Movie Recommendation System using Cosine Similarity - Complete Guide

Table of Contents

What is a Recommendation System?

Definition

A recommendation system is an algorithm that suggests relevant items to users based on their preferences, behavior, or similarities between items.

Real-World Examples

- **Netflix**: "Because you watched..."
- **Amazon**: "Customers who bought this also bought..."
- **Spotify**: "Discover Weekly" playlist
- **YouTube**: "Recommended for you"
- **IMDb**: "More like this"

Why Important?

- **User Experience**: Helps users discover new content
- **Business Value**: Increases engagement and revenue
- **Information Filtering**: Manages information overload
- **Personalization**: Tailors content to individual preferences

---

Types of Recommendation Systems

1. **Content-Based Filtering**

Recommends items similar to what the user has liked before.

How it works:

- Analyzes item features (genre, director, actors)
- Creates user profile based on liked items
- Finds similar items using these features

Example:

```
User likes: "The Dark Knight" (Action, Crime, Drama, Christian Bale)
Recommendations: Other Christopher Nolan films, Batman movies, Christian Bale movies
```

Pros:

- No cold start problem for items
- Transparent recommendations
- Works for niche interests

Cons:

- Limited diversity (filter bubble)
- Requires good item features
- Can't discover new genres

2. **Collaborative Filtering**

Recommends items based on similar users' preferences.

How it works:

- Finds users with similar tastes
- Recommends items liked by similar users
- "People like you also liked..."

Example:

```
User A likes: Inception, Interstellar, The Matrix
User B likes: Inception, Interstellar, Blade Runner
Recommendation for A: Blade Runner (because B, who has similar taste, liked it)
```

Pros:

- Discovers new genres/categories
- No need for item features
- Improves with more users

Cons:

- Cold start problem (new users/items)
- Sparsity issues
- Popular item bias

Content-Based vs Collaborative Filtering

Content-Based Filtering (Our Focus)

Mathematical Approach:

```
similarity(movie_A, movie_B) = cosine_similarity(features_A, features_B)

Where features = [genre, director, stars, rating, year, etc.]
```

Example Process:

```
# Movie representation
dark_knight = [Action=1, Crime=1, Nolan=1, Bale=1, Rating=9.0]
batman_begins = [Action=1, Crime=1, Nolan=1, Bale=1, Rating=8.2]

# High similarity because same director, actor, genres
similarity = cosine_similarity(dark_knight, batman_begins) = 0.95
```

Collaborative Filtering Theory

User-Based Collaborative Filtering:

```
1. Find users similar to target user
2. Recommend items liked by similar users
3. Weight recommendations by user similarity
```

Mathematical Formula:

```
prediction(user, item) = Σ(similarity(user, other_user) × rating(other_user, item)) / Σ(similarity scores)
```

Example User-Item Matrix:

```
         Movie_A  Movie_B  Movie_C  Movie_D
User_1      5        3        ?        4
User_2      4        ?        2        5
User_3      ?        2        3        3
User_4      3        4        4        ?

To predict User_1's rating for Movie_C:
1. Find users similar to User_1 (based on common ratings)
2. User_2 and User_4 are similar to User_1
3. User_2 rated Movie_C as 2, User_4 rated as 4
4. Weighted average: prediction ≈ 3.2
```

Item-Based Collaborative Filtering:

```
1. Find items similar to target item
2. Recommend based on user's ratings of similar items
3. "Users who liked X also liked Y"
```

Similarity Calculation Between Users:

```
# User rating vectors
user_A = [5, 3, 4, 2, 5]  # Ratings for 5 movies
user_B = [4, 2, 5, 3, 4]  # Same movies

# Pearson correlation or cosine similarity
similarity = cosine_similarity(user_A, user_B)
```

Cold Start Problem:

- **New User**: No rating history → Can't find similar users
- **New Item**: No ratings → Can't recommend
- **Solutions**: Hybrid systems, demographic data, popularity-based recommendations

Sparsity Problem:

```
User-Item Matrix (1000 users × 10000 movies):
- Total possible ratings: 10,000,000
- Actual ratings: ~50,000 (0.5% filled)
- Most cells are empty → Hard to find similarities
```

Understanding Cosine Similarity

What is Cosine Similarity?

Cosine similarity measures the cosine of the angle between two vectors. It determines how similar two items are regardless of their magnitude.

Mathematical Formula

```
cosine_similarity(A, B) = (A · B) / (||A|| × ||B||)

Where:
- A · B = dot product of vectors A and B
- ||A|| = magnitude (length) of vector A
- ||B|| = magnitude (length) of vector B
```

Step-by-Step Calculation Example:

```
Movie A: [1, 0, 1, 3]  # [Action, Comedy, Drama, Rating]
Movie B: [1, 0, 1, 4]  # Same features

Step 1: Calculate dot product (A · B)
A · B = (1×1) + (0×0) + (1×1) + (3×4) = 1 + 0 + 1 + 12 = 14

Step 2: Calculate magnitudes
||A|| = √(1² + 0² + 1² + 3²) = √(1 + 0 + 1 + 9) = √11 ≈ 3.32
||B|| = √(1² + 0² + 1² + 4²) = √(1 + 0 + 1 + 16) = √18 ≈ 4.24

Step 3: Calculate cosine similarity
cosine_similarity = 14 / (3.32 × 4.24) = 14 / 14.08 ≈ 0.994

Result: Very high similarity (almost identical movies)
```

Range of Values

- **1.0**: Identical vectors (perfect similarity)
- **0.0**: Orthogonal vectors (no similarity)
- **-1.0**: Opposite vectors (perfect dissimilarity)

Why Use Cosine Similarity for Movies?

1. **Scale Independence**

Different features have different scales:

- **IMDb Rating**: 1-10
- **Runtime**: 60-300 minutes
- **Number of votes**: 1K-2M

Cosine similarity ignores magnitude, focuses on proportions.

2. **Example Comparison**

```
Movie A: [Action=1, Drama=1, Rating=8.0, Votes=100K]
Movie B: [Action=2, Drama=2, Rating=8.0, Votes=200K]

Euclidean Distance: Large (due to different vote counts)
Cosine Similarity: 1.0 (same proportions/direction)
```

Movie B is just a "scaled up" version of Movie A - cosine similarity correctly identifies them as identical in nature.

Dataset Overview

IMDB Movies Dataset Structure

```
Features: 15 columns
Sample size: ~1000 movies

Columns:
- Poster_Link: URL to movie poster
- Series_Title: Movie name
- Released_Year: Release year
- Certificate: Rating (PG, R, etc.)
- Runtime: Duration in minutes
- Genre: Movie categories (comma-separated)
- IMDB_Rating: 1-10 scale
- Overview: Plot summary text
- Meta_score: Critics' score (0-100)
- Director: Director name
- Star1, Star2, Star3, Star4: Main actors
- No_of_votes: User votes count
- Gross: Box office earnings
```

Sample Data Row:

```
Series_Title: "The Shawshank Redemption"
Released_Year: 1994
Certificate: "R"
Runtime: "142 min"
Genre: "Drama"
IMDB_Rating: 9.3
Overview: "Two imprisoned men bond over a number of years..."
Director: "Frank Darabont"
Star1: "Tim Robbins"
Star2: "Morgan Freeman"
No_of_votes: 2343110
Gross: "16,000,000"
```

Mathematical Foundation

Vector Representation Process

Step 1: Feature Extraction

```
# Original movie data
movie = {
    'title': 'The Dark Knight',
    'genre': 'Action, Crime, Drama',
    'director': 'Christopher Nolan',
    'stars': ['Christian Bale', 'Heath Ledger'],
    'rating': 9.0,
    'year': 2008
}
```

Step 2: One-Hot Encoding for Categorical Features

```
# Genre encoding (if we have 5 total genres)
genres = ['Action', 'Comedy', 'Drama', 'Horror', 'Romance']
movie_genres = [1, 0, 1, 0, 0]  # Action=1, Drama=1, others=0

# Director encoding (if we track top 10 directors)
directors = ['Nolan', 'Spielberg', 'Scorsese', ...]
movie_director = [1, 0, 0, ...]  # Nolan=1, others=0
```

Step 3: Feature Vector Creation

```
# Final feature vector
movie_vector = [
    # Genre features (5 elements)
    1, 0, 1, 0, 0,
    # Director features (10 elements)
    1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    # Star features (20 elements)
    1, 1, 0, 0, 0, ...,
    # Numerical features (normalized to 0-1)
    0.9,    # IMDB rating (9.0/10)
    0.67,   # Year (normalized)
    0.85    # Runtime (normalized)
]

# Total vector length: 5 + 10 + 20 + 3 = 38 features
```

Code Implementation

```
# Import required libraries
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import StandardScaler, MultiLabelBinarizer
import warnings
warnings.filterwarnings('ignore')
```

Explanation:

- `pandas`: Data manipulation and CSV reading
- `numpy`: Numerical operations and arrays
- `TfidfVectorizer`: Converts text to numerical features
- `cosine_similarity`: Calculates similarity between movie vectors
- `StandardScaler`: Normalizes numerical features to same scale
- `MultiLabelBinarizer`: Converts categories (genres) to binary features

```
def load_and_preprocess_data(file_path):
    """
    Load IMDB dataset and clean the data for processing

    Problems solved:
    - Missing values in various columns
    - Mixed data types (strings with numbers)
    - Inconsistent formats ("142 min" vs "142")
    """
    # Load CSV file into pandas DataFrame
    df = pd.read_csv(file_path)

    # Handle missing values by filling with appropriate defaults
    df['Overview'].fillna('', inplace=True)              # Empty string for missing plots
    df['Certificate'].fillna('Not Rated', inplace=True)  # Default rating
    df['Gross'].fillna('0', inplace=True)                # Zero for missing earnings

    # Clean Runtime column: "142 min" → 142
    df['Runtime'] = df['Runtime'].astype(str).str.replace(' min', '').str.replace(',', '')
    df['Runtime'] = pd.to_numeric(df['Runtime'], errors='coerce')  # Convert to numbers
    df['Runtime'].fillna(df['Runtime'].mean(), inplace=True)       # Fill NaN with average

    # Clean Gross earnings: "16,000,000" → 16000000
    df['Gross'] = df['Gross'].astype(str).str.replace(',', '').str.replace('$', '')
    df['Gross'] = pd.to_numeric(df['Gross'], errors='coerce')
    df['Gross'].fillna(0, inplace=True)

    # Clean Meta_score: ensure it's numeric
    df['Meta_score'] = pd.to_numeric(df['Meta_score'], errors='coerce')
    df['Meta_score'].fillna(df['Meta_score'].mean(), inplace=True)

    # Clean Released_Year: ensure it's numeric
    df['Released_Year'] = pd.to_numeric(df['Released_Year'], errors='coerce')
    df['Released_Year'].fillna(2000, inplace=True)  # Default year if missing

    # Clean IMDB_Rating: ensure it's numeric
    df['IMDB_Rating'] = pd.to_numeric(df['IMDB_Rating'], errors='coerce')
    df['IMDB_Rating'].fillna(df['IMDB_Rating'].mean(), inplace=True)

    # Clean No_of_Votes: ensure it's numeric
    df['No_of_Votes'] = pd.to_numeric(df['No_of_Votes'], errors='coerce')
    df['No_of_Votes'].fillna(df['No_of_Votes'].mean(), inplace=True)

    return df
```

Key Concepts:

- **Data Cleaning:** Convert inconsistent formats to standardized numbers
- **Missing Value Handling:** Fill gaps with reasonable defaults (mean, zero, empty string)
- **Type Conversion:** Use `pd.to_numeric(errors='coerce')` to safely convert strings to numbers
- **Robust Processing:** Handle any unexpected data formats without crashing

```python
def create_movie_features(df):
    """
    Convert movie data into numerical feature vectors for similarity calculation

    Creates 5 types of features:
    1. Genre features (binary: has genre or not)
    2. Director features (binary: is by this director or not)
    3. Star features (binary: has this actor or not)
    4. Text features (TF-IDF from plot summaries)
    5. Numerical features (ratings, year, etc.)
    """

    # 1. GENRE FEATURES
    # Convert "Action, Drama, Thriller" → [1,0,1,0,1,0] (binary vector)
    genre_lists = []
    for genre_str in df['Genre'].fillna(''):
        # Split comma-separated genres into list
        genres = [g.strip() for g in genre_str.split(',') if g.strip()]
        genre_lists.append(genres)

    # MultiLabelBinarizer creates binary encoding
    # Example: [["Action", "Drama"], ["Comedy"]] → [[1,0,1], [0,1,0]]
    mlb_genre = MultiLabelBinarizer()
    genre_features = mlb_genre.fit_transform(genre_lists)

    # 2. DIRECTOR FEATURES
    # Only use top 50 directors to avoid too many sparse features
    director_counts = df['Director'].fillna('Unknown').value_counts()
    top_directors = director_counts.head(50).index.tolist()

    # Create binary matrix: 1 if movie is by this director, 0 otherwise
    director_features = np.zeros((len(df), len(top_directors)))

    for i, director in enumerate(df['Director'].fillna('Unknown')):
        if director in top_directors:
            director_idx = top_directors.index(director)
            director_features[i, director_idx] = 1

    # 3. STAR FEATURES
    # Collect all actors from Star1, Star2, Star3, Star4 columns
    all_stars = []
    star_columns = ['Star1', 'Star2', 'Star3', 'Star4']

    for col in star_columns:
        if col in df.columns:
            stars = df[col].fillna('').tolist()
            all_stars.extend([star for star in stars if star])

    # Get top 100 most frequent stars
    star_counts = pd.Series(all_stars)
    top_stars = star_counts.value_counts().head(100).index.tolist()

    # Create binary matrix for star features
    star_features = np.zeros((len(df), len(top_stars)))

    for i in df.index:
        # Get all stars for this movie
        movie_stars = []
        for col in star_columns:
            if col in df.columns and pd.notna(df.loc[i, col]):
                movie_stars.append(df.loc[i, col])

        # Mark which top stars appear in this movie
        for star in movie_stars:
            if star in top_stars:
                star_idx = top_stars.index(star)
                star_features[i, star_idx] = 1

    # 4. TEXT FEATURES FROM PLOT SUMMARIES
    # Convert movie overviews to numerical features using TF-IDF
    def clean_overview(text):
        if pd.isna(text):
            return ""
        return str(text).lower()

    clean_overviews = df['Overview'].apply(clean_overview)

    try:
        # TF-IDF: Term Frequency-Inverse Document Frequency
        # Gives higher weight to distinctive words, lower to common words
        tfidf = TfidfVectorizer(
            max_features=100,    # Use top 100 most important words
            stop_words='english', # Remove common words (the, and, is, etc.)
            ngram_range=(1, 2),  # Use single words and word pairs
            min_df=2             # Word must appear in at least 2 movies
        )
        text_features = tfidf.fit_transform(clean_overviews).toarray()
    except:
        # Fallback if text processing fails
        text_features = np.zeros((len(df), 10))

    # 5. NUMERICAL FEATURES
    # Normalize numerical columns so they contribute equally to similarity
    potential_cols = ['IMDB_Rating', 'Meta_score', 'Runtime', 'Released_Year', 'No_of_Votes', 'Gross']
    available_cols = [col for col in potential_cols if col in df.columns]

    if available_cols:
        numerical_data = df[available_cols].copy()

        # Ensure all data is numeric and handle any remaining NaN values
        for col in available_cols:
            numerical_data[col] = pd.to_numeric(numerical_data[col], errors='coerce')
            if numerical_data[col].isnull().any():
                mean_val = numerical_data[col].mean()
                numerical_data[col].fillna(mean_val, inplace=True)

        try:
            # StandardScaler: converts all features to mean=0, std=1
            # This ensures IMDB_Rating (1-10) and No_of_Votes (1K-2M) contribute equally
            scaler = StandardScaler()
            numerical_features = scaler.fit_transform(numerical_data)
        except:
            numerical_features = numerical_data.values
    else:
        numerical_features = np.zeros((len(df), 1))

    # 6. COMBINE ALL FEATURES INTO SINGLE MATRIX
```

```
    # Horizontally stack all feature types
    feature_matrix = np.hstack([
        genre_features,      # Binary: [1,0,1,0,0,1,...]
        director_features,   # Binary: [0,1,0,0,0,0,...]
        star_features,       # Binary: [1,1,0,1,0,0,...]
        text_features,       # Continuous: [0.23, 0.45, 0.12,...]
        numerical_features   # Normalized: [-0.5, 1.2, -0.8,...]
    ])

    return feature_matrix
```

Key Concepts:

1. **Binary Encoding:** Converts categories to 1/0 vectors
    - "Action, Drama" → [1,0,1,0,0] for [Action, Comedy, Drama, Horror, Romance]
2. **TF-IDF (Term Frequency-Inverse Document Frequency):**
    - **TF:** How often word appears in document
    - **IDF:** How rare word is across all documents
    - **Result:** Important words get higher weights
3. **Feature Normalization:**
    - IMDB Rating: 1-10 scale
    - Number of Votes: 1,000-2,000,000 scale
    - StandardScaler makes both contribute equally
4. **Dimensionality:** Final feature vector might be 300+ dimensions combining all feature types

---

```
def calculate_similarity_matrix(feature_matrix):
    """
    Calculate cosine similarity between all pairs of movies

    Creates N×N matrix where N = number of movies
    Each cell [i,j] = similarity between movie i and movie j

    Mathematical process:
    For each pair of movies A and B:
    1. Calculate dot product: A·B = sum(A[i] * B[i])
    2. Calculate magnitudes: ||A|| = sqrt(sum(A[i]²))
    3. Cosine similarity = (A·B) / (||A|| × ||B||)
    """
    return cosine_similarity(feature_matrix)
```

Mathematical Example:

```
Movie A features: [1, 0, 1, 0.5]  # [Action, Comedy, Drama, Rating_normalized]
Movie B features: [1, 0, 1, 0.6]  # Same genres, slightly higher rating

Dot product: (1×1) + (0×0) + (1×1) + (0.5×0.6) = 2.3
Magnitude A: √(1² + 0² + 1² + 0.5²) = √2.25 = 1.5
Magnitude B: √(1² + 0² + 1² + 0.6²) = √2.36 = 1.54

Cosine similarity = 2.3 / (1.5 × 1.54) = 0.995 (very similar!)
```

```
def get_movie_recommendations(movie_title, movies_df, similarity_matrix,
                              filter_genre=None, filter_director=None, filter_star=None,
                              top_n=10):
    """
    Get movie recommendations based on cosine similarity with optional filters

    Process:
    1. Find target movie in dataset
    2. Get its similarity scores with all other movies
    3. Sort movies by similarity (highest first)
    4. Apply optional filters (genre, director, star)
    5. Return top N recommendations

    Parameters:
    - movie_title: Name of movie to find similar movies for
    - filter_genre: Only recommend movies of this genre
    - filter_director: Only recommend movies by this director
    - filter_star: Only recommend movies with this actor
    - top_n: Number of recommendations to return
    """

    # STEP 1: Find the target movie
    movie_matches = movies_df[movies_df['Series_Title'].str.contains(movie_title, case=False, na=False)]

    if movie_matches.empty:
        print(f"Movie '{movie_title}' not found")
        return []

    # Get first matching movie
    movie_idx = movie_matches.index[0]

    # STEP 2: Get similarity scores for this movie with all others
    similarity_scores = similarity_matrix[movie_idx]  # Get row from similarity matrix

    # STEP 3: Create (movie_index, similarity_score) pairs and sort
    movie_similarity_pairs = [(i, score) for i, score in enumerate(similarity_scores)]
    sorted_similar_movies = sorted(movie_similarity_pairs, key=lambda x: x[1], reverse=True)[1:]  # [1:] excludes self

    # STEP 4: Apply filters and collect recommendations
    recommendations = []

    for movie_idx, similarity_score in sorted_similar_movies:
        movie_info = movies_df.iloc[movie_idx]

        # Apply genre filter
        if filter_genre and filter_genre.lower() not in movie_info['Genre'].lower():
            continue

        # Apply director filter
        if filter_director and filter_director.lower() not in movie_info['Director'].lower():
            continue

        # Apply star filter
        if filter_star:
            movie_stars = [str(movie_info.get('Star1', '')), str(movie_info.get('Star2', '')),
                          str(movie_info.get('Star3', '')), str(movie_info.get('Star4', ''))]
            star_found = any(filter_star.lower() in star.lower() for star in movie_stars if star and star != 'nan')
            if not star_found:
                continue

        # Movie passed all filters - add to recommendations
        recommendations.append({
            'rank': len(recommendations) + 1,
            'title': movie_info['Series_Title'],
            'year': int(movie_info['Released_Year']) if pd.notna(movie_info['Released_Year']) else 'Unknown',
            'genre': movie_info['Genre'],
            'director': movie_info['Director'],
            'stars': f"{movie_info.get('Star1', '')}, {movie_info.get('Star2', '')}".strip(', '),
            'imdb_rating': movie_info['IMDB_Rating'] if pd.notna(movie_info['IMDB_Rating']) else 'N/A',
            'similarity_score': similarity_score
        })

        # Stop when we have enough recommendations
        if len(recommendations) >= top_n:
            break

    return recommendations
```

Algorithm Explanation:

1. **Movie Search:** Uses fuzzy matching with `str.contains()` for flexible search
2. **Similarity Retrieval:** Gets pre-computed similarities from matrix (O(1) lookup)
3. **Sorting:** Sorts all movies by similarity score (highest = most similar)
4. **Filtering:** Applies multiple filters simultaneously using AND logic
5. **Top-N Selection:** Returns only the requested number of recommendations

Filtering Logic:

- **Genre Filter:** Checks if filter string appears in movie's genre list
- **Director Filter:** Checks if filter string matches director name
- **Star Filter:** Checks if filter string matches any of the 4 main stars
- **Multiple Filters:** Must pass ALL active filters to be recommended

---

```
def display_recommendations(recommendations):
    """
    Display movie recommendations in clean, readable format

    Shows key information for each recommended movie:
    - Title and year
    - IMDb rating and similarity score
    - Genre and director
    - Main stars
    """
    if not recommendations:
        print("No recommendations found")
        return

    for rec in recommendations:
        print(f"{rec['rank']}. {rec['title']} ({rec['year']})")
        print(f"   Rating: {rec['imdb_rating']} | Similarity: {rec['similarity_score']:.3f}")
        print(f"   Genre: {rec['genre']}")
        print(f"   Director: {rec['director']}")
        print(f"   Stars: {rec['stars']}")
        print("-" * 60)
```

Output Format:

```
1. The Dark Knight Rises (2012)
   Rating: 8.4 | Similarity: 0.892
   Genre: Action, Crime
   Director: Christopher Nolan
   Stars: Christian Bale, Tom Hardy
------------------------------------------------------------
```

```
def setup_recommendation_system(file_path):
    """
    One-function setup for the entire recommendation system

    Process:
    1. Load and clean the data
    2. Extract and combine all features
    3. Calculate similarity matrix
    4. Return ready-to-use components

    This function does all the heavy computation once,
    then recommendations are fast lookups
    """
    movies_df = load_and_preprocess_data(file_path)
    feature_matrix = create_movie_features(movies_df)
    similarity_matrix = calculate_similarity_matrix(feature_matrix)
    return movies_df, similarity_matrix
```

System Architecture:

- **Preprocessing:** Clean data once upfront
- **Feature Engineering:** Convert all movies to numerical vectors once
- **Similarity Calculation:** Compute all pairwise similarities once (expensive)
- **Recommendation:** Fast lookups from pre-computed similarity matrix

---

Usage Examples in Jupyter

```
# Cell 1: Setup (run once)
movies_df, similarity_matrix = setup_recommendation_system("imdb_top_1000.csv")
print(f"Loaded {len(movies_df)} movies")
print(f"Feature matrix shape: {similarity_matrix.shape}")

# Cell 2: Basic recommendations
recs = get_movie_recommendations("Inception", movies_df, similarity_matrix, top_n=5)
display_recommendations(recs)

# Cell 3: Filter by genre
recs = get_movie_recommendations("Inception", movies_df, similarity_matrix,
                                filter_genre="Sci-Fi", top_n=5)
display_recommendations(recs)

# Cell 4: Filter by director
recs = get_movie_recommendations("Inception", movies_df, similarity_matrix,
                                filter_director="Christopher Nolan", top_n=5)
display_recommendations(recs)

# Cell 5: Filter by star
recs = get_movie_recommendations("Inception", movies_df, similarity_matrix,
                                filter_star="Leonardo DiCaprio", top_n=5)
display_recommendations(recs)

# Cell 6: Multiple filters
recs = get_movie_recommendations("Inception", movies_df, similarity_matrix,
                                filter_genre="Sci-Fi",
                                filter_director="Christopher Nolan",
                                top_n=3)
display_recommendations(recs)

# Cell 7: Explore dataset
print(f"Dataset info:")
print(f"Movies: {len(movies_df)}")
print(f"Years: {movies_df['Released_Year'].min()}-{movies_df['Released_Year'].max()}")
print(f"Avg Rating: {movies_df['IMDB_Rating'].mean():.2f}")
print(f"Genres: {len(set([g.strip() for genres in movies_df['Genre'].fillna('') for g in genres.split(',')]))}")
```

How It Works - Complete Flow

1. **Data Loading:** Read CSV → Clean inconsistent formats → Handle missing values
2. **Feature Engineering:**
    - Genres → Binary vectors [1,0,1,0,...]
    - Directors → Binary vectors [0,1,0,0,...]
    - Stars → Binary vectors [1,1,0,1,...]
    - Text → TF-IDF vectors [0.23, 0.45, ...]
    - Numbers → Normalized vectors [-0.5, 1.2, ...]
3. **Similarity Calculation:** Cosine similarity between all movie feature vectors
4. **Recommendation:** For target movie, find most similar movies using pre-computed similarities
5. **Filtering:** Apply genre/director/star filters to results
6. **Display:** Show formatted recommendations with key movie information

Why This Works:

- **Content-Based:** Recommends based on movie characteristics, not user behavior
- **Comprehensive Features:** Uses multiple types of information (plot, cast, ratings, etc.)
- **Scalable:** Pre-computed similarities make recommendations fast

- **Flexible:** Multiple filter options for targeted recommendations