ALGONQUIN COLLEGE

CST8507: NATURAL LANGUAGE PROCESSING

LECTURE#3

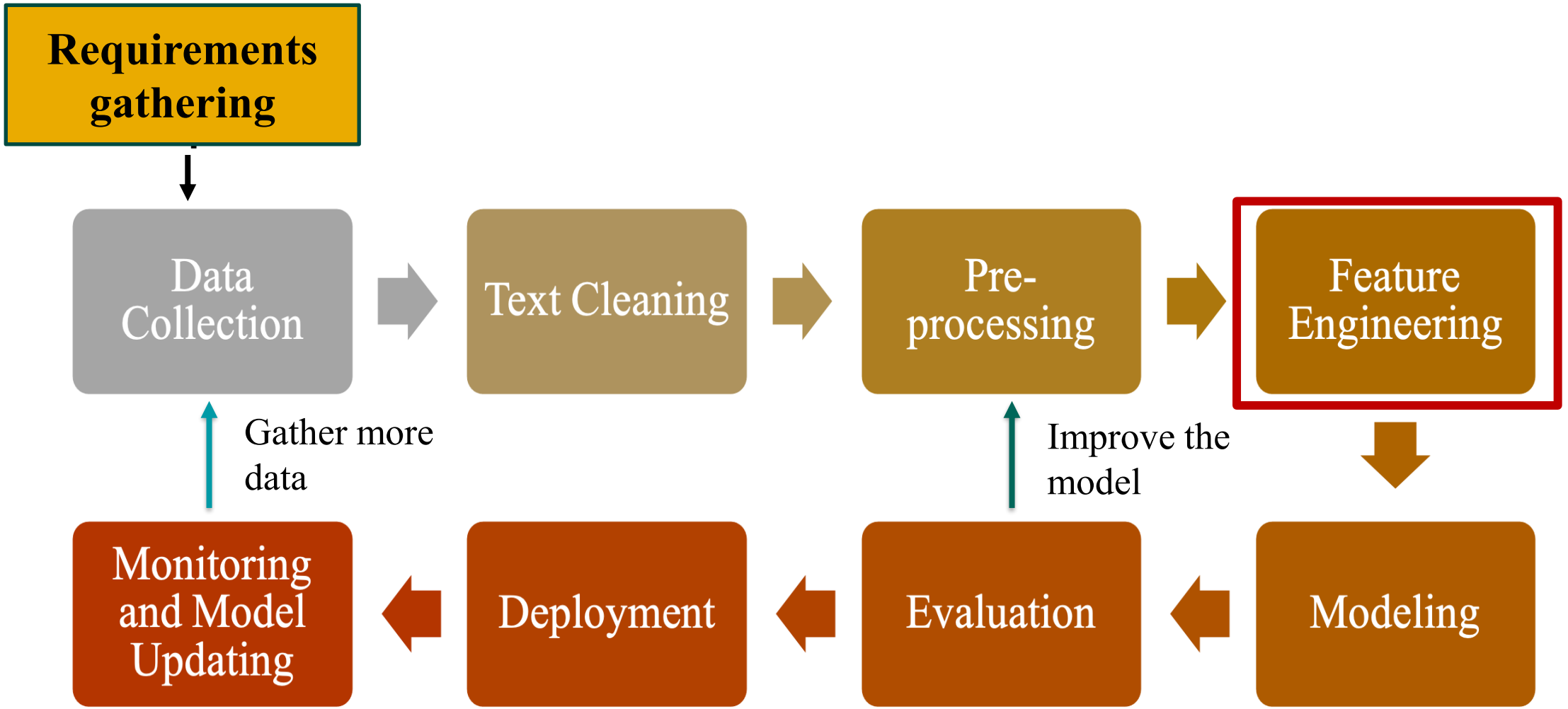
TEXT VECTORIZATION & SIMILARITY

DEVELOPED BY
HALA OWN, PH.D.

Lesson Agenda

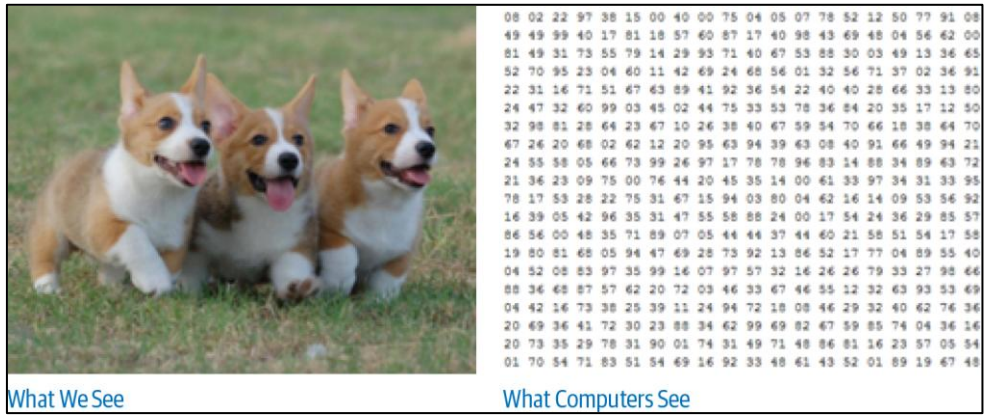
- Assignment 1 & Lab 2
- Introduction to Feature representation
- Text representation techniques
  - One-Hot Encoding
  - Bag of Words model
  - Bag of N-Grams model
- TF-IDF model
- Word similarity

NLP Development Life Cycle

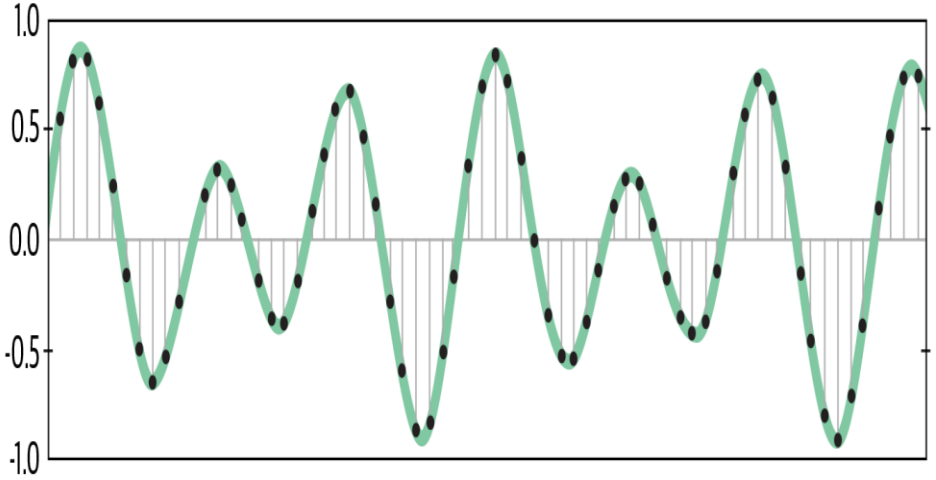


Feature Representation

Image feature representation



speech feature representation



Text Representation Techniques

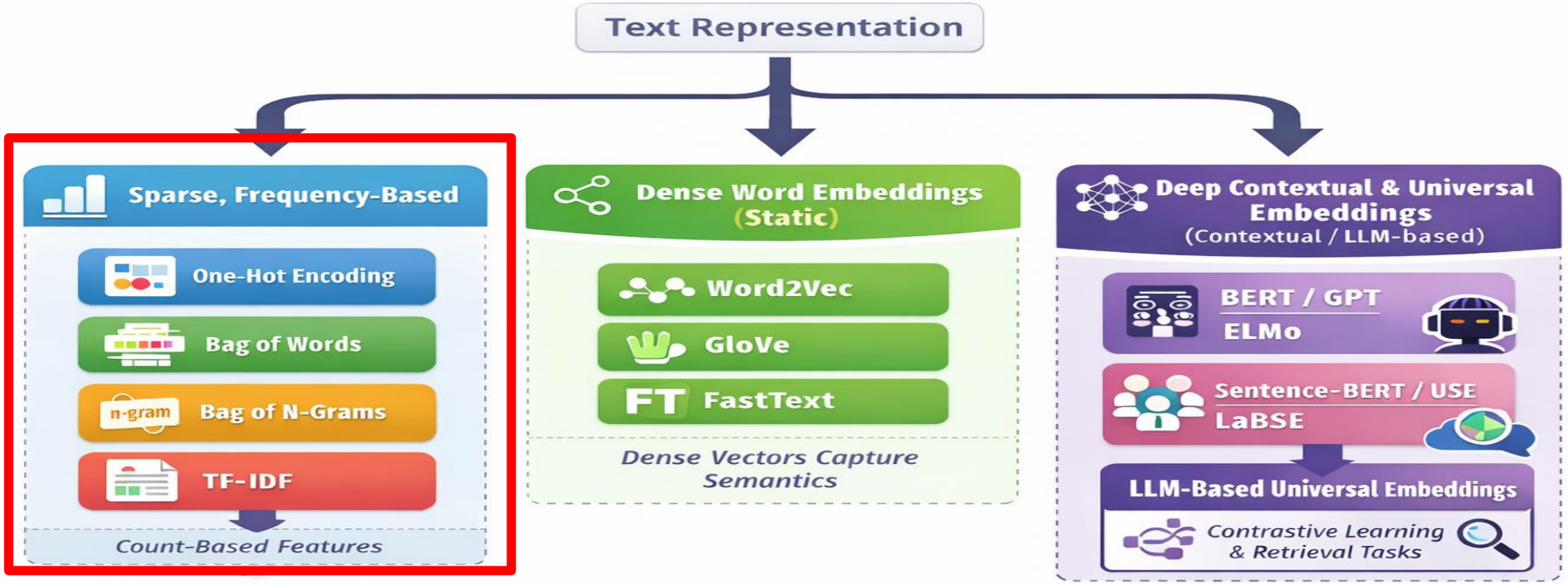


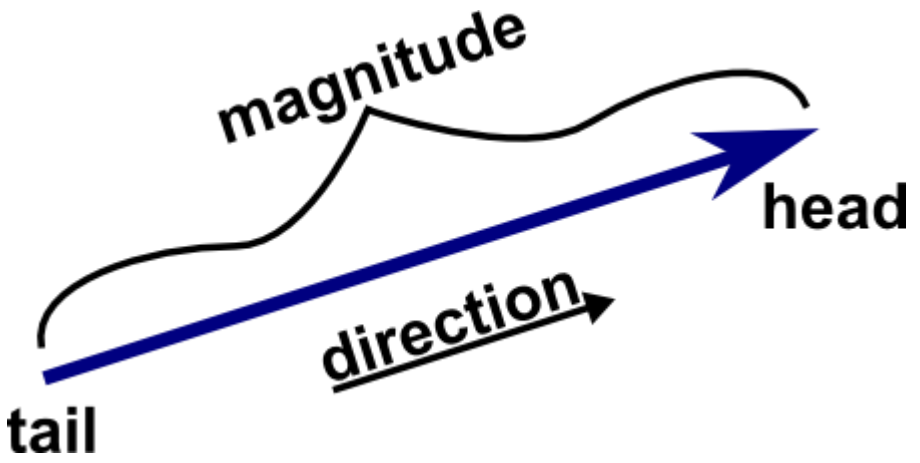
Image source: An automated approach to aspect-based sentiment analysis of apps reviews using machine and deep learning September 2023Automated Software Engineering , then modified using ChatGPT

Vector space Model

- **Mathematical and algebraic model transforming and representing text document in numeric vector.**
- Each word = vector
- Similar words are "**nearby in semantic space**"

Vectors : An introduction

- A vector is an **object** that has both a magnitude and a direction.



Vectors : An introduction…

$\mathbf{x} = (x_1, x_2, x_3, ..., x_n)$ is a vector in an n-dimensional vector space

Length of **x** is given by (extension of Pythagoras's theorem)

$$|\mathbf{x}|^2 = x_1^2 + x_2^2 + x_3^2 + ... + x_n^2$$

$$|\mathbf{x}| = \sqrt{(x_1^2 + x_2^2 + x_3^2 + ... + x_n^2)} \quad (L_2 \text{ Norm})$$

If $\mathbf{x}_1$ and $\mathbf{x}_2$ are vectors:

Inner product (or dot product) is given by

$$\mathbf{x}_1 \cdot \mathbf{x}_2 = x_{11}x_{21} + x_{12}x_{22} + x_{13}x_{23} + ... + x_{1n}x_{2n}$$

Text Vectorizing

- **Vectorizing**
  - Process of encoding text as integers to create feature vectors.
- **Feature Vector**
  - n-dimensional vector of numerical features that represent a text object.

Text Representation :One-Hot Encoding

Each **unique** word in a text converts into a **binary vector**.
Only one element is "**hot**" (set to 1) and all others are "**cold**" (set to 0), indicating the presence of that word.

"This is an example"

Split Text Into Words

['This','is','an','example']

Numerically Encode Words

This	[1,0,0,0]
is	[0,1,0,0]
an	[0,0,1,0]
example	[0,0,0,1]

Tokenization

One-Hot Encoding

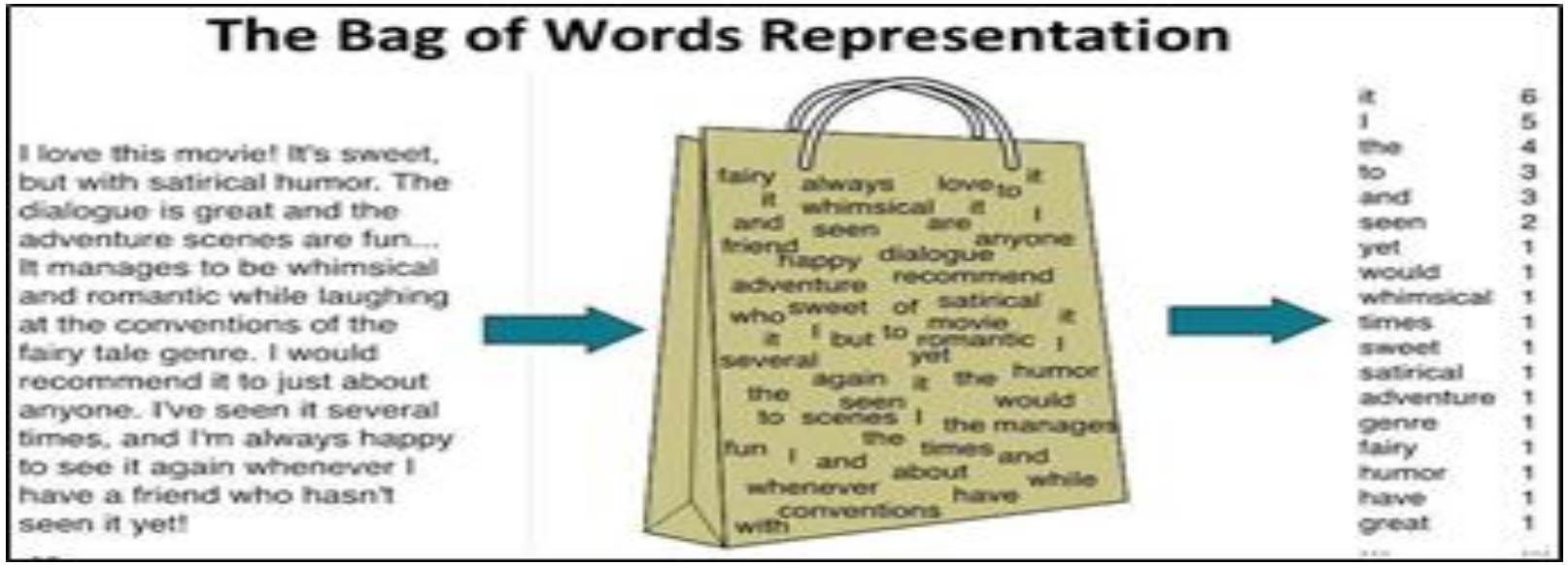
One-Hot Encoding for Documents

	I	like	love	NLP	AI	fun
Doc 1	1	1	0	1	0	0
Doc 2	1	0	1	0	1	0
Doc 3	0	0	0	1	0	1

One-Hot Encoding…

Discussion

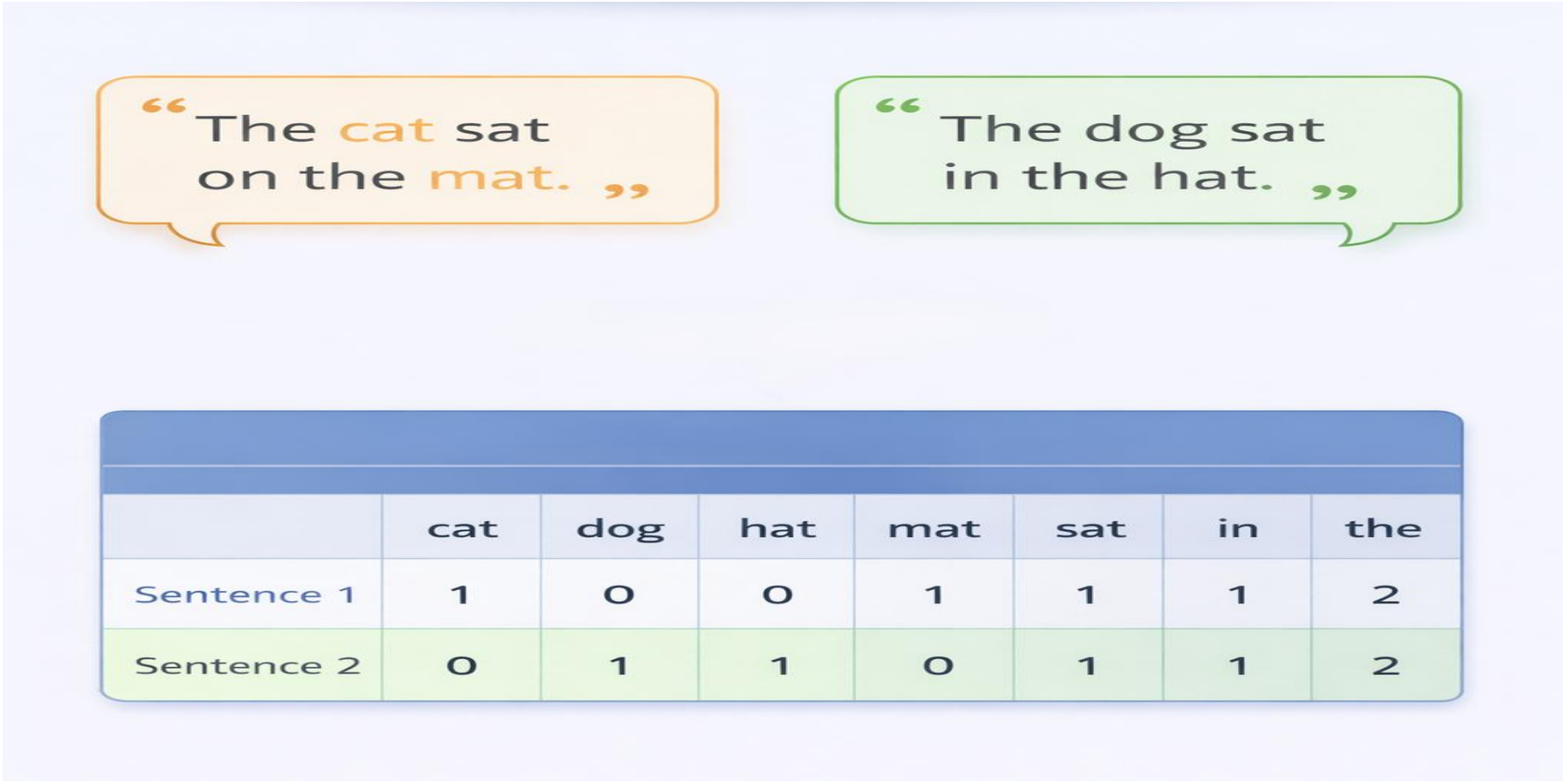
Bag of Words(BOW)



Document is represented as an **unordered collection** of its tokens, disregarding **word order**, and syntax etc.., while keeping track of word **presence or frequency**

Image source: https://sep.com/blog/a-bag-of-words-levels-of-language/

BOW Technique: Count Vectorization



	cat	dog	hat	mat	sat	in	the
Sentence 1	1	0	0	1	1	1	2
Sentence 2	0	1	1	0	1	1	2

Count Vectorization for Documents…

Input:

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer

corpus = ['This is the first document.',
          'This is the second document.',
          'And the third one. One is fun.']

cv = CountVectorizer()
X = cv.fit_transform(corpus)
pd.DataFrame(X.toarray(),columns=cv.get_feature_names())
```

Output:

terms

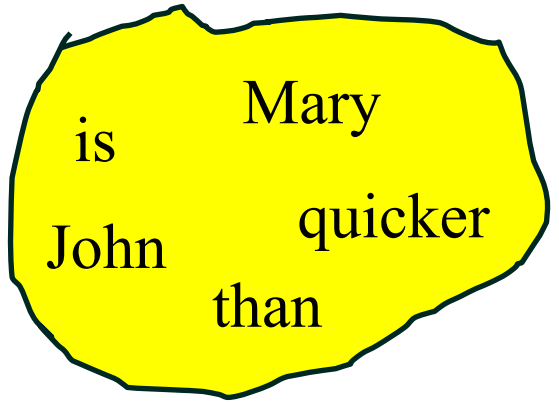
document

	and	document	first	fun	is	one	second	the	third	this
0	0	1	1	0	1	0	0	1	0	1
1	0	1	0	0	1	0	1	1	0	1
2	1	0	0	1	1	2	0	1	1	0

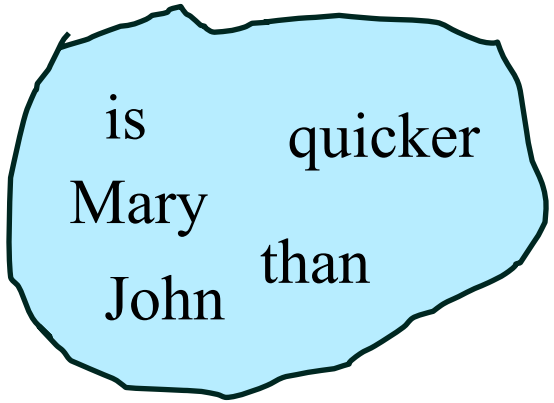
Document-Term Matrix

Bag of Words(BOW)…

John is quicker than Mary.



Mary is quicker than John.



Bag of Words(BOW)…

Document D1	*The child makes the dog happy* the: 2, dog: 1, makes: 1, child: 1, happy: 1
Document D2	*The dog makes the child happy* the: 2, child: 1, makes: 1, dog: 1, happy: 1

	child	dog	happy	makes	the	BoW Vector representations
D1	1	1	1	1	2	[1,1,1,1,2]
D2	1	1	1	1	2	[1,1,1,1,2]

Bag of Words(BOW)…

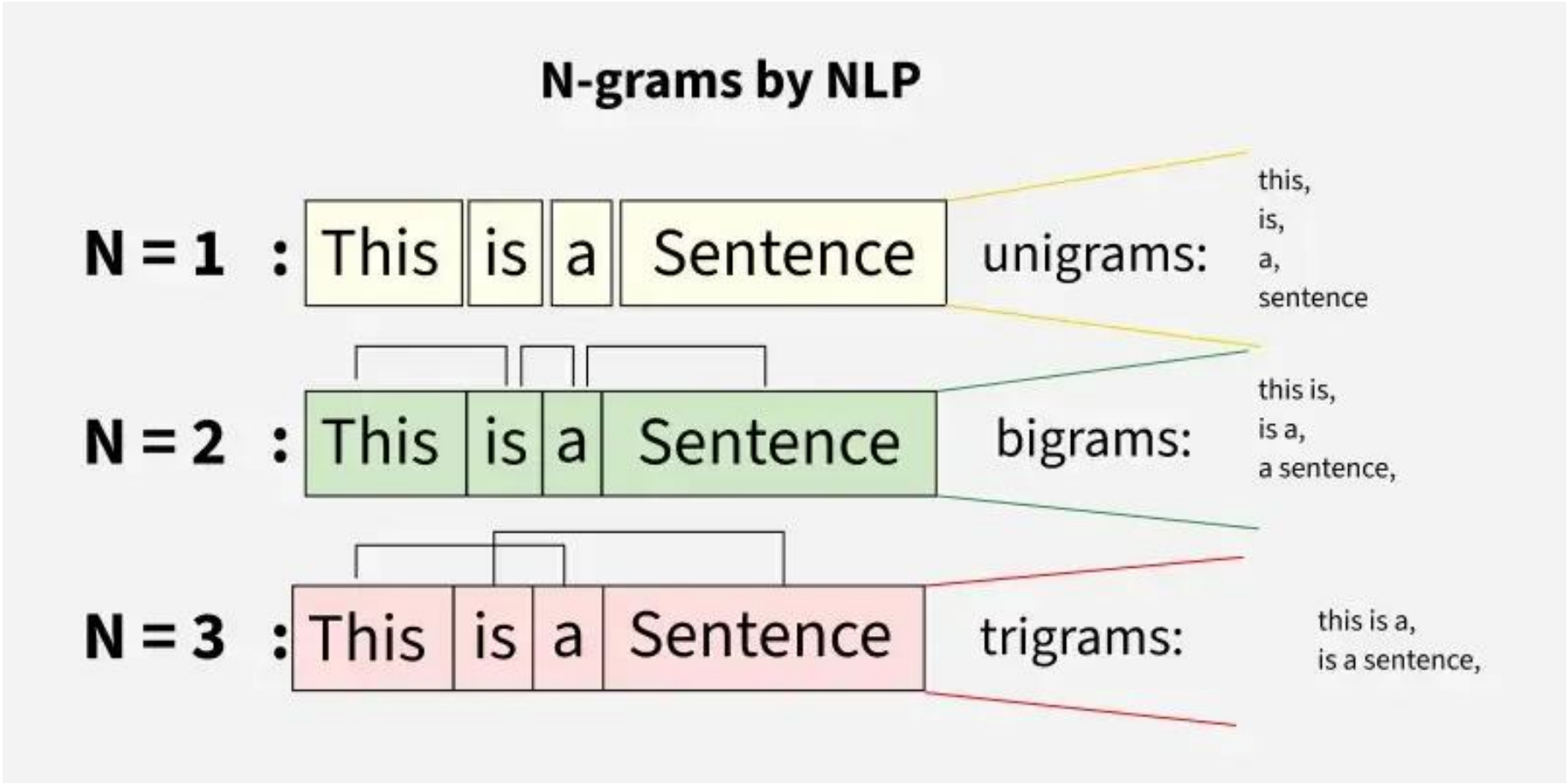
Advantages

- Simplicity and Interpretability
- Works well for text classification and information retrieval.
- Computational Efficiency
- Language Agnostic

Disadvantages

- Ignores context and word order
- High-dimensional feature space
- Sparsity
- Lack of semantic information
- Out Of Vocabulary(**OOV**)

Bag of N-Grams(BON)



Bag of N-Grams(BON)…

Unigrams are the unique words present in the sentence.

Bigram is the combination of 2 words.

Trigram is 3 words.

I am learning NLP

Unigrams: “am”, “ learning”, “NLP”

Bigrams: “am learning”, “learning NLP”

Trigrams: “am learning NLP”

Bag of N-Grams(BON)

Advantages

- It captures some context and word-order information
- Simple and efficient method for representing text data

Disadvantages

- Sparsity
- Computationally expensive
- Ignores the overall structure and meaning of a text.
- Choice of **N**
- Out Of Vocabulary(**OOV**)

Term Frequency-Inverse Document Frequency

TF-IDF Intuition:

- TF-IDF assigns **more weight to rare words** and less weight to commonly occurring words.
- Tells us how frequent a word is in a document **relative to its frequency in the entire corpus.**
- Tells us that two documents are **similar** when they have more rare words in common.

Term Frequency-Inverse Document Frequency…

$$TF\text{-}IDF\ score = TF * IDF$$

Term Frequency-Inverse Document Frequency

Term Frequency

- So far, we've been recording the term (word) count

"This is an example" 

This	is	an	example
1	1	1	1

- A better way to compare them is by a normalized term frequency, which is (term count)/ (total terms).

$$TF(term) = \frac{term\ frequency}{total\ words\ in\ document}$$

Term Frequency-Inverse Document Frequency…

- Besides term frequency, another thing to consider is how common a word is among all the documents
- **Rare words should get additional weight**
- Measures the importance of the term across a corpus.

Term Frequency-Inverse Document Frequency…

Inverse Document Frequency

It aims to **quantify the importance of a given word.** Words that appear in many documents get a **low IDF score**, while words that appear in only a few documents get a **high IDF score**

$$IDF(term) = log_{10}(\frac{total\ number\ of\ documents}{number\ of\ documents\ with\ term})$$

Term Frequency-Inverse Document Frequency…

$$TF\text{-}IDF\ score = TF * IDF$$

TF-IDF :Example

- The sky is blue.
- The sun is bright today.
- The sun in the sky is bright.
- We can see the shining sun, the bright sun

$f_{t,d}$

	blue	bright	can	see	shining	sky	sun	today
1	1	0	0	0	0	1	0	0
2	0	1	0	0	0	0	1	1
3	0	1	0	0	0	1	1	0
4	0	1	1	1	1	0	2	0

$$\mathrm{tf}(t,d) = \frac{f_{t,d}}{\sum_{t'} f_{t',d}}$$

	blue	bright	can	see	shining	sky	sun	today
1	1/2	0	0	0	0	1/2	0	0
2	0	1/3	0	0	0	0	1/3	1/3
3	0	1/3	0	0	0	1/3	1/3	0
4	0	1/6	1/6	1/6	1/6	0	1/3	0

TF-IDF :Example…

$f_{t,d}$

	blue	bright	can	see	shining	sky	sun	today
1	1	0	0	0	0	1	0	0
2	0	1	0	0	0	0	1	1
3	0	1	0	0	0	1	1	0
4	0	1	1	1	1	0	2	0
n_t	**1**	**3**	**1**	**1**	**1**	**2**	**3**	**1**

$N = 4$

$$\mathrm{idf}(t, D) = \log_{10} \frac{N}{n_t}$$

	blue	bright	can	see	shining	sky	sun	today
	0.602	0.125	0.602	0.602	0.602	0.301	0.125	0.602

$\log_{10} \frac{4}{1} = 0.602$ $\log_{10} \frac{4}{3} = 0.125$

$\mathrm{tf}(t, d)$

	blue	bright	can	see	shining	sky	sun	today
1	1/2	0	0	0	0	1/2	0	0
2	0	1/3	0	0	0	0	1/3	1/3
3	0	1/3	0	0	0	1/3	1/3	0
4	0	1/6	1/6	1/6	1/6	0	1/3	0

- TF-IDF: Multiply TF and IDF scores, use to rank importance of words within documents
  - Most important word for each document is highlighted

x

$\mathrm{idf}(t, D)$

	blue	bright	can	see	shining	sky	sun	today
	0.602	0.125	0.602	0.602	0.602	0.301	0.125	0.602

$$\mathrm{tfidf}(t, d, D) = \mathrm{tf}(t, d) \cdot \mathrm{idf}(t, D)$$

	blue	bright	can	see	shining	sky	sun	today
1	**0.301**	0	0	0	0	0.151	0	0
2	0	0.0417	0	0	0	0	0.0417	**0.201**
3	0	0.0417	0	0	0	**0.100**	0.0417	0
4	0	0.0209	**0.100**	**0.100**	**0.100**	0	0.0417	0

Group Work: Compute TF-IDF

D1 *Dog bites man.*
D2 *Man bites dog.*
D3 *Dog eats meat.*
D4 *Man eats food.*

Count Vectorizer vs TF-IDF Vectorizer

```
import pandas as pd
corpus = ['This is the first document.',  'This is the second
       document.',  'And the third one. One is fun.']

# original Count Vectorizer
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
X = cv.fit_transform(corpus).toarray()
pd.DataFrame(X, columns=cv.get_feature_names())

# new TF-IDF Vectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
cv_tfidf = TfidfVectorizer()
X_tfidf = cv_tfidf.fit_transform(corpus).toarray()

pd.DataFrame(X_tfidf, columns=cv_tfidf.get_feature_names())
```

Count Vectorizer vs TF-IDF Vectorizer…

Count Vectorizer Output:

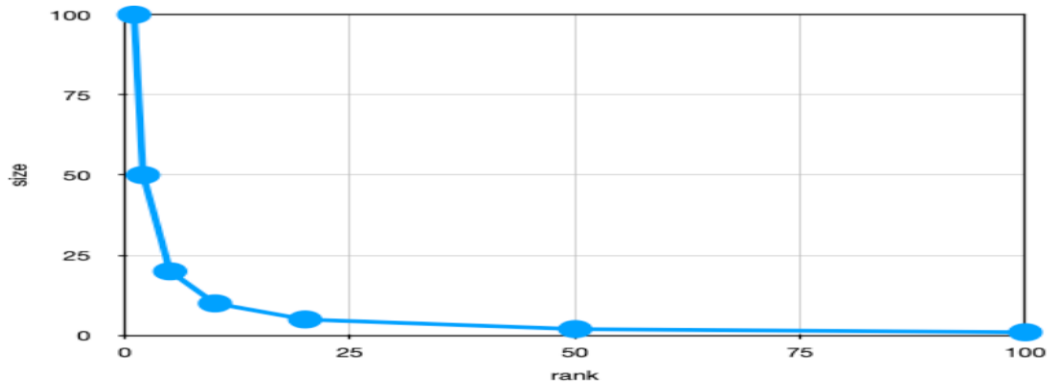
	and	document	first	fun	is	one	second	the	third	this
0	0	1	1	0	1	0	0	1	0	1
1	0	1	0	0	1	0	1	1	0	1
2	1	0	0	1	1	2	0	1	1	0

TF-IDF Vectorizer Output:

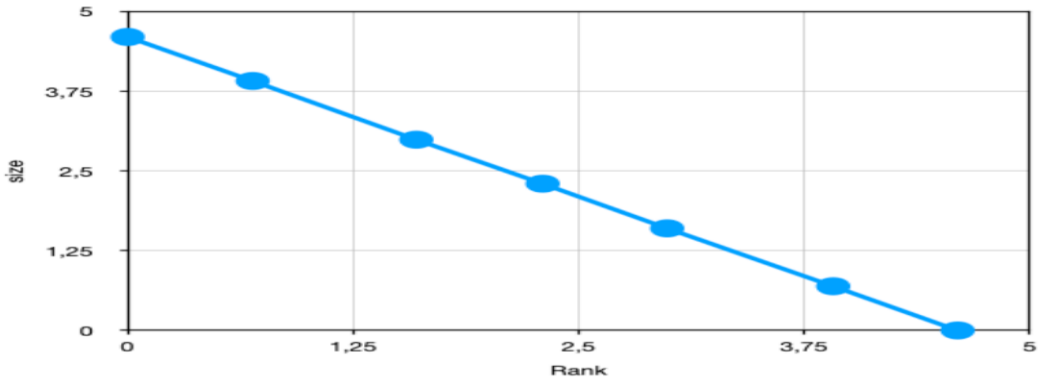
	and	document	first	fun	is	one	second	the	third	this
0	0.00000	0.450145	0.591887	0.00000	0.349578	0.00000	0.000000	0.349578	0.00000	0.450145
1	0.00000	0.450145	0.000000	0.00000	0.349578	0.00000	0.591887	0.349578	0.00000	0.450145
2	0.36043	0.000000	0.000000	0.36043	0.212876	0.72086	0.000000	0.212876	0.36043	0.000000

Why we need *log()*

rank	freq or size
1	100
2	50
5	20
10	10
20	5
50	2
100	1



log(rank)	log(freq or size)
0	4.6
0.693	3.91
1.6	2.99
2.3	2.33
2.99	1.6
3.91	0.693
4.6	0

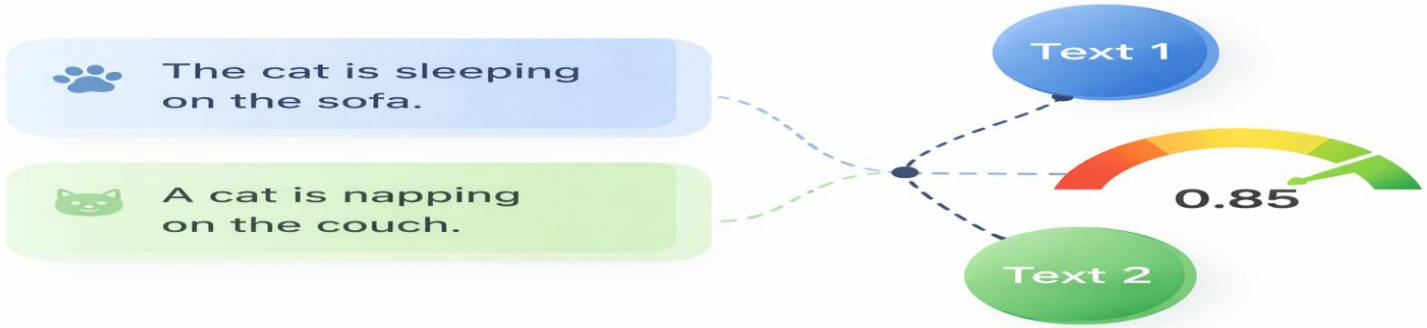


TF-IDF Limitation

- Miss information about the relationships between words
- Sparsity
- Lack of semantic understanding
- Not well suited for small corpora
- **O**ut **O**f **V**ocabulary **OOV** words.

Text Similarity Measure (lexical similarity)

Computational measure of the degree to which two or more documents are semantically or lexically alike.



➢ Applications:

- 👉 Speech Recognition
- 👉 Plagiarism Detection
- 👉 Text Classification
- 👉 Machine Translation
- 👉 Information Retrieval
- 👉 Search engine

Image generated by ChatGPT

Text Similarity Measures…

- Jaccard Similarity
- **Cosine Similarity**
- **Euclidean Distance**
- Hamming Distance
- **Levenshtein Distance**

Text Similarity : Levenshtein distance

Levenshtein distance: Minimum number of operations to get from one word to another.

Levenshtein operations are:

- Deletions: Delete a character
- Insertions: Insert a character
- Mutations: Change a character
  - Example: kitten —> sitting
    - kitten —> sitten (1 letter change)
    - sitten —> sittin (1 letter change)
    - sittin —> sitting (1 letter insertion)

Levenshtein distance = 3

Text Similarity :Euclidean Distance

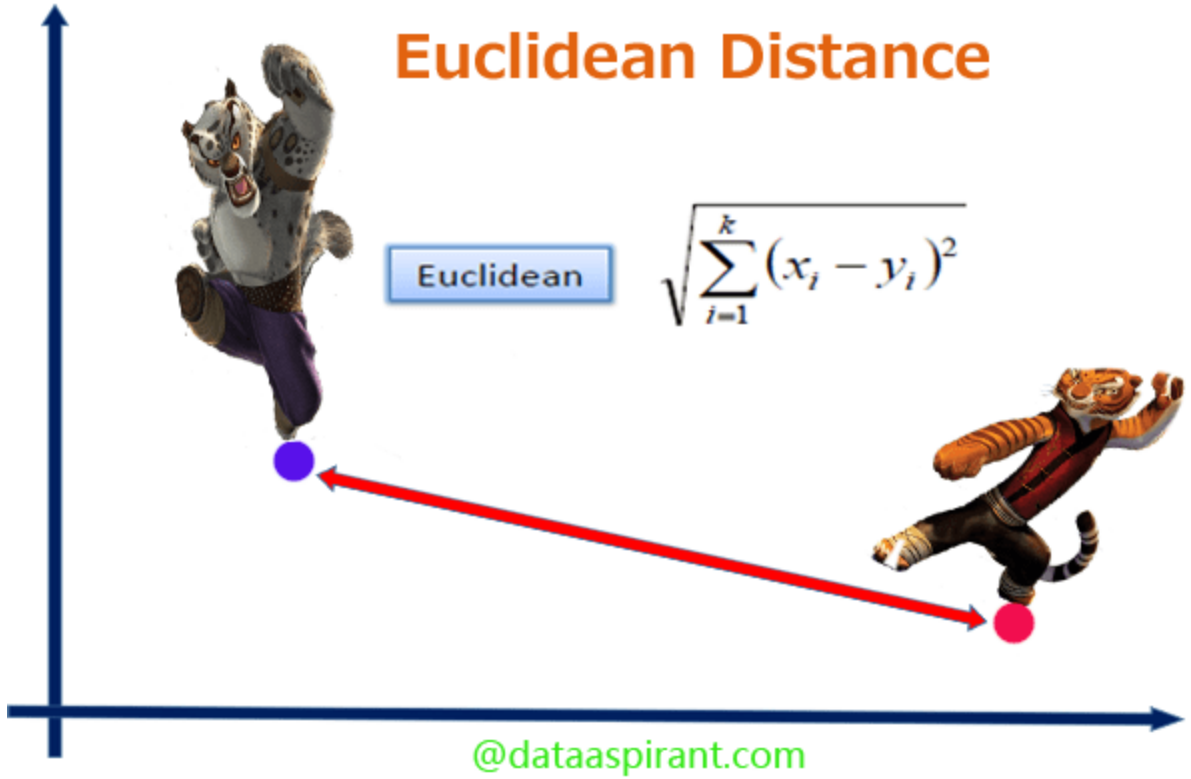
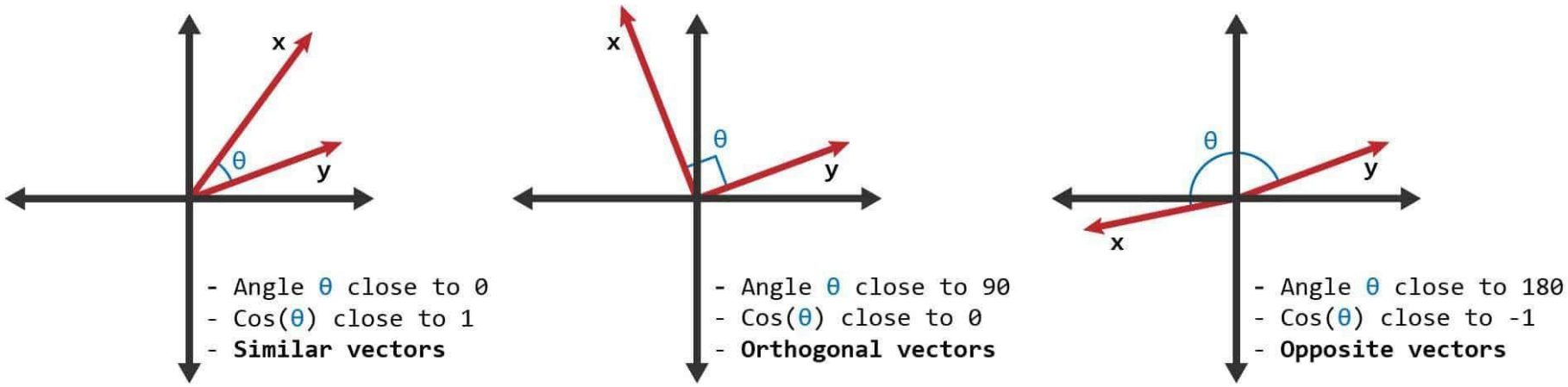


Photo Credit: http://dataaspirant.com/2015/04/11/five-most-popular-similarity-measures-implementation-in-python/

Text Similarity :Cosine

Measures how similar two vectors are by comparing the angle between them

$$\cos(\theta) = (A \cdot B) / (||A|| \, ||B||)$$



39

Cosine Similarity: Example

Cosine Similarity calculation steps:

- Step 1: Put each document in vector format
- Step 2: Find the cosine of the angle between the documents

$$\mathbf{A} \cdot \mathbf{B} = \|\mathbf{A}\| \, \|\mathbf{B}\| \cos\theta$$

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|_2 \|\mathbf{B}\|_2}$$

	i	love	you	nlp
"I love NLP"	1	1	1	0
"I love you"	1	1	0	1

a = [1, 1, 1, 0]
b = [1, 1, 0, 1]

= 0.667

Your turn

$$\cos(\vec{v},\vec{w}) = \frac{\vec{v}\cdot\vec{w}}{|\vec{v}||\vec{w}|} = \frac{\vec{v}}{|\vec{v}|}\cdot\frac{\vec{w}}{|\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

$$\cos(\text{cherry},\text{information}) = ? \frac{442*5+8*3982+2*3325}{\sqrt{442^2+8^2+2^2}\sqrt{5^2+3982^2+3325^2}} = .017$$

$$\cos(\text{digital},\text{information}) = ? \frac{5*5+1683*3982+1670*3325}{\sqrt{5^2+1683^2+1670^2}\sqrt{5^2+3982^2+3325^2}} = .996$$

Your turn

Consider you have the following documents

document	text
d_1	*ant ant bee*
d_2	*dog bee dog hog dog ant dog*
d_3	*cat gnu dog eel fox*

Use cosine measure to compute the pairwise similarity between d_1, d_2 and d_3 using Count Vectorizer, BOW and TF-IDF

	d_1	d_2	d_3
d_1			
d_2			
d_3			

Document Similarity: Example

Here are five documents. Which ones seem most similar?

"The weather is hot under the sun"

"I make my hot chocolate with milk"

"One hot encoding"

"I will have a chai latte with milk"

"There is a hot sale today"

With Count Vectorizer, these two documents were the most similar

Document Similarity: Example…

Output:

```
[(0.40824829, ('The weather is hot under the sun', 'One hot encoding')),
 (0.40824829, ('One hot encoding', 'There is a hot sale today')),
 (0.35355339, ('I make my hot chocolate with milk', 'One hot encoding')),
 (0.33333333, ('The weather is hot under the sun', 'There is a hot sale today')),
 (0.28867513, ('The weather is hot under the sun', 'I make my hot chocolate with milk')),
 (0.28867513, ('I make my hot chocolate with milk', 'There is a hot sale today')),
 (0.28867513, ('I make my hot chocolate with milk', 'I will have a chai latte with milk')),
 (0.0, ('The weather is hot under the sun', 'I will have a chai latte with milk')),
 (0.0, ('One hot encoding', 'I will have a chai latte with milk')),
 (0.0, ('I will have a chai latte with milk', 'There is a hot sale today'))]
```

	chai	chocolate	encoding	hot	latte	make	milk	sale	sun	today	weather
0	0	0	0	1	0	0	0	0	1	0	1
1	0	1	0	1	0	1	1	0	0	0	0
2	0	0	1	1	0	0	0	0	0	0	0
3	1	0	0	0	1	0	1	0	0	0	0
4	0	0	0	1	0	0	0	1	0	1	0

- These two documents are most similar, but it's just because the term "hot" is a popular word
- "Milk" seems to be a better differentiator, so how we can mathematically highlight that?

Document Similarity: Example with TF-IDF

```
[(0.23204485, (‘I make my hot chocolate with milk', 'I will have a chai latte with milk’))

(0.18165505, ('The weather is hot under the sun', 'One hot encoding')),
(0.18165505, ('One hot encoding', 'There is a hot sale today')),
(0.16050660, ('I make my hot chocolate with milk', 'One hot encoding')),
(0.13696380, ('The weather is hot under the sun', 'There is a hot sale today')),
(0.12101835, ('The weather is hot under the sun', 'I make my hot chocolate with milk')),
(0.12101835, ('I make my hot chocolate with milk', 'There is a hot sale today')),
(0.0, (‘The weather is hot under the sun', 'I will have a chai latte with milk')),
(0.0, ('One hot encoding', 'I will have a chai latte with milk')),
(0.0, ('I will have a chai latte with milk', 'There is a hot sale today'))]
```

By weighting “milk” (rare) > “hot” (popular), we get a smarter similarity score

Discussion

What are the **ethical concerns** related to frequency-based text presentation?

Summary

Today we discussed :

- Traditional methods for text representation.
  - One-hot encoding
  - Bag of words
  - Bag of N-grams
  - TF-IDF
- Similarity measures
  - Lexical Similarity – Levenshtein
  - Lexical Similarity – Cosine
  - Lexical Similarity – Euclides distance

Q&A