VIT-AP UNIVERSITY

Natural Language Processing

(Course Code: CSE 3015)

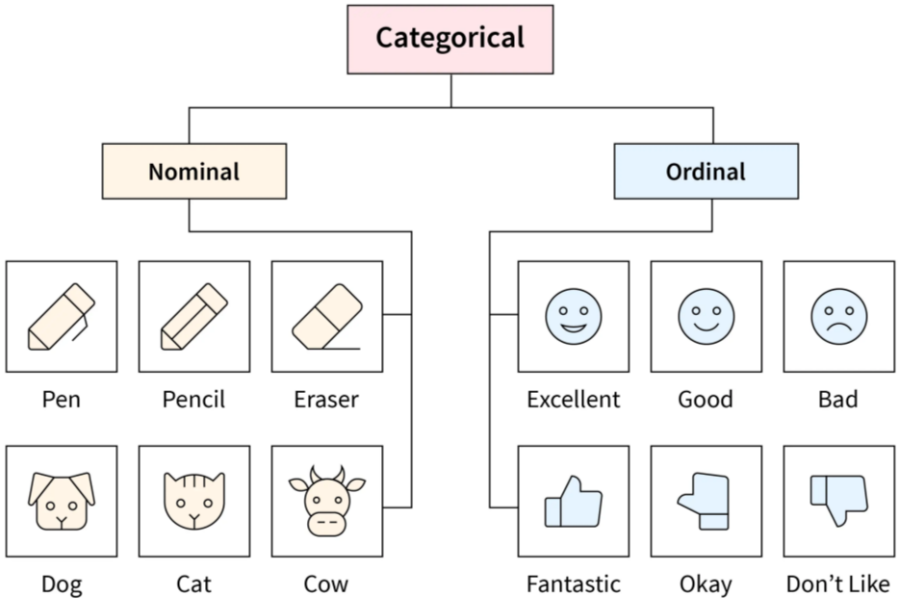
Module-1:Lecture-3: One Hot Encoding Bag of Words and Similarity

Gundimeda Venugopal, Professor of Practice, SCOPE

Categorical Variables/Features

- Most Machine Learning and Deep Learning algorithms require input and output variables/features to be in numerical format.
- In real-world datasets, many important features are not numeric but rather categorical.

- Categorical features can be referred to as a variable that can only contain a limited and fixed number of categories or labels
  - A color variable with values red, blue, and green
  - A country variable with values India, USA, and Germany
  - A gender variable with values Male, Female
  - A product category variable with values Smartphone, Headphone, Computer.
- categorical features/variables can be divided into two categories
  - Ordinal Categorical Variable - Ordinal variables have clear, natural, and intrinsic ordering to their categories. A few examples of ordinal variables are economic status (low income, middle income, high income), educational experience (high school, bachelor's, master's), customer feedback ratings (strongly dislike, dislike, neutral, like, strongly like), etc.
  - Nominal Categorical Variable - Categories in nominal variables have no relationship with each other. For example, age (male, female, transgender), colors (blue, red, green, yellow), blood group (A+, B+, O+, O-), etc.



One Hot Encoding

- One Hot Encoding can be defined as a process of **transforming categorical variables into numerical features** that can be used as input to Machine Learning and Deep Learning algorithms.
- It is a binary vector representation of a categorical variable where the length of the vector is equal to the number of distinct categories in the variable, and in this vector, all values would be 0 except the i^{th} value which will be 1, representing the i^{th} category of the variable.

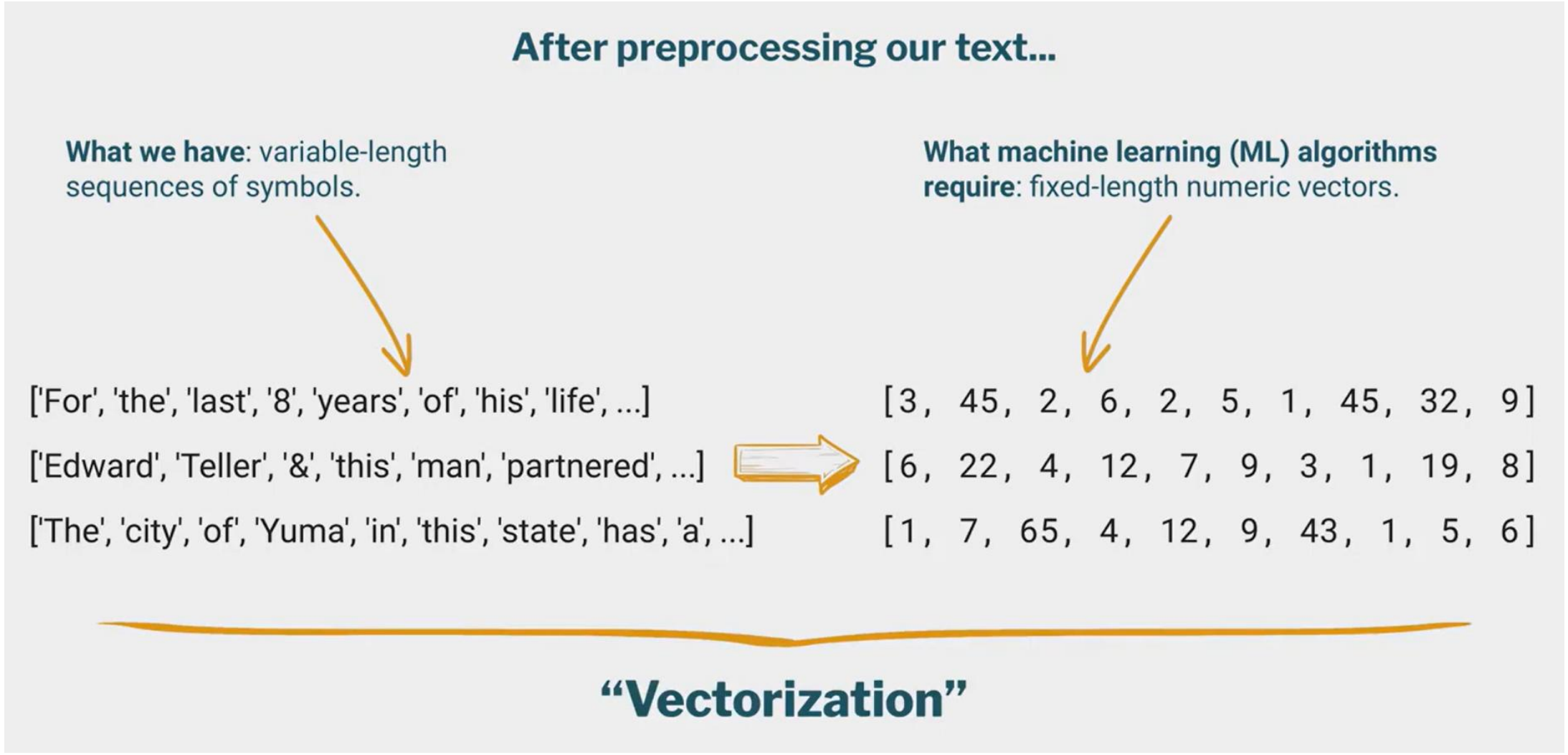
Feature (Color)
Red
Green
Yellow
Green
Red

One Hot Encoding

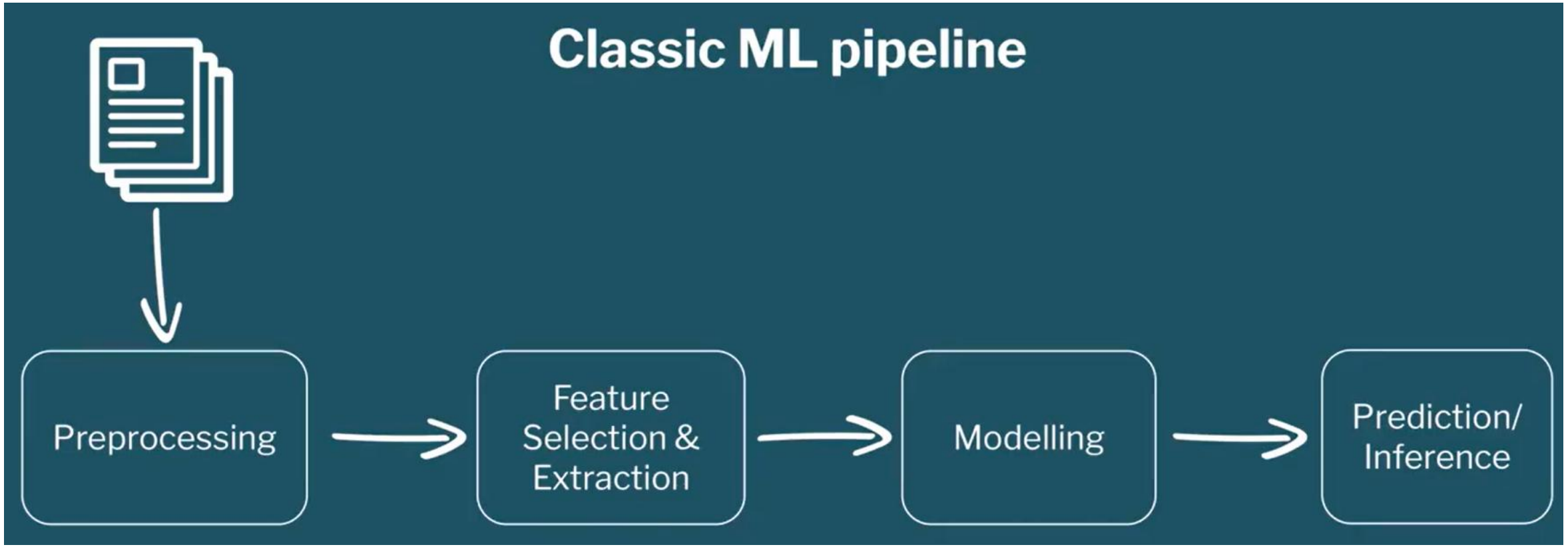
One Hot Encoded Vector
[1,00]
[0,1,0]
[0,0,1]
[0,1,0]
[1,00]

Red	Green	Yellow
1	0	0
0	1	0
0	0	1
0	1	0
1	0	0

Text Processing with Machine/Deep Learning



Classic ML pipeline



Feature Selection and Extraction

Feature Selection & Extraction

Feature: Any property in your data you think is useful for making predictions or explaining some relationship.

We have
Farming data showing various growing conditions and the resulting crop yields.

We want
To predict crop yield.

Possible useful features from the data.

Amount of fertilizer used.

Amount of water applied.

Number of sunny days.

Feature Selection and Extraction

Possible useful features in NLP


Word count


Document age

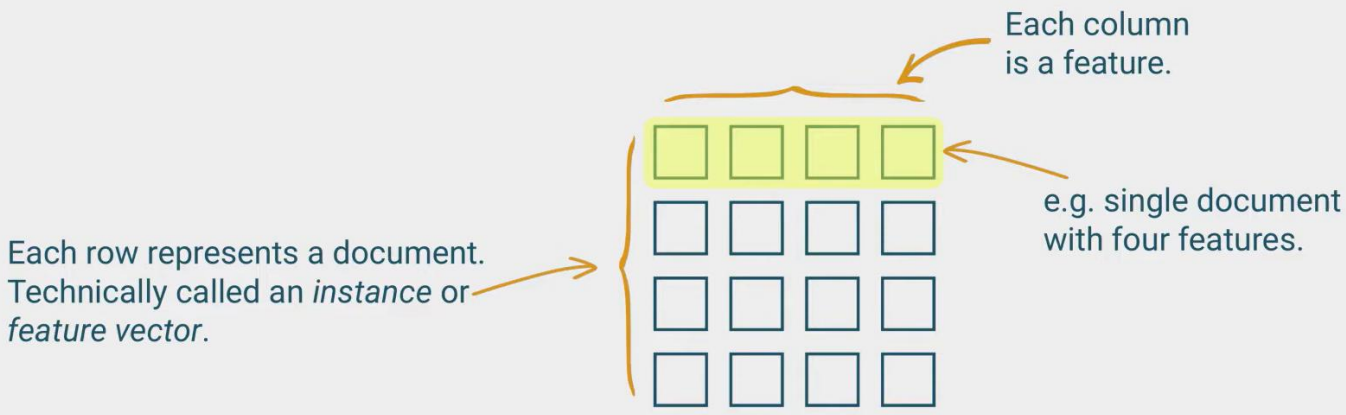

Author ID

Feature *selection*

Feature *extraction*

Combining features or reducing the number of features through *dimensionality reduction*.

What we want to end up with is a matrix



Selecting good features is critical in classic ML

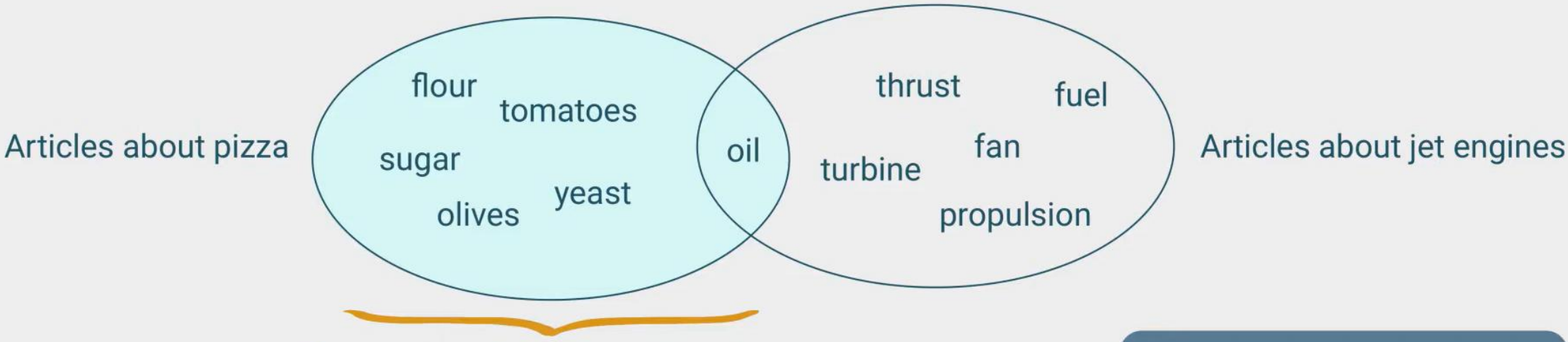
- Domain knowledge is an advantage. Good features with an ordinary algorithm can beat mediocre features with an advanced algorithm.
- Data cleaning and feature engineering is the majority of the work.

Bag-of-Words

Describing documents by word occurrences (multiple approaches).

Primary Idea

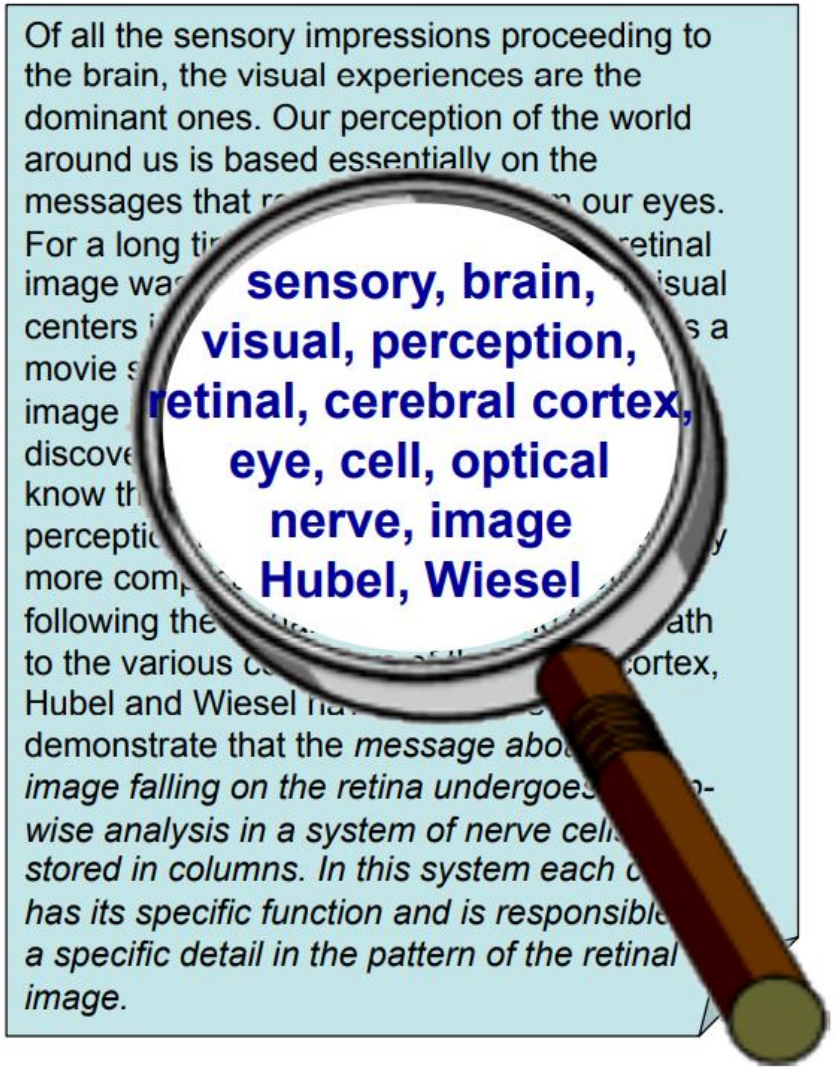
Meaning and similarity are encoded in vocabulary.



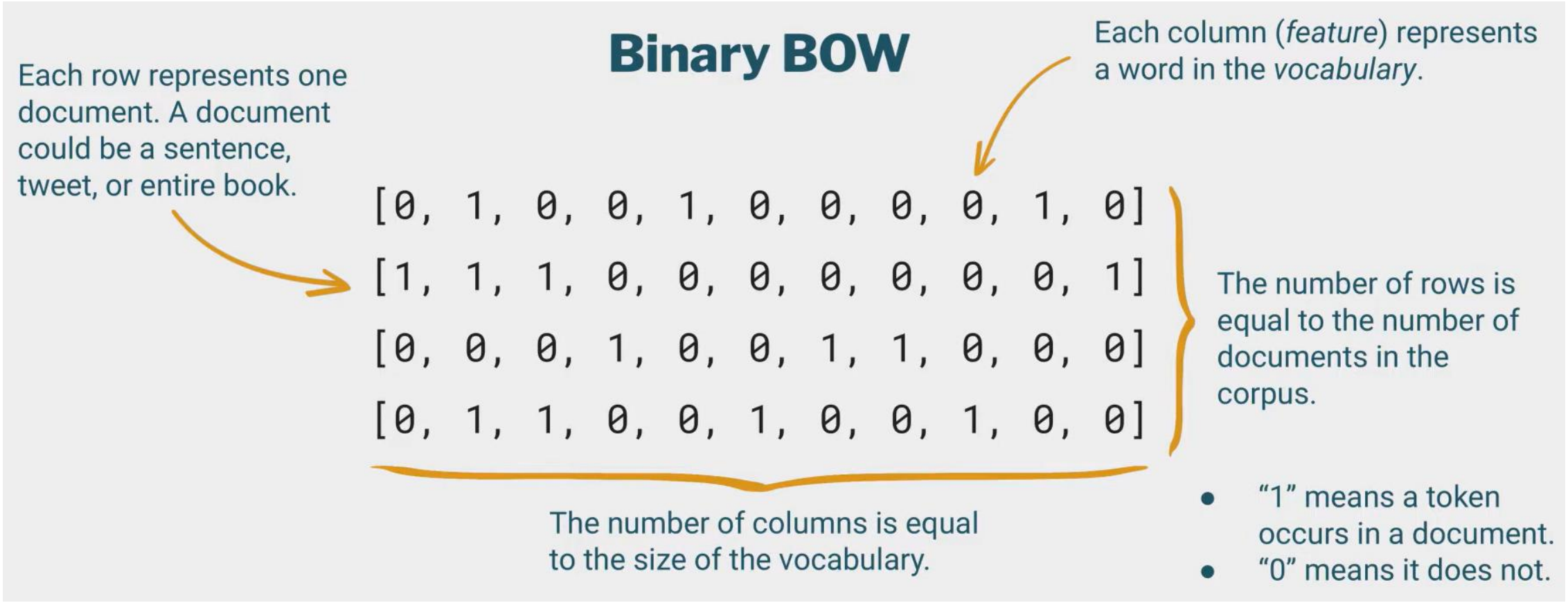
The more two documents share the same vocabulary, the more likely they belong to the same class.

Called a “bag” because we throw out word order.

Bag of words: From two different documents



Binary BoW


Binary BOW
Each row represents one document. A document could be a sentence, tweet, or entire book.
Each column (*feature*) represents a word in the *vocabulary*.
[0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0]
[1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0]
[0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0]
The number of rows is equal to the number of documents in the corpus.
The number of columns is equal to the size of the vocabulary.
"1" means a token occurs in a document.
"0" means it does not.

Binary BoW Example

d_1 : ['Red', 'Bull', 'drops', 'hint', 'on', 'F1', 'engine']
d_2 : ['Honda', 'exits', 'F1', 'leaving', 'F1', 'partner', 'Red', 'Bull']
d_3 : ['Hamilton', 'eyes', 'record', 'eighth', 'F1', 'title']
d_4 : ['Aston', 'Martin', 'announces', 'sponsor']

Let's say we have four documents...

['Aston', 'Bull', 'F1', 'Hamilton', 'Honda', 'Martin', 'Red', 'announces', 'drops', 'eighth', 'engine', 'exits', 'eyes', 'hint', 'leaving', 'on', 'partner', 'record', 'sponsor', 'title']

Our resulting 20-word vocabulary (sorted)

```
{
    'Aston':     0,
    'Bull':      1,
    'F1':        2,
    'Hamilton':  3,
    'Honda':     4,
    'Martin':    5,
    'Red':       6,
    'announces': 7,
    'drops':     8,
    'eighth':    9,
    'engine':    10,
    'exits':     11,
    'eyes':      12,
    'hint':      13,
    'leaving':   14,
    'on':        15,
    'partner':   16,
    'record':    17,
    'sponsor':   18,
    'title':     19
}
```

d_1 : ['Red', 'Bull', 'drops', 'hint', 'on', 'F1', 'engine']
d_2 : ['Honda', 'exits', 'F1', 'leaving', 'F1', 'partner', 'Red', 'Bull']
d_3 : ['Hamilton', 'eyes', 'record', 'eighth', 'F1', 'title']
d_4 : ['Aston', 'Martin', 'announces', 'sponsor']

Resulting binary BOW encoding

D_1: [0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0]

d_1 : ['Red', 'Bull', 'drops', 'hint', 'on', 'F1', 'engine']
d_2 : ['Honda', 'exits', 'F1', 'leaving', 'F1', 'partner', 'Red', 'Bull']
d_3 : ['Hamilton', 'eyes', 'record', 'eighth', 'F1', 'title']
d_4 : ['Aston', 'Martin', 'announces', 'sponsor']

We have our fixed-length numeric vectors, but also...

Resulting binary BOW encoding

D_1: [0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0]
D_2: [0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0]
D_3: [0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1]
D_4: [1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]

Frequency BoW Example

```
{
    'Aston':     0,
    'Bull':      1,
    'F1':        2,
    'Hamilton':  3,
    'Honda':     4,
    'Martin':    5,
    'Red':       6,
    'announces': 7,
    'drops':     8,
    'eighth':    9,
    'engine':    10,
    'exits':     11,
    'eyes':      12,
    'hint':      13,
    'leaving':   14,
    'on':        15,
    'partner':   16,
    'record':    17,
    'sponsor':   18,
    'title':     19
}
```

d_1 : ['Red', 'Bull', 'drops', 'hint', 'on', 'F1', 'engine']

d_2 : ['Honda', 'exits', 'F1', 'leaving', 'F1', 'partner', 'Red', 'Bull']

d_3 : ['Hamilton', 'eyes', 'record', 'eighth', 'F1', 'title']

d_4 : ['Aston', 'Martin', 'announces', 'sponsor']

Resulting binary BOW encoding

D_1: [0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0]

D_2: [0, 1, 2, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0]

D_3: [0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1]

D_4: [1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]

Issues with Binary and Raw frequency BOW

Binary BOW

All words are equally important.

Raw Frequency BOW

Words such as “the” and “it” occur frequently and aren’t informative.

We can remove stop words but...

Raw Frequency BOW

There can still be highly frequent but uninformative words.

e.g. a corpus of sports articles will contain “play” and “game” frequently across most articles.

Sports Articles Corpus

Ice Hockey icing winger puck	Cricket LBW chinaman googly
Tennis deuce ace volley	Chess mate enpassant zugzwang

Use relative frequency instead

Main Idea

Corpus

Given a word...

If a word occurs frequently in one or few documents but not in the rest, it’s likely meaningful for those documents and should be considered more “important”.

Given a word...

But if that word occurs frequently across the corpus, then it should be considered **less** “important” and meaningful.

TF-IDF
Term Frequency – Inverse Document Frequency

- Captures relative importance of words in each document.
- Simple and efficient.
- Easy to compute similarities.
- Dampens stop words automatically.

… some words are more equal than others

TF-IDF: Term Frequency

Term Frequency (TF)

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

Given a word(***t***) in a document(***d***)...

The *term frequency* is just how many times the term occurs in the document.

But some documents can result in much higher frequencies...

$\mathrm{tf}(\text{“dog”}, D_1) = 10$

$\mathrm{tf}(\text{“dog”}, D_2) = 1000$

Doesn't necessarily mean “dog” is that much more meaningful in D_2. The document could just be longer.

So TF is often scaled to dampen the effect

A variant

$$\mathrm{tf}(t, d) = \log_{10}(f_{t,d} + 1)$$

$\mathrm{tf}(t, d) = \log_{10}(0 + 1) = 0$

$\mathrm{tf}(t, d) = \log_{10}(100 + 1) = 2.0$

$\mathrm{tf}(t, d) = \log_{10}(10000 + 1) = 4.0$

TF-IDF: Inverse Document Frequency

Inverse Document Frequency (IDF)

$$\text{idf}(t, D) = \left(\frac{N}{n_t}\right)$$

N is the number of documents.

n_t is the number of documents **t** appears in.

A variant

Inverse Document Frequency (IDF)

$$\underbrace{\text{idf}(t, D)}_{\text{Given a term(}t\text{) and a }\textbf{corpus}\text{(}D\text{)...}} = \underbrace{\log_{10}}_{\text{We take the log here as well.}} \left(\frac{N}{n_t}\right)$$

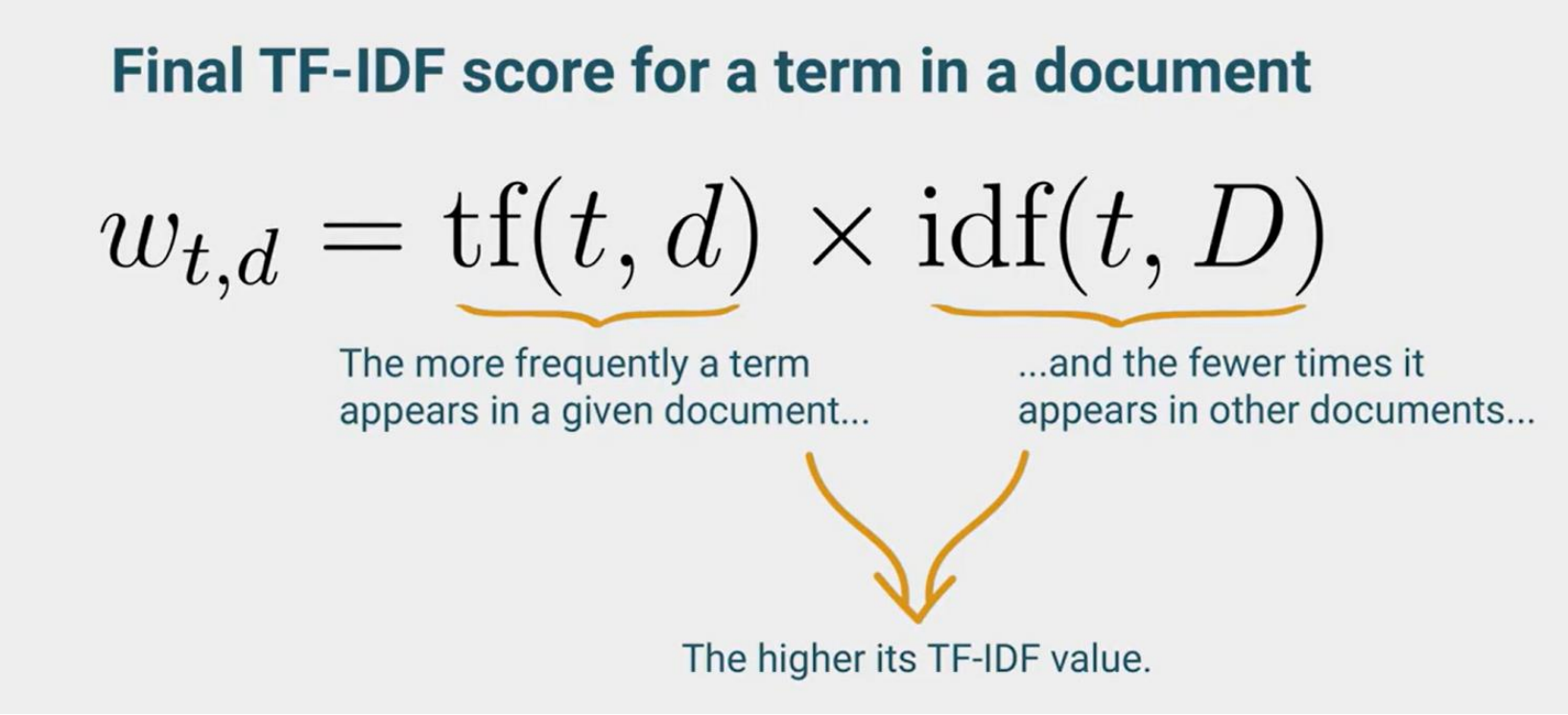
N is the number of documents.

n_t is the number of documents **t** appears in.

Intuition

The fewer documents the term *t* appears in, the *higher* this IDF value will be, and the more weight it'll give to the word.

TF-IDF: Term frequency-Inverse Document Frequency



TF-IDF Example

d_1: “you have brains in your head”

d_2: “you have feet in your shoes”

d_3: “you can steer yourself”

d_4: “any direction you choose”

d_1	TF	IDF	TF-IDF
“you”	0.3	0	0
“brains”	0.3	0.6	0.18

Many TF-IDF variations out there...

e.g. the *default* TF-IDF vectorizer in scikit-learn

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

TF is just raw count.

$$\mathrm{idf}(t, D) = \ln\left(\frac{N + 1}{n_t + 1}\right) + 1$$

Uses natural logarithm. (base e)

Avoids zero division.

Text Similarity

How similar are two strings?

- Spell correction
  - The user typed “graffe”

    Which is closest?
    - graf
    - graft
    - grail
    - giraffe

- Computational Biology
  - Align two sequences of nucleotides

    AGGCTATCACCTGACCTCCAGGCCGATGCCC
    TAGCTATCACGACCGCGGTCGATTTGCCCGAC

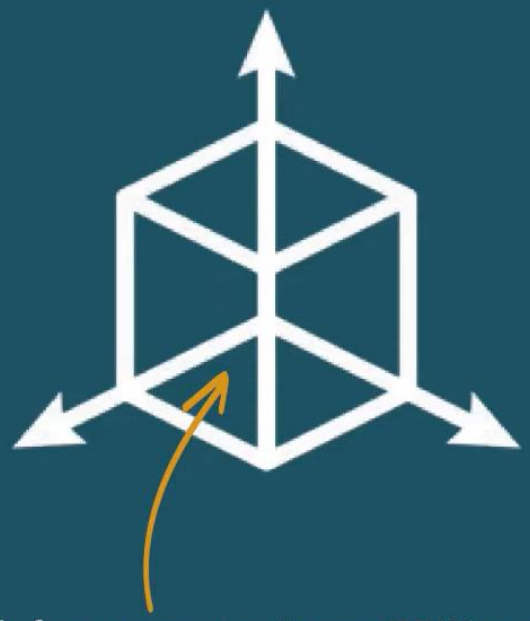
  - Resulting alignment:

    –AGG**CTATCAC**CT**GACC**T**C**CA**GG**C**CGA**--**TGCCC**---
    T**AG**–**CTATCAC**--**GACC**G**C**--**GG**T**CGA**TT**TGCCC**GAC

- Also for Machine Translation, Information Extraction, Speech Recognition

Vector Space Model

We have shifted our thinking about text...



Each feature vector in our BOW is now a point in this multidimensional* space.

Vector Space Model (VSM)

*The number of dimensions is equal to the number of features (i.e. size of vocabulary).

What we can now do

Two documents which have similar vocabulary should be closer together in this space. This allows us to start measuring document similarity which is useful for:

- Relevance ranking
- Plagiarism detection
- Document classification

And other applications...

Quantifying similarity between text units:

Most common ways to capture similarity between text units are:

- Cosine Similarity
- Longest Common Substring (LCS)
- Levenshtein Edit Distance
- Euclidean Distance
- Hamming Distance
- Jaccard Distance

Vector Dot Product

The basis for most vector similarity metrics

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_1 b_1 + a_2 b_2 + \ldots + a_n b_n$$

Two vectors of equal length (*a*, *b*)

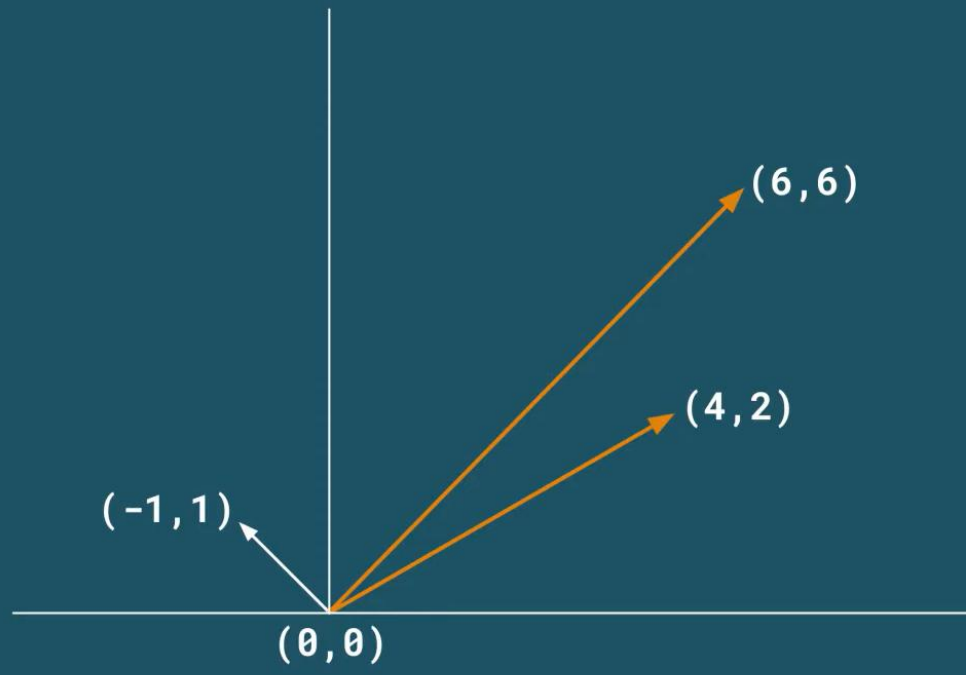
Multiply their corresponding values and add the products together.

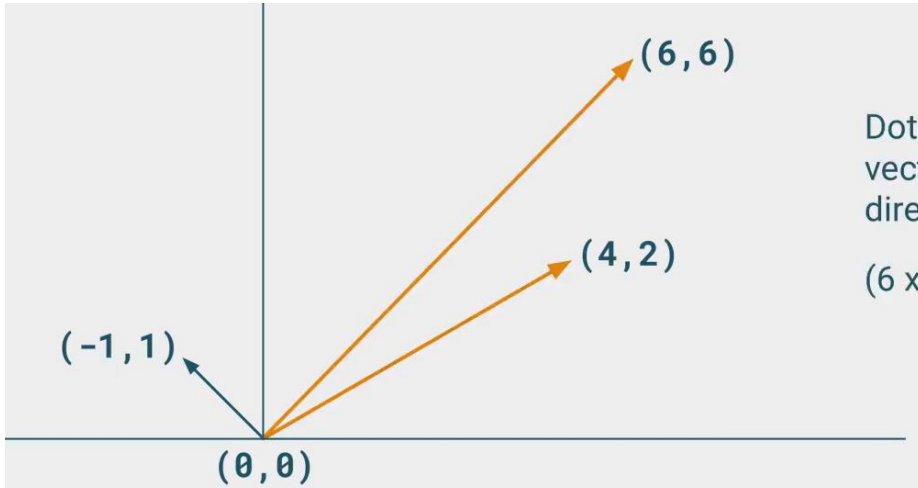
a = [1, 2, 3]
b = [4, 5, 6]

a · **b** = (1x4) + (2x5) + (3x6) = **32**

Vector Dot Product Example

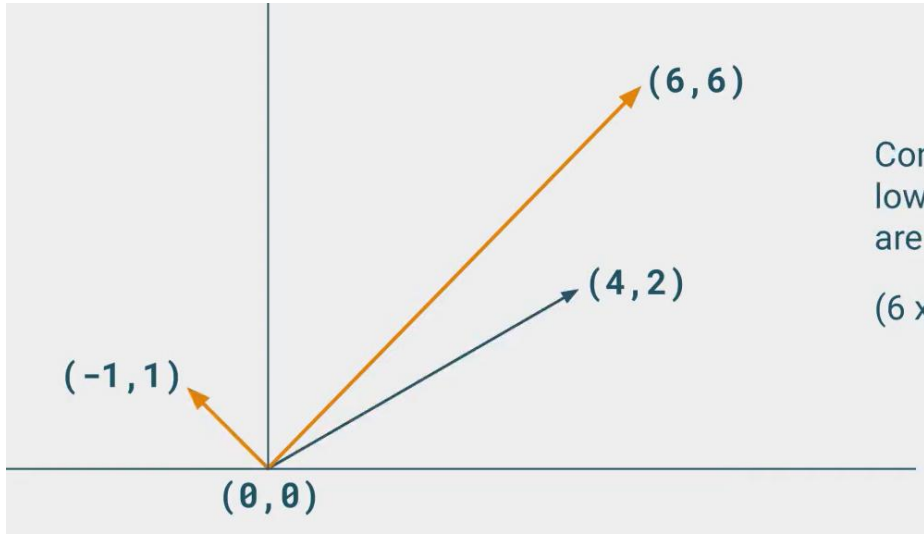
The more two vectors share the same direction and magnitude, the more similar they are.





Dot product will be high when two vectors share a similar or same direction, and with large magnitudes.

(6 x 4) + (4 x 2) = 30



Conversely, the dot product will be low or 0 (orthogonal vectors) if they are dissimilar.

(6 x -1) + (6 x 1) = 0

Vector Dot Product (for Text Documents Comparison)

But high frequency words will lead to larger dot products...

So we normalize by vector length:

$$\|\mathbf{v}\| = \sqrt{\sum_{i=1}^{n} v_i^2}$$

Also known as the L^2 *norm* or *Euclidean norm*.

Which leads to

$$\frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|} = \cos\theta$$

The same as the cosine of the angle between two vectors

Cosine Similarity

$$\cos\theta = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|}$$

The value of **cos θ** ranges from 0 to 1.*

- 1 when the vectors point in the same direction.
- 0 when the vectors are orthogonal (dissimilar).

*won't be lower than zero because we won't have negative frequencies.

Cosine Similarity Example (for document comparison)

d_1 : ['Red', 'Bull', 'drops', 'hint', 'on', 'F1', 'engine']

[0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0]

d_2 : ['Honda', 'exits', 'F1', 'leaving', 'F1', 'partner', 'Red', 'Bull']

[0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0]

d_3 : ['Hamilton', 'eyes', 'record', 'eighth', 'F1', 'title']

[0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1]

d_4 : ['Aston', 'Martin', 'announces', 'sponsor']

[1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]

$$\cos\theta = \frac{\mathbf{d}_1 \cdot \mathbf{d}_2}{\|\mathbf{d}_1\| \, \|\mathbf{d}_2\|} = 0.43$$

$$\cos\theta = \frac{\mathbf{d}_1 \cdot \mathbf{d}_3}{\|\mathbf{d}_1\| \, \|\mathbf{d}_3\|} = 0.15$$

$$\cos\theta = \frac{\mathbf{d}_1 \cdot \mathbf{d}_4}{\|\mathbf{d}_1\| \, \|\mathbf{d}_4\|} = 0.0$$

The threshold you select for “similar” will depend on your data.

Note: Cosine similarity gives us “How much vocabulary overlap is there between the two documents”

Cosine Similarity (using BoW) issues

"Surface" similarity only...

"Dog bites man."

"Canine nips human."

} Zero cosine similarity

Binary or raw frequency BOW can be a good start but...

Drawbacks

- Doesn't capture similarity between synonyms.
- No way to handle Out-of-Vocabulary (OOV) words.
- Creates sparse vectors which can be inefficient.
- Word order information is lost. "*Chelsea beats Barcelona*" is different from "*Barcelona beats Chelsea*".

Longest Common Substring (LCS)

- Given two strings, s1 of length n1 and s2 of length n2, it simply considers the length of the longest string (or strings) which is substring of both s1 and s2.
- Applications: Data deduplication and Plagiarism detection.
- Example: The LCS of strings ‘Lydia’ and ‘Lydoko’ simply returns 3.

```
LCS('Lydia', 'Lydoko') = 3
```

		L	Y	D	I	A
	0	0	0	0	0	0
L	0	1	0	0	0	0
Y	0	0	2	0	0	0
D	0	0	0	3	0	0
O	0	0	0	0	0	0
K	0	0	0	0	0	0
O	0	0	0	0	0	0

Edit Distance

- The minimum edit distance between two strings
- Is the minimum number of editing operations
  - Insertion
  - Deletion
  - Substitution
- Needed to transform one into the other

Minimum Edit Distance

- Two strings and their **alignment**

```
I N T E * N T I O N
| | | | | | | | | |
* E X E C U T I O N
d s s   i s
```

d: Deletion
s: Substitution
i: Insert

- If each operation has cost of 1
  - Distance between these is 5
- If substitutions cost 2 (Levenshtein)
  - Distance between them is 8

Alignment in Computational Biology

- Given a sequence of bases

```
AGGCTATCACCTGACCTCCAGGCCGATGCCC
TAGCTATCACGACCGCGGTCGATTTGCCCGAC
```

- An alignment:

```
-AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
TAG-CTATCAC--GACCGC--GGTCGATTTGCCCGAC
```

- Given two sequences, align each letter to a letter or gap

Uses of Edit Distance in NLP

❖Evaluating Machine Translation and speech recognition

```
R Spokesman confirms    senior government adviser was appointed was shot
H Spokesman said    the senior          adviser was appointed was shot dead
              S      I             D                                  I
```

❖Named Entity Extraction and Entity Coreference

IBM Inc. announced today!

IBM profits

Stanford President John Hennessy announced yesterday

for Stanford University President John Hennessy

Edit distance method

Edit Distance measures dissimilarity between two strings by finding the minimum number of operations needed to transform one string into the other.

The transformations that can be performed are:

- inserting a new character:
  - bat -> bats (insertion of 's')
- Deleting an existing character.
  - care -> car (deletion of 'e')
- Substituting an existing character.
  - bin -> bit (substitution of n with t)
- Transposition of two existing consecutive characters.
  - sing -> sign (transposition of ng to gn)

Edit Distance (a.k.a. Levenshtein Distance) is a measure of similarity between two strings referred to as the source string and the target string.

The distance between the source string and the target string is the minimum number of edit operations (deletions, insertions, or substitutions) required to transform the source into the target. The lower the distance, the more similar the two strings.

Among the common applications of the Edit Distance algorithm are: spell checking, plagiarism detection, and translation memory systems.

Edit Distance Python NLTK

Example 1:

```
import nltk
w1 = 'mapping'
w2 = 'mappings'

nltk.edit_distance(w1, w2)
```

Example #2

Basic Spelling Checker: Let’s assume you have a mistaken word and a list of possible words and you want to know the nearest suggestion.

```
Import nltk
mistake = "ligting"

words = ['apple', 'bag', 'drawing', 'listing', 'linking', 'living', 'lighting', 'orange', 'walking', 'zoo']

for word in words:
    ed = nltk.edit_distance(mistake, word)
    print(word, ed)
```

As you can see, comparing the mistaken word “ligting” to each word in our list, the least Edit Distance is 1 for words: “listing” and “lighting” which means they are the best spelling suggestions for “ligting”. Yes, a smaller Edit Distance between two strings means they are more similar than others.

Example #3

Sentence or paragraph comparison is useful in applications like plagiarism detection (to know if one article is a stolen version of another article), and translation memory systems (that save previously translated sentences and when there is a new untranslated sentence, the system retrieves a similar one that can be slightly edited by a human translator instead of translating the new sentence from scratch).

```
import nltk
sent1 = "It might help to re-install Python if possible."
sent2 = "It can help to install Python again if possible."
sent3 = "It can be so helpful to reinstall C++ if possible."
sent4 = "help It possible Python to re-install if might."
sent5 = "I love Python programming."

ed_sent_1_2 = nltk.edit_distance(sent1, sent2)
ed_sent_1_3 = nltk.edit_distance(sent1, sent3)
ed_sent_1_4 = nltk.edit_distance(sent1, sent4)
ed_sent_1_5 = nltk.edit_distance(sent1, sent5)

print(ed_sent_1_2, 'Edit Distance between sent1 and sent2')
print(ed_sent_1_3, 'Edit Distance between sent1 and sent3')
print(ed_sent_1_4, 'Edit Distance between sent1 and sent4')
print(ed_sent_1_5, 'Edit Distance between sent1 and sent5')
```

So it is clear that sent1 and sent2 are more similar to each other than other sentence pairs.

Jaccard Distance

- We get Jaccard distance by subtracting the Jaccard coefficient from 1.
- We can also get it by dividing the difference between the sizes of the union and the intersection of two sets by the size of the union.
- Jaccard Distance is given by the following formula

$$JaccardIndex = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

$$JaccardDistance = 1 - JaccardIndex$$

Jaccard Similarity defined as an intersection of two documents divided by the union of that two documents that refer to the number of common words over a total number of words. Here, we will use the set of words to find the intersection and union of the document.

The mathematical representation of the **Jaccard Similarity** is:

$$J(doc_1, doc_2) = \frac{doc_1 \cap doc_2}{doc_1 \cup doc_2}$$

The Jaccard Similarity score is in a range of **0 to 1**. If the two documents are identical, Jaccard Similarity is **1**. The Jaccard similarity score is **0** if there are no common words between two documents.

doc_1 = "Data is the new oil of the digital economy"
doc_2 = "Data is a new oil"

Let’s get the set of unique words for each document.

words_doc1 = {'data', 'is', 'the', 'new', 'oil', 'of', 'digital', 'economy’}

words_doc2 = {'data', 'is', 'a', 'new', 'oil'}

$$J(doc_1, doc_2) = \frac{\{'data', 'is', 'the', 'new', 'oil', 'of', 'digital', 'economy'\} \cap \{'data', 'is', 'a', 'new', 'oil'\}}{\{'data', 'is', 'the', 'new', 'oil', 'of', 'digital', 'economy'\} \cup \{'data', 'is', 'a', 'new', 'oil'\}}$$

$$= \frac{\{'data', 'is', 'new', 'oil'\}}{\{'data', 'a', 'of', 'is', 'economy', 'the', 'new', 'digital', 'oil'\}}$$

$$= \frac{4}{9} = 0.444$$

Jaccard Distance Of Strings

Jaccard distance is a measure of similarity between two sets. It quantifies how similar the sets are by comparing their shared elements with their total combined elements. To calculate it, you need to find the size of the intersection (shared elements) divided by the size of the union (all unique elements) of the sets.

The Jaccard similarity is calculated by dividing the number of observations in both sets by the number of observations in either set. In other words, the Jaccard similarity can be computed as the size of the intersection divided by the size of the union of two sets.

Example 1:

```
import nltk
w1 = set('mapping')
w2 = set('mappings')

nltk.jaccard_distance(w1, w2)
```

Unlike Edit Distance, you cannot just run Jaccard Distance on the strings directly; you must first convert them to the set type.

Example #2

Basic Spelling Checker: It is the same example we had with the Edit Distance algorithm; now we are testing it with the Jaccard Distance algorithm. Let's assume you have a mistaken word and a list of possible words and you want to know the nearest suggestion.

```
Import nltk
mistake = "ligting"

words = ['apple', 'bag', 'drawing', 'listing', 'linking', 'living', 'lighting', 'orange', 'walking', 'zoo']

for word in words:
    jd = nltk.jaccard_distance(set(mistake), set(word))
    print(word, jd)
```

Again, comparing the mistaken word "ligting" to each word in our list, the least Jaccard Distance is 0.166 for words: "listing" and "lighting" which means they are the best spelling suggestions for "ligting" because they have the lowest distance.

```
Import nltk
sent1 = set("It might help to re-install Python if possible.")
sent2 = set("It can help to install Python again if possible.")
sent3 = set("It can be so helpful to reinstall C++ if possible.")
sent4 = set("help It possible Python to re-install if might.")
sent5 = set("I love Python programming.")

jd_sent_1_2 = nltk.jaccard_distance(sent1, sent2)
jd_sent_1_3 = nltk.jaccard_distance(sent1, sent3)
jd_sent_1_4 = nltk.jaccard_distance(sent1, sent4)
jd_sent_1_5 = nltk.jaccard_distance(sent1, sent5)



print(jd_sent_1_2, 'Jaccard Distance between sent1 and sent2')
print(jd_sent_1_3, 'Jaccard Distance between sent1 and sent3')
print(jd_sent_1_4, 'Jaccard Distance between sent1 and sent4')
print(jd_sent_1_5, 'Jaccard Distance between sent1 and sent5')
```

Just like when we applied Edit Distance, sent1 and sent2 are the most similar sentences.

However, look to the other results; they are completely different.

The most obvious difference is that the Edit Distance between sent1 and sent4 is 32 and the Jaccard Distance is zero, which means the Jaccard Distance algorithms sees them as identical sentence because Edit Distance depends on counting *edit* operations from the start to end of the string while Jaccard Distance just counts the number characters and then apply some calculations on this number as mentioned above.

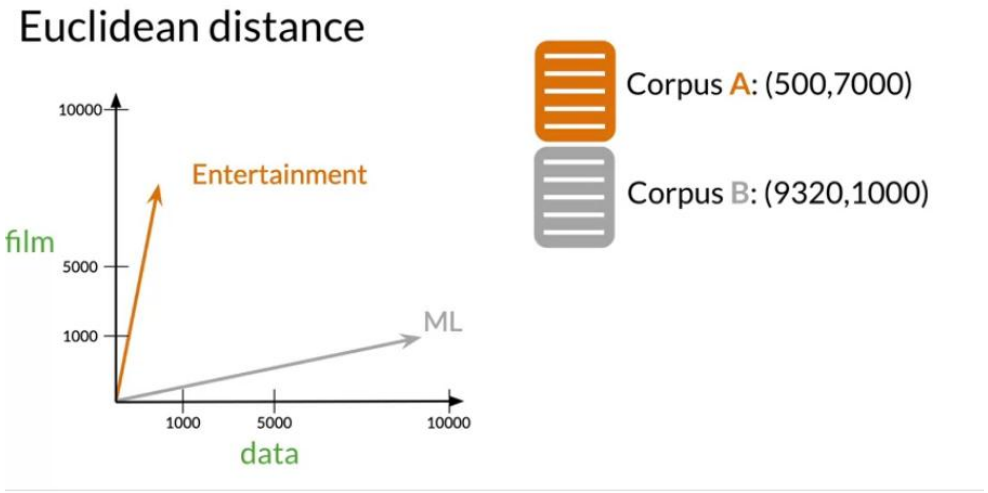
Actually, there is no “right” or “wrong” answer; it all depends on what you really need to do.

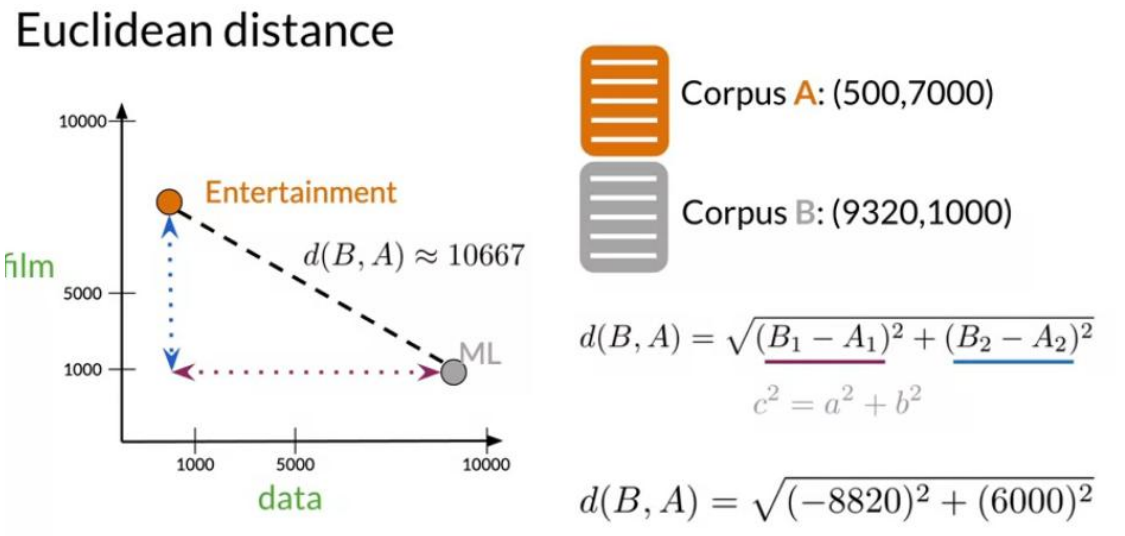
Euclidean distance

- Euclidean distance basically calculated based on the Pythagoras theorem. To find the distance between two points on a plane using Euclidean distance, the length of the straight line connecting the two points is measured.

Example:

- Corpus A will be the entertainment corpus and Corpus B will be the machine-learning corpus.
- The number of times that the word “data” and the word “film” appeared in the corpus.
- The feature used to represent the “Entertainment” and “ML” corpus vectors: word frequency of “film” and “data”





Euclidean distance between the two corpus vectors: “Entertainment” and “ML” = *d (B,A)*

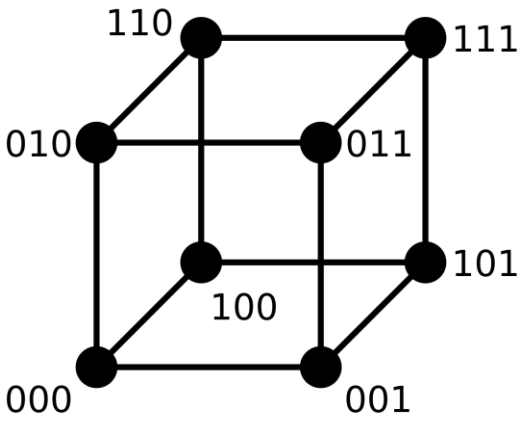
Hamming Distance

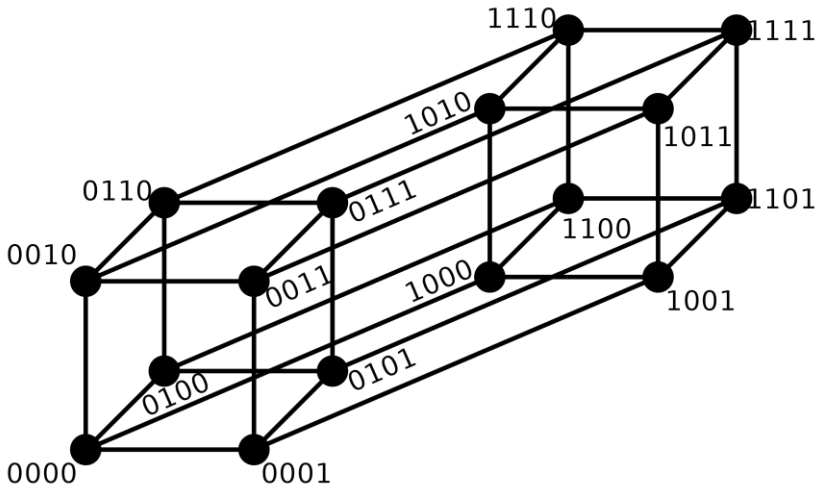
- Hamming distance between two equal size strings measures the minimum number of replacements required to change one string into the other.
- Hamming distance is a character-based similarity measure.

$$\text{Hamming distance:} \quad D_{hamming}(x, y) = \sum_{i=1}^{n} 1_{x_i \neq y_i}$$

- Properties:
  - $\mathrm{D}_{hamming}(s_1, s_2) = 0 \Leftrightarrow s_1 = s_2$
  - $\mathrm{D}_{hamming}(s_1, s_2) \leqslant \mathrm{D}_{hamming}(s_1, s_3) + \mathrm{D}_{hamming}(s_2, s_3)$

- Application: mainly used in coding theory for error detection and correction (note that the following representation also exhibits the fact that the Hamming distance of binary chains is equivalent to the Manhattan distance)





Euclidean distance

Euclidean Distance

$$dist = \sqrt{\sum_{k=1}^{n} (p_k - q_k)^2}$$

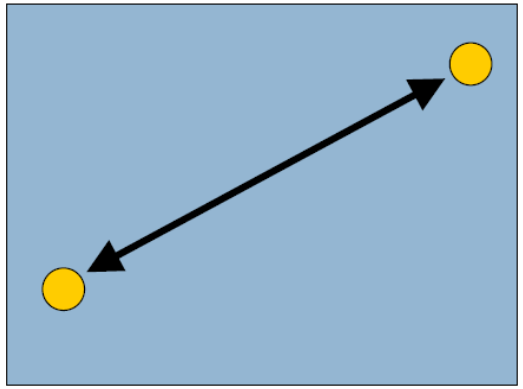
Where n is the number of dimensions (attributes) and p_k and q_k are, respectively, the k^{th} attributes (components) or data objects p and q.

Euclidean distance:

Point 1 is: $(x_1, x_2, \ldots, x_n)$

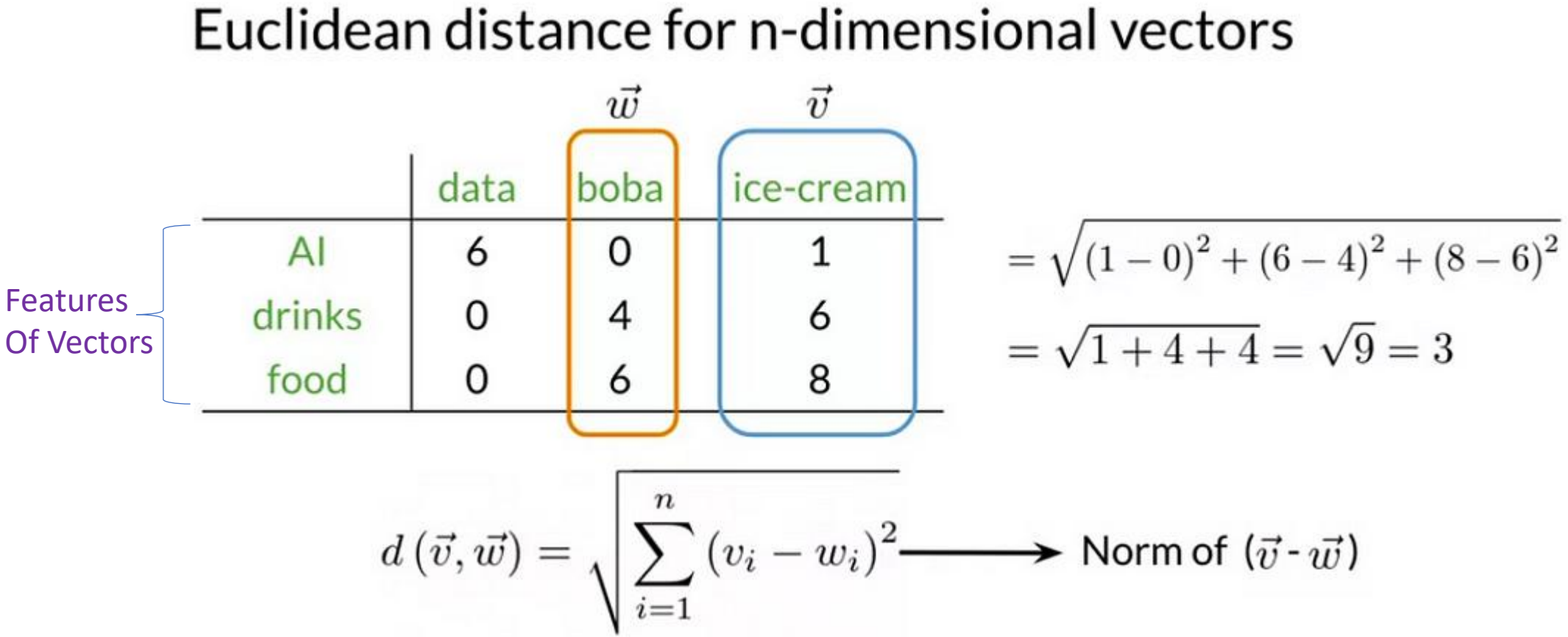
Point 2 is: $(y_1, y_2, \ldots, y_n)$

Euclidean distance is:



$$\sqrt{(y_1 - x_1)^2 + (y_2 - x_2)^2 + \ldots + (y_n - x_n)^2}$$

Euclidean Distance for n-dimensional Vectors



Euclidean distance between the vector v of the word ice cream and the vector representation w of the word the boba.

Minkowski Distance

Minkowski Distance is a generalization of Euclidean Distance

$$dist = \left(\sum_{k=1}^{n} | p_k - q_k |^r \right)^{\frac{1}{r}}$$

Where r is a parameter, n is the number of dimensions (attributes) and p_k and q_k are, respectively, the kth attributes (components) or data objects p and q.

Minkowski Distance: Examples

- $r = 1$. City block (Manhattan, taxicab, L_1 norm) distance.
  - A common example of this is the Hamming distance, which is just the number of bits that are different between two binary vectors
- $r = 2$. Euclidean distance
- $r \rightarrow \infty$. "supremum" (L_{max} norm, L_∞ norm) distance.
  - This is the maximum difference between any component of the vectors
  - Example: L_infinity of (1, 0, 2) and (6, 0, 3) = ??
  - Do not confuse *r* with *n*, i.e., all these distances are defined for all numbers of dimensions.

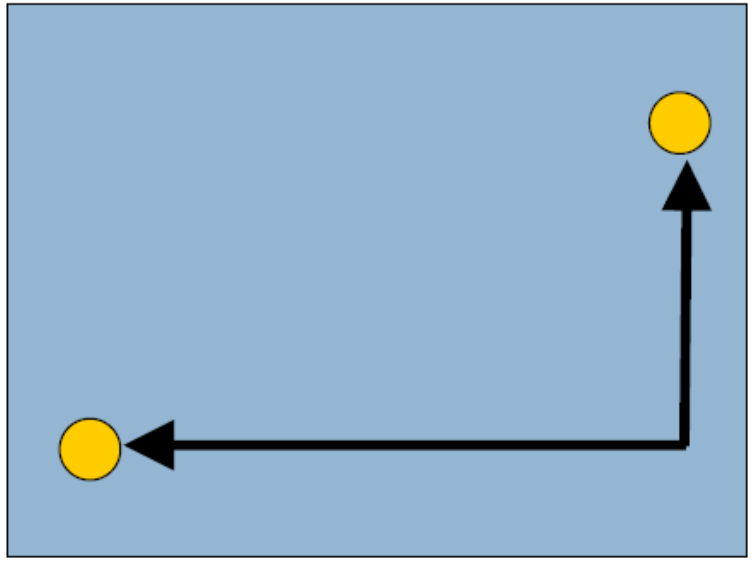
Manhattan distance

Manhattan distance
(aka city-block distance)

Point 1 is: $(x_1, x_2, \ldots, x_n)$

Point 2 is: $(y_1, y_2, \ldots, y_n)$

Manhattan distance is:



$$|y_1 - x_1| + |y_2 - x_2| + \ldots + |y_n - x_n|$$

(in case you don't know: $|x|$ is the absolute value of x.)

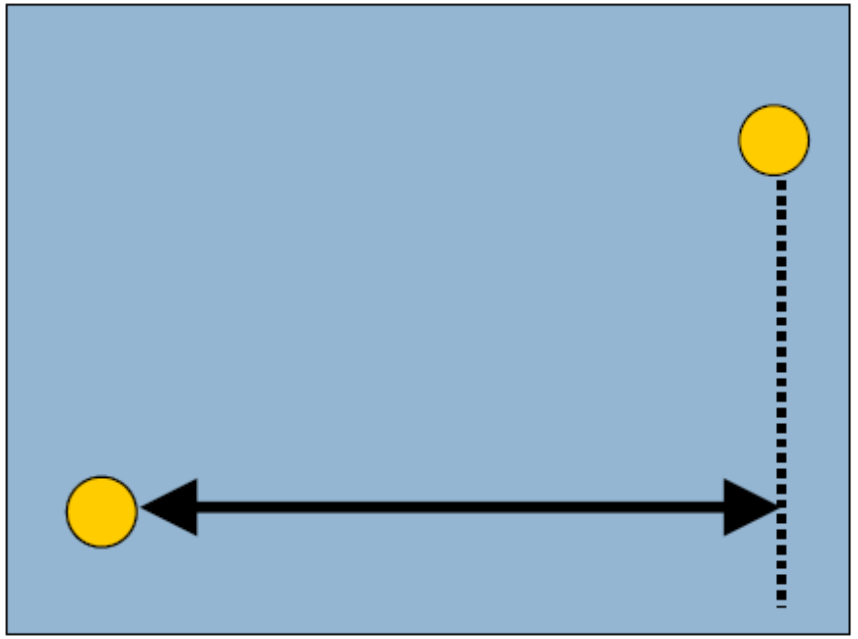
Chebychev Distance

Chebychev distance

Point 1 is: $(x_1, x_2, \ldots, x_n)$

Point 2 is: $(y_1, y_2, \ldots, y_n)$

Chebychev distance is:



$$\max\{|\, y_1 - x_1 \,|, |\, y_2 - x_2 \,|, \ldots, |\, y_n - x_n \,|\}$$

L1, L2, … Distances

point	x	y
p1	0	2
p2	2	0
p3	3	1
p4	5	1

L1	p1	p2	p3	p4
p1	0	4	4	6
p2	4	0	2	4
p3	4	2	0	2
p4	6	4	2	0

L2	p1	p2	p3	p4
p1	0	2.828	3.162	5.099
p2	2.828	0	1.414	3.162
p3	3.162	1.414	0	2
p4	5.099	3.162	2	0

L_∞	p1	p2	p3	p4
p1	0	2	3	5
p2	2	0	1	3
p3	3	1	0	2
p4	5	3	2	0

Distance Matrix

Key differences between Chebyshev Euclidean and Manhattan Distances

Distance	Formula	Applicability
Chebyshev Distance	$D_{Chebyshev}(p, q) = \max(\lvert p_1 - q_1 \rvert, \lvert p_2 - q_2 \rvert, \ldots, \lvert p_n - q_n \rvert)$	Useful for grid-based systems with free movement, like chess (king's moves) and certain robotic paths.
Euclidean Distance	$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$	Ideal for continuous spaces where straight-line distance is required, like geometry and physics.
Manhattan Distance	$\lvert x_1 - x_2 \rvert + \lvert y_1 - y_2 \rvert$	Relevant in grids with only horizontal and vertical movement, such as city blocks or chess (rook's moves).
Minkowski distance	$(\lvert X_1 - Y_1 \rvert^p + \lvert X_2 - Y_2 \rvert^p + \lvert X_2 - Y_2 \rvert^p)^{1/p}$	A flexible metric for various multidimensional spaces; often used in machine learning and clustering, for various values of p.

References

- NLP Demystified 5: Basic Bag-of-Words and Measuring Document Similarity
- https://flavien-vidal.medium.com/similarity-distances-for-natural-language-processing-16f63cd5ba55
- https://www.iitr.ac.in/media/facspace/patelfec/16Bit/slides/Lecture-2-Data-Preprocessing-Part-1.pdf (94-127