# Supervised Machine Learning for Product Review Feedback
# Amazon Video Games 5-core

Zhuolun Wu
Brown University

December 14, 2025

https://github.com/jackwu-beep/Data1030-project

## 1 Introduction

### 1.1 Motivation and problem statement

Online product reviews play a central role in shaping consumer decisions, and star ratings provide a concise numerical summary of user opinion. Accurate rating prediction can support recommender systems, large-scale sentiment analysis, and automated feedback summarization.

In this project, we study the task of predicting 1–5 star ratings for Amazon video game reviews using supervised learning methods. We frame the problem as a five-class classification task with an underlying ordinal structure, where misclassifying a review by one star is less severe than predicting an extreme opposite rating. This motivates the use of both standard classification metrics and ordinal-aware evaluation measures.

### 1.2 Dataset description

We use the Amazon product review data released by Ni et al. [1], specifically the Video Games 5-core subset. Each review includes (`reviewText` and `summary`), Rating (`overall`), and additional metadata for user information. The 5-core constraint ensures that every user and product appears in at least five reviews. So there are no missing values. To balance computational feasibility with dataset size, we restrict attention to reviews from 2014 onward while preserving a large and diverse sample.

### 1.3 Previous work

Many projects on Amazon reviews treat the task as multi-class classification from 1–5 stars using TF–IDF features and linear models. Previous studies report that relatively simple models with TF–IDF representations still perform competitively, with similar test score and performance around 60-65 percent.[4]

## 2 Exploratory Data Analysis

### 2.1 Distribution of ratings and review activity

Figure 1 shows the distribution of star ratings in the dataset. Ratings are skewed toward higher values. Though not extreme, this skew motivates the use of metrics beyond accuracy.

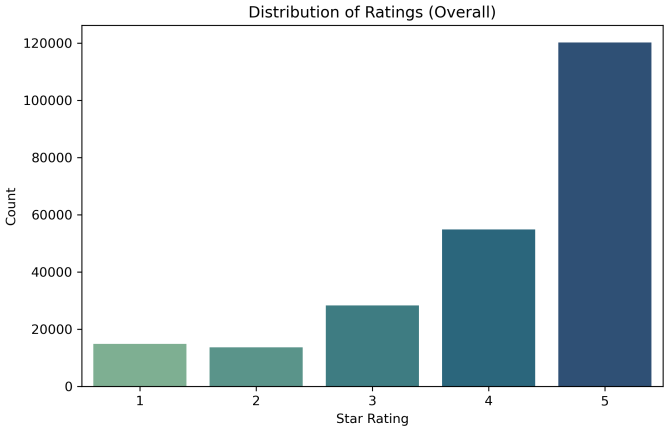


Figure 1: Distribution of star ratings.

User and product activity are highly uneven. Most users contribute only a small number of reviews, while a small group of "power users" write many reviews. These heavy-skewed distributions, shown on a logarithmic scale, suggest strong user-level and product-level effects that may influence observed ratings.

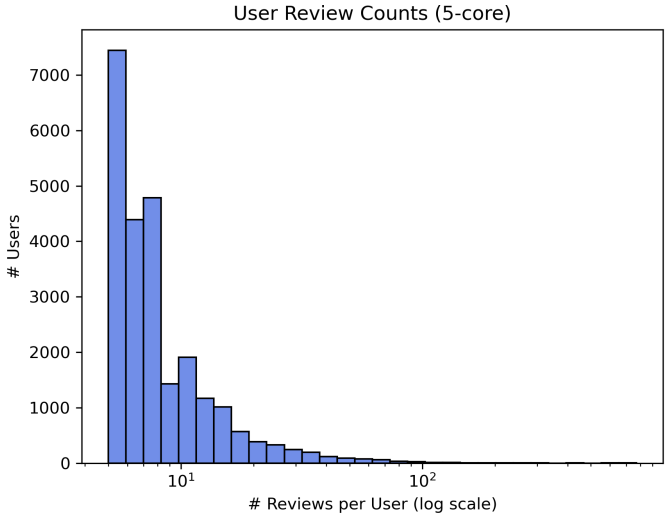


Figure 2: User review count distribution.

## 2.2 Review length, sentiment, and punctuation

Review text length is highly skewed, with many short reviews and fewer very long ones. When stratified by rating, mid-range ratings (2–4 stars) tend to contain longer reviews, while many 5-star reviews are relatively concise. This suggests that satisfied players often express approval succinctly, whereas less-than-perfect experiences are described in greater detail.

Additionally, sentiment polarity increases monotonically with star rating. Extreme ratings exhibit more emotionally charged language. These patterns motivated the inclusion of sentiment

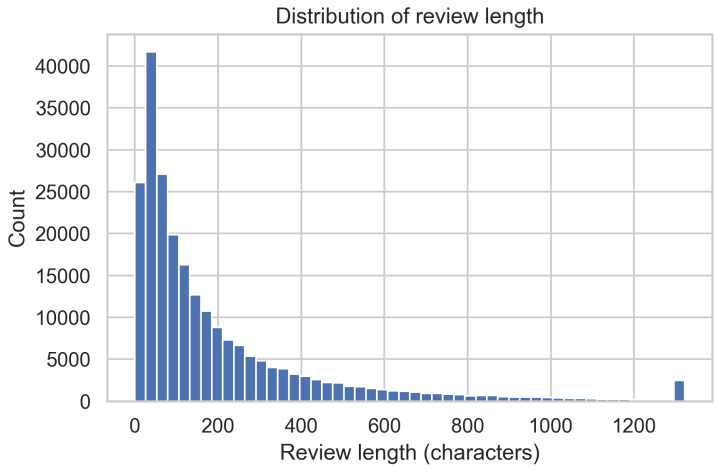


Figure 3: Review length by rating.

scores and punctuation-based features in the modeling stage.

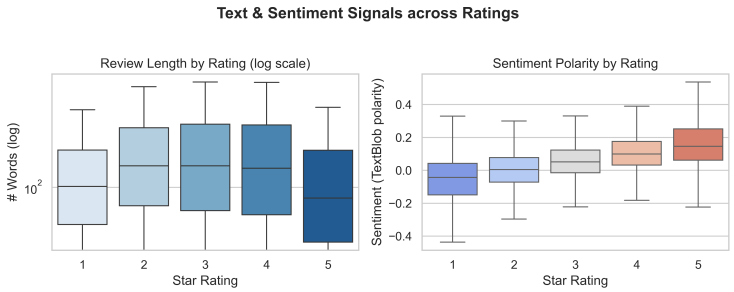


Figure 4: Sentiment polarity by star rating.

## 2.3 User and product rating bias

Finally, we examine average ratings per user and per product. The distributions in Figure 6 reveal systematic bias: some users consistently give higher or lower ratings, and some products tend to receive uniformly positive or negative feedback. These findings highlight the importance of careful train–test splitting by user to avoid information leakage.

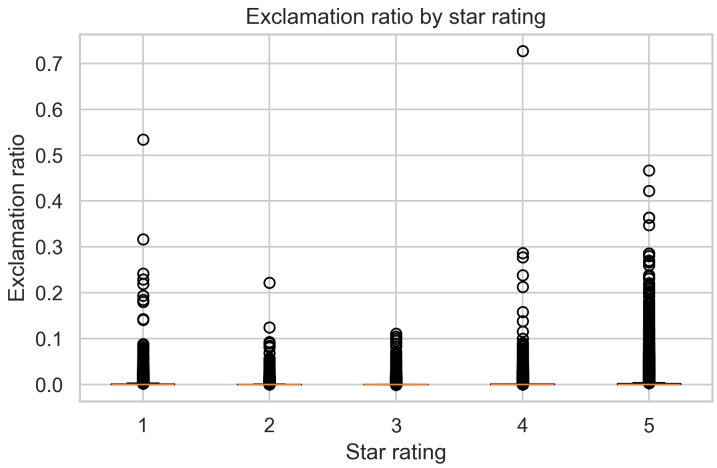


Figure 5: Exclamation mark ratio by rating.

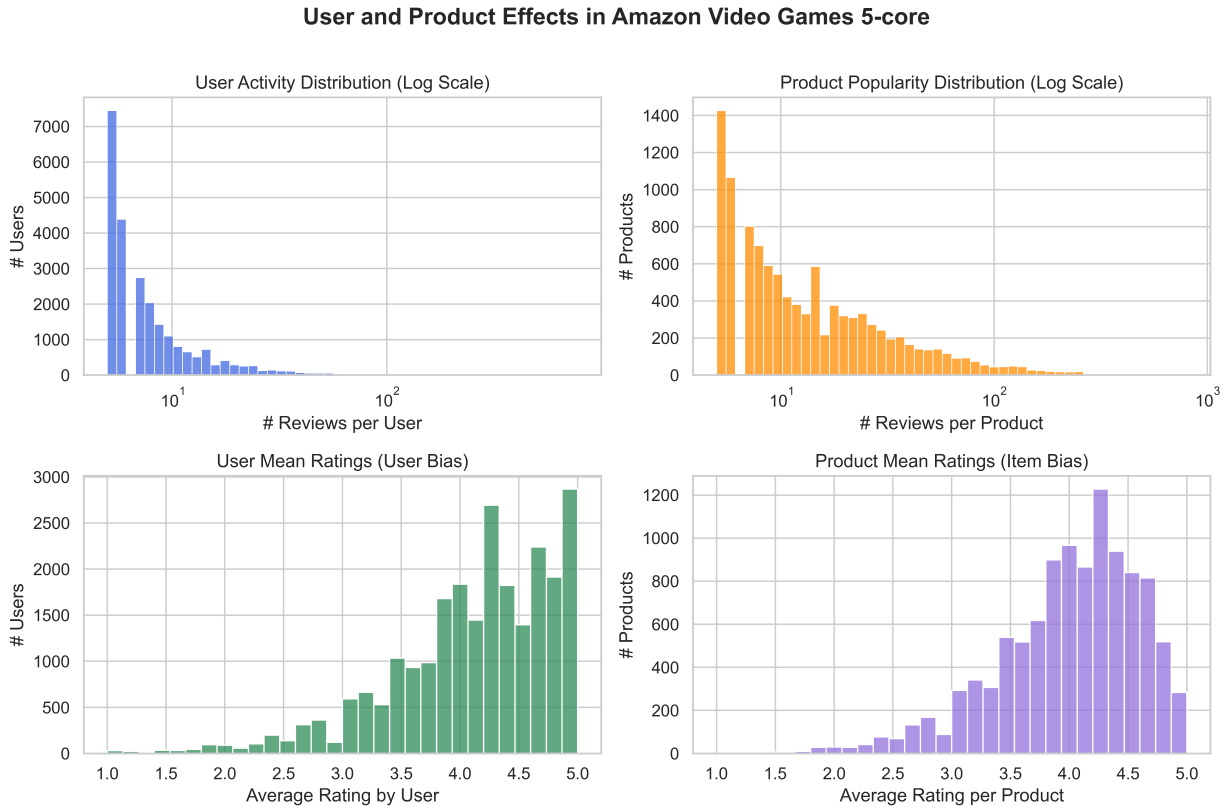


Figure 6: User and product effects.

Overall, the EDA reveals substantial class skew, strong user and product effects, and clear linguistic signals correlated with ratings, which gives valuable information for next steps.

# 3 Methods

## 3.1 Target Encoding

The original rating takes values from 1 to 5. For compatibility with certain libraries, we encode the target as

$$\texttt{overall\_enc} = \texttt{overall} - 1 \in \{0, 1, 2, 3, 4\}.$$

The main input text field is `reviewText`.

## 3.2 Feature construction

Beyond the raw review text, we construct a set of numeric features designed to capture helpfulness, temporal context, and simple stylistic properties of the reviews.

**Helpfulness.** Each review contains a `helpful` field of the form $[up, total]$. We derive the scalar features:

- `helpful_total`: the total number of helpfulness votes.

**Review-level numeric features.** From precomputed fields we include:

- `review_length`: the length of the review text.
- `exclam_ratio`: the ratio of exclamation marks to total characters.
- `question_ratio`: the ratio of question marks to total characters.
- `sentiment`: a scalar sentiment score derived from the review text.

We collect these into the set

$$\texttt{numeric\_cols}$$

## 3.3 Data splitting

Because individual reviewers often have consistent rating habits, simply splitting rows at random would allow the model to see the same user in both train and test sets. To prevent this, we use group-aware splitting based on reviewer IDs.

First, we perform a user-aware split of the full dataset using `GroupShuffleSplit`. All reviews by the same reviewer are assigned to the same side of the split. Within the val and test, we apply a second `GroupShuffleSplit` to obtain half-half validation and test sets. This yields non-overlapping train, validation, and test sets with proportions approximately 0.6/0.2/0.2.

For computational reasons, we further restrict the training pool to reviews from year $\geq 2014$, which still leaves a large number of training examples and preserves the no-leakage constraint within each fold.

Formally, the input matrices for each split are

$$X_{\text{split}} = \big[\texttt{reviewText}, \texttt{numeric\_cols}\big], \quad y_{\text{split}} = \texttt{overall\_enc},$$

for $\texttt{split} \in \{\text{train}, \text{val}, \text{test}\}$.

### 3.4 Preprocessing and feature pipeline

We implement preprocessing through a `ColumnTransformer` that combines a numeric pipeline and a text pipeline.

**Numeric pipeline.** For the numeric columns in `numeric_cols`, we apply:

1. `StandardScaler`, which centers each feature and scales it to unit variance.

This ensures that numeric features are on a comparable scale for models that are sensitive to feature magnitudes.

**Text pipeline.** For the textual feature `reviewText`, we use a TF–IDF representation implemented with `TfidfVectorizer`, key configurations:

- English stopword removal,
- $n$-gram range of (1, 2), capturing unigrams and bigrams,
- `min_df` $= 10$ and `max_df` $= 0.5$ to prune very rare and very common tokens,

**Combined transformer.** The `ColumnTransformer` combines these two pipelines:

$$\texttt{preproc} = \begin{cases} \text{numeric pipeline} & \text{on } \texttt{numeric\_cols}, \\ \text{TF–IDF pipeline} & \text{on } \texttt{reviewText}. \end{cases}$$

### 3.5 Models and hyperparameter tuning

We compare four supervised learning models, all wrapped in the shared preprocessing pipeline (`preproc`): two linear models in the TF–IDF feature space (logistic regression and LinearSVC) and two non-linear tree ensembles (random forest and XGBoost).

For logistic regression, we use a multinomial formulation with $\ell_2$ regularization and the `saga` solver. We first fit a baseline model with $C = 1.0$, then tune the regularization strength over

$$C \in \{0.1, 0.2, 0.5, 1.0, 2.0\}$$

via `GridSearchCV` with same GroupKFold.

For LinearSVC, we tune its regularization parameter over

$$C \in \{0.1, 0.5, 1.0, 2.0\}.$$

For non-linear models, we fit a random forest classifier with the grid

$$n_{\text{estimators}} \in \{200, 300\}, \quad \texttt{max\_depth} \in \{\texttt{None}, 20\}, \quad \texttt{min\_samples\_leaf} \in \{1, 2, 5\}.$$

For XGBoost, we use a multi-class softmax objective and a small focused grid,

$$n_{\text{estimators}} = 80, \quad \texttt{max\_depth} = 4, \quad \texttt{learning\_rate} = 0.05,$$

with subsampling parameters

$$\texttt{subsample} \in \{0.8, 1.0\}, \quad \texttt{colsample\_bytree} \in \{0.8, 1.0\}.$$

The same cross-validation framework is also used in the uncertainty analysis described below, both across folds (split uncertainty) and, for tree-based models, across multiple random seeds (algorithmic uncertainty).

### 3.6 Evaluation metrics and uncertainty estimation

For the final comparison on test set, we also report F1 scores, mean absolute error (in star units), and a within-1 accuracy metric that counts predictions within ±1 star of the true rating, to reflect both class imbalance and the ordinal structure of the labels.

To quantify *uncertainty due to data splitting*, we use three-fold group-aware cross-validation (`GroupKFold`) on the training subset, grouping by reviewer ID. For each model, we record accuracy on each fold and summarize it by the mean and standard deviation across folds.

To quantify *uncertainty due to non-deterministic training*, we refit the tuned models with different random seeds, keeping both the data and the cross-validation splits fixed, and report the mean and standard deviation of cross-validated accuracy across seeds.

## 4 Results

### 4.1 Baseline comparison

As a simple baseline, we consider a majority-class classifier that always predicts the most frequent rating in the training data. This baseline achieves an accuracy of approximately **0.41** on the test set and a very low macro-F1 score (around **0.13**).

In contrast, our best model (the tuned logistic regression) attains a cv accuracy of **0.6329** with a std of only **0.0018** across folds, which shows a great improvement.

### 4.2 Model comparison

Figure 7 summarizes the performance of the main models, together with cross-validation accuracy and its standard deviation across folds.

| model | type | cv_mean_acc | cv_std_acc | seed_std_acc | test_acc | test_macro_f1 | test_weighted_f1 | test_mae | test_within1 |
|---|---|---|---|---|---|---|---|---|---|
| Majority baseline | baseline | nan | nan | nan | 0.4118 | 0.1354 | 0.3465 | 0.9367 | 0.7496 |
| LogReg_baseline | linear | 0.6327 | 0.0024 | 0.0 | 0.585 | 0.3902 | 0.5336 | 0.6493 | 0.8548 |
| LogReg_tuned | linear | 0.6329 | 0.0018 | 0.0 | 0.5893 | 0.4105 | 0.5472 | 0.6246 | 0.8648 |
| LinearSVC | linear | 0.6306 | 0.0043 | 0.0 | 0.5837 | 0.3668 | 0.5172 | 0.6673 | 0.8464 |
| RandomForest | non-linear | 0.5977 | 0.0051 | 0.0003 | 0.5297 | 0.2083 | 0.3903 | 0.8682 | 0.772 |
| XGBoost | non-linear | 0.6123 | 0.004 | 0.0004 | 0.5603 | 0.3127 | 0.4631 | 0.748 | 0.8128 |

Figure 7: Summary of model performance.

Overall, the tuned logistic regression model achieves the best combination of accuracy and stability. LinearSVC performs similarly but slightly worse. The two tree-based models underperform the linear models with cross-validated accuracies around 0.60 and 0.61, respectively. Nonetheless, all learned models beat the baseline.

### 4.3 Uncertainty analysis

Across three user-grouped folds, the tuned logistic regression achieves 0.6329 ± 0.0018 accuracy, LinearSVC achieves 0.6306 ± 0.0043, the random forest 0.5977 ± 0.0051, and XGBoost

0.6123 ± 0.0040. The variation across folds indicates that the exact set of users held out can move accuracy by about 0.3–0.5 percentage.

For non-deterministic part, we refit each tuned model with different random seeds while keeping the data and cross-validation splits fixed. The random forest accuracy across seeds is 0.5986 ± 0.0003; XGBoost achieves 0.6128 ± 0.0004; In all cases, variability across seeds is much smaller than variability across folds.

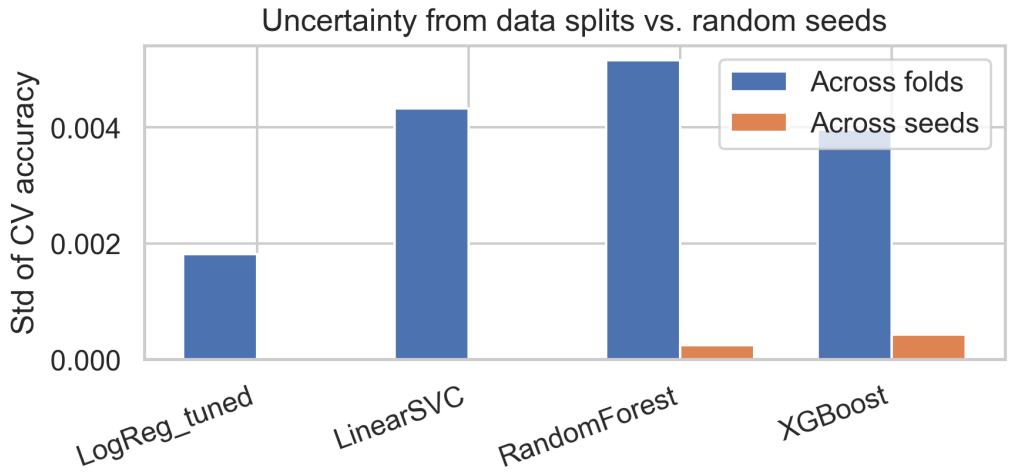


Figure 8: uncertainty summary

### 4.4 Global feature importance

We first examine global feature importance using the coefficients of the tuned logistic regression model (Figure 9). The most strongly positive features for high ratings include lexical cues such as *love*, *great*, *amazing*, and *favorite*, while the most negative features for low ratings include words such as *worst*, *waste*, and *money*. Numeric features are generally less important than the top TF-IDF features.

To understand model's choice of importance, we compute permutation importance on the validation set for the tuned logistic regression pipeline (Figure 10). This analysis confirms that sentiment score, review length, and exclamation-mark ratio are the most important numeric features, while removing highly polarized words (e.g., *love*, *worst*) produces the largest drops in accuracy.

We also examine tree-based feature importance from the tuned XGBoost model (Figure 11). The result is similar: emotionally charged words and overall sentiment dominate rating predictions.

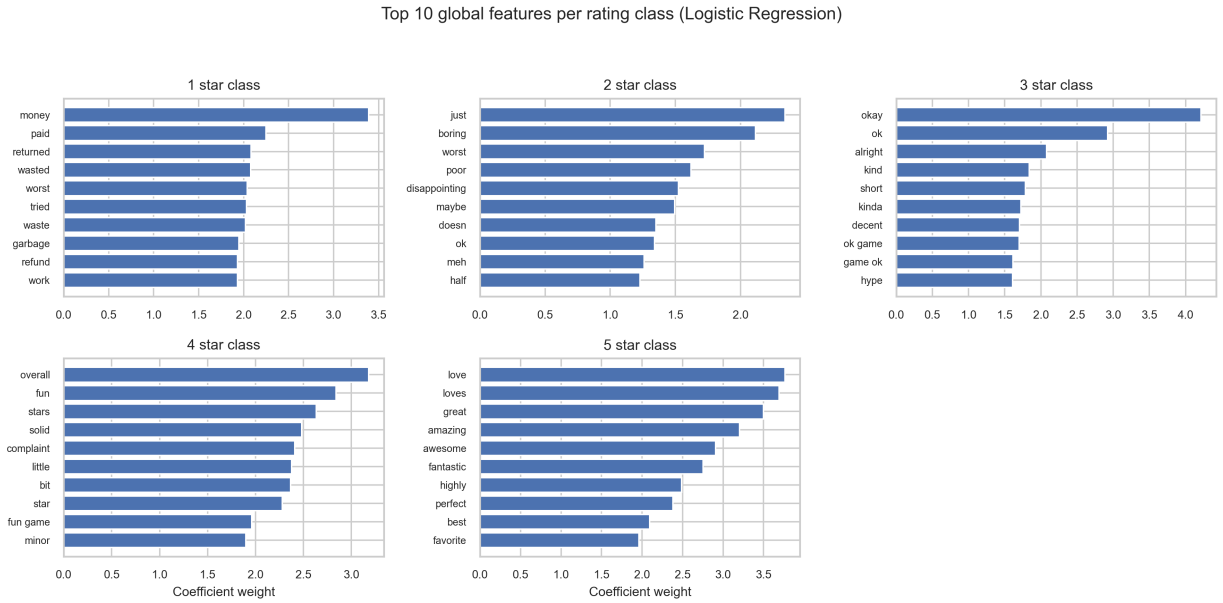


Figure 9: GFI-Log

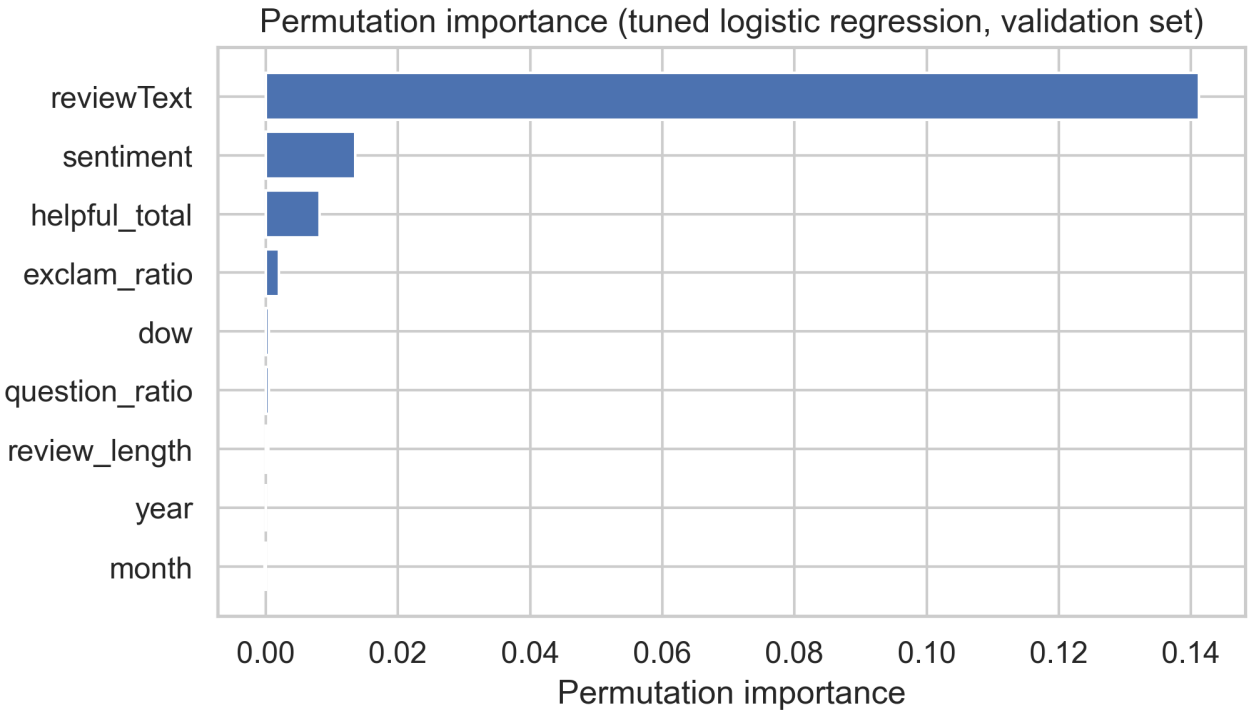


Figure 10: Permutation GFB-Log.

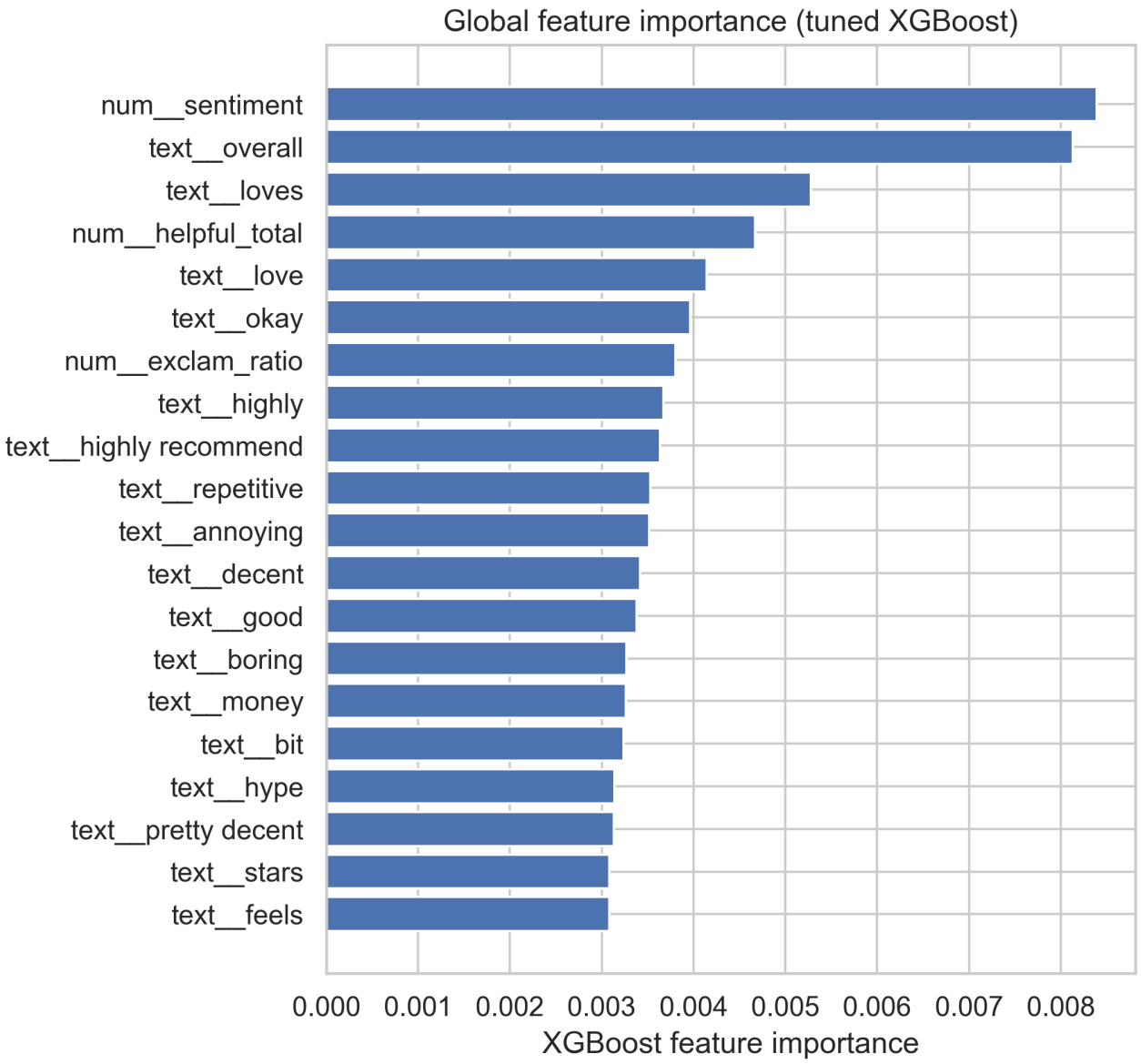


Figure 11: GFI-XGB.

### 4.5 Local feature importance and interpretation

Figure 12 shows a local SHAP explanation for one correctly predicted 3-star review. Positive contributions from tokens such as `text__game progress`, and `text__absolutely` push the prediction toward a higher rating. In contrast, tokens like `text__game problem` and `text__game probably` contribute negatively. Overall, the explanation is consistent with how a human might read the review: a neutral comment with praise and critics.

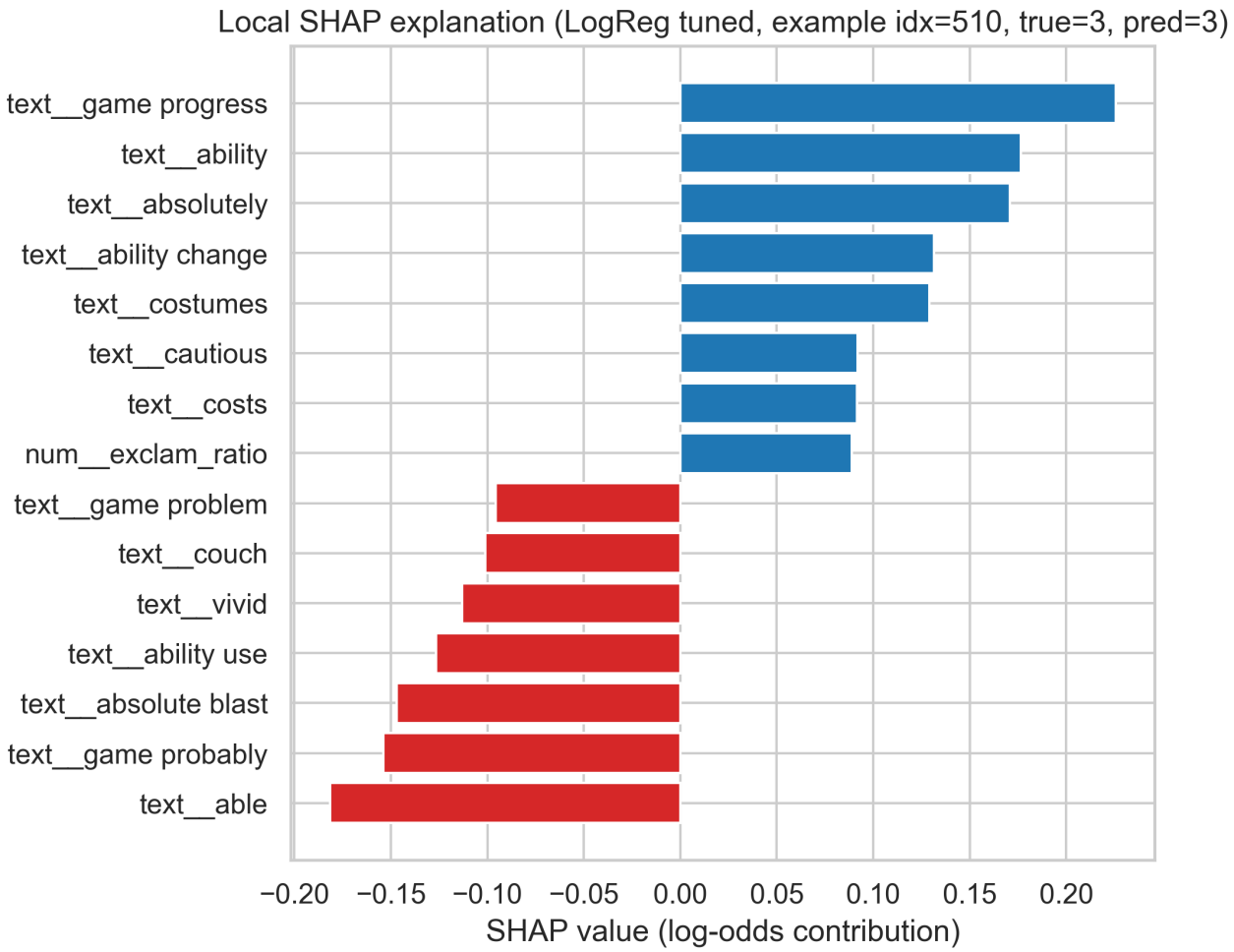


Figure 12: Local SHAP explanation for a correctly predicted 3-star review.

Overall, the interpretation results align well with domain expectations: strongly positive and negative words are the most influential features, while sentiment scores and punctuation ratios act as secondary signals. This supports the use of these models for understanding the language patterns associated with rating system.

## 5 Outlook

- **Exploit ordinal structure more directly:**
  - Train ordinal regression models that optimize MAE or RMSE.
  - Use loss functions that penalize large rating differences (e.g. predicting 1 instead of 5) more heavily.
- **Improved model selection:**
  - Run a more extensive hyperparameter search
  - Explore simple ensembles (e.g., averaging probabilities of logistic regression and LinearSVC).

- **Category-specific and user-aware explanations:**
  - Analyze feature importances separately for different game genres to see if players value different aspects.
  - Incorporate user and product embedding or bias terms explicitly to better capture personalization.

## References


[1] J. Ni, J. Li, and J. McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 188–197, Hong Kong, China, 2019.

[2] J. McAuley, C. Targett, Q. Shi, and A. van den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 43–52, 2015.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[4] A. Taparia and T. Bagla. Sentiment analysis: Predicting product reviews' ratings using online customer reviews. *SSRN Electronic Journal*, 2020. https://doi.org/10.2139/ssrn.3655308.