Summarization

Summarization is the task of shortening a document to a smaller version while retaining the **most important information**.

Frequency based method

Doc:

"AI is transforming the world. AI technologies are used in many industries. The world is seeing rapid AI adoption. AI helps in automation and data analysis"

After removing stop words:

1. ***Sentence 1:*** `ai transforming world`
2. ***Sentence 2:*** `ai technologies used many industries`
3. ***Sentence 3:*** `world seeing rapid ai adoption`
4. ***Sentence 4:*** `ai helps automation data analysis`

📊 Step 2: Calculate Term Frequency (TF)

Word	Frequency
ai	4
world	2
transforming	1
technologies	1
used	1
many	1
industries	1
seeing	1
rapid	1
adoption	1
helps	1
automation	1
data	1
analysis	1

🧮 Step 3: Score Each Sentence

Score = sum of frequencies of words in the sentence

Sentence	Words	Score
S1: "AI transforming world"	ai(4), transforming(1), world(2)	**7**
S2: "AI technologies used many industries"	ai(4), technologies(1), used(1), many(1), industries(1)	**8**
S3: "World seeing rapid AI adoption"	world(2), seeing(1), rapid(1), ai(4), adoption(1)	**9**
S4: "AI helps automation data analysis"	ai(4), helps(1), automation(1), data(1), analysis(1)	**8**

🧠 Step 4: Select Top Sentences

Pick top 2 highest-scoring sentences for the summary.

- **S3:** "World seeing rapid AI adoption" (Score = 9)
- **S2 or S4:** Tie at Score = 8. Let's choose **S2**.

✅ Final Extractive Summary:

> *"World seeing rapid AI adoption. AI technologies used many industries."*

Types of Summarization

Type	Description	Example Summary
Extractive	Selects key **sentences or phrases** from the original text	"The study finds X and concludes Y." (copied from original)
Abstractive	**Rewrites** the content using new words and phrasing like a human would	"The study shows X and ends with Y." (not copied exactly)

Extractive Summarization with TextRank

TextRank is an extractive summarization algorithm based on Google's PageRank (used for ranking web pages).

TextRank is a graph-based extractive summarization algorithm. It works by identifying the most important sentences in a document based on sentence similarity rather than understanding semantics.

Steps:

1. Sentence as Nodes: Each sentence in the document is treated as a node in a graph.

2. Edge Weights = Similarity: Edges are weighted by similarity (often cosine similarity between TF-IDF vectors or word embeddings).

3. Ranking via Iterative Scoring: A variant of the PageRank algorithm is applied to calculate the importance (score) of each sentence based on how well it connects to other sentences.

4. Select Top Sentences: The top-ranked sentences are extracted and presented in the same order as in the original document to form the summary.

Note: The algorithm assumes that a sentence is important if it is similar to many other important sentences, much like how PageRank ranks web pages based on links from other important pages.

Example

S1: Natural language processing (NLP) is a field of artificial intelligence.

S2: NLP helps computers understand human language.

S3: It involves tasks like machine translation and sentiment analysis.

S4: NLP is widely used in voice assistants.

S5: Understanding context and ambiguity is a challenge in NLP.

Tokenize and lowercase each sentence:

Sentence ID	Tokens
S1	[natural, language, processing, nlp, field, artificial, intelligence]
S2	[nlp, helps, computers, understand, human, language]
S3	[it, involves, tasks, like, machine, translation, sentiment, analysis]
S4	[nlp, widely, used, voice, assistants]
S5	[understanding, context, ambiguity, challenge, nlp]

Vocab:

[1] natural

[2] language

[3] processing

[4] nlp

[5] field

[6] artificial

[7] intelligence

[8] helps

[9] computers

[10] understand

[11] human

[12] it

[13] involves

[14] tasks

[15] like

[16] machine

[17] translation

[18] sentiment

[19] analysis

[20] widely

[21] used

[22] voice

[23] assistants

[24] understanding

[25] context

[26] ambiguity

[27] challenge

TF-IDF Matrix

compute the TF-IDF vector for each sentence (like we did earlier) and then calculate the cosine similarity between each pair of sentences.

Term	Frequency in S1	TF = f / total terms
natural	1	1/7 ≈ 0.143
language	1	1/7 ≈ 0.143
processing	1	1/7 ≈ 0.143
nlp	1	1/7 ≈ 0.143
field	1	1/7 ≈ 0.143
artificial	1	1/7 ≈ 0.143
intelligence	1	1/7 ≈ 0.143

$$\mathrm{IDF}(t) = \log_{10}\left(\frac{N}{1 + \mathrm{df}(t)}\right)$$

Term	df(t)	IDF(t) = log(5 / (1+df))
natural	1	log(5 / 2) ≈ 0.3979
language	2	log(5 / 3) ≈ 0.2218
processing	1	log(5 / 2) ≈ 0.3979
nlp	4	log(5 / 5) ≈ 0.0000
field	1	log(5 / 2) ≈ 0.3979
artificial	1	log(5 / 2) ≈ 0.3979
intelligence	1	log(5 / 2) ≈ 0.3979

$$\text{TF-IDF}(t) = \text{TF}(t) \times \text{IDF}(t)$$

Term	TF	IDF	TF-IDF
natural	0.143	0.3979	≈ 0.0569
language	0.143	0.2218	≈ 0.0317
processing	0.143	0.3979	≈ 0.0569
nlp	0.143	0.0000	= 0
field	0.143	0.3979	≈ 0.0569
artificial	0.143	0.3979	≈ 0.0569
intelligence	0.143	0.3979	≈ 0.0569

🔷 Cosine Similarity Formula

Given two vectors A and B, the **cosine similarity** is:

$$\text{cosine\_similarity}(A, B) = \frac{A \cdot B}{\|A\| \times \|B\|}$$

Where:

- $A \cdot B$ = dot product of the vectors
- $\|A\|$ = magnitude (length) of vector A
- $\|B\|$ = magnitude (length) of vector B

Cosine similarity measures angle-based similarity between vectors — close to 1 means they are similar.

It’s widely used in text summarization to connect similar sentences and find the most central and informative ones.

Assume the cosine similarity matrix:

	S1	S2	S3	S4	S5
S1	0.0	0.4	0.1	0.3	0.2
S2	0.4	0.0	0.2	0.3	0.3
S3	0.1	0.2	0.0	0.2	0.1
S4	0.3	0.3	0.2	0.0	0.3
S5	0.2	0.3	0.1	0.3	0.0

TextRank Algorithm and Numerical (Imp)

1. **Graph Representation**:
   - Sentences are nodes in a graph $G = (V, E)$, where V is the set of sentences ($|V| = n$) and E is the set of edges.
   - Edges are weighted by a **similarity function** w_{ij}, representing how similar sentences S_i and S_j are.
2. **Similarity Function**:
   - A common similarity measure is the **overlap of words** between sentences, adjusted for length to avoid bias toward longer sentences:

$$w_{ij} = \frac{|\text{words in } S_i \cap \text{words in } S_j|}{\log(|S_i|) + \log(|S_j|)}$$

     - Numerator: Count of common words (excluding stopwords for better focus on content).
     - Denominator: Normalizes by sentence lengths to prevent favoring long sentences.
   - Alternatively, cosine similarity over TF-IDF vectors or embeddings (e.g., BERT) can be used for semantic similarity.
3. **TextRank Score Update**:
   - Each sentence S_i has a score $WS(S_i)$, initialized to 1.
   - Scores are updated iteratively using a damped PageRank formula:

$$WS(S_i) = (1 - d) + d \cdot \sum_{S_j \in \text{In}(S_i)} \frac{w_{ji}}{\sum_{S_k \in \text{Out}(S_j)} w_{jk}} WS(S_j)$$

     - d: Damping factor (typically 0.85), representing the probability of following an edge (vs. jumping randomly).
     - $\text{In}(S_i)$: Sentences with edges pointing to S_i.
     - $\text{Out}(S_j)$: Sentences S_j points to.
     - w_{ji}: Weight of edge from S_j to S_i.
     - The term $\frac{w_{ji}}{\sum_{S_k \in \text{Out}(S_j)} w_{jk}}$ normalizes the contribution of S_j based on its outgoing edge weights.

4. **Convergence**:
   - Iterate until scores stabilize (e.g., change < 0.0001) or for a fixed number of iterations.
   - Final scores rank sentences; top k sentences form the summary.

Intuition (optional)

The TextRank formula is a **weighted random walk** on the sentence graph:

- **Damping Factor ($d = 0.85$)**: Models a reader who follows similar sentences 85% of the time and jumps randomly 15% of the time, ensuring all sentences have some score.
- **Edge Weights (w_{ji})**: Stronger similarity (higher w_{ji}) means a sentence contributes more to another's score.
- **Normalization ($\sum_{S_k \in \text{Out}(S_j)} w_{jk}$)**: Ensures a sentence's influence is distributed proportionally to its outgoing connections, preventing bias from highly connected nodes.
- **Iterative Updates**: Scores propagate through the graph, amplifying sentences that are central (connected to other high-scoring sentences), akin to an eigenvector computation.

This process ensures that sentences with high overlap or semantic similarity to others are ranked higher, capturing the document's core themes.

1. **Random Jump Component ($1 - d$)**:
   - This is the **15% weightage** when $d = 0.85$, since $1 - 0.85 = 0.15$.
   - It represents the probability of "randomly jumping" to any sentence in the document, regardless of its connections in the graph. This ensures every sentence has a non-zero score, even if it has few or no edges (similar to a reader randomly picking a sentence to start reading).
   - Mathematically, the $(1 - d)$ term contributes a baseline score to $WS(S_i)$, independent of the graph structure.
2. **Graph-Based Component (d)**:
   - This is the **85% weightage** when $d = 0.85$.
   - It represents the probability of "following" the graph's edges, where the score of sentence S_i is influenced by the scores of its neighboring sentences (S_j), weighted by their similarity (w_{ji}).
   - The term $\sum_{S_j \in \text{neighbors}(S_i)} \frac{w_{ji}}{\sum_{S_k \in \text{neighbors}(S_j)} w_{jk}} WS(S_j)$ calculates the contribution from connected sentences, normalized by the outgoing edge weights of each neighbor.

If a sentence is similar to many, it summarizes the main content — so it gets a high score.

Numerical

📄 Input Sentences (with IDs)

ID	Sentence
S1	"The new AI model improves translation accuracy."
S2	"It handles idioms better than older models."
S3	"The model was tested on English and Spanish."
S4	"Results show a 20% accuracy increase."

📉 Step 1: Build the Sentence Similarity Graph

Using the **given cosine similarities**, we build an **undirected weighted graph**:

Edge	Similarity
S1 – S2	0.5
S1 – S3	0.3
S1 – S4	0.4
S2 – S3	0.2
S2 – S4	0.3
S3 – S4	0.25

🔢 Step 2: Initialize TextRank Scores

Assume each sentence gets an initial score of **1.0**.

$$S(S1) = S(S2) = S(S3) = S(S4) = 1.0$$

🧮 Step 3: TextRank Formula

TextRank update formula for sentence *Si*:

$$S(S_i) = (1-d) + d \cdot \sum_{S_j \in N(S_i)} \frac{sim(S_i, S_j)}{\sum_{S_k \in N(S_j)} sim(S_j, S_k)} \cdot S(S_j)$$

Where:

- $d = 0.85$ (damping factor)
- $sim(S_i, S_j)$: similarity between sentences
- $N(S_i)$: neighbors of S_i

🔁 Step 4: Calculate Sum of Outgoing Similarities for Each Sentence

Sentence	Neighbors	Sum of Similarities
S1	S2, S3, S4	0.5 + 0.3 + 0.4 = **1.2**
S2	S1, S3, S4	0.5 + 0.2 + 0.3 = **1.0**
S3	S1, S2, S4	0.3 + 0.2 + 0.25 = **0.75**
S4	S1, S2, S3	0.4 + 0.3 + 0.25 = **0.95**

🔁 Step 5: Iteration 1

We now compute **updated scores** using the formula.

⭐ S1:

$$S(S1) = 0.15 + 0.85 \cdot \left[\frac{0.5}{1.0} \cdot 1.0 + \frac{0.3}{0.75} \cdot 1.0 + \frac{0.4}{0.95} \cdot 1.0\right]$$

$$= 0.15 + 0.85 \cdot (0.5 + 0.4 + 0.421) = 0.15 + 0.85 \cdot 1.321 \approx 0.15 + 1.123 = \ast\ast 1.273 \ast\ast$$

⭐ S2:

$$S(S2) = 0.15 + 0.85 \cdot \left[\frac{0.5}{1.2} + \frac{0.2}{0.75} + \frac{0.3}{0.95}\right]$$

$$= 0.15 + 0.85 \cdot (0.417 + 0.267 + 0.316) = 0.15 + 0.85 \cdot 1.0 = \ast\ast 1.0 \ast\ast$$

⭐ S3:

$$S(S3) = 0.15 + 0.85 \cdot \left[\frac{0.3}{1.2} + \frac{0.2}{1.0} + \frac{0.25}{0.95}\right]$$

$$= 0.15 + 0.85 \cdot (0.25 + 0.2 + 0.263) = 0.15 + 0.85 \cdot 0.713 \approx 0.15 + 0.606 = \ast\ast 0.756 \ast\ast$$

⭐ S4:

$$S(S4) = 0.15 + 0.85 \cdot \left[\frac{0.4}{1.2} + \frac{0.3}{1.0} + \frac{0.25}{0.75}\right]$$

$$= 0.15 + 0.85 \cdot (0.333 + 0.3 + 0.333) = 0.15 + 0.85 \cdot 0.966 \approx 0.15 + 0.821 = \ast\ast 0.971 \ast\ast$$

✅ Iteration 1 Scores:

Sentence	Score
S1	1.273
S2	1.000
S3	0.756
S4	0.971

🔁 Step 6: Iteration 2 (Repeat with new scores)

Let's now plug these new scores back into the formula and compute **Iteration 2**.

⭐ **S1:**

$$S(S1) = 0.15 + 0.85 \cdot \left[\frac{0.5}{1.0} \cdot 1.0 + \frac{0.3}{0.75} \cdot 0.756 + \frac{0.4}{0.95} \cdot 0.971\right]$$

$$= 0.15 + 0.85 \cdot (0.5 + 0.302 + 0.409) \approx 0.15 + 0.85 \cdot 1.211 \approx 0.15 + 1.029 = **1.179**$$

⭐ **S2:**

$$S(S2) = 0.15 + 0.85 \cdot \left[\frac{0.5}{1.2} \cdot 1.273 + \frac{0.2}{0.75} \cdot 0.756 + \frac{0.3}{0.95} \cdot 0.971\right]$$

$$= 0.15 + 0.85 \cdot (0.531 + 0.202 + 0.306) = 0.15 + 0.85 \cdot 1.039 = **1.033**$$

⭐ **S3:**

$$S(S3) = 0.15 + 0.85 \cdot \left[\frac{0.3}{1.2} \cdot 1.273 + \frac{0.2}{1.0} \cdot 1.0 + \frac{0.25}{0.95} \cdot 0.971 \right]$$

$$= 0.15 + 0.85 \cdot (0.318 + 0.2 + 0.255) = 0.15 + 0.85 \cdot 0.773 = **0.805**$$

⭐ **S4:**

$$S(S4) = 0.15 + 0.85 \cdot \left[\frac{0.4}{1.2} \cdot 1.273 + \frac{0.3}{1.0} \cdot 1.0 + \frac{0.25}{0.75} \cdot 0.756 \right]$$

$$= 0.15 + 0.85 \cdot (0.424 + 0.3 + 0.252) = 0.15 + 0.85 \cdot 0.976 = **0.981**$$

✅ Final Scores After 2 Iterations:

Sentence	Score
S1	1.179
S2	1.033
S3	0.805
S4	0.981

📝 Final Step: Summary Selection

Pick **top 2 sentences** by score:

1. **S1**: "The new AI model improves translation accuracy."
2. **S2**: "It handles idioms better than older models."

📰 Final Extractive Summary:

> **"The new AI model improves translation accuracy. It handles idioms better than older models."**

TextRank Strengths for Short Documents:

1. No need for large corpora:
   - TF-IDF needs a corpus to calculate IDF (Inverse Document Frequency) accurately.
   - TextRank just needs sentence similarity, so it's ideal for single or short documents.
2. Captures sentence centrality:
   - In short texts, a few key sentences carry most of the meaning.
   - TextRank finds these by seeing which sentences are most connected via similarity.
3. More semantic understanding:
   - Even if a key idea is phrased differently, it still connects to others.
   - TextRank's graph handles this better than frequency counts.
4. Reduces redundancy:
   - Frequency methods might pick two similar sentences just because they contain frequent words.
   - TextRank scores based on connectivity and novelty, leading to better variety in summaries.