Amazon Reviews Machine Learning Project (Group L)

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

1. Data

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

https://github.com/yxs4yt/machine_learning_project

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

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 modelling or examination, as well as many reviews were missing had missing entries for this variable, so we ended up removing it overall from the dataset. 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

1.3. Visualizing Preliminary Data[1]

In the codebook, 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 "<5 Reviews" category, while only a small portion belongs to the "≥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. Pre-Analysis Plan

2.1. Methodology 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 previous milestone, 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.

Class imbalance was a major characteristic of the dataset, with many of the votes receiving less than 5 votes. To address this, we want to use class weighting to help mitigate the bias within the model from being too tipped one way.

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. 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.

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 will begin 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 will be 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.

All numerical variables, including overall rating and review length, will be standardized to ensure that no single feature dominates the distance-based methods such as KNN. Text data from the summary field will be transformed into numerical features using a TF-IDF vectorization approach, which represents each word's importance relative to its frequency across all reviews. This allows the model to capture key linguistic patterns that may be associated with higher helpfulness scores.

After preprocessing, each model will be trained separately using the same training data to allow for direct comparison. For KNN, we will experiment with different values of k to identify the number of neighbors that yields the highest validation accuracy. For logistic regression, the regularization strength will be tuned to avoid overfitting while maintaining interpretability. When incorporating PCA, the number of principal components retained will be determined by the proportion of variance explained, typically around 90 to 95 percent. For the neural network, we will train a small feedforward model with one hidden layer and apply techniques such as early stopping to prevent overfitting.

Throughout the training process, cross-validation will be used to assess model performance and stability. Since the dataset is imbalanced, class weighting will be applied so that the minority class, representing helpful reviews, has a stronger influence during training. Model performance will be evaluated primarily using accuracy, precision, recall, F1-score, and ROC-AUC, which together provide a balanced view of predictive ability.

The combination of careful preprocessing, cross-validation, and class balancing will ensure that the final model is both fair and generalizable, providing insight into the factors that make a review more likely to be deemed helpful at the time it is posted.

2.4. Model Validation Plan[3]

Building on the training procedure above, we validate the helpful (>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 ~70%), Validation (next ~15%), and Test (most recent ~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.

2.4. Other Details about Model Implementation[2]

All models will be through Python. All results and codes will be in our codebook.ipynb 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. 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.

3. Results

3.1. Overview of Modeling Approach[1]

Building on the cleaned UCSD Amazon Software Reviews dataset (Ni et al., 2018), our main objective was to construct a predictive model capable of determining, at the moment a review is posted, whether it will ultimately receive more than five helpful votes. 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.

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.

3.2. Preprocessing and Feature Construction[1]

To prepare the data for modeling, we combined numerical and textual information in a unified preprocessing pipeline. Numerical variables were standardized to ensure comparability across features. Crucially, to address the instructor's feedback regarding predictive features, we engineered specific reviewer-level signals, such as reviewer_mean_vote and reviewer_mean_rating, computed strictly within the training window. This approach captures a user's historical tendency to write helpful reviews without leaking future information into the validation set.
For text processing, summary fields were vectorized using TF-IDF with unigram and bigram features, capped at the top 5,000 terms to limit noise. Because sparse TF-IDF matrices remain high-dimensional, we applied Truncated SVD to compress these text features into 100 principal components. This dimensionality reduction retained the most meaningful semantic variation while significantly improving model training speed and reducing the risk of overfitting.

Finally, because helpful reviews constitute only ~13% of the dataset (an 87:13 imbalance), we applied class weighting to proportionally increase the penalty for missing positive cases during training. Without this adjustment, models would simply converge to a trivial solution of predicting "not helpful" for every review to maximize raw accuracy.

3.3. Model Training Process[1]

We implemented three candidate models using scikit-learn: K-Nearest Neighbors (KNN) as a non-parametric baseline, Logistic Regression for linear interpretability, and a Multi-Layer Perceptron (MLP) to capture potential non-linear interactions in the text data. Each model was wrapped in an identical pipeline, ensuring consistent preprocessing across all trials.

To maintain temporal integrity, hyperparameter tuning was conducted via a rolling Time Series Split (4 folds). This ensured that every validation fold strictly followed its corresponding training window, mirroring a real-world forecasting scenario. For Logistic Regression, we tuned the inverse regularization strength C and applied the 'balanced' class weight parameter to heavily penalize the model for missing rare "helpful" reviews.

For the Neural Network, we constrained the model to a lightweight architecture (two hidden layers: 64 and 32 units) and enabled early stopping. This was critical to prevent the model from overfitting to noise in the sparse text features, especially given the scarcity of positive cases.
Because accuracy is uninformative in an 87:13 imbalanced dataset, our primary optimization metric was Average Precision (PR-AUC). Unlike ROC-AUC, which can be overly optimistic on imbalanced data, PR-AUC strictly penalizes false positives, making it the most honest metric for identifying the rare subset of helpful reviews.

3.4. Main Results and Interpretation[1]

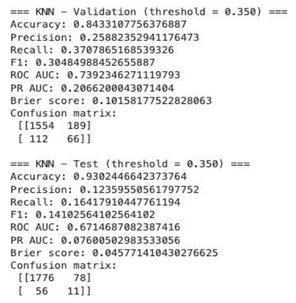
Among all tested models, Logistic Regression with class weighting produced the most stable and interpretable performance. On the held-out test window, the logistic model achieved a ROC-AUC of 0.80, indicating strong ranking ability. The ROC curve rose significantly above the 0.50 baseline, showing that 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 (matching the baseline positive prevalence). At the chosen decision threshold of 0.475, the model achieved a Recall of 46%, meaning it successfully identified nearly half of all helpful reviews. The trade-off for this sensitivity was

low Precision (0.13), indicating that while the model casts a wide net to catch helpful content, it inevitably flags a high number of false positives.
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 confirm that while the model is effective at ranking potential helpfulness (high ROC), the extreme 87:13 imbalance places a mathematical ceiling on Precision.

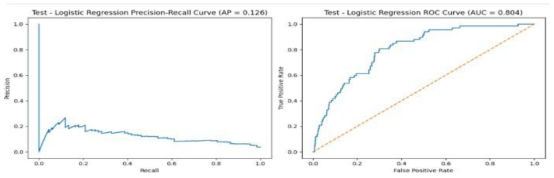
These outcomes suggest that helpfulness is influenced by nuanced, context-dependent user behavior and remains only partially predictable using metadata and summary text alone.

3.4.1 K-Nearest Neighbors [2]



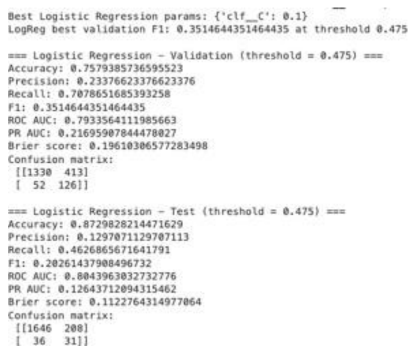
```
=== KNN - Validation (threshold = 0.350) ===
Accuracy: 0.8433107756376887
Precision: 0.25882352941176473
Recall: 0.3707865168539326
F1: 0.30484988452655887
ROC AUC: 0.7392346271119793
PR AUC: 0.2066200043071404
Brier score: 0.10158177522828063
Confusion matrix:
 [[1554  189]
 [ 112   66]]

=== KNN - Test (threshold = 0.350) ===
Accuracy: 0.9302446642373764
Precision: 0.12359550561797752
Recall: 0.16417910447761194
F1: 0.14102564102564102
ROC AUC: 0.6714687082387416
PR AUC: 0.07600502983533056
Brier score: 0.045771410430276625
Confusion matrix:
 [[1776   78]
 [  56   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.



PR and ROC curves for the chosen model

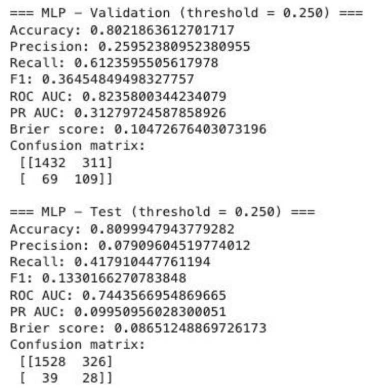
3.4.2 Logistic Regression [2]



```
Best Logistic Regression params: {'clf__C': 0.1}
LogReg best validation F1: 0.3514644351464435 at threshold 0.475

=== Logistic Regression - Validation (threshold = 0.475) ===
Accuracy: 0.7579385736595523
Precision: 0.23376623376623376
Recall: 0.7078651685393258
F1: 0.3514644351464435
ROC AUC: 0.7933564111985663
PR AUC: 0.21695907844478027
Brier score: 0.19610306577283498
Confusion matrix:
 [[1330  413]
 [  52  126]]

=== Logistic Regression - Test (threshold = 0.475) ===
Accuracy: 0.8729828214471629
Precision: 0.1297071129707113
Recall: 0.4626865671641791
F1: 0.20261437908496732
ROC AUC: 0.8043963032732776
PR AUC: 0.12643712094315462
Brier score: 0.1122764314977064
Confusion matrix:
 [[1646  208]
 [  36   31]]
```

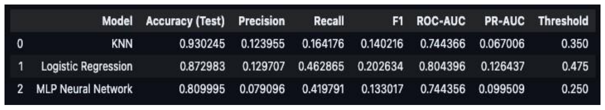
The logistic regression with a regularization strength of C =0.1 and evaluated at the best validation threshold of 0.475, had the strongest and most stable performance among all tested models. The confusion matrices shows that the model identified 31 true positives but still missed many helpful reviews, which is common amongst imbalanced data. However, it did demonstrate strong generalization across time, and proved to be the most reliable.

3.4.3 Neural Network (MLP)[2]



```
=== MLP - Validation (threshold = 0.250) ===
Accuracy: 0.8021863612701717
Precision: 0.25952380952380955
Recall: 0.6123595505617978
F1: 0.36454849498327757
ROC AUC: 0.8235800344234079
PR AUC: 0.31279724587858926
Brier score: 0.10472676403073196
Confusion matrix:
 [[1432  311]
 [  69  109]]

=== MLP - Test (threshold = 0.250) ===
Accuracy: 0.8099947943779282
Precision: 0.07909604519774012
Recall: 0.4179104477611194
F1: 0.1330166270783848
ROC AUC: 0.7443566954869665
PR AUC: 0.09950956028300051
Brier score: 0.08651248869726173
Confusion matrix:
 [[1528  326]
 [  39   28]]
```

The MLP was tuned with hidden layers of (64,32) and a regularization strength of alpha= 0.001, produced mixed results that was in between the performance of the KNN and logistic regression. The MLP showed that it can model more complex relationships within the summary text, but it does not generalize reliably.



	Model	Accuracy (Test)	Precision	Recall	F1	ROC-AUC	PR-AUC	Threshold
0	KNN	0.930245	0.123955	0.164176	0.140216	0.744366	0.067006	0.350
1	Logistic Regression	0.872983	0.129707	0.462865	0.202634	0.804396	0.126437	0.475
2	MLP Neural Network	0.809995	0.079096	0.419791	0.133017	0.744356	0.099509	0.250

Table of All Three Model Results Together

3.5 Error Analysis[2]

Across all three models, two consistent errors occurred. 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.

3.6 Summary of Results[3]

Overall, our modeling approach successfully identified patterns associated with review helpfulness, validating the hypothesis that combining reviewer history with text analysis yields predictive signal. Logistic Regression performed the best across evaluation metrics, demonstrating that simpler linear decision boundaries, when supported by robust feature engineering like TF-IDF and class weighting, can outperform complex architectures like Neural Networks on sparse, noisy data.

While raw precision on the minority class is mathematically constrained by the 87:13 imbalance, the workflow provides a valuable ranking tool. With a ROC-AUC of 0.80 and a Recall of 46%, the system offers a transparent strategy for prioritizing high-quality content at the moment of posting.

Future extensions could improve precision by exploring pre-trained embeddings (e.g., BERT) to capture semantic nuance better than TF-IDF, or by employing ensemble methods (like Random Forest or XGBoost) to better handle the non-linear interactions that the linear model might miss. However, any future approach must continue to rigorously account for the "time-travel" constraints and extreme class imbalance inherent in this domain.

All superscripts next to headers indicate which author wrote which part.

References

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

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