Content-Based Recommender Systems

UNIT II: Basic Components of Content-Based Systems, Pre-processing and Feature Extraction, Learning User Profiles and Filtering, Nearest Neighbor Classification.

Introduction

Comparison with Collaborative Systems:

- Collaborative systems use **correlations in user ratings**, while content-based systems rely on **item attributes and user preferences.**

- Content-based systems do not require other **users' ratings**, making them suitable for scenarios where **user-specific data is sufficient**.

Core Functionality of Content-Based Systems:

- Match **users to items** similar to what they have liked in the **past** using **item attributes rather than ratings correlations.**

- Leverage two main data sources:
  - **Item Descriptions:** **Content-centric attributes**, such as keywords, genre, and manufacturer.
  - **User Profiles:** Built from **explicit (ratings)** or **implicit (actions)** feedback, or specified keywords of interest.

Advantages of Content-Based Systems:

- Effective in **cold-start scenarios** for items (new items with no user ratings).
- Suitable for **text-rich and unstructured domains**, like web pages and product descriptions.
- Personalized recommendations based **solely on the user's past interactions.**

Disadvantages of Content-Based Systems:

- Limited **diversity and novelty in recommendations**, as items are often too similar to past preferences.
- Struggles with the **cold-start problem for new users**, as it requires prior user interaction data.
- Recommendations may **lack surprise or creativity**.

Applications of Content-Based Systems:

- Widely used in domains with **text-rich data**, such as:
  - **Web page recommendations** based on browsing history.
  - **E-commerce recommendations** using product descriptions and relational attributes (e.g., price, manufacturer).

Structured vs. Unstructured Representations:

- Attributes can be unstructured (text-based) or structured (e.g., numerical, relational).
- Both can be combined into a single structured representation for recommendation tasks.

Relation to Knowledge-Based Systems:

- Both systems use **content attributes for recommendations**.
- **Differences:**
  - **Knowledge-based** systems allow **explicit specification** of user requirements and interactive interfaces.
  - **Content-based systems** rely on **past user behavior** using learning-based approaches.

Hybrid Systems:

- Combine **content-based and collaborative methods** to address the limitations of each approach.
- Provide a **unified framework** for **leveraging both** learning-based and interactive aspects of recommendations.

Basic Components of Content-Based Systems

General Characteristics:

- Content-based systems **convert unstructured data** into **standardized descriptions**, often **keyword-based vector-space representations**.
- These systems largely operate in the **text domain** and are commonly used in applications like **news recommendation systems.**
- **Text classification and regression modeling** are the primary tools for **content-based recommenders.**

Main Components of Content-Based Systems

Preprocessing and Feature Extraction:

- Extract features from various sources (e.g., web pages, product descriptions, news articles).
- Convert features into a keyword-based vector-space representation.
- Effective feature extraction is critical and domain-specific.

Content-Based Learning of User Profiles:

- Construct **user-specific models** based on past interactions (e.g., ratings, purchases).
- Leverage explicit feedback (e.g., ratings) or implicit feedback (e.g., activity logs) to build training data.

- Use classification (for categorical feedback) or regression (for numerical feedback) to relate user interests to item attributes.

- **Filtering and Recommendation:**
  - Use the **learned model** to generate **recommendations for users in real-time**.
  - Efficiency is crucial since **predictions need to be performed quickly.**

Item Representation (Movie Example): Each movie is described using its attributes, such as **title**, **genre**, **director**, **actors**, and **synopsis.** These attributes are broken down into **keywords.**

Example:

- **Movie 1 (The Matrix):**
    - Keywords: [Sci-fi, Action, Cyberpunk, AI, Virtual Reality]
- **Movie 2 (Inception):**
    - Keywords: [Sci-fi, Thriller, Dream, Heist, Mind-bending]

These keywords form a **vector space**, with each keyword representing a dimension in this space.

Vector Representation: Each movie is transformed into a **vector**.
Example for the keywords across movies:
Keywords: [Sci-fi, Action, Cyberpunk, AI, Virtual Reality, Thriller, Dream, Heist, Mind-bending]

- **Movie 1 (The Matrix):**
    - Vector: [1, 1, 1, 1, 1, 0, 0, 0, 0]
      (*Presence of Sci-fi, Action, Cyberpunk, AI, Virtual Reality; absence of others.*)
- **Movie 2 (Inception):**
    - Vector: [1, 0, 0, 0, 0, 1, 1, 1, 1]
      (*Presence of Sci-fi, Thriller, Dream, Heist, Mind-bending; absence of others.*)

User Preferences as a Vector: Suppose a user has previously liked movies with the keywords **Sci-fi**, **Action**, and **Thriller**.

- User preference vector: [1, 1, 0, 0, 0, 1, 0, 0, 0]

Similarity Computation: Use similarity metrics like **cosine similarity** to measure how similar each movie is to the user preferences.

- **Cosine Similarity Formula:**

$$\text{Similarity} = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}||||\mathbf{B}||}$$

  Where:

  - A and B are the vectors for a movie and user preferences.
  - $||\mathbf{A}||$ is the magnitude of vector A.

- Compute similarity between **User Preferences** and **Movie 1 (The Matrix):**

$$\text{Similarity} = \frac{(1 \cdot 1 + 1 \cdot 1 + 0 \cdot 0)}{\sqrt{1^2 + 1^2 + 0^2} \cdot \sqrt{1^2 + 1^2 + 1^2 + 1^2 + 1^2}} = \frac{2}{\sqrt{2} \cdot \sqrt{5}}$$

- Compute similarity between **User Preferences** and **Movie 2 (Inception):** Follow the same method.

Based on similarity scores, the system recommends the movie with the highest similarity.

If **The Matrix** has a higher similarity score, it will be recommended to the user over **Inception**.

This recommendation is entirely based on the **keywords** extracted from item descriptions and matched to the user preferences.

Model Utilization:

- Classification models are commonly used in the learning phase.
- Content-based systems can use these models as black-box components, focusing on how they relate user profiles to item attributes.

Additional Notes:

- The learning phase is often based on well-known classification or regression techniques.

Preprocessing and Feature Extraction

- **General Overview:**
  - The first phase in content-based systems is extracting discriminative features to represent items effectively.
  - Discriminative features are predictive of user interests and vary based on the application (e.g., product recommendation vs. web pages).

Feature Extraction:

- Convert item descriptions into keywords or structured representations for processing.
- Common approaches:
  - **Bag of Keywords**: Extract text descriptions and convert them into keyword-based vectors.
  - **Structured Representation**: Use numerical (e.g., price) or categorical attributes (e.g., color, genre).

- **Feature Weighting:**
  - Assign different levels of importance to attributes.
  - Approaches:
  - **Domain-Specific Knowledge:** Heuristics to decide keyword weights (e.g., title and main actor in movies).
  - **Automated Methods:** Learn feature weights algorithmically (closely related to feature selection).

Examples of Feature Extraction in Various Applications:

- **Product Recommendation (e.g., IMDb):**
  - Attributes include movie synopsis, director, actors, and genre.
  - Example: For the movie *Shrek*, attributes like "ogre," "princess," and "magical creatures" form the keyword set.
  - Importance of features (e.g., actors vs. synopsis) can be determined using:
    - **Domain-Specific Knowledge:** Weight features like title or primary actor higher.
    - **Automated Methods:** Use feature weighting or selection algorithms.

Web Page Recommendation:

- Extract structured data from HTML fields like title, meta-data, and body.
- Weight fields differently; for instance, title and meta-data are given higher importance than the body.
- Handle irrelevant blocks (e.g., ads or disclaimers) using:
  - **Tree-Matching Algorithms:** Learn document layouts and extract main content blocks.
  - **Classification Methods:** Identify main content versus irrelevant blocks.

Music Recommendation (e.g., Pandora):

- Features are extracted from the **Music Genome Project**, including attributes like:
  - “Trance roots,” “synth riffs,” “tonal harmonies,” “straight drum beats.”
- Users create a "station" by specifying one track, and similar songs are recommended.
- User feedback (likes/dislikes) refines recommendations over time.
- Keywords or structured attributes (e.g., genres or beats) form the basis for recommendation.

Scenario:

Recommend products based on a user's preference for specific features like "brand," "price range," and "category."

1. Data Collection

Dataset: A collection of products with descriptions.

Example:

```makefile
Product 1: "Apple MacBook Air"
Category: Laptop
Brand: Apple
Price: High
Description: Lightweight, powerful, and perfect for professionals.

Product 2: "Dell Inspiron 15"
Category: Laptop
Brand: Dell
Price: Medium
Description: Affordable, reliable performance, and suitable for students.
```

2. Preprocessing

Goal: Clean the raw data to extract useful features for the recommendation.

Steps:

1. **Stop-Word Removal:**
    - Remove common words like "and," "for," "the," etc., from product descriptions.
2. **Stemming:**
    - Convert words to their root forms:
        - "Lightweight" → "Light"
        - "Professionals" → "Professional"
3. **Phrase Extraction:**
    - Identify meaningful phrases:
        - "Lightweight laptop"
        - "Reliable performance"

Processed Data:

- **Product 1 (Apple MacBook Air):**
  Keywords: Laptop, Apple, High, Lightweight, Powerful, Professional.
- **Product 2 (Dell Inspiron 15):**
  Keywords: Laptop, Dell, Medium, Affordable, Reliable, Student.

3. Feature Representation

Goal: Represent products as vectors in a keyword-based vector space.

Keywords Across Products:

csharp Copy Edit

```csharp
[Laptop, Apple, High, Lightweight, Powerful, Professional, Dell, Medium, Affordable, Relia
```

Vector Representation:

- **Product 1 (Apple MacBook Air):**

  csharp Copy Edit

  ```csharp
  [1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0]
  ```

- **Product 2 (Dell Inspiron 15):**

  csharp Copy Edit

  ```csharp
  [1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1]
  ```

4. User Profile

User Preference:

The user prefers high-end laptops with lightweight designs and professional usage.

User Profile Vector:

```csharp
[Laptop, Apple, High, Lightweight, Professional]
[1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0]
```

5. Similarity Computation

Goal: Compare user preferences with product vectors to find the best match.

Use **Cosine Similarity:**

$$\text{Similarity} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \cdot \|\mathbf{B}\|}$$

Calculation for Product 1 (Apple MacBook Air):

- **User Profile:** `[1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0]`
- **Product 1 Vector:** `[1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0]`

$$\text{Similarity} = \frac{(1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1)}{\sqrt{5} \cdot \sqrt{6}} = \frac{5}{\sqrt{30}} \approx 0.912$$

Calculation for Product 2 (Dell Inspiron 15):

- **User Profile:** `[1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0]`
- **Product 2 Vector:** `[1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1]`

$$\text{Similarity} = \frac{(1 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 + 0 \cdot 1)}{\sqrt{5} \cdot \sqrt{6}} = \frac{1}{\sqrt{30}} \approx 0.183$$

6. Recommendation

Similarity Scores:

- **Apple MacBook Air:** 0.912
- **Dell Inspiron 15:** 0.183

Recommendation:

The system recommends **Apple MacBook Air** as it aligns better with the user's preferences.

1. Data Collection

- **Dataset:** A collection of movies with descriptions. Example:

makefile   Copy   Edit

```makefile
Movie 1: "The Matrix"
Genre: Sci-fi, Action
Director: Lana Wachowski
Synopsis: A hacker discovers the truth about his reality and fights AI machines.

Movie 2: "Inception"
Genre: Sci-fi, Thriller
Director: Christopher Nolan
Synopsis: A thief who steals corporate secrets through dreams undertakes a final missi
```

2. Preprocessing

Goal: Clean the raw data to extract useful features for the recommendation.

- **Steps:**
  - **Stop-Word Removal:** Remove common words like "the," "and," etc., from synopses.
  - **Stemming:** Convert words to their root forms. For instance:
    - "fights" → "fight"
    - "machines" → "machine"
  - **Phrase Extraction:** Identify significant phrases like "AI machines" or "corporate secrets."

Processed Data:

- Movie 1 (The Matrix):
  - Keywords: Sci-fi, Action, Hacker, Reality, Fight, AI, Machine.
- Movie 2 (Inception):
  - Keywords: Sci-fi, Thriller, Thief, Dream, Corporate, Secret, Mission.

3. Feature Representation

Goal: Represent movies as vectors in a keyword-based vector space.

Keywords Across Movies:

```bash
[Sci-fi, Action, Hacker, Reality, Fight, AI, Machine, Thriller, Thief, Dream, Corporate, S
```

Vector Representation:

- Movie 1 (The Matrix):

  ```csharp
  [1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0]
  ```

- Movie 2 (Inception):

  ```csharp
  [1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1]
  ```

4. User Profile

User Preference:

- The user likes Sci-fi movies with themes related to AI and hacking.

User Profile Vector:

```csharp
[Sci-fi, Action, Hacker, AI]
[1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

5. Similarity Computation

Goal: Compare user preferences with movie vectors to find the best match.

- Use **Cosine Similarity**:

$$\text{Similarity} = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}|| \, ||\mathbf{B}||}$$

Calculation for Movie 1 (The Matrix):

- User Profile: `[1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]`
- Movie 1 Vector: `[1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0]`

$$\text{Similarity} = \frac{(1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 1)}{\sqrt{4} \cdot \sqrt{7}} = \frac{4}{\sqrt{28}} \approx 0.755$$

Calculation for Movie 2 (Inception):

- User Profile: `[1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]`
- Movie 2 Vector: `[1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1]`

$$\text{Similarity} = \frac{(1 \cdot 1 + 0 \cdot 1 + 0 \cdot 1 + 0 \cdot 1)}{\sqrt{4} \cdot \sqrt{7}} = \frac{1}{\sqrt{28}} \approx 0.189$$

6. Recommendation

- Based on similarity scores:
  - **The Matrix:** Similarity = 0.755
  - **Inception:** Similarity = 0.189
- **Recommendation:** The system recommends **The Matrix** as it aligns better with the user's preferences.

Key Insights:

- **Domain-Specific Techniques:** Feature extraction and weighting are tailored to specific applications (e.g., movies vs. music vs. web pages).
- **Combination of Attributes:** Both unstructured (text-based) and structured (numerical/categorical) attributes can be combined for robust recommendations.
- **Knowledge-Based Systems:** In cases like Pandora, the initial track specification resembles **knowledge-based systems**, but user feedback transitions the approach to content-based recommendations.

Feature Representation and Cleaning

Importance:

- Transform unstructured data (e.g., product descriptions, web pages) into a cleaned and structured format suitable for analysis.
- Represent text as **bags of words** for further processing.

Key Steps in the Cleaning Process:

1. Stop-Word Removal:

- Remove **high-frequency, non-informative words** (e.g., "a," "an," "the") that are not specific to the item.
- Common stop-words include articles, prepositions, conjunctions, and pronouns.
- Pre-defined stop-word lists are available for various languages.

2. Stemming:

- Consolidate variations of words into their root forms.
  - Example: "hoping" → "hop," "machines" → "machine."
- **Caution:** Stemming can sometimes lead to loss of meaning (e.g., "hop" as a word itself).

3. Phrase Extraction:

- Identify significant word combinations that occur frequently.
  - Example: "hot dog" has a distinct meaning compared to "hot" and "dog."
- Methods: Use manually defined dictionaries or automated algorithms.

Vector-Space Representation:

- **Definition:**
  - After cleaning, text data is converted into a vector of terms, where each term is associated with its frequency.
- **Challenges:**
  - Commonly occurring words are less discriminative and may bias results.
  - Use weighting methods to emphasize more meaningful terms.

Weighting with TF-IDF:

- **Term Frequency (TF):**
  - Frequency of a term in a document.
- **Inverse Document Frequency (IDF):**
  - Weights terms inversely proportional to their occurrence in the entire dataset.
- Formula: $\mathbf{IDF}=\boldsymbol{log}\left(\frac{\boldsymbol{n}}{\boldsymbol{n_i}}\right)$
- $\boldsymbol{n}$: Total number of documents.
- $\boldsymbol{n_i}$: Number of documents containing the term.

TF-IDF Formula:

- Combines term frequency and inverse document frequency
- **h(x_i)=f(x_i).IDF**
- **f(x_i):** Damping function (optional) to reduce the influence of high-frequency terms.
- Examples: $\sqrt{x_i}$ **or** $log(x_i)$

Applications of TF-IDF:

- Improves recommendation quality by prioritizing discriminative terms.
- Reduces the influence of frequently occurring but unimportant words (e.g., "common" vs. "rare" terms).

Example: Feature Representation and Cleaning with TF-IDF

Step 1: Dataset

Suppose we have the following movie descriptions:

1. **Movie 1: The Matrix**
    - **Description:** "A hacker discovers the truth about his reality and fights AI machines."
2. **Movie 2: Inception**
    - **Description:** "A thief who steals corporate secrets through dreams undertakes a final mission."

Step 2: Preprocessing

Stop-Word Removal:

- Remove common words like "a," "the," "about," "and," etc.

Processed Descriptions:

1. Movie 1: "hacker discovers truth reality fights AI machines"
2. Movie 2: "thief steals corporate secrets dreams undertakes final mission"

Stemming:

- Convert words to their root forms.

Stemmed Descriptions:

1. Movie 1: "hack discover truth real fight AI machine"
2. Movie 2: "thief steal corporate secret dream undertake final mission"

Step 3: Bag of Words

Unique Terms Across Both Movies:

php

Copy Edit

```php
[hack, discover, truth, real, fight, AI, machine, thief, steal, corporate, secret, dream,
```

Vector Representation (Term Frequencies - TF):

Term	Movie 1 TF	Movie 2 TF
hack	1	0
discover	1	0
truth	1	0
real	1	0
fight	1	0
AI	1	0
machine	1	0
thief	0	1
steal	0	1
corporate	0	1
secret	0	1
dream	0	1
undertake	0	1
final	0	1
mission	0	1

Step 4: Compute TF-IDF

IDF Formula:

$$\text{IDF} = \log\left(\frac{N}{n_i}\right)$$

Where:

- $N = 2$ (total number of movies)
- n_i = number of movies containing the term.

IDF Values:

Term	n_i	IDF
hack	1	$\log(2/1) = 0.693$
discover	1	0.693
truth	1	0.693
real	1	0.693
fight	1	0.693
AI	1	0.693
machine	1	0.693
thief	1	0.693
steal	1	0.693
corporate	1	0.693
secret	1	0.693
dream	1	0.693
undertake	1	0.693
final	1	0.693
mission	1	0.693

TF-IDF Scores:

TF-IDF = $\text{TF} \times \text{IDF}$

Term	Movie 1 TF-IDF	Movie 2 TF-IDF
hack	$1 \times 0.693 = 0.693$	$0 \times 0.693 = 0.000$
discover	$1 \times 0.693 = 0.693$	$0 \times 0.693 = 0.000$
truth	$1 \times 0.693 = 0.693$	$0 \times 0.693 = 0.000$
real	$1 \times 0.693 = 0.693$	$0 \times 0.693 = 0.000$
fight	$1 \times 0.693 = 0.693$	$0 \times 0.693 = 0.000$
AI	$1 \times 0.693 = 0.693$	$0 \times 0.693 = 0.000$
machine	$1 \times 0.693 = 0.693$	$0 \times 0.693 = 0.000$
thief	$0 \times 0.693 = 0.000$	$1 \times 0.693 = 0.693$
steal	$0 \times 0.693 = 0.000$	$1 \times 0.693 = 0.693$
corporate	$0 \times 0.693 = 0.000$	$1 \times 0.693 = 0.693$
secret	$0 \times 0.693 = 0.000$	$1 \times 0.693 = 0.693$
dream	$0 \times 0.693 = 0.000$	$1 \times 0.693 = 0.693$
undertake	$0 \times 0.693 = 0.000$	$1 \times 0.693 = 0.693$
final	$0 \times 0.693 = 0.000$	$1 \times 0.693 = 0.693$
mission	$0 \times 0.693 = 0.000$	$1 \times 0.693 = 0.693$

Step 5: Use in Recommendations

- **Goal:** Find which movie is most similar to a user profile.
- For example, if a user prefers terms like "AI" and "fight," Movie 1 will be recommended as it has higher TF-IDF scores for these terms.

Collecting User Likes and Dislikes

- **Purpose:**
  - Gather user **preferences** (likes and dislikes) during the **offline phase** to generate recommendations during the online phase.
  - Combine the user's preferences with content data to create predictions for the **active user** (the user interacting with the system at any given time).

Forms of User Feedback:

Ratings:

- Users explicitly specify ratings for items.
- Types of ratings:
  - **Binary (e.g., like/dislike).**
  - **Interval-based (e.g., 1–5 stars).**
  - **Ordinal or real-valued ratings.**
- The type of rating influences the model used for learning user profiles.

Implicit Feedback:

- Captures user actions such as:
  - **Positive preferences:** Buying, browsing, or clicking an item.
- Does not typically include negative preferences.

Text Opinions:

- Users express preferences through **textual descriptions** (e.g., reviews or comments).
- Preferences are extracted using techniques like:
  - **Opinion Mining.**
  - **Sentiment Analysis.**

Cases:

- Users specify examples or cases of items they are interested in.
- These cases are used:
  - As implicit feedback for models like **nearest neighbor** or **Rocchio classifiers**.
  - In **case-based recommender systems** where domain knowledge is used to find matches.

Special Notes:

- **Hybrid Approaches:**
  - Some systems blend knowledge-based and content-based approaches:
    - Example: **Spotify**:
      - Starts as a knowledge-based system (user specifies an initial case, such as a favorite music album).
      - Transitions to a content-based and collaborative system using user feedback.
- **Output Representation:**
  - User feedback (likes/dislikes) is ultimately converted into:
    - **Unary, binary, interval-based, or real ratings.**
  - These ratings serve as a **class label** or dependent variable for learning purposes.

Supervised Feature Selection and Weighting

Objective of Feature Selection & Weighting

- Ensure only **informative words** are retained in the vector-space representation.
- Reduce the risk of **overfitting** by filtering out noisy features.
- Improve recommendation accuracy, especially when **limited data** is available.
- Used in recommender systems to limit **keywords between 50 and 300**.

Two Aspects of Feature Selection

- **Feature Selection:** Removing irrelevant words.
- **Feature Weighting:** Assigning different importance to words.

Examples of unsupervised selection methods:

  - **Stop-word removal**
  - **Inverse Document Frequency (IDF)**

- **Supervised selection methods:**
  - Consider user ratings to rank features.

Methods for Feature Selection & Weighting

1. Gini Index (For Categorical Ratings)

- Measures how well a word discriminates between rating values.
- **Formula :** **Gini (w) =1-$\sum_{i=1}^{t} p_i(w)^2$**
- **Key Properties:**
  - **Lower Gini values → Higher discriminative power.**
  - If a word always corresponds to a **specific rating**, its Gini score is **0**.
  - If a word is **randomly distributed across ratings**, its Gini score is **1 - 1/t**.

2. Entropy (Information Theory-Based)

- Similar to Gini Index but based on information theory.
- **Formula:** $Entropy(w) = \sum_{i=1}^{t} p_i(w) \, log \, p_i(w)$
- **Key Properties:**
  - Lower entropy → More informative word.
  - Often yields results **similar to Gini Index** but is based on **mathematical principles**.

3. Chi-Square (χ^2) Statistic

- Tests if a word is **statistically significant** in predicting ratings.
- Compares **expected vs. observed word occurrences** in different rating categories.
- **Formula for a 2×2 contingency table:**

$$\chi^2 = \frac{(O_1 + O_2 + O_3 + O_4) \times (O_1 O_4 - O_2 O_3)^2}{(O_1 + O_2) \times (O_3 + O_4) \times (O_1 + O_3) \times (O_2 + O_4)}$$

- **Key Properties:**
  - **Higher χ^2 value → Stronger correlation** between word and rating.
  - If **expected = observed**, χ^2 = **0** (word is irrelevant).
  - Only the **top-k words with highest χ^2 scores** are retained.

4. Normalized Deviation (For Continuous Ratings)

- Measures how **word occurrences affect rating averages**.
- ***Formula:*** $Dev(w) = \frac{|\mu_+(w) - \mu_-(w)|}{\sigma}$
- **μ^+(w):** Average rating when word **is present**.
- **μ^-(w):** Average rating when word **is absent**.
- **σ:** Overall rating variance.
- **Key Properties :** Higher deviation → More discriminative word.
- Used when ratings have **many possible values** (e.g., continuous scores).

Feature Weighting (Soft Selection)

- Instead of removing words, assign **weights** based on informativeness.
- **Example Weighting Formula:**
- $g(w) = a - \mathrm{Gini}(w)$
  - **a:** Adjustable sensitivity parameter.
  - Smaller **a** → Higher sensitivity to Gini scores.
  - Multiplies weight with **TF-IDF** scores to refine recommendations.
- **Other weighting strategies:**
  - **Entropy-based weighting**
  - **Inverse Document Frequency (IDF)**
  - **Cross-validation** to fine-tune weights.

Key Takeaways

- **Feature selection** improves efficiency by removing noisy words.
- **Supervised methods use ratings** to find the most useful words.
- **Different selection techniques** apply based on **data type** (categorical vs. continuous ratings).
- **Feature weighting** fine-tunes importance **without hard removal**.
- **Hybrid approaches (Selection + Weighting)** yield the best recommendation performance.

Learning User Profiles and Filtering

User Profiles and Recommendation Learning:

- User profile learning is **similar to classification and regression modeling**.
- Ratings can be **discrete** (e.g., "thumbs up" or "thumbs down") → Similar to **text classification**.
- Ratings can be **numerical** → Similar to **regression modeling**.

Structured vs. Unstructured Learning:

- Learning methods can be applied to **both structured and unstructured data**.
- In this discussion, **text-based item descriptions** are assumed.

Training Dataset (DL) and Active User:

- **DL (Labeled Training Documents)**:
  - Contains item descriptions and ratings assigned by a **specific active user**.
  - These ratings form the **user profile**.

- **No collaborative filtering** → Ratings from other users are **not considered**.

- This approach builds **personalized** models instead of a **global** one.

Testing Dataset (DU) for Recommendations:

- **DU (Unlabeled Test Documents)**:
  - Contains item descriptions for **potential recommendations**.
  - These items have **not been rated** by the user yet.

- DU varies based on domain, e.g.:
  - **News recommendation** → DU contains **candidate news articles**.
  - **E-commerce** → DU contains **potential product suggestions**.

Recommendation Process Using the Model:

- **Training model (from DL) is applied to DU**.
- The system provides:
  - **Predicted ratings** for DU items.
  - **Ranked top-k recommendations**.

Comparison with Collaborative Filtering:

- Unlike **collaborative filtering** (e.g., matrix factorization):
  - **Each user has a separate model**.
  - No **cross-user data sharing**.
- This ensures **personalized** recommendations **without requiring** other users' preferences.

- **Relation to Text Classification & Regression:**
  - Problem structure is **similar to classification and regression modeling in NLP**.
  - Models used for classification can be adapted to **recommendation systems**.

Nearest Neighbor Classification

Definition and Similarity Function:

- Nearest neighbor classifier is a **simple and effective** classification technique.
- Uses a **similarity function** to compare documents.
- **Cosine similarity** is the most common measure in text-based classification:
- **Other similarity measures:** Euclidean distance, Manhattan distance for structured data.

$$Cosine(X,Y) = \frac{\sum_{i=1}^{d} x_i y_i}{\sqrt{\sum_{i=1}^{d} x_i^2} \cdot \sqrt{\sum_{i=1}^{d} y_i^2}}$$

Prediction Process:

- For each **document in DU (test set)**:
  - Find **k-nearest neighbors** from **DL (training set)** using cosine similarity.
  - Compute the **average rating** of the k-nearest neighbors.
  - Assign this **average rating** to the document in DU.
- **Categorical Ratings:** Uses a **majority vote** from the k-nearest neighbors.

Challenges: Computational Complexity:

- **Finding nearest neighbors is expensive**

  (complexity = $|DL| \times |DU|$)
- **Each document in DU requires comparisons** with all documents in DL.
- This makes the method computationally expensive for **large datasets**.

Optimization Using Clustering:

- **Solution:** Reduce the number of training documents using **clustering**.
- **Steps:**
  - Cluster **DL** into **p groups per rating value**.
  - Each group is represented as a **single aggregated document**.
  - Only **k-nearest clusters** are compared with test documents.
- **Speeds up classification** while maintaining accuracy.

Special Case: Prototype-Based Approach:

- **All documents of a rating value are combined into a single prototype vector**.
- Instead of finding k-nearest neighbors:
  - The **closest prototype** is selected.
  - The **rating of the prototype** is assigned to the test document.
- **Related to Rocchio Classification**, which incorporates relevance feedback.

A streaming platform wants to **predict whether a user will like a new movie** based on their past preferences. The system uses a **nearest neighbor approach** with cosine similarity.

Step 1: Dataset (Past User Ratings)

We have four movies with descriptions and **user ratings (0 = Dislike, 1 = Like)**.

Movie	Description	User Rating
Movie A	"Action-packed thrilling adventure"	**1 (Like)**
Movie B	"Slow and dramatic with deep storytelling"	**0 (Dislike)**
Movie C	"Exciting action with great special effects"	**1 (Like)**
Movie D	"A boring drama with weak character development"	**0 (Dislike)**

The new movie to be classified:

"An intense action film with amazing stunts"

Step 2: Convert Descriptions into Feature Vectors

To compare descriptions, we create a set of unique **keywords** from all descriptions:

arduino Copy Edit

```arduino
["action", "thrilling", "adventure", "slow", "dramatic", "deep", "storytelling",
 "exciting", "special", "effects", "boring", "drama", "weak", "character", "development",
 "intense", "film", "amazing", "stunts"]
```

Each movie is then represented as a **vector** based on the presence of these words.

Movie	Feature Vector Representation (Example)
Movie A	`[1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]`
Movie B	`[0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]`
Movie C	`[1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]`
Movie D	`[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0]`
New Movie	`[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1]`

Step 3: Compute Cosine Similarity

To find the most similar movies, we use **cosine similarity** between the new movie and each existing movie.

Formula for Cosine Similarity:

$$\text{Cosine Similarity} = \frac{\sum(A_i \times B_i)}{\sqrt{\sum A_i^2} \times \sqrt{\sum B_i^2}}$$

Where:

- A and B are movie vectors.
- Higher values indicate more similarity.

Step 4: Finding Nearest Neighbors

Movie	Cosine Similarity with New Movie	User Rating
Movie A	0.87	1 (Like)
Movie C	0.82	1 (Like)
Movie B	0.12	0 (Dislike)
Movie D	0.05	0 (Dislike)

Conclusion:

The new movie is **most similar to Movie A and Movie C**, both of which were **liked** by the user.

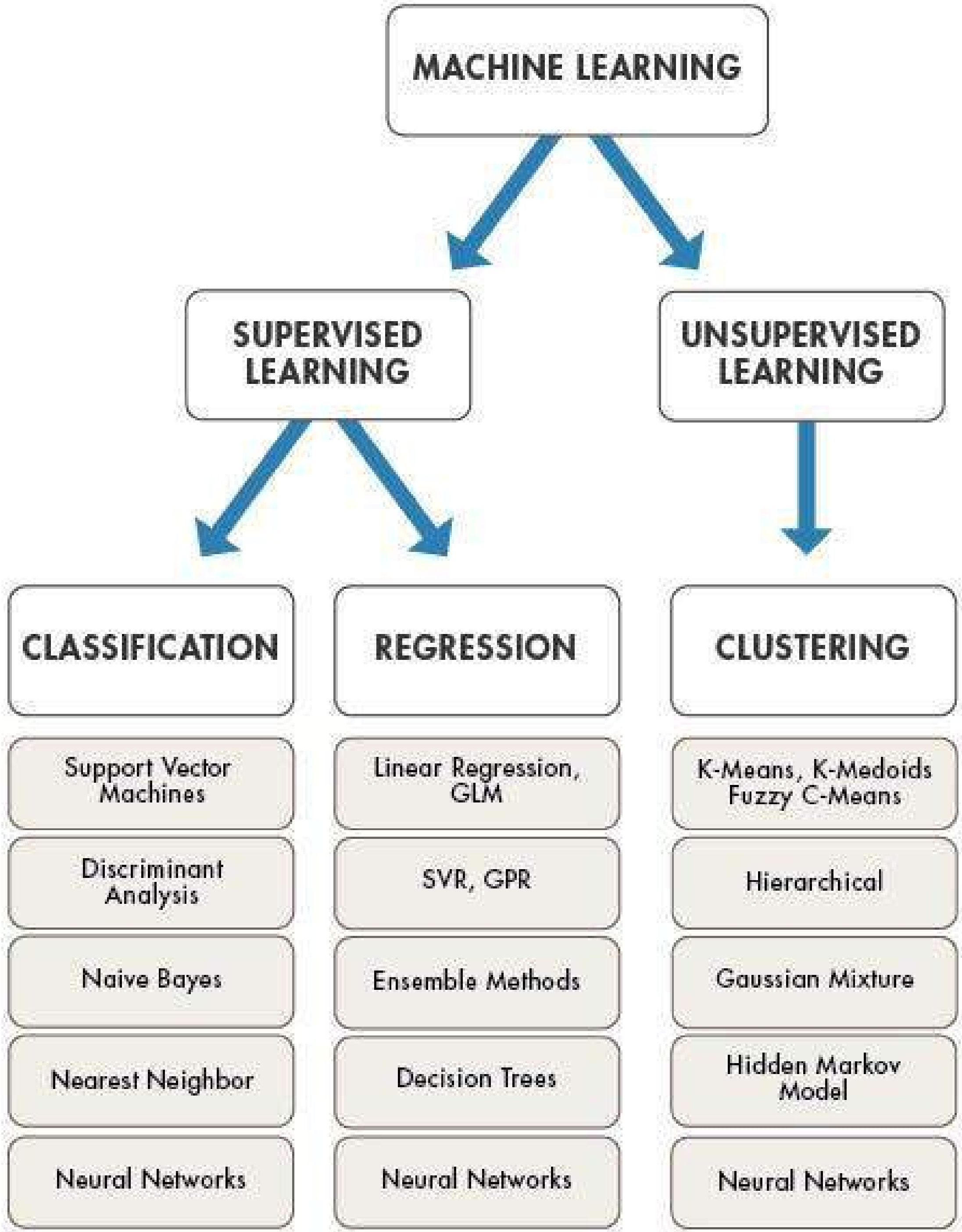
Step 5: Predict Rating Using k-Nearest Neighbors

- Choose **k = 2** (Top 2 most similar movies).
- **Movie A and Movie C** both have a rating of **1 (Like)**.
- **Majority voting:** Since both similar movies are liked, we predict **"Like" (1) for the new movie.**

Final Recommendation

🔷 Prediction: The user will like the new movie! ✅

🎬 Recommended Movie: "An intense action film with amazing stunts" is suggested to the user.



Algorithm	Regression, Classification
Linear Regression	Regression
Logistic Regression	Classification
Decision Trees	Both
Random Forests	Both
SVM	Both
KNN	Both
Gradient Boosting	Both
Naive Bayes	Classification