- **Basic Concepts of IR**: The study of finding relevant, unstructured information (like text) that satisfies a user's need; it's the foundation for all other topics.
- **Data Retrieval & Information Retrieval**: The contrast between finding exact matches in structured databases (Data Retrieval) and finding relevant information in unstructured text (IR); this distinction necessitates all the specialized techniques in the syllabus.
- **Text mining and IR relation**: IR finds the relevant documents, and text mining then extracts structured patterns and new knowledge from them; IR is often the first step for text mining.
- **IR system block diagram**: A high-level flowchart of an IR system (indexing -> query processing -> matching -> ranking); it shows how components like Indexing and Automatic Text Analysis connect to form a complete system.
- **Automatic Text Analysis: Luhn's ideas**: The foundational concept that a word's frequency and distribution determine its significance; this is the core idea behind Index Term Weighting.
- **Conflation Algorithm**: A method (like stemming) to group word variants (e.g., 'computing', 'computer') to a single root ('comput'); it's used during Automatic Text Analysis to improve search recall.
- **Indexing and Index Term Weighting**: The process of creating a fast lookup structure (index) and assigning importance scores (weights like TF-IDF) to terms; this uses Conflation's output to enable efficient and relevant searching.
- **Probabilistic Indexing**: A model that ranks documents based on the probability of their relevance to a query; it's a more formal, statistical alternative to standard Index Term Weighting.
- **Automatic Classification**: The task of assigning documents to predefined categories (e.g., "Sports"); it's a supervised learning application that uses IR techniques to organise information.
- **Measures of Association**: The concept of using statistical formulas to measure the similarity between two items (documents or terms); this is the mathematical basis for all Clustering Techniques.
- **Different Matching Coefficients**: Specific formulas (e.g., Jaccard, Dice) that calculate a similarity score; they are the practical tools used to implement Measures of Association in clustering algorithms.
- **Cluster Hypothesis**: The fundamental assumption that closely associated documents are relevant to the same user queries; this is the theoretical justification for using Clustering Techniques to improve IR.
- **Clustering Techniques**: Algorithms that group similar documents together without predefined labels; they rely on the Cluster Hypothesis and Matching Coefficients to find structure in data.
- **Rocchio’s Algorithm**: A method for query refinement based on user feedback; it effectively clusters relevant documents around a query to improve its performance.

- **Single pass algorithm**: A fast, efficient clustering algorithm that processes each document only once; it's one of the main Clustering Techniques, useful for very large datasets.
- **Single Link algorithm**: A hierarchical clustering algorithm that merges clusters based on the single closest pair of documents; it's another key Clustering Technique that produces a detailed hierarchy of relationships.

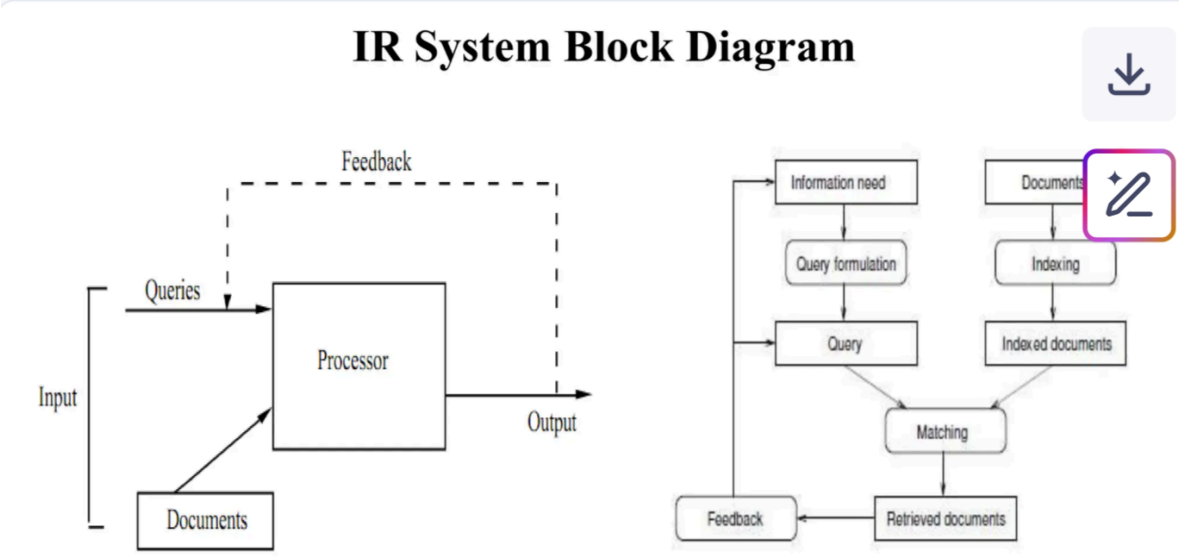

Fig : Typical IR System (Black Box)

Fig : Information Retrieval (IR) Process

1. Core Concepts: Data vs. Information Retrieval

- **Data Retrieval (DR)**: This is about finding data that matches a query **perfectly**. The data is **structured**, like in a database (e.g., SQL). Think of it as asking a very specific question and getting a precise, unambiguous answer.
    - **Example Query**: `SELECT name FROM employees WHERE employee_id = 101;`
    - **Goal**: Find all objects that satisfy a clearly defined condition. It's a one-step process of matching.
- **Information Retrieval (IR)**: This is about finding information that is **relevant** to a user's need from a collection of **unstructured** data (like text documents, web pages, emails). The user's need is often vague, and the results are ranked by how relevant the system thinks they are. 🧐
    - **Example Query**: "latest advancements in artificial intelligence"

- **Goal**: Find information that is *about* a subject. It's a process of interpretation, matching, and ranking. It deals with uncertainty.

---

2. The Information Retrieval Process

An IR system's main job is to take a user's query, search through a large collection of documents, and return a ranked list of documents that are most relevant to that query.

- **Block Diagram of an IR System**:
- **The Process Explained**:
    1. **Document Collection**: The raw source of information (e.g., websites, articles, emails).
    2. **Text Analysis (Indexing)**: This is the crucial pre-processing step. Documents aren't stored as-is. The system analyses the text to create a structured representation, called an **index**. This involves:
        - Parsing documents to extract text.
        - Removing **stop words** (common words like 'the', 'is', 'a').
        - Applying a **conflation (stemming) algorithm** to group word variations.
        - Assigning **weights** to index terms to signify their importance.
    3. **User Query**: The user enters their information need as a query (e.g., a few keywords).
    4. **Query Processing**: The user's query undergoes the same text analysis as the documents (stop word removal, stemming) to ensure it can be matched against the index.
    5. **Searching & Matching**: The processed query is compared to the document index to find documents that contain the query terms.
    6. **Ranking**: This is the magic of IR. Instead of just returning all matching documents, the system uses a ranking algorithm (like TF-IDF) to score each document based on its relevance to the query.
    7. **Ranked Results**: The user is presented with a sorted list of documents, with the most relevant ones at the top. The user's interaction (e.g., clicking a link) can be used as **relevance feedback** to improve future results.

---

3. Automatic Text Analysis & Representation

Conflation (Stemming) Algorithm

Conflation is the process of grouping different variations of a word into a single root form, called a **stem**. This is done to ensure a search for "computer" also finds documents with "computing" or "computes". The most famous example is the **Porter Stemmer**.

- **How it Works (Example with Porter Stemmer)**: The algorithm applies a series of rules to strip suffixes from words.
    - **Word**: `retrievals`
    - Rule 1 (Plurals): `retrievals` -> `retrieval` (removes 's')
    - Rule 2 (Common Endings): `retrieval` -> `retriev` (removes 'al')
    - The final stem is `retriev`. Now, `retrieving`, `retrieved`, and `retrieval` all conflate to the same stem `retriev`.
- **Generating a Document Representative**:
    - **Original Document Text**: "Information retrieval systems are used for retrieving relevant information."
    - **Tokenise & Remove Stop Words**: `[Information, retrieval, systems, used, retrieving, relevant, information]`
    - **Apply Conflation Algorithm**:
        - `Information` -> `inform`
        - `retrieval`, `retrieving` -> `retriev`
        - `systems` -> `system`
        - `used` -> `use`
        - `relevant` -> `relev`
    - **Document Representative (Bag of Stems)**: The document is now represented by the set of stems it contains, often with their frequencies: `{inform:2, retriev:2, system:1, use:1, relev:1}`. This compact representation is what gets stored in the index.
- **Advantages**:
    - **Reduces Index Size**: Maps many word forms to one stem, making the index smaller and faster to search.
    - **Increases Recall**: Broadens the search. A query for "retrieve" will match documents containing "retrieving" or "retrieved," which is usually what the user wants.
- **Disadvantages**:
    - **Over-stemming**: Incorrectly conflating words that mean different things (e.g., `university` and `universe` might both become `univers`). This hurts precision.
    - **Under-stemming**: Failing to conflate words that should be grouped (e.g., `adhere` and `adhesion` might remain separate).

Index Term Weighting

Why is it used? It's used to quantify the importance of a term within a document and across the entire collection. Not all words are equally important. A word that appears many times in one document but rarely in others is a very good indicator of that document's topic.

- The most common weighting scheme is **TF-IDF (Term Frequency - Inverse Document Frequency)**.

    - **Term Frequency (TF)**: How often a term appears in a document. A higher TF suggests the term is important *to that document*.
        - TF(t,d)=(Number of times term t appears in document d)
    - **Inverse Document Frequency (IDF)**: How rare a term is across the entire document collection. A higher IDF means the term is rare and thus more discriminative.
        - IDF(t,D)=logNumber of documents with term tTotal number of documents ∣D∣
    - **TF-IDF Weight**: The final weight is the product of the two.
        - W(t,d)=TF(t,d)×IDF(t,D)
    - This scheme gives a high weight to terms that are frequent in a specific document but rare in the overall collection, making them excellent keywords.

---

4. Clustering in Information Retrieval

Clustering is the unsupervised task of grouping similar documents together. It's based on the **Cluster Hypothesis**: "Documents that cluster together tend to be relevant to the same queries."

Rocchio's Algorithm

Rocchio's algorithm is primarily used for **relevance feedback**. It modifies a query vector based on user feedback about which documents are relevant or non-relevant. It aims to move the query closer to the good documents and away from the bad ones.

- **The Goal**: To produce an "optimal" query vector qopt that maximizes the separation between relevant and non-relevant documents.
- **The Formula**: The modified query vector qm is calculated from the original query qo, the set of relevant documents Dr, and the set of non-relevant documents Dnr.
  $qm=\alpha qo+\beta|Dr|1dj\in DrΣdj-\gamma|Dnr|1dk\in DnrΣdk$
- **Explanation**:
    - α,β,γ are weights that control the influence of each component.
    - It starts with the **original query** (qo).
    - It **adds** the vectors of documents the user marked as **relevant** (Σdj). This is like adding good keywords to the search.
    - It **subtracts** the vectors of documents the user marked as **non-relevant** (Σdk). This is like penalizing misleading keywords.

Single Pass Clustering Algorithm

This is a simple and fast clustering method. It goes through the documents one by one and decides whether to add the current document to an existing cluster or create a new one.

- **Algorithm Steps**:
    - Make the first document the centroid of a new cluster.
    - For the next document, calculate its similarity to the centroids of *all existing clusters*.
    - If the similarity to the closest centroid is above a pre-defined **threshold (T)**, add the document to that cluster and update the cluster's centroid.
    - If no centroid is similar enough (i.e., similarity is below T), create a new cluster with this document as its centroid.
    - Repeat for all documents.
- **Example**:
    - Documents: D1, D2, D3, D4. Threshold T = 0.7.
    - **D1 arrives**: Create Cluster 1 (C1). Centroid of C1 = D1.
    - **D2 arrives**: Calculate Sim(D2, C1). Let's say it's 0.8. Since 0.8 > 0.7, add D2 to C1. Update C1's centroid.
    - **D3 arrives**: Calculate Sim(D3, C1). Let's say it's 0.5. Since 0.5 < 0.7, D3 is not similar enough. Create a new cluster, C2. Centroid of C2 = D3.
    - **D4 arrives**: Calculate Sim(D4, C1) = 0.6 and Sim(D4, C2) = 0.9. The max similarity is 0.9 (with C2), which is > 0.7. Add D4 to C2 and update its centroid.
    - **Result**: Two clusters: {D1, D2} and {D3, D4}.

Single Link Clustering Algorithm

This is a type of **hierarchical agglomerative clustering (HAC)**. It starts with each document in its own cluster and iteratively merges the two most similar clusters until only one remains.

- **Algorithm Steps**:
    1. Start by placing each document in its own individual cluster.
    2. Compute a similarity matrix between all pairs of clusters. The similarity between two clusters is defined as the **maximum similarity** between any two documents in those clusters (one from each).
        - $\text{Sim}(C_i, C_j) = \max_{d_x \in C_i, d_y \in C_j} \text{Sim}(d_x, d_y)$
    3. Find the two most similar clusters in the matrix and merge them.
    4. Update the similarity matrix to reflect the merge.
    5. Repeat steps 3 and 4 until only one cluster (containing all documents) is left.
- **Example**:
    1. We have documents A, B, C, D.
    2. **Initial Clusters**: {A}, {B}, {C}, {D}.
    3. **Similarity Matrix**: Let's say the most similar pair is (A, B) with similarity 0.9.
    4. **Merge**: Merge {A} and {B} to form a new cluster {A, B}.
    5. **Update**: Now we have clusters {A, B}, {C}, {D}. We recalculate similarities. The similarity between {A, B} and {C} is `max(Sim(A,C), Sim(B,C))`.

6. **Repeat**: Let's say the next most similar pair is ({A, B}, D). Merge them to get {A, B, D}.
7. The process continues, building a hierarchy that can be visualized as a **dendrogram**.

Comparison: Single Pass vs. Single Link

Feature	Single Pass Algorithm	Single Link Algorithm
Speed	Very fast, linear time complexity O(N·k) where N is docs, k is clusters.	Slower, at least quadratic time complexity O(N2).
Document Order	**Order-dependent**. The final clusters depend on the order in which documents are processed.	**Order-independent**. The result is always the same regardless of document order.
Structure	Produces flat, non-overlapping clusters.	Produces a hierarchy of clusters (a dendrogram).
Advantages	✅ Simple and extremely efficient for large datasets.	✅ Produces a more detailed and stable clustering structure.
Disadvantages	❌ Result is not unique (order-dependent). ❌ Can produce poor-quality clusters if the threshold is badly chosen.	❌ Computationally expensive, not suitable for very large datasets. ❌ Suffers from "chaining effect" (can merge two clusters if just one pair of documents is close, even if the rest are far apart).

Best Used When	You need to cluster a massive, streaming dataset quickly and an approximate result is acceptable.	You have a small-to-medium sized dataset and need a detailed, deterministic hierarchical structure.

---

5. Measuring Performance & Association

Evaluation Metrics: Precision & Recall

These are the two most fundamental metrics for evaluating the quality of search results.

- **Precision**: Of the documents you retrieved, what fraction were actually relevant? It measures the **exactness** of the search.
    - Precision=∣Total documents retrieved∣∣Relevant documents retrieved∣=TP+FPTP
    - **High Precision**: The results returned are very likely to be good. You don't see a lot of junk.
- **Recall**: Of all the relevant documents that exist in the collection, what fraction did you manage to retrieve? It measures the **completeness** of the search.
    - Recall=∣Total relevant documents in collection∣∣Relevant documents retrieved∣=TP+FNTP
    - **High Recall**: You found most of the relevant stuff. You didn't miss much.
- **The Trade-off**: There is a natural trade-off. To get higher recall, you can retrieve more documents, but this will likely lower your precision (more junk). To get higher precision, you can be more selective, but you might miss some relevant documents (lower recall).
- **Other Similar Terms**:
    - **F-Measure (or F1-Score)**: The harmonic mean of precision and recall, providing a single score that balances both.
        - F1=2⋅Precision+RecallPrecision⋅Recall

Measures of Association & Matching Coefficients

These are used to measure the similarity between two objects (e.g., two documents). Documents are often represented as binary vectors where a '1' means a term is present and '0' means it's absent.

Let's consider two documents, X and Y, represented by binary vectors.

- a = Number of terms where X=1 and Y=1 (present in both)
- b = Number of terms where X=1 and Y=0 (present in X, not Y)
- c = Number of terms where X=0 and Y=1 (present in Y, not X)

- d = Number of terms where X=0 and Y=0 (present in neither)

Here are three common matching coefficients:

1. **Simple Matching Coefficient (SMC)**: Measures both presences and absences.
    - **Formula**: $SMC = \frac{a+b}{c+d}$
    - **Explanation**: "What fraction of attributes match, including matching absences?"
    - **Example**: Let vocabulary be {red, blue, green, yellow}.
        - Doc X: {red, green} -> [1, 0, 1, 0]
        - Doc Y: {red, yellow} -> [1, 0, 0, 1]
        - a=1 (red), b=1 (green), c=1 (yellow), d=1 (blue).
        - $SMC = \frac{1+1}{1+1+1+1} = \frac{2}{4} = 0.5$
2. **Jaccard Coefficient**: Ignores shared absences (d). It's better for sparse data (like text, where most words are absent).
    - **Formula**: $J = \frac{a}{b+c}$
    - **Explanation**: "Of the terms present in at least one document, what fraction are present in both?"
    - **Example (using above data)**:
        - a=1, b=1, c=1.
        - $J = \frac{1}{1+1} = \frac{1}{2} \approx 0.33$
3. **Dice Coefficient**: Similar to Jaccard but gives more weight to shared presences (a).
    - **Formula**: $Dice = \frac{2a}{b+c}$
    - **Explanation**: It's the ratio of twice the shared terms to the total number of terms present in both documents.
    - **Example (using above data)**:
        - a=1, b=1, c=1.
        - $Dice = \frac{2 \cdot 1}{1+1+2 \cdot 1} = \frac{2}{4} = 0.5$