Traditional Text Representation

Chapter 3

Text Representation

Text representation is the process of converting unstructured text data into a structured format (usually numerical) that a machine can understand

Raw text (e.g., "The cat sat on the mat") can’t be directly processed by machine learning models

We need to represent or transform it into numbers (vectors or matrices) that preserve its meaning, context, and structure

How can you convert them to numbers that will be meaningful to the machine?

Encoding

In NLP, encoding is the crucial process of converting human-readable text (words, sentences) into numerical formats (vectors or numbers) that machine learning models can understand and process, enabling tasks like translation or sentiment analysis by capturing meaning and context through techniques such as one-hot encoding, word embeddings (Word2Vec, GloVe), or transformer-based contextual embeddings.

Label Encoding

It is a method used to convert categorical text data into numeric format, where each unique word or label is assigned an integer value commonly used for target labels in classification tasks helps reduce memory usage and improves performance for simpler algorithms

words = ["apple", "banana", "cherry"]

Text	Encoding
apple	0
banana	1
cherry	2

One Hot Encoding

It is a technique used in Natural Language Processing (NLP) to convert categorical text data into binary vectors one of the simplest and most interpretable methods to represent words or characters for machine learning models

Each unique word is represented as a vector where:

- Only one element is 1 (hot) - the index corresponding to that word
- All other elements are 0

Text	Encoding
apple	[1,0,0]
banana	[0,1,0]
cherry	[0,0,1]

Challenges in encoding

1. Large Vocabularies:

With a large number of unique words (high cardinality), each word gets its own column in the one-hot encoded matrix

This results in a very high-dimensional vector, increasing memoryusage and computational cost.

Challenges in encoding

2. Sparse Matrices:

Most elements in the one-hot encoded vector are zeros, making the data sparse.

This sparsity can slow down model training, especially for algorithms that don’t handle it well.

Challenges in encoding

3. No Word Relationships:

One-hot encoding treats all words as completely independent, even if they have semantic similarities (e.g., "king" and "queen" are as different as "king" and "table").

Challenges in encoding

4. No Contextual Understanding:

It doesn’t capture the contextual meaning of words, such as the different meanings of "bank" (financial vs. river bank)

Challenges in encoding

5. Out-of-Vocabulary (OOV) Words:

If a word is not present in the training vocabulary, it cannot be represented during deployment.

Bag of Words

Bag of Words (BoW) is a technique used in NLP to convert text data into numerical feature vectors that can be fed into a machine

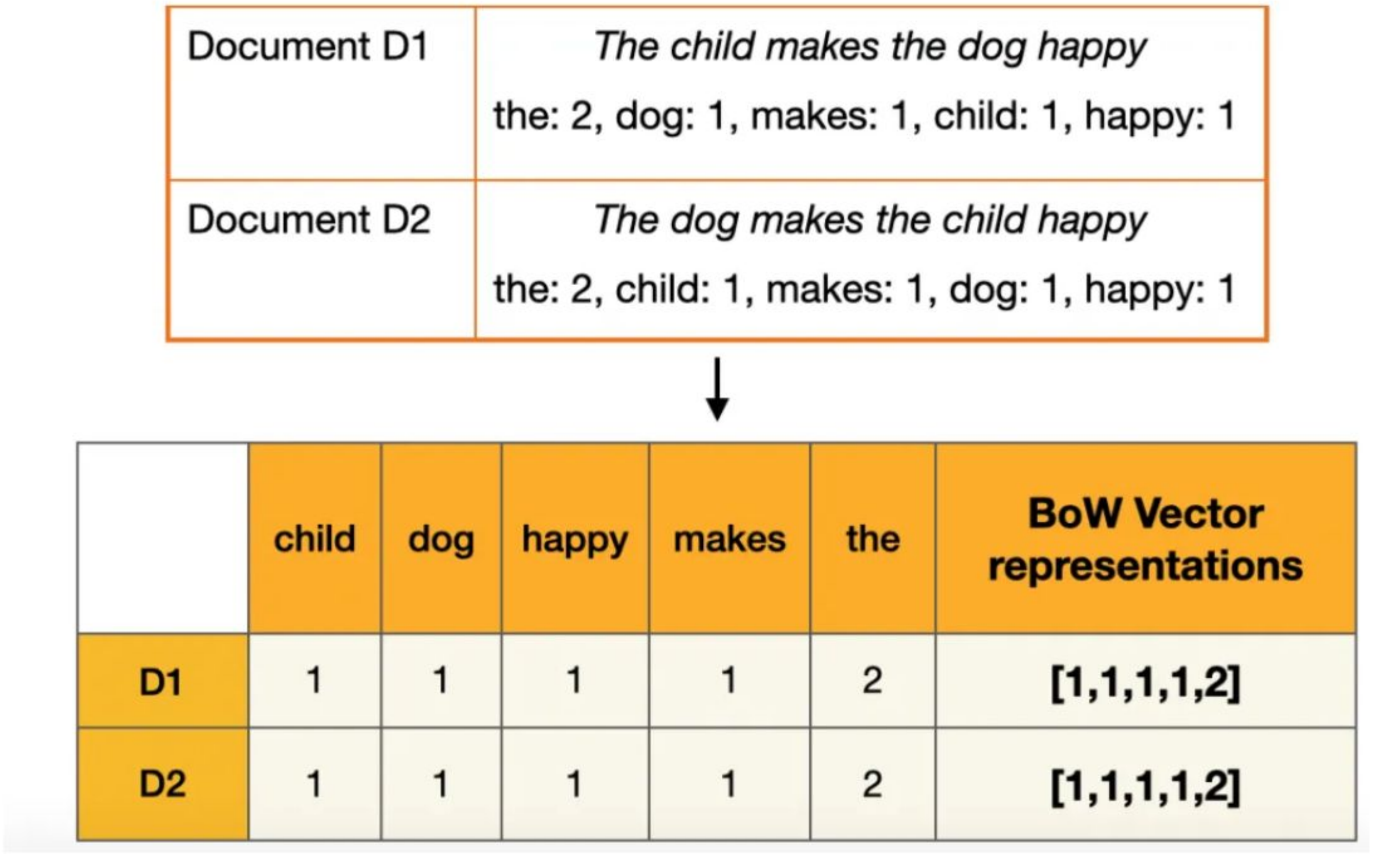
The BoW model represents a text (such as a sentence or document) as a multiset (or “bag”) of its words, ignoring grammar and word order but keeping multiplicity (frequency) of the words

John is quicker than Mary and Mary is quicker than John both are represented the same way

Bow: Binary

	it	is	a	cat	dog	my	not	wolf	old
“It is a dog”	1	1	1	0	1	0	0	0	0
“My cat is old”	0	1	0	1	0	1	0	0	0
“It is not a dog, it is a wolf”	1	1	1	0	1	0	1	1	0

Bow: Word counts



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]**

Steps in Bow

Step 1: Build the Vocabulary

- Collect all unique words from the entire text corpus
- Each unique word becomes a feature in your representation

Step 2: Create Vectors

- For each document or sentence, count how many times each word from the vocabulary occurs.
- Represent the document as a vector of word counts.

Bag of n grams

D1: This movie is very good.

D2: This movie is not good.

BOW

	This	movie	is	very	not	good
“This movie is very good”	1	1	1	1	0	1
“This movie is not good”	1	1	1	0	1	1

Bag of n grams

D1: This movie is very good.

D2: This movie is not good.

BOW

	This	movie	is	very	not	good	This movie	Movie is	Very good	Not good
“This movie is very good”	1	1	1	1	0	1	1	1	1	0
“This movie is not good”	1	1	1	0	1	1	1	1	0	1

Challenges in Bow

1. Ignores Word Order:

   BoW models only consider word frequency, disregarding the order in which words appear in a sentence.

   "dog bites man" vs "man bites dog" are same in BoW

2. Ignores Context and Semantic Meaning:

   BoW models treat each word as an independent entity, ignoring its context and potential semantic relationships with other words

   For example, the phrases "not good" and "good" would be treated as separate words, even though they have opposite meanings.

3. Insensitivity to Grammatical Structure:

   BoW models are insensitive to grammatical structures and punctuation, potentially leading to the same BoW vector for

   sentences with different meanings, such as "Let's eat, Grandma" and "Let's eat Grandma".

Term Frequency Inverse Document Frequency(TF-IDF)

TF-iDF term weighting scheme:

TF-iDF measures the relative concentration of a term in a given piece of document

- Term frequency (TF)
- Inverse document frequency (iDF)

Term Frequency

It is a statistical measure used to evaluate how important a word is to a document relative to a collection (corpus) of documents

Term frequency $tf_{t,d}$ is a measure that denotes how frequently the term t appears in the document d

$$tf_{t,d} = \frac{frequency\ of\ term\ 't'\ in\ documents'd'}{total\ terms\ in\ document\ 'd'}$$

Term Frequency

If a term appears 5 times in a 100-word document, then

Term frequency = 5/100 = 0.05

A document d1 with 10 occurrences of a term t is probably more relevant than a document d2 with 1 occurrence

But is it 10 times more relevant?

Term Frequency

A document d1 contains 100,000 tokens and 4 occurrences of term t whereas the document d2 contains 500 tokens and 3 occurrences of term t.

Which document is more relevant?

Relevance does not increase linearly with term frequency

Raw term frequency does not account for document length

Why “10 times the term frequency” ≠ “10 times the relevance”

If you treated relevance as strictly proportional to raw term frequency

$$tf(d1, t) = 1 \Rightarrow contribution \sim 1$$
$$tf(d2, t) = 10 \Rightarrow contribution \sim 10$$

That implies document d2 is 10 times better on that term. Empirically this does not match user behavior.

- The first occurrence of a term gives a strong signal that the document is about that topic.
- The next few occurrences refine that signal somewhat.
- After some point, more repetitions mostly indicate style, redundancy, or length, not a proportionally higher relevance.

So the relationship between tf and “evidence of relevance” is sublinear, not linear.

Log Scaled term frequency

$$tf_{t,d} = \begin{cases} 1 + \log_{10}(f_{t,d}) & \text{if } f_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

tf = 1=> 1 + log(1) = 1

tf = 10 => 1+log(10) = 2

So the weight roughly doubles going from 1 to 10 occurrences not 10X.

Term Frequency

Raw term frequency does not account for document length

- Let's normalize with the frequency of the most frequent term in the document
- One of the technique is to normalize the ft,d weights of all terms occurring in a document by the maximum ft,d in that document
  - For each document d, let $f_{max}(d) = \max_{\tau \in d} f_{\tau,d}$ where τ ranges over all terms in d
  - To compute a normalized term frequency for each term t in document d by

$$tf_{t,d} = \frac{f_{t,d}}{f_{\max}(d)}$$

- log TF formula

$$tf_{t,d} = \begin{cases} \frac{1+\log_{10}(f_{t,d})}{1+\log_{10}(f_{\max}(d))} & \text{if } f_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Desired weight for rare terms

Term frequency does not capture the importance of terms that occur rarely across individual documents of a collection

Rare terms are more informative than frequent terms

- Recall stop words

Consider a term in the query that is rare in the collection (e.g.,arachnocentric)

A document containing this term is very likely to be relevant to the Query

- We want a high weight for rare terms like arachnocentric

Desired weight for frequent terms

Frequent terms are less informative than rare terms like good, increase, shop

For frequent terms, we want want positive weights

- but lower weights than for rare terms

Inverse Document Frequency

We want high weights for rare terms like arachnocentric

We want low (positive) weights for frequent words like good, increase

We will use document frequency to factor this into computing the matching score.

- The document frequency df_t is the number of documents in the collection that the term t occurs in

Zipf’s law

How many frequent vs. infrequent terms should we expect in a collection?

- In natural language, there are a few very frequent terms and very many very rare terms

In any given corpus of document that the relationship between the frequency that a word is used and its rank has been the subject of study

Let cf_i is collection frequency (the number of occurrences of the term t_i) in the collection.

Zipf's law

Zipf's law states that the frequency that a word appears is inversely proportional to its rank

- The ith most frequent term has frequency cf_i proportional to 1/i

$$cf_i \sim \frac{1}{t_i}$$

Zipf'law

Based on observation: frequent words and rare words

- "the" and "of" make up 10% of all occurrences
- hardly ever see rare words like archnocentric, aardvark

So if the most frequent term (the) occurs cf1 times, then the second most frequent term (of) has half as many occurrences cf2 =½ cf1

and the third most frequent term (and) has a third as many occurrences cf3 =⅓ cf1

Equivalent: $cf_i = p \cdot i^k$ and $\log cf_i = \log p + k \log i$ (for $k=-1$)

Inverse Document Frequency

Terms are distributed in a text according to Zipf's Law

- we sort the vocabulary terms in decreasing order of document frequencies

Let N be the total number of documents in a collection, dft be the document frequency of term t

The inverse document frequency of a term t, denoted idft , is given by

$idf_t = \log_{10(}N/df_t)$

Inverse document frequency (IDF) is a single value for the term on the whole document collection D (does not depend on particular document)

Tf-IDF weights

The weight for the term ti within the document dj is computed by multiplying the TF (local) and IDF (global) components

$$\begin{aligned} w_{i,j} &= tf_{t_i,d_j} \times idf_{t_i} \\ &= \frac{1 + \log_{10}(f_{t_i,d_j})}{1 + \log_{10}(\max_{\tau \in d_j} f_{\tau,d_j})} \times \log_{10} \frac{N}{df_{t_i}} \end{aligned}$$

Example

Document 1: "The cat sat on the mat."

Document 2: "The dog played in the park."

Document 3: "Cats and dogs are great pets."

Calculate the TF-IDF score of the word "cat" using Raw tf.

Document 1: the, cat, sat, on, the, mat

Total words: 6

Occurrences of "cat": 1

Document 2: the, dog, play, in, the, park

Total words: 6

Occurrences of "cat": 0

Document 3: cat, and, dog, are, great, pet

Total words: 6

Occurrences of "cat": 1

we know that

$$tf_{t,d} = \frac{\text{frequency of term 't' in documents 'd'}}{\text{total terms in documents 'd'}}$$
$$idf_t = \log_{10} \frac{\text{total number of documents}}{\text{total documents with term 't'}}$$
$$tf - idf = tf_{t,d} \times idf_t$$

Doc	tf	idf	tf-idf
1	1/6		0.029
2	0/6	$\log_{10} \frac{3}{2}$	0
3	1/6		0.029

TF-IDF Example

1. d1: "The sky is blue.
2. d2: "The sun is bright today."
3. d3: "The sun in the sky is bright."
4. d4: "We can see the shining sun, the bright sun."

Task: Determine the tf-idf scores for each term in each document after removing the stopwords

Step1: Filter out the stopwords

- d1: "sky blue"
- d2: "sun bright today"
- d3: "sun sky bright"
- d4: "can see shining sun bright sun"

Step2: Compute TF, therefore, we find document-word matrix and then normalize the rows to sum to 1

$f_{t,d}$

	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

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

	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

Step3: Compute IDF: Find the number of documents in which each word occurs, then compute the formula:

$f_{t,d}$

	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**

$N = 4$

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

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

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

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

Step4: Compute TF-IDF: Multiply TF and IDF scores.

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

	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

x

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

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

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

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

	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

Problems with tf-idf

The idf part is fine, but the tf part has several problems

Let w be a word, and D1 and D2 be two documents

Problem 1: assume that D1 is longer than D2

- Then tf for w in D1 tends to be larger than tf for w in D2, because D1 is longer, not because it is more about w.

Problem 2: assume that D1 and D2 have the same length,

- The tf if w in D1 is twice the tf of w in D2
- Then it is reasonable to assume that D1 is more about w than D2 but just a little more and not twice more

If the term is common in one article, but relatively rare elsewhere

- then the TFIDF score will be high
- Documents that have higher TFIDF score would be considered as very relevant to the search term

BM25

Best Match term Weighting improves TF-IDF by casting relevance as a probability problem.

$$\text{BM25 score} = tf^{*} . \log_{10} \frac{N}{df}$$

$$tf^{*} = tf \frac{k+1}{k(1-b+b \cdot DL/DL_{avg}) + tf}$$

where tf= term frequency, DL = document length, DL_{avg} = average document length

For BM25: k= 1.75,b= 0.25 are tuning parameters

Normalize by the length of the document

The essence of document length normalization: reward short documents and penalize long documents

Average document length of a collection DL_{avg} , is central to this approach

Documents whose length is less than DL_{avg}: short documents rewarded proportionately by increasing their relevance score through boosting the actual frequency of term occurrence

Documents whose length is greater than DL_{avg}: long documents penalized by decreasing their relevance score

The reward/penalty increases linearly, and is dependent on how short/long a document is with reference to DL_{avg}

Normalization: $\alpha = (1 - b + b \cdot DL/DL_{avg})$

Normalization: $\alpha = (1 - b + b \cdot DL/DL_{avg})$

1. When $b = 0 \Rightarrow \alpha = 1$ No normalization
2. When $b = 1 \Rightarrow \alpha = DL/DL_{avg}$ Full normalization

This gives us

$$\begin{aligned} tf^* &= tf/\alpha \cdot \frac{k+1}{k + tf/\alpha} \\ &= tf \cdot \frac{k+1}{k \cdot \alpha + tf} \\ &= tf \cdot \frac{k+1}{k(1 - b + b \cdot DL/DL_{avg}) + tf} \end{aligned}$$

The final BM25 score is then $tf^* \cdot idf$

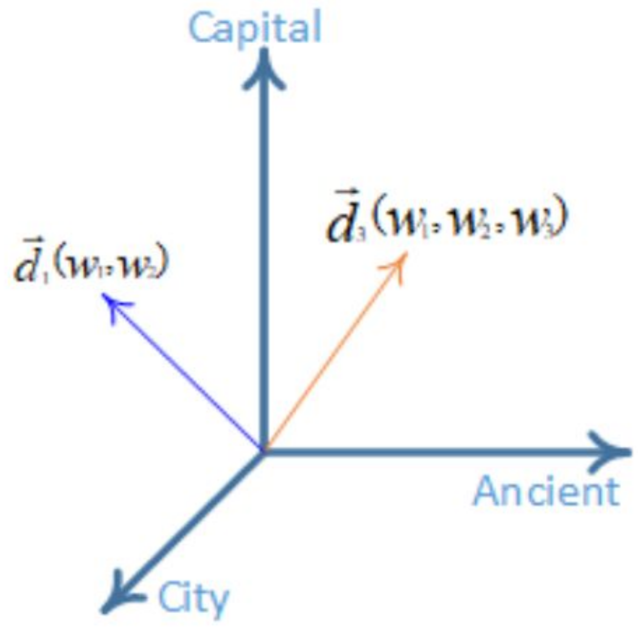
Vector Space Model

Each document as a vector with one component vector corresponding to each term in the dictionary

d1 = {0 2.32 3.5 0 0 0 0 0}

Terms that do not occur in a document, this weight is zero

Documents are represented as n-dimensional vectors



Vector Space Model

Documents and queries considered to be bags of words can represent as vectors of TF-IDF weights of vocabulary terms

Ranking function: Similarity/distance between the two TF-IDF vectors

What distance metric to use?

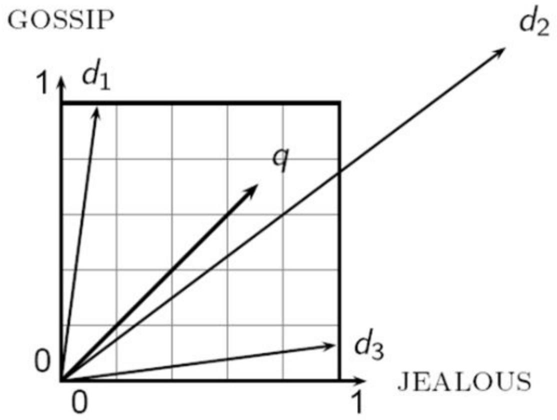
- Euclidean distance
- Similarity metric

Euclidean Distance

Measures the distance between the ends (points) of the two vectors

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

- The Euclidean distance between q and d_2 is large
  - But the distribution of terms in the query q and the distribution of terms in the query d_2 are very similar
- Euclidean distance is a bad idea
  - Because Euclidean distance is large for vectors of different lengths



Use angle instead of distance

The angle between the query q and the documents d2 is small, corresponding to high similarity

Angle between the vectors better captures the actual similarity

Rank documents according to angle with query

Cosine Similarity

The smaller the angle between two vectors is, the larger is the value of the cosine of that angle

$$\cos(\mathbf{q},\mathbf{d}) = \frac{\mathbf{q}}{\|\mathbf{q}\|}\cdot\frac{\mathbf{d}}{\|\mathbf{d}\|} = \frac{\mathbf{q}\cdot\mathbf{d}}{\|\mathbf{q}\|\|\mathbf{d}\|}$$

$$= \frac{\sum_{k=1}^{|V|} w_{kq}w_{ki}}{\sqrt{\sum_{k=1}^{|V|} w_{kq}^2}\sqrt{\sum_{k=1}^{|V|} w_{ki}^2}}$$

q_i is the weight of term i in the query

d_i is the weight of term i in the document

Example

Assume two documents:

D1: "the cat sat on the mat"

D2: "the dog sat on the log"

Vocabulary: [cat,dog,sat,mat,log]
Compute term frequencies per document (e.g., in D1: cat=1, sat=1, mat=1,others=0).

Example

Compute IDF for each term, for example with

$\mathrm{idf}(t)=\log(N/\mathrm{df}_t)$ where $N=2$ is number of documents and df_t is in how many documents term t appears.

cat appears only in D1: $\mathrm{df}_{cat}=1$ and $\mathrm{idf}_{cat}=\log(2/1)$

dog appears only in D2: $\mathrm{idf}_{dog}=\log(2/1)$

sat appears in both documents: $\mathrm{df}_{sat}=2$ and $\mathrm{idf}_{sat}=\log(2/2)=0$

TF-IDF weight for a term in a document is $\mathrm{tf}(t,d)\times\mathrm{idf}(t)$

So the TF-IDF vectors (in that term order) become:

- $v1=[1\cdot\log2,\ 0,\ 1\cdot0,\ 1\cdot\log2,\ 0]$
- $v2=[0,\ 1\cdot\log2,\ 1\cdot0,\ 0,\ 1\cdot\log2]$

Example

Apply cosine similarity of v1 and v2

$$cos\Theta = \frac{v1.v2}{\parallel v1 \parallel . \parallel v2 \parallel}$$

$$v1.v2 = 0$$

$$\parallel v1 \parallel = \sqrt{(log2)^2 + (log2)^2} = \sqrt{2}log2$$

$$\parallel v2 \parallel = \sqrt{(log2)^2 + (log2)^2} = \sqrt{2}log2$$

Cosθ = 0. So with this tiny vocabulary and definition, the two documents are orthogonal (similarity 0) because they share only the term "sat", whose IDF is 0 and therefore contributes nothing in TF-IDF space

Another Example

Documents:

- D1: "machine learning is fun"
- D2: "deep learning is fun"
- D3: "football is fun"

Ignore stopwords only if not listed; here keep all tokens.

Vocabulary (sorted):

[machine, deep, learning, football, is, fun]

Another Example

Document frequencies df_t

- machine: 1
- deep: 1
- learning: 2
- football: 1
- is: 3
- fun: 3

Use $idf(t)=\log_{10}(N/df_t)$

- idf(machine)=log10(3/1)≈0.477
- idf(deep)=0.477
- idf(learning)=log10(3/2)≈0.176
- idf(football)=0.477
- idf(is)=log10(3/3)=0
- idf(fun)=0

So “is” and “fun” get weight 0 because they appear in every document

Each term appears once when present, so $\mathrm{tf}(t,d)=1$ if in document, else 0.

TF-IDF for each doc in the vocabulary order:

$\mathrm{D1}$: "machine learning is fun"

$$\mathrm{v1}=[1\cdot 0.477,\ 0,\ 1\cdot 0.176,\ 0,\ 1\cdot 0,\ 1\cdot 0]=[0.477,\ 0,\ 0.176,\ 0,\ 0,\ 0]$$

$\mathrm{D2}$: "deep learning is fun"

$$\mathrm{v2}=[0,\ 1\cdot 0.477,\ 1\cdot 0.176,\ 0,\ 0,\ 0]=[0,\ 0.477,\ 0.176,\ 0,\ 0,\ 0]$$

$\mathrm{D3}$: "football is fun"

$$\mathrm{v3}=[0,\ 0,\ 0,\ 1\cdot 0.477,\ 0,\ 0]=[0,\ 0,\ 0,\ 0.477,\ 0,\ 0]$$

Cosine Similarity: D1 vs D2

Calculate Cosθ=??

Also calculate D1 vs D3.