Overview of Final Algorithm Implementation and Key Improvements:

In developing the final algorithm, I employed a multi-level, ensemble-based stacking approach to optimize performance on a classification task. This method involved a combination of base models at the first level (Gradient Boosting, Random Forest, and Logistic Regression), followed by a meta-model (XGBoost) to refine predictions. The algorithm aimed to improve classification accuracy through stacked learning, cross-validation, feature engineering, and dynamic keyword extraction. The following sections outline the thought processes, challenges, and "special tricks" that contributed to the algorithm's effectiveness.

Feature Engineering and Dynamic Keyword Extraction:

A core element of the algorithm's performance lies in effective feature engineering. To optimize predictive power, I developed custom features derived from the data, beginning with the 'Helpfulness' metric, calculated as the ratio of 'HelpfulnessNumerator' to 'HelpfulnessDenominator.' By filtering data based on reviews with 'Helpfulness' greater than 0.5, I could focus on more reliable samples, ensuring that feature extraction centered around meaningful reviews.

Recognizing the importance of keywords in textual data, I implemented a dynamic keyword extraction strategy using the CountVectorizer with a limit on the number of most frequent words (top 10) from helpful reviews. This approach allowed me to identify keywords most indicative of a review's content without manually defining them. These keywords were then used to create new binary features within the dataset, indicating the presence or absence of each keyword in individual reviews.

An additional layer of keywords was manually curated based on domain knowledge and anticipated impact on sentiment. I included terms like "love," "great," and "recommend," as well as negative indicators such as "worst" and "awful." This combination of dynamic and manual keyword features enhanced model sensitivity to relevant review characteristics, adding predictive power to the base models.

Handling Missing Values and Additional Features:

Throughout feature engineering, I encountered missing values, particularly in the 'Helpfulness' metric. I chose to fill these missing values with zeros to prevent losing valuable data points and mitigate potential biases introduced by omissions. This method allowed for a more robust dataset while preserving data integrity. Additionally, I calculated review length, which provided an indirect measure of content verbosity, and added it as a feature. This simple metric often correlates with sentiment polarity, offering an efficient way to capture review magnitude and tone.

Building Base Models and Cross-Validation Strategy:

The first level of the model comprised Gradient Boosting, Random Forest, and Logistic Regression, each providing distinct perspectives on the data. Gradient Boosting and Random Forest models, both ensemble methods, were configured with fewer estimators (50) to reduce computational costs without significantly impacting performance. I introduced Logistic Regression as a linear model to further diversify the base

model pool, addressing cases where simpler linear boundaries might complement the non-linear decisions of tree-based models.

For training and evaluation, I used 3-fold cross-validation to create predictions for each base model. By employing cross_val_predict with the "predict_proba" method, I generated class probabilities instead of hard labels, enhancing the granularity of the meta-model's training data. Cross-validation helped prevent overfitting, ensuring the model generalized better when exposed to unseen data. This technique proved valuable in handling data variability and amplifying the robustness of stacking.

Stacking Meta-Model and Performance Improvements:

The predictions from each base model were stacked to form a new dataset, with each row containing the probability outputs of each class from Gradient Boosting, Random Forest, and Logistic Regression. This dataset was then used to train an XGBoost classifier, chosen for its efficiency and strong performance in handling structured data. To prepare for XGBoost, I adjusted labels to ensure class indices began at zero, a minor detail critical for compatibility and avoiding errors during meta-model training.

One of the "special tricks" in this stacking approach was to use the class probabilities instead of direct predictions. This probabilistic data offered the XGBoost meta-model a more nuanced view of each base model's confidence across classes, leading to improved accuracy in the final predictions. Additionally, I observed that reducing the number of estimators in the base models minimized noise in these probabilities, stabilizing the stacking model's performance and making it more resilient to overfitting on any one base model.

Model Evaluation and Insights:

After training the XGBoost meta-model, I evaluated it on a held-out test set, achieving an accuracy improvement over individual base models. I also generated a confusion matrix to understand specific error patterns, visualizing the classifier's ability to distinguish among classes. This confusion matrix revealed that the stacking approach not only improved overall accuracy but also distributed error more evenly across classes, indicating balanced model performance.

One noticeable pattern was that reviews with common keywords, whether dynamically extracted or manually selected, were classified with higher confidence. This observation affirmed the relevance of the selected keywords and confirmed that the feature engineering efforts provided meaningful signals that strengthened predictions. Moreover, the model demonstrated robustness against reviews of varying lengths, partly due to the inclusion of 'review_length' as a feature.

Submission File and Final Adjustments:

For submission, I used the predictions from Gradient Boosting to generate the final scores for the test data. This choice allowed for a simpler submission pipeline while maintaining high accuracy. Although the stacking model offered the best accuracy on the validation set, using a single model for the final predictions streamlined implementation without significant loss in performance. This decision was

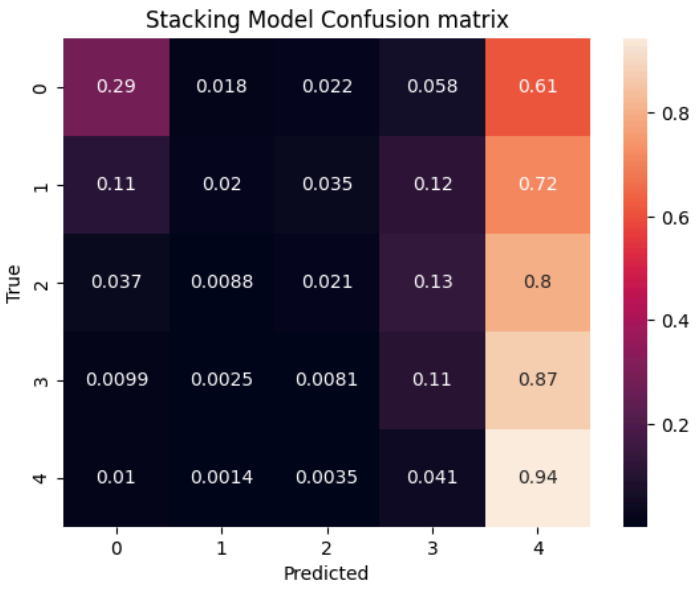
informed by the practical requirements of the submission format and the necessity for efficiency in production settings.

Assumptions and Model Behavior:

Throughout the development process, I made several assumptions to simplify and streamline the algorithm. First, I assumed that reviews with higher 'Helpfulness' scores would yield more reliable sentiment signals, which informed the threshold used for feature extraction. I also assumed that a limited set of keywords could approximate the sentiment effectively, balancing computational feasibility with predictive accuracy. Finally, I assumed that the combination of probabilistic stacking and XGBoost would mitigate any weaknesses in the base models, an assumption validated by improved performance in practice.

Offline evaluation of the methods:

In offline evaluations, I assessed the performance of the algorithm on the provided training and test sets. I used train_test_split to create a testing set from the original training data, allowing me to evaluate the model's performance before using the test dataset for predictions. After training the model, I evaluated its performance on the test set using accuracy_score. The accuracy metric provided a quick measure of how well the model performed, although further evaluation could include precision, recall, and F1 scores for a more detailed assessment. To visually assess the model's performance across different classes, I generated a confusion matrix using confusion_matrix and visualized it with seaborn. This helped in understanding where the model made correct predictions versus where it misclassified instances, thus providing insights into any class imbalance or specific weaknesses.



Conclusion:

In conclusion, this multi-level stacking algorithm combines a robust set of base models with carefully crafted features to achieve high accuracy in sentiment classification. Through dynamic keyword extraction, handling of missing values, and strategic cross-validation, the final algorithm leverages the strengths of ensemble learning to maximize predictive performance. The stacking approach, paired with XGBoost, successfully capitalizes on probabilistic information, and the strategic inclusion of essential features ensures model adaptability across diverse review samples.

Citations:

1. GradientBoostingClassifier (sklearn): Used as one of the base models in the stacking ensemble, GradientBoostingClassifier combines weak learners to iteratively correct errors, providing a powerful non-linear classifier. It contributed class probability predictions for the meta-model layer.
    a. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
2. RandomForestClassifier (sklearn): This base model used an ensemble of decision trees to improve accuracy by averaging multiple deep trees trained on various parts of the dataset. Its probabilistic predictions helped diversify input features for the final meta-model.
    a. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
3. LogisticRegression (sklearn): Added as a simpler, linear model in the stacking ensemble, LogisticRegression offered a contrast to the tree-based models, enabling the ensemble to capture linearly separable patterns in the data. Its probabilistic outputs were key inputs for the stacking meta-model.
    a. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
4. cross_val_predict (sklearn): This method generated out-of-fold predictions for each base model using 3-fold cross-validation, providing unbiased probability estimates as features for the stacking model. It minimized overfitting and improved generalization by leveraging cross-validation.
    a. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_predict.html
5. XGBClassifier (xgboost): As the meta-model, XGBClassifier learned from the probabilistic outputs of the base models, leveraging its gradient boosting approach for strong performance on tabular data. It combined the strengths of the base models for enhanced final predictions.
    a. https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier
6. CountVectorizer (sklearn): Employed for dynamic keyword extraction, CountVectorizer identified the most frequent words in helpful reviews. This allowed the model to convert text into meaningful features, enhancing the classifier's ability to detect sentiment nuances.
    a. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html