Anomaly Detection in Movie Recommendation Systems

Bernardinus Matteo Woenardi, Naufal Syaqil Bin Azmi, Chan Jess Myn, Justin Dalva Wicent, Oliver Loh

April 2025

Abstract

This paper presents an approach to anomaly detection in user behavior within a movie recommendation system. The goal is to detect anomalous user behavior and identify patterns indicative of abnormal interactions. We compare multiple supervised and unsupervised methods and analyze the characteristics of the detected anomalies. The full implementation, including code and data preprocessing scripts, is available at `https://github.com/WiceKiwi/cs421-project`.

I. Introduction

The goal of this project is to build a machine learning model capable of identifying anomalous users from user-interaction data. The data set is based on the famous movie recommendation dataset, where users rate movies with one of four types of feedback: dislike (-10), neutral (0), like (10) or watched (1). However, unlike traditional recommender system tasks, the focus here is not on predicting user preferences, but on detecting synthetic anomalies, users whose entire interaction histories were artificially generated.

The task involves classifying each user into one of six classes: one for natural users (label 0) and five corresponding to different types of anomalous behavior (labels 1 to 5). The dataset is highly imbalanced, with far more natural users than anomalous ones. Evaluation is based on the Area Under the Curve (AUC) in a one-vs-rest setting for each class, with the overall performance computed using a weighted average of all class AUCs.

II. Literature Review

A. Overview of Anomaly Detection

Anomaly detection is crucial in domains such as fraud detection and recommendation systems, where detecting users with unusual behavior is vital. In recommendation systems, anomalies can represent fraudulent activity, bots, or synthetic users. Techniques for anomaly detection can be broadly categorized into **supervised** and **unsupervised** methods.

B. Supervised Techniques

Supervised anomaly detection methods require labeled data to train classifiers. Algorithms such as **XGBoost**, **Random Forests**, and **SVMs** are commonly used for this task. **XGBoost**, in particular, is effective in detecting anomalies due to its ability to handle imbalanced datasets and model non-linear relationships between features. Its interpretability through feature importance (e.g., SHAP values) is also beneficial in understanding the key factors contributing to the classification of anomalous users.

C. Unsupervised Techniques

Unsupervised anomaly detection methods, such as **autoencoders**, **isolation forests**, and **DBSCAN**, do not require labeled data. Autoencoders detect anomalies based on poor reconstruction, while Isolation Forests isolate anomalies by partitioning the data. These methods are useful when labeled data is scarce, but they can struggle with subtle anomalies that exhibit slight deviations from normal behavior.

D. Challenges in Anomaly Detection for Recommender Systems

Anomaly detection in recommender systems faces two main challenges: **Subtlety of anomalies** and **Class imbalance**. Anomalies are often subtle and not easily distinguishable from normal behavior, which makes feature engineering critical. In addition, the imbalance between normal and anomalous users can lead to biased models. **SMOTE** and **Tomek Links** are commonly used to address this, by generating synthetic minority class samples and removing borderline examples, respectively.

E. Feature Engineering and Exploratory Data Analysis (EDA)

Effective feature engineering is crucial for distinguishing between normal and anomalous users. Our approach was guided by insights uncovered during **Exploratory Data Analysis (EDA)**, which revealed distinct patterns in user behavior.

We engineered features across several dimensions:

- **User–Rating Relationships**: Captured user tendencies through features like rating distributions, rating entropy, dominant rating type, and polarity (e.g., like-to-dislike ratio).
- **User–Movie Relationships**: Analyzed how users interact with movies of varying popularity, including metrics like rare movie engagement rate and deviation from average item ratings.
- **Sequential and Temporal Patterns**: Modeled behavior over time using features such as rating direction changes, rating stability, and interaction gaps.
- **Platform-Specific Behavior**: Inspired by typical platform usage patterns, we introduced features to capture contrarian behavior, such as users who consistently liked unpopular movies or disliked mainstream ones.

These features were selected and refined based on patterns observed during EDA, such as irregular rating behaviors, inconsistent interaction frequencies, and deviations from community norms. By capturing these nuanced behaviors, our features enabled the model to better detect subtle anomalies (see Appendix B for the full feature summary table).

III. Motivation of Algorithm Design

The core challenge of this project lies in accurately detecting multiple types of subtle, synthetic anomalies in a highly imbalanced user population. Our algorithm design was guided by the need to balance predictive power, robustness to class imbalance, interpretability, and adaptability to evolving data batches.

A. Why XGBoost?

After evaluating several models, including unsupervised methods like **Autoencoders**, **Isolation Forests**, and **DBSCAN**, we found that these models struggled to consistently distinguish between the anomaly types due to the subtle, structured nature of the anomalies. These methods also lacked access to label information, which limited their ability to capture class-specific behaviors.

Supervised models, on the other hand, could leverage the labeled training data to detect fine-grained behavioral differences. Among the supervised options, **XGBoost** emerged as the most effective due to its:

- **Boosting Power**: Sequential tree construction allows XGBoost to learn from

hard-to-classify examples, which is essential for detecting rare anomalies.

- **Class Imbalance Handling**: XGBoost allows sample weighting and inherently shifts focus toward harder samples, enabling it to prioritize minority classes during training.

- **Non-linear Feature Interaction**: Its tree-based nature captures complex patterns in high-dimensional feature space.

- **Interpretability**: Integration with **SHAP** makes it possible to understand model decisions and refine features iteratively. [SHAP Developers, 2024]

B. Hyperparameter Tuning

To optimize performance, we conducted extensive hyperparameter tuning using **Optuna**, an efficient optimization framework. Key parameters tuned included the learning rate, tree depth, and regularization terms. We applied early stopping based on validation AUC to prevent overfitting and speed up training. [Optuna Contributors, 2024]

C. Handling Class Imbalance

Due to the overwhelming presence of natural users (label 0) in the dataset, we adopted a hybrid resampling strategy using **SMOTE-Tomek** [Imbalanced-learn Developers, 2023].

- **SMOTE (Synthetic Minority Oversampling Technique)** synthetically generates new samples for minority classes, improving class representation.

- **Tomek Links** remove ambiguous, borderline samples near class decision boundaries, cleaning up noisy regions of the feature space.

Applying this strategy after train-validation splitting allowed us to balance the training set without leaking test information.

D. Evaluation Strategy

Given the imbalance and multi-class nature of the task, we used **AUC (Area Under the Curve)** as the primary evaluation metric. AUC was computed in a **one-vs-rest** setting for each class, and the final score was a weighted average across all six classes. This allowed us to assess how well the model separated each anomaly type from the rest, which is more informative than raw accuracy in our context.

The evaluation pipeline mimicked a production environment: for week n, we trained on batches 1 to $n-2$ and evaluated on batch $n-1$, simulating the arrival of new data over time.

IV. Results

A. Performance Metrics

The final model, trained using **XGBoost**, was evaluated across four weekly batches using the Area Under the Curve (AUC) in a one-vs-rest setting. The final performance was computed as a weighted average of the per-class AUCs. The table below summarizes the AUC scores and team ranking over time.

Week	Weighted AUC	Team Ranking
9	0.9476	1
10	0.8223	3
11	0.8719	4
12	0.9191	3

Table 1: XGBoost performance over weekly evaluation batches

Despite the fluctuating nature of the weekly test sets, our model remained consistently within the top 4 teams. The strongest performance was recorded in Week 9, where feature exploits (e.g., movie ID patterns) were most effective. In later weeks, as these exploits were patched, we relied more heavily on behavioral features to maintain strong performance.

B. Anomaly Class Hypotheses

Based on SHAP dependency plot analysis, we propose the following behavioral hypotheses for

each anomaly class (see Figure 1):

- **Class 1:** Users who primarily engage with rare content but provide limited explicit feedback or exhibit a negative bias. Their behavior shows moderate deviation from community norms and inconsistent rating patterns.
- **Class 2:** Users who prefer rare and high-index items, with irregular interaction sequences and high variability in item popularity. Their behavior suggests exploratory and erratic engagement.
- **Class 3:** Contrarian users who consistently rate popular content negatively and avoid neutral feedback. They display high inconsistency and deviate from mainstream preferences.
- **Class 4:** Users with a bias toward low-rated or unpopular content, active across a wide range of movie IDs. Their pattern reflects consistent contrarianism.
- **Class 5:** Niche-oriented users who consistently assign high ratings to rare items, with strong deviations in both item selection and rating polarity.

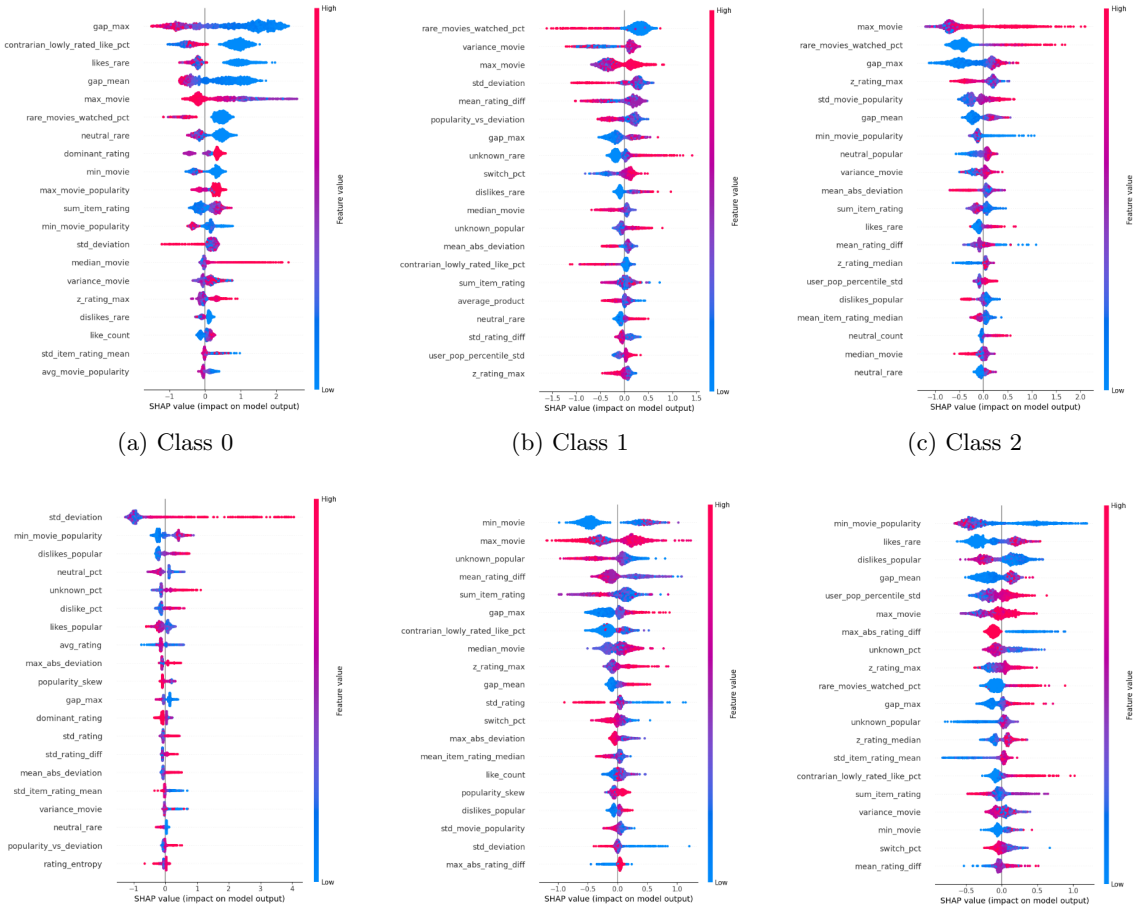
References

[Imbalanced-learn Developers, 2023] Imbalanced-learn Developers (2023). SMOTETomek: Combine over- and under-sampling using smote and tomek links. https://imbalanced-learn.org/stable/references/generated/imblearn.combine.SMOTETomek.html. Library component of imbalanced-learn. Accessed: 2025-04-12.

[Optuna Contributors, 2024] Optuna Contributors (2024). Optuna: A hyperparameter optimization framework. https://optuna.org. Version 3.5.0. Accessed: 2025-04-12.

[SHAP Developers, 2024] SHAP Developers (2024). SHAP: Explainable machine learning with shapley values. https://shap.readthedocs.io/en/latest/. Python library for model interpretability. Accessed: 2025-04-12.

I. SHAP Summary Plots for All Anomaly Classes



(a) Class 0 (b) Class 1 (c) Class 2

(d) Class 3 (e) Class 4 (f) Class 5

Figure 1: SHAP Summary Plots for all anomaly classes.

II. Feature Summary Table

Feature Name	How It's Calculated / Meaning
review_count	Total number of ratings by the user
avg_rating	Mean of all ratings given by the user
std_rating	Standard deviation of the user's ratings
like_count	Count of ratings equal to 10
dislike_count	Count of ratings equal to -10
unknown_count	Count of ratings equal to 1
neutral_count	Count of ratings equal to 0
like_pct	Proportion of 'like' ratings
dislike_pct	Proportion of 'dislike' ratings
unknown_pct	Proportion of 'watched but unknown' ratings
neutral_pct	Proportion of 'neutral' ratings
rating_entropy	Entropy of rating type distribution per user
avg_movie_popularity	Average popularity of movies rated by the user
std_movie_popularity	Std deviation of movie popularity
min_movie_popularity	Minimum movie popularity rated by the user
max_movie_popularity	Maximum movie popularity rated by the user
rare_movies_watched_pct	Proportion of movies rated that are rare
unique_movies	Number of unique movies rated
mean_deviation	Average deviation of user's ratings from movie average
std_deviation	Standard deviation of rating deviations
mean_abs_deviation	Mean absolute deviation from average rating
max_abs_deviation	Maximum absolute deviation from average rating
mean_rating_diff	Average difference between sequential ratings
std_rating_diff	Standard deviation of sequential rating differences
mean_abs_rating_diff	Mean absolute difference in sequential ratings
max_abs_rating_diff	Maximum absolute difference in sequential ratings
rating_changes_count	Count of rating changes across items
rating_changes_pct	Proportion of sequential changes
z_rating_mean	Mean of z-scored ratings
z_rating_std	Standard deviation of z-scored ratings
z_rating_max	Maximum z-score of ratings
z_rating_min	Minimum z-score of ratings

Feature Name	How It's Calculated / Meaning
z_rating_median	Median z-score of ratings
z_rating_skew	Skewness of z-scored ratings
likes_popular	Proportion of popular movies liked
likes_rare	Proportion of rare movies liked
dislikes_popular	Proportion of popular movies disliked
dislikes_rare	Proportion of rare movies disliked
neutral_popular	Proportion of neutral ratings on popular movies
neutral_rare	Proportion of neutral ratings on rare movies
unknown_popular	Proportion of 'watched' ratings on popular movies
unknown_rare	Proportion of 'watched' ratings on rare movies
interaction_entropy	Entropy of unique item-rating combinations
like_dislike_ratio	Ratio of like_count to dislike_count
rating_range	Maximum sequential absolute rating difference
popularity_vs_deviation	Product of avg_movie_popularity and mean_abs_deviation
entropy_by_count	rating_entropy weighted by log of review_count
review_count_bin	Binned review count into quantiles
min_movie	Minimum item ID rated by user
max_movie	Maximum item ID rated by user
median_movie	Median item ID rated
variance_movie	Variance of item IDs rated
sum_item_rating	Sum of item * rating
sum_rating	Sum of all user ratings
average_product	Average of item * rating over reviews
product_above_zero	Flag if sum_item_rating ¿ 0
sum_above_zero	Flag if sum_rating ¿ 0
avg_product_vs_avg_rating	Ratio of average_product to avg_rating
gap_mean	Mean difference between sorted item IDs
gap_std	Std deviation of item ID gaps
gap_max	Max gap between item IDs
user_pop_percentile_mean	Mean percentile popularity of movies rated
user_pop_percentile_std	Std deviation of movie popularity percentiles
rating_pct_-10	Proportion of -10 ratings
rating_pct_0	Proportion of 0 ratings
rating_pct_1	Proportion of 1 ratings

Feature Name	How It's Calculated / Meaning
rating_pct_10	Proportion of 10 ratings
mean_item_rating_mean	Mean of average item ratings
std_item_rating_mean	Std of average item ratings
mean_item_rating_std	Mean std dev of item ratings
mean_item_rating_median	Mean of item rating medians
mean_item_rating_skew	Mean skewness of item ratings
mean_item_review_count	Mean number of reviews per rated item
max_item_review_count	Max number of reviews per rated item
min_item_review_count	Min number of reviews per rated item
item_mean_vs_user_avg	Difference between mean item rating and user's avg_rating
item_skew_bias	User's average item skewness
normalized_movie_popularity	avg_movie_popularity / mean_item_review_count
popularity_skew	max_movie_popularity - min_movie_popularity
rating_polarity	like_pct - dislike_pct
activity_weighted_skew	z_rating_skew * log1p(review_count)
switch_pct	Percentage of rating direction switches
dominant_rating	Most frequent rating value by user
dominance_ratio	Proportion of dominant rating
highly_rated_movie_count	Count of movies from top 5% liked
highly_rated_movie_pct	Proportion of top-rated movies rated
contrarian_lowly_rated_like_pct	Pct of likes on bottom 10% rated movies