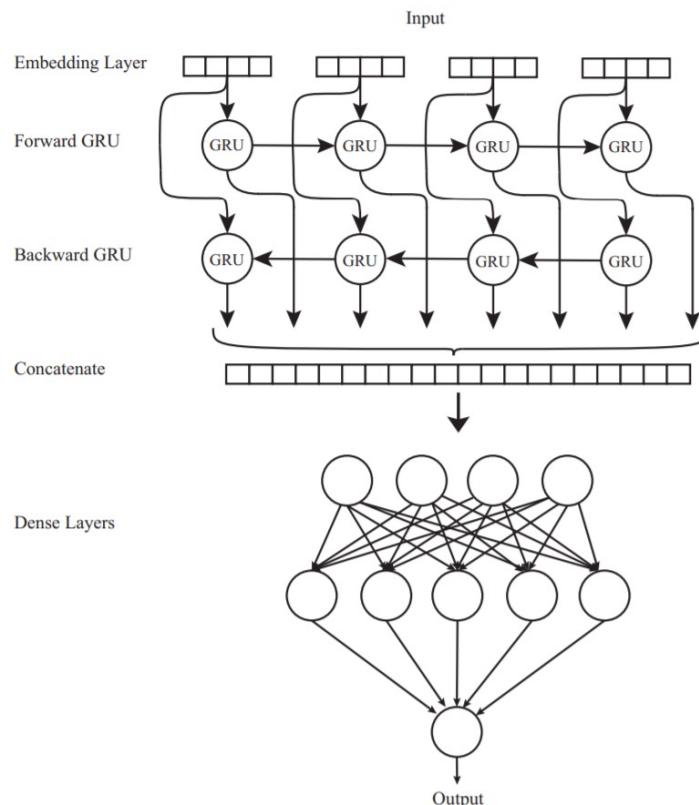


# Text mining: definition 3

- Text mining describes a set of linguistic, statistical, and machine learning techniques that model and **structure the information content of textual sources.** (Wikipedia)

# Example 3: Automatic detection of disease codes in cardiology discharge letters

<https://www.nature.com/articles/s41746-021-00404-9>



**Box 1:** An example of a Dutch discharge letter from the dataset

Bovengenoemde patiënt was opgenomen op <DATUM-1> op de <PERSOON-1> voor het specialisme Cardiologie.

**Reden van opname** STEMI inferior

**Cardiale voorgeschiedenis.** Blanco

**Cardiovasculaire risicofactoren:** Roken(-) Diabetes(-) Hypertensie(?) Hypercholesterolemie (?)

**Anamnese.** Om 18.30 pijn op de borst met uitstraling naar de linkerarm, zweten, misselijk. Ambulance gebeld en bij aansluiten monitor beeld van acuut onderwandinfarct.

AMBU overdracht. 500 mg aspetic iv, ticagrelor 180 mg oraal, heparine, zofran eenmalig, 3x NTG spray. HD stabiel gebleven. Medicatie bij presentatie. Geen.

**Lichamelijk onderzoek.** Grauw, vegetatief, Halsvenen niet gestuwd. Cor s1 s2 geen souffles. Pulm schoon. Extr warm en slank.

**Aanvullend onderzoek.** AMBU ECG: Sinusritme, STEMI inferior III/II C/vermoedelijk RCA.

Coronair angiografie. (...). Conclusie angio: 1-vatslijden..PCI

**Conclusie en beleid**

Bovengenoemde <LEEFTIJD-1> jarige man, blanco cardiale voorgeschiedenis, werd gepresenteerd vanwege een STEMI inferior waarvoor een spoed PCI werd verricht van de mid-RCA. Er bestaan geen relevante nevenletselens. Hij kon na de procedure worden overgeplaatst naar de CCU van het <INSTELLING-2>...Dank voor de snelle overname...Medicatie bij overplaatsing. Acetylsalicyzuur dispersable tablet 80 mg; oraal; 1x per dag 80 milligram; <DATUM-1>. Ticagrelor tablet 90 mg; oraal; 2x per dag 90 milligram; <DATUM-1>. Metoprolol tablet 50 mg; oraal; 2x per dag 25 milligram; <DATUM-1>. Atorvastatine tablet 40 mg (als ca-zout-3-water); oraal; 1x per dag 40 milligram; <DATUM-1>

**Samenvatting**

Hoofddiagnose: STEMI inferior wv PCI RCA. Geen nevenletselens. Nevendiagnoses: geen.

**Complicaties:** geen Ontslag naar: CCU <INSTELLING-2>.

# ChatGPT

A You

What is Text Mining?



ChatGPT

Text mining, also known as text data mining or text analytics, is the process of extracting meaningful information and insights from large amounts of unstructured text data. This process involves a combination of techniques from linguistics, data mining, and computer science, particularly natural language processing (NLP). The main steps in text mining typically include:

1. **Text Collection:** Gathering a large corpus of text from various sources such as books, articles, websites, social media, etc.
2. **Preprocessing:** Cleaning and organizing the text. This step may include removing irrelevant data (like HTML tags), converting text to a uniform format, eliminating stop words (common words that don't contribute much meaning), and reducing words to their base or root form through stemming or lemmatization.

# Why text mining?

- **Text data is everywhere**, websites (e.g., news), social media (e.g., twitter), databases (e.g., doctors' notes), digital scans of printed materials, ...
- A lot of world's data is in **unstructured text format**
- **Text is high dimensional**
- Applications in industry: search, machine translation, sentiment analysis, question answering, ...
- Applications in science: cognitive modeling, understanding bias in language, automated systematic literature reviews, ...

## High dimensional data

- All possible words & phrases

## Complex & subtle relationships in text

- "Jumbo merges with Hema"
- "Jumbo is bought by Hema"

## Ambiguity & context sensitivity

- car = automobile = vehicle
- kapsalon (hairdresser) or kapsalon (fast food)

## Homographs: same words can mean different things

- Bat (sports, animal, ...)

## Synonyms

## Misspellings

## Abbreviations

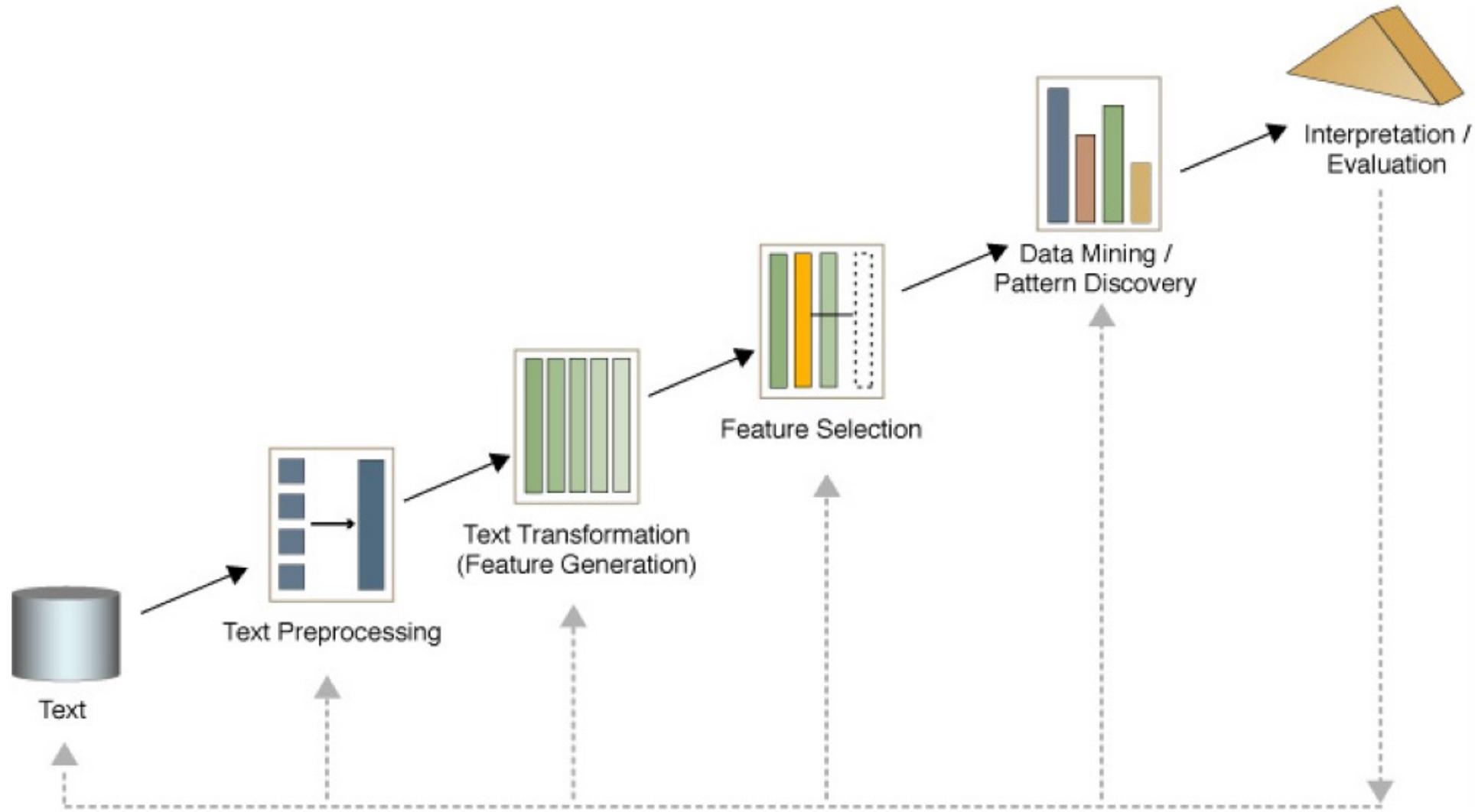
## Negations

## Spelling variations

## LANGUAGE!

**What to do with the TM challenges?  
We will answer this question during the course.**

# Text mining process



# Text mining process

- **Data: Text**
- **Text Preprocessing:** is the process of cleaning, normalizing, and structuring raw text data into a format suitable for analysis or input into NLP models.
- **Text transformation, feature generation:** involves converting text data into a different format or structure, such as numerical vectors or simplified forms, to make it suitable for analysis or modeling.
- **Feature selection:** is the process of identifying and selecting the most relevant features from a dataset to improve model performance and reduce complexity.
- **Data mining, pattern discovery:** is the process of extracting meaningful patterns and knowledge from text.
- **Interpretation / Evaluation:** is the process of understanding and explaining the model and patterns / is the assessment process to measure performance and quality.

# Many tasks

- Feature selection
- Text classification
- Text clustering
- Topic modelling
- Sentiment analysis
- Responsible text mining
- Bias detection
- Text summarization
- Translation
- Automatic reviewing: ASReview
- Chatbot design (conversational AI)
- And more..

# Regular Expressions

You can design a chatbot at the end of the course,  
but we need to start with the basics!

# Regular expressions (regex)

- **Powerful** and a **very useful** tool for text (pre)processing
- Used in pretty much every pipeline involving text
- Typical applications:
  - Extracting numbers, emails, IP-addresses, etc.
  - Validating text inputs in GUIs
  - Reformatting annoying incorrect dates (everything not yyyy-mm-dd)
  - Scrubbing names and addresses for pseudonimization
- **Cryptic** & takes a lot of practice!

# Basic matching using the ‘re’ library

- Find all occurrences of a pattern in the string.

```
text = "cat, bat, rat, mat"
matches = re.findall(r"\b[a-z]at\b", text)
print("Matches:", matches) # Output: ['cat', 'bat', 'rat', 'mat']
```

# Basic matching using the ‘re’ library

- Replace a pattern with a new string.

```
text = "The rain in Spain"  
result = re.sub(r"ain", "123", text)  
print("Result:", result) # Output: The r123 in Sp123
```

# Basic matching using the ‘re’ library

- **Anchors: Start and End of String**

- By default, regular expressions will match any part of a string. It’s often useful to **anchor** the regular expression so that it matches from the start or end of the string. You can use:
  - `^` matches the start of the string.
  - `$` matches the end of the string

```
text = "Hello, world!"  
print(bool(re.search(r"^Hello", text))) # Output: True  
print(bool(re.search(r"world!$", text))) # Output: True
```

# How to write regular expressions

# 1. Simple regex searches

RE	Example Patterns Matched
/woodchucks/	“interesting links to <u>woodchucks</u> and lemurs”
/a/	“Mary Ann stopped by Mona’s”
/!/	“You’ve left the burglar behind again!” said Nori

## 2. Disjunction (OR)

The use of the brackets [] to specify a disjunction of characters:

RE	Match	Example Patterns
/[wW]oodchuck/	Woodchuck or woodchuck	“ <u>Woodchuck</u> ”
/[abc]/	‘a’, ‘b’, <i>or</i> ‘c’	“In uomini, in soldat <i>i</i> ”
/[1234567890]/	any digit	“plenty of <u>7</u> to 5”

The pipe symbol | is also for disjunction:

Pattern	Matches
groundhog woodchuck	
yours mine	yours mine
a b c	= [abc]

# 3. Brackets and dash

The use of the brackets [] plus the dash - to specify a range:

<b>RE</b>	<b>Match</b>	<b>Example Patterns Matched</b>
/[A-Z]/	an upper case letter	“we should call it ‘ <u>Drenched Blossoms</u> ’ ”
/[a-z]/	a lower case letter	“ <u>my</u> beans were impatient to be hoed!”
/[0-9]/	a single digit	“Chapter <u>1</u> : Down the Rabbit Hole”

# 4. Negation

The caret ^ for negation or just to mean ^:

RE	Match (single characters)	Example Patterns Matched
/[^A-Z]/	not an upper case letter	“Oyfn pripetchik”
/[^Ss]/	neither ‘S’ nor ‘s’	“I have no exquisite reason for’t”
/[^.]/	not a period	“our resident Djinn”
/[e^]/	either ‘e’ or ‘^’	“look up ^ now”
/a^b/	the pattern ‘a^b’	“look up a^ b now”

# 5. Question and period marks

The question mark ? marks optionality of the previous expression:

RE	Match	Example Patterns Matched
/woodchucks?/	woodchuck or woodchucks	“ <u>woodchuck</u> ”
/colou?r/	color or colour	“ <u>color</u> ”

The use of the period . to specify any character:

RE	Match	Example Matches
/beg.n/	any character between <i>beg</i> and <i>n</i>	<u>begin</u> , <u>beg'n</u> , <u>begun</u>

# 6. Anchors

RE	Match
^	start of line
\\$	end of line
\b	word boundary
\B	non-word boundary

# 7. Common sets

Aliases for common sets of characters:

RE	Expansion	Match	First Matches
\d	[0-9]	any digit	Party_of_5
\D	[^0-9]	any non-digit	Blue_moon
\w	[a-zA-Z0-9_]	any alphanumeric/underscore	Daiyu
\W	[^\w]	a non-alphanumeric	!!!
\s	[ \r\t\n\f]	whitespace (space, tab)	
\S	[^\s]	Non-whitespace	in_Concord

The backslash for escaping!

# 8. Operators for counting

Aliases for common sets of characters:

<b>RE</b>	<b>Expansion</b>	<b>Match</b>	<b>First Matches</b>
\d	[0-9]	any digit	Party_of_5
\D	[^0-9]	any non-digit	Blue_moon
\w	[a-zA-Z0-9_]	any alphanumeric/underscore	Daiyu
\W	[^\w]	a non-alphanumeric	!!!!
\s	[ \r\t\n\f]	whitespace (space, tab)	
\S	[^\s]	Non-whitespace	in_Concord

The backslash for escaping!

# 9. Other

Some characters that need to be backslashed:

<b>RE</b>	<b>Match</b>	<b>First Patterns Matched</b>
\*	an asterisk “*”	“K <u>A</u> *P*L*A*N”
\.	a period “.”	“Dr. <u>,</u> Livingston, I presume”
\?	a question mark	“Why don’t they come and lend a hand <u>?</u> ”
\n	a newline	
\t	a tab	

# 10. Operator precedence

Parenthesis

()

Counters

\* + ? {}

Sequences and anchors

the ^my end\$

Disjunction

|

# Examples

## Regex

hello

gray|grey

gr(a|e)y

gr[ae]y

b[aeiou]bble

[b-chm-pP]at|ot

colou?r

## Matches any string that

contains {hello}

contains {gray, grey}

contains {gray, grey}

contains {gray, grey}

contains {babble, bebble, bibble, bobble, bubble}

contains {bat, cat, hat, mat, nat, oat, pat, Pat, ot}

contains {color, colour}

# More complicated examples

## Regex

\d

1\d{10}

\d+(\.\d\d)?

^dog

dog\$

^dog\$

## Matches any string that

contains {0,1,2,3,4,5,6,7,8,9}

contains an 11-digit string starting with a 1

contains a positive integer or a floating point number with exactly two characters after the decimal point.

begins with "dog"

ends with "dog"

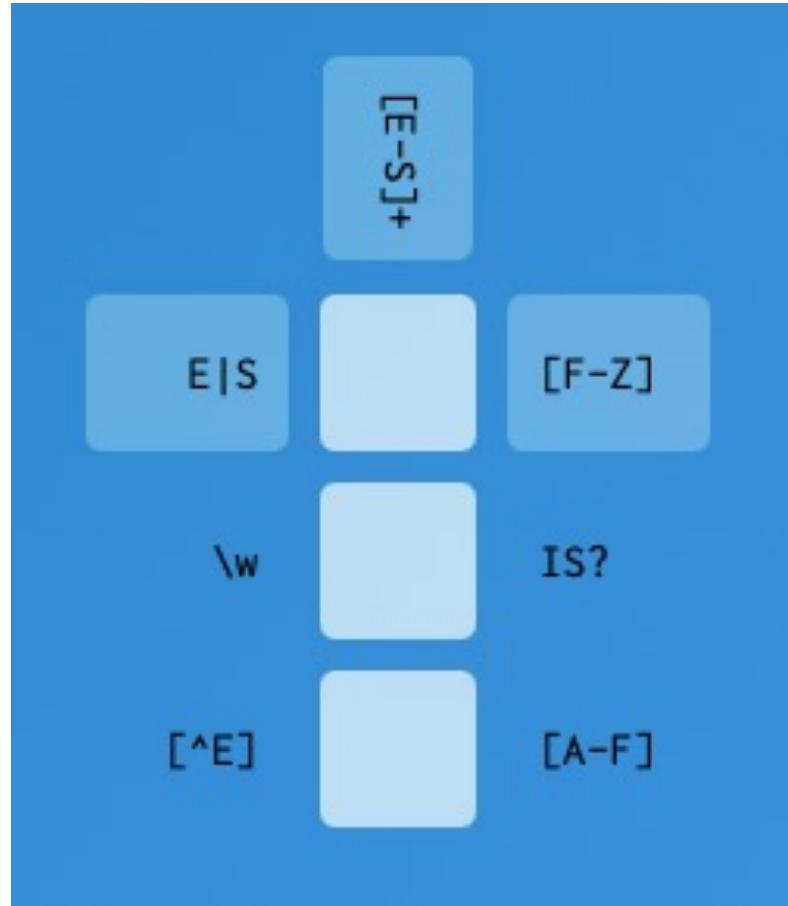
is exactly "dog"

# Regular expression conclusion

- You have now *heard of* regular expressions
- And might have a basic idea of what you might do with them
- The only way to really learn, however, is **practice**
- Next time you encounter some text you need to work on think “can I do this using regular expressions?”
- The answer is probably “yes”.

# Challenge problem: regex crossword

<https://regexcrossword.com/>



# Break

# Text Preprocessing

# Why preprocessing?

- Remove irrelevant information
- Add relevant information
- Convert your text into a suitable data format
- Allocate less memory to with lower dimensional data
- Improve performance

# Typical preprocessing steps

- Lowercasing (“Ruppert 22”→“ruppert 22”)
- Stopword removal (“text ming is **the** best!”)
- Punctuation removal (“text ming is the best!”)
- Number removal (“Ruppert **22**”)
- Spell correction (“text **ming** is the best!”)
- Stemming (“running”→“run”) or Lemmatization (“were”→“is”)
- Tokenization (“text”, “ming”, “is”, “the”, “best”, “!”)
- And more..

**Not all of these are appropriate at all times!**

# Example

Text mining is to identify useful information.

↓ Tokenization

‘Text’, ‘mining’, ‘is’, ‘to’, ‘identify’, ‘useful’, ‘information’, ‘:’

↓ Stemming

‘text’, ‘mine’, ‘is’, ‘to’, ‘identify’, ‘use’, ‘inform’, ‘:’

↓ Bigrams

‘text mine’, ‘mine is’, ‘is to’, ‘to identify’, ‘identify use’, ‘use inform’, ‘inform .’

↓ Stopwords & punctuations

‘text mine’, ‘to identify’, ‘identify use’, ‘use inform’

↓ Vectorization

Vector Space Model

# Tokenization

- Tokenization in Natural Language Processing (NLP) is the process of breaking down a piece of text into smaller units, called **tokens**.
- Tokens are the “units” of a text, words, phrases, sentences, subwords, characters, etc.

# Word tokenization

- Splits text into individual words.
- This is the most common type of tokenization.
  - Example: "I love NLP!" → ["I", "love", "NLP", "!"]
- N-grams
  - N-grams are sequences of **n** consecutive tokens extracted from the text. These provide context by considering adjacent words together.
  - Example for bigrams ( $n=2$ ): "I love NLP" → [("I", "love"), ("love", "NLP")]
  - Example for trigrams ( $n=3$ ): "I love NLP" → [("I", "love", "NLP")]
  - N-grams are useful for capturing local word dependencies, commonly used in tasks like text classification, machine translation, and language modeling.

# Other tokenization

## Sentence Tokenization:

- Splits text into sentences
- Example: "I love NLP. It is amazing." → ["I love NLP.", "It is amazing."]

## Subword Tokenization:

- Splits text into subword units
- Example: "unbelievable" → ["un", "believable"]

## Character Tokenization:

- Splits text into individual characters
- Example: "NLP" → ["N", "L", "P"]

# Purpose of tokenization

- Tokenization is a crucial preprocessing step because most NLP models work with tokens instead of raw text.
- Enables easier text manipulation.
- Compatibility with machine learning models.
- Handling of out-of-vocabulary (OOV) words through subword tokenization.
- Efficient encoding of textual data into numerical formats.

# Challenges in tokenization

- Handling contractions ("can't" → ["ca", "n't"] or ["can", "not"]).
- Splitting compound words ("New York" as one token vs. two tokens).
- Dealing with punctuation and special characters.
- Supporting multilingual text with varying tokenization rules.

# Removing stopwords

- Stopwords are very common words, like the, is, for, etc.
- Stopwords make up a lot of a text, but don't really tell you what a text is about.

- Example



**original**

A powerful portrayal of a totalitarian dystopia, but too dull  
and depressive to really enjoy as a novel.



**stopwords removed**

powerful portrayal totalitarian dystopia, dull depressive  
really enjoy novel.

# Removing stopwords

- To remove stopwords, we need a list of words to remove.
- You can:
  - Use a list assembled for your language or task
  - Make a list from your data, e.g. the 100 most common words

# Stemming & lemmatisation

The following words are quite similar:

*political, politically, politics, politician,  
politicians, politicise, politicised, politicising*

We can simplify our further analysis by saying that these are all variations of *politic-*

# Stemming

Stemming strips words and only keeps the **stem**

Examples:

- *time* -> *tim*
- *times* -> *tim*
- *timing* -> *tim*

# Lemmatisation

Lemmatisation removes inflection

Examples:

- *times* -> *time*
- *timing* -> *time*

Unlike stemming, lemmatising tries to give complete words as output, rather than stems

# Stemming & lemmatisation



## Example

Cleverly crafted cases, a plucky problem solver, and a unique backdrop. A delightful way to relax with a book in the evening.



## stemmed

clever craft case, a plucki problem solver, and a uniqu backdrop. a delight way to relax with a book in the even.



## lemmatised

cleverly craft case, a plucky problem solver, and a unique backdrop. a delightful way to relax with a book in the evening.

# Stemming & lemmatisation

- Lemmatising preserves more information
- Stemming is less "careful": it may be more effective
- Lemmatising results in readable output
- Stemming requires fewer resources

# Part-of-speech (POS) tagging

**Parts of speech (POS)** are types of words (nouns, verbs, etc.)



## Example

New setting for me. And all the characters are well developed, all flawed. I got lost in it.



## POS tags

New/ADJ setting/NOUN for/ADP me/PRON ./PUNCT And/CCONJ all/PRON the/DET characters/NOUN are/AUX well/ADV developed/VERB, all/PRON flawed/VERB ./PUNCT I/PRON got/VERB lost/VERB in/ADP it/PRON ./PUNCT

# Part-of-speech (POS) tagging

- Remove ambiguity: I **walk** to the bus stop vs a **walk** in the park
- Help to parse sentence structure
- Search for particular structures

# Language detection

For mixed-language data, categorise the language before moving further

A lot of other preprocessing steps are language-dependent!

# Named Entity Recognition

Tag all **named entities**: names of people, places, organisations, etc.

The output of NER can be used to answer questions like:

- Which people get mentioned in a lot of book reviews?
- How many reviews mention a particular place?

In other cases, we are not interested in names, and use NER to filter them out.

# Preprocessing choices

- There is no single "correct" method for preprocessing.
- Each of these steps can be helpful or detrimental depending on the task.
- Important factors:
  - Language
  - Type of text
  - Task
  - Amount of data
  - Analysis method

# Preprocessing choices

- We try to balance:
  - preserving information versus removing noise
  - maximising the informativity or likelihood of observations
- As a general rule:
  - With small dataset and simple models, your analysis will struggle to handle noisy data.
  - A large model with lots of data will benefit from richer input.

We can stop here and work on  
lab/practical 1 until Q19.

# **Text Representation**

## **Vector Space Model**

# Basic idea

- Text is “unstructured data”
- How do we get to something structured that we can compute with?
- **Text must be represented somehow**
- Represent the text as something that makes sense to a computer

# Vector space model

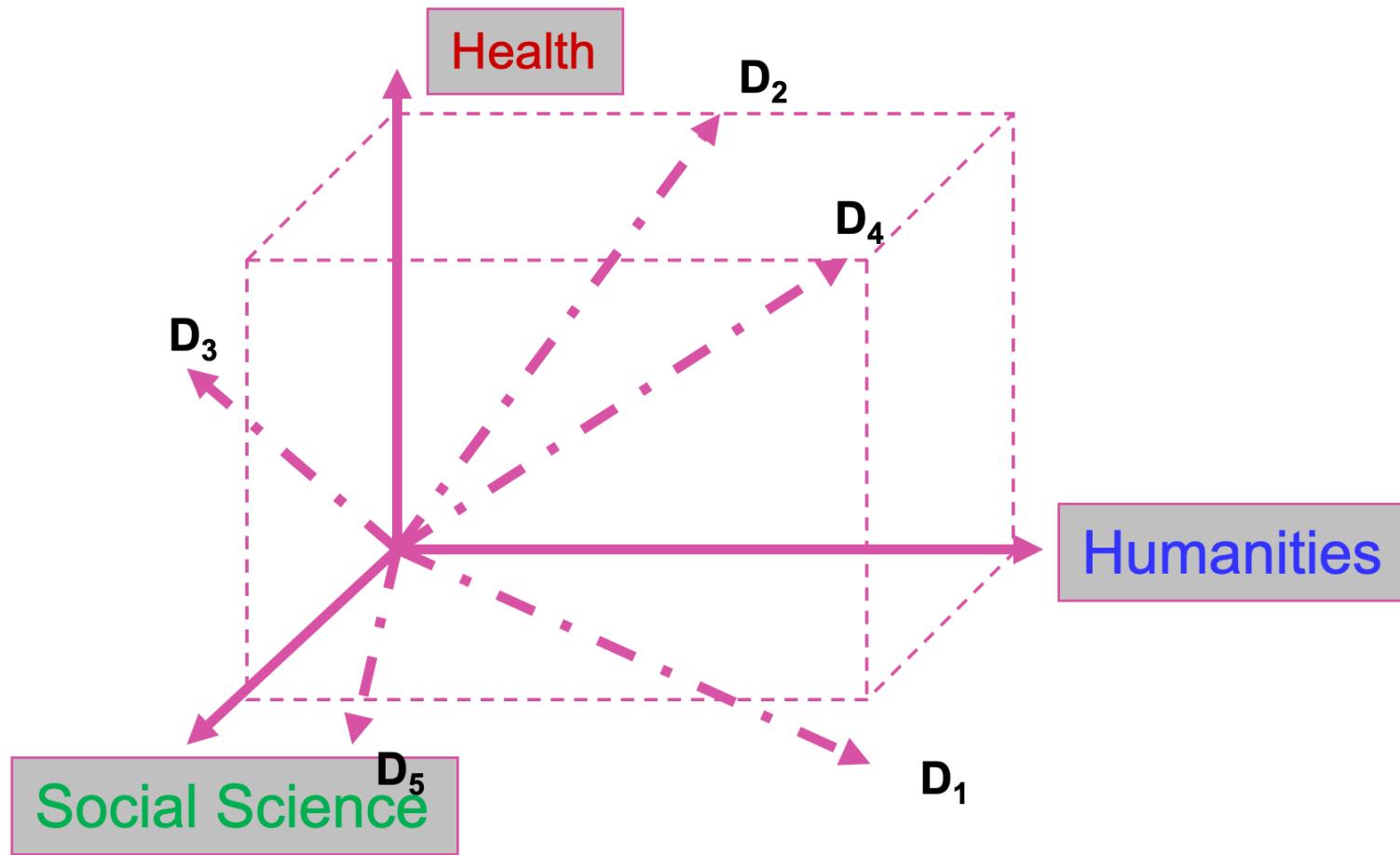
- A vector space is a collection of vectors
- Represent documents by concept vectors
  - Each concept defines one dimension
  - $k$  concepts define a high-dimensional space
  - Element of vector corresponds to concept weight

# Vector space model

- Terms / words are generic features that can be extracted from text
- Typically, terms are single words, keywords, n-grams, or phrases
- Documents are represented as vectors of terms
- Each dimension (concept) corresponds to a separate term

$$d = (w_1, \dots, w_n)$$

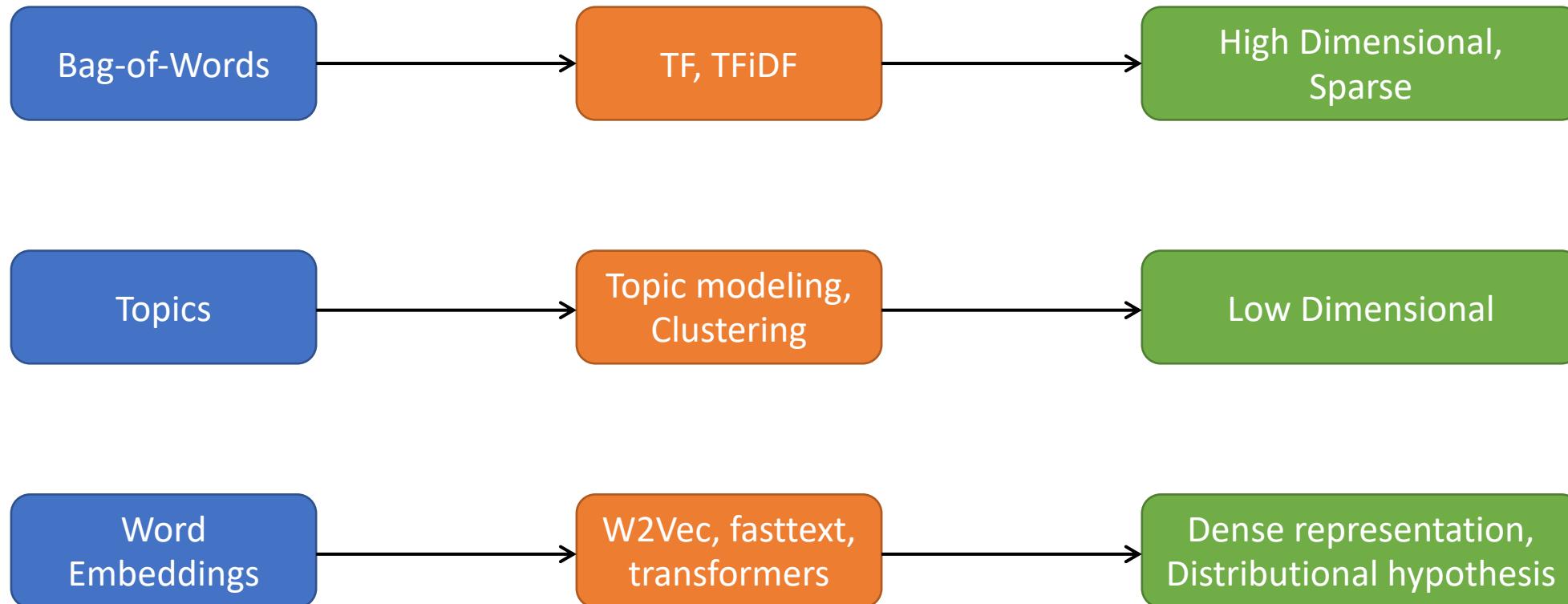
# An illustration



# Vectorization

- The process of converting text into numbers is called **Vectorization**
- Distance between the vectors in this concept space
  - Relationship among documents

# VSM representations



# Bag-of-Words

- **Terms** are words (more generally we can use n-grams)
- **Weights** are number of occurrences of the terms in the document
  - Binary
  - Term Frequency (TF)
  - Term Frequency inverse Document Frequency (TFiDF)

# Example

Doc1: Text mining is to identify useful information.

Doc2: Useful information is mined from text.

Doc3: Apple is delicious.

Document-Term matrix (DTM):

	text	information	identify	mining	mined	is	useful	to	from	apple	delicious
Doc1	1	1	1	1	0	1	1	1	0	0	0
Doc2	1	1	0	0	1	1	1	0	1	0	0
Doc3	0	0	0	0	0	1	0	0	0	1	1

# Term Frequency (TF)

- A measure to evaluate how often a word (or term) appears in a document relative to the total number of words in that document.
- TF helps in ranking terms based on their prominence within a document.
  - **Formula:**

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

- **Example:**
  - Document: "I love NLP. NLP is amazing."
  - Term: "NLP"
    - Frequency of "NLP" = 2
    - Total terms = 6
    - $TF(\text{NLP}) = \frac{2}{6} = 0.33$

# TF-IDF

**TF-IDF** combines **TF** with **Inverse Document Frequency (IDF)** to assess how important a term is within a document relative to an entire corpus/dataset.

- Formula:

$$TF-IDF(t) = TF(t) \times IDF(t)$$

$$IDF(t) = \log \left( \frac{\text{Total number of documents}}{\text{Number of documents containing term } t} \right)$$

- Example:

- Corpus: 3 documents:

- Doc1: "I love NLP"

- Doc2: "NLP is amazing"

- Doc3: "AI and NLP are evolving"

- Term: "NLP"

- $TF(\text{NLP in Doc1}) = \frac{1}{3}$

- $IDF(\text{NLP}) = \log \left( \frac{3}{3} \right) = 0$  (common in all documents, so it's less unique)

- $TF-IDF(\text{NLP in Doc1}) = \frac{1}{3} \times 0 = 0$

# What to do with the vectors?

- We can check documents' similarity!
  - Cosine similarity

$$\text{Cosine Similarity} = \frac{\mathbf{D}_1 \cdot \mathbf{D}_2}{\|\mathbf{D}_1\| \|\mathbf{D}_2\|}$$

Where:

- $\mathbf{D}_1$  and  $\mathbf{D}_2$  are the term frequency (TF) or TF-IDF vectors for two documents.
- $\mathbf{D}_1 \cdot \mathbf{D}_2$ : The dot product of the vectors.
- $\|\mathbf{D}_1\|$ : Magnitude of the first document vector.
- $\|\mathbf{D}_2\|$ : Magnitude of the second document vector.

# What to do with the vectors?

- Doc<sub>1</sub> : "IloveNLP"
- Doc<sub>2</sub> : "NLPisamazing"

## 1. TF Representation:

- Vectors (after term extraction):

$$\mathbf{D}_1 = [1, 1, 1, 0, 0]$$

$$\mathbf{D}_2 = [0, 0, 1, 1, 1]$$

- Cosine Similarity:

$$\text{Cosine Similarity} = \frac{1 \times 0 + 1 \times 0 + 1 \times 1 + 0 \times 1 + 0 \times 1}{\sqrt{1^2 + 1^2 + 1^2 + 0^2 + 0^2} \times \sqrt{0^2 + 0^2 + 1^2 + 1^2 + 1^2}}$$

## 2. TF-IDF Representation:

- Vectors reflect term importance:

$$\mathbf{D}_1 = [0, 0, 0.58, 0.41, 0.41]$$

$$\mathbf{D}_2 = [0.41, 0.41, 0.58, 0, 0]$$

# What to do with the vectors?

- **We can check documents' similarity!**
  - **Document Similarity:**
    - Used in search engines to rank documents based on their relevance to a query.
    - Example: Comparing a user's query with a corpus of documents to find the most relevant ones.
  - **Plagiarism Detection:**
    - Compares textual content to identify overlapping or similar phrases and ideas.
  - **Clustering and Classification:**
    - Groups similar documents or sentences in tasks like topic modeling and text categorization.
  - **Recommendation Systems:**
    - Finds content (e.g., articles, products) similar to a given input based on textual metadata.

# Conclusion

- ❖ Text is everywhere!
- ❖ Text Mining and NLP is the topic of the day!
- ❖ Regular expression is a good start.
- ❖ **Text Preprocessing:** is the process of cleaning, normalizing, and structuring raw text data into a format suitable for analysis or input into NLP models.
- ❖ **Vector Space Model is the common way for text representation.**
- ❖ **Cosine similarity** is a fundamental metric in NLP, enabling the comparison of textual data.

# Questions?