CINEMATCH.AI

Smart Personalized Movie Recommendation AI System

https://github.com/Sdhandre/Cinematch_AI

https://cinematchai-cvnfjuwaa8zybhbq7ccsbt.streamlit.app/

Presented By:

Sujal Dhandre

PROBLEM STATEMENT

In today’s digital era, viewers are overwhelmed by the sheer volume of content across OTT platforms like Netflix, Amazon Prime, and Disney+. Navigating this content tsunami to find a personalized, mood-based, or contextually appropriate movie is frustrating for users. Traditional recommender systems are either too generic or tied to a single platform.

CineMatch.AI addresses this gap by offering an intelligent, cross-platform movie recommendation system based on user mood, genre preference, pacing, and context (like who you're watching with). Powered by NLP and Machine Learning, the system goes beyond basic filters and provides context-aware, personalized, and dynamic suggestions in real time.

PRESENT MARKET OVERVIEW

The Indian OTT streaming market is witnessing exponential growth, expected to surge from **₹1.2 lakh crore (~USD 150 billion)** globally in 2023 to **₹2.5 lakh crore (~USD 300 billion)** by 2027. India alone contributes significantly to this growth, fueled by:

- The **widespread adoption of smartphones**, with over **700 million internet users**.
- Increasing demand for **regional language content** and **personalized viewing** experiences.
- The proliferation of platforms like **Netflix, Amazon Prime Video, Disney+ Hotstar, JioCinema, and SonyLIV**.

However, Indian users face major challenges:

- **Decision Fatigue**: With 40+ OTT platforms and thousands of titles, users spend excessive time deciding what to watch.
- **No Unified Discovery**: Recommendations are siloed within individual platforms, with no centralized intelligent engine.
- **Mismatch with Preferences**: Generic trending lists often don’t align with users’ moods, languages, or family-friendly viewing needs.

CineMatch.AI taps into this massive untapped value by acting as a **smart assistant for content discovery**. It addresses the Indian OTT consumer’s pain points by:

- Offering **mood-based, genre-based, and audience-based recommendations** across platforms.
- Supporting **regional language filtering** (Hindi, Tamil, Telugu, Marathi, etc.).
- Helping users avoid **content fatigue** and discover what’s truly worth their time.

This **personalized recommendation engine** enhances:

- **User satisfaction** through faster, accurate suggestions.
- **Platform stickiness** by promoting better content engagement.
- Potential for **OTT affiliate partnerships**, driving monetization via referrals.

PRODUCT INTRODUCTION

CineMatch.AI is an AI-driven movie recommendation platform that learns from user inputs and behavior patterns to suggest content:

Core Features:

- **Mood-Based Recommendation** (Happy, Sad, Thrilling, Romantic…)
- **Genre and Language Filters**
- **Context-Aware Matching** (e.g., watching with friends or partner)
- **Cross-platform Suggestions**
- **Direct Links to OTT apps (if available)**
- **Trailer View & Ratings Summary**
- **Smart Filtering using NLP & TF-IDF**
- **Mobile-Responsive Interface (via Streamlit)**

Dataset & Preprocessing

CineMatch.AI is powered by a highly curated and cleaned movie dataset originally sourced from **TMDb (The Movie Database)**. The raw dataset initially contained over **1 million records**, including both high-quality and noisy or incomplete movie entries.

To ensure the recommendation engine delivers relevant, high-quality suggestions in real-time, we applied the following data cleaning and filtering process.

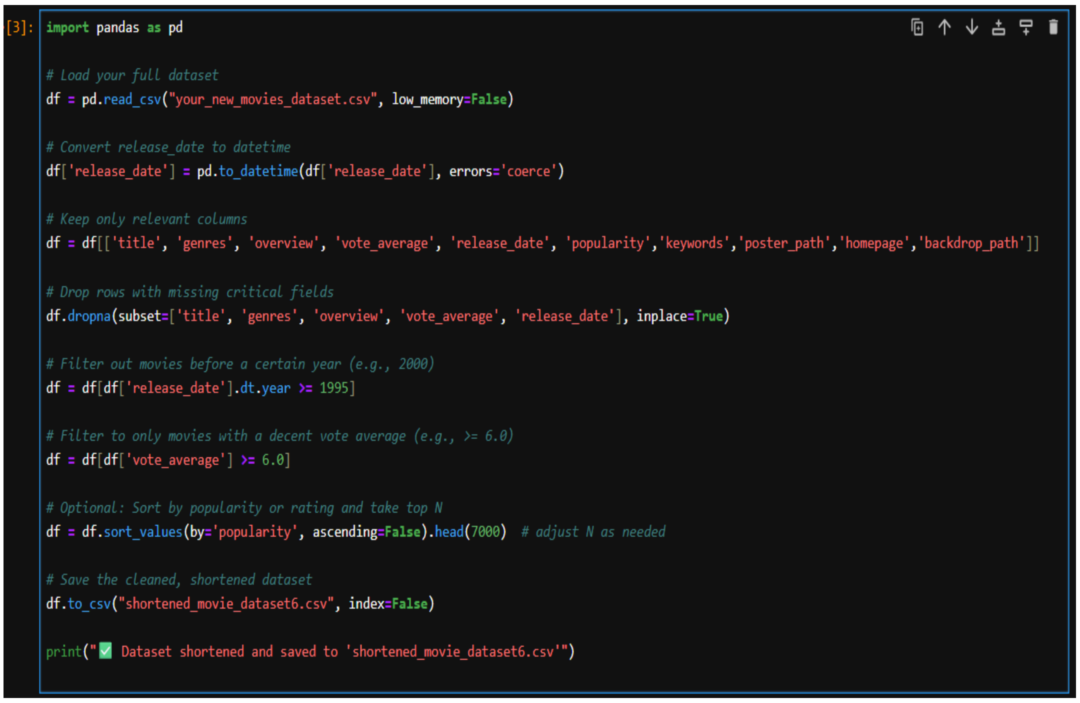
Data Cleaning & Reduction Process

We filtered and processed the dataset using the following steps:

1. **Loaded Raw Dataset**
   Loaded the full TMDb dataset using pandas.
2. **Date Conversion**
   Converted release_date fields into standardized datetime format, handling invalid entries gracefully.

3. **Column Selection**
   Retained only the relevant columns needed for recommendations:
   - title
   - genres
   - overview
   - vote_average
   - release_date
   - popularity
   - keywords
   - poster_path
   - homepage
   - backdrop_path
4. **Missing Value Removal**
   Removed all entries with missing critical fields such as title, genres, overview, vote_average, or release_date.
5. **Time Filter**
   Removed all movies released before **1995** to maintain relevance for modern users.
6. **Quality Filter**
   Filtered out movies with a vote_average below **6.0**, ensuring only well-received content remains.
7. **Top-N Selection by Popularity**
   Sorted movies by popularity and selected the top **7,000** entries for optimal balance between diversity and performance.

CODE:



```
[3]: import pandas as pd

# Load your full dataset
df = pd.read_csv("your_new_movies_dataset.csv", low_memory=False)

# Convert release_date to datetime
df['release_date'] = pd.to_datetime(df['release_date'], errors='coerce')

# Keep only relevant columns
df = df[['title', 'genres', 'overview', 'vote_average', 'release_date', 'popularity','keywords','poster_path','homepage','backdrop_path']]

# Drop rows with missing critical fields
df.dropna(subset=['title', 'genres', 'overview', 'vote_average', 'release_date'], inplace=True)

# Filter out movies before a certain year (e.g., 2000)
df = df[df['release_date'].dt.year >= 1995]

# Filter to only movies with a decent vote average (e.g., >= 6.0)
df = df[df['vote_average'] >= 6.0]

# Optional: Sort by popularity or rating and take top N
df = df.sort_values(by='popularity', ascending=False).head(7000)  # adjust N as needed

# Save the cleaned, shortened dataset
df.to_csv("shortened_movie_dataset6.csv", index=False)

print("✅ Dataset shortened and saved to 'shortened_movie_dataset6.csv'")
```

BUSINESS NEED ASSESSMENT

1. Market Dynamics

- **High Demand for Content Aggregation**: Users use 2–3 OTT platforms on average.
- **Rising Need for Hyper-Personalization**: OTT fatigue is driving need for customized recommendations.
- **Tech-savvy Youth Demographic**: Major target audience includes users aged 16–35 in urban markets.

2. Key Customer Pain Points

- Overwhelmed by content on OTT platforms.
- Lack of consolidated recommendation across services.
- Repetitive or inaccurate algorithmic suggestions.
- No mood-based or context-driven discovery.

3. Business Requirements

- Cross-platform OTT integration or affiliate linking.
- Accurate personalization through AI/ML.
- Affordable subscription and optional freemium access.
- Simple user interface accessible to all age groups.

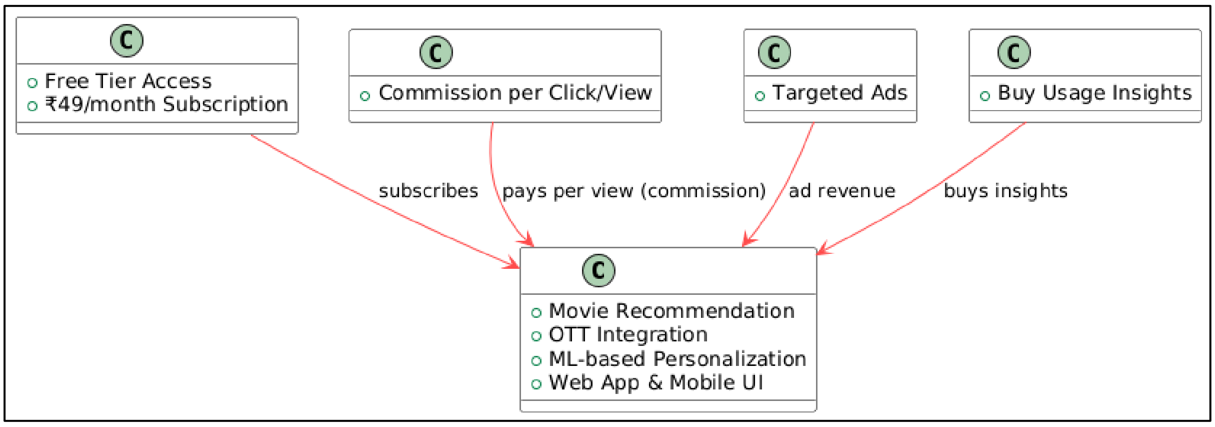
TARGET AUDIENCE

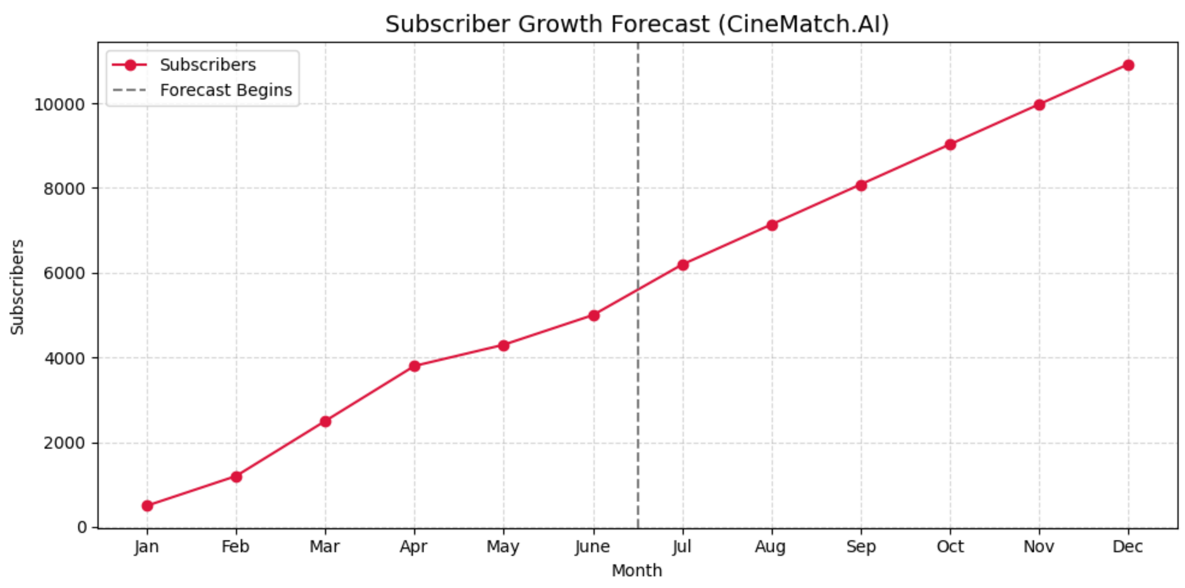
- **Casual Streamers**: Looking for easy suggestions.
- **Binge Watchers**: Exploring new genres and moods.
- **Students and Young Adults**: Time-constrained users needing fast recommendations.
- **OTT Subscribers**: Users across Netflix, Prime Video, etc.

BUSINESS MODEL

Monetization Strategy:

1. **Subscription Model**
   - ₹49/month for ad-free, smart features.
2. **Freemium Access**
   - Free version with ads and basic filters.
3. **Affiliate Revenue**
   - Earn commission per OTT click-through and signup via CineMatch.
4. **In-App Ads**
   - Targeted movie and streaming ads per user interaction.
5. **Data Analytics as a Service**
   - Offering trend analytics to OTT platforms or content producers.





FINANCIAL MODELING

Assumptions:

- **Subscription Fee:** ₹49
- **Subscribers (June):** 5,000
- **Affiliate Success (20% of users):** 1,000 users × ₹15 = ₹15,000
- **Ad Revenue:** ₹0.5/user/day × 30 × 5,000 = ₹75,000
- **Operational Costs (Monthly):** ₹40,000

Total Revenue for July (just a prediction)

Revenue Stream	**Amount (INR)**
Subscription (49×5000)	₹2,45,000
OTT Affiliate (15×1000)	₹15,000
Ad Revenue	₹75,000
Gross Revenue	₹3,35,000
− Operational Cost	₹40,000
Net Revenue (July)	₹2,95,000

Final Practical Revenue Equation:

Let:

- x = total subscribers/month
- r = % of subscribers that click affiliate link (default 0.2)
- a = ad income per user per day (₹0.5)
- Fixed Cost = ₹40,000

Then:

$$y = 49x + (r \times x \times 15) + (a \times x \times 30) - 40{,}000$$

With default values:

$$y = 49x + 3x + 15x - 40{,}000 = 67x - 40{,}000$$

So the **final revenue function** is:

$$y = 67x - 40{,}000$$

Technical Features of CineMatch AI

1. User Interface (Frontend)

- **Framework**: Streamlit is used for rapid UI prototyping.
- **Theme**: A modern, animated UI with custom CSS (glass morphism, animations, mobile responsiveness).
- **User Input Form**:
  - Mood, Genre, Language, Pace, Ending Preference, and Viewing Context.
  - Clean three-column layout for desktop UX.
- **Recommendation Display**:
  - Poster images fetched from TMDB.
  - Movie overview, genre tags, year of release, and keywords.
  - Watch trailer via dynamic YouTube link.
- **Pagination & Refresh**:
  - Display 10 movie cards per page.
  - Supports shuffle (random), previous, and next views.

---

2. Data Handling & Preprocessing

a. Dataset:

- CSV: shortened_movie_dataset6.csv

- Fields used: genres, keywords, overview, release_date, original_language, poster_path, homepage, trailer_url

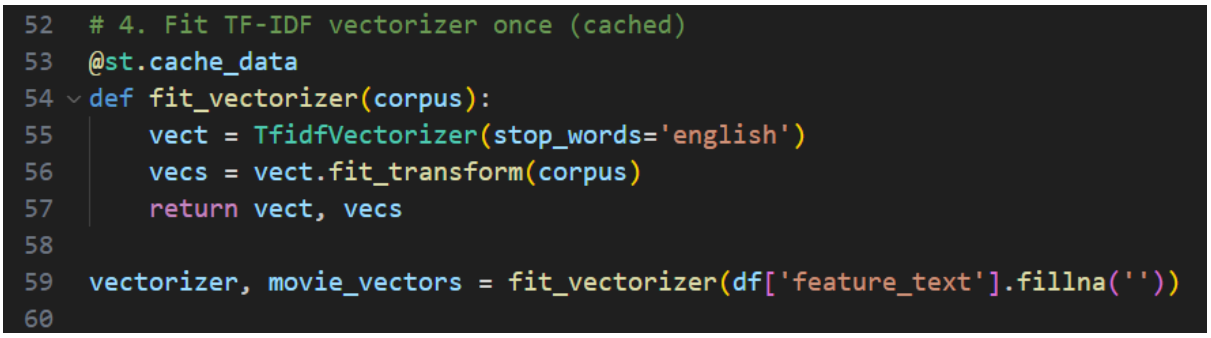
b. Preprocessing Steps:

- **Datetime Parsing**: Robust parsing for release_date (ISO, DD-MM-YYYY, etc.).
- **Feature Text Generation**:
  - Merges genres, keywords, and other attributes into a single string for ML vectorization.
- **Filters Applied**:
  - By Language
  - By Era (custom decade-based filtering using a dictionary)
  - Other preference filters collected from UI

Machine Learning Pipeline (Core Logic)

1. Text Vectorization using TF-IDF

- **Library**: TfidfVectorizer from sklearn.feature_extraction.text
- **Purpose**: Converts textual data (like genres and keywords) into a numerical vector space that can be mathematically compared.
- **Corpus**: Feature text from all movies (genres + keywords etc.)
- **Result**: Sparse matrix representing movie content in feature space.



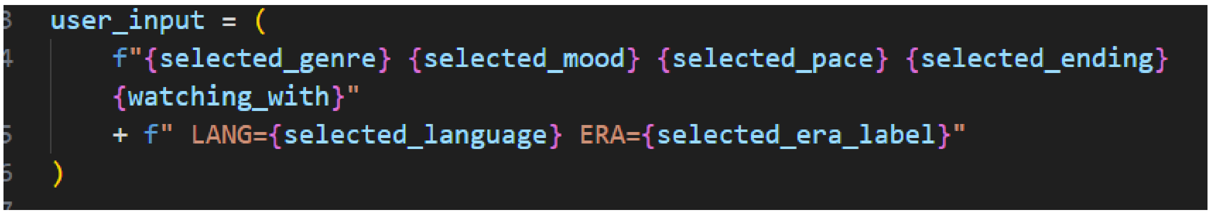
```python
# 4. Fit TF-IDF vectorizer once (cached)
@st.cache_data
def fit_vectorizer(corpus):
    vect = TfidfVectorizer(stop_words='english')
    vecs = vect.fit_transform(corpus)
    return vect, vecs

vectorizer, movie_vectors = fit_vectorizer(df['feature_text'].fillna(''))
```

---

2. User Profile Vector Creation

- A new user vector is created dynamically based on the selected preferences:



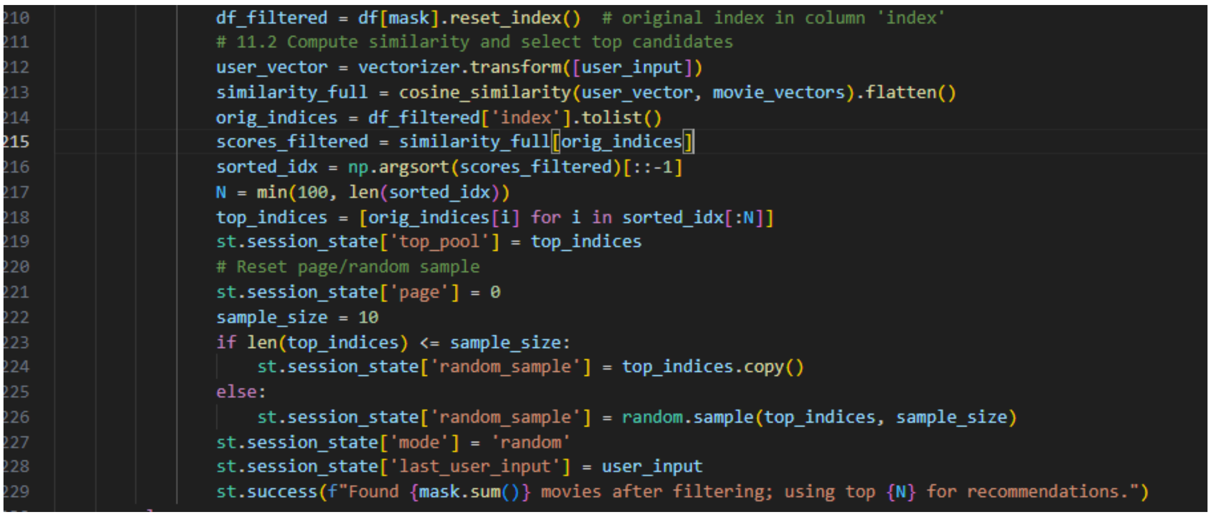
```
user_input = (
    f"{selected_genre} {selected_mood} {selected_pace} {selected_ending}
    {watching_with}"
    + f" LANG={selected_language} ERA={selected_era_label}"
)
```

user_vector = vectorizer.transform([user_input])

---

3. Similarity Computation using Cosine Similarity

- **Library**: cosine_similarity from sklearn.metrics.pairwise
- **Purpose**: Measures how similar a movie vector is to the user's preference vector (angle-based similarity in high-dimensional space).



```
df_filtered = df[mask].reset_index()  # original index in column 'index'
# 11.2 Compute similarity and select top candidates
user_vector = vectorizer.transform([user_input])
similarity_full = cosine_similarity(user_vector, movie_vectors).flatten()
orig_indices = df_filtered['index'].tolist()
scores_filtered = similarity_full[orig_indices]
sorted_idx = np.argsort(scores_filtered)[::-1]
N = min(100, len(sorted_idx))
top_indices = [orig_indices[i] for i in sorted_idx[:N]]
st.session_state['top_pool'] = top_indices
# Reset page/random sample
st.session_state['page'] = 0
sample_size = 10
if len(top_indices) <= sample_size:
    st.session_state['random_sample'] = top_indices.copy()
else:
    st.session_state['random_sample'] = random.sample(top_indices, sample_size)
st.session_state['mode'] = 'random'
st.session_state['last_user_input'] = user_input
st.success(f"Found {mask.sum()} movies after filtering; using top {N} for recommendations.")
```

ML Algorithm Used

Component	**Algorithm / Method**	**Purpose**
TF-IDF	Term Frequency–Inverse Document Frequency	Feature extraction from text
Cosine Similarity	Vector Similarity Measure	Comparing user preferences with movie features
Filtering	Rule-based Filtering	Filter data based on language and time
Recommendation	Content-Based Filtering	Personalization without needing user ratings

Future ML Enhancements

To evolve CineMatch into a more intelligent engine:

- Integrate **Collaborative Filtering** (e.g., Matrix Factorization, SVD).
- Add **Sentiment Analysis** on user feedback to refine models.
- Build **Neural Recommendation Systems** using embedding layers (e.g., TensorFlow Recommenders).
- Implement **User Behavioral Clustering** using K-Means or DBSCAN to segment users.