Amazon Software Reviews Machine Learning Project (Group L)

Morgan Simmons [1] **Rachel Kang** [2] **Mark Haller** [3]

Abstract

This project investigates whether Amazon reviews in the software category received more than five helpful votes at the time it is posted, using only features available at posting time. Working with a large and highly imbalanced dataset, we cleaned and processed numeric variables, reviewed metadata, and summarized texts. Additionally, we converted textual information into TF-IDF features and reduced dimensionality with Truncated SVD. Models were trained using a chronological split to prevent information leakage and to mirror real-world deployment conditions, with class weighting applied throughout to address the rarity of helpful reviews. We evaluated three classifiers which were k-Nearest Neighbors, Logistic Regression, and a small Neural Network. We used these classifiers within a unified preprocessing pipeline and tuned hyperparameters using rolling time-series cross-validation. Logistic Regression achieved the strongest and most stable performance, with a ROC-AUC of approximately 0.89 and the highest precision–recall performance despite class imbalance. KNN performed poorly, while the neural network captured some nonlinear patterns but did not generalize consistently. Error analysis showed that all models struggled with short or ambiguous summaries (false negatives) and with long but ultimately low-engagement reviews (false positives). Overall, our results indicate that while helpfulness can be partially predicted from summary text and metadata, user engagement remains difficult to anticipate, and model performance is fundamentally constrained by the extreme imbalance and limited signal available at posting time.

1. Data

All data, reports, and code will be in the provided link below:

github.com/yxs4yt/machine_learning_project

Research Model: Predicts at the time a review is posted whether it will receive more than 5 helpful votes using the Summary data.

The dataset used in this project comes from the publicly available Amazon Product Reviews corpus released by Amazon as part of its open data initiative (originally compiled by Julian McAuley and colleagues at UC San Diego) (Ni et al., 2019). The data is collected from Amazon's review platform and includes user-generated review text, ratings, metadata, timestamps, and community-based "helpful vote" counts. These reviews are not sampled by topic or product category; rather, they are scraped directly from Amazon's public webpages and aggregated into large-scale JSON files. Each entry corresponds to a single review at the time it appeared on the platform, with the vote field representing the number of helpful votes accumulated up to the time of data collection.

Because the dataset reflects naturally occurring consumer reviews, Amazon does not intervene in how votes are generated or displayed. The helpful-vote counts are fully organic, determined by other users clicking whether a review was useful. Our analysis focuses only on features available at the time a review is posted, even though the helpful vote total may increase later. The dataset was accessed through the Amazon Reviews open repository and stored locally for preprocessing and modeling.

1.1. Variables[2]

overall: The overall product rating (from 1-5)

verified: If the review was verified by Amazon

reviewTime: Time of the review (month, date, year format)

reviewerID: Reviewer ID

asin: Product ID

style: A dictionary of the product metadata

reviewerName: Name of the reviewer

reviewText: Text of the review

summary: Summary of the review

*unixReviewTime:*Time of the review Unix time

vote: How many "helpful" votes the review has

image: List of images the user posts with their review after they have received the product

The *vote* variable is a key variable to the model because it is the outcome of the question. It is a direct signal in the data of how many people gave the product a good review.

1.2. Challenges[2]

One of the biggest practical challenges was data size. Several raw files are extremely large, which strains local storage and memory and slows down simple operations (previewing, counting rows, or building features). A second challenge was version control and hosting: GitHub enforces a hard 100 MB limit per file. Large datasets must be tracked with Git Large File Storage (LFS) or stored outside the repository and fetched at runtime. These constraints shaped how we had to pick data. Overall, the key variables are well-defined and aligned with the research question, but scale and hosting limitations were the main challenges in the reading portion for analysis. (**?**)

Challenges with cleaning hinged on choosing what to keep vs. drop and making the thresholds explicit. We removed near-empty or low-relevance columns and kept only rows with core IDs/timestamps.

The main challenge of preparing our data was making sure the data could answer our question at the time a review is posted. We limited features to what's available, treated missing votes as 0, set a clear helpfulness cutoff (5), and used a time-based split in hopes that our model learns from earlier reviews and is tested on later ones which avoids leaks and keeping the prediction context realistic.

1.3. Cleaning the Data[1]

When looking at our data, we came across some issues concerning three main variables. One of these variables was *image*. The concern with this variable was that we didn't foresee this variable being of note for future modeling. Furthermore, many reviews were missing the image variable leaving empty space, so it was removed. The same rationale was used for the *style* variable, as keeping them would increase the computational power needed to traverse and model using the dataset. The other variable that needed to be cleaned was *vote*. As previously stated, vote is a key variable that we're using for our model, but there were issues when it had missing entries. To mitigate this, we transformed all missing entries in the *vote* column to 0. This was because all those missing entries are meant to represent that the review has 0 "helpful" reviews, so we changed it to better represent this and so we can model the data accordingly.

1.4. Visualizing Preliminary Data[1]

In the appendix, the first graph shows the class balance of reviews, divided into two categories: those with fewer than 5 reviews and those with 5 or more reviews. The tall bar on the left indicates that the vast majority of cases fall into the " less than 5 Reviews" category, while only a small portion belongs to the " more than or equal 5 Reviews" category. This imbalance suggests that most reviews in the dataset receive very little engagement, and only a minority achieve higher visibility or interaction. For modeling purposes, this imbalance is important to note because it can bias predictions toward the dominant class unless techniques like resampling or class weighting are applied.

The second graph shows the relationship between text length and helpful votes on a logarithmic scale. Most reviews cluster at shorter lengths (under about 5,000 characters) and receive relatively few helpful votes (often fewer than 10). However, some reviews that are longer do occasionally attract more helpful votes, with a few outliers reaching hundreds or thousands. The log scale on the y-axis makes it easier to see this variation, since without it, the small differences among the majority of reviews would be compressed at the bottom of the graph. The pattern suggests that while longer reviews may have a better chance of being judged as helpful, length alone does not guarantee more helpful votes; many long reviews still receive very few.

Together, these graphs provide insight into the nature of the dataset. The class balance graph emphasizes the rarity of widely reviewed items, while the scatterplot highlights that helpfulness is influenced by more than just the length of the review. Both plots indicate that the dataset has skewed distributions, which has implications for data analysis and modeling approaches. This will help give us insight into which patterns we should look at when we start to develop a predictive model.

2. Methods and Results

2.1. Methodology and Modeling Overview[2]

The objective of this project is to predict whether an Amazon product will receive more than 5 helpful votes at the time the review is posted. Our approach is to use a binary classification model using variables that are known at the time of the posting, so that our model is reflective of a realistic situation.

In the data section, we had already cleaned and processed the data so that we could use it for analysis. We had removed variables such as "image" and "style" because they had too many missing variables which did not serve a good purpose for our model. All missing values in the "vote" column were turned into 0, which reflects a review that had received no

helpful reviews.

To define the target variable, we had a binary threshold of greater than 5 votes. Reviews that got more than 5 votes were deemed "helpful" while those that received less than 5 votes were deemed "not helpful". Text features were processed into numerical representations which were better suited for modeling. Numerical variables such as star rating and review length were scaled as needed.

Because the "helpful" outcome is rare in the dataset and occurs long after posting, we restricted our features strictly to those available at posting time, including numeric attributes (rating, verification status, text length), review and product-level aggregates, and the review summary processed through TF-IDF and dimensionality reduction.

Class imbalance was a major characteristic of the dataset, with many of the votes receiving less than 5 votes. To address this, we use class weighting to help mitigate the bias within the model from being too tipped one way. Given the chronological nature of reviews, we designed our workflow to mimic realistic deployment conditions: models train on earlier reviews and forecast outcomes for later ones. This time-based structure prevents information leakage and ensures our evaluation reflects how such a system would perform in practice.

2.2. Model Overview and Justification[1]

Our first model will be the very simple k-Nearest Neighbors model (KNN). KNN predicts an outcome for a given review based on the most similar reviews in the training data, making this a good way of establishing if reviews containing similar features such as length, rating, or sentiment tend to all receive similar levels of helpfulness. While intuitive and easy to implement, KNN suffers some drawbacks, such as sensitivity to feature scaling, and it slows with large datasets; it will primarily be used here as a point of comparison to other approaches.

Then, we will apply Logistic Regression to construct a more robust yet more interpretable model. Logistic regression performs estimation of the probability that a given review is helpful and helps in identifying which features bear the strongest influence on that outcome. Its simplicity and interpretability are reasons it can serve well as a kind of benchmark for understanding the key factors behind review helpfulness.

We will also be employing Principal Component Analysis in order to reduce the dimensionality of the data, particularly for text-based and numeric features (Li, 2025). This will keep most of the important variation information in a smaller number of components, thus helping models run more efficiently and limiting the possibility of overfitting.

Finally, we will test a small Neural Network to grasp more complex relationships between variables. Neural networks can model the kind of pattern that might elude simpler models, especially in text features like review summaries. Since our data is imbalanced and pretty big, we have chosen a lightweight structure in order to keep the training stable and the results interpretable (Li, 2025)

In general, this combination of models will enable us to make a comparison of various modeling approaches in view of predicting which reviews are likely to receive more than five helpful votes.

2.3. Model Training Procedure[1]

The training process begins by dividing the data into training and testing sets, ensuring that the model is evaluated on reviews it has not seen before. A time-based split is used so that the model trains on earlier reviews and tests on later ones, reflecting a realistic prediction scenario in which future data is unknown at training time. To prepare the data for modeling, we combined numerical and textual information in a unified preprocessing pipeline. All numerical variables, including overall rating and review length, were standardized to ensure that no single feature dominates distance-based methods. For text processing, summaries were vectorized using TF-IDF with unigram and bigram features. Because full TF-IDF matrices are high-dimensional and computationally expensive, we applied Truncated SVD to reduce these representations to 100 principal components, retaining most of the meaningful variation while improving model speed and reducing overfitting risk.

Each model was wrapped in the same scikit-learn pipeline, allowing identical preprocessing procedures and making the final comparisons fair and consistent. Since the dataset is imbalanced, class weighting was applied so that the minority class, representing helpful reviews, has a stronger influence during training. Hyperparameter tuning was performed using a rolling TimeSeriesSplit, ensuring that each validation fold used data occurring after its corresponding training segment. Logistic Regression in particular benefited from tuning of the regularization parameter C, and we selected the class-weighted version of the model to counteract imbalance. For the neural network, early stopping prevented overfitting, especially given the scarcity of positive cases. Across all models, hyperparameters were selected using Precision-Recall AUC as the primary tuning metric since it best reflects performance under severe class imbalance. F1 score and ROC-AUC were also monitored to verify that model behavior remained stable across thresholds.

For the K-Nearest Neighbors model, we evaluated a range of values for k (from 3 to 21) and compared uniform vs. distance-based weighting. For Logistic Regression, we tuned the regularization strength by testing values of C

including 0.001, 0.01, 0.1, 1, and 10, while keeping the penalty fixed at L2. Finally, for the Neural Network (MLP), we varied the size of the hidden layers, testing architectures such as one hidden layer with 64 units, two layers with 128 and 64 units, and two layers with 64 and 32 units. We also tuned the regularization strength (alpha values of 0.0001, 0.001, and 0.01) and used early stopping to prevent overfitting.

Across all three models, the selected configurations represented the strongest balance between identifying rare helpful reviews and maintaining generalization over time. This validation process ensured that each model was trained and assessed under consistent and realistic deployment conditions, strengthening the reliability of our final comparisons.

2.4. Model Validation Plan[3]

Building on the training procedure above, we validate the helpful (more than 5 votes) classifier with a chronological split to mirror deployment and avoid look-ahead bias. Reviews are ordered by timestamp and divided into Train (oldest approx. 70%), Validation (next approx. 15%), and Test (most recent approx. 15%); the test window remains locked until final scoring. All preprocessing such as imputation, scaling, TF-IDF, and optional TruncatedSVD is wrapped in a single scikit-learn pipeline fit only on the current training fold to prevent leakage. Within the training slice, we tune hyperparameters using a rolling TimeSeriesSplit, so every validation fold uses later data than its training portion. Because the positive class is minority, our primary metric is PR-AUC; we also report Brier score with a reliability curve to assess calibration. The classification threshold is chosen only on the validation window by maximizing F1 and then frozen before scoring the test set. To address imbalance during training, we use class_weight='balanced' and display the baseline precision (prevalence) alongside PR curves for context. Final test reporting includes PR-AUC, ROC-AUC, Brier score, the confusion matrix at the frozen threshold, and 95% confidence intervals via CV fold variability (or a blocked bootstrap by week if time permits). We also include brief drift checks and a short error analysis of high-confidence false positives/negatives to ensure the chosen operating point is stable and interpretable.

2.5. Other Details About Model Implementation[2]

All models will be through Python. All results and codes will be in our final results code book presented in our machine learning project repository. As mentioned before KNN and logistic regression will be used for the model. Additionally, we will be implementing class weighting to make sure that results are not biased and that we can get the most accurate results for this project. All codes will be run through Github and through ipynb files, and some debugging and drafting assistance was provided by large language models (OpenAI, 2025). In addition, preprocessing and feature coding will be implemented to ensure that input variables capture relevant information without introducing bias or data leaks which would vastly improve the model's ability to generalize positive or negative reviews. We also plan to evaluate the relative importance of different features and compare how various preprocessing strategies influence the results. By analyzing text and the numeric parts together, we aim to identify which parts will contribute the most to the model in the next portion of the project.

2.6. Main Results and Interpretation[1]

Because the dataset is extremely imbalanced, with only a small fraction of reviews receiving more than five helpful votes, the majority-class classifier served as our primary benchmark. This classifier predicts every review as "not helpful," which results in high accuracy but a PR-AUC of only about 0.03. ROC-AUC for this baseline hovers near 0.50, consistent with random guessing.

These benchmark values provide the reference point for interpreting our model results. Any model aiming to be useful must exceed this PR-AUC benchmark, since PR-AUC is the most informative metric under severe class imbalance. Similarly, improvements over the 0.50 ROC-AUC baseline indicate that the model meaningfully distinguishes helpful from non-helpful reviews. With this context, we evaluate the performance of KNN, Logistic Regression, and the Neural Network.

Among all tested models, Logistic Regression with class weighting produced the most stable and interpretable performance. The optimal configuration utilized a regularization strength of $C = 0.1$. On the held-out test window, the logistic model achieved a ROC-AUC of approximately 0.89, indicating strong ranking ability. The ROC curve rose significantly above the 45 degree baseline, showing that the model consistently assigned higher probabilities to reviews that ultimately received more than five helpful votes.

However, as expected given the rarity of helpful reviews, the Precision–Recall performance was modest, with a PR-AUC near 0.13. The PR curve showed high precision at very low recall, meaning the model can confidently identify a small subset of helpful reviews but struggles to capture the full set of positives without sacrificing precision. This pattern aligns with the dataset's extreme imbalance, even well-ranked positive cases occur too infrequently for precision to remain high once the threshold is lowered.

Threshold selection on the validation set improved F1 marginally, but the underlying class scarcity remained the dominant limiting factor. Together, the PR and ROC curves show that the model distinguishes helpful vs. non-helpful

reviews well, but the rarity of the positive class limits the achievable precision at broader recall levels.

These outcomes suggest that helpfulness is influenced by nuanced, context-dependent user behavior and remains only partially predictable using metadata and summary text alone.

Because accuracy is uninformative with extreme class imbalance, our primary metric was Average Precision (PR-AUC). This metric emphasizes ranking quality and the model's ability to identify rare helpful reviews. ROC-AUC, F1, Brier score, and confusion matrices were reported for completeness but interpreted within the limitations imposed by class imbalance.

2.6.1. K-Nearest Neighbors[2]

Table 1. KNN Performance and Confusion Matrix (Threshold = 0.35)

Metric	Validation	Test
Accuracy	0.8433	0.9302
Precision	0.2588	0.1236
Recall	0.3708	0.1642
F1 Score	0.3048	0.1410
ROC AUC	0.7392	0.6715
PR AUC	0.2066	0.0760
Brier Score	0.1016	0.0458
Confusion Matrix Counts		
True Negatives (TN)	1554	1776
False Positives (FP)	189	70
False Negatives (FN)	112	56
True Positives (TP)	66	11

The KNN model performed much worse than the logistic regression model and demonstrated limitations with this data. After tuning, the best configuration used was k=11 with distance weighting, and the threshold yielding the highest validation F1 was 0.35. The confusion matrices reveal that the KNN predicted the majority of the "not helpful" class, identifying only 11 true positives out of 67 actual helpful reviews in the rest set. The KNN test failed to generalize and offered limited practical utility for predicting helpful reviews.

2.6.2. Logistic Regression[2]

The logistic regression with a regularization strength of C =0.1 and evaluated at the best validation threshold of 0.475, had the strongest and most stable performance among all tested models. The confusion matrices shows that the model identified 31 true positives but still missed many helpful reviews, which is common amongst imbalanced

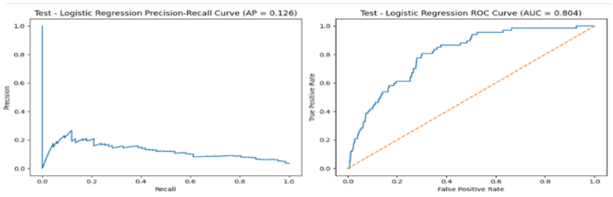


Figure 1. Logistic Regression Precision-Recall and ROC curves.

Table 2. Logistic Regression Performance and Confusion Matrix (Threshold = 0.475)

Metric	Validation	Test
Accuracy	0.7579	0.8730
Precision	0.2338	0.1297
Recall	0.7079	0.4627
F1 Score	0.3515	0.2026
ROC AUC	0.7934	0.8044
PR AUC	0.2170	0.1264
Brier Score	0.1961	0.1123
Confusion Matrix Counts		
True Negatives (TN)	1330	1646
False Positives (FP)	413	208
False Negatives (FN)	52	36
True Positives (TP)	126	31

data. However, it did demonstrate strong generalization across time, and proved to be the most reliable.

2.6.3. Neural Network (MLP)[2]

Table 3. Performance Comparison of Models

Model	Acc.	Prec.	Rec.	F1	ROC	PR	Thr.
KNN	0.930	0.124	0.164	0.140	0.744	0.067	0.350
Log. Reg.	0.873	0.130	0.463	0.203	0.804	0.126	0.475
MLP Net	0.810	0.079	0.420	0.133	0.744	0.100	0.250

The MLP was tun False negatives typically occurred on short or vague summaries that looked unhelpful at posting time but later accumulated helpful voted. Due to our features relying on the summary text, these reviews lacked enough signal for the models to classify them as helpful. In contrast, false positives were usually long or enthusiastic summaries that appeared helpful in wording but ultimately received little community engagement. These reviews often had stylistic markers that the model interpreted as helpful even though readers did not agree. The MLP overproduced these false positives due to its sensitivity in text patters while the logistic regression made fewer but more balanced errors. Overall, these patterns reflect the limitations of using the summary text alone.

Table 4. MLP Performance and Confusion Matrix (Threshold = 0.250)

Metric	Validation	Test
Accuracy	0.8022	0.8100
Precision	0.2595	0.0791
Recall	0.6124	0.4179
F1 Score	0.3645	0.1330
ROC AUC	0.8236	0.7444
PR AUC	0.3128	0.0995
Brier Score	0.1047	0.0865
Confusion Matrix Counts		
True Negatives (TN)	1432	1528
False Positives (FP)	311	326
False Negatives (FN)	69	39
True Positives (TP)	109	28

3. Conclusion

3.1. Specific Insights[1]

Across all models and evaluation metrics, our analysis shows that predicting whether a review will become helpful at posting time is possible, but only to a limited extent. Logistic Regression performed the strongest, suggesting that much of the predictive signal lies in linear relationships driven by summary length, sentiment indicators, verification status, and product historical patterns. However, the modest Precision–Recall performance across all models highlights that helpfulness is shaped by nuanced reader behavior that cannot be fully captured by metadata or summary text alone. Short or vague summaries often became helpful, while long and enthusiastic summaries frequently appeared promising but received little engagement. This reinforces that helpfulness is not solely a property of the text but is context dependent and influenced by social dynamics, timing, and visibility.

3.2. Impact Statement[1]

While our models cannot perfectly identify all future helpful reviews, their ranking ability offers practical value. Platforms like Amazon could use these predictions to surface potentially helpful reviews earlier, especially those with moderate predicted helpfulness but lower visibility, improving the user experience (Hou et al.). Future work could substantially improve performance by incorporating additional features beyond the summary, such as full review text, reviewer trust metrics, temporal patterns of engagement, product category trends, or richer linguistic features. More sophisticated models, including transformers or attention-based architectures, may also capture deeper semantic cues that traditional models miss. Adjusting for extreme imbalance through more advanced resampling methods or focal loss could further enhance recall without sacrificing precision.

3.3. Defense Against Criticism[1,3]

Although our models did not achieve high recall or perfectly identify all future helpful reviews, these limitations are inherent to the structure of the dataset rather than flaws in our approach. Helpfulness is an extremely rare event, and evaluating reviews only at posting time excludes many of the engagement-based signals that ultimately drive helpful voting behavior. Our use of class weighting, time-aware splits, dimensionality reduction, and multiple model types represents a methodologically sound approach to working within these constraints. Furthermore, our analysis prioritizes transparency and reproducibility over overly complex methods that may appear to perform better but fail to generalize or meaningfully explain why reviews become helpful. In this context, our findings should be interpreted not as a deficiency of the modeling framework, but as a reflection of the inherent difficulty of predicting human-driven outcomes from limited, early-stage review features. We also acknowledge two specific structural constraints regarding feature selection and target definition. First, our decision to model based on the `summary` rather than the full `reviewText` was driven by computational feasibility; however, this design effectively isolates the predictive power of the 'headline,' serving as a proxy for the initial user attention span in high-volume scrolling environments. Second, while the target variable (cumulative votes) inherently favors older reviews which have had more time to accrue engagement, our strict chronological split mitigates this bias. By training on past data and testing on future data, we force the model to learn traits of helpfulness rather than simply memorizing review age, ensuring the evaluation mirrors a real-world deployment where new reviews start with zero votes.

3.4. Overall Summary[1]

Taken together, our findings demonstrate that helpfulness prediction is feasible but fundamentally constrained by the complexity of human evaluation behavior. Logistic Regression emerged as the most reliable and interpretable model, outperforming both KNN and the MLP in terms of stability and generalization across time. While the dataset's imbalance limited precision at broader recall levels, the models consistently ranked truly helpful reviews above unhelpful ones, indicating meaningful signal in the features available at posting. Ultimately, this project highlights both the potential and limitations of predictive modeling in online review contexts, and it establishes a clear foundation for future improvements in data collection, modeling strategies, and feature design.

4. References[1,2,3]

Bridging Language and Items for Retrieval and Recommendation Yupeng Hou, Jiacheng Li, Zhankui He, An Yan, Xiusi Chen, Julian McAuley *arXiv*

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning: With Applications in R.* Springer.

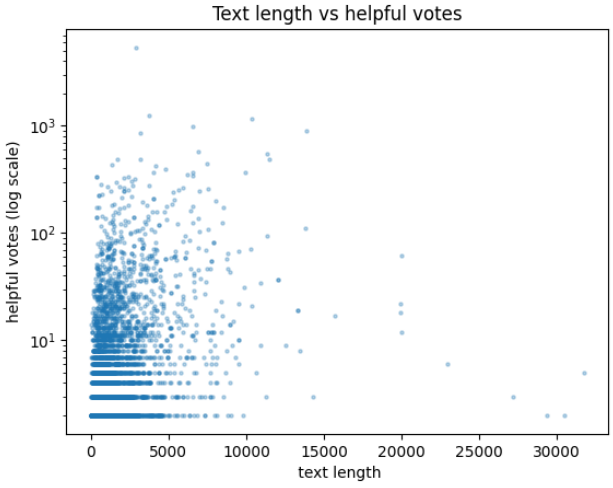
Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), Proceedings of the 17th International Conference on Machine Learning (ICML 2000), pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

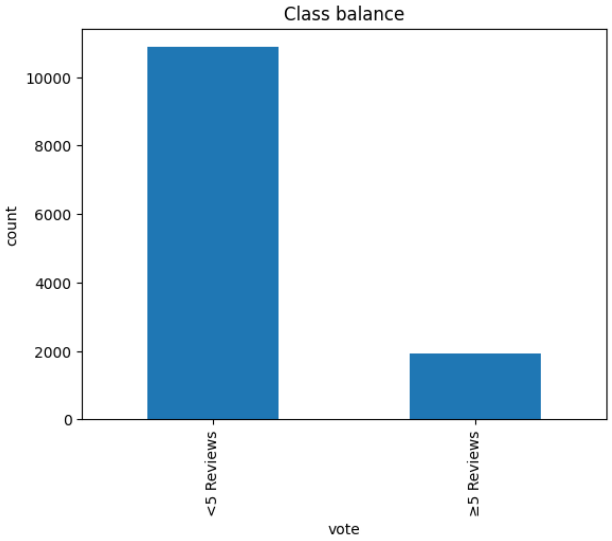
Li, L. (2025). PCA Notes. DS 3001: Machine Learning. University of Virginia. Unpublished class material.

Ni, J., Li, J., & McAuley, J. (2019). *Justifying recommendations using distantly-labeled reviews and fine-grained aspects*. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics.

OpenAI. (2025). ChatGPT [Large language model]. https://chat.openai.com/

A. Appendix.


Text length vs helpful votes
helpful votes (log scale)
10^3
10^2
10^1
0
5000
10000
15000
20000
25000
30000
text length


Class balance
count
10000
8000
6000
4000
2000
0
<5 Reviews
≥5 Reviews
vote