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]

We will begin with k-Nearest Neighbors (KNN) as an intuitive baseline model. KNN makes predictions based on the most similar observations in the training data, allowing us to visualize how reviews with similar characteristics (e.g., length, rating, sentiment) tend to cluster together in terms of helpfulness. While simple and easy to implement, KNN can be sensitive to scale and computationally expensive for large datasets, so it will primarily serve as an introductory comparison.

Next, we will implement Logistic Regression, which provides a probabilistic interpretation of helpfulness and offers clear insight into which predictors most influence the likelihood of receiving more than five votes. Because logistic regression assumes a linear relationship between features and log-odds of the outcome, it will help us establish a strong and interpretable benchmark for the project.

We also plan to incorporate Principal Component Analysis (PCA) as a dimensionality reduction step for our text-based and numeric features. PCA will help simplify high-dimensional representations into a smaller number of components while retaining most of the variance. This approach can improve model efficiency and reduce the risk of overfitting.

Finally, to explore nonlinear relationships, we will test a simple Neural Network model. Neural networks can capture more complex feature interactions and patterns in the data, particularly from the text summaries, compared to linear models. However, due to the class imbalance and dataset size, we will keep

the architecture small to balance interpretability and performance.

Together, these models reflect a progression from simple to more complex approaches covered in the course. This will allow us to compare models in terms of interpretability, computational efficiency, and predictive accuracy, and determine which approach most effectively predicts whether a review is likely to be deemed helpful.

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.

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.