International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 11 | Nov - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Steam Game Popularity Prediction Using Machine Learning

Dr. M. SENGALIAPPAN, ANVINRAJ M R

Professor & Head, Department of Computer Applications, Nehru College of Management, Coimbatore, Tamil Nadu, India, ncmdrsengaliappan@nehrucolleges.com

Student of II MCA, Department of Computer Applications, Nehru College of Management, Coimbatore, Tamil Nadu, India, anvinrajmr@gmail.com

Abstract

Game development requires substantial investment under uncertain market demand. This paper presents the design of a pre-release decision-support system that estimates the popularity category of a proposed Steam game using features available before launch. The proposed Python and Streamlit application accepts genre, supported operating systems, languages, planned price, multiplayer support, and release year. A supervised classification pipeline predicts low, medium, or high popularity and reports model-derived probabilities. A separate retrieval and natural-language-processing module identifies historically similar games, summarizes Steam recommendation labels, and extracts recurring player feedback. To avoid target leakage, post-release reviews and ownership counts are excluded from predictive inputs, although they may define historical targets or support explanatory analysis. The manuscript specifies data integration, validation, model comparison, and evaluation methods. Quantitative results are intentionally not reported because model training and independent testing have not yet been verified.

Keywords—Steam, game popularity, machine learning, Streamlit, sentiment analysis, decision support.

I. INTRODUCTION

The commercial outcome of a new video game is difficult to forecast before release. Genres, pricing, platform coverage, language availability, and multiplayer features may influence demand, but no single feature guarantees success. A developer-facing tool that summarizes historical evidence can support early design decisions and highlight uncertainty.

This work proposes a practical mini-project centered on Steam games. Developers describe a planned game through a Streamlit form. A trained classifier produces a popularity category and calibrated class probabilities, while a distinct historical-analysis component retrieves comparable games and summarizes their player reviews. The system is intended for comparative planning, not as a financial guarantee.

The principal contributions are: (i) a pre-release feature-only prediction design; (ii) an AppID-based integration strategy for game metadata and review records; (iii) similarity-based historical comparison; and (iv) an evaluation protocol that explicitly addresses leakage, selection bias, and probability calibration.

II. RELATED WORK AND MOTIVATION

Prior research in predictive analytics has used supervised classification and ensemble learning to identify associations between product attributes and observed outcomes. Random forests combine multiple decision trees to reduce variance, while logistic regression provides a transparent baseline for classification [1], [2]. Text-mining techniques can complement numerical predictions by revealing common themes in customer feedback [3].

Steam offers a useful study setting because game listings contain descriptive attributes and review interfaces provide recommendation labels and free-text opinions. However, post-release variables such as owner estimates, review counts, and playtime cannot be treated as inputs to a genuine pre-release forecast. Doing so would produce misleadingly optimistic evaluation scores.

A second challenge is representativeness. Datasets containing only best-selling or top-ranked games may omit the unsuccessful examples required for learning the difference between low and high popularity. Review datasets may also disproportionately represent active players, particular languages, or recent periods. These limitations must be documented rather than hidden.

III. PROBLEM DEFINITION

Given a planned game with pre-release feature vector x, estimate a class y from {low, medium, high} and class probabilities $P(y \mid x)$. Historical success labels must be operationalized using a consistent popularity measure, such as estimated ownership observed over a comparable time horizon. If only cumulative ownership snapshots are available, game age and censoring require special consideration.

A separate similarity function ranks released games according to comparable genres, platforms, prices, and other known attributes. Reviews attached to those historical games are then analyzed to estimate recommendation shares and identify frequent feedback themes. These review-derived findings explain the reference games; they are not predictions of reviews for an unreleased title.

IV. DATA SOURCES AND PREPARATION

The intended data sources are a Steam Store Games Dataset and Steam Reviews Dataset, obtained from Kaggle. Exact field availability, provenance, licensing, and record counts must be checked before use. Steam AppID serves as the primary linking key. The original ZIP archives should remain unchanged; processed training and review samples are stored separately.

For memory-efficient review processing, Python can read members directly from ZIP files and write deduplicated rows incrementally to CSV. A target sample of 100,000 authentic reviews is practical for a student project, but the actual attainable count depends on archive contents and valid AppID matches.

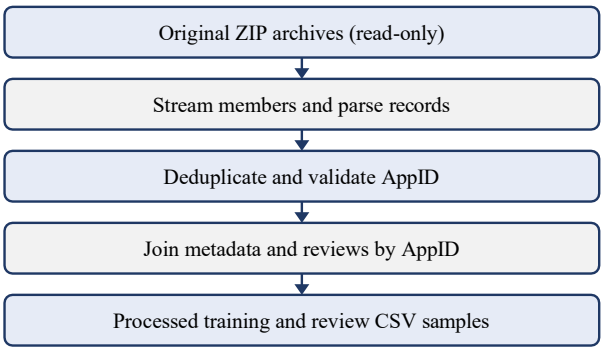


Fig. 1. Data integration and preparation workflow.

A. Popularity Label Construction

Low, medium, and high classes may be defined by thresholds or quantiles of a single popularity measure, such as estimated ownership, computed within a comparable observation window. The thresholds must be fixed before model comparison and reported with the class distribution, because changing them after viewing test results would bias the evaluation. Games that are too recent to have a stable measure should be excluded or flagged.

B. Data Quality Checks

Quality checks include verifying AppID uniqueness, removing duplicate review rows, rejecting records with missing recommendation labels, and recording the share of games that have no linked reviews. These counts should be reported so that readers can judge how representative the final sample is.

V. PROPOSED SYSTEM ARCHITECTURE

The architecture separates the developer interface, prediction pipeline, and historical review analysis. This separation prevents review-derived post-release information from entering the pre-release classifier and makes each component independently testable.

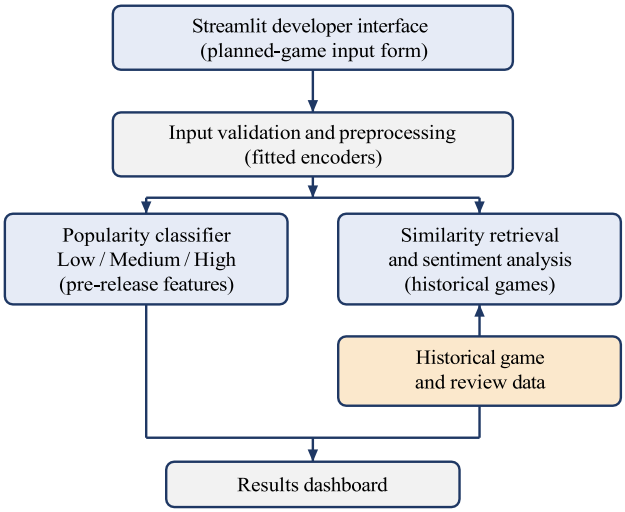


Fig. 2. Proposed application architecture with independent prediction and historical-review modules.

The Streamlit interface validates user inputs, including genre, operating systems, languages, planned price, multiplayer configuration, and release year. The prediction service applies the same fitted transformations used during model training. The historical service retrieves comparable released games and summarizes linked reviews. Both outputs are displayed on a single results dashboard.

VI. FEATURE ENGINEERING

Categorical variables may be represented using one-hot or multi-label encoding. Platform support is encoded as separate binary indicators; languages may be represented as supported-language flags or a count, depending on data completeness. Price must use a consistent currency and historical price basis. Release year can capture temporal changes, but extrapolation to future years should be treated cautiously.

TABLE I
PRE-RELEASE INPUT FEATURES

Feature	Encoding
Genre(s)	One-hot / multi-label
Operating systems	Binary indicator per platform
Supported languages	Flags or language count
Planned price	Single currency, validated
Multiplayer support	Binary indicator
Release year	Numeric (cautious extrapolation)

The planned game name is retained for display but excluded from prediction unless a defensible text-based feature model is separately validated. Missing categories and invalid prices should be handled explicitly. A fitted preprocessing pipeline must be saved with the classifier to ensure inference matches training.

VII. MACHINE LEARNING METHODOLOGY

Historical game records are cleaned and labeled according to a stated popularity definition. The training process compares logistic regression, a decision tree, and a random forest using consistent features and partitions (Table II). The final model is selected on held-out validation results rather than by assumption.

TABLE II
CANDIDATE CLASSIFIERS

Model	Role in comparison
Logistic regression	Transparent linear baseline
Decision tree	Interpretable non-linear model
Random forest	Ensemble reducing variance

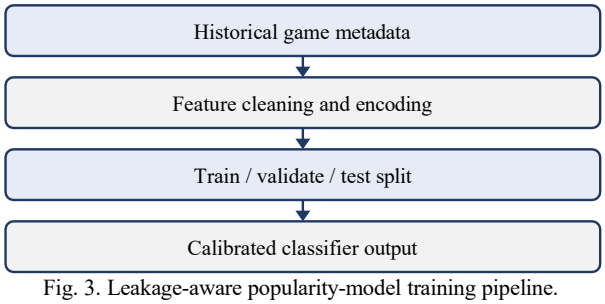


Fig. 3. Leakage-aware popularity-model training pipeline.

The dataset should be partitioned by release date where feasible, training on older games and testing on newer games. If time-based evaluation is impossible, a stratified holdout may be used with an explicit limitation. Duplicate titles, remasters, and

related releases should be checked for train-test contamination.

Because popularity categories may be imbalanced, macro-averaged precision, recall, and F1-score are more informative than accuracy alone. A confusion matrix reveals which categories are confused. Probability calibration should be assessed using reliability plots or a proper scoring rule before displaying a percentage as model confidence.

VIII. PREDICTION INTERPRETATION

For a new proposal, the application displays the most probable class and the probability distribution across all three classes. A sample interface might show "High: 0.76," but such a number must be computed by the trained model and must not be inserted as a fixed demonstration result in a live system. A high predicted class probability does not establish a 76% chance of profitability.

The dashboard should communicate dataset coverage, prediction limitations, and the distinction between popularity and revenue. If the model encounters unsupported categories or severe out-of-distribution inputs, it should provide a caution rather than an unjustifiably precise estimate.

IX. SIMILAR GAME RETRIEVAL

The system identifies historical reference games using feature similarity. After encoding genres, platform flags, multiplayer support, and normalized price, cosine similarity or nearest-neighbor search can rank comparable released titles. The similarity weights should be specified and tested, as genre overlap may be more meaningful than a small difference in release year.

The retrieval layer presents the game name, AppID, release date, available popularity indicators, and observed review statistics. Only games with valid metadata should be ranked; games lacking reviews can still appear, with missing sentiment metrics clearly marked. Similarity is descriptive and must not be mistaken for causal evidence.

X. REVIEW SENTIMENT AND FEEDBACK

Steam reviews provide a binary recommendation field and free-text feedback. For a set of retrieved games, the observed positive recommendation share is the number of positive recommendations divided by the number of reviews with valid labels. The negative share is computed analogously. These proportions describe the sampled historical reviews, not the proposed game.

A text-processing pipeline normalizes review text, removes empty or duplicate entries, and may use TF-IDF features to identify common terms or phrases. Theme extraction should distinguish praise from complaints, and example quotations must be traceable to real source reviews. If a sentiment classifier is used, its text-based sentiment output must be reported separately from Steam recommendation labels.

XI. IMPLEMENTATION PLAN

The software is organized into data ingestion, preprocessing, model training, similarity retrieval, sentiment analysis, and Streamlit presentation modules. Pandas and NumPy support

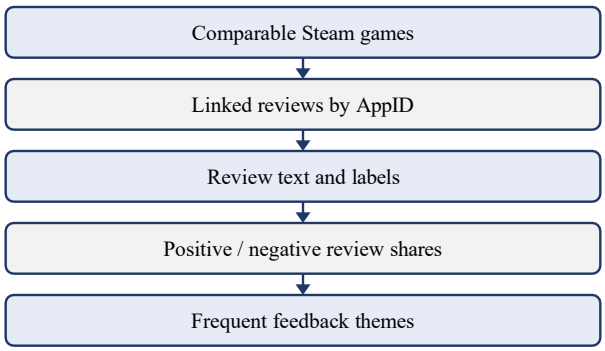


Fig. 4. Historical review analysis and feedback extraction.

structured processing; scikit-learn implements preprocessing and candidate classifiers; Joblib stores fitted artifacts; Plotly renders probability bars and sentiment charts.

The repository should contain the application entry point, reusable modules, training scripts, test cases, model metadata, and a requirements file. On Windows, a virtual environment and a launch script can simplify setup. The large raw review archive need not be included in the final software ZIP. Instead, a documented sampled CSV and reproducible extraction script are sufficient when redistribution rights permit.

At run time, the Streamlit application loads the saved preprocessing pipeline, classifier, and sampled historical tables once and reuses them across user sessions, so that each form submission only performs validation, encoding, prediction, and retrieval. Model metadata, including the training date range and package versions, is displayed alongside predictions to support traceability.

XII. TESTING STRATEGY

Functional tests should verify valid input submission, rejection of invalid prices, handling of empty platform selections, prediction output shape, similarity ranking, and graceful behavior when reviews are unavailable. Data tests should verify AppID joins, deduplication, missing-value handling, and target-label generation.

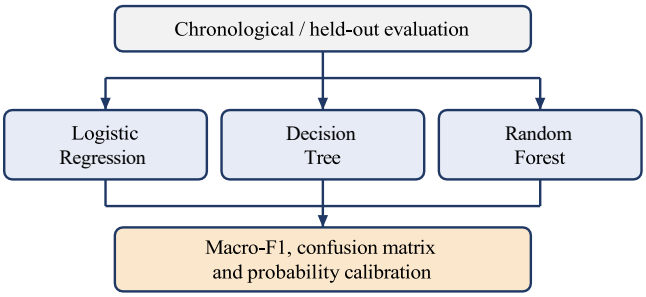


Fig. 5. Candidate model comparison and evaluation protocol.

Performance reporting should include the number of games and reviews used, class distribution, date boundaries, model hyperparameters, confusion matrix, macro F1-score, and calibration diagnostics. Repeated experiments should use fixed seeds and record package versions. The same held-out test set must not be repeatedly used to choose the final model.

No experimental score is asserted in this manuscript because the final datasets and trained artifacts have not been independently inspected or evaluated. This avoids presenting

TABLE III
PLANNED EVALUATION METRICS

Metric / diagnostic	Purpose
Macro precision, recall, F1	Balanced view of all three classes
Confusion matrix	Shows which classes are confused
Reliability plot	Checks probability calibration
Proper scoring rule	Scores probabilistic quality
Time-based split report	Checks generalization to newer games

illustrative figures as research findings.

XIII. EXPECTED APPLICATION OUTPUTS

The proposed application is expected to display (1) a low/medium/high popularity category with model-derived class probabilities; (2) a list of similar historical Steam games; (3) observed positive and negative recommendation percentages; (4) a sentiment distribution pie chart; and (5) frequent positive and negative review themes. A summary panel may provide practical suggestions grounded in the retrieved historical evidence.

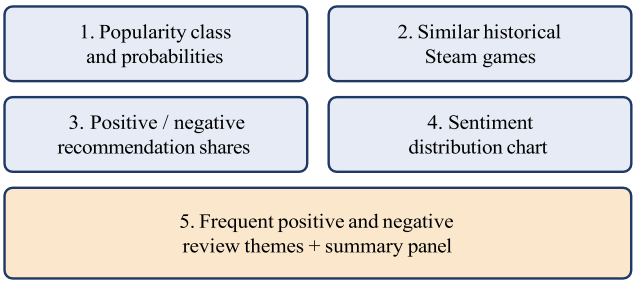


Fig. 6. Expected layout of the results dashboard.

For example, if comparable games repeatedly receive complaints about technical performance, the system can advise additional optimization testing. Such recommendations should include review counts and acknowledge sampling bias. The software should avoid implying that historical reviews establish the quality or future performance of an unreleased game.

XIV. LIMITATIONS AND ETHICAL CONSIDERATIONS

Popularity is not equivalent to financial success: owner estimates, revenue, profit, and player engagement measure different outcomes. Pricing, marketing expenditure, brand recognition, and external events may be missing from public metadata. Genre and platform categories are imperfect proxies for game quality. Consequently, the system should support developer judgment rather than replace market research.

Public Steam reviews may contain personal information or copyrighted text. The system should minimize unnecessary author identifiers, retain provenance, respect source terms, and avoid redistributing large raw archives without permission. Multilingual reviews require language-aware preprocessing; an English-only sample limits generalization.

XV. FUTURE WORK

Future extensions could incorporate marketing-budget inputs, trailer or description embeddings, uncertainty estimates, temporal market trends, and additional external validation. A prospective evaluation could compare predictions made before launch against outcomes measured after a fixed release window. Such a study would provide stronger evidence of practical predictive value.

XVI. CONCLUSION

This paper specifies a Steam game popularity prediction system that combines pre-release machine learning with separate historical review analysis. The design enables developers to enter planned game attributes, obtain a popularity classification, inspect comparable released games, and understand common player feedback. Its central methodological requirement is the strict separation of pre-release predictors from post-release outcome and review information. A reproducible implementation, representative historical data, calibrated probabilities, and honest evaluation are necessary before claims about predictive performance can be made.

REFERENCES

[1] L. Breiman, “Random forests,” *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[2] F. Pedregosa *et al.*, “Scikit-learn: Machine learning in Python,” *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[3] B. Liu, *Sentiment Analysis and Opinion Mining*. San Rafael, CA, USA: Morgan & Claypool, 2012.

[4] Valve Corporation, “Steamworks Web API documentation.” [Online]. Available: https://partner.steamgames.com/doc/webapi

[5] Streamlit, “Streamlit documentation.” [Online]. Available: https://docs.streamlit.io/

[6] Valve Corporation, “Steam Store.” [Online]. Available: https://store.steampowered.com/