Cosine Similarity
BM25
Examples

Husni

Cosine Similarity

Cosine Similarity

- A metric used to measure how similar two vectors are, regardless of their magnitude.
- In text analysis, these vectors represent documents or sentences as numerical feature vectors (often using methods like TF-IDF, Word2Vec, etc.).
- Cosine similarity measures the cosine of the angle between two non-zero vectors (eg.: A and B) in a multi-dimensional space.

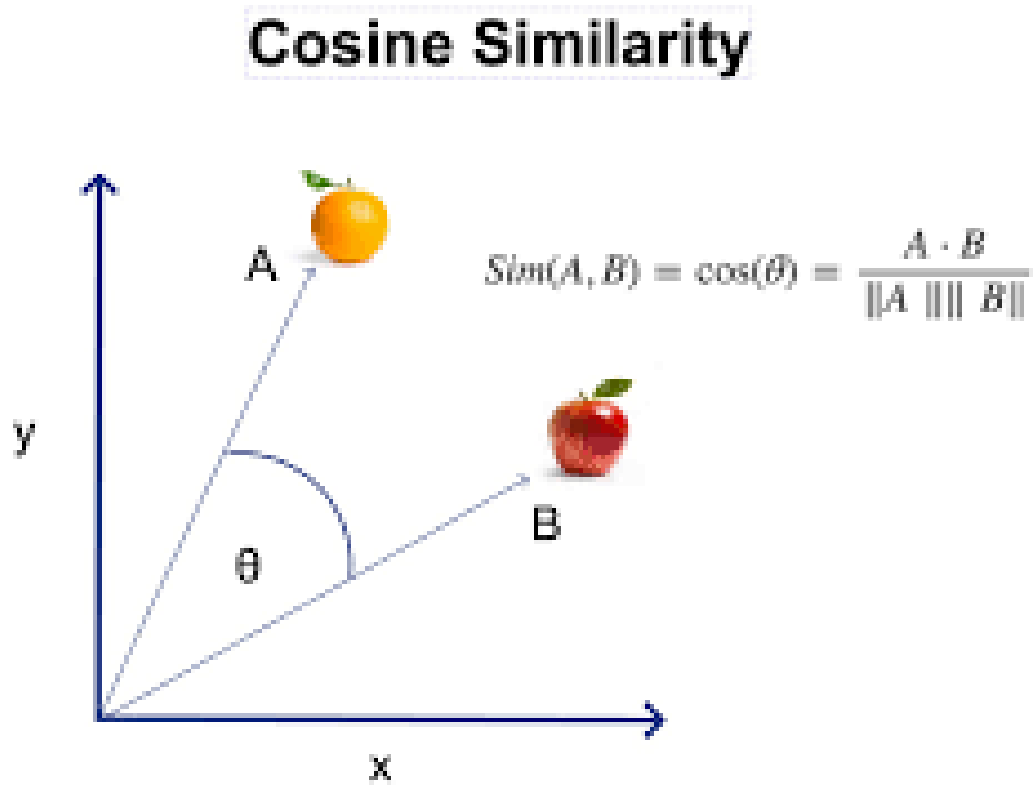
$$\text{Cosine Similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

- $A \cdot B$ is the dot product of the two vectors
- $\|A\|$ and $\|B\|$ are the magnitudes (or Euclidean norms) of the vectors

Key Properties

Range: [-1, 1], but in text analysis, it's usually between 0 and 1 since TF-IDF and other frequency-based vectors are non-negative.

- 1 = exactly the same direction (maximum similarity)
- 0 = orthogonal vectors (no similarity)
- -1 = opposite directions (inapplicable in most text tasks)



Cosine Formula and Text Similarity

Suppose we have:

- Doc1: “I like machine learning”
- Doc2: “I enjoy learning about machines”

After tokenization and TF-IDF vectorization (simplified), you get:

- Vector1 = [0.5, 0.5, 0.7, 0.0] → corresponds to “I”, “like”, “machine”, “learning”
- Vector2 = [0.4, 0.0, 0.6, 0.6] → corresponds to “I”, “enjoy”, “machine”, “learning”

Then compute cosine similarity as:

$$\cos(\theta) = \frac{(0.5 * 0.4 + 0.5 * 0 + 0.7 * 0.6 + 0 * 0.6)}{\sqrt{0.5^2 + 0.5^2 + 0.7^2 + 0^2} \times \sqrt{0.4^2 + 0^2 + 0.6^2 + 0.6^2}} \approx 0.75$$

This tells us that Doc1 and Doc2 are fairly similar.

Contoh

Step 1: Preprocessing

Dokumen Awal

Docs	Content
D1	"Machine learning is amazing in applications!"
D2	"Deep learning and machine learning improve AI applications."
D3	"Applications of AI are growing in healthcare."

Step 1.1: Convert to Lowercase

Why? Makes comparison case-insensitive.

Docs	Processed Text
D1	"machine learning is amazing in applications!"
D2	"deep learning and machine learning improve ai applications."
D3	"applications of ai are growing in healthcare."

Step 1: Preprocessing

Step 1.2: Remove Punctuation

Why? Punctuation does not add meaning.

Docs	Processed Text
D1	"machine learning is amazing in applications"
D2	"deep learning and machine learning improve ai applications"
D3	"applications of ai are growing in healthcare"

Step 1.3: Tokenization

Why? Splits text into individual words.

Docs	Tokenized Text
D1	['machine', 'learning', 'is', 'amazing', 'in', 'applications']
D2	['deep', 'learning', 'and', 'machine', 'learning', 'improve', 'ai', 'applications']
D3	['applications', 'of', 'ai', 'are', 'growing', 'in', 'healthcare']

Step 1: Preprocessing

Step 1.4: Stopword Removal

Why? Removes common words (e.g., "is", "in", "of", "and") that don’t carry much meaning.

Docs	After Stopword Removal
D1	['machine', 'learning', 'amazing', 'applications']
D2	['deep', 'learning', 'machine', 'learning', 'improve', 'ai', 'applications']
D3	['applications', 'ai', 'growing', 'healthcare']

Step 1.5: Stemming

Why? Reduces words to their root form (e.g., "running" → "run").

Docs	After Stemming
D1	['machine', 'learn', 'amaz', 'applic']
D2	['deep', 'learn', 'machine', 'learn', 'improv', 'ai', 'applic']
D3	['applic', 'ai', 'grow', 'healthcar']

Step 2: Compute TF-IDF (After Preprocessing)

Step 2.1: Compute Term Frequency (TF)

$$TF = \frac{\text{Number of times term appears in document}}{\text{Total words in document}}$$

Term	D1 (4 words)	D2 (7 words)	D3 (4 words)
machine	1/4 = 0.25	1/7 = 0.14	0
learn	1/4 = 0.25	2/7 = 0.29	0
applic	1/4 = 0.25	1/7 = 0.14	1/4 = 0.25
ai	0	1/7 = 0.14	1/4 = 0.25
healthcar	0	0	1/4 = 0.25

Step 2.2: Compute Inverse Document Frequency (IDF)

$$IDF = \log\left(\frac{\text{Total Documents}}{\text{Documents Containing Term}}\right)$$

Term	Docs Containing Term (DF)	IDF = log(3/DF)
machine	2	log(3/2) = 0.176
learn	2	log(3/2) = 0.176
applic	3	log(3/3) = 0
ai	2	log(3/2) = 0.176
healthcar	1	log(3/1) = 0.477

Step 2: Compute TF-IDF

Step 2.3: Compute TF-IDF

Term	**TF-IDF D1**	**TF-IDF D2**	**TF-IDF D3**
machine	$0.25 \times 0.176 = 0.044$	$0.14 \times 0.176 = 0.025$	0
learn	$0.25 \times 0.176 = 0.044$	$0.29 \times 0.176 = 0.051$	0
applic	$0.25 \times 0 = 0$	$0.14 \times 0 = 0$	$0.25 \times 0 = 0$
ai	$0 \times 0.176 = 0$	$0.14 \times 0.176 = 0.025$	$0.25 \times 0.176 = 0.044$
healthcar	$0 \times 0.477 = 0$	$0 \times 0.477 = 0$	$0.25 \times 0.477 = 0.119$

Step 3: Compute Cosine Similarity

Query: "machine learning applications"
Assumed Query TF-IDF Vector:
Query: [0.3, 0.4, 0.2, 0, 0]

$$\cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$$

Term	Query (Q)	D1	D2	D3
machine	0.3	0.044	0.025	0
learn	0.4	0.044	0.051	0
applic	0.2	0	0	0
ai	0	0	0.025	0.044
healthcar	0	0	0	0.119

Compute Dot Products

Q·D1 = (0.3×0.044)+(0.4×0.044) = 0.026
Q·D2 = (0.3×0.025)+(0.4×0.051) = 0.028
Q·D3 = 0

Compute Cosine Similarity

- Cos(Q, D1) = 0.91
- Cos(Q, D2) = 0.93
- Cos(Q, D3) = 0.50

Final Ranking

1. D2: Most relevant
2. D1: Second most relevant
3. D3: Least relevant

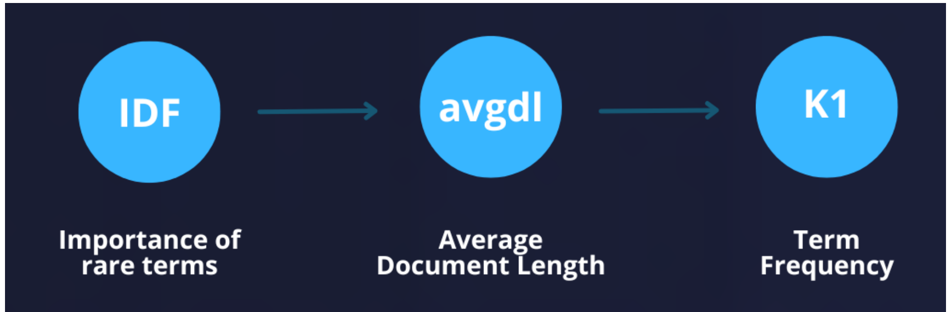
BM25

BM25: Best Matching 25

A ranking function used in search engines to estimate the relevance of documents to a given query. It is part of a broader family called probabilistic retrieval models and improves upon simpler models like TF-IDF.

At its core, BM25 is an advanced version of TF-IDF that:

- Rewards documents that have more occurrences of the query terms (like TF).
- Penalizes very long documents (because they tend to match more words just because they are long).
- Uses term saturation — meaning adding the same word over and over gives diminishing returns.



BM25 formula

Given:

- $f(q_i, D)$ = frequency of query term q_i in document D
- $|D|$ = length of document D (number of words)
- *avgdl* = average document length in the corpus
- k_1 = term frequency scaling parameter (usually around 1.2–2.0)
- b = length normalization parameter (usually around 0.75)

$$\mathrm{BM25}(D,Q) = \sum_{q_i \in Q} \mathrm{IDF}(q_i) \times \frac{f(q_i, D) \times (k_1 + 1)}{f(q_i, D) + k_1 \times \left(1 - b + b \times \frac{|D|}{\mathrm{avgdl}}\right)}$$

Where:

- N = total number of documents
- $n(q_i)$ = number of documents containing q_i

$$\mathrm{IDF}(q_i) = \log\left(\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} + 1\right)$$

Key Points & Intuition

- TF scaling: The impact of a term appearing 10 times vs. 20 times flattens out.
- Document length normalization: Longer documents are not unfairly advantaged.
- IDF factor: Terms that appear in fewer documents are given more importance (like TF-IDF).

BM25 tries to balance between:

- How often the query words appear in the document (TF)
- How rare the query words are across all documents (IDF)
- How long the document is (penalizes too-long documents)

Comparison: Cosine Similarity vs BM25

Feature	**Cosine Similarity**	**BM25**
Based on	Vector angle	Probabilistic model
Normalization	Vector length	Document length (adjustable)
Saturation of term freq	No	Yes
Common in	Embedding models, TF-IDF search	Traditional search engines

Contoh

Step 1: Preprocessing (Same as Before)

We'll use the same **preprocessed documents** as before:

Docs	Processed Text
D1	['machine', 'learn', 'amaz', 'applic']
D2	['deep', 'learn', 'machine', 'learn', 'improv', 'ai', 'applic']
D3	['applic', 'ai', 'grow', 'healthcar']

Step 2: Compute BM25 Score

Step 2.1: Define BM25 Formula

$$BM25(D,Q) = \sum_{t \in Q} IDF(t) \cdot \frac{TF(t,D) \cdot (k_1 + 1)}{TF(t,D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{\text{avgdl}})}$$

- $TF(t,D)$ = term frequency in document D
- $|D|$ = document length
- avgdl = average document length across all documents
- $k_1 = 1.5$ (controls saturation of term frequency)
- $b = 0.75$ (controls document length normalization)
- $IDF(t) = \log\left(\frac{N - DF + 0.5}{DF + 0.5} + 1\right)$

Step 2: Compute BM25 Score

Step 2.2: Compute Document Length & Average Document Length

Document Length:
D1 → 4
D2 → 7
D3 → 4

$$\text{avgdl} = \frac{4+7+4}{3} = \frac{15}{3} = 5$$

Step 2.3: Compute IDF for Each Term

$$IDF(t) = \log\left(\frac{N - DF + 0.5}{DF + 0.5} + 1\right)$$

Term	DF (Docs Containing Term)	IDFIDFIDF
machine	2	log((3−2+0.5)/(2+0.5)+1) = log(1.5) = 0.176
learn	2	log(1.5)=0.176
applic	3	log(1)=0
ai	2	log(1.5)=0.176
healthcar	1	log(2.5)=0.477

Step 2: Compute BM25 Score

BM25 for D1

Step 2.4: Compute BM25 Score for Each Document

We'll assume **k1=1.5k_1 = 1.5k1=1.5 and b=0.75b = 0.75b=0.75**.
Query: "machine learning applications"
Query terms: **[machine, learn, applic]**

$$BM25(D1,Q) = IDF(machine) \times \frac{TF + 1.5}{TF + 1.5(1 - 0.75 + 0.75 \times (4/5))}$$

"machine"

$$BM25(D1) = 0.176 \times \frac{1.0 \times (1.5 + 1)}{1.0 + 1.5 \times (1 - 0.75 + 0.75 \times (4/5))}$$

$$= 0.176 \times \frac{2.5}{1 + 1.5 \times 0.85} = 0.176 \times \frac{2.5}{2.275} = 0.176 \times 1.1 = 0.194$$

"learn"

$$BM25(D1)+ = 0.176 \times \frac{1.0 \times (1.5 + 1)}{1.0 + 1.5 \times (1 - 0.75 + 0.75 \times (4/5))}$$

$$= 0.176 \times 1.1 = 0.194$$

"applic"

$$BM25(D1)+ = 0 \times (\text{anything}) = 0$$

$$BM25(D1,Q) = 0.194 + 0.194 + 0 = 0.388$$

Step 2: Compute BM25 Score

BM25 for D2

“machine”

$$BM25(D2) = 0.176 \times \frac{1.0 \times (1.5+1)}{1.0 + 1.5 \times (1 - 0.75 + 0.75 \times (7/5))}$$

$$= 0.176 \times \frac{2.5}{1 + 1.5 \times 1.05} = 0.176 \times \frac{2.5}{2.575} = 0.176 \times 0.97 = 0.171$$

“learn”

$$BM25(D2) + = 0.176 \times \frac{2.0 \times (1.5+1)}{2.0 + 1.5 \times (1 - 0.75 + 0.75 \times (7/5))}$$

$$= 0.176 \times \frac{3.5}{3.575} = 0.176 \times 0.98 = 0.172$$

“applic”

$$BM25(D2) + = 0 \times (\text{anything}) = 0$$

$$BM25(D2, Q) = 0.171 + 0.172 + 0 = 0.343$$

Step 2: Compute BM25 Score

BM25 for D3

"machine"

$$BM25(D3) = 0.176 \times 0 = 0$$

"learn"

$$BM25(D3) + = 0.176 \times 0 = 0$$

"applic"

$$BM25(D3) + = 0 \times (\text{anything}) = 0$$

$$BM25(D3, Q) = 0 + 0 + 0 = 0$$

Step 3: Final Ranking

Docs	BM25 Score
D1	**0.388**
D2	**0.343**
D3	0

Final Order

1. **D1 (Most Relevant)**
2. **D2**
3. **D3 (Not Relevant)**

Conclusion

- BM25 differs from TF-IDF!
  - TF-IDF gave D2 as most relevant
  - BM25 gives D1 as most relevant

- BM25 accounts for document length
  - D1 had shorter length → higher BM25 score
  - D2 had longer length → BM25 penalized it slightly

- BM25 is better for ranking search results
  - It models real-world search relevance better than TF-IDF.

Python Code

The code will:

1. Preprocess the text (Tokenization, Stopword Removal, Stemming)
2. Compute TF-IDF & Cosine Similarity
3. Compute BM25
4. Rank documents using both methods

```python
import numpy as np
import pandas as pd
import math
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from collections import Counter
from nltk.tokenize import word_tokenize
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory

# Sample Documents
documents = [
    "Machine learning has amazing applications",
    "Deep learning and machine learning improve AI applications",
    "AI applications are growing in healthcare"
]

query = "machine learning applications"
```

```python
# Preprocessing (Tokenization, Stopword Removal, Stemming)
factory = StemmerFactory()
stemmer = factory.create_stemmer()

def preprocess(text):
    tokens = word_tokenize(text.lower())
    return ' '.join([stemmer.stem(word) for word in tokens])

documents = [preprocess(doc) for doc in documents]
query = preprocess(query)

# TF-IDF Vectorization
vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(documents)
query_vector = vectorizer.transform([query])

# Cosine Similarity Calculation
cosine_sim = cosine_similarity(query_vector, tfidf_matrix)[0]
```

```python
# BM25 Calculation
k1 = 1.5
b = 0.75
N = len(documents)
avgdl = sum(len(doc.split()) for doc in documents) / N
df = Counter()
for doc in documents:
    for word in set(doc.split()):
        df[word] += 1

idf = {word: math.log((N - df[word] + 0.5) / (df[word] + 0.5) + 1) for word in df}
```

```python
def bm25_score(doc, query):
    words = doc.split()
    dl = len(words)
    score = 0
    for term in query.split():
        tf = words.count(term)
        if tf == 0:
            continue
        numerator = tf * (k1 + 1)
        denominator = tf + k1 * (1 - b + b * dl / avgdl)
        score += idf.get(term, 0) * (numerator / denominator)
    return score

bm25_scores = [bm25_score(doc, query) for doc in documents]
```

```python
# Ranking Results
cosine_ranked = sorted(zip(range(len(documents)), cosine_sim), key=lambda x: x[1], reverse=True)
bm25_ranked = sorted(zip(range(len(documents)), bm25_scores), key=lambda x: x[1], reverse=True)

# Print Results
print("Ranking by Cosine Similarity:")
for idx, score in cosine_ranked:
    print(f"Document {idx + 1}: Score = {score:.4f}")

print("\nRanking by BM25:")
for idx, score in bm25_ranked:
    print(f"Document {idx + 1}: Score = {score:.4f}")
```