IR U 3 Probabilistic Retrieval Model

Probabilistic Retrieval is an **information retrieval model** that ranks documents based on the **probability that a document is relevant to a given user query**. The fundamental idea is that documents should be presented to the user in **decreasing order of their probability of relevance**.

The model estimates the probability P(R|D) where **R** denotes relevance and **D** denotes a document. A document with a higher probability of relevance is ranked higher.

Probability Concepts Used

- The **sample space** is the entire document collection.
- Events include:
  - A document is relevant (R)
  - A document contains a term t
- **Conditional probability** is used to compute how likely a document is relevant given that it contains certain query terms.

Use of Bayes' Theorem

Using Bayes' theorem:

P(R|D)=P(D|R)·P(R)/P(D) Since P(D) is the same for all documents, ranking depends mainly on:

- P(D|R): likelihood of the document given relevance
- P(R): prior probability of relevance

Term Independence Assumption

The model assumes **term independence**, meaning each query term contributes independently to relevance. Though not fully realistic, this assumption simplifies computation and enables efficient ranking.

Importance and Applications

- Forms the basis of the **Binary Independence Model (BIM)**
- Extended into practical ranking models such as **BM25**
- Supports **relevance feedback** to improve search results over time

Small Example Query: "machine learning"

- Term *machine* appears more frequently in relevant documents than non-relevant ones.
- Term *learning* also has high occurrence in relevant documents.

A document containing both terms will have a **higher probability of relevance** compared to documents containing only one or none of the terms. Hence, it will be ranked higher.

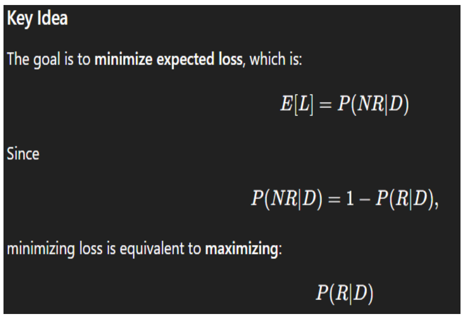
The Probability Ranking Principle (PRP): The 1/0 Loss Case

The **Probability Ranking Principle (PRP)** is a fundamental theory in Information Retrieval which states that:**"For optimal retrieval performance, documents should be ranked in decreasing order of their probability of relevance to the user's query."**

In the **1/0 loss case**, the retrieval process assumes only two outcomes:

- **Loss = 0** → if a relevant document is retrieved
- **Loss = 1** → if a non-relevant document is retrieved

This means the system is punished only when it retrieves non-relevant documents, and there is no extra penalty for missing relevant ones.



Key Idea

The goal is to **minimize expected loss**, which is:

$$E[L] = P(NR|D)$$

Since

$$P(NR|D) = 1 - P(R|D),$$

minimizing loss is equivalent to **maximizing**:

$$P(R|D)$$

Thus, PRP dictates ranking documents by *decreasing* probability of relevance.

Why PRP Is Optimal

- It ensures that users see the most relevant documents first.
- It provides a mathematically sound ranking rule under uncertainty.
- It assumes perfect probability estimates, making the ranking theoretically optimal.

Assumptions

- Relevance is binary
- Loss for retrieving a relevant document = 0
- Loss for retrieving a non-relevant document = 1
- User prefers to avoid non-relevant documents

The Probability Ranking Principle (PRP) with Retrieval Costs

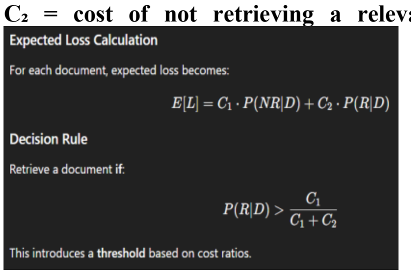
The basic PRP assumes equal loss for all retrieval errors. However, in real systems, **different types of errors have different costs**.

Realistic Scenario

- Missing a relevant document (false negative) may be very expensive
  → Example: medical diagnosis, legal information
- Retrieving a non-relevant document (false positive) may be less costly

Thus, losses are defined as:

- **C_1 = cost of retrieving a non-relevant document**
- **C_2 = cost of not retrieving a relevant document**



Expected Loss Calculation

For each document, expected loss becomes:

$$E[L] = C_1 \cdot P(NR|D) + C_2 \cdot P(R|D)$$

Decision Rule

Retrieve a document if:

$$P(R|D) > \frac{C_1}{C_1 + C_2}$$

This introduces a **threshold** based on cost ratios.

Interpretation

- If **C₂ is large** (missing relevant doc is costly) → Threshold becomes **smaller** → System retrieves more documents
- If **C₁ is large** (retrieving non-relevant doc is costly) → Threshold becomes **larger** → System becomes selective

Benefits

- Adapts retrieval to practical requirements.
- Allows designing retrieval systems for specific domains (healthcare, legal, finance).
- Provides a more flexible, cost-sensitive decision mechanism.

The Binary Independence Model (BIM)

The **Binary Independence Model (BIM)** is one of the earliest and most influential probabilistic models used in Information Retrieval. It provides a mathematical framework for ranking documents based on the **probability of relevance** to a user's query. The model makes two major assumptions: **terms are binary** (present or absent) and **terms occur independently** of one another.

For any document *D* and query *Q*, BIM estimates: P(R=1|D,Q)

A document is ranked higher if it has a higher probability of being relevant. Instead of calculating full probabilities, the model computes a **retrieval score** based on query terms.

2. Binary Representation of Terms

BIM uses **binary term incidence**, meaning:

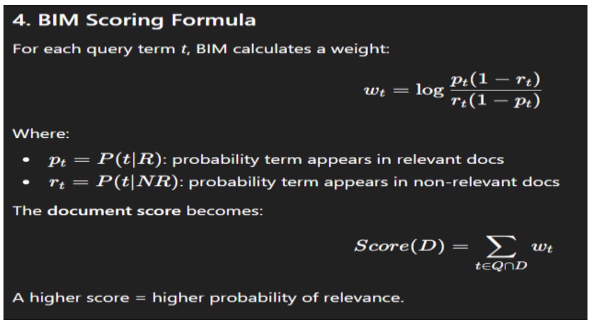
- Term present → 1
- Term absent → 0

Thus, term frequency is not needed.

Example:
If a term appears 1 time or 10 times → treated the same (presence only).

3. Independence Assumption

Each query term is assumed to contribute independently to the relevance of a document.
This assumption greatly simplifies probability calculations.



4. BIM Scoring Formula

For each query term t, BIM calculates a weight:

$$w_t = \log \frac{p_t(1-r_t)}{r_t(1-p_t)}$$

Where:

- $p_t = P(t|R)$: probability term appears in relevant docs
- $r_t = P(t|NR)$: probability term appears in non-relevant docs

The document score becomes:

$$Score(D) = \sum_{t \in Q \cap D} w_t$$

A higher score = higher probability of relevance.

6. Strengths of BIM

- **Mathematically sound** and probabilistic.
- **Simple to implement.**
- Directly supports **relevance feedback**, making weights more accurate.
- Provides the conceptual foundation for the widely used **BM25** ranking function.

7. Limitations of BIM

- Ignores **term frequency** (treats multiple occurrences as single).
- Assumes **independent terms**, which is unrealistic for correlated terms.
- Works best for short queries and simple collections.
- Cannot capture semantic relationships.

8. Importance and Applications

BIM is the theoretical root of modern retrieval models such as:

- **BM25**
- **BM25F**
- **Probabilistic relevance model variants**

Term Frequency (TF)

Term Frequency (TF) refers to the number of times a term appears in a document. It is an important factor in Information Retrieval because the more frequently a term appears in a document, the more likely it is that the document is relevant to that term or query.

Term Frequency is defined as:

TF(t,d)=number of times term t appears in document d

A higher TF value means the document discusses that term more prominently.

2. Importance in Retrieval Models

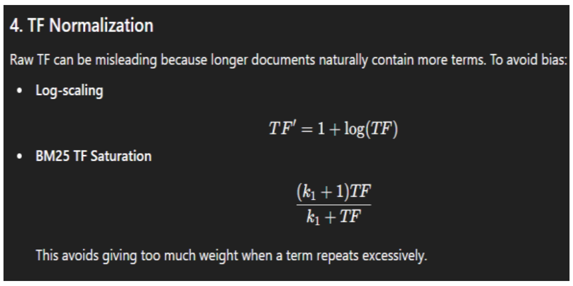
- TF helps differentiate between documents that merely **mention** a term and those that **focus heavily** on it.
- It improves ranking accuracy because a single occurrence does not carry the same importance as repeated occurrences.

3. TF in Probabilistic Models

Early probabilistic models like the **Binary Independence Model** ignored term frequency by using only term presence/absence. However, modern models (e.g., **BM25**) include TF because:

- Multiple occurrences strengthen evidence of relevance
- TF helps distinguish highly relevant documents from marginal ones

Thus TF became a core component in advanced probabilistic IR.



4. TF Normalization

Raw TF can be misleading because longer documents naturally contain more terms. To avoid bias:

- Log-scaling

$$TF' = 1 + \log(TF)$$

- BM25 TF Saturation

$$\frac{(k_1+1)TF}{k_1+TF}$$

This avoids giving too much weight when a term repeats excessively.

5. Role in Ranking

Higher TF means:

- Higher document relevance
- Higher ranking score
- Better matching of query intent

TF is combined with IDF (Inverse Document Frequency) to form scoring models like **TF-IDF** and **BM25**.

1. An Appraisal of Probabilistic Models – 6 Marks

Probabilistic models are one of the strongest theoretical approaches in Information Retrieval because they rank documents based on the **probability of relevance**. However, they also have limitations that lead to further improvements.

Strengths

1. **Strong Theoretical Foundation**
   Based on probability theory and the Probability Ranking Principle (PRP), making them mathematically sound.
2. **Effective Ranking**
   Models like **BM25** provide high retrieval performance and are widely used in modern search engines.
3. **Supports Relevance Feedback**
   Probabilities can be refined as the user marks documents as relevant or not, improving accuracy.
4. **Clear Interpretability**
   Each term contributes to relevance in a measurable way (e.g., RSJ weights).

Weaknesses

1. **Term Independence Assumption**
   Probabilistic models assume terms occur independently, which is unrealistic (e.g., “machine learning”, “data mining”).
2. **Binary Term Representation**
   Early models like BIM ignored term frequency, losing important relevance information.
3. **Difficulty Estimating Probabilities**
   Impossible to know true relevance probabilities; models rely on approximations.
4. **Limited Semantic Understanding**
   They cannot capture synonyms, context, or deep linguistic meaning.

Tree-Structured Dependencies Between Terms

Early probabilistic models assume **independence between terms**, but in real text, terms often have meaningful relationships. Tree-structured dependency models were introduced to capture these relationships more accurately.

Terms in a document are rarely independent. Examples:

- “machine” and “learning”
- “artificial” and “intelligence”

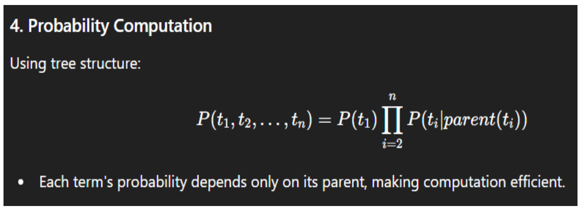
Ignoring such dependencies reduces retrieval accuracy.

2. Tree-Structured Model Concept

- Terms are represented as **nodes** in a tree.
- Edges represent **dependencies** (correlation or co-occurrence strength).
- The structure allows modeling how the presence of one term influences the probability of another.

3. How It Improves IR

- Captures relationships like synonymy, phrase structure, and co-occurrence patterns.
- Improves estimation of P(t|R) and overall relevance scoring.
- Reduces the unrealistic independence assumption of BIM.



4. Probability Computation

Using tree structure:

$$P(t_1, t_2, \ldots, t_n) = P(t_1) \prod_{i=2}^{n} P(t_i | parent(t_i))$$

- Each term's probability depends only on its parent, making computation efficient.

5. Applications

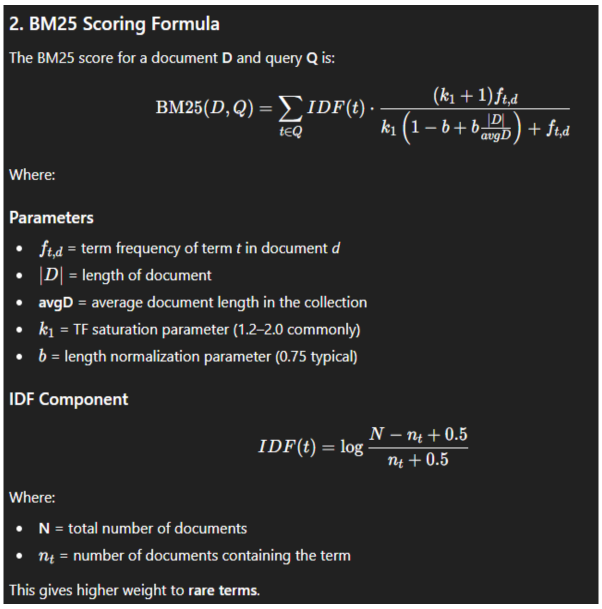
- Query expansion
- Better relevance modeling
- Basis for more advanced models like **Bayesian networks** and **Markov Random Fields**

Okapi BM25: A Non-Binary Model

Okapi Best Match 25 is one of the most widely used probabilistic ranking functions in Information Retrieval. It is considered the **gold standard baseline** in IR research. It evolved from the Binary Independence Model (BIM) but removed its major limitations by incorporating **term frequency**, **document length normalization**, and **saturation effects**. BM25 is considered the best classical IR model and is used in engines like **Lucene, ElasticSearch, Solr, and Terrier**. BM25 was created to address these practical needs.

Earlier models like BIM treated terms as **binary** (present/absent). However, real-world retrieval needs:

- Term Frequency (TF) contribution
- Importance of rare terms (IDF)
- Document Length Normalization
- Controlled TF saturation (avoiding overweighting repeated terms)



2. BM25 Scoring Formula

The BM25 score for a document **D** and query **Q** is:

$$\text{BM25}(D, Q) = \sum_{t \in Q} IDF(t) \cdot \frac{(k_1 + 1) f_{t,d}}{k_1 \left(1 - b + b \frac{|D|}{avgD}\right) + f_{t,d}}$$

Where:

Parameters

- $f_{t,d}$ = term frequency of term t in document d
- $|D|$ = length of document
- **avgD** = average document length in the collection
- k_1 = TF saturation parameter (1.2–2.0 commonly)
- b = length normalization parameter (0.75 typical)

IDF Component

$$IDF(t) = \log \frac{N - n_t + 0.5}{n_t + 0.5}$$

Where:

- **N** = total number of documents
- n_t = number of documents containing the term

This gives higher weight to **rare terms**.

3. Key Features of BM25

(a) Non-binary Term Frequency
Uses term frequency (TF); more occurrences increase relevance but with saturation.

(b) Document Length Normalization
Uses parameter **b** to reduce bias toward long documents.

(c) TF Saturation (k_1 parameter)
Initial term occurrences matter more; repeated terms add less weight.

(d) IDF Weighting
Rare terms get higher importance; common terms get lower weight.

4. Why BM25 Works So Well

- Captures real-world document behavior
- Smooth, balanced scoring mechanism
- Highly interpretable and tunable
- Works for both short keywords and long text queries
- Robust across domains and datasets

5. Advantages of BM25

1. Incorporates TF meaningfully
2. Length normalization prevents bias
3. No strict independence assumptions
4. Handles multiple query terms effectively
5. Fast to compute and easy to implement

6. Limitations

1. Assumes bag-of-words (no semantic understanding)
2. Parameters (k1, b) require tuning for best performance
3. Does not use contextual meaning or phrase-level matching.

Bayesian Network Approaches to IR

A **Bayesian Network (BN)** is a probabilistic graphical model that represents variables and their conditional dependencies using a **directed acyclic graph (DAG)**. In Information Retrieval (IR), Bayesian networks are used to model relationships between **terms, documents, and relevance** in a more flexible way than traditional probabilistic models like BIM.

Traditional probabilistic models assume:

- Independence between terms
- Binary term representation
- Limited structure in probability computation

These assumptions are unrealistic in real-world text, where terms are related (e.g., “artificial” + “intelligence”). Bayesian networks overcome these limitations by explicitly modeling dependencies.

2. Structure of a Bayesian Network in IR

The network consists of **nodes** and **directed edges**:

Nodes represent:

1. **Query terms (T_1, T_2, …)**
2. **Documents (D_1, D_2, …)**
3. **Relevance variable (R)**

Edges represent:

- Probabilistic dependencies between terms
- Influence of terms on documents
- Influence of document features on relevance

Example structure:
Terms → Document → Relevance

3. Probabilistic Representation

Each node has a **conditional probability table (CPT)** that quantifies relationships.

Example:

If a document contains both query terms (T_1=1, T_2=1):

$P(R)=P(D|T1,T2)\times P(R|D)$

$P(R)=0.8\times 0.9=0.72$

These probabilities combine to compute the final relevance score.

4. Inference for Ranking

Document ranking is based on computing:

$P(R=1|T1,T2,...,Tn)$

Using Bayesian inference:

$P(R|T)=\sum D\ \ P(R|D)\cdot P(D|T)$

Documents with higher probabilities are ranked higher.

5. Advantages of Bayesian Networks in IR

(a) Captures Term Dependencies
Models relationships like synonymy, co-occurrence, and semantic similarity between terms.

(b) Handles Uncertainty Naturally
Combines uncertain evidence using probability theory.

(c) Supports Query Expansion
Helps identify related or missing query terms through dependencies.

(d) Flexible Structure
Terms, documents, fields, and user behavior can all be represented as nodes.

(e) Multiple Evidence Integration
Can incorporate metadata, link structure, and user interactions.

6. Drawbacks

1. **High computational complexity** for large vocabularies.
2. **Requires learning the network structure**, which is difficult.
3. **Parameter estimation** for CPTs can be expensive.
4. Not as fast or scalable as BM25 or vector models.

7. Applications in IR

- Query expansion and refinement
- Modeling term-term correlations
- Filtering noisy results
- Expert and domain-specific retrieval systems
- Adaptive IR based on user feedback

Types of Bayesian Networks

A **Bayesian Network (BN)** is a probabilistic graphical model that represents a set of random variables and their conditional dependencies using a **Directed Acyclic Graph (DAG)**. Each node represents a variable, and directed edges encode conditional probability relationships. Different types of Bayesian networks are designed to balance model complexity, expressiveness, and computational efficiency.

1. General (Standard) Bayesian Network

This is the most flexible and expressive form of a Bayesian network.

Theory Points:

- Represented as a directed acyclic graph
- Each node has a Conditional Probability Table (CPT)
- Encodes the full joint probability distribution compactly
- Conditional independence reduces computational complexity
- Structure can represent complex dependency patterns

Applications: Diagnosis systems, risk analysis, expert systems

2. Naïve Bayes Network

Naïve Bayes is a highly simplified Bayesian network used mainly for classification.

Theory Points:

- Consists of one parent class node and multiple feature nodes
- Assumes conditional independence of features given the class
- Simplifies probability estimation and inference
- Works well even with small datasets
- Robust to irrelevant features

$$P(C \mid X) \propto P(C) \prod_i P(X_i \mid C)$$

Applications: Spam filtering, text classification, sentiment analysis

3. Tree-Augmented Naïve Bayes (TAN)

TAN relaxes the strong independence assumption of Naïve Bayes.

Theory Points:

- Each feature depends on the class variable and at most one other feature
- Allows limited feature dependencies
- Retains efficient inference
- Improves classification accuracy over Naïve Bayes
- Learned using maximum spanning tree algorithms

Applications: Medical diagnosis, bioinformatics

4. Dynamic Bayesian Network (DBN)

Dynamic Bayesian Networks extend Bayesian networks to model time-dependent systems.

Theory Points:

- Represent variables across multiple time slices
- Capture temporal dependencies using state transitions
- Assumes first-order Markov property
- Generalizes Hidden Markov Models (HMMs)
- Supports filtering, prediction, and smoothing

Applications: Speech recognition, time-series forecasting, tracking

5. Causal Bayesian Network

Causal Bayesian Networks explicitly model cause-and-effect relationships.

Theory Points:

- Directed edges represent causal influence
- Enable reasoning about interventions
- Support counterfactual analysis
- Distinguish correlation from causation
- Based on causal assumptions and domain knowledge

Applications: Policy analysis, healthcare decision support, economics

Relevance Feedback in Information Retrieval

Relevance Feedback (RF) is a technique in Information Retrieval where the user's judgments about which retrieved documents are relevant are used to improve the performance of the retrieval system.

1. Basic Concept of Relevance Feedback

After the user submits an initial query:

1. System retrieves a set of documents.
2. User marks some documents as **relevant** or **non-relevant**.
3. System uses this information to **recalculate term weights** or **modify the query model**.
4. New ranking is performed, usually yielding better results.

This process is iterative and continues until the user is satisfied.

2. Purpose of Relevance Feedback

- To refine the user's query
- To learn which terms are truly useful
- To enhance recall and precision
- To reduce the effect of ambiguous or incomplete queries

RF transforms a **short, vague, or imprecise query** into a more effective one.

3. Types of Relevance Feedback

(a) Explicit Relevance Feedback

User directly selects relevant/non-relevant documents.
Highly accurate but requires user effort.

(b) Implicit Relevance Feedback

System infers relevance automatically through:

- Click-through data
- Time spent on documents
- User interaction patterns
  Less accurate but easier to gather.

(c) Pseudo Relevance Feedback (Blind RF)

Assumes the top-k retrieved documents are relevant and updates query model accordingly.
Common in modern IR systems.

Relevance Feedback in Probabilistic Models

In probabilistic IR models, relevance feedback is used to **update the importance of terms** based on which documents the user marks as relevant or non-relevant.
If a term appears often in documents judged relevant, the system increases its weight because the term is a good indicator of relevance. If a term appears mostly in non-relevant documents, its weight is reduced. This updating process is called **RSJ weighting (Robertson–Sparck Jones weighting)**.
By adjusting term importance based on user feedback, the probabilistic model becomes more accurate and ranks future documents more effectively.

Relevance Feedback in Vector Space Models – Rocchio Algorithm

In vector space models, the **Rocchio Algorithm** modifies the original query using information from relevant and non-relevant documents.

- Terms from **relevant** documents are given more weight, pulling the query closer to useful content.
- Terms from **non-relevant** documents are reduced, pushing the query away from unwanted content.
- Part of the original query is kept to preserve user intent.

The result is an improved query that better represents what the user actually wants, giving more accurate and relevant search results.

6. Advantages of Relevance Feedback

1. **Improves Retrieval Quality**
   Usually increases both precision and recall.
2. **Reduces Query Ambiguity**
   Helps when the user's initial query is vague or short.
3. **Learns User Intent**
   Adapts the system to user's exact information need.
4. **Enables Query Expansion**
   Adds important terms from relevant documents.

7. Limitations of Relevance Feedback

1. Users may not want to mark documents manually.
2. Incorrect feedback can hurt performance.
3. Computationally expensive for large datasets.
4. Requires accurate UI and user cooperation.

Field Weights: BM25F (9 Marks)

BM25F is an extension of the popular BM25 ranking model, designed to handle **structured documents** that contain multiple fields such as *title*, *body*, *abstract*, *URL*, *headings*, etc. Since different fields contribute differently to relevance, BM25F introduces **field weights** to give more importance to certain fields over others.

1. Need for Field Weighting

Documents on the web or digital libraries are not uniform. A term appearing in the **title** is more important than the same term appearing in the **body**.
Examples:

- Title → very strong indicator of relevance
- Headings/subtitles → moderately important
- Body → normal importance
- Metadata/tags → high importance

BM25F solves this by applying **different weights to different fields**.

2. Core Idea of BM25F

Instead of using a single term frequency for the entire document, BM25F:

1. **Looks at each field separately**
2. **Applies field-specific weights**
3. **Combines them into a single score**

This allows the ranking model to respect the structure of the document.

3. Components Used in BM25F

BM25F extends BM25 by incorporating:

(a) Field Term Frequency (FTF)

Term frequency is computed separately for each field.

(b) Field Weights

Each field has a predefined importance value.
For example:

- Title weight > Body weight
- Abstract weight > Body weight

(c) Field Length Normalization

Each field may have different lengths.
Longer fields get lower weight; shorter fields get higher weight.

(d) Combined Score

The weighted and normalized frequencies from all fields are merged into a single score, similar to BM25's scoring mechanism.

4. Why Field Weights Improve Ranking

- **Title matches** are more meaningful than body matches.
- **Metadata**, **captions**, or **tags** can strongly indicate relevance.
- Helps avoid bias toward long fields (like large document bodies).
- Increases precision by using structural cues of the document.

BM25F is especially useful in web search engines and digital libraries.

5. Advantages of BM25F

1. **More accurate ranking** for structured documents
2. **Flexible system** where each field's importance can be tuned
3. **Better retrieval performance** than plain BM25 when documents have fields
4. Handles fields like title, URL, headings, keywords, etc.
5. Works extremely well for web search where structure matters

6. Limitations

1. Requires **manual tuning** of field weights (title vs body).
2. Slightly more computationally expensive than BM25.
3. Not effective if field weights are poorly chosen.
4. Cannot capture deeper semantic meaning—still bag-of-words.

Language Models for Information Retrieval: Generating Queries from Documents

Information Retrieval (IR) focuses on finding relevant documents for a given user query. Traditional IR systems treat queries as fixed inputs and rank documents based on similarity measures. **Language Models (LMs)** introduce a probabilistic framework where documents and queries are modeled as probability distributions over terms. One important approach in LM-based IR is **generating queries from documents**, where each document is treated as a language model that could generate the user's query.

Language Modeling Approach to IR

In the language modeling framework, each document $d$is assumed to define a probability distribution over words. The core idea is to rank documents by the probability that the document's language model generates the query q.

$$P(d \mid q) \propto P(q \mid d)P(d)$$

Since $P(d)$is usually assumed uniform, ranking depends mainly on $P(q \mid d)$.

Query Generation from Documents

- A **document language model** $M_d$is constructed for each document.
- The probability of a query given a document is calculated as:

$P(q \mid d) = \prod_{i=1}^{n} P(w_i \mid d)$ where:

- $q = \{w_1, w_2, \ldots, w_n\}$is the query,
- $P(w_i \mid d)$is the probability of word $w_i$occurring in document d.

Documents that assign higher probability to generating the query are considered more relevant.

Smoothing Techniques

Direct estimation of $P(w \mid d)$can lead to zero probabilities for unseen words. **Smoothing** is used to overcome this problem.

Common smoothing methods:

1. **Jelinek–Mercer smoothing**

2. **Dirichlet smoothing**

Advantages of Query Generation Approach

- Provides a **strong probabilistic foundation** for IR.
- Naturally integrates term frequency and document length normalization.
- Performs well for short queries.
- Avoids heuristic similarity measures.

Limitations

- Assumes query terms are generated independently.
- Sensitive to smoothing parameter selection.
- Computationally expensive for large document collections.
- Does not directly capture semantic relationships.

Language Models: Finite Automata and Language Models

A **language model** defines a set of valid strings (sentences) over an alphabet and assigns probabilities to them. In computer science and information processing, **finite automata** are one of the earliest and simplest formal models used to represent languages. Language models built using finite automata are fundamental in areas such as lexical analysis, pattern matching, and basic natural language processing.

Finite Automata

A **finite automaton (FA)** is a mathematical model of computation used to recognize patterns and define formal languages.

A finite automaton is defined as a 5-tuple:

$$(Q, \Sigma, \delta, q_0, F)$$

where:

- $Q$is a finite set of states
- $\Sigma$is a finite input alphabet
- $\delta$is the transition function
- $q_0$is the start state
- $F$is the set of accepting states

Finite automata can be:

- **Deterministic Finite Automata (DFA)**
- **Non-Deterministic Finite Automata (NFA)**

Both DFAs and NFAs recognize **regular languages**.

Finite Automata as Language Models

Finite automata can be viewed as **language models** by defining which strings are allowed or disallowed. A string is accepted if there exists a sequence of transitions from the start state to a final state.

In this context:

- The automaton defines the **structure of the language**
- All accepted strings belong to the language L
- Rejected strings are not part of the language

Probabilistic Finite Automata

To model uncertainty and variability in language, finite automata can be extended to **Probabilistic Finite Automata (PFA)**.

In a probabilistic FA:

- Each transition has an associated probability
- The sum of outgoing transition probabilities from a state equals 1
- The probability of a string is the product of transition probabilities along its path

$$P(w) = \prod_i P(q_{i+1} \mid q_i, w_i)$$

This makes finite automata suitable as **statistical language models**.

Advantages

- Simple and mathematically well-defined
- Efficient string recognition
- Suitable for regular patterns and local dependencies
- Widely used in lexical analyzers and speech recognition

Limitations

- Cannot model long-range dependencies
- Limited to regular languages
- Not suitable for complex natural language syntax
- State explosion problem for large vocabularies

Types of Language Models

A **language model (LM)** is a mathematical framework that assigns probabilities to sequences of words. It captures the structure, syntax, and statistical regularities of language and is a core component of many natural language processing systems such as speech recognition, machine translation, and information retrieval.

1. Rule-Based Language Models

Rule-based language models rely on explicitly defined grammatical and syntactic rules designed by linguists or domain experts. Based on formal grammar such as context-free grammar (CFG)

- Follow deterministic rules for sentence construction
- No learning from data is involved
- All valid sentences must strictly conform to predefined rules

Limitations:

- Difficult to cover all language variations
- Poor handling of ambiguity and real-world language usage

2. Statistical N-gram Language Models

N-gram models estimate the probability of a word based on a fixed number of previous words, using the Markov assumption. $P(w_1, w_2, \dots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_{i-1}, \dots, w_{i-(n-1)})$

- Common types: Unigram, Bigram, Trigram
- Probabilities are estimated from large text corpora
- Use smoothing techniques to handle unseen word sequences
- Capture local word dependencies effectively

Limitations:

- Cannot model long-range dependencies
- Suffers from data sparsity for large n

3. Finite Automata-Based Language Models

These models represent language using finite states and transitions between them. Each state represents a word history or context

- Transitions represent possible next words
- Can be deterministic or probabilistic
- Closely related to regular languages

Limitations:

- Limited expressive power
- State explosion for large vocabularies

4. Neural Network Language Models

Neural language models use neural networks to learn continuous representations of words and contexts.Replace discrete word counts with dense vector embeddings

- Learn complex syntactic and semantic relationships
- Use backpropagation for parameter optimization
- Can model variable-length context using RNNs and LSTMs

Advantages:

- Better generalization than n-gram models
- Reduced sparsity issues

5. Transformer-Based Language Models

Transformer models rely on self-attention mechanisms to model relationships between all words in a sequence.

- Eliminate recurrence and convolution
- Self-attention captures global context efficiently
- Support bidirectional and autoregressive modeling
- Enable large-scale pretraining and fine-tuning

Advantages:

- Handle long-range dependencies
- Achieve state-of-the-art results in NLP tasks

Multinomial Distributions over Words

In statistical language modeling and information retrieval, documents are often represented using the **bag-of-words assumption**, where the sequence of words is ignored and only word frequencies are considered. A **multinomial distribution over words** provides a probabilistic model to describe how words are generated within a document or query from a fixed vocabulary.

Vocabulary and Sample Space

- Let the vocabulary be $V = \{w_1, w_2, \dots, w_{|V|}\}$
- Each word corresponds to a distinct outcome of a random variable
- A document is treated as a sequence of independent word draws from this vocabulary
- The document length is fixed, but word occurrences can repeat

Multinomial Language Model

Each document $d$is associated with a parameter vector:

$$\theta_d = (P(w_1 \mid d), P(w_2 \mid d), \dots, P(w_{|V|} \mid d))$$

Theoretical properties:

- $\sum_{i=1}^{|V|} P(w_i \mid d) = 1$
- Each parameter represents the relative importance of a word in the document
- The model defines a probability distribution over all possible documents

Generative Assumptions

1. Words are generated **independently** of each other
2. Word order does not affect meaning
3. The probability of a word depends only on the document model
4. All words are drawn from the same vocabulary

Parameter Estimation (Maximum Likelihood)

The most common estimation method is **Maximum Likelihood Estimation (MLE)**:

$$P(w \mid d) = \frac{tf(w,d)}{\mid d \mid}$$

Theoretical implications:

- Directly reflects empirical word frequency
- Overfits small documents
- Produces zero probabilities for unseen words

Role in Information Retrieval

- Documents are ranked by the probability of generating a query
- Queries are also modeled as multinomial distributions
- Similarity between query and document is computed probabilistically
- Provides a theoretical alternative to TF-IDF and vector space models

Advantages

- Mathematically simple and interpretable
- Efficient for large text collections
- Strong probabilistic foundation
- Easy integration with smoothing and Bayesian methods

Limitations

- Ignores word order and syntax
- Assumes conditional independence
- Cannot model semantics or context
- Performance degrades for short documents

Divergence from Randomness (DFR)

Divergence from Randomness (DFR) is a probabilistic ranking framework in Information Retrieval that ranks documents based on how much the observed occurrence of query terms in a document **deviates from what is expected by chance**. The key idea is that informative terms occur **non-randomly** in relevant documents.

2. Randomness Hypothesis

DFR assumes that if a document is not relevant, the distribution of terms in it follows a **random process**. When a term appears more frequently than predicted by this random model, it provides evidence of relevance.

3. Measuring Divergence

The importance of a term is measured by the **information content** gained when its observed frequency diverges from randomness:

$$Information(t) = -\log_2 P_{random}(t)$$

Lower probability under randomness implies higher information and higher relevance.

4. Randomness Models

To estimate $P_{random}(t)$, DFR uses probabilistic distributions like:

- Binomial model
- Poisson model
- Geometric model

These models define how terms would behave if they occurred randomly in documents.

5. Term Frequency Normalization

Since longer documents naturally contain more terms, DFR applies **normalization** to adjust term frequency based on document length. This ensures that relevance is due to term informativeness rather than document size.

6. After-Effect (Risk Hypothesis)

DFR includes an **after-effect component** which models the risk of accepting a document as relevant after seeing a term. The first occurrence of a term contributes more information than repeated occurrences, reducing redundancy.

7. Document Ranking

The final score of a document is computed by **summing the divergence contributions of all query terms**, including randomness divergence, normalization, and after-effect. Documents with higher total divergence are ranked higher.

Passage Retrieval and Ranking

Passage Retrieval is an information retrieval technique in which **small text segments (passages)** are retrieved and ranked instead of entire documents. This approach is useful when only a specific part of a document is relevant to a user query. **Ranking** determines the order of retrieved passages based on their relevance.

In long documents, relevant information may be confined to a few sentences or paragraphs. Retrieving full documents can reduce precision. Passage retrieval improves:

- Relevance accuracy
- Focused information access
- User satisfaction in question answering systems

Thus, ranking passages instead of documents provides more precise results.

3. Passage Generation

Documents are divided into passages using:

- Fixed-length windows (e.g., N sentences)
- Paragraph-based segmentation
- Overlapping sliding windows

Each passage is treated as an independent retrieval unit.

4. Passage Representation

Each passage is represented using:

- Bag-of-words or vector representation
- Term frequency statistics
- Language model probabilities

This allows passages to be compared directly with the query.

5. Passage Ranking Principle

Passages are ranked based on their **relevance score** with respect to the query. In probabilistic models, ranking is often based on query likelihood:

$$Score(p,q) \propto P(q \mid p)$$

Passages that better explain or generate the query receive higher ranks.

6. Scoring and Normalization

Since passages vary in length, **normalization** is applied to avoid bias toward longer passages. Ranking functions may include:

- Term frequency normalization
- Length normalization
- Smoothing in language models

This ensures fair comparison between passages.

7. Advantages of Passage Ranking

- Higher precision than document-level ranking
- Better support for question answering
- Faster access to exact information
- Improved handling of long documents

Problems Associated with N-gram Models and Their Solutions

An **n-gram model** is a statistical language model that predicts a word based on the previous $n-1$words. Although simple and widely used, n-gram models suffer from several practical and theoretical limitations.

1. Data Sparsity Problem
As $n$increases, the number of possible n-grams grows exponentially. Many valid word sequences do not appear in the training corpus, even if the corpus is large.

- Use **smoothing techniques** such as Laplace, Good-Turing, or Kneser–Ney smoothing
- Reduce the value of n(bigram or trigram models)

2. Zero Probability Problem

If an n-gram never occurs in training data, it is assigned a probability of zero, which makes the probability of the entire sentence zero.

- Apply **smoothing** to assign small non-zero probabilities to unseen n-grams
- Use **backoff models**, which fall back to lower-order n-grams

3. Limited Context

N-gram models consider only the previous $n-1$words and cannot capture long-range dependencies in language.

- Increase the value of n(with caution)
- Use **interpolation** combining multiple n-gram levels
- Replace with **neural language models** for long-context handling

4. Large Memory Requirement

Storing all possible n-grams and their probabilities requires large memory, especially for higher-order models.

- Use **pruning techniques** to remove low-frequency n-grams
- Apply **compressed storage** methods
- Limit vocabulary size

5. Poor GeneralizationN-gram models rely on exact word sequences and do not generalize well to unseen or rare word combinations.

- Use **class-based n-gram models**

- Apply **word clustering**
- Switch to **neural embedding-based models**

6. Sensitivity to Domain
An n-gram model trained on one domain performs poorly when applied to another domain.

- Use **domain adaptation techniques**
- Combine multiple corpora
- Fine-tune the model on domain-specific data

Challenges in Natural Language Processing (NLP)
NLP enables computers to understand and process human language. However, human language is complex, ambiguous, and highly contextual, which makes NLP challenging.

1. Ambiguity
Words and sentences can have multiple interpretations.
Example: "I saw the man with a telescope."
It is unclear whether the man has a telescope or the observer used it.

2. Context Understanding
The meaning of words often depends on the surrounding text or situation.
Example: "It's cold here."
Could be a statement about temperature or a request to close a window.

3. Variability of Language
Different people, regions, or domains use language differently.
Example: "Big, large, huge" all convey the same idea but vary in usage.NLP systems must recognize synonyms, paraphrases, and style variations.

4. Long-Range Dependencies
Understanding relations between words far apart in a sentence is difficult.
Example: "The book that you gave me yesterday is interesting."
"The book" is the subject, and "is interesting" is the verb phrase separated by intervening words.

5. World Knowledge and Common Sense
Understanding often requires external knowledge beyond the text.
Example: "The glass fell and broke."
The model needs to know that falling objects made of glass usually break.

6. Noisy and Informal Text
Text from social media, chats, and SMS includes slang, abbreviations, typos, and emojis.
Example: "Gr8 job 👍"
NLP systems must interpret informal and noisy language correctly.

Different Levels of Natural Language Processing (NLP)
Natural Language Processing (NLP) involves analyzing, understanding, and generating human language using computational methods. NLP tasks are often organized into **multiple levels**, each focusing on different aspects of language.

1. Lexical Level
The lexical level deals with **words and their structure**. It focuses on identifying meaningful units of text and their forms.
Tasks at this level:

- **Tokenization:** Splitting text into words or sentences
- **Stemming:** Reducing words to their root form
- **Lemmatization:** Converting words to their dictionary form
- **Morphological Analysis:** Understanding word structure (prefix, root, suffix)

Example:

- Text: *"The children were playing in the gardens."*
- Tokens: *The, children, were, playing, in, the, gardens*
- Stemmed forms: *child, play, garden*

Purpose: Helps in standardizing text and reducing redundancy.

2. Syntactic Level
The syntactic level analyzes **sentence structure** and grammatical relationships between words. It ensures that words are arranged in a grammatically correct way.
Tasks at this level:

- **Part-of-Speech (POS) Tagging:** Identifying nouns, verbs, adjectives, etc.
- **Parsing:** Constructing a parse tree representing sentence structure
- **Grammar Checking:** Detecting grammatical errors

Example:

- Sentence: *"The cat sat on the mat."*
- POS Tags: *The (Det), cat (Noun), sat (Verb), on (Prep), the (Det), mat (Noun)*
- Parse Tree: Shows subject → verb → object relationships

Purpose: Provides syntactic structure for higher-level analysis.

3. Semantic Level
The semantic level deals with **meaning of words and sentences**. It captures the relationships between words and their meanings in context.
Tasks at this level:

- **Word Sense Disambiguation (WSD):** Identifying correct meaning of ambiguous words
- **Named Entity Recognition (NER):** Identifying entities like names, locations, dates
- **Semantic Role Labeling:** Understanding who did what to whom

Example:

- Word: *"bank"*
  - Financial institution: *"I deposited money in the bank."*
  - River edge: *"He sat on the river bank."*

Purpose: Enables systems to understand the actual meaning rather than just words.

4. Discourse Level
The discourse level analyzes **meaning across multiple sentences** and focuses on coherence and relationships between sentences.
Tasks at this level:

- **Coreference Resolution:** Identifying pronouns or phrases referring to the same entity
- **Discourse Analysis:** Understanding how sentences relate to each other

Example:

- Text: *"John went to the park. He saw a dog."*
- Coreference: *"He" refers to John.*

Purpose: Ensures context and consistency in understanding long texts.

5. Pragmatic Level
The pragmatic level deals with **intended meaning and real-world knowledge**. It considers context, speaker intention, and social conventions.
Tasks at this level:

- **Intent Detection:** Understanding the purpose behind a statement
- **Speech Act Recognition:** Recognizing requests, commands, or questions
- **Contextual Interpretation:** Using world knowledge to understand implied meaning

Example:

- Sentence: *"Can you open the window?"*
- Literal meaning: Question about ability

- Intended meaning: Request to open the window

Purpose: Helps machines understand user intention beyond literal words.

IR U 4

Text Classification is the process of automatically assigning predefined categories or labels to text documents. It is a major task in Natural Language Processing (NLP) and Information Retrieval systems. The goal is to analyze the content of a document and determine the class it belongs to based on patterns learned from training data.

Key Concepts

- **Supervised learning:** The model learns from a labeled dataset.
- **Document representation:** Text is converted into numerical vectors using bag-of-words, TF-IDF, or embeddings.
- **Learning algorithms:** Models such as Naive Bayes, KNN, SVM, and neural networks are used.

Process

1. **Preprocessing:** Tokenization, stop-word removal, stemming, normalization.
2. **Feature Extraction:** Converting text into vectors.
3. **Training:** Model identifies patterns using labeled examples.
4. **Prediction:** New documents are assigned the most likely class.

Applications

- Email spam detection
- Sentiment analysis
- News categorization
- Document indexing and filtering

Advantages

- Scales to large datasets
- Automates manual classification
- Provides quick, consistent decisions

Naive Bayes Model for Text Classification

The **Naive Bayes model** is a probabilistic text classification technique based on **Bayes' theorem**. It classifies a document by calculating how likely it belongs to each class, given the words it contains. The model is called "naive" because it assumes that all words in a document occur **independently** of each other, which simplifies computation.

How Naive Bayes Works in Text Classification

1. The system first learns from a **labeled dataset** where each document has a known category.
2. It calculates how often each word occurs in documents of each class.
3. For a new document, the model evaluates how strongly its words indicate each class.
4. The class that gets the highest overall probability is selected as the predicted label.

Why Naive Bayes Works Well

- Text data is high-dimensional; Naive Bayes handles thousands of features easily.
- The independence assumption works surprisingly well in practice.
- It is very fast and efficient for both training and classification.

Applications of Naive Bayes Classifier

1. Text Classification
Used to categorize documents such as news articles or research papers based on word probabilities.
Performs well on high-dimensional text data.

2. Spam Email Filtering
Classifies emails as spam or non-spam using frequencies of suspicious words.
Fast and widely used in real-time email systems.

3. Sentiment Analysis
Determines whether text expresses positive, negative, or neutral sentiment.
Commonly applied to reviews, tweets, and feedback analysis.

4. Medical Diagnosis
Predicts the likelihood of diseases based on symptoms and patient data.
Helps in decision support systems for healthcare.

5. Recommendation Systems
Suggests products or content based on user preferences and past behavior.
Uses probability to predict user interest.

6. Fraud Detection
Identifies fraudulent transactions by classifying activities as normal or abnormal.
Useful due to its speed and ability to handle large datasets.

Advantages

- Extremely fast and scalable
- Requires small training data
- Performs well even when data is noisy
- Easy to implement

Limitations

- Fails to capture relationships between words (context)
- Independence assumption is unrealistic but practical

K-Nearest Neighbor (KNN) for Text Classification

K-Nearest Neighbor is a simple, instance-based learning method used for classifying text documents. Unlike other algorithms, KNN does not build an explicit training model. Instead, it classifies a new document based on how similar it is to previously labeled documents.

Working of KNN in Text Classification

1. Convert documents into numerical vectors using representations like TF-IDF.
2. When a new document arrives, compute its similarity to all existing documents using cosine similarity or distance measures.

3. Select the **k nearest documents** (neighbors) that are most similar.
4. Assign the new document to the class that occurs most frequently among these k neighbors.

Strengths

- Very simple and intuitive
- No training phase; the algorithm learns directly from stored data
- Works well for multi-class classification
- Effective for smaller datasets and non-linear decision boundaries

Weaknesses

- Slow classification process since it compares with every document
- Requires large memory to store all training documents
- Sensitive to noise and irrelevant features
- Performance depends heavily on the value of **k** and similarity metric

Applications

- Document recommendation
- Topic classification
- Basic categorization tasks in small systems

KNN is widely used as an easy-to-understand baseline model and performs well when data is clean and properly represented.

Spam Filtering

Spam filtering is a text classification task used to automatically detect and block unwanted email messages. It is one of the earliest successful applications of machine learning in text classification.

Purpose of Spam Filtering

Spam messages often contain phishing attempts, malware links, fake promotions, and advertising content. Filtering helps protect users and ensures inbox cleanliness.

How Spam Filtering Works

1. Collect a dataset of **spam** and **non-spam** emails.
2. Extract features such as word occurrences, sender address patterns, suspicious URLs, and email structure.
3. Train a classification model such as Naive Bayes, Support Vector Machine, KNN, or neural networks.
4. Apply the trained model to new emails to classify them as spam or legitimate.

Techniques Used

- **Naive Bayes filters** (most common due to speed and accuracy)
- **Rule-based filtering** using keywords and patterns
- **Blacklists and whitelists**
- **Machine learning-based classifiers**
- **Heuristic patterns** (e.g., excessive symbols, promotional language)

Common Indicators of Spam

- Words like “FREE”, “WIN”, “LIMITED OFFER”
- Suspicious sender domains
- Embedded tracking links or malware URLs
- Strange formatting or repeated characters

Advantages

- Protects users from phishing and fraud
- Reduces risk of malware infection
- Saves time by keeping inbox clean
- Highly accurate when trained with quality data

Limitations

- Spammers constantly change strategies
- False positives may block important emails
- Requires continuous updating and retraining

Spam filtering remains a crucial real-world application of text classification and forms a core part of email services today.

Support Vector Machine (SVM) Classifier

Support Vector Machine (SVM) is a powerful supervised learning algorithm widely used in text classification because it performs extremely well in high-dimensional spaces. Text data often contains thousands of features (words), and SVM is specifically designed to handle such situations.

Basic Principle

SVM aims to find the **best separating boundary**, known as a *hyperplane*, that divides documents of different classes. The key idea is to choose the hyperplane that **maximizes the margin**, i.e., the distance between the hyperplane and the nearest data points from each class (called support vectors). A larger margin generally leads to better generalization and classification accuracy.

Why SVM Works Well for Text

1. Text features are high-dimensional and sparse—SVM handles this efficiently.
2. It is robust to noise and irrelevant features.
3. SVM can model both linear and non-linear patterns using kernel functions.
4. It produces strong classification accuracy even with limited training data.

Key Characteristics

- Uses only the most informative points (support vectors) to define the decision boundary.
- Can handle multi-class classification with appropriate extensions.
- Avoids overfitting by maximizing the margin.

Advantages

- Very high accuracy for text classification tasks.
- Good performance for large feature spaces such as TF-IDF vectors.
- Works well when classes are well separated.
- Effective for spam filtering, sentiment analysis, and topic categorization.

Limitations

- Training can be slow on extremely large datasets.
- Requires careful tuning of parameters such as C and kernel type.
- Less interpretable compared to simpler models.

Vector Space Classification Using Hyperplanes

The Vector Space Model converts documents into numerical vectors where each dimension corresponds to a term (word) or feature. Classification in this space involves separating these vectors using **hyperplanes**.

A hyperplane is a linear decision boundary that divides the feature space into regions corresponding to different classes. Text classification models like SVM, perceptron, and logistic regression rely on hyperplanes to determine class membership.

How It Works

1. Convert documents into vectors using representations like Bag-of-Words or TF-IDF.
2. Each document becomes a point in a high-dimensional space.
3. The model identifies a hyperplane that best separates documents of different categories.
4. Documents fall on one side or the other depending on their features.
5. New documents are classified based on their position relative to the hyperplane.

Advantages

- Works very well for high-dimensional text data.
- Provides a clean geometric interpretation of classification.
- Can handle thousands of features without complexity issues.
- Forms the theoretical backbone of many modern classification algorithms.

Applications

- Spam vs non-spam classification
- Topic identification
- Sentiment classification
- Document clustering and categorization

3. Kernel Function

Kernel functions are mathematical tools used to extend linear classifiers like SVM to solve **non-linear classification problems**. They allow the classifier to operate in a higher-dimensional space **without explicitly computing the coordinates** in that space. This is known as the **kernel trick**.

Purpose of Kernel Functions

- Many classification problems cannot be separated by a straight hyperplane.
- Kernel functions transform data into a new space where the classes become linearly separable.
- This transformation is done implicitly, making the computation efficient.

1. Linear Kernel

$K(x,y)=x\cdot y$

Key Points:

- Best suited for **linearly separable data**
- Widely used in **text classification**
- Works efficiently with **high-dimensional sparse data**
- **Fast computation** and low memory usage

2. Polynomial Kernel

$K(x,y)=(x\cdot y+c)d$

Key Points:

- Captures **feature interactions** up to degree d
- Useful for **non-linear but structured relationships**
- Higher degree increases model complexity
- Risk of **overfitting** for large d

3. Radial Basis Function (RBF) Kernel

$K(x,y)=\exp(-\gamma\|x-y\|^2)$

Key Points:

- Handles **complex non-linear decision boundaries**
- Maps data into **infinite-dimensional space**
- Very powerful for **complex datasets**
- Sensitive to parameter γ

4. Sigmoid Kernel

$K(x,y)=\tanh(\alpha\, x\cdot y+c)$

Key Points:

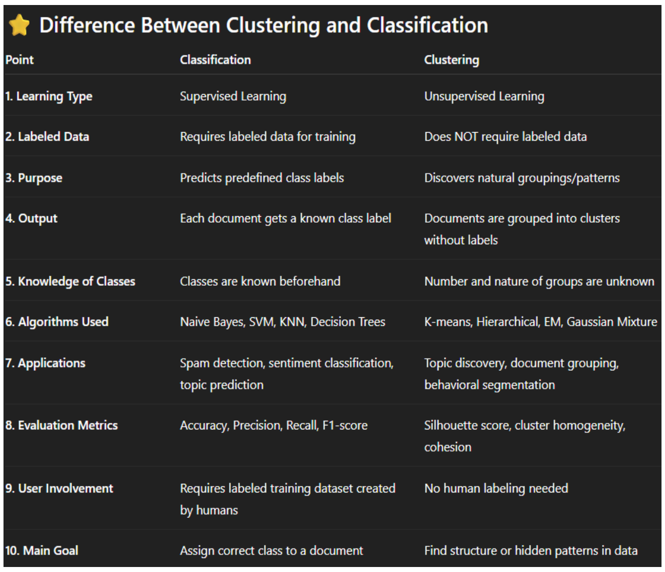
- Inspired by **neural network activation functions**
- Suitable for **specific applications**
- Not always positive semi-definite
- Performance depends strongly on parameters

Advantages of Kernel Functions

- Allow flexible and powerful classifiers.
- Enable SVM to handle complex, non-linear patterns.
- Avoid explicit mapping to huge feature spaces, reducing computation cost.
- Work extremely well for tasks where a linear classifier is insufficient.

Limitations

- Some kernels require parameter tuning.
- Incorrect kernel choice can reduce accuracy.
- Computational cost increases for large datasets.



⭐ Difference Between Clustering and Classification

Point	Classification	Clustering
1. Learning Type	Supervised Learning	Unsupervised Learning
2. Labeled Data	Requires labeled data for training	Does NOT require labeled data
3. Purpose	Predicts predefined class labels	Discovers natural groupings/patterns
4. Output	Each document gets a known class label	Documents are grouped into clusters without labels
5. Knowledge of Classes	Classes are known beforehand	Number and nature of groups are unknown
6. Algorithms Used	Naive Bayes, SVM, KNN, Decision Trees	K-means, Hierarchical, EM, Gaussian Mixture
7. Applications	Spam detection, sentiment classification, topic prediction	Topic discovery, document grouping, behavioral segmentation
8. Evaluation Metrics	Accuracy, Precision, Recall, F1-score	Silhouette score, cluster homogeneity, cohesion
9. User Involvement	Requires labeled training dataset created by humans	No human labeling needed
10. Main Goal	Assign correct class to a document	Find structure or hidden patterns in data

Can clustering be used for classification purposes?

Yes, clustering can be used for classification, but **indirectly**.

- **Clustering** is an **unsupervised learning** technique that groups data based on similarity **without class labels**.
- **Classification** is a **supervised learning** technique that assigns **predefined labels** to data.

How clustering helps in classification

1. **Labeling clusters**
   After clustering, each cluster can be assigned a class label based on the **majority class** of labeled samples in that cluster.
2. **New data classification**
   A new data point is assigned to the **nearest cluster**, and the **cluster's label** is used as the predicted class.
3. **Feature generation**
   Cluster IDs or distances to clusters can be used as **features** in a classifier.

Example

- Customer data is clustered into 3 groups based on behavior.
- Later, clusters are labeled as **Low-Value, Medium-Value, High-Value** customers.
- New customers are classified by finding the closest cluster.

Partitioning Methods

Partitioning methods are a family of clustering techniques that divide a dataset into a fixed number of clusters. Each document is assigned to exactly one cluster based on similarity to a central representative.

Partitioning methods treat clustering as an optimization problem. The dataset is partitioned into **k clusters**, and the algorithm tries to:

- Minimize within-cluster distance (documents in the same cluster are similar)
- Maximize between-cluster distance (different clusters are well separated)

The number of clusters **k** is usually user-defined.

2. Steps in Partitioning Methods

1. **Choose k**, the number of clusters.
2. **Initialize cluster centers** (randomly or using heuristics).
3. **Assign documents** to the nearest cluster based on a similarity/distance measure.
4. **Update centroids** based on newly assigned points.
5. Repeat until the cluster assignments stabilize.

Distance is often computed using **cosine similarity** for text data.

3. Common Partitioning Algorithms

(a) K-Means Clustering

- Most widely used partitioning algorithm.
- Uses mean value of documents as the cluster center.
- Repeats assignment and update steps until convergence.

(b) K-Medoids / PAM

- Uses actual data points (medoids) instead of means as cluster centers.
- More robust to outliers than k-means.

(c) CLARANS (Clustering Large Applications)

- Designed for large datasets.
- Combines sampling with k-medoids for scalability.

4. Strengths of Partitioning Methods

- Simple and easy to implement
- Efficient for large datasets
- Produces compact and meaningful clusters
- Works well when clusters are spherical and of similar size

5. Limitations

- Requires the user to specify the number of clusters (k).
- Sensitive to initial cluster centers.
- Not suitable for clusters with irregular shapes.
- Can get stuck in local optima.

6. Applications

- Text document grouping
- Customer segmentation
- Image clustering
- Topic discovery
- Web page grouping

K-Means Clustering for Text Clustering

K-means is one of the most popular algorithms used for **text clustering**, where the goal is to group similar documents together. It is a **partitioning-based** method that divides documents into *k* clusters, each represented by a **centroid**, which is the average vector of documents in that cluster.

How K-Means Works in Text Mining

1. **Choose k**, the number of clusters (topics).
2. **Initialize centroids** randomly from the document vectors.

3. **Assignment Step:** Assign each document to the nearest centroid using **cosine similarity**, which works well for text.
4. **Update Step:** Recompute each centroid as the mean TF-IDF vector of documents in that cluster.
5. **Repeat** the two steps until assignments stabilize.

Key Features for Text

- Represents documents as high-dimensional vectors (TF-IDF).
- Minimizes distance between documents and cluster centroid.
- Efficient for large collections like news articles, webpages, tweets.

Advantages

- Simple and fast even for millions of documents.
- Easy to implement and understand.
- Suitable for text datasets with well-defined topical clusters.

Limitations

- Requires pre-setting k, the number of topics.
- Sensitive to noisy or outlier documents.
- Clusters may not be meaningful if documents overlap in topics.
- Initialization affects final clusters.

Text Applications

- Topic grouping of news articles
- Clustering search engine results
- Document organization in digital libraries

Agglomerative Hierarchical Clustering for Text

Agglomerative hierarchical clustering is a **bottom-up text clustering method** that builds a hierarchical structure called a **dendrogram**. It is highly useful for exploring document similarity relationships.

How It Works

1. Begin with **each document as a separate cluster**.
2. Compute similarity between all pairs using cosine similarity (best for text).
3. **Merge the two most similar clusters**.
4. Update similarity matrix.
5. Repeat merging until all documents form a single tree-like structure.

Linkage Methods

- **Single linkage**: merges closest pair of documents.
- **Complete linkage**: merges clusters based on their farthest documents.
- **Average linkage**: averages distances.

Advantages

- No need to specify number of clusters; choose by cutting the dendrogram.
- Reveals hierarchical and nested document groups.
- Useful when studying how documents relate at multiple levels.

Limitations

- Computationally expensive for large text datasets.
- Once two clusters merge, it cannot be undone.
- Sensitive to noisy text or very short documents.

Text Applications

- Organizing research papers
- Building topic hierarchies
- Visualizing similarity between text documents

Expectation-Maximization (EM) Algorithm for Text Clustering

EM is a **probabilistic clustering method** which performs **soft clustering**, meaning each document belongs to clusters with certain probabilities. This captures the fact that documents may relate to multiple topics.

Two-Step Process

1. E-Step (Expectation Step)

- Compute the probability that each document belongs to each cluster.
- This uses current cluster parameters (means, variances).

2. M-Step (Maximization Step)

- Update cluster parameters using the probabilities calculated in E-step.
- Refines cluster centroids and spreads.

These steps repeat until there is no significant improvement.

Why EM is Good for Text

- Handles overlapping topics (e.g., a sports-business article).
- Assigns partial membership, which reflects real document behavior.
- More flexible than k-means because clusters can take different shapes.

Advantages

- Produces soft, probabilistic clusters.
- Useful for complex and ambiguous document sets.
- Works well when documents may belong to multiple topics.

Limitations

- Slower and more computationally expensive than k-means.
- Sensitive to initialization; may converge to local optima.
- Often uses k-means to get initial cluster positions.

Text Applications

- Clustering mixed-topic articles
- Soft topic assignment in news or blogs
- Document classification with overlapping themes

Mixture of Gaussians Model (GMM) for Text Clustering

A Gaussian Mixture Model assumes that text documents are generated from multiple **Gaussian distributions**, each representing a different cluster or topic. It is one of the most powerful probabilistic clustering methods.

Key Concepts for Text

- Each cluster corresponds to a Gaussian distribution in vector space.
- Documents have **probable membership**, not fixed membership.
- The EM algorithm estimates parameters for each Gaussian.

How GMM Works

1. Initialize Gaussian parameters (mean, covariance, weight).
2. **E-Step:** Compute probability of each document belonging to each Gaussian cluster.
3. **M-Step:** Update the Gaussian parameters using these probabilities.
4. Repeat until convergence.

Advantages

- Can model **elliptical or unevenly shaped** clusters, unlike k-means.
- Handles overlapping document groups naturally.
- Provides a full probabilistic interpretation of text clusters.
- More flexible than k-means because each Gaussian can have its own shape.

Limitations

- Computationally expensive for large text datasets.
- Requires assuming text data fits a Gaussian distribution.
- Sensitive to initialization.

Text Applications

- Topic modelling
- Grouping articles by complex themes
- Anomaly detection in textual logs or emails
- Speaker/document identification

IR U5

1. Parallel Query Processing

Parallel Query Processing is a technique in Information Retrieval (IR) where **multiple processors or machines work simultaneously** to process user queries. This is essential because modern search engines must handle **billions of documents** and **millions of queries per second**. Parallel processing improves **speed, scalability, throughput, and system performance**.

2. Need for Parallel Query Processing

- The web contains extremely large document collections.
- Single-processor indexing/search is too slow.
- Multiple queries arrive simultaneously.
- Real-time retrieval demands fast response times.
- Large-scale distributed search engines require parallel architecture.

3. Models of Parallel Query Processing

(a) Document Partitioning

- The document collection is split across machines.
- Each machine builds its own **local inverted index**.
- A user query is sent to all machines.
- Each machine returns its result, and the system merges them.
  Advantages: Simple, easy scaling
  Limitation: Merging results may be slow

(b) Term Partitioning

- The index is partitioned by **terms**, not documents.
- Machine 1 stores terms A–G
- Machine 2 stores terms H–P, etc.
- Query is split based on the terms it contains.
  Advantage: Compact index per machine
  Limitation: Query must communicate with many nodes

(c) Hybrid Partitioning

- Combination of document and term partitioning.
- Balances load between machines.
- Used in large-scale search engines.

4. Techniques in Parallel Query Processing

- **Load Balancing** → ensures equal workload distribution
- **Distributed Hashing** → maps documents/terms to specific nodes
- **Result Merging** → combines ranked lists from all machines
- **Caching** → stores common queries for faster reuse
- **Fault Tolerance** → continues processing even if a node fails

5. Advantages

- High-speed query processing
- Handles massive data efficiently
- Scales horizontally (add more machines)
- Improves throughput for multiple simultaneous users
- Reduces latency during heavy search loads

6. Challenges

- Synchronization between distributed machines
- Network overhead when merging results
- Fault tolerance complexity
- Data replication cost
- Uneven distribution in partitioning

2. MapReduce in Information Retrieval

MapReduce is a **distributed programming model** used for processing large-scale data across clusters. Introduced by Google, it simplifies tasks such as **index building, web mining, query analysis, and log processing** in IR. It follows two major phases: **Map** and **Reduce**, which automatically manage parallel execution, fault tolerance, and scalability.

2. Why MapReduce for IR?

- IR requires processing **massive text datasets**.
- Building inverted indexes manually is slow.
- Parallel clusters can process large amounts of web data.
- MapReduce automatically handles data distribution.

3. Working of MapReduce

(a) Map Phase

- Input text documents are divided into blocks.
- Each mapper processes one block.
- Outputs **key-value pairs** such as:
  (term, documentID)
  Example:
  Document: "data mining techniques"
  Mapper output:(data, doc1),(mining, doc1),(techniques, doc1)

(b) Shuffle & Sort Phase

- Intermediate results are grouped by key (term).
- All entries with the same term are collected together.
- Automatically handled by the framework.

(c) Reduce Phase

- Reducer receives all values for a term.
- Builds **posting lists** or inverted index entries:
  Example: **mining → [doc1, doc5, doc20]**

4. Applications of MapReduce in IR

- **Inverted Index Construction**
- **Query Log Analysis**
- **Web Mining**
- **PageRank Computation**
- **Spam Detection**
- **Text Classification & Clustering**

5. Advantages of MapReduce

- **Scalable:** works on thousands of nodes
- **Fault Tolerant:** automatically restarts failed tasks
- **Parallel Execution:** tasks distributed to multiple machines
- **Simple Programming Model:** only Map() and Reduce() functions
- **Handles Big Data Easily:** ideal for large IR tasks

6. Limitations

- High I/O overhead due to disk operations
- Not suitable for real-time search
- Slow for small datasets
- Multiple MapReduce stages may be needed for complex tasks

7. Tools & Frameworks

- Hadoop MapReduce
- Apache Spark (MapReduce-like faster system)
- Google Cloud Dataflow
- Amazon EMR

Web Crawler

A **Web Crawler** (also called Spider/Robot) is an automated program that systematically browses the World Wide Web to download and collect web pages. Crawlers form the foundation of search engines, data mining systems, and many online applications. They fetch pages, extract links, and continuously explore the web.

Web Crawler Structure

A Web Crawler typically has several key components that work together:

(a) Seed URLs

- Crawling begins from a set of starting URLs.
- These links act as the entry point to the web.

(b) URL Frontier (Queue System)

- A data structure (queue or priority queue) storing URLs to visit next.
- Maintains crawl order such as breadth-first, depth-first, or priority-based crawling.

(c) Fetching Module

- Sends HTTP requests to fetch HTML pages from servers.
- Follows politeness policies and robots.txt rules.

(d) Parser

- Extracts text content, metadata, and links from HTML.
- Uses HTML parsing libraries to identify hyperlinks.

(e) Duplicate & Loop Detection

- Prevents revisiting the same page multiple times.
- Uses hashing or checksums to detect duplicate URLs/content.

(f) Storage Component

- Stores the fetched pages, parsed data, or extracted information.
- Data may be saved in databases or indexing systems.

Web Crawler Libraries

Web Crawler libraries provide pre-built tools and frameworks that simplify the development of web crawlers. Instead of manually writing code for fetching pages, parsing HTML, and handling links, these libraries offer ready-made modules for efficient and scalable crawling. They support tasks like HTML parsing, link extraction, scheduling, request handling, and working with dynamic content.

(a) Scrapy

Scrapy is a **powerful, fast, and scalable Python framework** designed specifically for web crawling and scraping. It is used by professionals to build large-scale crawlers.

Key Features

- **Asynchronous architecture** → faster crawling
- **Spiders** → custom classes to define crawl behavior
- **Item Pipelines** → clean, validate, and store extracted data
- **Built-in link extractor** for discovering URLs
- **Auto-throttling** and **politeness policies**

Benefits

- Extremely fast
- Modular and easy to extend
- Suitable for crawling large websites or millions of pages

Use Cases

- E-commerce data extraction
- Price tracking
- News or blog crawling

(b) Beautiful Soup

Beautiful Soup is a Python library designed for **HTML and XML parsing**. It is commonly used for small to medium crawling tasks.

Key Features

- Parses poorly formed HTML gracefully
- Provides tree-based navigation (find, find_all)
- Works well with **Requests** to fetch pages
- Extracts specific elements like tables, headings, links

Benefits

- Very easy to use
- Great for scraping simple static websites
- Supports multiple parsers such as html.parser, lxml

Limitations

- Not suitable for large-scale crawling
- Cannot execute JavaScript

(c) Requests

Requests is a simple and elegant Python library used for **sending HTTP requests**.

Key Features

- Supports GET, POST, PUT, DELETE
- Handles cookies, headers, authentication
- Integrates easily with Beautiful Soup for parsing
- Easy error handling and redirection support

Benefits

- Human-readable API
- Lightweight and highly reliable
- Ideal for downloading web pages or interacting with APIs

Use Case

- Used as the base module for many small web crawlers

(d) Selenium

Selenium is a browser automation tool that can control a real browser (Chrome, Firefox, Edge). It is used for **JavaScript-heavy or dynamic websites**.

Key Features

- Executes JavaScript
- Interacts with web elements (click, fill forms, scroll)
- Supports full browser rendering
- Handles login-based or infinite scrolling pages

Benefits

- Extracts content that normal crawlers cannot see
- Ideal for testing and scraping dynamic web apps

Limitations

- Slower than Scrapy or Requests
- Requires more resources (browser automation)

1. Python Scrapy

Scrapy is a **powerful, open-source Python framework** designed for **web crawling and web scraping**. Unlike simple libraries, Scrapy provides a full architecture for large-scale crawlers. It is widely used to extract structured data from websites, especially when high speed and scalability are required.

2. Key Components of Scrapy

(a) Spiders

- Core component that defines *how* a website will be scraped.
- Contains start URLs and parsing logic.

(b) Engine

- Controls the entire data flow between different components.

(c) Scheduler

- Maintains the queue of pending URLs.
- Decides which request to process next.

(d) Downloader

- Responsible for sending HTTP requests and receiving responses.

(e) Item Pipeline

- Cleans, processes, and stores extracted data.

3. How Scrapy Works (Flow)

1. Spider sends initial request to start URLs.
2. Downloader fetches page and sends HTML back to Spider.
3. Spider parses content and extracts required data.
4. Extracted links are added to Scheduler for further crawling.
5. Cleaned items are passed through Item Pipelines and stored.

4. Features of Scrapy

- **Asynchronous Crawling:** High-speed and efficient.
- **Auto-Throttle:** Prevents overloading web servers.
- **Built-in Link Extractors:** Automatically follows hyperlinks.
- **Middleware Support:** Allows custom request/response handling.
- **Exporters:** Saves data as JSON, CSV, XML, DB, etc.

5. Advantages of Scrapy

- Very **fast** due to asynchronous architecture.
- Highly **scalable** for million-page crawls.
- Easy to extend and customize.
- Supports logging, caching, and error handling.
- Suitable for real-world industrial crawling.

6. Applications of Scrapy

- E-commerce price tracking
- News aggregation
- Blog crawling
- Market research & competitor analysis
- Scraping large datasets for machine learning

2. Beautiful Soup

Beautiful Soup is a **Python library for parsing HTML and XML documents**. It is not a crawler by itself, but is mainly used for **extracting information from downloaded web pages**. It works well with Requests library for small-scale scraping tasks.

2. Features of Beautiful Soup

(a) Easy HTML Parsing

- Handles badly formatted HTML gracefully.
- Finds elements using tags, attributes, CSS selectors.

(b) Supports Multiple Parsers

- html.parser
- lxml
- html5lib

(c) Tree-Based Navigation

- Methods like .find(), .find_all(), .select()
- Easy to navigate parent, child, and sibling elements.

3. How Beautiful Soup Works

1. Fetch the web page using **Requests**.
2. Load HTML content into BeautifulSoup object.
3. Parse required elements (titles, links, paragraphs, tables).
4. Clean and store extracted data.

4. Advantages of Beautiful Soup

- Very simple and beginner-friendly.
- Best for **small-scale or single-page scraping**.
- Handles incorrect HTML structures.
- Works well for text extraction and data cleaning.

5. Limitations

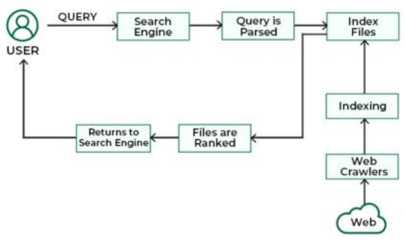
- Not suitable for large crawls (slow).
- Cannot execute JavaScript (unlike Selenium).
- Needs to be combined with Requests for full crawling functionality.

6. Applications of Beautiful Soup

- Extracting article content from blogs
- Scraping static product pages
- Extracting tables & HTML forms
- Data cleaning for research
- Academic and project-based scraping tasks

Web Retrieval: Search Engine Architectures

A **Search Engine Architecture** defines the overall structure and components used to collect, store, process, and retrieve web information efficiently in response to user queries. It is designed to handle **large-scale, dynamic, and distributed web data**.



1. Web Crawler (Spider)

- The crawler automatically visits web pages, follows hyperlinks, and downloads content.
- It ensures wide web coverage and periodically revisits pages to keep data fresh.
- Uses **URL frontier**, politeness policy, and duplicate detection.

2. Document Processing & Parsing

- Retrieved pages are cleaned by removing HTML tags, scripts, and ads.
- Text is tokenized, normalized (lowercasing, stemming), and stop words are removed.
- Extracts metadata like title, headings, anchor text, and links.

3. Indexing Module

- Processed documents are stored in an **inverted index** (term → document list).
- Supports fast lookup during query processing.
- Stores term frequency (TF), document frequency (DF), and positional information.

4. Document Repository

- Stores original or compressed copies of crawled web pages.
- Used for re-indexing, snippet generation, and cache retrieval.

5. Query Processor

- Accepts user queries and applies preprocessing (tokenization, stemming).
- Expands queries using synonyms, spelling correction, or relevance feedback.
- Converts queries into a form suitable for matching against the index.

6. Ranking Engine

- Assigns relevance scores to documents using ranking algorithms.
- Uses models like **TF-IDF, BM25, PageRank**, and learning-to-rank methods.
- Combines content relevance with link analysis and user behavior signals.

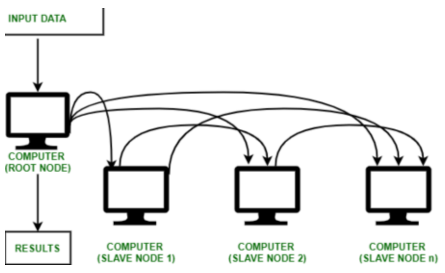
7. Result Formatter & Snippet Generator

- Selects top-ranked documents and generates short summaries (snippets).
- Highlights query terms and formats results for user presentation.

8. User Interface & Feedback Module

- Displays ranked results to users.
- Collects user interactions (clicks, dwell time) to improve future ranking.
- Supports advanced search options and personalization.

Cluster-based architecture uses a group of tightly connected computers (nodes) that work together as a single system to handle large-scale web retrieval tasks.



- In this architecture, multiple servers are organized into a **cluster**.
- Each node performs a specific role such as crawling, indexing, or query processing.
- Data is divided among nodes using **data partitioning** or **index sharding**.
- A **master node** coordinates tasks, while worker nodes execute them.
- Load balancing distributes user queries across nodes to avoid overload.
- If one node fails, others continue working, ensuring **fault tolerance**.

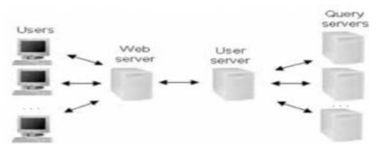
Advantages:

- High performance and fast query response
- Scalable by adding more nodes
- Efficient resource utilization

Limitation:

- Requires complex coordination and cluster management

Distributed architecture spreads search engine components across multiple geographically separated systems connected via a network.



- Components like crawlers, indexes, and query processors run on **different machines or data centers**.
- Each system maintains its **local index**.
- User queries are broadcast to multiple servers, and results are merged.
- Supports **horizontal scalability** across locations.
- Improves reliability by avoiding single-point failure.
- Enables regional data handling for faster local access.

Advantages:->Highly scalable and reliable,Supports global web search,Better fault isolation.

Limitation:->Network latency can affect performance.

Search engine ranking

It is the process by which a search engine **orders web pages or documents** in a results list based on their **relevance and importance** to a user's query.

When a user submits a query, the search engine:

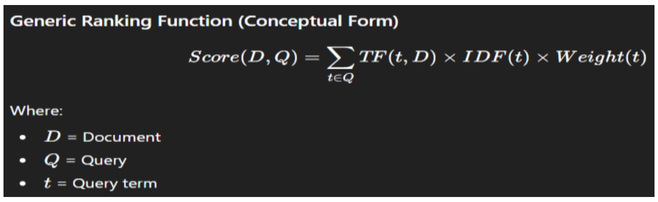
- Analyzes the query
- Evaluates all matching documents
- Assigns a **ranking score** to each document
- Displays results from **highest to lowest score**

The page ranked at the top is considered the **most relevant** to the query.

A **search engine ranking function** is a mathematical or algorithmic function used to **assign a relevance score** to each document for a given user query. Documents are then **sorted in descending order of score**, and the most relevant ones are shown at the top of the search results.

Explanation of Ranking Function (Step-wise)

1. **Query–Document Matching**
   - The ranking function measures how well a document matches the user query.
   - Matching is based on query terms appearing in the document content, title, headings, or anchor text.
2. **Term Frequency (TF)**
   - Measures how often a query term appears in a document.
   - Higher frequency usually indicates higher relevance.
3. **Inverse Document Frequency (IDF)**
   - Reduces the importance of common terms that appear in many documents.
   - Rare but meaningful terms get higher weight.
4. **Document Length Normalization**
   - Prevents long documents from getting unfairly high scores just because they contain more words.
   - Normalizes scores based on document size.
5. **Link-Based Importance**
   - Uses link analysis (e.g., inbound links) to estimate document authority.
   - Pages linked by many important pages are ranked higher.
6. **User and Context Signals**
   - Modern ranking functions consider click-through rate, dwell time, freshness, and location.
   - Improves relevance based on user behavior.


Generic Ranking Function (Conceptual Form)

$$Score(D,Q) = \sum_{t \in Q} TF(t,D) \times IDF(t) \times Weight(t)$$

Where:
- D = Document
- Q = Query
- t = Query term

Purpose of Ranking Function

- To retrieve **relevant documents efficiently**
- To order results based on **usefulness and importance**
- To improve **user satisfaction** by showing best results first

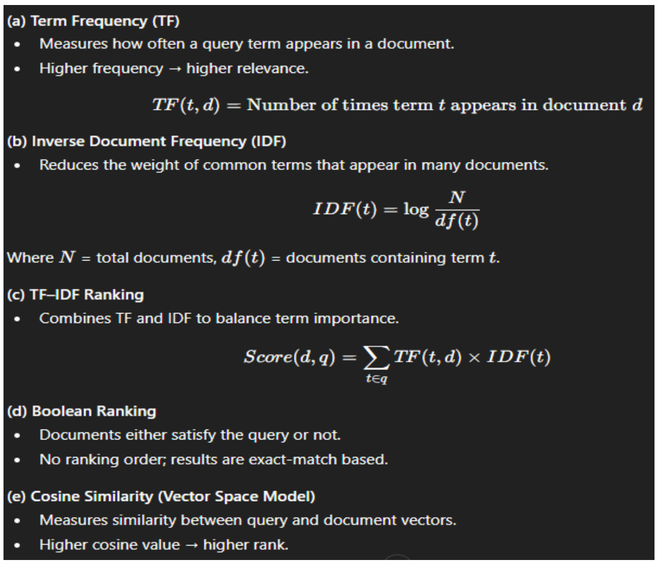
Link-based ranking

It is a search engine ranking approach that evaluates the **importance and authority of a web page based on hyperlinks** from other web pages. It is based on the idea that **a link from one page to another is a vote of confidence**.

1. **Basic Concept**
   - Web pages are treated as nodes and hyperlinks as edges in a graph.
   - A page with more incoming links (backlinks) is considered more important.
2. **Quality of Links Matters**
   - Not all links have equal value.
   - Links from authoritative or popular pages contribute more to ranking than links from less important pages.
3. **Recursive Importance**
   - Importance of a page depends on the importance of pages linking to it.
   - This creates a recursive scoring mechanism.
4. **PageRank Principle**
   - A famous link-based ranking method where a page's rank is distributed among the pages it links to.
   - Pages linked by many high-rank pages receive higher rank.
5. **Handling Link Spam**
   - Search engines reduce the effect of spam links using damping factors and filtering techniques.

Simple ranking functions

They assign a relevance score to documents based on basic statistical or heuristic measures. They are easy to compute and form the foundation of modern ranking models.


(a) Term Frequency (TF)
- Measures how often a query term appears in a document.
- Higher frequency → higher relevance.

$$TF(t,d) = \text{Number of times term } t \text{ appears in document } d$$

(b) Inverse Document Frequency (IDF)
- Reduces the weight of common terms that appear in many documents.

$$IDF(t) = \log \frac{N}{df(t)}$$

Where N = total documents, $df(t)$ = documents containing term t.

(c) TF–IDF Ranking
- Combines TF and IDF to balance term importance.

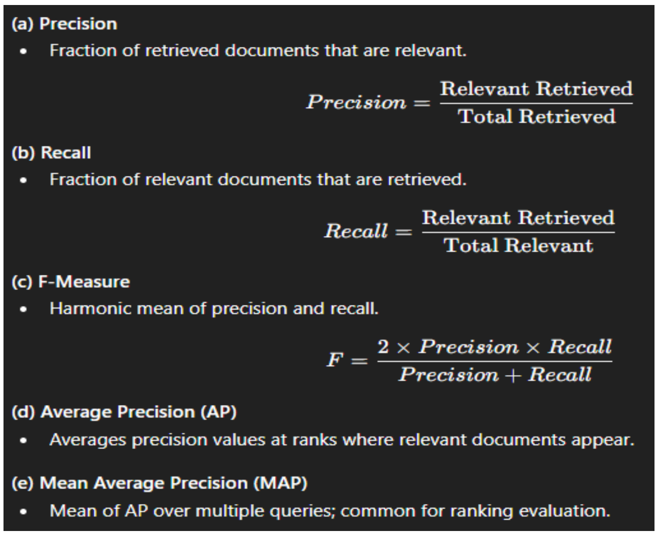
$$Score(d,q) = \sum_{t \in q} TF(t,d) \times IDF(t)$$

(d) Boolean Ranking
- Documents either satisfy the query or not.
- No ranking order; results are exact-match based.

(e) Cosine Similarity (Vector Space Model)
- Measures similarity between query and document vectors.
- Higher cosine value → higher rank.

2. Evaluation of Ranking Functions

Evaluation measures how effectively a ranking function retrieves relevant documents.


(a) Precision
- Fraction of retrieved documents that are relevant.

$$Precision = \frac{\text{Relevant Retrieved}}{\text{Total Retrieved}}$$

(b) Recall
- Fraction of relevant documents that are retrieved.

$$Recall = \frac{\text{Relevant Retrieved}}{\text{Total Relevant}}$$

(c) F-Measure
- Harmonic mean of precision and recall.

$$F = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

(d) Average Precision (AP)
- Averages precision values at ranks where relevant documents appear.

(e) Mean Average Precision (MAP)
- Mean of AP over multiple queries; common for ranking evaluation.

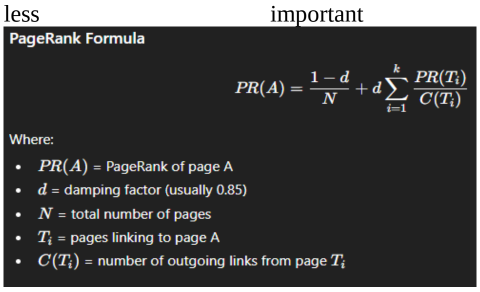
Page Ranking Algorithm (PageRank)

The **Page Ranking Algorithm (PageRank)** is a **link-based ranking algorithm** used by search engines to measure the **importance of web pages**. It was introduced to rank web pages based on the **structure of hyperlinks** rather than only text content. The basic idea is that a page is important if it is linked by other important pages. The web is modeled as a **directed graph**, where:

- Web pages are **nodes**
- Hyperlinks are **edges**

 Each link from one page to another is considered a **vote of importance**.

 Votes from important pages carry **more weight** than votes from less important pages.


PageRank Formula

$$PR(A) = \frac{1-d}{N} + d\sum_{i=1}^{k} \frac{PR(T_i)}{C(T_i)}$$

Where:
- $PR(A)$ = PageRank of page A
- d = damping factor (usually 0.85)
- N = total number of pages
- T_i = pages linking to page A
- $C(T_i)$ = number of outgoing links from page T_i

Damping Factor

- Represents the probability that a user **continues clicking links**.
- Typically set to **0.85**.
- Prevents rank sink problems and ensures convergence.

Working of PageRank Algorithm

1. **Initialization**
   - Assign equal rank to all pages initially.
2. **Rank Distribution**
   - Each page distributes its rank equally among its outgoing links.
3. **Rank Calculation**
   - PageRank of a page is updated using the ranks of pages linking to it.
4. **Iteration**
   - The process is repeated until PageRank values **converge**.

Key Features

- Recursive and iterative in nature
- Considers **quality and quantity of backlinks**
- Independent of query content
- Works well for large-scale web graphs

Advantages

- Identifies authoritative and trustworthy pages
- Reduces effect of keyword stuffing
- Improves overall search result quality

Limitations

- Can be manipulated using link spam
- Does not consider page content relevance
- Computationally expensive for very large web graphs.

Main Challenges of the World Wide Web

1. **Huge Size of the Web**
   - The web contains billions of web pages growing continuously.
   - Storing, indexing, and searching such massive data is difficult.
2. **Dynamic Nature of Web Content**
   - Web pages are frequently updated, deleted, or added.
   - Search engines must continuously crawl and update indexes.
3. **Heterogeneity of Data**
   - Web data exists in different formats: text, images, videos, audio, PDFs, etc.
   - Makes uniform processing and ranking difficult.
4. **Unstructured and Semi-Structured Data**
   - Most web content lacks a fixed structure (HTML pages).
   - Extracting meaningful information is challenging.
5. **Duplicate and Near-Duplicate Pages**
   - Same content appears on multiple URLs.
   - Increases storage and reduces result quality.
6. **Spam and Low-Quality Content**
   - Keyword stuffing, link spam, and fake pages try to manipulate rankings.
   - Requires spam detection algorithms.
7. **Multilingual Content**
   - Web pages exist in multiple languages.
   - Requires language detection and translation support.
8. **Scalability and Performance**
   - Search engines must serve millions of queries per second.
   - Requires highly scalable architectures.
9. **Freshness and Timeliness**
   - Users expect up-to-date information.
   - Maintaining freshness is computationally expensive.

Factors That Influence a Webpage's PageRank Score

1. **Number of Incoming Links (Backlinks)**
   - More incoming links generally increase PageRank.
2. **Quality of Incoming Links**
   - Links from high-ranked or authoritative pages carry more weight.
3. **Outgoing Links from Linking Pages**
   - If a page has many outgoing links, its PageRank is divided among them.
4. **Recursive Importance**
   - PageRank depends on the PageRank of linking pages, not just link count.
5. **Damping Factor**
   - Controls probability of random surfing (usually 0.85).
   - Prevents rank sinks and ensures stability.
6. **Internal Link Structure**
   - Well-structured internal links help distribute PageRank efficiently.
7. **Link Position and Context**
   - Contextual links are more valuable than footer or sidebar links.
8. **Handling of Dead Ends and Rank Sinks**
   - Pages with no outgoing links affect rank flow and are adjusted algorithmically.
9. **Spam and Manipulation Control**
   - Spam links are discounted to maintain ranking quality.

Challenges Involved in Web Search Engines

1. **Scalability**
   - Handling billions of documents and millions of queries simultaneously.
2. **Efficient Crawling**
   - Deciding which pages to crawl, how often, and avoiding overload on servers.
3. **Index Maintenance**
   - Updating indexes regularly to reflect changes in web content.
4. **Relevance Ranking**
   - Determining which documents best satisfy a user query.
5. **Web Spam and Manipulation**
   - Detecting keyword spam, link farms, and fake content.
6. **Query Ambiguity**
   - Same query can have different meanings for different users.
7. **Response Time (Latency)**
   - Users expect results in milliseconds.
8. **Multimedia and Non-Text Search**
   - Searching images, videos, and audio effectively is complex.
9. **Personalization and Privacy**
   - Balancing personalized results with user privacy concerns.

IR U6

Spoken Language Audio Retrieval

Spoken Language Audio Retrieval refers to the process of retrieving relevant audio documents that contain human speech, using information retrieval techniques. In this type of MIR (Multimedia Information Retrieval), the audio is first converted into a searchable text representation or acoustic features so that users can search using keywords or spoken queries.

Steps in Spoken Audio Retrieval

(a) Speech Recognition (ASR)

- Automatic Speech Recognition (ASR) converts speech audio into text.
- ASR output is used for indexing.

(b) Indexing

- The transcribed text is indexed using
  - **Inverted indexes**,
  - **TF–IDF**,
  - **Keyword-based indexing**.
- This allows fast matching with user queries.

(c) Phonetic Indexing (Alternative Approach)

- Instead of full words, the audio is converted into **phoneme sequences**.
- Helps handle accents, noise, and mispronunciation more effectively.

(d) Audio Segmentation

- Long audios are divided into smaller meaningful units:
  - Sentences
  - Conversational turns
  - Speaker segments
- Improves retrieval accuracy.

(e) Feature Extraction

- MFCC (Mel Frequency Cepstral Coefficients) and other speech features help identify speech patterns.
- Used during speech recognition and matching.

(f) Matching and Ranking

- Query terms are matched with indexed transcripts.
- Results ranked using:
  - Term frequency
  - ASR confidence score
  - Acoustic similarity

4. Applications

- Call center conversation monitoring
- Lecture and meeting search
- Podcast indexing
- Digital libraries
- Voice-based assistants (e.g., Siri, Alexa)

5. Challenges

- Background noise
- Speaker variability (age, gender, accent)
- Multilingual speech
- ASR errors reduce retrieval accuracy

Non-Speech Audio Retrieval

Non-Speech Audio Retrieval refers to retrieving audio files that do **not** contain human speech. These sounds include **music, animal sounds, environmental noises, alarms, machine sounds**, and many other acoustic signals. Unlike spoken language retrieval, this system focuses purely on **acoustic features** because text-based IR techniques cannot be directly applied.

2. Types of Non-Speech Audio

- **Music** (songs, instrument tones)
- **Environmental sounds** (rain, thunder, wind)
- **Biological sounds** (bird calls, animal noises)
- **Mechanical/Industrial sounds** (engine noise, alarms)
- **Sound effects** (footsteps, explosions)

3. Retrieval Process

(a) Feature Extraction (Core Step)

Non-speech audio is represented using numerical features that describe sound characteristics:

- **Pitch & Frequency**
- **Timbre**
- **Rhythm & tempo**
- **Energy & loudness**
- **MFCC (Mel-Frequency Cepstral Coefficients)**
- **Zero Crossing Rate**
- **Spectral Centroid, Spectral Flux**

These features allow the system to uniquely represent each sound.

(b) Feature Vector Representation

Extracted features are stored as **feature vectors**, which help in comparing and matching sounds.

(c) Query Types

1. **Keyword Search** – e.g., “rain sound”, “dog barking”
2. **Query by Example** – user uploads a sample audio
3. **Humming-based Search** – used in music information retrieval

(d) Similarity Matching

The system compares the query’s feature vector with database vectors using:

- **Euclidean Distance**
- **Cosine Similarity**
- **Dynamic Time Warping (DTW) for music**

Higher similarity → higher ranking in results.

(e) Classification & Clustering

- **Classification** (SVM, Neural Networks, k-NN) → identifies sound type
- **Clustering** (k-means, hierarchical) → groups similar sounds
  - Example: nature sounds cluster, machine sounds cluster

4. Applications

- **Music Information Retrieval (MIR)**
- **Environmental sound monitoring** (forests, traffic, weather)
- **Smart surveillance systems** (detect alarms, gunshots)
- **IoT-based devices** (sound-triggered automation)
- **Digital libraries of sound effects** (movies, games)
- **Healthcare** (cough sound analysis, heart sound detection)

5. Challenges

- **Overlapping sounds** (traffic + people + music)
- **High background noise**
- **Difficulty describing sounds using text**
- **Large intra-class variation** (dog bark varies across dogs)
- **Ambiguity in user queries**

Graph Retrieval – 9 Marks (Structured Answer)

Graph Retrieval refers to retrieving information that is stored and represented in the form of **graph structures**. A graph contains:

- **Nodes (vertices)** → represent entities
- **Edges** → represent relationships between entities

In many modern IR applications, data naturally forms complex networks such as **social networks, knowledge graphs, biological networks, and web graphs**. Graph retrieval focuses on locating relevant nodes, edges, or subgraph patterns based on a user query.

2. Graph Data Representation

Graph data is typically stored using the following structures:

(a) Adjacency List

Stores each node along with a list of connected nodes. Efficient for sparse graphs.

(b) Adjacency Matrix

Represents graph connections in matrix form. Suitable for dense graphs.

(c) Graph Databases

Specialized systems like **Neo4j, OrientDB, Amazon Neptune** provide fast graph queries using languages like Cypher and Gremlin.

3. Types of Graph Queries

Users can express queries in several forms:

- **Node queries** – find nodes with certain properties
- **Edge queries** – retrieve relationships
- **Path queries** – find shortest or specific paths
- **Subgraph queries** – match a smaller graph pattern inside a larger graph
- **Neighborhood queries** – e.g., “find friends-of-friends within 2 hops”

Example: Searching for all nodes connected to “A” within two levels.

4. Techniques Used in Graph Retrieval

(a) Graph Traversal Algorithms

- **BFS (Breadth First Search)** – level-wise exploration
- **DFS (Depth First Search)** – depth-based exploration

(b) Shortest Path Algorithms

- **Dijkstra’s Algorithm**
- **Bellman–Ford**

These are used for path-based queries.

(c) Subgraph Matching / Graph Isomorphism

- Used to check if a query graph structure matches part of the stored graph.
- Computationally expensive (often NP-Complete).

(d) Graph Indexing

Indexes like graph fingerprints, node labels, or structural signatures are stored to speed up retrieval.

(e) Ranking Techniques

Ranking of graphs or nodes is done using:

- **PageRank** (importance of nodes)
- **Betweenness Centrality** (nodes connecting communities)
- **Closeness Centrality**
- **Semantic similarity** (in knowledge graphs)

5. Applications of Graph Retrieval

Graph retrieval is widely used in modern IR systems:

- **Social Network Analysis**
- **Fraud Detection**
- **Recommendation Systems**
- **Biological Networks**
- **Knowledge Graph Search**
- **Web Structure Mining**

6. Challenges in Graph Retrieval

- **NP-completeness** of subgraph matching
- **Large graph size** (millions of nodes)
- **Dynamic updates** (social networks change frequently)
- **Noise and missing data** in real-world networks
- **High computational cost** for complex queries

Imagery Retrieval (Image Retrieval)

Imagery Retrieval, also known as **Image Retrieval**, refers to the process of retrieving relevant images from a large image database using different types of user queries. It is a key component of **Multimedia Information Retrieval (MIR)**.

2. Types of Image Queries

(a) Text-Based Image Retrieval (TBIR)

- User provides keywords such as “sunset”, “mountain”, “car”.

- System uses **captions, metadata, tags, or annotations**.
- Modern systems use automatic **image captioning models** to generate metadata.

(b) Content-Based Image Retrieval (CBIR)

- Users provide an example image or sketch.
- Retrieval is based on **visual features**, not text.

(c) Feature-Based Queries

- Search is performed using extracted image features stored as numeric values.

3. Content-Based Image Retrieval (CBIR) Process

(a) Feature Extraction

Images are converted into feature vectors using:

- **Color Features** → histograms, color moments
- **Texture Features** → Gabor filters, LBP
- **Shape Features** → edges, contours
- **Deep Learning Features** → CNN embeddings (ResNet, VGG)

These features capture the visual characteristics of the image.

(b) Indexing

Extracted features are stored in feature databases for fast retrieval.

(c) Similarity Matching

Similarity between query image and stored images is measured using:

- **Euclidean Distance**
- **Cosine Similarity**
- **Neural network–based similarity**

Higher similarity = higher ranking.

(d) Ranking

Images are ranked based on:

- Feature similarity
- Relevance feedback
- Semantic classification using deep learning

4. Applications of Imagery Retrieval

1. **Medical Image Search** (CT, MRI comparison)
2. **Biometric Identification** (face, fingerprint retrieval)
3. **E-commerce Visual Search** (search by image in shopping apps)
4. **Satellite Image Analysis** (land use, weather detection)
5. **Security and Surveillance** (object or person retrieval)
6. **Digital Art and Multimedia Archives**

5. Challenges

- **High image variation** in scale, rotation, and background
- **Lighting and viewpoint changes** affect feature extraction
- **Semantic gap** between low-level features and human interpretation
- **Large database size** requiring efficient indexing

Video Retrieval

Video Retrieval refers to the process of retrieving relevant videos or specific video segments based on user queries. A video is a complex multimedia object because it contains **three modalities**:**Visual frames, Audio tracks**, **Textual metadata**.Effective retrieval requires analyzing all these components together.

2. Components Used in Video Retrieval

(a) Visual Component

- Frames contain color, texture, shape, and object information.
- Keyframes are used to represent scenes efficiently.

(b) Audio Component

- Speech transcripts (ASR)
- Background sounds and music
- Helps identify events like cheering, alarms, or conversations.

(c) Text Metadata

Titles, Descriptions, Tags, Subtitles and closed captions

These support **text-based video search**.

3. Video Retrieval Process

(a) Video Segmentation

- Video is divided into **shots** and **scenes**.
- Makes retrieval faster and more meaningful.

(b) Keyframe Extraction

- Representative frames selected from each scene.
- Used for indexing and comparison.

(c) Feature Extraction

Visual Features:

- Color histogram
- Texture patterns
- Edge/shape features
- Deep learning features (CNN embeddings)

Audio Features:

- MFCC
- Speech transcripts
- Sound patterns

Text Features:

- Metadata
- Captions/subtitles

(f) Ranking of Results

Ranking is done using **multimodal fusion**, which combines:

- Visual similarity
- Audio similarity
- Text similarity

4. Applications

- YouTube and OTT platform video search
- Surveillance and CCTV video analysis
- Sports highlights detection
- Movie and advertisement retrieval
- Medical and surgical video archives
- Educational video and lecture indexing

5. Challenges

- Very large storage requirements
- High computational cost
- Frequent scene transitions
- Noisy or unclear audio
- Object occlusion and fast motion
- "Semantic gap" between raw features and meaning

Recommender System

A **Recommender System** is an information filtering technique used to suggest items (movies, products, songs, courses, etc.) to users based on their preferences and behavior. It helps reduce information overload by predicting what a user is likely to prefer. Recommender systems mainly use three approaches:

1. **Collaborative Filtering**
2. **Content-Based Recommendation**
3. **Knowledge-Based Recommendation**

1. Collaborative Filtering

Collaborative Filtering (CF) is one of the most widely used recommendation techniques. It makes predictions about a user's preferences based on the **preferences of similar users** or **similar items**.
The core assumption is:
"Users who behaved similarly in the past will have similar preferences in the future."

2. Types of Collaborative Filtering

(a) User-Based Collaborative Filtering

- Recommends items liked or rated highly by those similar users.
  Example:
  If User A and User B rate similar movies, movies liked by B can be recommended to A.

(b) Item-Based Collaborative Filtering

- If a user likes a particular item, the system finds other items that are similar.
  Example:
  If a user likes Item X, then Item Y and Z (similar to X) are recommended.

4. Advantages

- Does not require item metadata or descriptions.
- Captures real community behavior.
- Can recommend diverse items beyond the user's history.

5. Limitations

- **Cold Start Problem:** New users/items lack enough ratings.
- **Sparsity:** Most users rate very few items, causing sparse rating matrices.
- **Scalability:** Heavy computation for large datasets.

6. Applications

- Netflix movie recommendation
- Amazon "Customers who bought this also bought…"
- Spotify music recommendations

2. Content-Based Recommendation

Content-Based Recommendation recommends items that are **similar to items the user has liked in the past**. It uses item features such as genre, category, keywords, or product descriptions.

2. How Content-Based Systems Work

(a) Item Feature Extraction

Features are extracted depending on the domain:

- Movies → genre, actors, director
- Products → specifications, brand, description
- Text → TF-IDF, keywords, embeddings

(b) User Profile Creation

- A model of the user's preferences is built.
- Example: The system concludes that the user prefers *action + sci-fi* movies.

(c) Matching & Recommendation

- Compare item features with the user profile using similarity measures.
- Items with the highest similarity scores are recommended.
- Common measure: **Cosine similarity**

3. Advantages

- Works well even when there are no ratings from other users.
- Provides personalized, user-specific recommendations.
- No cold start for items (only user needs history).

4. Limitations

- Limited to the user’s past behavior (no exploration).
- Cannot introduce new or unexpected items.
- Item feature extraction may be complex (especially for images/videos).

5. Applications

- YouTube video recommendations based on watch history
- Job recommendations on LinkedIn
- News article recommendations

3. Knowledge-Based Recommendation

Knowledge-Based Recommendation suggests items using **explicit knowledge** about user requirements and item characteristics. Unlike collaborative or content-based systems, it does **not** depend on past ratings or user history.

Types of Knowledge-Based Systems

(i) Constraint-Based Recommendation

- Works by defining clear user constraints.
- Example: Real estate, cars, electronics.

(ii) Case-Based Recommendation

- Compares current user needs with past user cases.
- Example: Suggesting laptops based on similar previous purchases.

4. Advantages

- Works well for **rare or expensive items** (houses, cars, travel packages).
- No cold-start problem.
- Highly accurate when user requirements are clear.

5. Limitations

- Requires a large amount of domain knowledge.
- Difficult and costly to build.
- Complex to maintain rules and cases.

6. Applications

- Travel recommendation systems
- Automobile selection tools
- Real estate property recommendation
- Medical decision-support systems

1. Extracting Data from Text

Extracting data from text refers to converting **unstructured text** into **structured, machine-readable information**. Since most web content, documents, and reports exist in natural language, text extraction is a crucial step in information retrieval and data mining.

2. Need for Text Extraction

- Most digital data is in textual form.
- Organizations need structured information for analytics.
- Helps in automation, decision-making, and search optimization.

3. Text Extraction Process

(a) Text Preprocessing

- Tokenization
- Stop-word removal
- Stemming/Lemmatization
- Normalization

Preprocessing prepares raw text for analysis.

(b) Feature Extraction

- **Bag of Words (BoW)**
- **TF-IDF**
- **Word Embeddings** (Word2Vec, GloVe)
- **Sentence Embeddings**

These convert text into numeric features.

(c) Named Entity Recognition (NER)

Extracts entities such as people, places, organizations, dates, currencies.

(d) Information Extraction (IE)

- Identifies relations, events, facts
- Example: From “Amazon acquired Whole Foods,” extract relation: *Acquisition*

(e) Text Classification

- Categorizes documents (spam/ham, sports/politics)

(f) Text Summarization

- Extractive or abstractive summaries
- Converts large content into concise information

4. Techniques Used

- Natural Language Processing (NLP)
- Regular expressions
- Machine Learning (SVM, Naive Bayes)
- Deep Learning (LSTM, Transformers)

5. Applications

- Search engines
- Chatbots
- Resume parsing
- Medical report extraction
- Social media analytics

6. Challenges

- Ambiguity in natural language
- Synonyms and context understanding
- Spelling variations
- Noisy or informal text (slang, social media)

Semantic Web

The Semantic Web is an extension of the existing web where information is given **well-defined meaning** so that both humans and machines can understand it. It allows computers to interpret data logically, enabling intelligent search and automation.

Goals of the Semantic Web

- Make web content machine-understandable
- Improve accuracy of search engines
- Enable data integration across websites
- Support intelligent applications and agents

3. Key Technologies in Semantic Web

(a) RDF (Resource Description Framework)

- Represents data as **subject–predicate–object** triples
- Forms the foundation of machine-readable data

(b) OWL (Web Ontology Language)

- Provides vocabulary to define classes, properties, relationships
- Allows reasoning and inference

(c) SPARQL

- Query language for RDF data
- Similar to SQL but used for semantic web databases

(d) Ontologies

- Structured knowledge models that define concepts and relationships
- Example: Medical ontology describing diseases, symptoms, treatments.

4. How Semantic Web Works

- Websites publish data with semantic markup (RDF/OWL).
- Intelligent agents crawl the web and interpret meaning.
- SPARQL engines retrieve relevant structured data.

Example: Searching for "movies directed by Nolan" retrieves precise results because data is semantically connected.

5. Applications

- Intelligent search engines
- Personal assistants (Siri, Alexa use semantic knowledge graphs)
- Healthcare decision systems
- E-commerce product linking
- Academic knowledge graphs (Google Scholar)

6. Challenges

- High effort required to create ontologies
- Integration issues with existing unstructured web
- Scalability and reasoning cost
- Lack of universal standards for all domains

Collecting and Integrating Specialized Information on the Web

The web contains vast and diverse data spread across multiple sites, formats, and structures. Collecting and integrating specialized information refers to gathering **domain-specific data** (medical, financial, scientific, educational, etc.) and combining it into a unified, usable format.

2. Need for Specialized Information Integration

- Data is scattered across multiple sources
- Formats vary: HTML, XML, JSON, PDFs
- Essential for analytics, decision-making, and research
- Enables domain applications (e.g., medicine, agriculture, finance)

3. Process of Collecting & Integrating Information

(a) Web Crawling & Scraping

- Automated agents collect data from web pages
- Extract text, tables, product details, reviews, etc.

(b) Data Cleaning

- Remove noise, ads, duplicates, inconsistencies
- Standardize units, formats, and naming

(c) Data Transformation

- Convert various formats into a unified structure
- Use schemas, ontologies, APIs

(d) Data Integration

- Combine information from multiple sources
- Resolve conflicts, duplicates, and data mismatches
- Use integration techniques:
  - Schema matching
  - Entity resolution
  - Data fusion

(e) Storage in Structured Repositories

- Databases
- Data warehouses
- Knowledge graphs

4. Tools & Technologies

- APIs (REST, GraphQL)
- Web scrapers (BeautifulSoup, Scrapy)
- Semantic web technologies (RDF, OWL)
- ETL tools (Extract-Transform-Load)

5. Applications

- **Healthcare:** Integrating patient data, research papers
- **Finance:** Stock data aggregation
- **E-commerce:** Price comparison engines
- **Education:** Course and research database integration
- **Travel:** Flight and hotel aggregator websites

6. Challenges

- Inconsistent data formats
- Dynamic web pages and frequent updates
- Legal issues (copyright, scraping policies)
- Ambiguity in merging information
- Large computational costs

Steps to Build a Recommender System with an Example

1⃣ Data Collection Phase

This phase involves **gathering data** required for making recommendations.

- Collects **user data** such as ratings, clicks, likes, search history, and purchase history.
- Collects **item data** such as category, description, tags, and features.
- Data can be:
  - **Explicit** (ratings, reviews)
  - **Implicit** (clicks, views, time spent)

☞ This phase forms the **foundation** of the recommender system.

2⃣ Data Preprocessing Phase

Raw data is cleaned and transformed to make it usable.

- Removes missing, noisy, or inconsistent data.
- Normalizes ratings and user behavior data.
- Converts data into suitable formats (user–item matrix).
- Handles sparsity and cold-start issues partially.

☞ Improves data quality and system accuracy.

3⃣ Recommendation / Model Building Phase

In this phase, algorithms are applied to generate recommendations.

- Uses techniques such as:
  - **Collaborative Filtering**
  - **Content-Based Filtering**
  - **Hybrid Methods**
- Computes similarity between users or items.
- Predicts ratings or relevance scores.

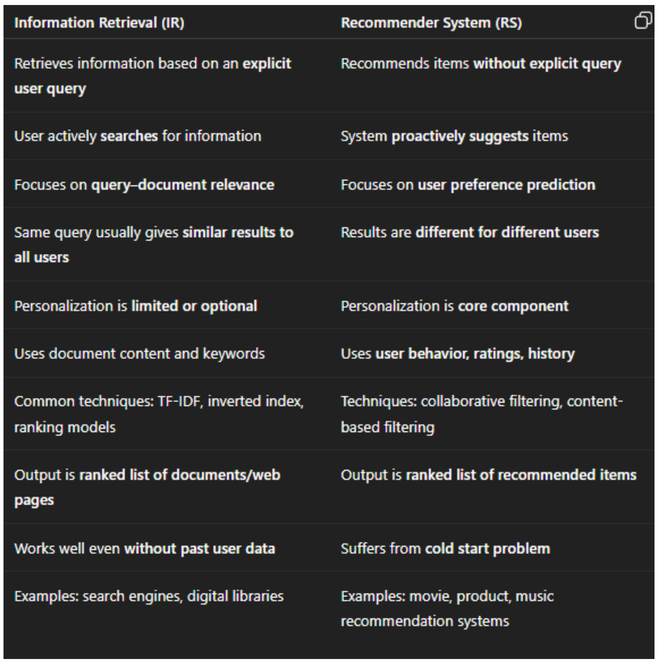
☞ This phase decides **what to recommend**.

4⃣ Recommendation Delivery & Feedback Phase

The final recommendations are presented to users and feedback is collected.

- Displays ranked list of recommended items.
- Collects user feedback (clicks, ratings, likes).
- Feedback is fed back into the system for improvement.
- Supports personalization and real-time updates.

☞ Enables **continuous learning** and system improvement.



Information Retrieval (IR)	Recommender System (RS)
Retrieves information based on an **explicit user query**	Recommends items **without explicit query**
User actively **searches** for information	System **proactively suggests** items
Focuses on **query–document relevance**	Focuses on **user preference prediction**
Same query usually gives **similar results to all users**	Results are **different for different users**
Personalization is **limited or optional**	Personalization is **core component**
Uses document content and keywords	Uses **user behavior, ratings, history**
Common techniques: TF-IDF, inverted index, ranking models	Techniques: collaborative filtering, content-based filtering
Output is **ranked list of documents/web pages**	Output is **ranked list of recommended items**
Works well even **without past user data**	Suffers from **cold start problem**
Examples: search engines, digital libraries	Examples: movie, product, music recommendation systems

Different Evaluation Metrics for Recommender Systems

Step 1⃣ Problem Definition

- Clearly define **what to recommend** and **to whom**.
- Decide the goal: increase sales, engagement, or personalization.

Example:
Recommend **movies** to users on an online streaming platform.

Step 2⃣ Data Collection

- Collect **user data**: ratings, clicks, watch history.
- Collect **item data**: movie genre, cast, director, release year.
- Data can be explicit (ratings) or implicit (views).

Example:
Users rate movies from 1–5 stars and their watch history is stored.

Step 3⃣ Data Preprocessing

- Clean noisy or missing data.
- Normalize ratings.
- Build a **user–item interaction matrix**.

Example:
Convert movie ratings into a matrix:
Users × Movies with rating values.

Step 4⃣ Choose Recommendation Technique

Select the appropriate method:

- **Content-Based Filtering** – based on item features
- **Collaborative Filtering** – based on similar users/items
- **Hybrid Approach** – combination of both

Example:
Use **collaborative filtering** to find users with similar movie tastes.

Step 5⃣ Model Building

- Compute similarity (cosine similarity / Pearson correlation).
- Predict unknown ratings or relevance scores.

Example:
If User A and User B like similar movies, recommend movies liked by B to A.

Step 6⃣ Generate Recommendations

- Rank items based on predicted scores.
- Select top-N items.

Example:
Recommend **Top 5 movies** that the user has not watched.

Step 7⃣ Recommendation Presentation

- Display recommendations in a user-friendly manner.
- Can be personalized sections like “Recommended for You”.

Example:
Homepage shows “Movies You May Like”.

Step 8⃣ Feedback Collection

- Collect user feedback such as clicks, ratings, likes, skips.
- Feedback can be implicit or explicit.

Example:
User watches or skips recommended movies.

Step 9⃣ Model Evaluation

- Measure performance using metrics like:
  - Precision
  - Recall
  - RMSE
  - F-measure

Example:
Check how many recommended movies were actually watched.

Step 🔟 Continuous Improvement

- Update model regularly with new data.
- Handle cold-start users and new items.

Example:
As the user watches more movies, recommendations become more accurate.