Machine Learning Project:

Movie Recommendation System

Marcel YAMMINE
Han WANG
Thomas WARTELLE

DIA 6

2025/2026

Contents

1 Introduction and Business Case

The objective of this project is to build an end-to-end movie recommendation system capable of suggesting relevant films to users based on their preferences. In contrast to a rating-prediction task, the goal here is to focus on content similarity: instead of predicting whether a user will like a movie, the system identifies films that are similar to those already selected.

The motivation behind this project comes from the growing importance of personalized recommendation engines in digital platforms, particularly in entertainment. Streaming services such as Netflix, Amazon Prime, and Disney+ rely heavily on recommender systems to improve the user experience, increase engagement, and maximize content consumption.

This project therefore aims to replicate, on a smaller scale, the logic behind such systems by combining techniques from natural language processing, machine learning, and information retrieval. The system analyses movie metadata such as genres, keywords, cast, crew, and plot summaries to build a numerical representation of each movie. These representations are then used to compute similarity between films and generate personalized recommendations.

The second part of the project explores classification models to understand whether movie metadata can be used to predict whether a film will be well-rated, defined as a vote average of 7 or above. Finally, the project includes a Streamlit web application that allows users to interact with the recommender system in real time.

2 Dataset Description

The project uses metadata originating from the **TMDb (The Movie Database)** public dataset. Three CSV files are provided:

- `movies_metadata.csv`: Contains the main attributes of each film, including title, overview, genres, popularity, release date, vote information, etc.
- `credits.csv`: Contains cast and crew information for each movie.
- `keywords.csv`: Contains descriptive keywords associated with each film.

The dataset originally contains over 45,000 movies, but many entries are incomplete, duplicated, or irrelevant. After cleaning, merging, and filtering — in particular removing entries without an overview — the final dataset contains approximately 42,000 usable movies. The most recent movie in the dataset was released in December 2020, so all movies after this date are not available.

For the collaborative filtering component, a second dataset is used:

- `ratings_small.csv`: Contains 100,000 user–movie ratings, used for SVD-based recommendation.

This combination of content metadata and user interaction data allows the construction of both content-based and hybrid recommendation systems.

3 Data Cleaning and Preprocessing

Significant preprocessing was required before building the models. The metadata files include corrupted IDs, inconsistent formats, nested JSON fields stored as strings, and missing values. A full cleaning pipeline was implemented.

3.1 Cleaning Movie Metadata

The following steps were applied:

- Removal of irrelevant columns (`homepage`, `poster_path`, `adult`, `imdb_id`, `video`).
- Conversion of numerical fields such as `budget`, `revenue`, `popularity`, `runtime` with `pd.to_numeric`.
- Conversion of release dates to datetime and extraction of release year.
- Removal of duplicate movies based on title.
- Handling missing values:
  - Numeric values replaced with median or mean depending on the variable.
  - Categorical values replaced with "Unknown".
  - Budgets or revenues equal to zero replaced by the dataset median.

3.2 Cleaning Nested JSON Attributes

Columns such as `genres`, `keywords`, `cast`, and `crew` contain JSON objects converted to strings. We parsed them using:

```
ast.literal_eval()
```

We extracted:

- Genre names
- Keyword names
- Top 3 most important cast members (based on "order")
- Director name from crew list

Each extracted attribute was converted to lowercase and formatted consistently (spaces replaced with "_").

3.3 Merging Datasets

All three files were merged using the movie ID, after ensuring that the IDs were numeric. After merging and cleaning, the final dataframe used for content-based modeling contained:

- Title

- Overview
- Genres list
- Keywords list
- Cast list
- Director
- Popularity
- Vote average
- Release year

This structured representation was used for all subsequent modeling steps.

4 Exploratory Data Analysis

Exploratory analysis confirmed large variation between movies:

- Popularity scores follow a long-tail distribution.
- Vote averages cluster around 6.0–6.5.
- Genres and keywords show strong sparsity.
- Some directors and actors appear in hundreds of films, while most appear only once.

Missing values were mostly concentrated in:

- `belongs_to_collection`
- `tagline`
- `release_date`

Visualizations such as histograms, barplots of genres, and distribution curves showed clear patterns that support the relevance of content-based recommendations. For example, action films tend to share recurring keywords such as *battle*, *violence*, or *military*, while romantic films consistently include terms such as *relationship* or *love*.

5 Formalization of the Problem

5.1 Content-Based Similarity Problem

Given a movie m, represented by a feature vector $\mathbf{x}_m$, the goal is to retrieve the movies whose vectors are closest to $\mathbf{x}_m$.

Let: $\mathbf{x}_i$ = feature vector of movie i

Similarity is computed using cosine similarity:

$$\text{cosine\_sim}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{x}_i \cdot \mathbf{x}_j}{\|\mathbf{x}_i\| \|\mathbf{x}_j\|} \tag{1}$$

The problem therefore reduces to:

$$\text{Recommend}(m) = \arg\max_{j \neq m} \text{cosine\_sim}(\mathbf{x}_m, \mathbf{x}_j) \tag{2}$$

5.2 Classification Problem

A second task consists of predicting whether a film is "well-rated":

$$y = \begin{cases} 1 & \text{if vote\_average} \geq 7 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

The challenge is the imbalance between classes, since most of the movies have ratings below 7.

6 Model Presentation

6.1 SVD Collaborative Filtering

Using the small ratings dataset, an SVD model was trained:

$$R \approx U\Sigma V^T \tag{4}$$

where U represents users in latent space, and V represents movies. This allows estimating how much a user will like a movie they have not rated.

RMSE evaluation confirmed the model's reliability for preference estimation.

6.2 Baseline TF-IDF Content Model

The simplest recommender was built using the movie overview. TF-IDF vectorization transforms the corpus into a sparse high-dimensional matrix:

$$\text{TF-IDF}(t, d) = \text{tf}(t, d) \cdot \log\left(\frac{N}{\text{df}(t)}\right) \tag{5}$$

where:

- $\text{tf}(t, d)$ = frequency of term t in document d
- N = number of documents
- $\text{df}(t)$ = number of documents containing t

Cosine similarity on this matrix provides a baseline recommendation system capable of retrieving thematic similarity.

6.3 Multi-Feature Content Model

To capture more nuanced relationships, the system incorporates four additional feature spaces:

- TF-IDF (overview)
- CountVectorizer for genres
- CountVectorizer for keywords
- CountVectorizer for cast and director (crew_list)

These matrices are horizontally stacked:

$$X = [X_{\text{tfidf}} \mid X_{\text{genre}} \mid X_{\text{keywords}} \mid X_{\text{crew}}] \tag{6}$$

6.4 Interactive Weight Analysis of the Content-Based Model

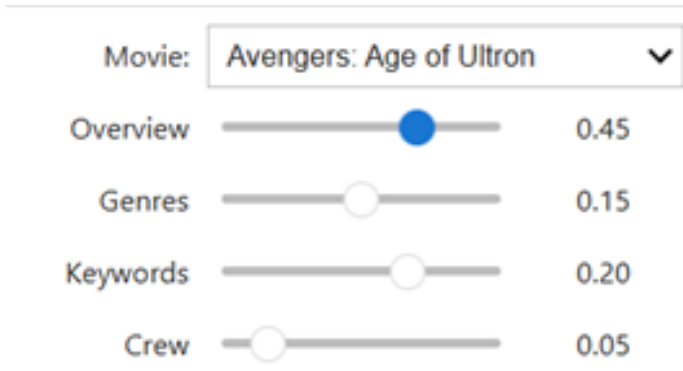
To better understand the influence of each content feature on the recommendation results, an interactive analysis tool was implemented using `ipywidgets`. This component allows a dynamic exploration of the weighted content-based model by manually adjusting the relative importance of each feature space.

A representative subset of 20,000 movies was first sampled from the cleaned dataset to ensure responsiveness while preserving sufficient diversity. For this subset, the previously trained vectorization models were reused to compute four feature matrices: TF-IDF vectors for movie overviews, and CountVectorizer representations for genres, keywords, and crew information.

Four sliders were then introduced to control the weights associated with each feature type: overview, genres, keywords, and crew. These weights are applied directly to the corresponding feature matrices before horizontally stacking them into a single combined representation. A Nearest Neighbors model using cosine distance is fitted on this weighted feature space.

When a movie is selected from the dropdown menu, the system recomputes similarity scores in real time and displays the top recommended movies along with their vote average and popularity. By adjusting the sliders, it becomes possible to observe how emphasizing narrative content, thematic categories, or personnel information affects the recommendation output.

This interactive analysis provides an intuitive illustration of the model's internal behavior and highlights the role of feature weighting in shaping recommendation relevance. It also supports interpretability by making explicit how different semantic dimensions contribute to similarity computation.



	title	vote_average	popularity
2925	Captain America: Civil War	7.1	145.882135
2926	Captain America: The Winter Soldier	7.6	18.717704
7544	Iron Man 3	6.8	23.721243
18154	Thor: The Dark World	6.8	34.905447
7543	Iron Man 2	6.6	19.083344

6.4.1 Example: Avengers: Age of Ultron

For example, when selecting *Avengers: Age of Ultron* and assigning a higher weight to the movie overview, the system primarily recommends other films from the Marvel Cinematic Universe such as *Captain America: Civil War*, *Iron Man 3*, and *Thor: The Dark World*.

This result illustrates the model's ability to capture narrative continuity and shared fictional universes, rather than relying solely on genre labels. The example highlights how adjusting feature weights allows the recommendation system to emphasize storytelling elements over production metadata, leading to coherent and interpretable recommendations.

6.5 Nearest Neighbors Recommender

Once the combined feature matrix was built, a `NearestNeighbors` model was trained:

```
nn_model = NearestNeighbors(n_neighbors=n_neighbors_best,
                            metric=metric_best,
                            algorithm='brute')
nn_model.fit(X_final)
```

With the best parameters being the following: 5 for the number of neighbors and 'cosine' as the best metric. This enabled instant retrieval based on similarity in the feature space.

A weighted combination was introduced to tune the importance of each feature, and a hyperparameter tuning using GridSearch was performed over:

- Weights of each feature block
- Number of neighbors
- Similarity metric (cosine or Euclidean)

```
Best parameters found: (0.3, 0.2, 0.1, 0.1, 5, 'cosine')
```

The first four parameters are for the different weights, while the last two ones are dedicated to the Nearest Neighbor recommender.

This significantly improved recommendation quality, measured by genre overlap.

6.6 Hybrid Recommender

A hybrid model was created by combining two complementary signals:

- Content similarity, capturing how close a movie is in terms of narrative and metadata
- SVD-based predicted rating, reflecting the user's personal taste patterns

$$\text{FinalScore} = \alpha \cdot \text{ContentScore} + (1 - \alpha) \cdot \text{SVDScore} \tag{7}$$

Here, the hybrid formulation helps the system recommend movies that are both:

- Contextually relevant, based on their intrinsic features
- Aligned with user preferences, through collaborative filtering

This combination leverages the strengths of both approaches and generally leads to more balanced and personalized recommendations.

6.7 Classification Models

To explore whether metadata can predict movie success, the following classifiers were trained:

- SVM (linear kernel)
- Decision Tree
- Bagging (ensemble of decision trees)

Performance was evaluated using confusion matrices, precision, recall, and F1-score. Results showed:

```
=== SVM ===
Confusion Matrix:
 [[6375  830]
 [1481  441]]
F1-score: 0.27622925148762917
              precision    recall  f1-score   support

           0       0.81      0.88      0.85      7205
           1       0.35      0.23      0.28      1922

    accuracy                           0.75      9127
   macro avg       0.58      0.56      0.56      9127
weighted avg       0.71      0.75      0.73      9127


=== Decision Tree ===
Confusion Matrix:
 [[5829 1376]
 [1401  521]]
F1-score: 0.27284629484158157
              precision    recall  f1-score   support

           0       0.81      0.81      0.81      7205
           1       0.27      0.27      0.27      1922

    accuracy                           0.70      9127
   macro avg       0.54      0.54      0.54      9127
weighted avg       0.69      0.70      0.70      9127


=== Bagging ===
Confusion Matrix:
 [[6846  359]
 [1678  244]]
F1-score: 0.19326732673267327
              precision    recall  f1-score   support

           0       0.80      0.95      0.87      7205
           1       0.40      0.13      0.19      1922

    accuracy                           0.78      9127
   macro avg       0.60      0.54      0.53      9127
weighted avg       0.72      0.78      0.73      9127
```

Performance was evaluated using confusion matrices, precision, recall, and F1-score. However, predicting movie quality remains difficult due to sparsity and imbalance.

7 Obstacles and How They Were Addressed

Several challenges were encountered:

7.1 Incomplete or Corrupted Data

Many movies had missing fields, causing errors when vectorizing. The solution was:

- Remove rows with missing overviews
- Replace invalid JSON with empty lists
- Normalize text fields

7.2 Sparse High-Dimensional Vectors

TF-IDF created vectors with over 5,000 dimensions. The model remained efficient thanks to:

- Sparse matrix representations
- NearestNeighbors using brute-force cosine similarity

7.3 Class Imbalance in Rating Prediction

Most movies are not well-rated. Strategies used:

- Stratified train-test split
- Weighted F1-score as the main metric

7.4 Overfitting in Decision Trees

Trees captured noise rather than signal. Bagging and Stacking helped stabilize performance.

7.5 Web Deployment Constraints

Deploying the model with Streamlit required additional performance optimizations to ensure a responsive user experience. Running similarity computations on the full dataset of over 40,000 movies caused noticeable latency. To address this, the system was restricted to a subset of approximately 10,000 movies, significantly reducing computation time while preserving recommendation quality. Feature vectorization and similarity structures were also precomputed to allow near-instant responses during user interaction.

8 Comparison of Models

Let us compare the results of the different models used.

8.1 Classification Performance

Table 1: Classification Model Performance

Model	Accuracy	F1-score (class 1)
SVM	0.75	0.28
Decision Tree	0.70	0.27
Bagging	0.78	0.19
Stacking	0.78	0.20

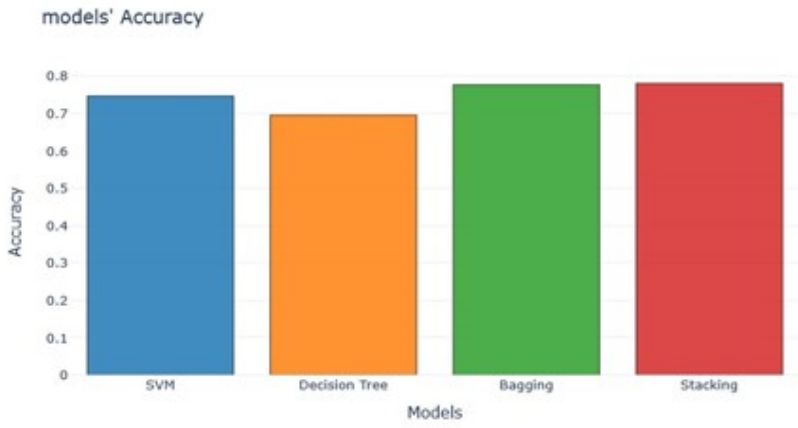


Figure 1: Models' Accuracy comparison across different classification approaches

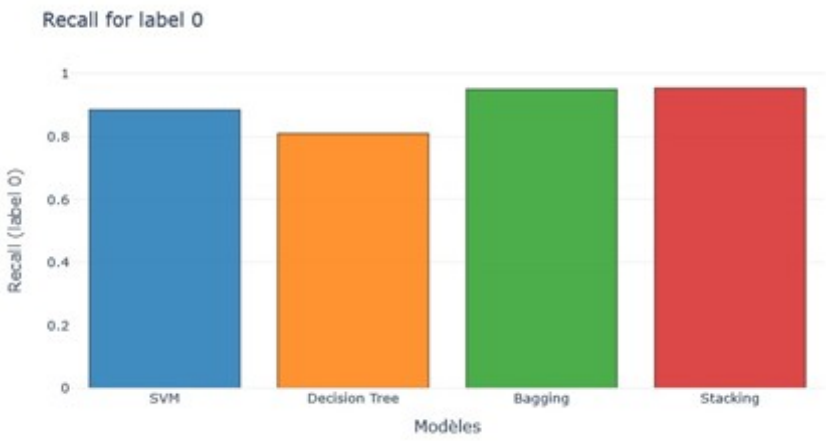


Figure 2: Recall for label 0 (poorly-rated movies)

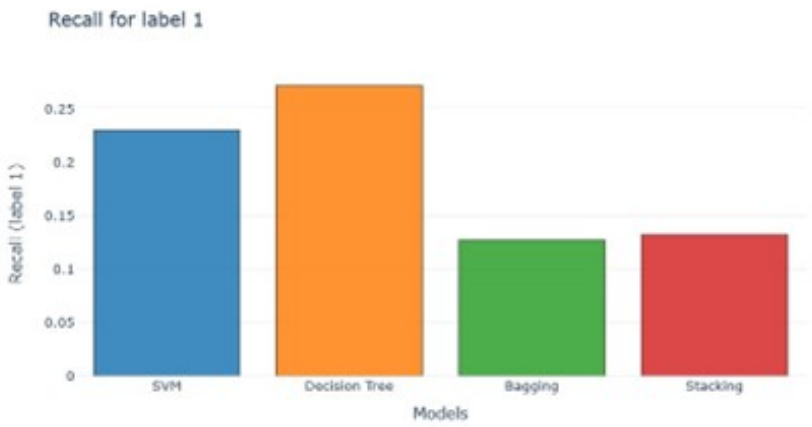


Figure 3: Recall for label 1 (well-rated movies)

Looking at the table of results, the stacking meta-model seems to perform similarly to the Bagging classifier in terms of accuracy. Both models reach the highest accuracy values among the different classifiers, which might initially suggest that they are the most reliable. However, when shifting the focus to the F1-score, the situation becomes different. The SVM clearly stands out with the highest F1-score, meaning it achieves a more balanced trade-off between recall and precision for the positive class. This indicates that while its accuracy might not be the very best, it handles the detection of well-rated movies in a more consistent way than the other models.

When examining the plots, it becomes apparent that the meta-model behaves almost exactly like the Bagging model. This suggests that the stacking approach is overly influenced by the predictions of Bagging. Since Bagging struggles to correctly identify true positives, especially for the positive class, this weakness gets transferred into the meta-model. As a result, the stacking model inherits Bagging's limitations and ends up performing worse on the aspects where identifying actual positives is important.

Overall, the F1-scores are quite low (below 0.3), which indicates that none of the models are performing well at distinguishing well-rated movies from poorly rated ones using metadata alone. This suggests that while the metadata can be helpful for exploratory analysis or understanding general patterns, it is not strong enough to produce high-quality classification results on its own. Adding richer information, such as more detailed textual, visual, or contextual features, would likely be necessary to improve the predictive performance.

Overall, the F1-scores are quite low (below 0.3), which indicates that none of the models are performing well at distinguishing well-rated movies from poorly rated ones using metadata alone. This suggests that while the metadata can be helpful for exploratory analysis or understanding general patterns, it is not strong enough to produce high-quality classification results on its own. Adding richer information, such as more detailed textual, visual, or contextual features, would likely be necessary to improve the predictive performance.

9 Streamlit Web Application

To make the recommendation system accessible beyond the Jupyter notebook environment, we developed a Python application using Streamlit. This application (`movie_rec_app.py`) was created as a separate file in an independent project folder, allowing for deployment and user interaction outside of the notebook-based development workflow. The web app replicates the core functionality of our recommendation engine while providing an intuitive graphical interface. The application will allow users to:

A fully interactive web app was built using Streamlit.

The application allows users to:

- Search for a movie using a dynamic search bar
- Add multiple liked movies
- Receive recommendations instantly
- View similarity scores, IMDb rating, genres, and summary
- Explore data in a clean, intuitive interface

The application transforms the notebook models into a real, usable product.

Technically, the app loads:

- The processed dataframe
- The TF-IDF vectorizer
- The cosine similarity matrix
- A real-time recommendation function

This showcases how machine learning models can be deployed to end users.

9.1 Example of Recommendations

Once the application is launched, users interact with a simple and intuitive interface designed to make movie discovery easy. The homepage invites users to select movies they enjoy through a search bar. As a movie title is typed, relevant results appear and can be added to the selection list.

Selected movies are displayed in a dedicated section where users can remove individual titles or clear all selections at once. A slider located on the left side of the interface allows users to choose how many recommendations they want to receive, offering flexibility between 5 and 20 suggested movies.

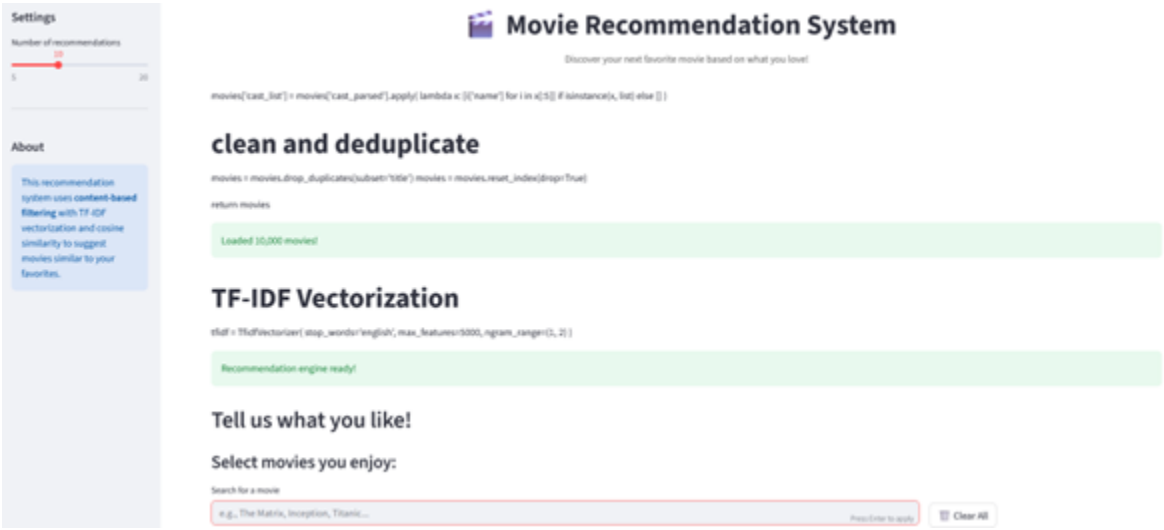


Figure 4: Streamlit web application homepage with movie selection interface

After confirming their choices, users click the "Get Recommendations" button. The system then generates a personalized list of recommended movies based on content similarity, taking into account elements such as plot overview, genres, cast, and director.

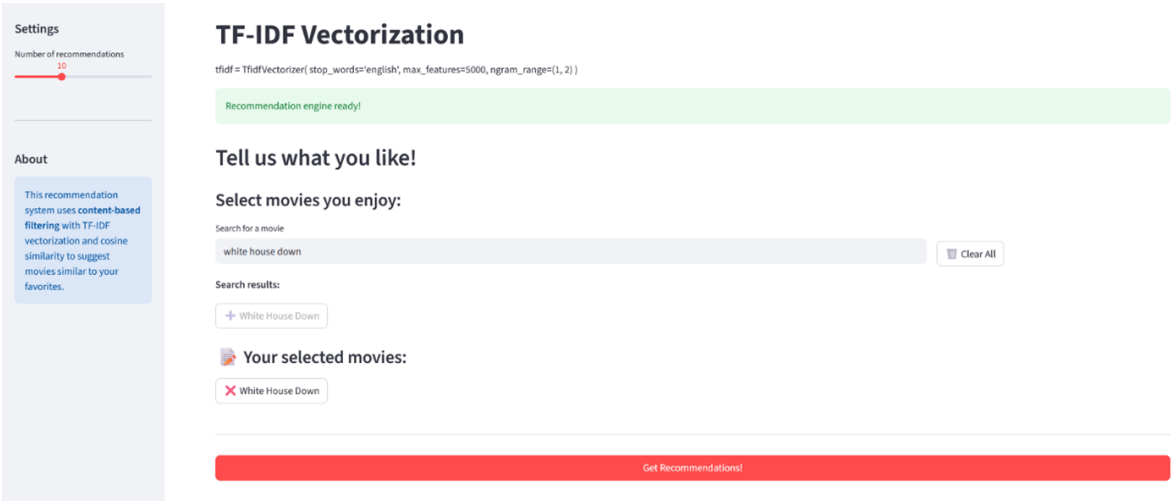


Figure 5: Movie selection interface showing search results and selected movies list

Each recommended movie is presented with key information including the title, a similarity match percentage, the average rating, genres, and release year, allowing users to quickly understand why the movie was suggested.

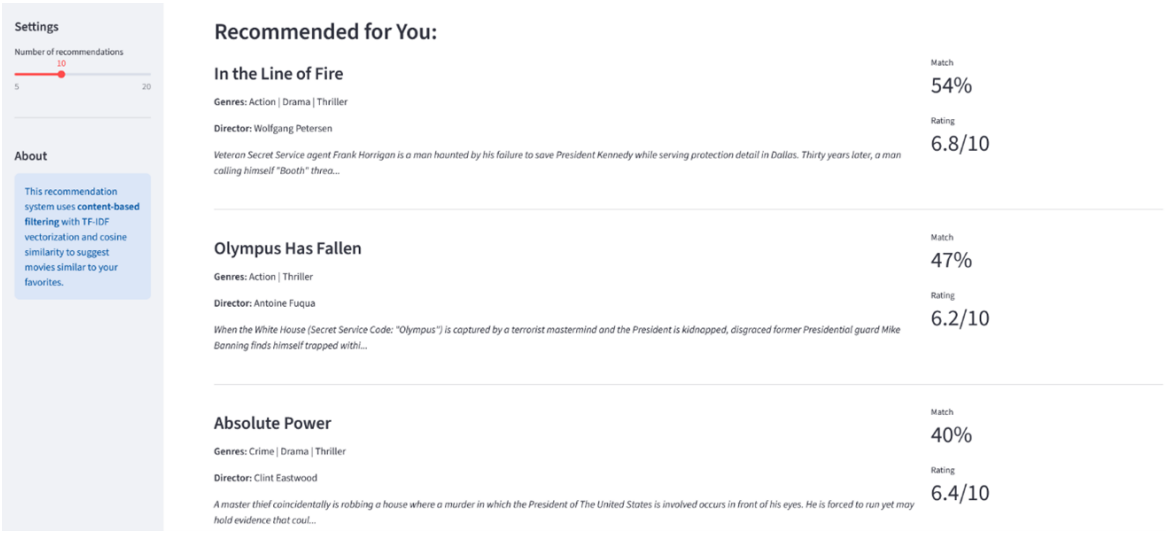


Figure 6: Recommendation results for "White House Down" showing top matching films with similarity scores and ratings

9.1.1 Case Study: White House Down

User Input: Selected *White House Down* (action–thriller).

Top Recommendations:

- *Olympus Has Fallen* (strong thematic match: political tension, action, terrorism).
- *In the Line of Fire* (similar genre: action, political drama, presidential protection).
- *Absolute Power* (less action-focused, but still relevant with political intrigue and crime).

Match Scores:

- High match scores for films closely related in themes and plot.
- Lower match for *Absolute Power*, but still contextually relevant.

Analysis:

- System effectively identifies and recommends movies with similar plots, genres, and themes.
- Recommendations are meaningful and aligned with the selected movie's narrative.

10 Conclusion

The project successfully demonstrates how content-based and collaborative filtering techniques can be used to build a personalized movie recommendation system. The final system integrates advanced preprocessing, several machine learning models, and an interactive interface, producing a complete workflow from raw data to a user facing application.

The content-based model performs well in capturing thematic similarity, while the hybrid approach enhances the personalization by taking user preferences into account. The Streamlit web application provides a practical and accessible interface, showing how the system could be integrated into a real-world platform.

Future improvements could include:

- sentiment analysis of reviews
- transformer-based embeddings (e.g., BERT) for movie descriptions
- large-scale user rating datasets to strengthen collaborative filtering
- deployment through a public cloud platform

Overall, the project meets its objectives and offers a strong foundation for more sophisticated recommendation engines.

11 Scientific References

- Zhang, S., Yao, L., Sun, A., & Tay, Y. (2019): *Deep Learning Based Recommender System: A Survey*
- Aggarwal, C. (2016). *Recommender Systems: The Textbook*
- TMDb Official Documentation

12 GitHub Link

https://github.com/13MY12/ML_MoviesRecommendation