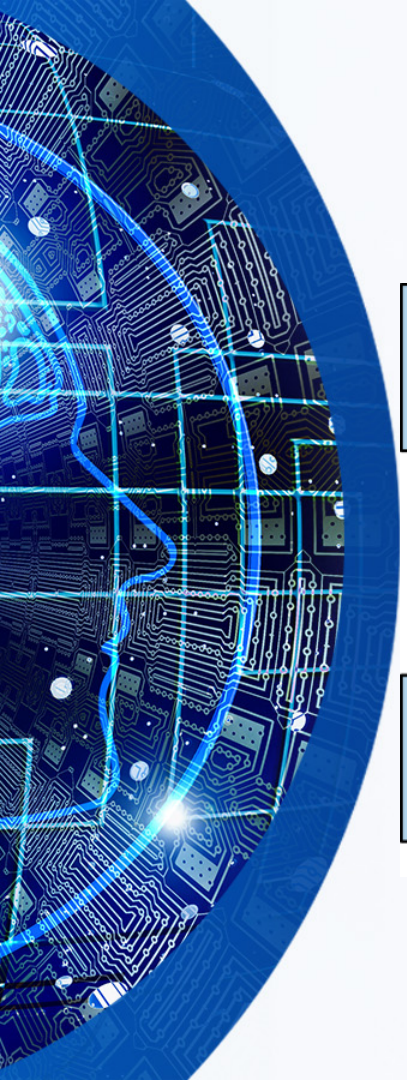
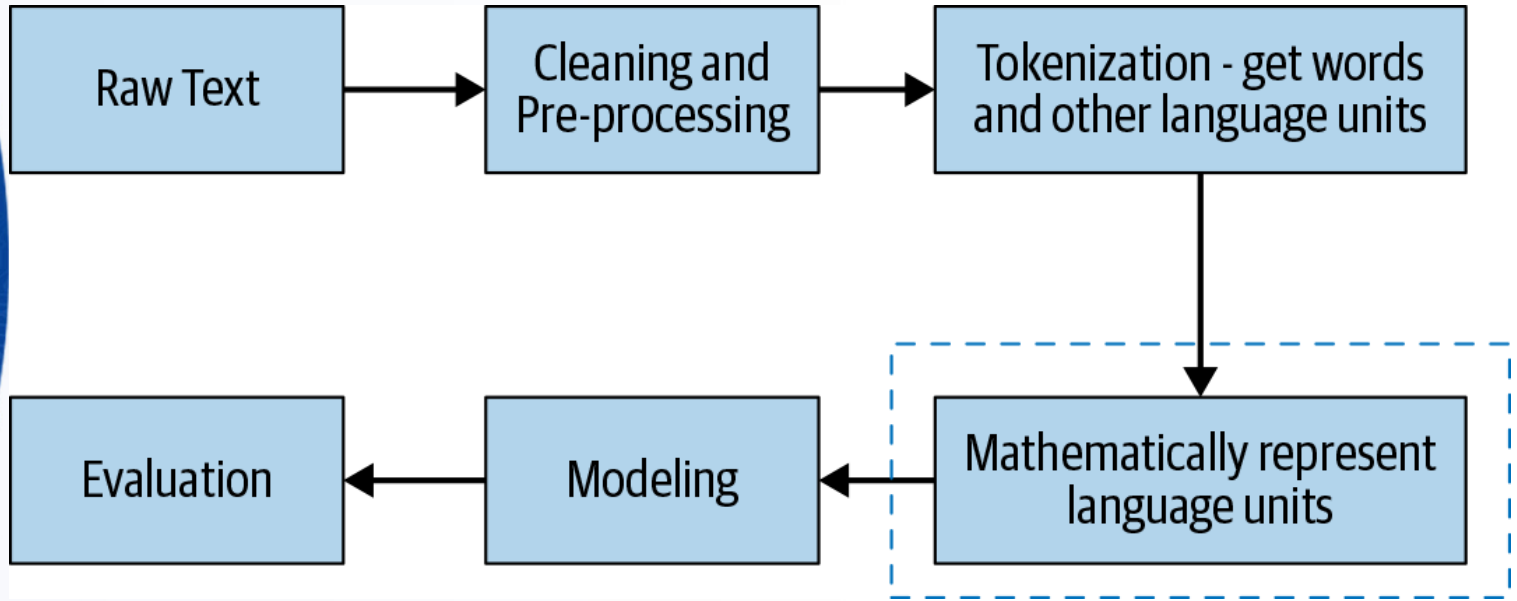
Natural Language Processing

Dr. Hamada Nayel

Lecture 04



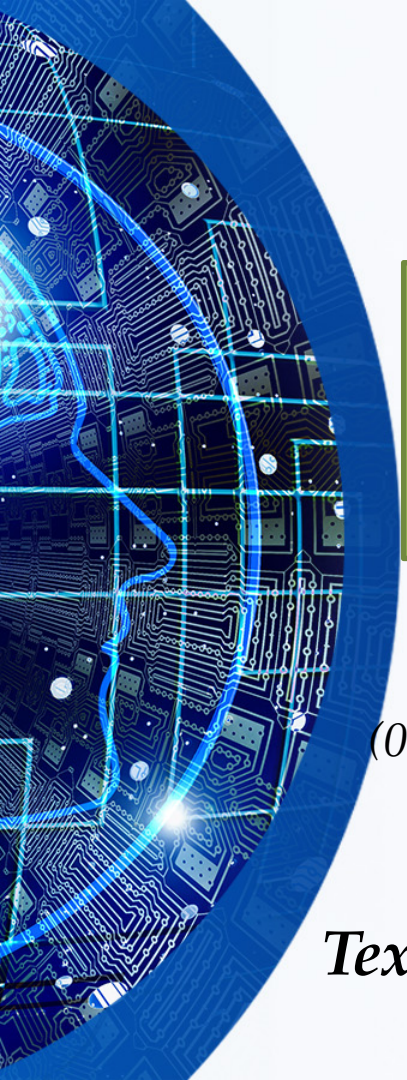
Text Representation

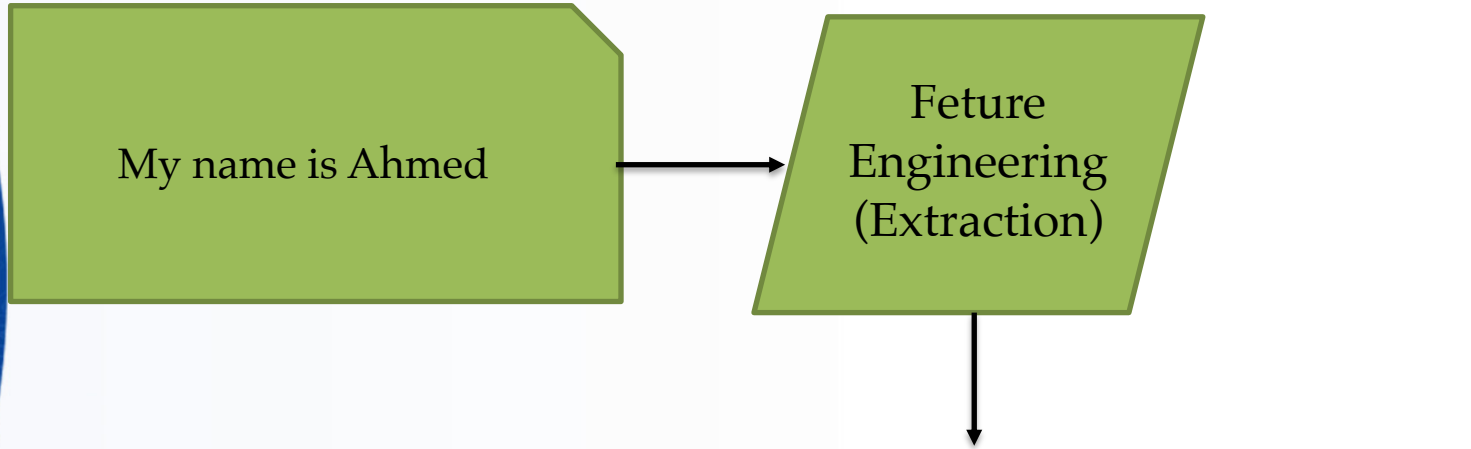


Scope of text representation in NLP pipeline

Text Representation

How do we go about doing feature engineering for text data?





Text representation

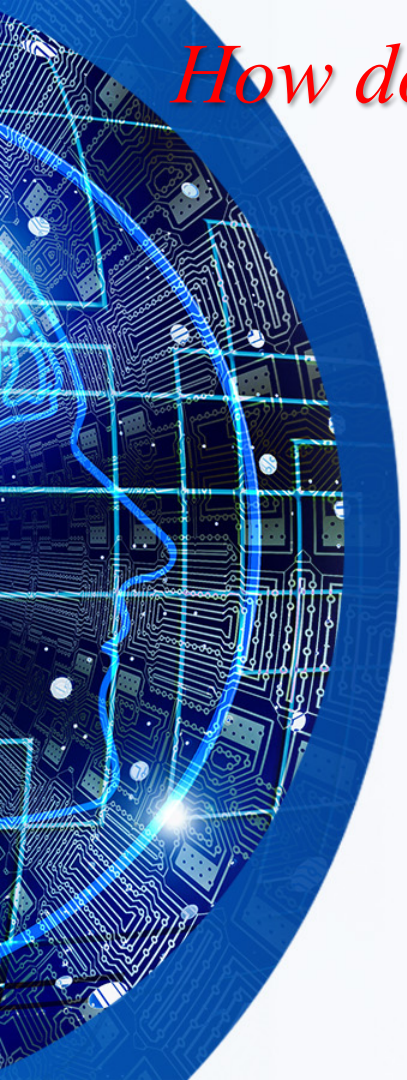
How do computers see? vs How do we see?

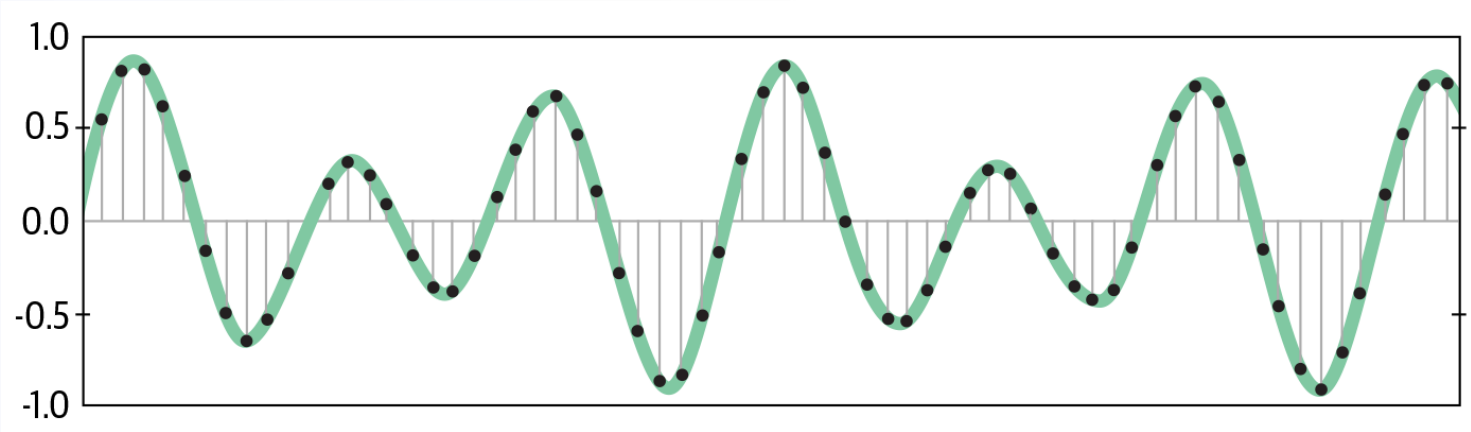
What We See

```
08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50
32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70
67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21
24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57
86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66
88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36
20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54
01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48
```

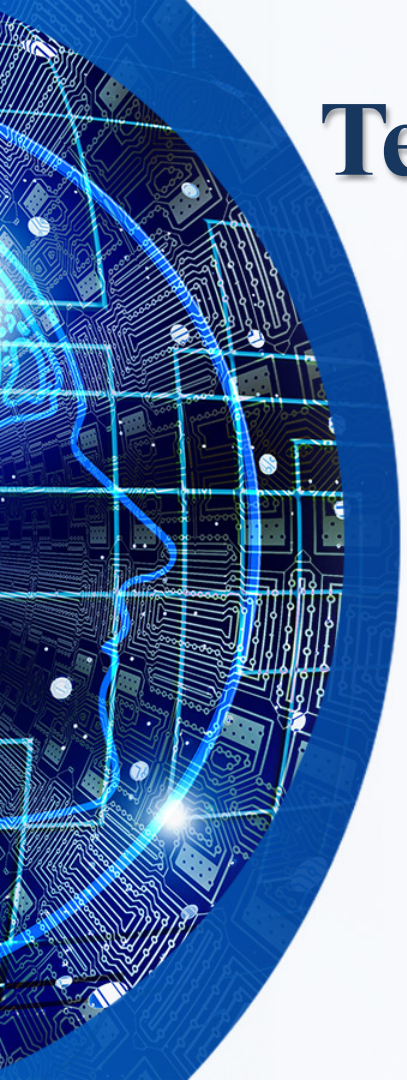
What Computers See

How do computers listen? vs How do we listen?



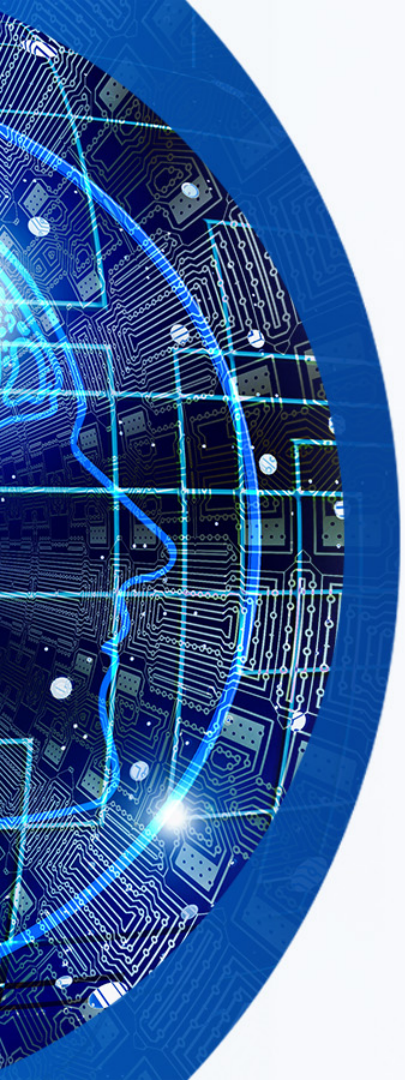



```
[-1274, -1252, -1160, -986, -792, -692, -614, -429, -286, -134, -57, -41,
-169, -456, -450, -541, -761, -1067, -1231, -1047, -952, -645, -489, -448,
-397, -212, 193, 114, -17, -110, 128, 261, 198, 390, 461, 772, 948, 1451,
1974, 2624, 3793, 4968, 5939, 6057, 6581, 7302, 7640, 7223, 6119, 5461,
4820, 4353, 3611, 2740, 2004, 1349, 1178, 1085, 901, 301, -262, -499,
-488, -707, -1406, -1997, -2377, -2494, -2605, -2675, -2627, -2500, -2148,
-1648, -970, -364, 13, 260, 494, 788, 1011, 938, 717, 507, 323, 324, 325,
350, 103, -113, 64, 176, 93, -249, -461, -606, -909, -1159, -1307, -1544]
```



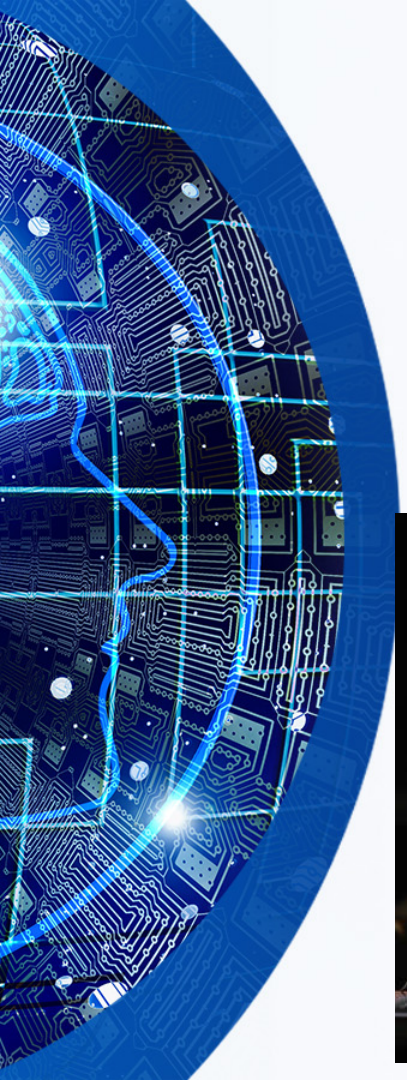
Text Representation Approaches

- Basic Vectorization Approaches
- Distributed Representations
- Universal Language Representation
- Handcrafted Features



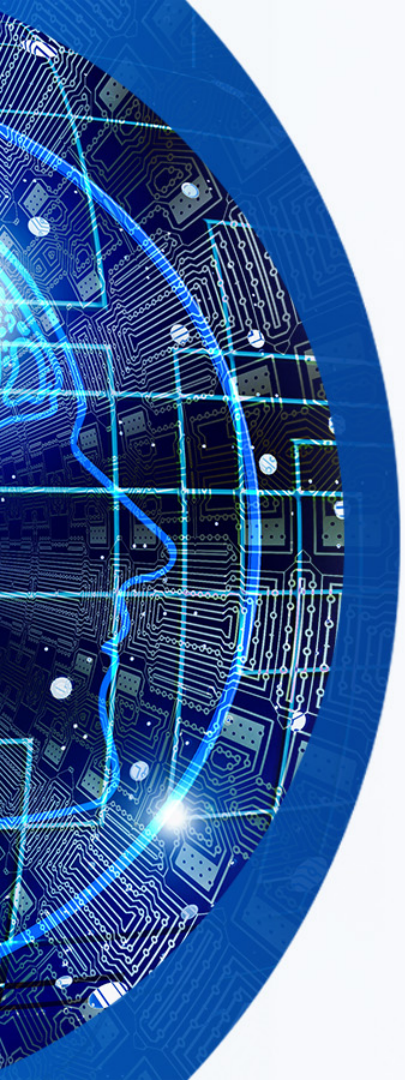
- In order to correctly *extract the meaning of the sentence*, the most crucial data points are:

  1. Break the sentence into *lexical units* such as words, and phrases
  2. Derive the *meaning* for each of the lexical units
  3. Understand the syntactic (*grammatical*) structure of the sentence
  4. Understand the *context* in which the sentence appears



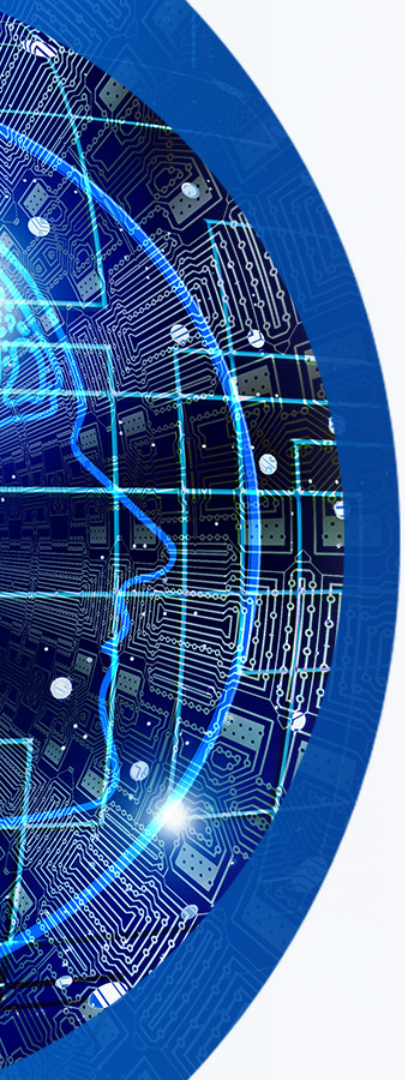
Good vs *Bad* Representation?

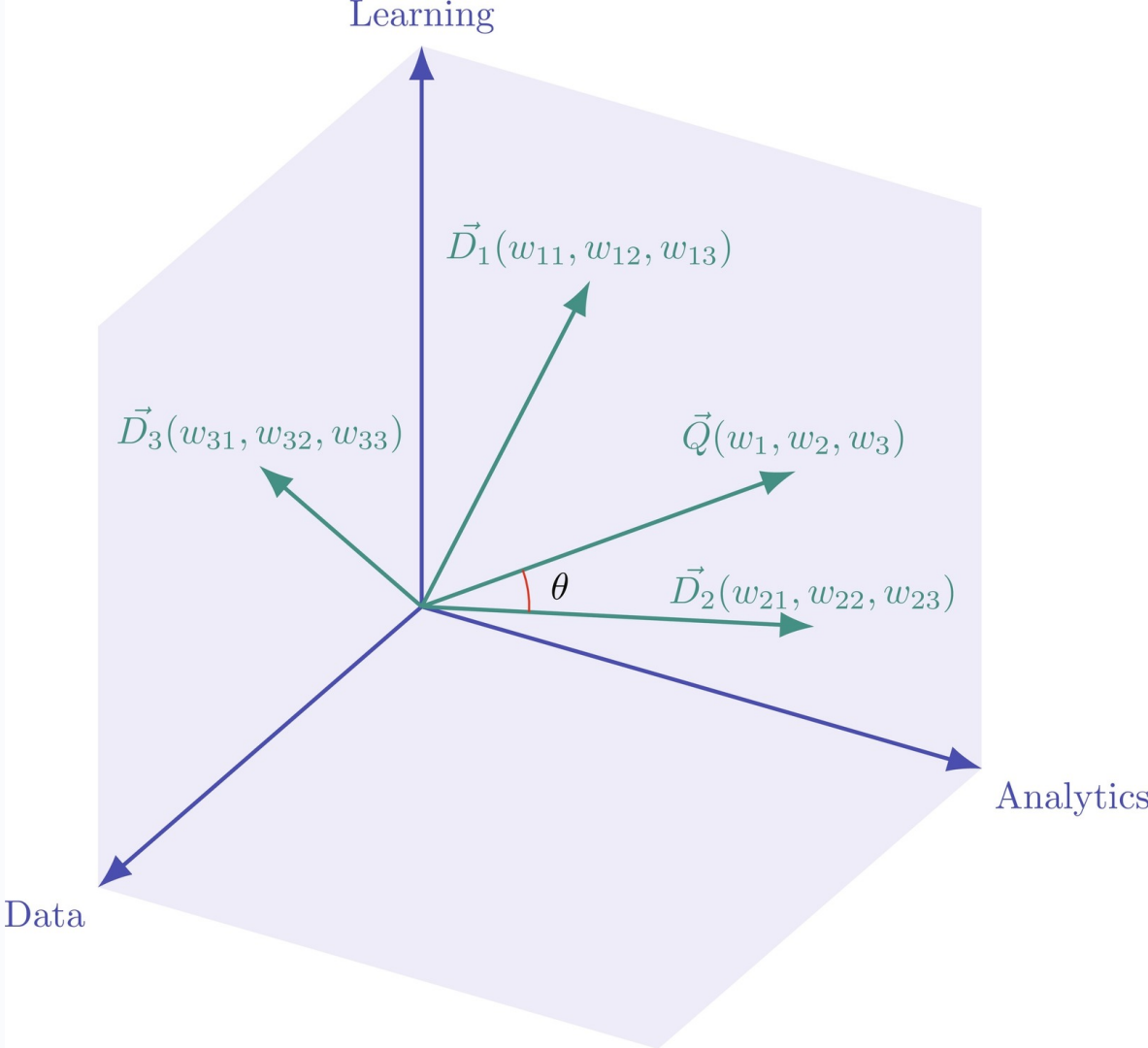
In NLP, feeding a good *text representation* to an ordinary algorithm will get you much farther compared to applying a *top-notch* algorithm to an ordinary text representation.

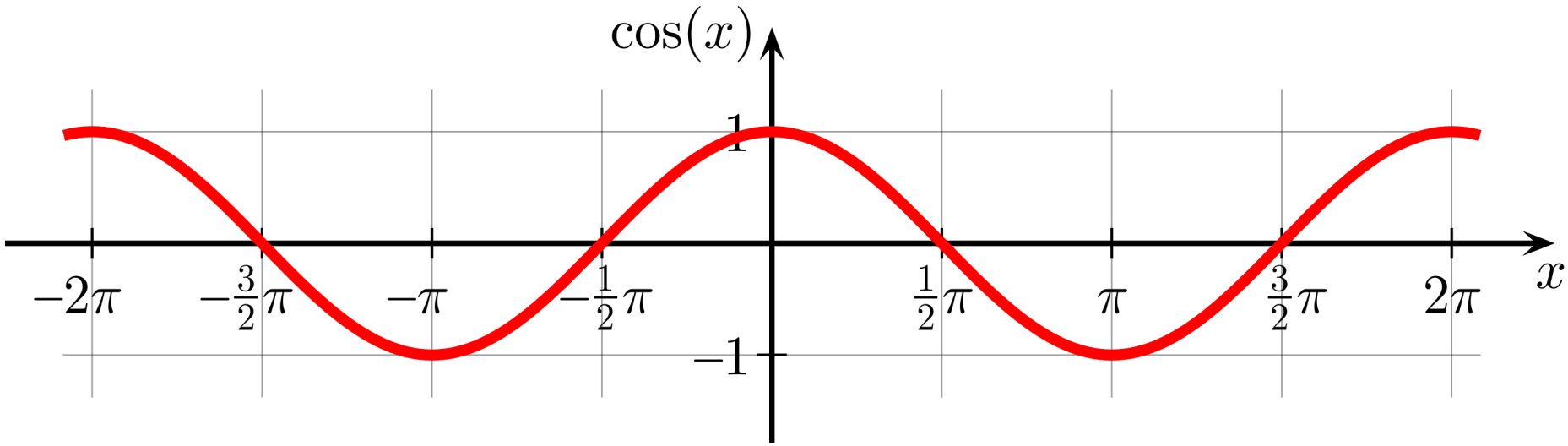


Vector Space Models (*VSM*)

- Fundamental to many ***IR*** (document classification, clustering and scoring documents on a query).
- ***VSM*** is a mathematical model that represents text units as vectors.
- Vectors of *identifiers*, such as index numbers in a corpus vocabulary.



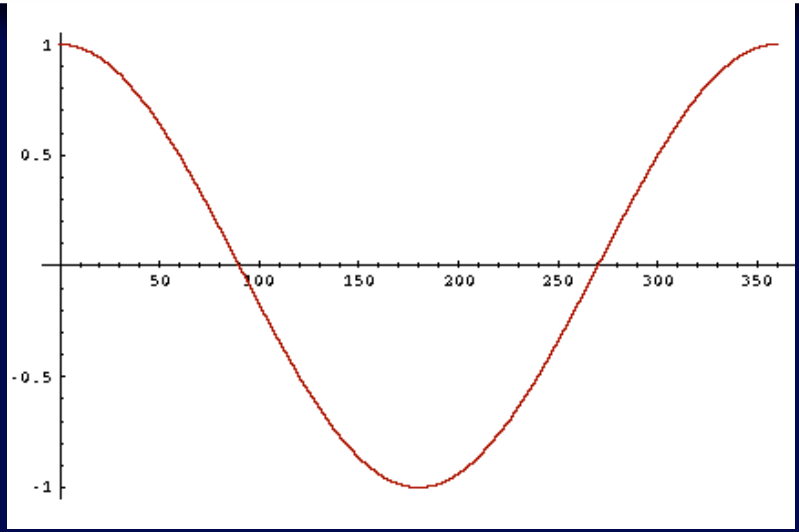

Learning
$\vec{D}_1(w_{11}, w_{12}, w_{13})$
$\vec{D}_3(w_{31}, w_{32}, w_{33})$
$\vec{Q}(w_1, w_2, w_3)$
θ
$\vec{D}_2(w_{21}, w_{22}, w_{23})$
Analytics
Data



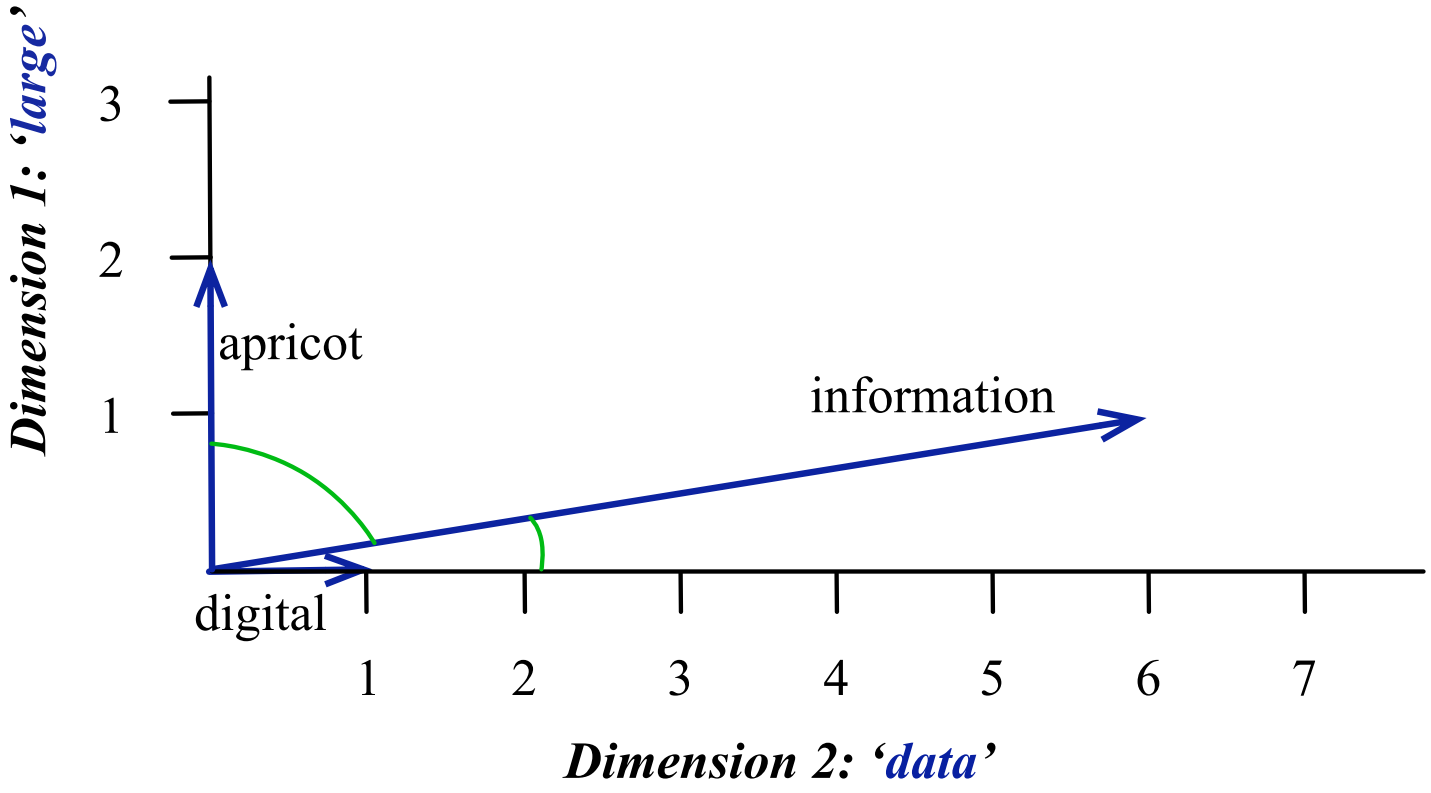
$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|_2 \|\mathbf{B}\|_2} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

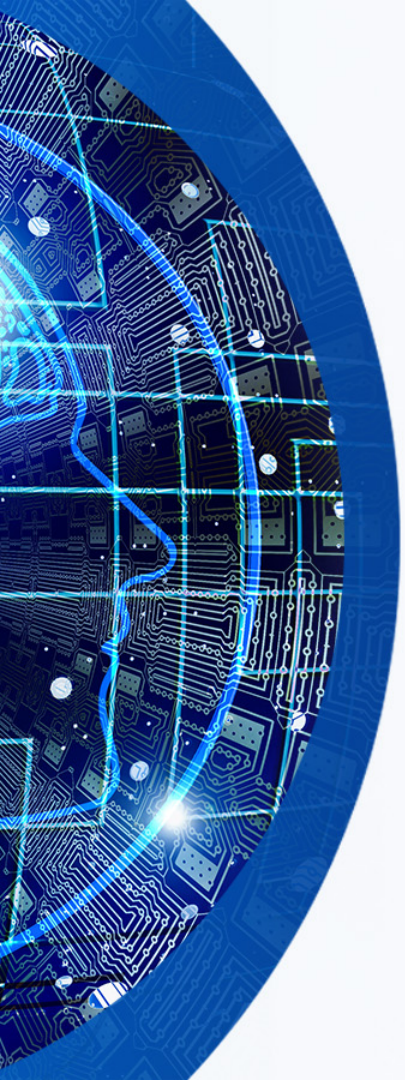
Cosine as a Similarity Metric

- **-1**: vectors point in opposite directions
- **+1**: vectors point in same directions
- **0** : vectors are orthogonal



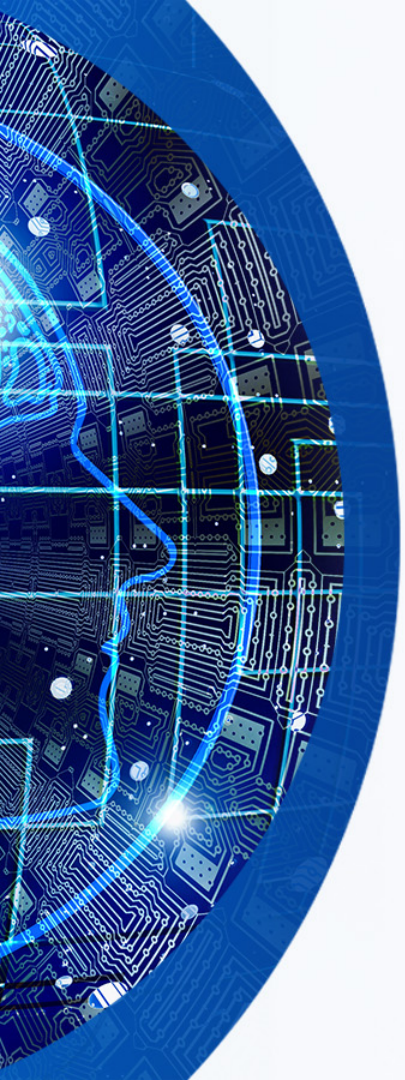
Cosine as a Similarity Metric





Basic Vectorization Approaches

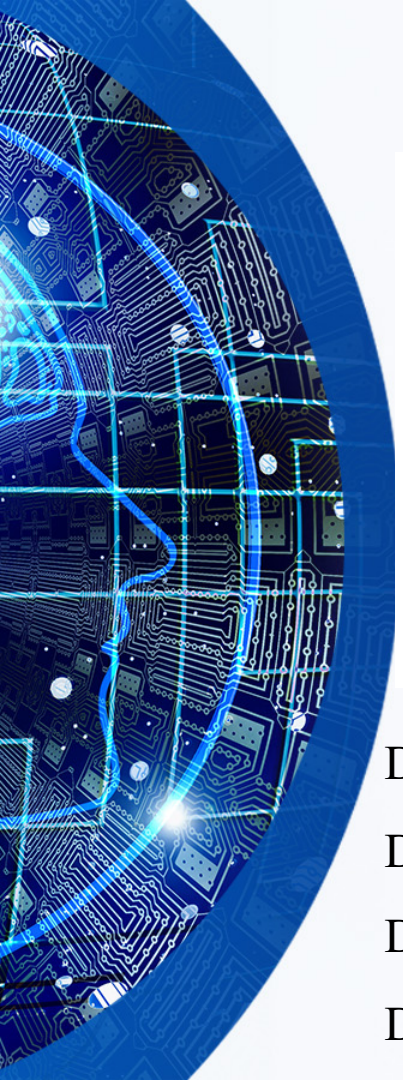
1. **One-Hot Encoding**
2. **Bag of Words (BoW)**
3. **Bag of N-Grams (BoN)**
4. **Term Frequency – Inverse Document Frequency (TF-IDF)**



One-Hot Encoding

- Each word w in the corpus vocabulary is given an ID w_{id} that is between **1** and $|V|$.
- V is the set of the corpus vocabulary
- $V = \{dog\ ,\ bites,\ man,\ meat,\ food\ ,\ eats\}$

$dog = 1,\ bites = 2,\ man = 3,\ meat = 4\ ,\ food = 5,\ eats = 6$



One-Hot Encoding: Toy corpus

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

dog = [**1**,0,0,0,0,0],

bites = [0,**1**,0,0,0,0],

man = [0,0,**1**,0,0,0]

meat = [0,0,0,**1**,0,0],

food = [0,0,0,0,**1**,0],

eats = [0,0,0,0,0,**1**]

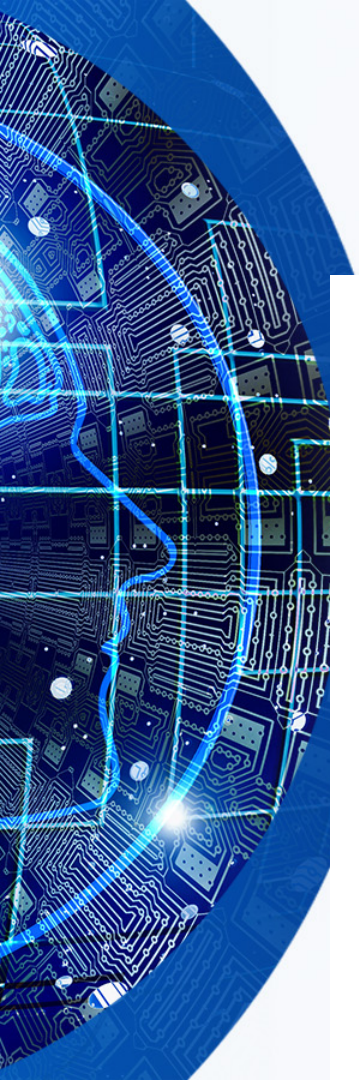
D1 = [[1 0 0 0 0 0] [0 1 0 0 0 0] [0 0 1 0 0 0]]

D2 = [[0 0 1 0 0 0] [0 1 0 0 0 0] [1 0 0 0 0 0]]

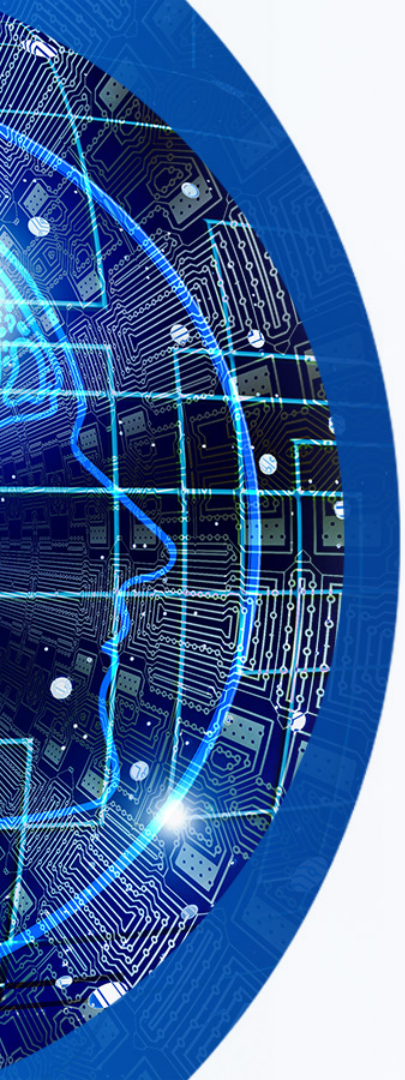
D3 = [[1 0 0 0 0 0] [0 0 0 0 0 1] [0 0 0 1 0 0]]

D4 = [[0 0 1 0 0 0] [0 0 0 0 0 1] [0 0 0 0 0 1]]

One-Hot Encoding: *Code*

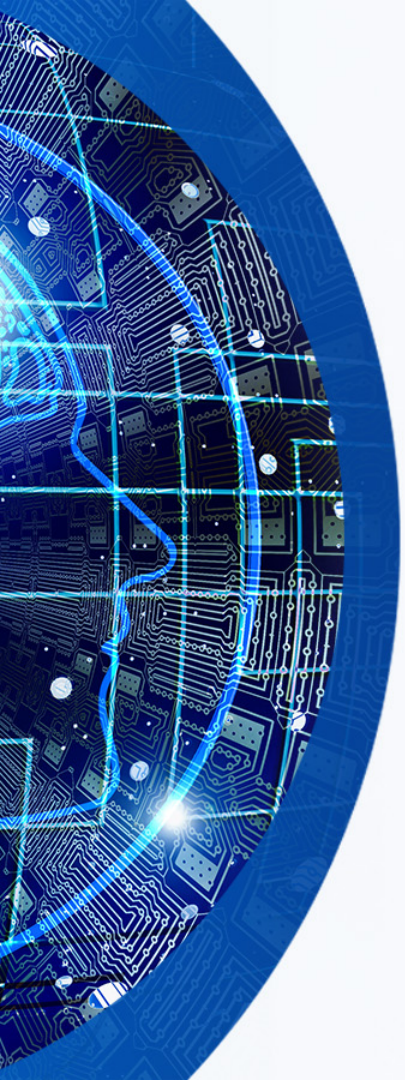


```
def get_onehot_vector(somestring):
  onehot_encoded = []
  for word in somestring.split():
            temp = [0]*len(vocab)
            if word in vocab:
                      temp[vocab[word]-1] = 1
            onehot_encoded.append(temp)
  return onehot_encoded

get_onehot_vector(processed_docs[1])
```



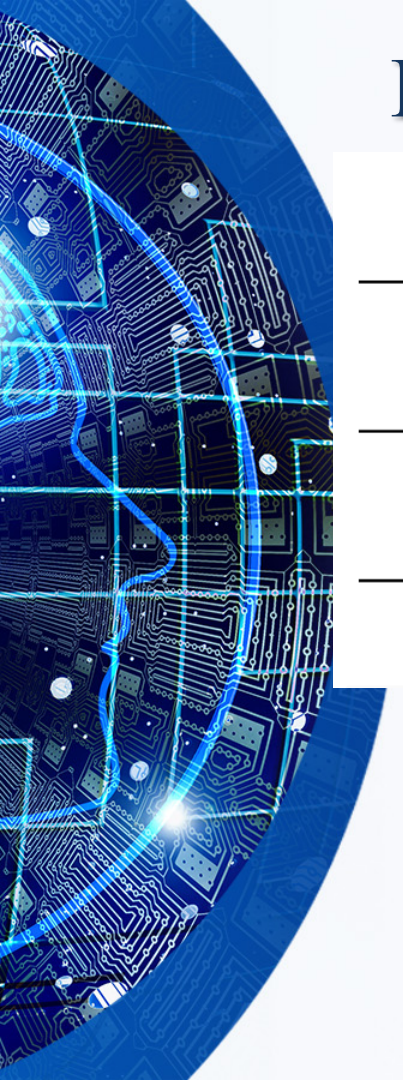
One-Hot Encoding: *Disadvantages*

- **A *sparse representation,* (many zeroes), computationally inefficient to *store*, *compute* with, and *learn* from (sparsity leads to *overfitting*).**
- **Not fixed-length representation for text, for most *learning algorithms*, we need the *feature vectors* to be of the *same length*.**
- **Words are atomic units, no notion of *(dis)similarity* between words. run, ran, and apple , semantically, they're very poor at capturing the meaning.**
- **Out of vocabulary (*OOV*) problem (*man eats fruit*)**



Bag of Words (*BoW*) Model

- **Represent the text under consideration as a *collection* (*bag*) of words while *ignoring* the *order* and *context*.**
- **Assume the text belonging to a given class in the dataset is characterized by a set of words.**
- **If two text pieces have nearly the same words, then they belong to the same *bag* (*class*).**
- **Thus, by analyzing the words present in a piece of text, one can identify the *class* (*bag*) it belongs to.**



Bag of Words (*BoW*) Model: *Toy Corpus*

D1 Dog bites man.

D2 Man bites dog.

D3 Dog eats meat.

D4 Man eats food.

dog = [**1**,0,0,0,0,0],

bites = [0,**1**,0,0,0,0],

man = [0,0,**1**,0,0,0]

meat = [0,0,0,**1**,0,0],

food = [0,0,0,0,**1**,0],

eats = [0,0,0,0,0,**1**]

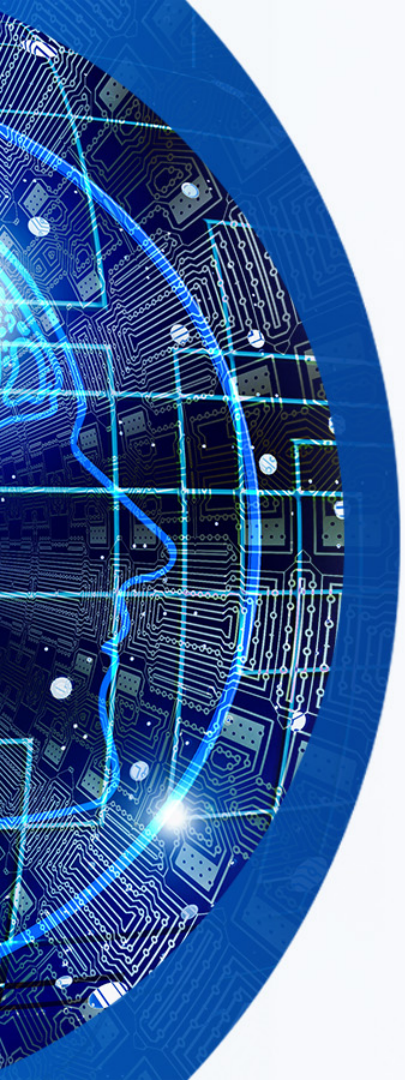
$\mathbf{R}^6$ = [dog, bites, man, meat, food, eat]

D1 = [1 1 1 0 0 0]

D2 = [1 1 1 0 0 0]

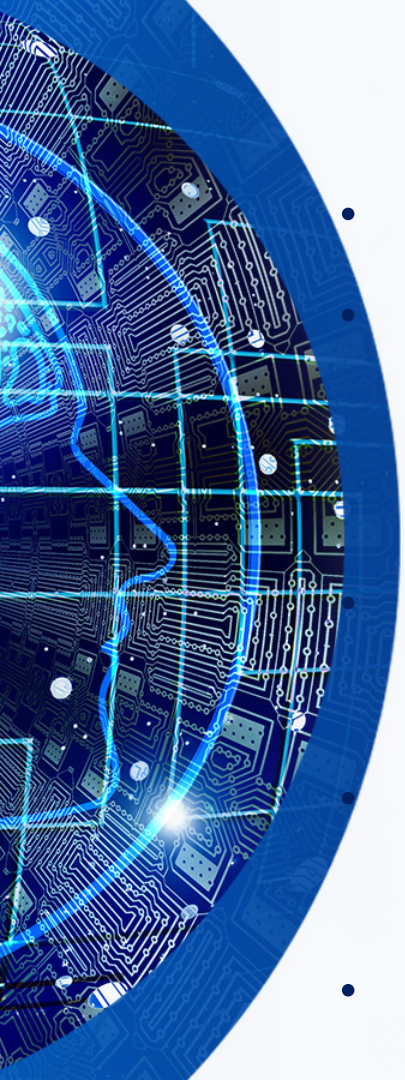
D3 = [1 0 0 1 0 1]

D4 = [0 0 1 0 1 1]



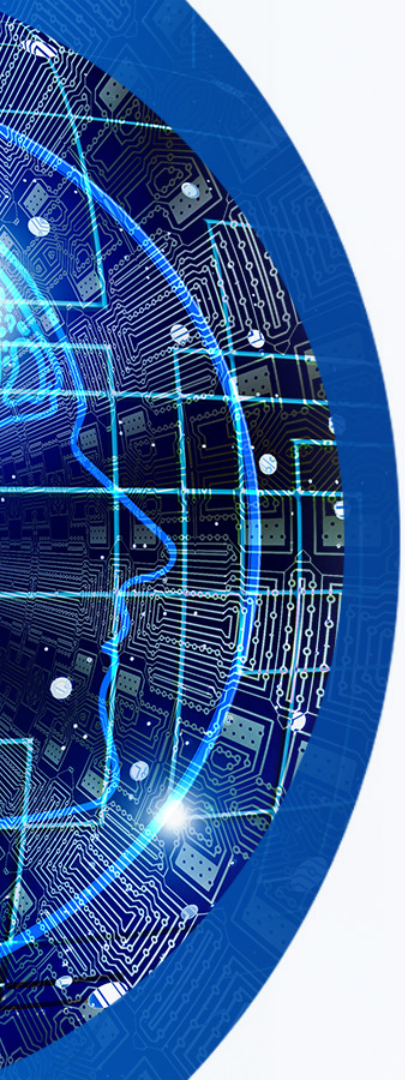
Bag of Words (*BoW*) Model: *Code*

```
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()

#Build a BOW representation for the corpus
bow_rep = count_vect.fit_transform(processed_docs)

#Look at the vocabulary mapping
print("Our vocabulary: ", count_vect.vocabulary_)

#See the BOW rep for first 2 documents
print("BoW representation for 'dog bites man': ", bow_rep[0].toarray())
print("BoW representation for 'man bites dog: ",bow_rep[1].toarray())

#Get the representation using this vocabulary, for a new text
temp = count_vect.transform(["dog and dog are friends"])
print("Bow representation for 'dog and dog are friends':",

temp.toarray())

count_vect = CountVectorizer(binary=True)
bow_rep_bin = count_vect.fit_transform(processed_docs)
temp = count_vect.transform(["dog and dog are friends"])
print("Bow representation for 'dog and dog are friends':", temp.toarray())
```



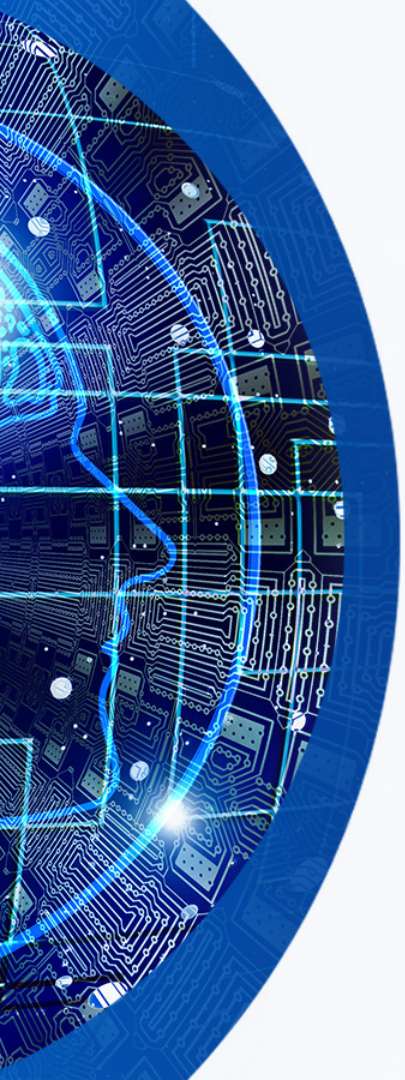
Bag of Words (*BoW*): *Advantages*

- **BoW is fairly *simple* to *understand* and *implement*.**
- **Documents having the *same* words will have representations *closer* to each other in *Euclidean space* compared to documents with completely *different* words.**
- **The vector space resulting from the BoW scheme captures the *semantic similarity* of documents.**
- **So if two documents have *similar vocabulary*, they'll be *closer* to each other in the *vector space* and *vice versa*.**
- **We have a *fixed-length* encoding for any sentence of arbitrary length.**



Bag of Words (*BoW*): *Disadvantages*

- Sparsity (*control* by limiting to *n* number of the *most frequent words*).
- It *does not* capture the *similarity* between *different words* that mean the same thing ("I run", "I ran", and "I ate".)
- Does not have any way to handle *OOV* (i.e., new words that were not seen in the corpus that was used to build the vectorizer).
- Word order information is lost in this representation "*Ali is quicker than Ahmed*" and "*Ahmed is quicker than Ali*" OR "*man bites dog*" and "*dog bites man*" will have the *same representation* in this scheme.



Bag of N-Grams (BoN)

- **Prev. Words as independent units.**
- **Prev. No notion of phrases or word ordering.**
- **The *bag-of-n-grams* (*BoN*) approach tries to remedy this.**
- **It does so by breaking text into chunks of *n-contiguous* words (tokens).**
- **Capture some *context*, which earlier approaches could not do.**
- **Each chunk is called an *n-gram*.**

Bag of N-Grams (BoN)

- The corpus vocabulary, V, is then nothing but a collection of all *n-grams* across the text corpus.
- Each document is represented by a vector of length |V|.
- This vector contains the frequency counts of *n-grams* present in the document and zero for the *n-grams* that are not present.
- At *n=1* (*unigram*)=====> BoN == BoW

  n=2, *bigram*

  n=3, *trigram*

  n=4, 4-gram,……..

Bag of N-Grams (BoN): *Example*

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

Let ***n=2***, bigram,

V= {“dog bites”, “bites man”, “man bites”, “bites dog”, “dog eats”, “eats meat”, “man eats”, “eats food”},

$|V| = 8$,

D1 = [1 1 0 0 0 0 0 0]

D2 = [0 0 1 1 0 0 0 0]

D3 = [0 0 0 0 1 1 0 0]

D4 = [0 0 0 0 0 0 1 1]

Bag of N-Grams (BoN): *Example*

```
#n-gram vectorization example with count vectorizer and uni, bi, trigrams
count_vect = CountVectorizer(ngram_range=(1,3))

#Build a BOW representation for the corpus
bow_rep = count_vect.fit_transform(processed_docs)

 #Look at the vocabulary mapping
 print("Our vocabulary: ", count_vect.vocabulary_)

 #Get the representation using this vocabulary, for a new text
 temp = count_vect.transform(["dog and dog are friends"])
 print("Bow representation for 'dog and dog are friends':", temp.toarray())
```

Here are the main pros and cons of BoN:

Bag of N-Grams (BoN): *pros vs cons*

- It captures some context and word-order information in the form of ***n-grams***.
- Vector space captures some semantic similarity.
- Documents having the same ***n-grams*** will have their vectors closer to each other in *Euclidean* space as compared to documents with completely different *n-grams*.

- As ***n*** increases, sparsity increases rapidly.
- It still provides no way to address the ***OOV*** problem.

Term Document Matrix (TDM)

Let ***n=1***, unigram (BoW)

V= {“dog”, “bites”, “man”, “eats”, “meat”, “food”},

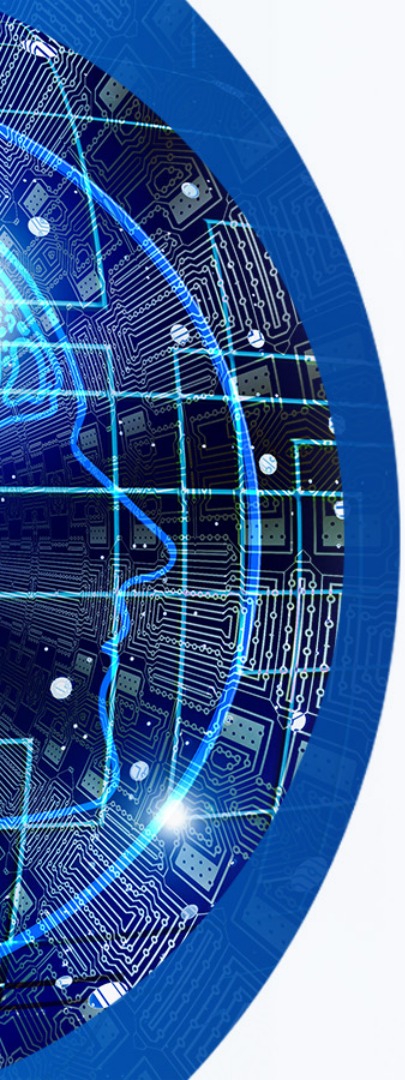
$|V| = 6$ and we have $|D| = 4$.

TDM is calculated as follows.

TDM =

	dog	bites	man	eats	meat	food
D1	1	1	1	0	0	0
D2	1	1	1	0	0	0
D3	1	0	0	1	1	0
D4	0	0	1	1	0	1

Term Document Matrix (TDM)



TDM

- Binary TDM
- Count TDM


thank
you