Harnessing Helpfulness: Amazon Product Reviews to Elevate Marketplace Experience

Sneha Karri

Department of Applied Data Science, San Jose University

DATA 270: Data Analytics

Professor Dr. Linsey Pang

May 06, 2024

Abstract

In the digital era, online reviews wield significant influence over customer decisions, notably on platforms like Amazon. This project seeks to improve review utility by categorizing them based on product types and predicting their helpfulness through machine learning leveraging classification algorithms to analyze features extracted from review text, vocabulary richness, and metadata, aiming to augment users with insights for informed purchasing decisions. We aim to perform text processing on raw review text features using techniques like TFIDF, Lemmatization, Stopword removal, etc., to generate features. These features will be processed using machine-learning models like Logistic Regression, Random Forest, Support Vector Machine, and XGBoost algorithms offering unique advantages such as simplicity, robustness, and efficiency. We achieved a peak accuracy of 93.75% in Cellphones category and measured success using other metrics like precision, recall, area under the curve (AUC), and F1 score. Our primary objective is to develop predictive models that evaluate review helpfulness and its drivers across various Amazon product categories. We will integrate the predictive models into the review-writing process by providing customers with keywords to enhance review quality. We will also enable sellers to optimize product listings by using critical keywords to cater to customer needs. Project outcomes have broad applications like enhancing consumer access to relevant information, improving Amazon product listings, and enhancing review effectiveness.

Keywords: Amazon, e-commerce, predictive models, review helpfulness, classification algorithms

Modeling

4.1. Model Proposal

For the project to analyze and improve the value of customer evaluations across a variety of product categories, machine learning methods like Support Vector Machine (SVM), Random Forest, XGBoost, and Logistic Regression are essential. To further understand the dynamics of review helpfulness, four categories which include appliances, cellphones and accessories, automobiles, and tools and home improvement that have been specifically explored within the project's scope. These methods are applied for modeling purposes, which focuses on the category of cellphones and accessories. Logistic Regression serves as a foundational tool for predicting the likelihood of a review being helpful based on various features extracted from the reviews (Hudgins et al., 2023), while Random Forest model experimented by Abhilasha Singh Rathor et al. (2018) to handle complex interactions within the data, thus contributing to accurate predictions. Furthermore, XGBoost, which is well-known for its effectiveness and performance, provides improved prediction by iteratively developing weak learners, which may be able to identify complex patterns in the data (M. I. Hossain et al., 2021). Finally, by utilizing parameters like review length, sentiment ratings, and product category to create appropriate decision boundaries inside the feature space, Support Vector Machine (SVM) assists in classifying reviews as useful or unhelpful (B. K. Shah et al., 2021). By utilizing these algorithms, important information is obtained that helps to achieve the main objective of enhancing the shopping experience for customers for cellphones and accessories category.

4.4.1. Logistic Regression

Logistic regression is widely used in machine learning applications, because it generates probabilities and uses discrete measurements to classify new samples. It classifies the binary dependent variable using supervised learning techniques. Helpfulness is the dependent variable for our project. Because the results of logistic regression are discrete, meaning that there are only two possible outcomes for a given variable in our project which is helpful or not helpful. Logistic regression makes sense when predicting a likelihood score between 0 and 1 or 0 and 1.

The likelihood values between 0 and 1 are provided by the logistic or sigmoid function. The solutions to a linear equation between 0 and 1 are squashed. This function fits an S-shaped curve that indicates, given the weighted input of metadata and word TF-IDF scores, whether the likelihood of a review is useful. The probability is represented by the y-axis on the sigmoid graph (review helpfulness). (1) defines this logistic function, often known as the sigmoid function.

$$\sigma(z) = \frac{1}{1 + e^{(-z)}} \tag{1}$$

Equation (1) represents the combination of input features, or metadata and processed text data, along with the corresponding regression coefficient weights as mentioned in (2).

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n \tag{2}$$

When applying linear hypothesis to the logistic sigmoid function, we obtain (3), where x is the input feature matrix and θ is the model's vector of parameters (coefficients). This function, which is parameterized by θ in (4), returns the likelihood that, given x, y = 1.

$$h_\theta(x) = \frac{1}{1 + e^{(-\theta^T x)}} \tag{3}$$

$$\mathrm{h(x)} = \mathrm{P(y} = 1|\mathrm{x};\ \theta) \tag{4}$$

Regarding h(x), values less than 0.5 are predicted to be 0, whereas values more than or equal to 0.5 are predicted to be 1. We employ maximum likelihood estimate to describe the link between input and target attributes. The most suited model is thus produced. Logistic regression's cost function is (5).

$$J(\theta) = - \frac{1}{n}\sum_{i=1}^{n}[y^i \; log(h_\theta(x^{(i)})) + (1 - y^i)log(1 - h_\theta(x^{(i)}))] \tag{5}$$

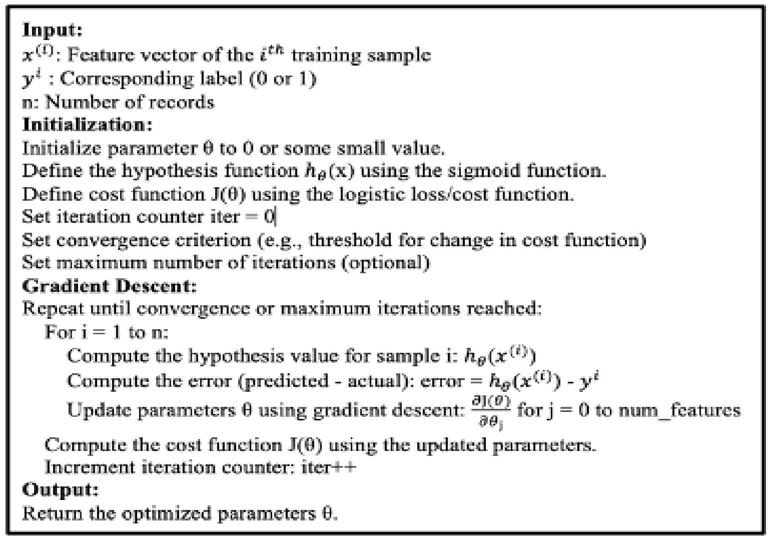
Our model's cost function will be minimized by using gradient descent to determine the ideal parameters. It is expressed as (6) and employs an iterative process.

$$\theta_{new} = \theta_{old} - \alpha \frac{\partial J(\theta)}{\partial \theta_j} \tag{6}$$

Figure 1 displays the logistic regression pseudo-algorithm, whereas Figure 2 displays the logistic regression architecture.

Figure 1

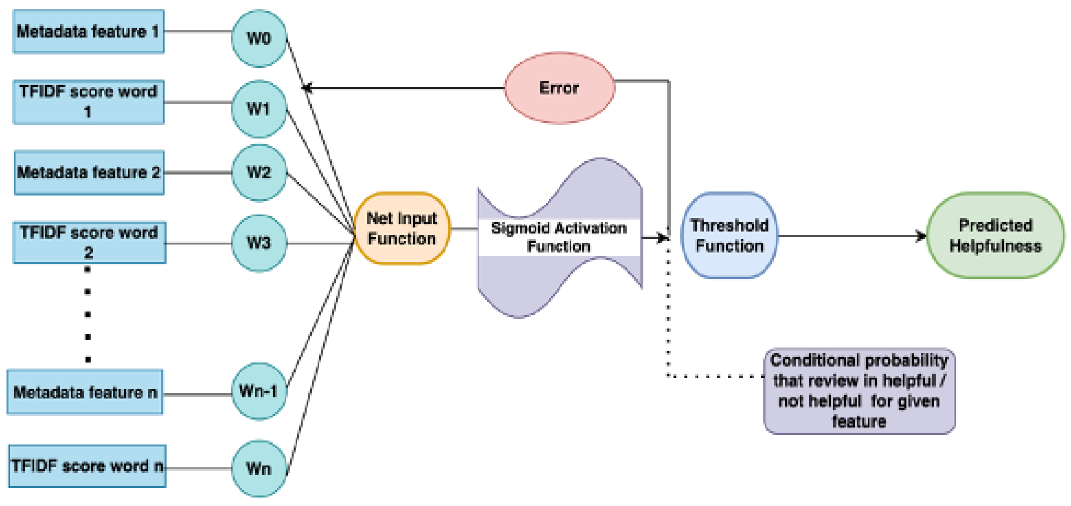
Algorithm of Logistic Regression Model



```
Input:
$x^{(i)}$: Feature vector of the $i^{th}$ training sample
$y^i$ : Corresponding label (0 or 1)
n: Number of records
Initialization:
Initialize parameter θ to 0 or some small value.
Define the hypothesis function $h_\theta$(x) using the sigmoid function.
Define cost function J(θ) using the logistic loss/cost function.
Set iteration counter iter = 0
Set convergence criterion (e.g., threshold for change in cost function)
Set maximum number of iterations (optional)
Gradient Descent:
Repeat until convergence or maximum iterations reached:
    For i = 1 to n:
        Compute the hypothesis value for sample i: $h_\theta(x^{(i)})$
        Compute the error (predicted - actual): error = $h_\theta(x^{(i)})$ - $y^i$
        Update parameters θ using gradient descent: $\frac{\partial J(\theta)}{\partial \theta_j}$ for j = 0 to num_features
    Compute the cost function J(θ) using the updated parameters.
    Increment iteration counter: iter++
Output:
Return the optimized parameters θ.
```

Figure 2

The architecture of the logistic regression model for review helpfulness



4.1.2. Support Vector Machine (SVM)

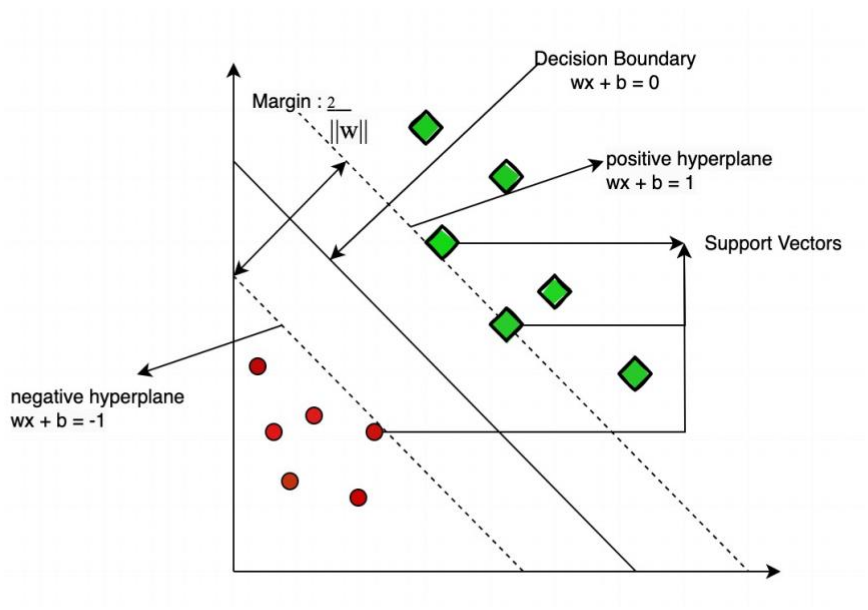
The most sophisticated, widely used, and effective technique for resolving classification issues is support vector machines (SVM). Face recognition, spam filtering, image classification, and stock price prediction are a few applications of SVM. While they can also be used to solve regression problems, SVM is mostly used to solve classification tasks.

In order to divide the n-dimensional data points into different classes, SVM creates a decision boundary that is represented as a hyperplane, which is a line in 2D or a plane in 3D. For example, accurately classifying new tweets is necessary in order to distinguish between true and false tweets. SVM makes this categorization easier by making sure that all of the points belong to one category and are on one side of the hyperplane (line, for example, for tweets), and on the other side, are points belonging to the other group. While there may be more than one hyperplane (or line) that can divide two classes, Support Vector Machines (SVM) seek to find the hyperplane that divides the two classes as best they can, keeping the hyperplane as far away from the closest datapoint in each class as feasible.

Each row in the dataset is known to be a vector in space. Thus, the term "supporting vectors" refers to the vectors (points) that are situated along the edges. We establish a decision boundary and assign the new data point to the appropriate category based on the support vector. It is known as support vector machines for this reason. The linear SVM demonstration is shown in Figure 3.

Figure 3

Linear Support Vector Machine Illustration



A hyperplane that maximizes the margin which is the distance between it and the nearest point on either side of each class is the aim. The requirement that every data point be on the correct side of the hyperplane guides this process. Since the data can be separated linearly, the margins are expressed in terms of the linear equation above. The max-margin classifier is the middle hyperplane, and the distance between them is referred to as the margin.

The equation for hyperplanes is shown in (7), (8), (9).

Hyperplane $$w^T . x_i + b = 0 \tag{7}$$

Negative Hyperplane $$w^T . x_i + b = -1 \tag{8}$$

Positive Hyperplane $$w^T . x_i + b = 1 \tag{9}$$

The kernel trick is an SVM technique used to handle non-linear data. The data is mapped via the kernel method into a higher-dimensional feature space, where it can be separated linearly. This is accomplished by applying kernel functions, which determine the similarity between data points in the higher-dimensional space. Examples of these kernel functions are the polynomial kernel and the radial basis function (RBF) kernel. The SVM may then describe non-linear decision boundaries by locating the ideal hyperplane in this altered feature space.

4.1.3. ***Random Forest***

The main objective of the Random Forest is to improve prediction accuracy while maintaining resilience and reducing the danger of overfitting in comparison to a single decision tree. This is achieved by training a large number of decision trees and producing a class that is representative of the majority vote of the classes that each tree predicted. With this ensemble approach, complex datasets may be performed more effectively by utilizing the capabilities of several learners by effectively capturing a range of patterns and correlations in the data. Regression and classification can both be accomplished with this technique. In classification, measures like entropy and Gini impurity are frequently used to evaluate the split quality. With the goal of reducing impurity across the network, these metrics calculate the impurity inside each group. For more precise classification, a more homogeneous grouping is indicated by lower impurity levels. In classification, recall, accuracy, and precision are additional metrics that are employed. The formula to calculate the Gini index is shown in (10).

$$Gini(p) = 1 - \sum_{i=1}^{n} p_i^2 \qquad (10)$$

where pi is the proportion of samples belonging to class i. The objective is to minimize the Gini impurity across all branches of the decision tree.

The formula used to calculate the entropy is shown in the (11).

$$E(S) = -\sum_{i=1}^{n} p_i log_2\, p_i \qquad (11)$$

where pi represents the proportion of data points in class i within the set S. The summation goes over all classes in the dataset.

The formula used to calculate the accuracy is shown in the (12).

$$Accuracy = \frac{Number\ of\ predictions}{Total\ number\ of\ predictions} \qquad (12)$$

The formula to calculate precision is shown in the (13).

$$Precision = \frac{TP}{TP + FP} \qquad (13)$$

where, TP is the number of true positives and FP is the number of false positives.

The formula to calculate recall is shown in the (14).

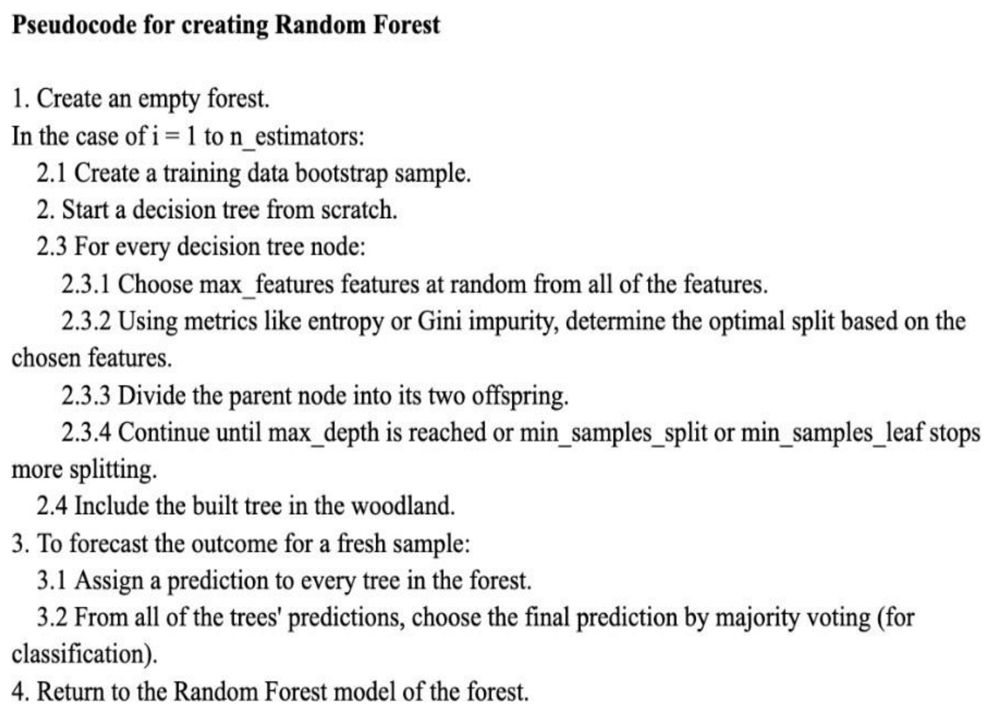
$$Recall = \frac{TP}{TP + FN} \qquad (14)$$

where, FN is the number of false negative.

The pseudo algorithm for Random Forest is shown in the Figure 4.

Figure 4

Algorithm for Random Forest



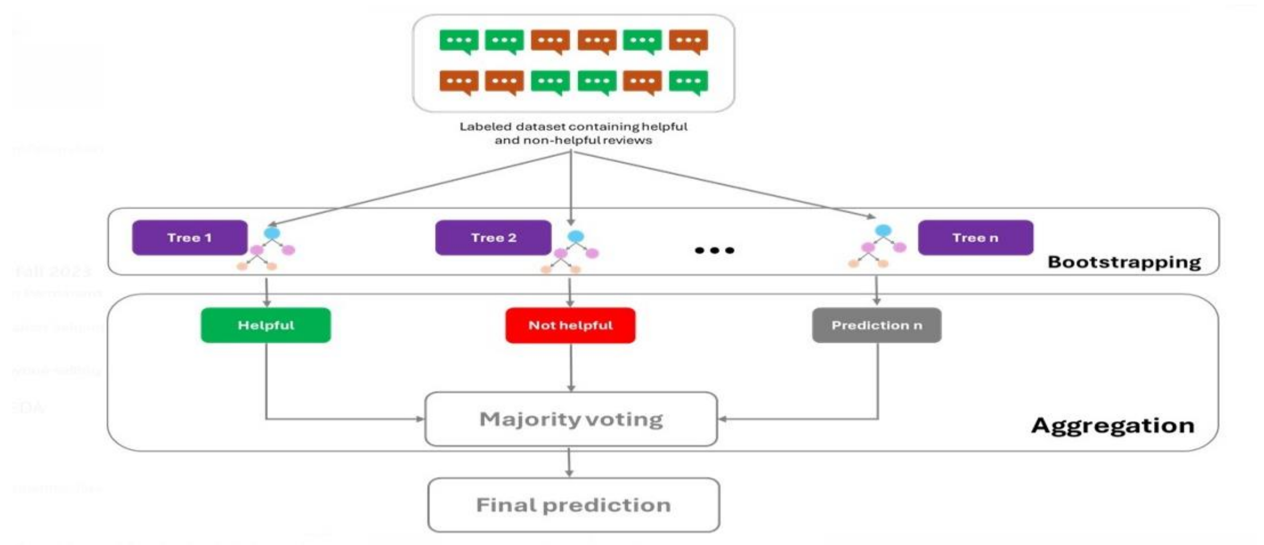
Pseudocode for creating Random Forest

```
1. Create an empty forest.
In the case of i = 1 to n_estimators:
    2.1 Create a training data bootstrap sample.
    2. Start a decision tree from scratch.
    2.3 For every decision tree node:
        2.3.1 Choose max_features features at random from all of the features.
        2.3.2 Using metrics like entropy or Gini impurity, determine the optimal split based on the
chosen features.
        2.3.3 Divide the parent node into its two offspring.
        2.3.4 Continue until max_depth is reached or min_samples_split or min_samples_leaf stops
more splitting.
    2.4 Include the built tree in the woodland.
3. To forecast the outcome for a fresh sample:
    3.1 Assign a prediction to every tree in the forest.
    3.2 From all of the trees' predictions, choose the final prediction by majority voting (for
classification).
4. Return to the Random Forest model of the forest.
```

The Random Forest algorithm, a dependable ensemble learning method that is used in machine learning for regression and classification tasks, is shown in Figure 5. The diagram shows the structure of the Random Forest, which is used to classify reviews as ‘helpful’ or ‘not helpful’. It begins with a pre-classified, tagged dataset of reviews that indicate whether they are helpful or not. The dataset is bootstrapped, which is a method used to train each forest tree independently by creating multiple subsets (replacement samples) of the original dataset. This leads to the production of several decision trees, called Tree 1 through Tree n, each of which is trained with a different bootstrap sample. Each of these trees predicts on its own whether the reviews will be helpful or not. Once these predictions are compiled and integrated using a majority voting approach, the final categorization for each review is determined by the majority vote across all trees. Through variance limitation and overfitting prevention, the ensemble technique enhances the accuracy and resilience of the model. The ultimate prediction, which is either "helpful" or "not helpful," reflects the forest's collective opinion.

Figure 5

Random Forest Model: Ensemble Learning through Bootstrap Aggregation and Decision Trees



Note. This is a Random Forest model which is an ensemble learning through Bootstrap Aggregation and Decision Trees.

A Random Forest is utilized, which leverages a variety of variables, to predict the usefulness of reviews. TF-IDF, which emphasizes unique terms in reviews by comparing their frequency in a document to their corpus frequency, and other textual characteristics are integrated into the model along with sentiment analysis scores from tools like NLTK, a bag of words model that counts word occurrences. Other factors considered include the attributes, such as the number of reviews a user has shared their prior helpful votes and average rating. Features like price, brand, category, and past sales data are used to contextualize reviews. The Random Forest model builds many decision trees, each using random features and data samples, to increase generalization. The nodes in each tree are divided, as previously noted, by entropy and Gini impurity, which measure dataset randomness and label homogeneity, respectively. Overfitting can be prevented and complexity can be managed by modifying parameters such as max depth and min samples split. Hyperparameter tweaking, which also includes the number of trees and the subset of characteristics considered at each split, ensures the efficacy and efficiency of the model. The user experience can be enhanced by the Random Forest Classifier's ability to effectively rank and highlight the most insightful reviews in the Amazon marketplace.

4.4.1. XGBoost (Extreme Gradient Boosting)

XGBoost(Extreme Gradient Boosting) is a tree-based ensemble machine learning technique for supervised learning, and can be applied to both the regression and classification tasks. In this project, helpfulness was predicted using XGBoost, which distinguished between reviews that were helpful and those that weren't. XGBoost is based on an ensemble modeling technique called boosting, which involves feeding the output of one decision tree into the next, resulting in weak learners becoming stronger over a series of rounds. As seen in the (15), the goal function for XGBoost is the product of a loss function and a regularization term.

$$Obj = \sum_{i=1}^{n} loss(y_i, y_i) + \sum_{k=1}^{K} \Omega(f_k) \qquad (15)$$

where Obj is the objective function to be minimized, n is the number of training instances, $loss(y_i, y_i)$ the loss function which measures the difference between the actual value y_i and predicted value y_i, K is the number of trees, $\Omega(f_k)$is the regularization parameter that penalizes complex models to avoid overfitting. Instead of learning the tree all at once which makes the optimization harder, an additive strategy is applied.

For binary classification, since helpfulness is binary, the logistic loss function can be used:

$$Logistic\ Loss\ (Binary\ classification): loss(y_i, \hat{y}_i) \tag{16}$$

$$= -[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

where the actual value is y_i and predicted value is $\hat{y}_i$ and y_{ij} is the 1 if sample i belongs to class j, 0 otherwise.

Regularization is represented as

$$\Omega(f_k) = \gamma T + \frac{1}{2}\lambda \|w\|_2^2 \tag{17}$$

Where γ is the regularization parameter for controlling the number of leaves in the decision trees, T is the number of leaves in tree k, λ is the L2 regularization parameter and $\|w\|_2^2$ is the L2 norm of the weights in the tree.

Three essential processes are involved in the decision tree ensemble development process which includes calculating the gradient, building the tree, and assembling and updating the learners. The gradient of a loss function with respect to its expected value, gi, and its second derivative (Hessian), for a given loss function, are both represented by the symbol hi. The following formulas are used to calculate the gradient and Hessian for each training instance i in the gradient calculation process.

$$g_i = \frac{\partial L(y_i, \hat{y}_i)}{\partial \hat{y}_i^2} \tag{18}$$

$$h_i = \frac{\partial^2 L(y_i, y_i)}{\partial {y_i}^2} \tag{19}$$

For the Tree construction, given the gradients g_i and Hessians h_i for each training instance, a new tree is fit to the negative gradient residuals. For each leaf j in the tree, the optimal weight w_j is computed by minimizing the following objective function for that leaf.

$$Obj = \frac{1}{2}\sum_{i \in I_j} \left(\frac{g_i}{h_i}\right)^2 + \lambda \|w\| \frac{2}{2} \tag{20}$$

Where I_j is set of indices of training instances assigned to leaf j, λ is the L2 regularization parameter and the optimal weight w_j is obtained by taking the derivative of the objective function with respect to w_j.

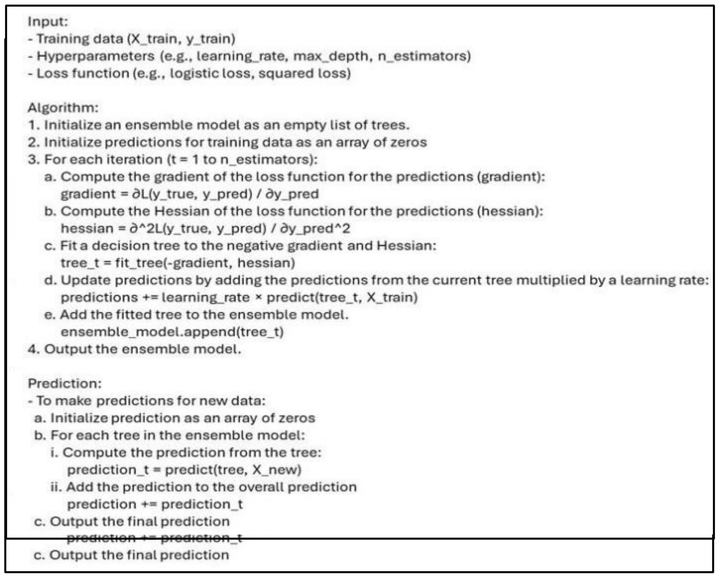
In order to avoid overfitting, each newly constructed tree is then added to the ensemble with a calculated step size (learning rate). The main boosting element of the ensemble's forecast for a given instance is calculated by adding the predictions of each tree in the ensemble.

$$y_i = \sum_{k=1}^{K} f_k(x_i) \tag{21}$$

where y_i is the predicted value for instance i, $f_k(x_i)$ is the prediction of tree k for instance i and K is the total number of trees in the ensemble. The pseudo algorithm for XGBoost is shown in Figure 6 and architecture of XGBoost algorithm is shown in Figure 7.

Figure 6

Algorithm for XGBoost

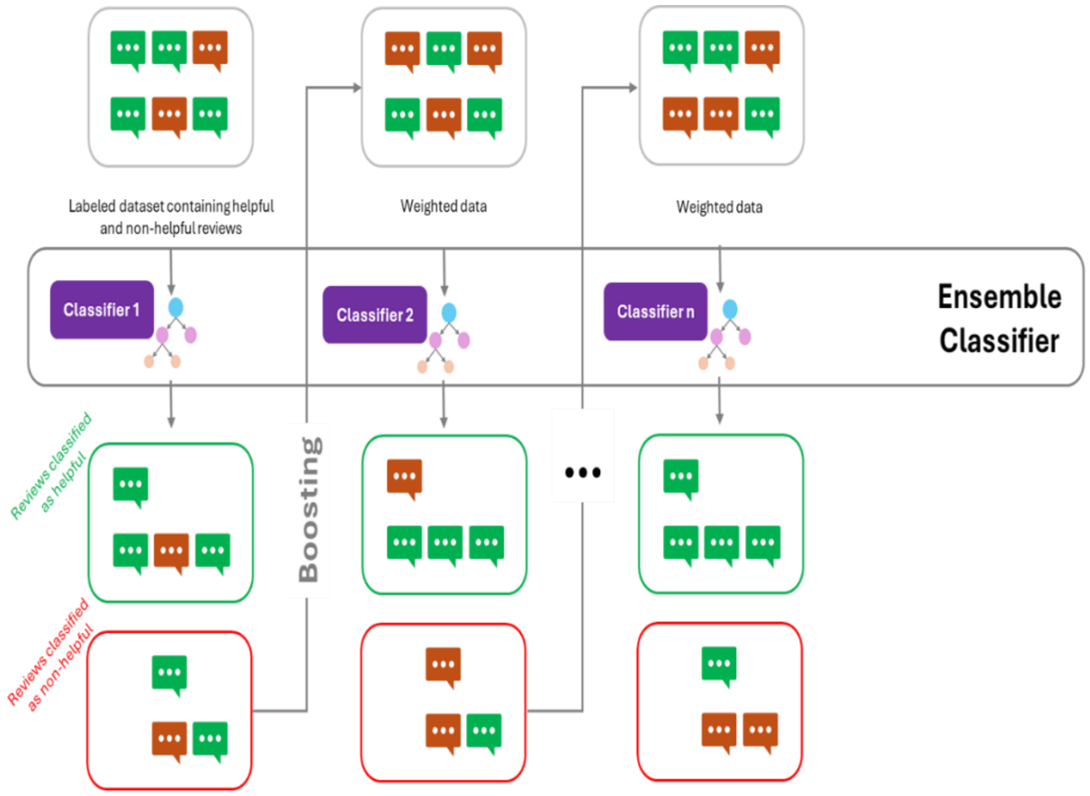


```
Input:
- Training data (X_train, y_train)
- Hyperparameters (e.g., learning_rate, max_depth, n_estimators)
- Loss function (e.g., logistic loss, squared loss)

Algorithm:
1. Initialize an ensemble model as an empty list of trees.
2. Initialize predictions for training data as an array of zeros
3. For each iteration (t = 1 to n_estimators):
   a. Compute the gradient of the loss function for the predictions (gradient):
      gradient = ∂L(y_true, y_pred) / ∂y_pred
   b. Compute the Hessian of the loss function for the predictions (hessian):
      hessian = ∂^2L(y_true, y_pred) / ∂y_pred^2
   c. Fit a decision tree to the negative gradient and Hessian:
      tree_t = fit_tree(-gradient, hessian)
   d. Update predictions by adding the predictions from the current tree multiplied by a learning rate:
      predictions += learning_rate × predict(tree_t, X_train)
   e. Add the fitted tree to the ensemble model.
      ensemble_model.append(tree_t)
4. Output the ensemble model.

Prediction:
- To make predictions for new data:
 a. Initialize prediction as an array of zeros
 b. For each tree in the ensemble model:
   i. Compute the prediction from the tree:
      prediction_t = predict(tree, X_new)
   ii. Add the prediction to the overall prediction
      prediction += prediction_t
 c. Output the final prediction
```

```
      prediction += prediction_t
 c. Output the final prediction
```

Figure 7

The architecture of the XGBoost algorithm for predicting review helpfulness


Labeled dataset containing helpful and non-helpful reviews
Weighted data
Weighted data
Classifier 1
Classifier 2
Classifier n
Ensemble Classifier
Reviews classified as helpful
Reviews classified as non-helpful
Boosting

4.2. Model Supports

4.2.1. *Environment, Platform, and Tools*

For the development and execution of machine learning algorithms, we have employed the Alienware AMD Ryzen 9 7950X 16-core CPU, 4501 MHz high-performance computing (HPC) computers at SJSU campus, Google Collab Pro, and local machines. Data transformation and cleansing were first done on local machines since it was more convenient. After the preprocessing was complete, the whole code, including the modeling, was moved to an HPC system for quicker processing, particularly during the model training and hyperparameter tuning stages. Because our data was so large, it was simple to obtain a stratified sampling for it using powerful computing resources. We have utilized Python as our primary programming language throughout the entire project. Because Jupyter Notebooks allows for an interactive and collaborative development process, all of the coding was done there. We used Collab Pro for some preprocessing, and the HPC lab was used for model execution because we needed more processing capacity. As Table 1 illustrates, we have utilized a variety of different libraries that are tailored for machine learning.

Table 1

Libraries used for review helpfulness

Libraries	Usage
pandas	For data manipulation and analysis
NumPy	For numerical computing
matplotlib.pyplot	Data visualization
random	Generating random numbers
plotly	Interactive plots
seaborn	Data visualization
squarify	Tree map plots
Counter	Counting hashable objects
nltk	Natural language processing
contractions	Expanding contractions in text
sklearn.feature_extraction.text.TfidfVectorizer	Text feature extraction
TextBlob	Sentiment analysis
emoji	Working with emojis
sklearn.model_selection.train_test_split	Splitting data into train, validation, and test sets.
XGBClassifier	XGBoost model
sklearn.metrics.classification_report	Classification Metrics
sklearn.metrics.accuracy_score,	Model evaluation

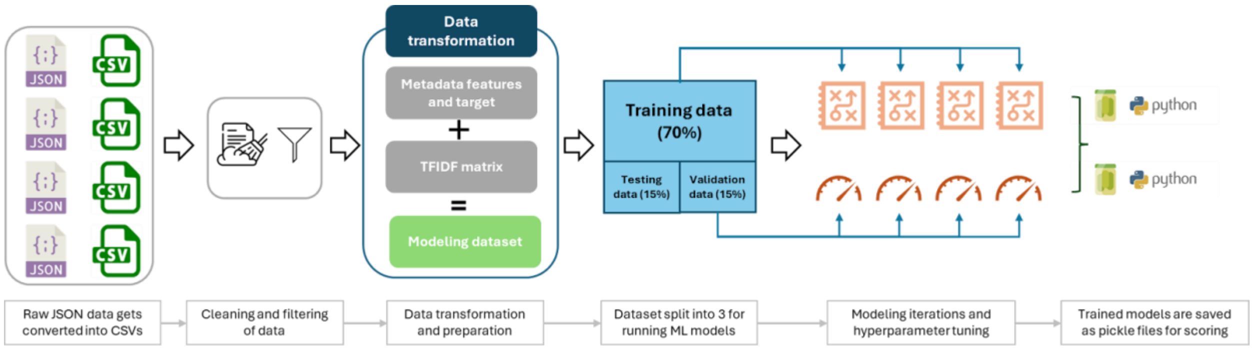
Note. These are the specific libraries used in our project.

4.2.2. Model Architecture and Dataflow

To start, the raw JSON must be converted to CSV, which makes handling and processing data easier. To get rid of redundant and unnecessary data, we undertake data cleaning and filtering in the following phase. We generate target variables (review helpfulness) and metadata characteristics for stratified samples during the transformation process. The relevance of words is determined using the TF IDF approach. Lastly, modeling is done using TFIDF matrices and metadata characteristics. The dataset is then divided into three segments, representing 70%, 15%, and 15% of the total, respectively which are train, test, and validate. We have employed SVM, XG Boost, Random Forest, and Logistic Regression as machine learning methods. In order to identify the best model performance, the model will be trained on training data several times. During each iteration, the hyperparameters of the model will be adjusted using the validation dataset. After the models are trained and fine-tuned, we test them using test data, and they are then saved as pickle files.

Figure 8

Architecture of amazon product review helpfulness


JSON
CSV
Data transformation
Metadata features and target
+
TFIDF matrix
=
Modeling dataset
Training data (70%)
Testing data (15%)
Validation data (15%)
python
python
Raw JSON data gets converted into CSVs
Cleaning and filtering of data
Data transformation and preparation
Dataset split into 3 for running ML models
Modeling iterations and hyperparameter tuning
Trained models are saved as pickle files for scoring

4.3. Model Comparison and Justification

Our project's goal was to develop machine learning models that could categorize and forecast how helpful online product reviews will be for Cellphones and Accessories category. To determine how beneficial a review was, the models took into account metadata, language richness, and text processing strategies and attributes taken from the review text. We have used TF-IDF algorithms to extract features from our cleaned and pre-processed review datasets as part of the data transformation process. Using this method, the most informative phrases are chosen based on their TF-IDF scores. This helps in lowering the feature space's dimensionality, which can enhance the model's performance and computing efficiency. All the described machine learning techniques used these generated TF-IDF feature vectors as input. I have explored the following machine learning models which include Random Forest, XGBoost, SVM, and Logistic Regression for Cellphones and Accessories category.

The main reason logistic regression was selected because it is easy to apply and understand in text categorization applications. It is quite efficient in terms of computing and simple to implement. It is less likely to overfit, and the model coefficients can offer insightful information about the relative significance of various review text elements. It works effectively for jobs involving binary objectives in categorization. Given that our objective variable was binary, it made sense for us to begin our modeling process with this model in order to spot trends in the data right away. Before going further into more complex algorithms, we employed Logistic Regression to quickly establish a baseline.

Our next model to be trained and assessed was the Support Vector Machine (SVM). It was mainly selected because of its capacity to define an ideal border, particularly in high dimensional spaces. High dimensionality is a significant problem in text classification tasks, which makes modeling difficult and resource-intensive. The goal was to find the best possible combination of SVM model kernels and parameters through hyperparameter optimization. Our data had a very high dimensionality after feature extraction using TF-IDF, so SVM was the best option.

Because ensemble learning techniques perform well in identifying complex patterns and correlations in textual data, we looked into them next. They are capable of managing class imbalance, which is typically the case with text review datasets because most reviews are useless. Two ensemble learning techniques, Random Forest and XGBoost, combine weak learners to produce a powerful prediction model. To increase performance overall, Random Forest mixes several decision trees. We decided to use it in our research because of its capacity to capture complex patterns and handle noisy elements with efficiency. A strong gradient boosting approach that works well with textual data is called XGBoost (Extreme Gradient Boosting). An essential component of XGBoost is regularization, such as L1 and L2, which can aid in preventing overfitting. Additionally, it enhances generalization performance, which is advantageous when working with data that has a high dimensionality. A measure of feature importance, which is crucial for determining the most beneficial words or terms related to our categories, is offered by both Random Forest and XGBoost.

These models were selected for our investigation for the reasons listed in Table 2. As they relate to our use case, we have discussed the advantages and disadvantages of each model based on aspects like interpretability, scalability, training time, prediction time, handling of missing data, robustness to outliers, etc.

Table 2

Model Comparison

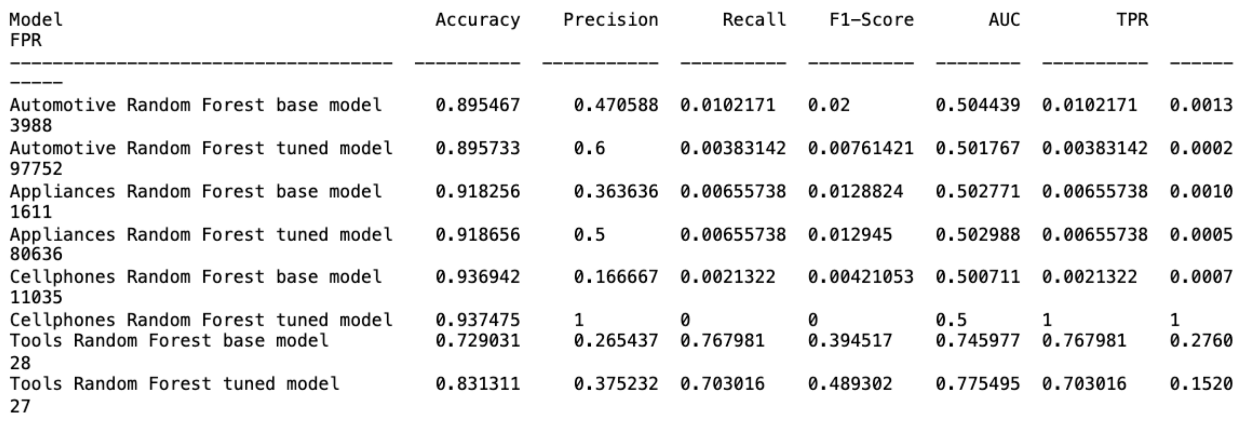
Model	Strengths	Weakness
Logistic Regression	• Simple and relatively interpretable • Beneficial for small datasets • It provides the probabilistic output, can be helpful for scoring the review helpfulness	• Assumes the linear relationship which may not hold true for textual data • Extensive feature engineering requirement • Limited performance to complex problems
Support Vector Machine	• Can model non-linear decision boundaries using Kernel trick • Effective for Binary classification • Ability to handle imbalanced dataset • Robustness to overfitting, even with high dimensionality data	• Increase the computational complexity and training time • Less interpretable than logistic regression • Constraints on memory requirements as data size grows
Random Forest	• Ensemble Learning can lead to better predictive performance • Reduced overfitting • Resistance to noise and outliers • Parallelization and scalability	• Computational complexity • Bias towards corelated features • Sensitive to noisy features
XGBoost	• Includes built in regularization techniques which helps prevent overfitting • Supports parallel processing • Memory efficient • Provides a measure of feature importance	• Sensitive to hyperparameters • Computational complexity • Potential overfitting if not properly tuned

Note. These are the comparisons of all the four models of our project

The Random Forest model's performance in four various categories which includes automotive, appliances, cellphones and accessories, and tools and home improvement that shows varying degrees of efficacy for both the base and tuned versions of the model in terms of metrics like Accuracy, Precision, Recall, F1-Score, AUC, and TPR. Both the base and tuned models in the automobile sector displayed accuracy levels around 0.895, however, the tuned model's precision increased dramatically from 0.470588 to 0.6, although recall stayed low and only marginally improved the F1-score. Appliances saw a moderate increase in the F1-score due to the tuned model's slight increase in accuracy to 0.918656 and improvement in precision to 0.5. This suggests a careful balance in the model's modifications. The tuned model achieved perfect precision but zero recall, resulting in an F1-score of 0 and showing an extremely cautious model approach. The cellphones category had a high accuracy of over 0.936 in both models. With a higher F1-score of 0.489302, the tools category which had the lowest base model accuracy at 0.729031 saw notable improvements in the adjusted model, with accuracy reaching 0.831311 and improved precision and recall balance. These findings illustrate the trade-offs involved in model optimization by showing that while tweaking can increase some metrics, such precision or accuracy, it frequently decreases others, like recall. The unique features of each category have a significant impact on the model's performance, thus it is important to carefully evaluate how various metrics should be ranked and balanced in accordance with the real-world requirements and the type of product reviews being analyzed. Table 3 shows the comparison of all four categories for all the models.

Table 3

Comparison of Random Forest model across all categories

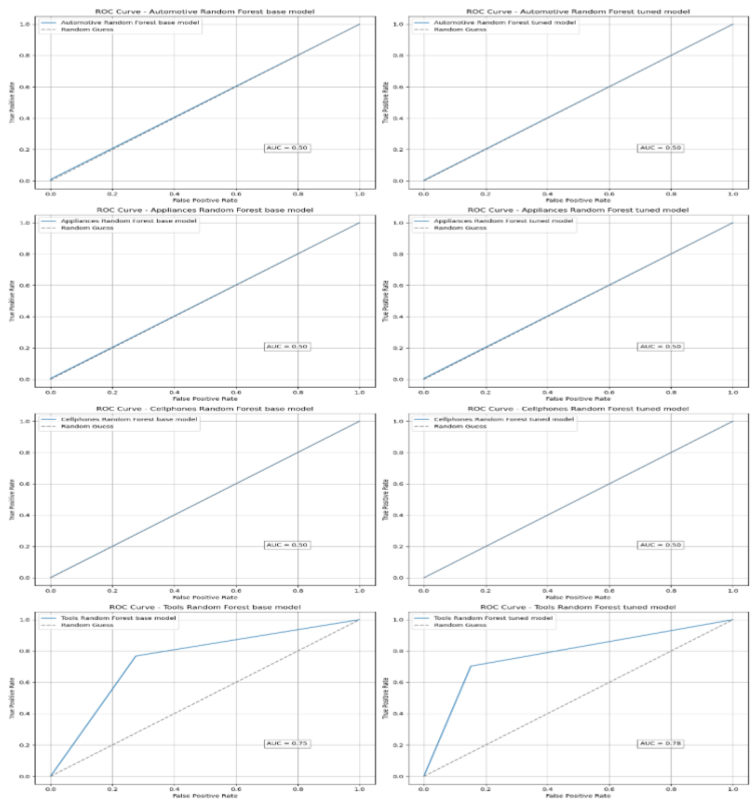


```
Model                                  Accuracy    Precision      Recall    F1-Score       AUC         TPR
FPR
-----------------------------------  ----------  -----------  ----------  ----------  --------  ----------  ------
-----
Automotive Random Forest base model    0.895467     0.470588  0.0102171   0.02        0.504439  0.0102171   0.0013
3988
Automotive Random Forest tuned model   0.895733     0.6       0.00383142  0.00761421  0.501767  0.00383142  0.0002
97752
Appliances Random Forest base model    0.918256     0.363636  0.00655738  0.0128824   0.502771  0.00655738  0.0010
1611
Appliances Random Forest tuned model   0.918656     0.5       0.00655738  0.012945    0.502988  0.00655738  0.0005
80636
Cellphones Random Forest base model    0.936942     0.166667  0.0021322   0.00421053  0.500711  0.0021322   0.0007
11035
Cellphones Random Forest tuned model   0.937475     1         0           0           0.5       1           1
Tools Random Forest base model         0.729031     0.265437  0.767981    0.394517    0.745977  0.767981    0.2760
28
Tools Random Forest tuned model        0.831311     0.375232  0.703016    0.489302    0.775495  0.703016    0.1520
27
```

Figure 9 shows the ROC (Receiver Operating Characteristic) curves for several random forest models applied to a variety of datasets, including automotive, appliances, cellphones, and tools, are displayed in the figure you sent. Each graph displays the performance of both the base and tuned models by plotting the True Positive Rate (TPR) on the y-axis against the False Positive Rate (FPR) on the x-axis. The percentage of genuine positives that are accurately detected is represented by the TPR, whereas the percentage of negatives that are mistakenly classified as positives is represented by the FPR.

Figure 9

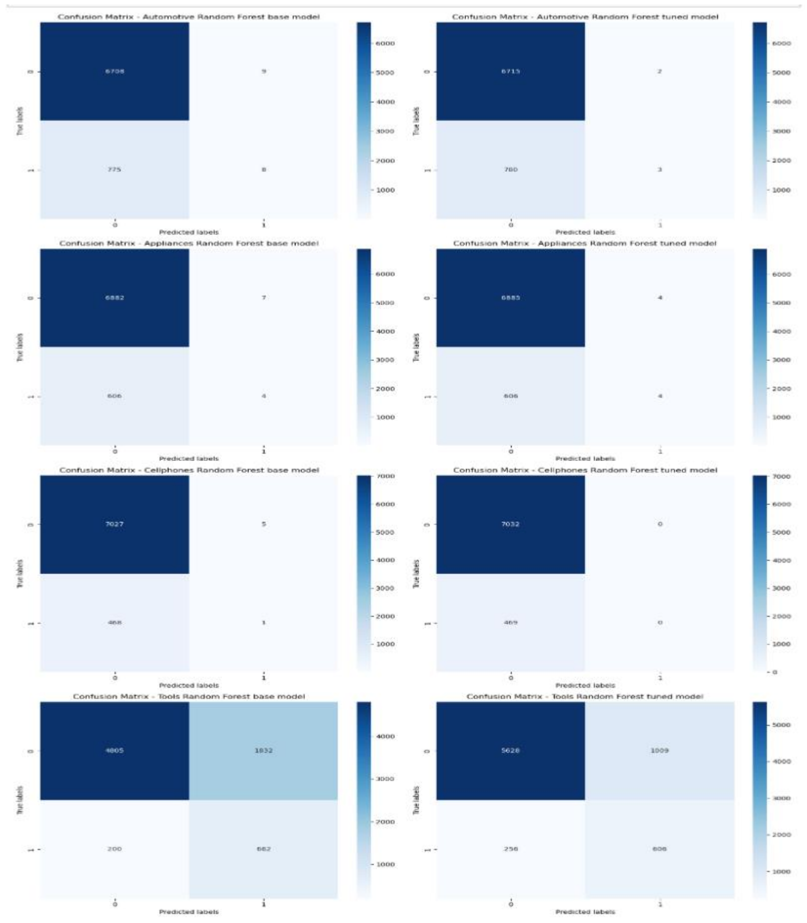
ROC curves of Random Forest for all the four categories



The Figure 10 explains the models' overall ability to distinguish between classes is measured by the AUC (Area Under the Curve), where 1 represents perfect discrimination and 0.5 is no better than random guessing. The models for cellphones and accessories, appliances, and automotives all have poor predicting abilities in these graphs, with AUC values that are very close to 0.550, or guesswork. On the other hand, the Tools models do better, particularly the adjusted model that attains an AUC of 0.778, which is a significant improvement above the base model's AUC of 0.735 and indicates a fair discriminatory power. This implies that fine-tuning parameters can greatly improve model performance, as seen in the Tools dataset in particular.

Figure 10

Confusion matrix for all the four categories



4.4. Model Evaluation Methods

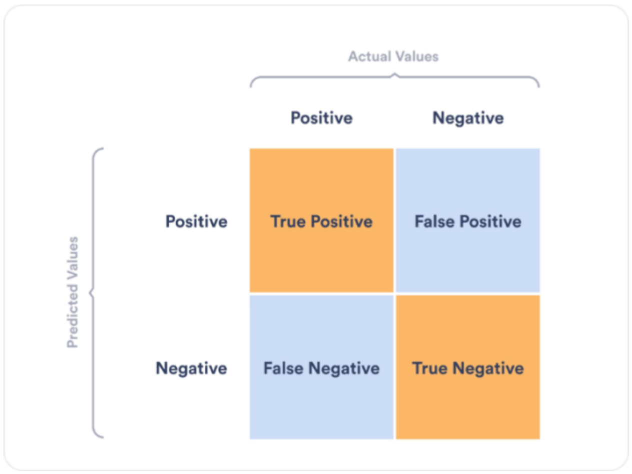
To predict review helpfulness for each of the selected product categories, we have four classification models. We have employed a number of metrics, including F1-score, Accuracy, Precision, and Recall, to assess the models' metrics. The confusion matrix, which shows the actual vs the projected values, can be used to determine these measures. A graph of sensitivity against 1-specificity, known as the Receiver Operating Characteristic (ROC) curve, is also being plotted. The criteria used to forecast the review's helpfulness are described in the subsections below, based on a study report by Hudgins et al. (2023).

4.4.1. *Confusion Matrix*

One sort of representation that may be used to assess the quality of the categorization between the actual class and the expected class is a confusion matrix. Another name for it is the error matrix. The confusion matrix can show which classification is accurate or incorrect for the review. A 2x2 grid is a common confusion matrix, as shown in Figure 11.

Figure 11

Exemplary *Confusion Matrix*



Note. Exemplary Confusion Matrix from "What Is a Confusion Matrix And How to Read It? " by Lory Seraydarian (2022), *Plat ai* (https://plat.ai/blog/confusion-matrix-in-machine-learning/)

The reviews in our project that have been accurately identified are called True Positives (TP). Reviews that are labeled as not helpful but are actually not helpful are called True Negatives (TN). Reviews that are truly beneficial but are mistakenly categorized as false positives (FP) or false negatives (FN) by the model as good reviews are actually helpful but are improperly labeled as false positives (FP). There should be extremely few instances of misclassification or neither FP nor FN in an ideal classification model.

4.4.2. *Accuracy*

The total percentage of reviews that are accurately classified is called accuracy. The calculation is done with (22).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{22}$$

For our project, the total correctness of a model's predictions is determined by how well it predicts overall helpfulness. Even while it may be misleading and does not provide us with an accurate picture of predictions, especially given the imbalance in our data, it is a straightforward statistic that may be used to assess a model's overall performance and facilitate comparison with other models.

4.4.3. *Precision*

The precision of a review is the percentage of all reviews that are predicted as helpful that are true positives, or TP, and that are successfully predicted. The formula for it is (23).

$$Precision = \frac{TP}{TP + FP} \tag{23}$$

Predicting helpfulness is a key component of precision in our approach. The percentage of anticipated helpful reviews that are helpful is what it helps us to understand. Reduced false positive predictions are the result of a highly precise helpfulness prediction model. In order to make things simpler, a very accurate model prevents misclassifying reviews as helpful while correctly identifying helpful reviews.

4.4.4. *Recall*

The percentage of correctly anticipated positive cases, or true positives, relative to all actual positive cases is called recall. The percentage of accurately anticipated beneficial reviews, or true positives, or TP, among all reviews that are genuinely helpful is known as recall, also known as sensitivity. Another name for recall is the True Positive Rate, or TPR. The formula for it is (24).

$$Recall = \frac{TP}{TP + FN} \tag{24}$$

Recall in our project is a metric that indicates how many reviews are properly classified as being helpful. It helps us determine whether a large proportion of reviews are actually helpful that are being captured by the model.

4.4.5. *F1 Score*

The F1-score helps to achieve a balance between accuracy and recall, successfully addressing the trade-off between the two measures. It is computed as the harmonic mean of precision and recall. The formula for it is (25).

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{25}$$

The F1-score gives us a comprehensive assessment of a model's capacity to accurately classify helpful evaluations, which is useful in our project as it serves as a gauge of the model's performance. As a harmonic mean, it assigns equal weight to memory and precision. Since our data is unbalanced and the fraction of helpful reviews is substantially lower than the share of non-helpful reviews, the F1 score is helpful for our project. Another objective statistic that is useful for assessing the effectiveness of various classification models designed to predict helpfulness is the F1 score.

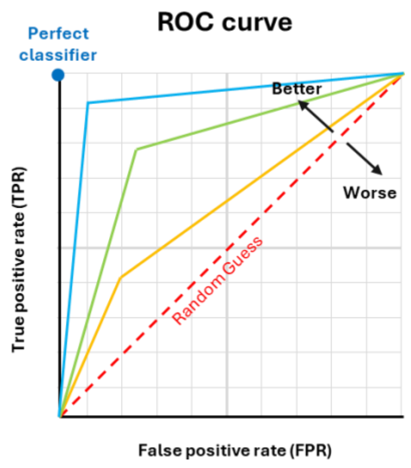
4.4.6. *ROC curve*

Plotting the True Positive Rate (TPR, also known as sensitivity) against the False Positive Rate (FPR, usually computed as 1 - specificity) is known as the Receiver Operating Characteristic (ROC) curve. When forecasting helpfulness, there are two different kinds of ROC curves. The decision threshold is plotted across a range of expected helpfulness probabilities in a ROC curve using probabilities.

A fixed choice threshold (50%) is shown on the ROC curve with binary outcomes to convert projected helpfulness into binary predictions. Because of its simplicity, we are utilizing the ROC curve with binary outcomes in our project. Plotting ROC curves for various models allows one to compare the discriminative capacity of helpfulness prediction. Furthermore, the Area Under the Curve (AUC), which gauges how well reviews are predicted overall, can be computed using the ROC curve. An AUC of one is a perfect classifier that can fully identify between helpful and not helpful reviews; a higher AUC value implies stronger differentiation between helpful and not helpful reviews. With an AUC of 0.5, our model performs no better than a random guess as to whether or not a review is helpful. In Figure 12, a typical ROC curve is shown.

Figure 12

Classic ROC Curve



4.5. Model Validation and Evaluation

Logistic Regression

Base Model. For the category of cellphones and accessories, multiple logistic regression baseline models were trained and evaluated using a 70,15,15 split for training, testing, and validation. Accuracy, precision, recall, f1-score, and classification results with labels for useful (1) and not helpful (0) are included in the output. The accuracy for cell phones and accessories is 93.07%.

Hyperparameter-tuned. We chose to undertake hyperparameter tuning because the baseline models did not perform well. Grid search is used to tune hyperparameters by searching over all potential combinations of the hyperparameter ranges in order to identify the best possible combination. I have adjusted the logistic regression models' two hyperparameters, C and penalty. Since C is the inverse of regularization strength, higher regularization is indicated by lower values, while penalization is indicated by higher values. I utilized C values of 0.01, 0.1, 1, 10, 20, and L1 (Lasso Regression) and L2 (Ridge Regression) regularization for the review helpfulness prediction.

A lower C value suggests that while some overfitting was managed with some assistance from better regularization, useful prediction measures were not improved. Table 4 displays the optimal parameter for cell phones and accessories.

Table 4

Optimal parameters for logistic regression for cellphones and accessories category

Category	Inverse Regularization C	Penalty	Best Parameter
Cellphones and Accessories	[0.01, 0.1, 1, 10, 20]	[l1 and l2]	['C': 0.1, 'penalty': 'l2']

Test data. The testing dataset, comprising 15% of stratified sample data, was used to evaluate the tuned models. We use confusion matrices to analyze the performance of the classification model. There are more genuine negatives than true positives in every category because of the imbalance in the class. Underfitting for the minority class is a result of the appliances, smartphones, and automotive categories' inability to detect helpfulness, which shows a strong bias towards the majority class and is not helpful. But because the model was trained utilizing class weights, the AUC and Recall for the Tools category significantly increased in comparison to the base models. This enhancement came at the expense precision and accuracy. Thus, we were unable to obtain a fully balanced model using logistic regression. The confusion matrix for logistic regression for cellphones and accessories category is shown in Figure 13.

Figure 13

Confusion matrix for cellphones and accessories category for baseline and tuned models for logistic regression

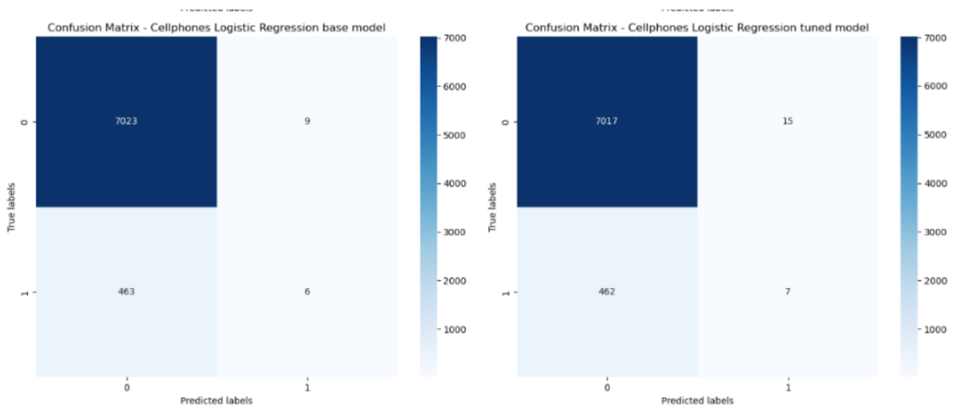
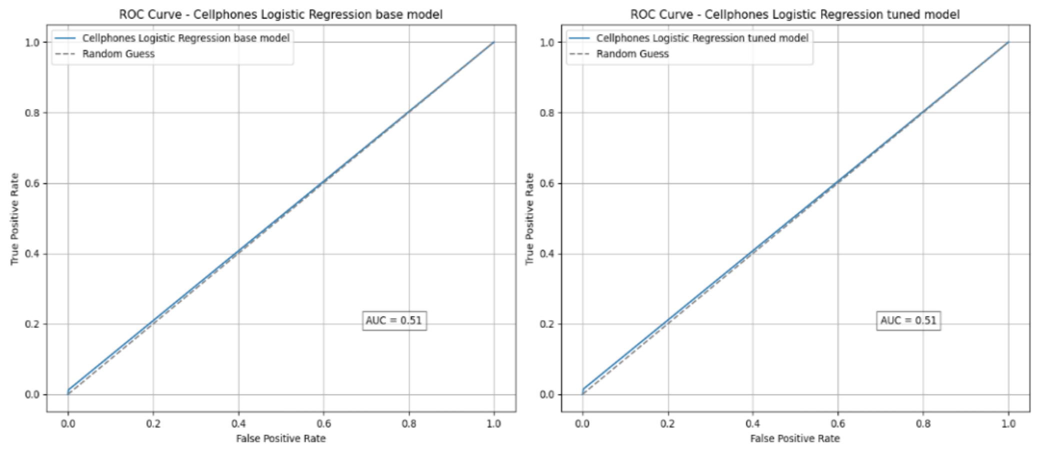


Figure 14, is the ROC curves for true positive rate against false positive rate for logistic regression model for cellphones and accessories category. For cellphones and accessores category the model performance is average where it is able to discriminate between classes for ROC curve.

Figure 14

AUC-ROC for cellphones and accessories category for baseline and tuned models for logistic regression



Support Vector Machine (SVM)

Base Model. To provide a baseline for all of our models, we trained our models using the default parameters. With the parameters "kernel = rbf" for SVM classification, the baseline models were trained using the SVC class for cellphones and accessories category. Table 5 displays the assessment metrics for all the models that were trained on the training set.

High overall accuracy of 93% was obtained by the baseline models trained with SVC class for cellphones and accessories category examined, indicating that it accurately classifies the majority class. But the remaining indicators show how limited its capacity is to discriminate between good and bad examples. The performance of this model is no better than random guessing, with an AUC of 0.5 overall. Furthermore, the recall score of 0.0 indicates that these models are not able to classify the positive cases. The models attain an impeccable precision score of 1.0, signifying that there are no false positives for any particular class. Nevertheless, the recall of 0.0 for that class implies that the models are not producing any true positive predictions. The model's inability to achieve a balance between precision and recalls is further indicated by the F-1 score of 0.0, which also shows that it performs poorly in terms of pinpointing instances. Consequently, despite the great accuracy, it is highly biased in favor of the dominant class and has trouble finding any examples of the minority class.

Table 5

Comparison of evaluation metrics for baseline models for cellphones and accessories category

Category	Accuracy	AUC	Recall	Precision	F1-Score
CellPhones and Accessories	0.937475	0.5	0.00	1.0	0.0

Hyperparameter-tuned. We chose to employ hyperparameter tuning to find the ideal modeling parameters because the baseline models were underperforming. To determine the optimal parameters for the model training, a grid search is carried out on the validation set for the first three categories found in Table 6. "Kernel = linear, rbf, poly" was the search parameter taken into consideration for the SVC class. "CV = 3" was the number of folds for cross validation, and "accuracy" was the scoring setting. "kernel = rbf" was the best parameter that was returned for cellphones and accessories category. The final models were trained using this parameters.

Table 6

List of Parameters used for Hyperparameter Tuning for cellphones and accessories category

Category	SVM Class Used	Parameters for Tuning	Best parameters
Cell Phones and Accessories	SVC	kernel = linear, rbf, poly	kernel = rbf

Note. These are the list of parameters for hyperparameter tuning for cellphones and accessories category.

Test Data. We used the best parameters outlined in Table 6 to train our final model on the test dataset. The evaluation metrics from the final tuned model for cellphones and accessories category are available in the Table 7.

Table 7

Comparison of evaluation metrics for tuned models for cellphones and accessories category

Categories	Accuracy	AUC	Recall	Precision	F-1 score
Cell Phones and Accessories	0.937475	0.5	0.00	1.0	0.0

Note. This is the evaluation metrics for tuned models for SVM

We noticed that with SVC classifier for cellphones and accessories category the tuned models returned the exact same metrics as the baseline models. This was primarily since the tuned parameter was same as the baseline model which was "kernel=rbf". SVM being highly sensitive to hyperparameters did not perform well with either of the approaches. In conclusion SVM cannot be considered an ideal algorithm for this problem. The metric comparisons are available in the Figure 15 and the confusion matrix comparisons are presented in Figure 16.

Figure 15

Comparison of AUC-ROC across categories for baseline and tuned models for cellphones and accessories category

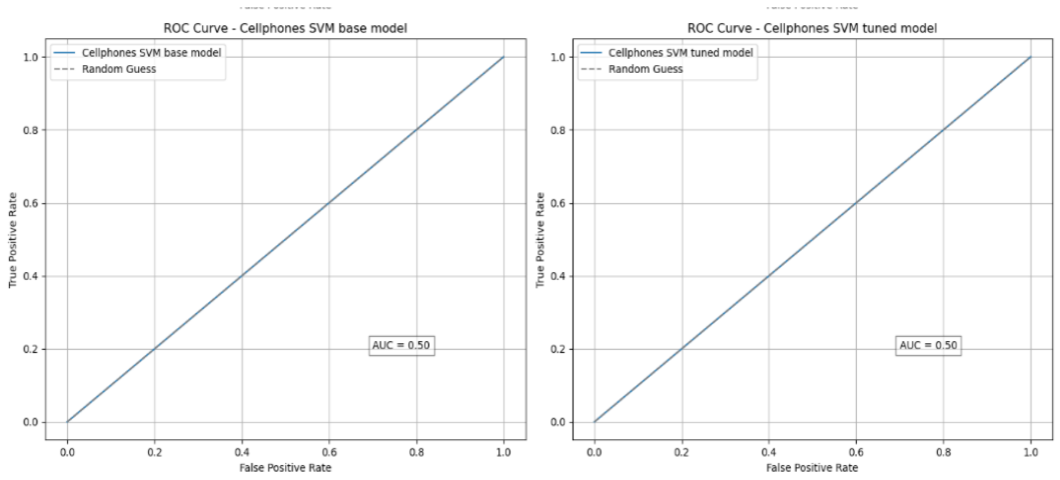
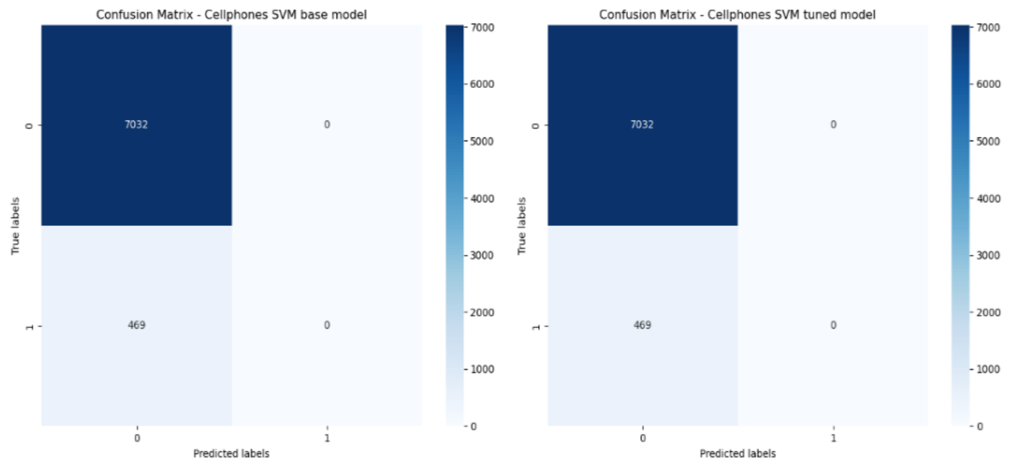


Figure 16

Comparison of confusion matrix across categories for baseline and tuned models



Random Forest

Base Model. Each category's baseline, or base, Random Forest model is trained using default parameters on the training dataset for various categories. There was a slight variation for the “Tools & Home Improvement” category as we used the concept of class weights to handle imbalance in the dataset. This helps the model focus on the minority class instead of being biased towards the majority class.

Hyperparameter-tuned. The validation dataset is used to fine-tune the models using the best possible parameters. The validation dataset was used for tuning since this method lessens overfitting and enhances the model's capacity for generalization. Additionally, since hyperparameter tuning occurs on a smaller dataset than training, which comprises 70% of the dataset, employing the validation dataset, which comprises 15% of the modeling dataset, speeds up the process.

We tested with "n_estimators" and "max_depth" as hyperparameters for the Random Forest models in all categories. In order to estimate how beneficial reviews will be, the "max depth" option sets the maximum depth for each tree in the ensemble. It is a measure of a tree's potential complexity and affects the model's ability to identify patterns in the input data. While a smaller value prevents the model from learning, a higher value or a deeper tree might capture more subtle aspects of the data but also run the risk of overfitting. As a result, determining the ideal value for the "max_depth" parameter is essential. We tried three different values: 3, 5, and 7.

The number of boosting rounds or trees to be constructed in the ensemble is indicated by the "n_estimators" parameter. Since the model would be better able to learn complicated feature interactions if there were more trees in the ensemble, a higher number would ideally improve model performance. However, there is a trade-off, as we have limited computational resources, a higher number would also significantly lengthen the model's training period and computational requirements.

After the value ranges are established, grid search is used to tune the hyperparameters. It searches through all possible combinations of the hyperparameter ranges to identify the best possible combination. We have optimized the training time using 3-fold cross-validation because of our restricted computational resources. Finding the best possible parameter combination to get the highest accuracy is our aim when it comes to hyperparameter tweaking. The Table 8 displays the ideal hyperparameter values for every category run.

Table 8

Optimal hyperparameters for Random Forest for cellphones and accessories category

Category	max_depth	n_estimators
Cellphones and accessories	10	50

Note. All the hyperparameters are calculated with max features as 1000, along with metadata features.

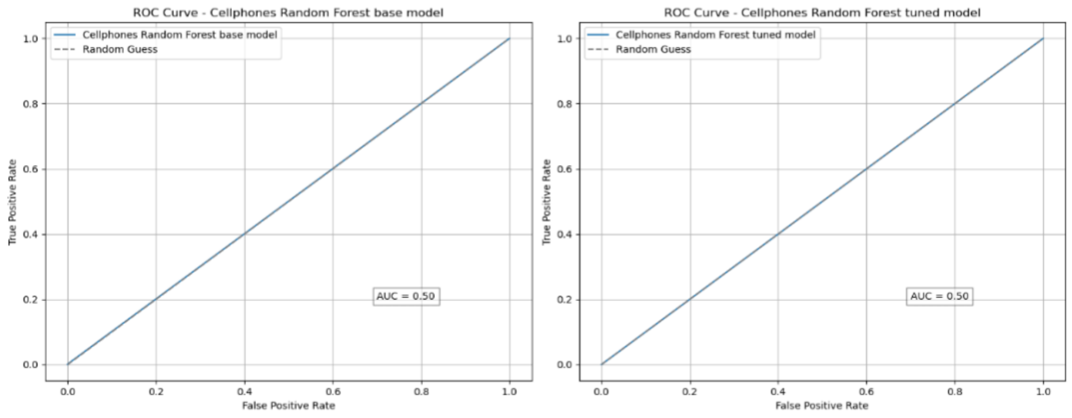
Test data. The testing dataset, comprising 15% of stratified sample data, was used to evaluate the tuned models. By testing the model with data that hasn't been seen before, we can determine how effectively it generalizes and whether it is overfitting the training set. To get baseline model predictions and tuned model predictions, respectively, we employed the baseline and tuned models on the test dataset. This made it possible for us to assess how much the tweaked model's performance improved over the baseline.

However, this result came as a trade-off with accuracy and precision which means that the model identifies some negative results as positive. This indicates that handling class imbalance helps to tune the model, however, there is no significant improvement in the overall performance in review helpfulness classification.

A comparison of Random Forest models ROC (Receiver Operating Characteristic) curves before and after hyperparameter adjustment is shown in Figure 17. The performance of the model is depicted in each set of plots; Notably, tuning did not have any significant impact on models' performance. Though precision and accuracy improved, however, the recall was down to 0 meaning the model is doing random guess and is not able to correctly distinguish between positive and negative instances. Tuning had no major impact on tools category as well, however, with a recall of 0.7, the mode is performing better in distinguishing between positive and negative instances. However, this is with a trade-off on a accuracy and precision.

Figure 17

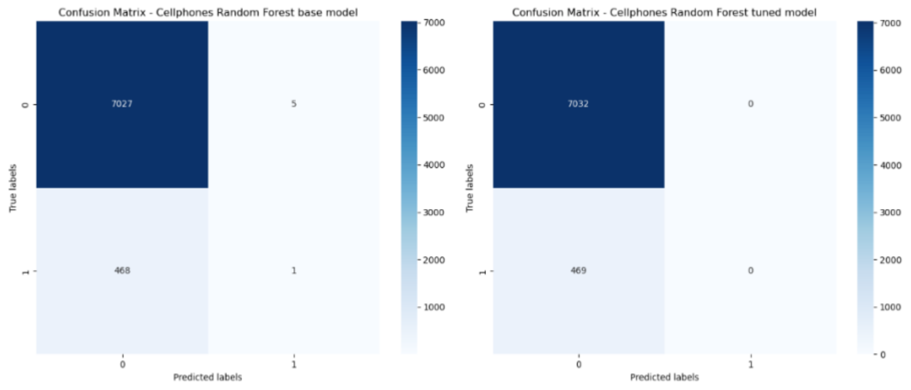
ROC curve for baseline and tuned Random Forest model cellphones and accessories category



A set of confusion matrices for Random Forest models applied to different categories, both before and after hyperparameter adjustment, are shown in Figure 18. For every prediction produced by the models, the number of true positives, true negatives, false positives, and false negatives is shown in each matrix.

Figure 18

Confusion matrix for baseline and tuned Random Forest model for all categories



Comparing the base models and their customized models, the Table 9 shows classification metrics for Random Forest models for cellphones and accessories category. Accuracy, precision, recall, F1-score, and AUC-ROC are among the measurements. Precision is marginally increased for cellphones and accessories category, but tuning has little effect on other metrics, such as the extremely low recall and F1-scores, which show poor performance in accurately predicting the positive class. Significantly, the tuned model for accessories and cellphones exhibits a zero precision, recall, and F1-score, indicating an inability to anticipate positive events upon tuning.

Table 9

Classification metrics for Random Forest model for cellphones and accessories category

Category	Accuracy	Precision	Recall	F1-score	AUC-ROC
Cellphones and Accessories (base model)	93.69	0.1666	0.0021	0.0042	0.5007
Cellphones and Accessories (tuned model)	93.74	0.0000	0.0000	0.0000	0.5000

Note. These are the classification metrics for Random Forest model for cellphones and accessories category

XGBoost

Base Model. The baseline or base XGBoost model for each category is trained with default parameters on the respective category's training dataset. The output includes accuracy, precision, recall, f1-score, and classification outcomes labeled as not helpful and helpful reviews. The accuracy for cellphones and accessories category is 93.60%. All the models were trained on the training set, tested on the test set and Table 10 shows the evaluation metrics.

Hyperparameter-tuned. The validation dataset is used to fine-tune the models using the best possible parameters. The validation dataset was used for tuning since this method lessens the model's tendency to overfit and increases its capacity for generalization. Additionally, since hyperparameter tuning occurs on a smaller dataset than training, which comprises 70% of the dataset, employing the validation dataset, which makes up 15% of the modeling dataset, speeds up the process.

For the XGBoost models in cellphones and accessories category, we experimented with the hyperparameters "learning_rate," "max_depth," and "n_estimators." Since XGBoost is an ensemble of several trees, the contribution of each ensemble member to the final helpfulness prediction is managed by the "learning_rate" parameter. A smaller value would indicate that the model needs to be trained over a longer period of time, which would slow down learning. Although it speeds up learning, a greater value raises the risk of overfitting. As a result, determining the ideal value for the "learning_rate" parameter is essential. We tried three different values: 0.01, 0.1, and 0.2.

In order to estimate how beneficial reviews will be, the "max depth" option sets the maximum depth for each tree in the ensemble. It is a measure of a tree's potential complexity and affects the model's ability to identify patterns in the input data. While a smaller value prevents the model from learning, a higher value or a deeper tree might capture more subtle aspects of the data but also run the risk of overfitting. As a result, determining the ideal value for the "max_depth" parameter is essential. We tried three different values which are 3, 5, and 7.

The number of boosting rounds or trees to be constructed in the ensemble is indicated by the "n_estimators" parameter. Since the model would be better able to learn complicated feature interactions if there were more trees in the ensemble, a higher number would ideally improve model performance. However, there is a trade-off as we have limited computational resources, a higher number would also significantly lengthen the model's training period and computational requirements.

Once the ranges of values are defined, hyperparameter tuning is done using grid search (we used GridSearchCV from sklearn library), which performs a search over all possible combinations of the hyperparameter ranges to find the optimal combination of hyperparameters. Given our limited computational resources, we have used 3-fold cross validation for optimizing the training time. Our purpose in hyperparameter tuning is to determine the optimal combination of parameters that will result in the highest possible accuracy. In the cellphones and accessories category, the learning rate is 0.2 with max depth as 3 and n_estimator as 50. The optimal hyperparameter values for cellphones and accessories category runs are shown in the Table 10.

Table 10

Optimal hyperparameters for XGBoost for cellphones and accessories category

Category	learning_rate	max_depth	n_estimators
CellPhones and Accessories	0.2	3	50

Note: All the hyperparameters are calculated for XGboost model

Test data. The testing dataset, comprising 15% of stratified sample data, was used to evaluate the tuned models. By testing the model with data that hasn't been seen before, we can determine how effectively it generalizes and whether it is overfitting the training set. To get baseline model predictions and tuned model predictions, respectively, we employed the baseline and tuned models on the test dataset. This made it possible for us to assess how much the tweaked models' performance improved over the baseline.

Figure 19

ROC curve for baseline and tuned XGBoost model for cellphones and accessories category

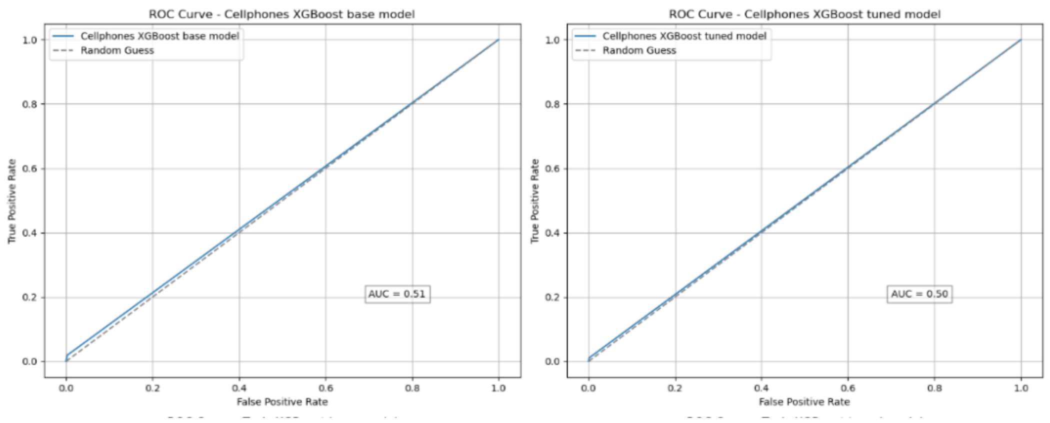


Figure 20

Confusion matrix for baseline and tuned XGBoost model for cellphones and accessories category

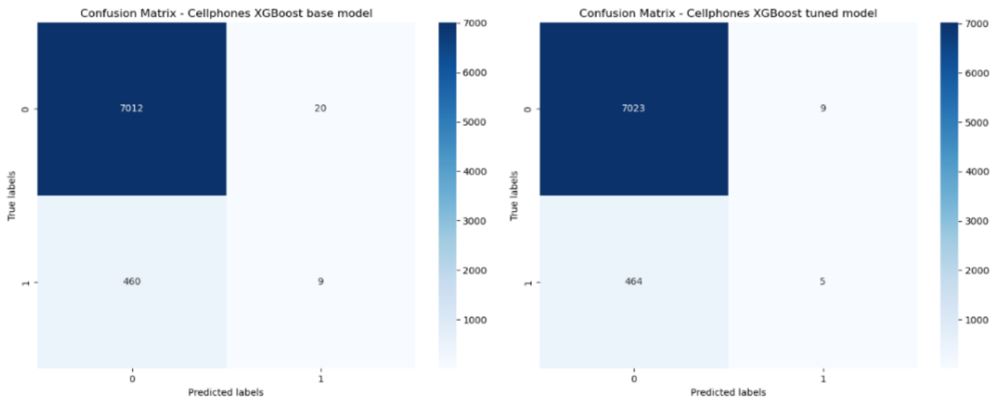


Table 11

Classification metrics for XGBoost cellphones and accessories category

Category	Accuracy	Precision	Recall	F1-Score	ROC AUC
Cellphones and Accessories (Baseline)	93.60	0.3103	0.0191	0.0361	0.5081
Cellphones and Accessories (Tuned)	93.69	0.3571	0.0106	0.0207	0.5046

Note. These values are specific to our project for XGBoost model.

4.5.2. Comparison of Model Evaluation Results

The model performance for each category has been summarized in the Table 12. We have chosen the best-performing models across categories based on accuracy. Even though the accuracies is low for tools and home improvement category, but the AUC and Recall improved using class weights.

Table 12

Summarized accuracy comparison of model evaluation

Algorithm	Cellphones	Model (Base or Tuned)
Logistic Regression	93.70%	Base model
Random Forest	93.74%	Tuned model
SVM	93.75%	Base model
XGBoost	93.69%	Tuned model

Note. These are the comparison of accuracies for all the models of cellphonesand accessories category

For cellphones and accessories, Support Vector Machine(SVM) showed the highest accuracy. We can observe that cellphones and accessories category has different best models indicating that the model performance is a function of the underlying data and there is no clear winner across the board. It is to be noted that accuracy is not the only measure of performance. Some models show higher recall, higher F1-score, or higher Area Under Curve.

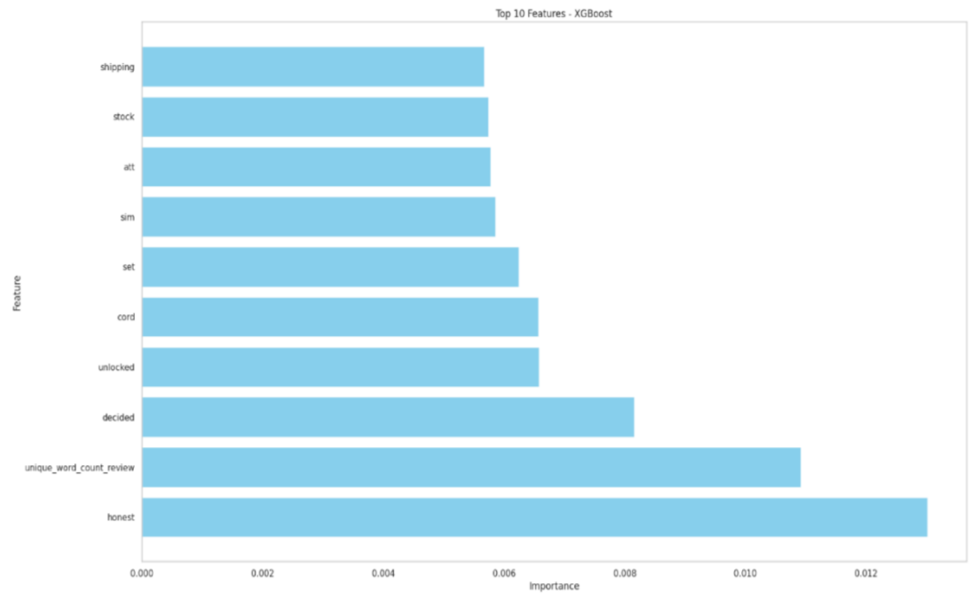
4.5.2 Model Explainability

It is very crucial that we correctly understand the model and provide an explanation for its operation. Explainability of the model is crucial to provide transparency regarding the features the model considers significant in predicting helpfulness, the direction of the feature's influence (positive or negative) in predicting the outcome, and whether the features make intuitive sense in their contribution to the model. Model explainability can be expressed and quantified in a variety of ways, depending on the methodology.

The "feature_importances_" attribute, which determines the "gain" of each feature, can be used to retrieve feature importance straight from the model object in tree-based techniques like Random Forest and XGBoost. When decision trees are being constructed in the ensemble to predict helpfulness, the significance score of each feature is determined by how much it helps to reduce the loss function. The improvement in the model's performance (a decrease in the loss) attributable to each split using the feature serves as a proxy for its contribution. To see the relative importance of each attribute, these values can be sorted and plotted. The cellphones and accessories category's relative feature relevance as determined by the XGBoost model is displayed in Figure 21.

Figure 21

Feature Importance for XGBoost

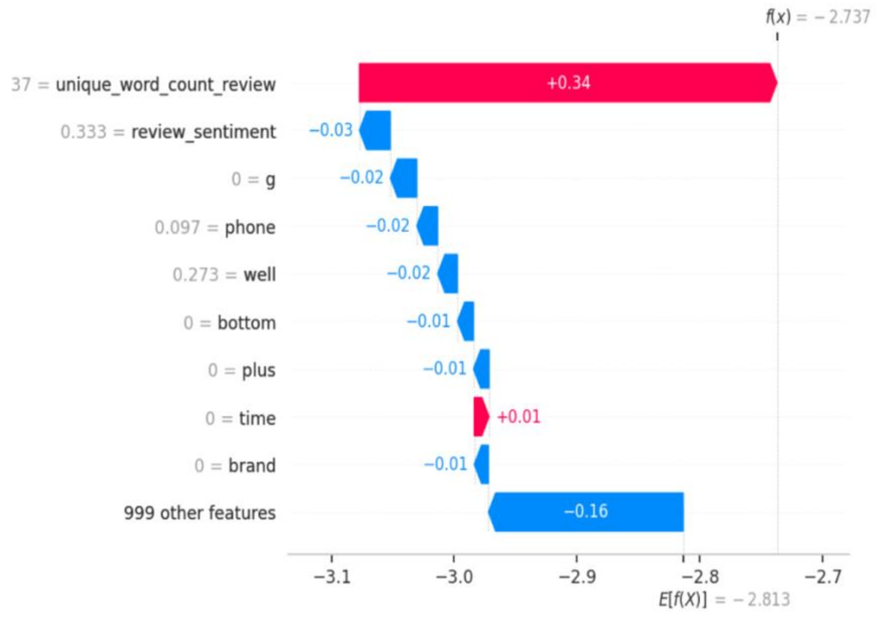


It is evident that the feature "unique_word_count_review" is crucial in determining helpfulness. The drawback of this feature importance method is that, while I can determine the direction of the influence that is, whether the unique word count in the review is positively or negatively affecting helpfulness, I am only able to determine its magnitude, not its direction. This is something that will require further analysis.

Shapley values can also be used to assess a feature's explainability. Values known as SHAP (SHapley Additive exPlanations) can be used to explain machine learning model output. They offer a means of comprehending the part played by every feature in the helpfulness forecast made by the model. Cooperative game theory serves as the foundation for SHAP values, and economics is where the idea of Shapley values originated. Their goal is to equitably allocate each feature's "credit" or "contribution" among all potential feature combinations. Positive SHAP values show that the characteristic helps to improve the helpfulness prediction, whilst negative values show the opposite. Larger magnitudes indicate greater influence. The magnitude of the values represents the impact of the attribute on the forecast. A waterfall chart representing the SHAP values for the XGBoost tuned model for the category of cellphones and accessories is displayed in Figure 22.

Figure 22

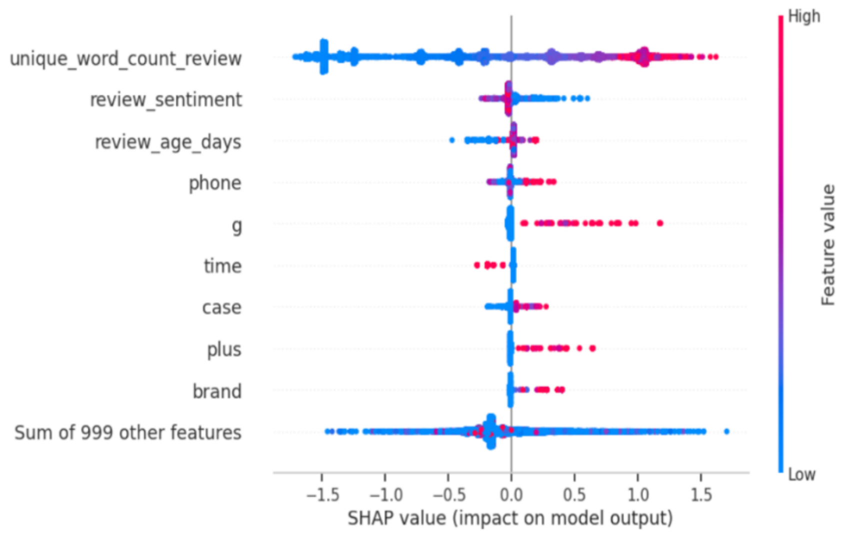
Waterfall chart visualizing the SHAP values



Features that are expected to be more beneficial are indicated in red, and those that are expected to be less helpful are indicated in blue. It is evident now that the unique word count has a negative effect on the prediction; that is, reviews with higher unique word counts tend to be less helpful. In a similar vein, we see that the sentiment expressed in reviews is favorably impacting the model; that is, the more positive the review's sentiment score, the more helpful the review is. A different method of showing the SHAP values for the XGBoost adjusted model for the cellphones and accessories category is the beeswarm chart, as shown in Figure 23, which distributes the data points according to helpfulness.

Figure 23

Beeswarm chart for SHAP values



4.5.3. Conclusion

Due to the unbalanced and highly dimensional nature of the data, models in the Accessories and Cellphones category exhibit low precision and recall but high accuracy. According to the data, ‘not helpful’ is the majority class and ‘helpful’ is the minority class. Given that there are few and exclusive helpful reviews, it is indicative of real-world data. We have several dimensions because we are employing TF-IDF features and metadata. The drawback of using dimensionality reduction methods like PCA is that our model's explainability will be constrained. Unquantifiable characteristics related to human behavior are probably additional qualities that we have not taken into account. We tried running the model through several iterations with a smaller number of TF-IDF features (100), a larger TF-IDF feature matrix (2000), and another iteration with only metadata features. The results did not significantly change between the iterations, suggesting that the features we currently have are insufficient to predict helpfulness. According to our theory, reviews' usefulness varies depending on the individual and how they behave. It could take more research into various NLP approaches to produce highly suitable models for our use case.

4.5.4.Limitations and Future Scope

We have not been able to achieve a balance with our trained models using the usual machine learning approaches. Even though we were able to create models that performed decently, they are not good enough for the real world. Our main focus was on TF-IDF features, but we also looked at certain metadata elements including age, sentiment, and length of reviews. Other methods, such as Named Entity Recognition (NER), Part-of-Speech (POS), Word Embeddings, etc., were not taken into account for this analysis. Due to the large dimensionality of our data and our limited computational capabilities, it was challenging to train the models on traditional computers. The next phase of this research involves investigating more sophisticated feature extraction methods in addition to more potent algorithms such as deep learning methods like CNN (Convolutional Neural Networks), Bi-LTSM, and XLNet, or transformer-based models (e.g., BERT, GPT).

References

Abhilasha Singh Rathor, Amit Agarwal, Preeti Dimri (2018). Comparative Study of Machine Learning Approaches for Amazon Reviews*, Procedia Computer Science,* Volume 132, 2018, Pages 1552-1561, ISSN 1877-0509. https://doi.org/10.1016/j.procs.2018.05.119

B. K. Shah, A. K. Jaiswal, A. Shroff, A. K. Dixit, O. N. Kushwaha and N. K. Shah (2021). Sentiments Detection for Amazon Product Review, *2021 International Conference on Computer Communication and Informatics (ICCCI)*, Coimbatore, India, 2021, pp. 1-6. https://doi.org/10.1109/ICCCI50826.2021.9402414.

Chatterjee, I., Zhou, M., Abusorrah, A., Sedraoui, K., Alabdulwahab (2021). *A Statistics-Based Outlier Detection and Correction Method for Amazon Customer Reviews. Entropy 2021, 23, 1645*. https://doi.org/10.3390/e23121645

Hudgins, Triston, Joseph, Shijo, Yip, Dougla, and Besanson, Gaston (2023). Identifying Features and Predicting Consumer Helpfulness of Product Reviews. *SMU Data Science Review: Vol. 7: No. 1, Article 11.* https://scholar.smu.edu/datasciencereview/vol7/iss1/11

Kapil Kaushik, Rajhans Mishra, Nripendra P. Rana, Yogesh K. Dwivedi (2018). Exploring reviews and review sequences on e-commerce platform: A study of helpful reviews on Amazon.in. *Journal of Retailing and Consumer Services, Volume 45, 2018, Pages 21-32, ISSN 0969-6989.* https://doi.org/10.1016/j.jretconser.2018.08.002

M. I. Hossain, M. Rahman, T. Ahmed and A. Z. M. T. Islam (2021). Forecast the Rating of Online Products from Customer Text Review based on Machine Learning Algorithms. *2021 International Conference on Information and Communication Technology for Sustainable Development (ICICT4SD)*, Dhaka, Bangladesh, 2021, pp. 6-10. https://doi.org/10.1109/ICICT4SD50815.2021.9396822.

T. U. Haque, N. N. Saber and F. M. Shah (2018). Sentiment analysis on large scale Amazon product reviews. *2018 IEEE International Conference on Innovative Research and Development (ICIRD)*, Bangkok, Thailand, 2018, pp. 1-6. https://doi.org/10.1109/ICIRD.2018.8376299.

Appendix

GitHub is used for handling source code developed as part of this project implementation and to maintain the code. GitHub link:

https://github.com/sne2k3/Amazon_product_review_helpfulness/blob/main/Cellphone_and_Accessories_textmining_modeling_Sneha_karri.ipynb

The original dataset is available here: https://cseweb.ucsd.edu/~jmcauley/datasets/amazon_v2/