**Movie Recommendation System**

**YouTube Video for reference**

<https://youtu.be/1xtrIEwY_zY?si=oMtIErMtezVk__1->

**What are Recommender Systems:**

A recommender system is an algorithmic tool designed to suggest items to users based on their preferences and behaviors. They are widely used in various applications to enhance user experience by providing personalized recommendations.

**Types of Recommender Systems**

**Collaborative Filtering:** This approach relies on the preferences and behaviors of similar users. For example, if User A and User B have similar tastes in movies, a movie that User A likes will be recommended to User B.

**Content-Based Filtering:** This method recommends items based on the characteristics of the items themselves and a user's past preferences. For example, if a user has watched and liked several action movies, the system will recommend other action movies.

**Hybrid Recommender Systems:** These systems combine collaborative filtering, content-based filtering, and sometimes other methods to provide more accurate recommendations.

**Example:**

Netflix's Recommendation System:

Collaborative Filtering: Recommending movies and shows based on what similar users have liked.

Content-Based Filtering: Recommending movies and shows based on their genres, actors, etc., that the user has shown interest in.

Hybrid Approach: Combines both methods to improve the accuracy of recommendations.

**Applications:**

E-commerce: Amazon recommending products.

Streaming Services: Netflix recommending movies and TV shows.

Social-Media: Facebook recommending friends or pages.

Music Streaming: Spotify recommending songs and playlists.

News Websites: Google News recommending articles.

**Advantages:**

Personalization: Delivers a personalized experience to users.

Increased User Engagement: Keeps users engaged by showing content relevant to their interests.

Revenue Boost: Helps in upselling and cross-selling products.

Improved User Retention: Enhances the overall user experience, encouraging them to return.

Data Utilization: Makes effective use of user data to provide meaningful recommendations.

**Disadvantages:**

Cold Start Problem: Difficulties in making accurate recommendations for new users due to the lack of data.

Scalability: High computational cost for large datasets.

Privacy Concerns: Requires access to personal user data, which can raise privacy issues.

Overfitting: Risk of making recommendations too narrow, which might miss out on serendipitous discoveries.

Bias and Diversity Issues: Can reinforce existing user biases, reducing the diversity of recommendations.

**Project Flow:**

Data -> Pre-Processing -> ML Model -> Website -> Deploy

**Dataset Used:**

TMDB 5000 Movie Dataset available in Kaggle

<https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata>

**Modules Used:**

**1. NumPy (NumPy)**

* **NumPy (Numerical Python)** is a powerful library for **numerical computing** in Python.
* It provides **multi-dimensional arrays (N-d Array)** that are faster and more efficient than Python lists.
* Includes mathematical functions for **linear algebra, statistics, and random number generation**.
* Supports element-wise operations, broadcasting, and integration with other libraries like Pandas and SciPy.

**Uses of NumPy:**

* Efficient storage and manipulation of large datasets.
* Performing mathematical operations on arrays (e.g., matrix multiplication, Fourier transforms).
* Handling large-scale numerical computations in **Machine Learning & Data Science**.

**2. Pandas (pandas)**

* **Pandas** is a data analysis and manipulation library built on top of NumPy.
* It provides **Data Frames** (tabular, spreadsheet-like structures) and **Series** (one-dimensional labeled arrays).
* Used for **data cleaning, transformation, analysis, and visualization**.

**Uses of Pandas:**

* Handling structured data (CSV, Excel, SQL, JSON, etc.).
* Data filtering, grouping, merging, and reshaping.
* Performing statistical and analytical operations on datasets.
* Essential for **Data Science, Data Analytics, and Machine Learning** tasks.

->Genres, id, keywords, title, overview, cast, crew. These will be same and other cols will be removed as they are not necessary.

->We create tags by mixing overview, genres, keywords, cast and crew and make as single column

**Explanation of AST (Abstract Syntax tree) its uses and its usage in this project**

**Step 1: Understanding the import ast**

->import ast

* This imports the **Abstract Syntax Trees (ast)** module.
* This module helps in safely evaluating Python expressions **without executing code**.

**Step 2: The Expression Inside ast.literal\_eval()**

-> '[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}, {"id": 14, "name": "Fantasy"}, {"id": 878, "name": "Science Fiction"}]'

* This is a **string** (enclosed in ' single quotes).
* The string **looks like** a Python list of dictionaries, but right now it is just a string.
* Each dictionary inside the list has two keys:
  + "id" → A number representing the genre ID.
  + "name" → The name of the movie genre.

**Example of what this string looks like (but as an actual Python list)**

[

{"id": 28, "name": "Action"},

{"id": 12, "name": "Adventure"},

{"id": 14, "name": "Fantasy"},

{"id": 878, "name": "Science Fiction"}

]

* The **list contains 4 dictionaries**.
* Each dictionary represents a **movie genre** with an ID and a name.

**Step 3: Using ast.literal\_eval()**

ast.literal\_eval('[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}, {"id": 14, "name": "Fantasy"}, {"id": 878, "name": "Science Fiction"}]')

* ast.literal\_eval() **converts the string** into an actual Python object (a list of dictionaries).
* It **safely** evaluates only literals (like numbers, strings, lists, dictionaries) without executing any code.

**Why is this needed?**

Imagine this:

s = '[1, 2, 3]'

* This is a **string**, not a real list.
* If we do:
* ast.literal\_eval(s)

It converts the string into a real **Python list**:

[1, 2, 3] # Now it's an actual list, not a string.

**Step 4: Final Output**

When you run:

result = ast.literal\_eval('[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}, {"id": 14, "name": "Fantasy"}, {"id": 878, "name": "Science Fiction"}]')

print(result)

It will output:

[

{"id": 28, "name": "Action"},

{"id": 12, "name": "Adventure"},

{"id": 14, "name": "Fantasy"},

{"id": 878, "name": "Science Fiction"}

]

* Now, result is a **real Python list of dictionaries**, not a string.

**Why Not Use eval() Instead?**

eval('[1, 2, 3]')

* This also converts a string into a Python list, but **it is dangerous** ⚠️.
* eval() **executes code**, meaning someone could inject harmful commands.

Example:

eval('\_\_import\_\_("os").system("rm -rf /")') # This could delete all files! ❌

But ast.literal\_eval() **only allows literals** and would prevent such dangerous execution.

**Final Summary**

1. **import ast** → Import the AST module.
2. **Input is a string** → A string that looks like a Python list.
3. **ast.literal\_eval() converts it** → The string turns into a real Python object (list of dictionaries).
4. **Safer than eval()** → It prevents running harmful code.

**Vectors & Text Vectorization in Machine Learning**

**1. What are Vectors in Machine Learning?**

* A **vector** is a mathematical representation of data as an **array of numbers**.
* In Machine Learning, vectors are used to represent **features** of data points in a high-dimensional space.
* Vectors are points on a 2D space
* Stop words in English when removed doesn’t change meaning are not considered like and, or, a, to etc.
* Example: A movie recommendation system may represent movies using a vector like [0.2, 0.8, 0.5], where each value represents a specific feature (e.g., action, comedy, drama).
* **Vectors enable ML models to compute distances, similarities, and transformations efficiently.**

✅ **Uses of Vectors in ML:**

* Representing numerical features in datasets.
* Measuring similarity between data points (e.g., cosine similarity in recommendation systems).
* Feeding structured data into machine learning models (e.g., feature vectors in classification).

**2. What is Text Vectorization?**

* **Text Vectorization** is the process of converting textual data (words, sentences, or documents) into **numerical vectors** that machine learning models can process.
* Since ML models work with numbers, text needs to be transformed into a format they can understand.

✅ **Common Text Vectorization Techniques:**

1. **Bag of Words (BoW):**
   * Converts text into a **word frequency vector** (how many times each word appears in a document).
   * We are using this method
2. **TF-IDF (Term Frequency-Inverse Document Frequency):**
   * Assigns importance to words based on their frequency in a document vs. the entire dataset.
3. **Word Embeddings (Word2Vec, GloVe, FastText):**
   * Maps words to continuous vector space, capturing semantic relationships.
4. **Transformer-based Embeddings (BERT, GPT, etc.):**
   * Uses deep learning to generate contextual word representations.

✅ **Uses of Text Vectorization in ML:**

* Sentiment analysis
* Chatbots and NLP applications
* Document classification
* Spam detection
* Machine translation

**Cosine Similarity in Movie Recommendation Systems**

* **Cosine Similarity** measures the **angle** between two vectors in a high-dimensional space.
* It is used to determine how **similar** two vectors are.
* The **smaller the angle**, the **more similar** the vectors (i.e., the more similar the movies).

**Formula for Cosine Similarity**

**cos(θ)=( A⋅B​) / (∣∣A∣∣×∣∣B∣∣)**

Where:

* A.B is the **dot product** of vectors A and B.
* ∣∣A∣∣ and ∣∣B∣∣ are the **magnitudes (lengths)** of the vectors.
* θ is the **angle** between the vectors.

**Key Points**

✔ **Cosine Similarity ranges from -1 to 1**

* 1 → Identical vectors (exactly the same movies).
* 0 → Completely different vectors (unrelated movies).
* -1 → Opposite vectors (completely dissimilar movies).

✔ **Smaller angle → Higher similarity**

* A lower angle means that the movies share **more similar features**, such as genre, actors, or user ratings.

✔ **Used in Recommendation Systems**

* If a user watches a **sci-fi movie**, cosine similarity can find other **sci-fi movies** with similar features.