# Learning vs. Heuristic Scoring for Content-Based Movie Recommendation Using Semantic and Metadata Features

Anna Vadella
Michigan State University
Department of Computer Science
East Lansing, MI, USA
vadellaa@msu.edu

Isha Atteri
Michigan State University
Department of Computer Science
East Lansing, MI, USA
atteriis@msu.edu

Prabhaav Ravikumar Pillai
Michigan State University
Department of Computer Science
East Lansing, MI, USA
prabhaav@msu.edu

## Abstract

The goal of this project is to develop a movie recommendation system by comparing two approaches: a supervised learning model and a feature-based heuristic method. With the rapid growth of streaming platforms and the overwhelming number of available titles, semantically meaningful recommendations have become highly useful. We integrate Wikipedia-scraped movie metadata and the Netflix prize dataset into a unified data pipeline, used both to train an XGBoost regression model and define a weighted heuristic similarity function. Evaluated on 500,000 movie pairs (60%/40% train/eval split) at a minimum of 150 co-raters per pair, the XGBoost model achieves Pearson $r = 0.185$ and Spearman $\rho = 0.159$ on the evaluation set, roughly 2.7× and 1.5× the heuristic's 0.068 and 0.103. Recall@100 improves from 0.195 (heuristic) to 0.293 (model) on the evaluation set, though up until $K = 50$ the heuristic is actually competitive. Recommendations are assessed using correlation and ranking-based metrics. An interactive prototype and source code are available at this link and this GitHub repository.

## Keywords

Movie recommendation, collaborative filtering, content-based filtering, text embeddings, cosine similarity, Netflix Prize, NLP

## 1 Introduction

With the rapid growth of digital media platforms, users are exposed to an overwhelming number of movies, making it increasingly difficult to discover content that aligns with their preferences. The goal of this project is to design a movie recommendation system that identifies similar movies based on user ratings and structured features.

(1) Recommendation systems play a critical role in helping to alleviate choice paralysis. Data mining techniques helps in identify patterns and extracting meaningful representations across large heterogeneous datasets. In the context of movies, plot summaries and other metadata, along with user ratings, provide information that can be utilized to generate recommendations. A key challenge lies in evaluating the performance of the recommendation system, as it is an inherently subjective problem. Another challenge lies in identifying reliable data sources with accurate information that also permit web scraping, as most websites have restrictions against automated data collection.

(2) This project involves the construction of a supervised learning framework in which collaborative similarity, derived from Netflix user rating data, is used as ground truth. We extract and preprocess movie metadata, generate semantic embeddings for plot descriptions using a transformer-based model, and compute structured similarity features such as genre overlap, cast similarity, director match, and temporal proximity. These features are used in two approaches: (1) a supervised XGBoost regression model trained to predict collaborative similarity, and (2) a feature-based heuristic scoring function that computes similarity directly without learning. The project evaluates and compares these approaches using statistical correlation and ranking-based metrics.

Unlike traditional hybrid systems that combine multiple methods into a single model, this work isolates key components of a hybrid framework within a unified pipeline to allow for direct side-by-side comparison. This design allows us to better understand how different similarity formulations influence recommendation quality and ranking behavior.

(3) The main contributions of this project are:
- Construction dataset by combining content features with collaborative similarity derived from user ratings.
- Integration of unstructured text embeddings and structured metadata into a unified feature representation for similarity calculation.
- Development of a regression-based model to predict user preference similarity from content features.
- Implementation of a feature-based heuristic baseline for comparison with the learned model.
- Empirical analysis comparing supervised and heuristic approaches using correlation and ranking-based evaluation metrics.

Our experiments demonstrate that the XGBoost model achieves a Pearson $r = 0.185$ and Spearman $\rho = 0.159$ on the evaluation set, outperforming the heuristic baseline by approximately 2.7× and 1.5×, respectively. Recall@100 improves from 0.195 (heuristic) to 0.293 (model), with the model taking a clear lead from $K$=20 onwards. These findings confirm that supervised learning on content-based features can meaningfully approximate user-preference similarity, particularly for higher-$K$ ranking tasks.

The remainder of this paper is organized as follows: Section 2 reviews related work; Section 3 defines the problem statement; Section 4 describes the methodology, including data collection, preprocessing, feature engineering, and model development; Section 5 presents the experimental evaluation, including dataset

statistics, correlation and ranking metrics, feature importance, and an error analysis; Section 6 concludes and outlines future work.

## 2 Related Work

Recommendation systems have been extensively studied, with prior work spanning a multitude of different paradigms. For this project, we focus on two of the three main types of filtering methods: collaborative filtering and content-based filtering. Each approach has its own strengths and limitations, and many real-world systems combine multiple techniques to provide better recommendations.

### 2.1 Collaborative Filtering

Collaborative filtering methods leverage user-item interaction data in order to identify patterns in user behavior and generate recommendations. Some methods compute similarities between users or items using measures like cosine similarity [11], while other methods learn latent factors from user-item matrices using techniques such as matrix factorization [7]. Collaborative filtering is widely adopted in industrial platforms like Netflix and Amazon to recommend content based on user ratings, reviews, and observed viewing or purchasing patterns, using both explicit and implicit feedback [12].

The Netflix Prize challenge demonstrated the scalability and effectiveness of model-based collaborative filtering for large-scale rating prediction [7]. While collaborative filtering is effective at capturing implicit user preferences through observed user behavior, it heavily relies on the availability of sufficient interaction data. As a result, it suffers from several well-known limitations like sparsity issues and the cold-start problem, where insufficient data prevents accurate recommendations [5, 9]. These challenges have motivated research into hybrid methods that incorporate content features to improve overall accuracy and performance of recommendations [5, 10].

### 2.2 Content-Based Filtering

Recommendations from content-based filtering are generated by analyzing item attributes and matching them to a user's preferences without relying on data from other users [5, 9]. For movies, these item attributes are represented by features like genre, director, cast, or plot summaries, and recommendations are made by matching these to a user's past preferences [10, 12].

Modern content-based filtering approaches often represent textual features using methods like Bag-of-Words (BoW), which produces sparse vector representations that can be used with machine learning models like logistic regression or support vector machines for tasks such as genre classification. More recent approaches leverage semantic embeddings, which enable similarity computations using measures such as cosine similarity, allowing recommendations to reflect narrative and thematic similarities between movies.

Supervised learning methods, such as XGBoost, can further combine these embedding-based similarities with structured metadata to predict pairwise movie similarity [3]. Additionally, because movies often belong to multiple genres, content-based systems often use models designed for multi-label classification [6]. Clustering and similarity-based algorithms, including K-means and K-nearest neighbors, are frequently used to group similar items and identify the closest matches to a user's profile [1].

These methods allow for content-based systems to generate accurate, semantically meaningful recommendations while maintaining scalability. Advantages of content-based filtering include the ability to recommend niche items or items with little interaction data and improved interpretability. Limitations include over-specialization, where recommendations closely resemble previously liked items [5].

### 2.3 Combining Content and Collaborative Filtering

Although this project does not implement a traditional hybrid filtering approach, prior research has shown that combining content-based and collaborative methods can improve recommendation quality [10, 12]. Hybrid filtering systems typically integrate collaborative filtering and content-based filtering approaches by combining predictions from separate models or incorporating content features into collaborative frameworks [9, 10]. These methods are particularly effective at addressing limitations such as cold-start and sparsity that often affect single-method systems.

Rather than implementing a full hybrid architecture, this project adopts a pairwise supervised learning approach that incorporates both content-based and collaborative methods. We train a regression model to predict similarity between movie pairs using content-derived features – such as plot embeddings and cast similarity – while using a collaborative similarity derived from user ratings to supervise the model. This approach allows the model to learn how content features relate to user-driven similarity patterns, effectively bridging content-based and collaborative information without explicitly combining separate recommendation systems.

Prior work demonstrates that hybrid recommendation systems can improve accuracy and address limitations of single-method approaches, resulting in more effective and personalized recommendations. Building on these insights, this project adopts a pairwise supervised learning framework that integrates both content-derived features and a collaborative similarity measure derived from user rating data. Rather than explicitly combining separate recommendation systems, this approach enables a unified evaluation of content-based and collaborative techniques within a single learning framework, allowing for the complementary strengths of both approaches to be assessed.

## 3 Problem Statement

The objective of this project is to design a content-based movie recommendation system that identifies and suggests similar movies based on two methods: an objective algorithm and a subjective model. The problem domain involves analyzing movie-related data to uncover meaningful relationships between movies.

The dataset consists of two components: (1) movie metadata, including plot descriptions, genre, cast, director, and release date, constructed by combining publicly available information from IMDb with data extracted from Wikipedia through web scraping,

and (2) user rating data from the Netflix dataset. The metadata includes both unstructured text (plot) and structured attributes, while the rating data is used to derive a collaborative similarity signal between movie pairs.

Formally, the task is defined as a supervised regression problem over movie pairs. For each pair, a feature vector is constructed using content-based similarity measures, including semantic similarity of plot embeddings, genre overlap, cast similarity, director match, and temporal proximity. The target variable is defined as the collaborative similarity between the two movies, computed using centered cosine similarity over user ratings.

The objective is to learn a function that predicts user-driven similarity from content features. In parallel, a heuristic scoring function is defined as a weighted combination of the same features, without supervision.

Thus, the project involves two data mining tasks:

- Supervised learning (regression): learning a mapping from content features to collaborative similarity.
- Similarity analysis (heuristic modeling): computing pairwise similarity using predefined feature weights.

The performance of these approaches is evaluated to determine how effectively content-based features can approximate user preference similarity

# 4 Methodology

## 4.1 Data Sources

The dataset integrates three primary sources.

(1) **IMDb Non-Commercial Dataset**: The `title.basics.tsv` file provides titles, release years, runtime, and genre labels for approximately 600,000 entries of type `movie`. This dataset does not include plot summaries, so we use it to drive Wikipedia lookups rather than as a content source itself [4].

(2) **Wikipedia (parsed from ZIM)**: Rather than issuing live API calls, we mount a local English Wikipedia ZIM snapshot via the `libzim` Python library, enabling rapid offline article retrieval. For each IMDb entry, we issue a search query of the form "⟨title⟩ (⟨year⟩ film)" and examine up to the first five hits, accepting the first one whose infobox matches on year and contains the expected "Directed by" plus "Produced by" or "Written by" fields. BeautifulSoup parses the matched HTML to extract the `Plot/Synopsis` section from the article body and structured metadata (director, release date, up to the first five credited cast members, poster filename) from the infobox. Genre is extracted from the introductory sentence of the article body via a regular expression match on the pattern "is a ⟨adjective phrase⟩" [13]. After scraping and preprocessing, 75,636 movies with complete metadata remain.

(3) **Netflix Prize Dataset**: Provides approximately 100 million user ratings for 17,770 movies on a 1–5 scale. Netflix titles are aligned to our Wikipedia-derived catalog using `rapidfuzz` fuzzy matching with a string-similarity cutoff of 85, backed off to a $\geq 70$ score plus an exact director-and-year match. After filtering to pairs with at least 150 shared raters, 4, 294 matched movies enter the pairwise training/evaluation sets (Section 5). These 4,294 movies form the complete basis for all model training and evaluation: each movie carries five content features, plot cosine similarity (computed from 384-dim Sentence-BERT embeddings), genre Jaccard similarity, cast Jaccard similarity, a binary director-match indicator, and normalized year proximity, alongside a collaborative-similarity target derived from centered-cosine over the user–movie rating matrix. Together, these features and labels constitute the dataset on which the XGBoost regressor is trained (300,000 pairs) and on which both the model and heuristic are evaluated (200,000 pairs); the ratings provide the co-rating ground-truth signal used to supervise the learned model and to evaluate both methods [8].

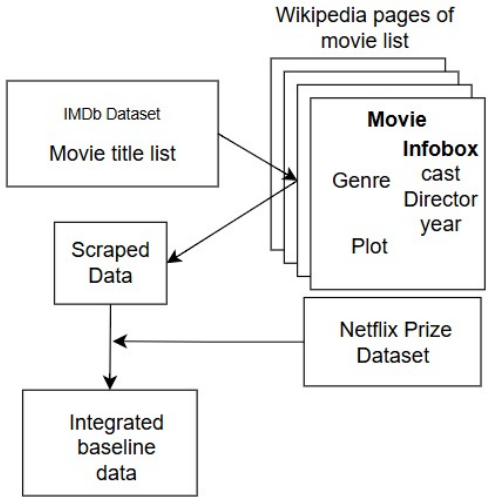


**Figure 1: Data collection and construction.**

## 4.2 Data Parsing and Dataset Construction

The raw dataset is constructed by scraping Wikipedia movie pages in the IMDb dataset. For each movie, the HTML content of the Wikipedia page is parsed using BeautifulSoup. The structured layout of the page is leveraged to extract specific fields: the plot section from the main content body, and metadata such as director and cast from the infobox. Genre information is extracted from the introductory sentence of the page.The missing entries and irregularities are addressed by filtering out rows with missing critical fields (plot and genre) and standardizing the remaining data.

User interaction data was obtained from the Netflix dataset, consisting of user–movie rating triples. To integrate the two sources, a fuzzy matching procedure based on string similarity was applied to align Wikipedia movie entries with Netflix movie identifiers. Additional constraints, such as release year consistency, were used to improve matching accuracy.

The final integrated dataset consists of a set of movies with associated metadata and aligned user ratings.

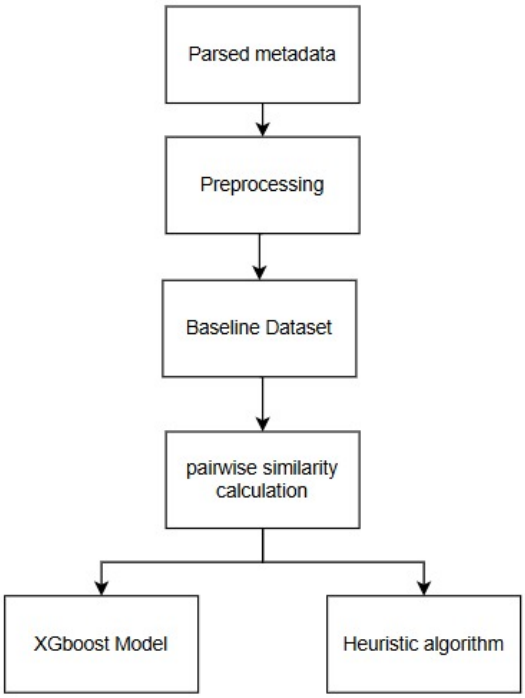


**Figure 2: Methodology pipeline.**

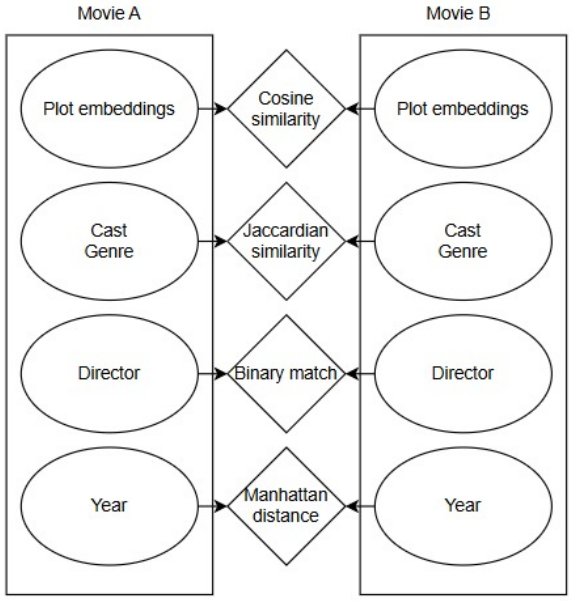


**Figure 3: Pairwise similarity calculation.**

## 4.3 Data Preprocessing and Feature Engineering

Raw movie metadata was preprocessed to standardize both unstructured and structured features. Plot text was normalized through lowercasing, punctuation removal, tokenization, stopword removal, and stemming/lemmatization to reduce noise and dimensionality. Structured attributes such as genre, director, and cast were cleaned and converted into consistent formats.

Semantic representations of plot text were generated using the Sentence-BERT model (all-MiniLM-L6-v2), which produces 384-dimensional dense vector embeddings capturing contextual similarity between movie plots. Additional feature engineering was performed to derive pairwise similarity measures, including cosine similarity for embeddings, Jaccard similarity for genre and cast overlap, binary matching for directors, and normalized temporal proximity based on release year differences.

These preprocessing and feature engineering steps transform heterogeneous raw data into a unified feature space suitable for similarity modeling and supervised learning.

## 4.4 Pairwise Dataset Construction

A pairwise dataset was constructed by generating movie–movie pairs and associating each pair with both content-based features and a collaborative similarity target. User rating data was transformed into a movie–user matrix, where ratings were mean-centered per user to remove individual bias. Collaborative similarity between movie pairs was then computed using cosine similarity over the centered ratings, providing the ground truth signal.

Movie pairs with a minimum number of shared users (co-raters) were retained, reducing noise in similarity estimation. For each valid pair, a feature vector was constructed using plot embedding similarity, genre overlap, cast similarity, director match, and year proximity.

To reduce noise in the similarity estimate, only pairs with at least MIN_CO_RATERS = 150 shared raters are retained. From the surviving candidates we sample up to MAX_PAIRS = $5 \times 10^5$ pairs and split 60%/40% into training and evaluation sets (fixed seed 42), yielding 300,000 training and 200,000 evaluation pairs over 4,294 distinct movies. The resulting dataset consists of labeled movie pairs with the five content features from Table **??** and the collaborative-similarity target. Algorithm 1 summarizes the end-to-end pipeline.

## 4.5 Model Development

A supervised regression model was developed to predict similarity. The model takes as input the pairwise feature vector and learns a mapping to the target similarity using an XGBoost regressor, which is well-suited for handling heterogeneous feature types and non-linear relationships. The fitted model and the list of feature columns are serialized with *joblib*.

In parallel, a feature-based heuristic scoring function was implemented as a baseline with fixed-weights as specified in Table 3. This method computes similarity as a weighted linear combination of the same features used in the supervised model, with predefined weights assigned to each feature (e.g., plot similarity, genre overlap, cast similarity, director match, and year proximity).

**Table 1: End-to-end recommendation pipeline**

1. For each IMDb movie, query local Wikipedia ZIM; accept first match where infobox year and required section headers validate.
2. Parse HTML with BeautifulSoup; extract plot, director, genre, up to 5 cast members, release date, poster URL.
3. Normalize plot: lowercase, strip punctuation/non-word, tokenize, remove stopwords, stem (Porter), lemmatize (WordNet).
4. Encode plots with Sentence-BERT `all-MiniLM-L6-v2` (384-dim); L2-normalize once.
5. Fuse with Netflix Prize: fuzzy-match titles (cutoff 85, director+year back-off at 70); persist Netflix–IMDb slug map.
6. Center ratings per user; build sparse movie×user matrix; compute cosine similarities and co-rater counts in bulk.
7. Filter pairs with $\geq 150$ co-raters; cap at $5 \times 10^5$ pairs; compute the five features per pair.
8. Split 60/40 (seed 42); train XGBoost regressor on the train fold to predict centered-cosine collaborative similarity.
9. In parallel, score each pair with the weighted heuristic (weights in Table 2).
10. Evaluate both scorers on train and eval folds with Pearson, Spearman, RMSE, and Recall@$K$ for $K \in \{5, 10, 20, 50, 100\}$.

The supervised model captures complex interactions between features through learning, while the heuristic approach provides a deterministic, interpretable baseline. These two approaches are subsequently compared to evaluate the recommendation system.

**Table 2: Heuristic scorer weights (sum to 1)**

| Feature | Weight |
|---|---|
| `plot_sim` | 0.30 |
| `genre_sim` | 0.20 |
| `cast_sim` | 0.20 |
| `year_sim` | 0.15 |
| `director_match` | 0.15 |

## 4.6 Prototype Development

The web prototype developed for this project is divided into two components: a frontend responsible for user interface and interaction logic, and a backend responsible for data management/storage and recommendation logic.

(1) Frontend
- **Framework:** The frontend of this prototype was built with Next.js and React, utilizing client-side rendering for interactivity and a responsive user interface.
- **Dynamic Routing:** Implements dynamic routes that resolve movie slugs into individual detail pages. These pages dynamically fetch and display movie posters, plots, and other metadata specific to the queried slug.
- **Key Features:**
  - **Dynamic Components:** Features include an expandable plot section, a session-based "like" system, and similarity scores to compare user-selected titles with recommendations. The home page features a gallery of user likes and personalized recommendations, as well as a search tool with a dropdown interface.
  - **Search Functionality:** The homepage features a search functionality that handles user queries and routes them to a page displaying user-specified movie metadata, and users can query ant movie within the database to view its detailed metadata.
- **Recommendation API (FastAPI):** FastAPI manages recommendation logic, decoupled from the main frontend and database for modularity. The API fetches movie data from the backend and passes the selected movie content to the specified recommendation component.
  - **Dual-mode Strategy:** The frontend supports a UI-controlled toggle between **Algorithm-Based** filtering (using standard content-based filtering) and **ML-Based** recommendations (using high-dimensional embeddings).

(2) Backend
- **Platform:** Supabase serves as the backend-as-a-service (BaaS), hosting the database containing our structured movie metadata.
- **Data Schema:** The Supabase database stores structured movie information such as titles, plots, genres, cast, directors, release dates, movie slugs, and movie posters.
- **Data Access:** The frontend queries the Supabase database directly through the Supabase client, and the data retrieval pattern involves fetching a single movie record by querying its unique slug.
- **Access Patterns:** The database supports two primary access patterns: direct user navigation and content personalization.
  - **Direct User Navigation:** When a user searches for a title on the homepage, the system resolves the query to a unique slug to route them to a dedicated page populated with detailed metadata about the requested film.
  - **Content Personalization:** The database facilitates discovery by fetching metadata and posters for the movie-specific recommendations and user-specific "likes" and "recommendations" galleries on the homepage.
- **Trade-offs:** Using Supabase allows for real-time database access with minimal configuration, prioritizing rapid development and real-time synchronization. Though efficient for read-heavy lookup patterns, it is optimized for simplicity rather than scaling.

*4.6.1 Deployment on Railway.* The full prototype was deployed to **Railway** (https://railway.app), a cloud-hosting platform that supports containerized multi-service deployments from a single

GitHub repository. The frontend and the FastAPI backend are provisioned as two separate Railway services, each rebuilt automatically on every push to the `main-combined` branch via Railway's built-in CI/CD pipeline. Environment variables (Supabase URL, API keys, and inter-service URLS) are injected at deploy time through Railway's environment variable management, keeping secrets out of the repository. The live prototype is accessible at https://resilient-truth-production-eaba.up.railway.app/. Railway's always-on, low-configuration hosting and free-tier persistent URLs made it well-suited for a research prototype requiring a stable public endpoint without dedicated infrastructure. A key limitation is the number of available credits, which are consumed per second while services are running. This isn't a major issue, since users can either run the entire frontend locally or even deploy it to Railway themselves.

# 5 Experimental Evaluation

## 5.1 Experimental Setup

**Dataset Characteristics** Our dataset integrates multiple sources. Each movie record contains seven structured fields: title, director (abbreviated as 'dir' in Table 1), cast, genre, plot (preprocessed), release date, slug (IMDb identifier), and the movie poster image file. The Netflix Prize subset includes an additional sparse user-rating vector with up to 17,770 user columns.

Table 3 summarizes the dataset before and after preprocessing. After cleaning and filtering, we retained approximately 75,000 movies with complete plots and standardized metadata. The overlap with the Netflix Prize dataset provides roughly 5,000 movies with ground truth ratings for evaluation with over 59,000,000 user ratings.

**Table 3: Dataset Statistics Before and After Preprocessing**

| Source / Stage | # Movies | Key Fields |
|---|---|---|
| IMDb (raw, type=movie) | ~600,000 | title, year |
| Wikipedia ZIM (matched) | ~110,000 | dir., cast, genre, plot, poster |
| After preprocessing | 75,636 | normalized metadata |
| Netflix Prize Ground Truth | 4,294 | ~59M user-movie ratings |

From these 4,294 movies, 500,000 candidate pairs are sampled and split 300,000 / 200,000 (60% / 40%) for training and evaluation (seed 42).

**Computing Platform** Experiments ran on a Windows 11 laptop: Lenovo Slim 9 Pro with an Intel i7-1375H CPU, 32GB RAM, and a GeForce RTX 4050 GPU. The GPU is used to accelerate embedding computation with the Sentence-BERT encoding via SentenceTransformers.

**Software** The pipeline is pure Python (3.11+) and uses `beautifulsoup4` + `lxml` for HTML parsing, `libzim` for offline Wikipedia access, `rapidfuzz` for fuzzy title matching, `nltk` for text normalization, `sentence-transformers` for embeddings, `scipy` + `scikit-learn` for the sparse user–movie matrix and similarity computation, `xgboost` for the supervised scorer, and `FastAPI` + Next.js for the serving stack.

**Evaluation Method** Both scorers are evaluated against the centered-cosine collaborative-similarity target. After fitting a MinMax scaler on the training split (separately for the ground truth, model, and heuristic scores), we compute on both splits:

- **Pearson** $r$: measures linear correlation between predicted scores $\hat{y}_i$ and ground-truth scores $y_i$:

$$r = \frac{\sum_i (\hat{y}_i - \bar{\hat{y}})(y_i - \bar{y})}{\sqrt{\sum_i (\hat{y}_i - \bar{\hat{y}})^2}\,\sqrt{\sum_i (y_i - \bar{y})^2}} \tag{1}$$

  A higher $r$ indicates the scorer captures the linear trend in collaborative similarity across pairs. [2]
- **Spearman** $\rho$: measures rank-order correlation, making it robust to outliers and non-linear scaling:

$$\rho = 1 - \frac{6 \sum_i d_i^2}{n(n^2 - 1)} \tag{2}$$

  where $d_i$ is the difference between the predicted and ground-truth rank of pair $i$, and $n$ is the number of pairs. A higher $\rho$ means the scorer better preserves the relative ordering of movie pairs by similarity, directly reflecting recommendation ranking quality. [2]
- **RMSE**: measures absolute calibration error on normalized scores:

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (\hat{x}_i - y_i)^2} \tag{3}$$

  A lower RMSE indicates that predictions are closer to the ground-truth values in magnitude. Because the target distribution is concentrated near zero, RMSE rewards conservative near-zero predictions and is not the primary quality lens. [2]
- **Recall@$K$**: measures retrieval quality, treating a pair as "relevant" when its normalized ground-truth score exceeds 0.5:

$$\text{Recall@}K = \frac{|\{\text{relevant pairs ranked in top-}K\}|}{|\{\text{all relevant pairs}\}|} \tag{4}$$

  for $K \in \{5, 10, 20, 50, 100\}$. This metric directly reflects the system's ability to surface the most similar movies near the top of a ranked recommendation list, which is the goal of practical deployment. [2]

## 5.2 Experimental Results

Two approaches were evaluated: (1) a supervised XGBoost regression model trained to predict collaborative similarity, and (2) a feature-based heuristic scoring function using predefined weights. The dataset was split into 60 percent training and 40 percent evaluation sets.

**Ground-Truth Distribution (Fig. 4)** The centered-cosine collaborative-similarity target is tightly concentrated near zero on both splits ($\mu \approx 0.001$, $\sigma \approx 0.021$), with a long positive tail ($[-0.244, 0.723]$ on train; $[-0.191, 0.554]$ on eval) The practical implication is that "high-similarity" pairs are rare: relevant pairs under the gt_norm $\geq 0.5$ threshold used for Recall@$K$ make up well under 1% of each split, which pressures both methods toward the

mean and makes ranking, not calibration, the appropriate quality lens.

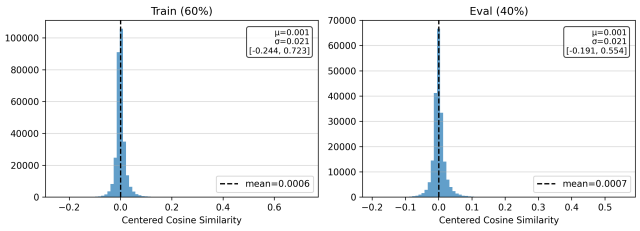


**Figure 4: Ground Truth collabrative similarity.**

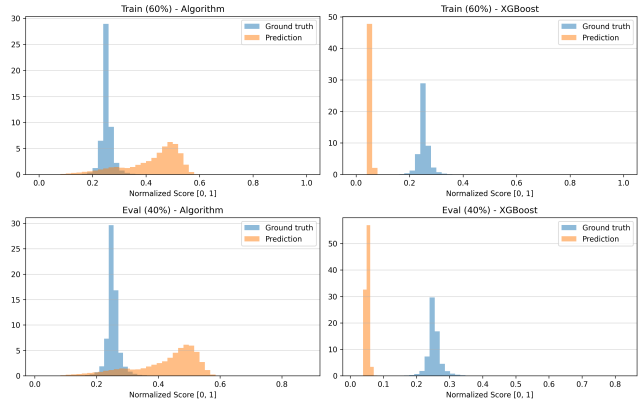


**Figure 5: Distribution of similarity scores Heuristic Vs. Model.**

**Predicted-Score Distributions (Fig.5)** The heuristic approach produces a wider spread of scores across $[0, 1]$, indicating higher variance and a tendency to assign moderate similarity values across many pairs. In contrast, the XGBoost model generates predictions concentrated near lower similarity values, reflecting the influence of the imbalanced target distribution and regression bias toward the mean.

Table 4 reports the full set of correlation and RMSE metrics on the normalized scores.

**Table 4: Regression metrics on normalized scores**

| Split | Method | Pearson $r$ | Spearman $\rho$ | RMSE |
|---|---|---|---|---|
| Train | XGBoost | 0.274 | 0.184 | 0.202 |
| Train | Heuristic | 0.067 | 0.101 | 0.199 |
| Eval | XGBoost | 0.185 | 0.159 | 0.203 |
| Eval | Heuristic | 0.068 | 0.103 | 0.199 |

**Correlation and Scatter (Figs. 6, 7, 8)** Across both splits, XGBoost attains roughly 2.7× the heuristic's Pearson $r$ and 1.5–1.8× its Spearman $\rho$ (Table 4) The scatter plots (Figs. 6-7) show this visually: the heuristic cloud is almost orthogonal to the perfect-agreement line, while the XGBoost cloud exhibits a modest but clear positive tilt.

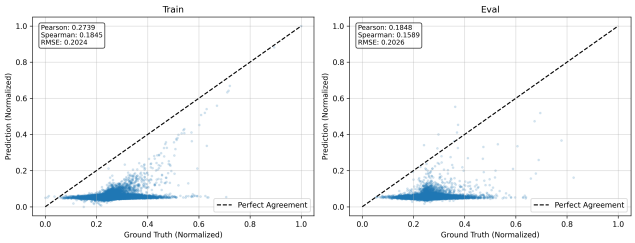


**Figure 6: Model scatter plot.**

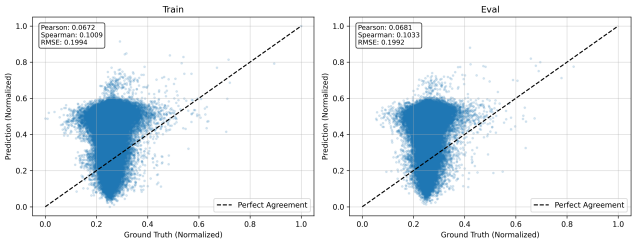


**Figure 7: Heuristic scatter plot.**

The RMSE picture is reversed: the heuristic posts a marginally lower RMSE (0.199 vs. 0.202–0.203) because its bias-toward-the-mean predictions match the bulk of the target mass near zero. Fig. 8 makes the generalization gap clear, with XGBoost's train Pearson of 0.274 dropping to 0.185 on eval (a ∼33% drop), while the heuristic's correlations are essentially identical across splits because no learning occurs.

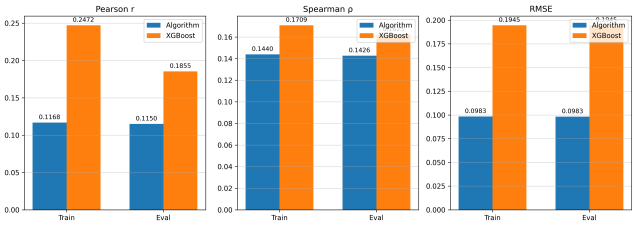


**Figure 8: Pearson and Spearman performance in Heuristic vs Algorithm.**

In terms of ranking performance, Fig. 9 shows that the XGBoost model outperforms the heuristic baseline across all values of K in Recall@K evaluation. On the train split, XGBoost dominates at every $K$ by a wide margin (e.g., 0.464 vs. 0.196 at $K$=100, a 2.4× gap) The eval split tells a more nuanced story: at $K$=5 the heuristic *beats* XGBoost (0.073 vs. 0.049); the two tie at $K$=10; and XGBoost pulls ahead only from $K$=20 onwards, reaching a 1.5× lead at $K$=100 (0.293 vs. 0.195). The improvement is consistent across both training

and evaluation sets, indicating that the model is more effective at identifying highly similar movie pairs and ranking them near the top of the recommendation list.

**Table 5: Recall@$K$ on both splits**

| Split | Method | $K$=5 | 10 | 20 | 50 | 100 |
|---|---|---|---|---|---|---|
| Train | XGBoost | 0.089 | 0.179 | 0.357 | 0.429 | 0.464 |
| Train | Heuristic | 0.018 | 0.089 | 0.107 | 0.161 | 0.196 |
| Eval | XGBoost | 0.049 | 0.098 | 0.146 | 0.220 | 0.293 |
| Eval | Heuristic | 0.073 | 0.098 | 0.122 | 0.195 | 0.195 |

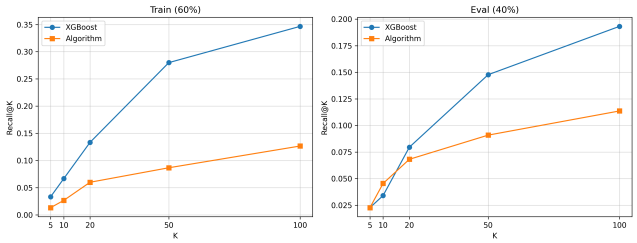


**Figure 9: Recall@ K Model Vs Heuristic.**

**Feature Importance (Fig. 10)** By gain (average loss reduction per split), the top features for XGBoost are `director_match` (0.392) and `cast_sim` (0.289), with `plot_sim` coming fourth at 0.105. By *weight* (number of splits), plot similarity is used most often (0.415), since it is continuous and can partition the training set many times at modest gain each. This demonstrates the importance of combining unstructured text embeddings with structured attributes for modeling user-driven similarity.

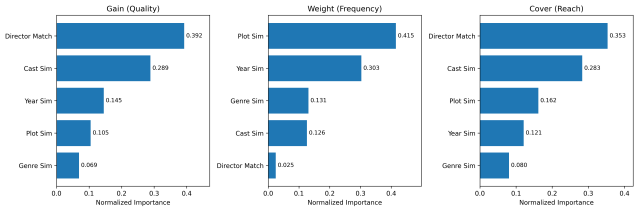


**Figure 10: Feature Importance.**

**Prototype Development (Fig. 11)** In order to evaluate our approach, we developed a publicly accessible web-based prototype (linked here) that integrates user interaction, movie metadata retrieval, and recommendation generation. The homepage provides search functionality, and users can query any movie in the database through the search interface, which resolves each query to a detailed metadata page. Additionally, the homepage shows personalized recommendations and previously liked movies, demonstrating the integration of user preferences into the recommendation pipeline.

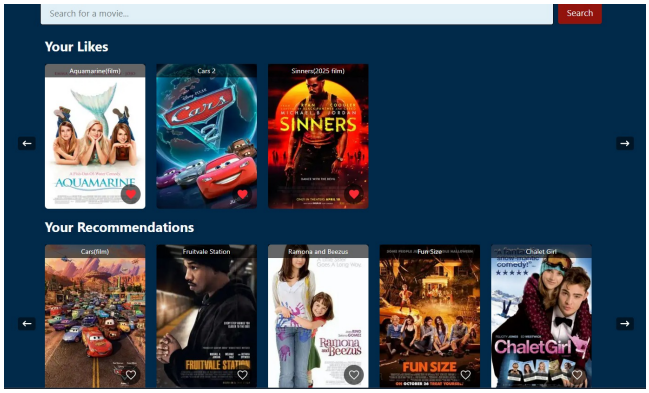


**Figure 11: Prototype website home page.**

## 5.3 Discussion

The results highlight the distinction between regression accuracy and ranking performance in recommendation systems. As shown in Fig. 8, the heuristic method achieves lower RMSE due to its tendency to produce conservative predictions centered around the mean. However, this behavior limits its ability to capture meaningful variation in similarity.

In contrast, the XGBoost model demonstrates improved correlation with the ground truth (Fig. 8) , indicating its ability to learn non-linear relationships between content features and user similarity.

The imbalance in the target distribution (Fig. 4), where most movie pairs exhibit low similarity, introduces challenges for supervised learning and contributes to prediction bias toward lower similarity values (Fig. 5). Despite this, the model achieves significantly better ranking performance, as shown in Fig. 8, indicating its effectiveness in identifying highly relevant movie pairs.

**Sources of Error** The observed prediction errors and the modest absolute correlation values to three compounding factors. First, the ground-truth collaborative similarity is derived from user ratings, which encode implicit human preferences that content features alone cannot fully capture. Two movies may share a director, genre, and similar plot embeddings yet receive very different rating profiles, introducing irreducible noise in the supervision signal. Second, the severe class imbalance in the target distribution ($\sigma \approx 0.021$; well under 1% of pairs exceed the relevance threshold) incentivizes the XGBoost regressor to minimize squared error by predicting conservatively. This regression-toward-the-mean behavior suppresses predicted scores for the rare high-similarity pairs, directly harming Pearson $r$ and Recall@$K$ at small $K$. Third, the heuristic's fixed weights were set manually rather than fit to data, meaning the relative contributions of plot similarity, genre, and cast are not tuned to the specific distributional properties of our Netflix–Wikipedia overlap. Addressing these factors, e.g., using a ranking-aware loss, oversampling high-similarity pairs, or learning heuristic weights from a held-out validation set, is the most direct path to address sources of error in this project.

## 6 Conclusion

This project developed a content-based movie recommendation system to model user preference similarity using both supervised learning and feature-based heuristic approaches. A complete data mining pipeline was implemented, including data collection, preprocessing, feature engineering, pairwise dataset construction, and model development. The supervised XGBoost model was trained to predict collaborative similarity derived from user ratings, while a weighted heuristic function was used as a baseline for comparison.

Experimental results demonstrate that the supervised model achieves higher correlation with ground truth similarity and significantly better ranking performance, as measured by Recall@K, Although the heuristic method achieves lower RMSE due to its conservative predictions. These findings highlight the advantage of data-driven approaches in learning complex relationships between heterogeneous content features.

However, the results also reveal challenges associated with the imbalanced distribution of similarity values, which leads to prediction bias toward lower similarity scores. Despite this limitation, the model effectively identifies highly relevant movie pairs, which is critical for recommendation tasks.

A key contribution of this work is the development of a unified experimental framework that enables direct comparison between heuristic and supervised similarity modeling within a shared data pipeline, incorporating both collaborative and content-based signals. Rather than fully integrating these approaches into a hybrid model, we chose to isolate and implement key components of a hybrid framework to be able to systematically evaluate their individual behavior and understand how different similarity formulations impact recommendation quality. The implementation of an interactive web-based prototype further demonstrates the practical relevance of these methods in a real-world setting.

Future work can focus on improving both model performance and system scalability. One potential direction is the development of a hybrid recommendation framework that combines the strengths of the heuristic and supervised approaches. In this system, the heuristic method can serve as a computationally efficient pre-filtering stage to effectively generate a candidate set of top-K movies, where can then be re-ranked using the supervised model to produce higher-quality recommendations. This two-stage architecture can reduce computational cost while still preserving the ranking performance of the learned model, making it more suitable for a large-scale deployment.

Additionally, future improvements could include incorporating richer feature representations, exploring ranking-based optimization objectives, and expanding the dataset to improve robustness and generalization. Overall, this work demonstrates how combining data-driven modeling with interpretable feature-based methods can provide a foundation for building effective and practical recommendation systems.

## References


[1] Rishabh Ahuja, Arun Solanki, and Anand Nayyar. 2019. Movie Recommender System Using K-Means Clustering and K-Nearest Neighbor. In *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*. 263–268. doi:10.1109/CONFLUENCE.2019.8776969

[2] Aman Chadha. 2024. Recommendation Systems: Evaluation Metrics. https://aman.ai/recsys/metrics/. Accessed: 2026.

[3] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 785–794. doi:10.1145/2939672.2939785

[4] IMDb. 2024. IMDb Non-Commercial Datasets. https://developer.imdb.com/non-commercial-datasets/.

[5] Sambandam Jayalakshmi, Narayanan Ganesh, Robert Čep, and Janakiraman Senthil Murugan. 2022. Movie Recommender Systems: Concepts, Methods, Challenges, and Future Directions. *Sensors* 22, 13 (2022), 4904. doi:10.3390/s22134904

[6] Gun Il Kim, Jae Heon Kim, Minkyoung Kim, Taekyoung Kwon, and Beakcheol Jang. 2024. An Approach on Improving the Recommender System: Predicting Movie Genres Based on Plot Summaries. In *2024 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*. doi:10.1109/ICAIIC60209.2024.10463349

[7] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (2009), 30–37. doi:10.1109/MC.2009.263

[8] Netflix Inc. 2009. Netflix Prize Data. https://www.kaggle.com/datasets/netflix-inc/netflix-prize-data.

[9] N. Pradeep, K. K. Rao Mangalore, B. Rajpal, N. Prasad, and R. Shastri. 2020. Content Based Movie Recommendation System. *International Journal of Research in Industrial Engineering* 9, 4 (2020), 337–348.

[10] Bihi Sabiri, Amal Khtira, Bouchra El Asri, and Maryem Rhanoui. 2025. Hybrid Quality-Based Recommender Systems: A Systematic Literature Review. *Journal of Imaging* 11, 1 (2025), 12. doi:10.3390/jimaging11010012

[11] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-Based Collaborative Filtering Recommendation Algorithms. In *Proceedings of the 10th International World Wide Web Conference (WWW)*. ACM, Hong Kong, 285–295. doi:10.1145/371920.372071

[12] Nisha Sharma and Mala Dutta. 2020. Movie Recommendation Systems: A Brief Overview. In *Proceedings of the 8th International Conference on Computer and Communications Management (ICCCM)*. 59–62. doi:10.1145/3411174.3411194

[13] Wikipedia contributors. 2024. Wikipedia. https://download.kiwix.org/zim/wikipedia/.


## A Code and Prototype URLs

The source code for this project is publicly available at: https://github.com/IshaAtteri/datamining_881/tree/main-combined

The interactive web-based prototype can be accessed at: https://resilient-truth-production-eaba.up.railway.app/

If the prototype is not accessible (such as if our credits expire), the source code README will be updated with a more up-to-date link: https://github.com/IshaAtteri/datamining_881/blob/main/README.md

### A.1 Prototype Notes

The web-based prototype was built using Next.js and FastAPI, with Supabase for data storage. Instructions for running the prototype locally are included in the repository.