Initial Explorations and Observations

1. Basic Data Exploration

Key observations from initial exploration were:

- **Score Imbalance**: Higher ratings (4 and 5) dominated, potentially biasing predictions toward these classes.
- **Text Length Variation**: Positive reviews tended to be longer, suggesting that detailed feedback correlated with higher scores.
- **Time Patterns**: Reviews posted at various times could reflect different sentiment trends over time, which could impact model predictions.

2. Initial Model Attempts and Observations

Using basic metadata features, I tested **K-Nearest Neighbors**[1] as discussed in lecture and **Random Forest**[2,3], a model that I found in a research paper in regards to predicting rating on yelp reviews[2]. Despite reasonable accuracy, these models struggled to capture nuanced sentiment patterns due to a lack of text-based features. This prompted further analysis on textual data. From a paper by Biyani [4] and a TA's help, I decided to utilize **TF-IDF vectors**[5] and **sentiment-related features** to improve model depth and accuracy.

Feature Engineering Process

1. Core Features and Their Impact

- **Helpfulness Ratio**: Calculated as `HelpfulnessNumerator / HelpfulnessDenominator`, this ratio provided insight into user-perceived helpfulness. Higher ratings often correlated with higher helpfulness, aiding the model in distinguishing positive reviews.
- **Review Age**: Derived from the `Time` feature, this variable captured the age of each review in days. Observing that newer reviews showed more polarized sentiments, I retained this feature to provide context.

2. Text-Derived Features

To capture sentiment nuances, I engineered several features from the text:

- **TF-IDF Components**[5]: Vectorizing the text with TF-IDF (3000 features) and reducing it via **Truncated SVD**[6] to 100 components captured essential sentiment while maintaining manageable runtime.
- **Sentiment Keyword Counts**: Positive and negative words (e.g., "excellent," "poor") were counted, creating **Polarity Ratios** (positive-to-negative word ratio) and **Review Length** features. These features were especially valuable for distinguishing mid-range scores, as they captured subtle variations in sentiment.

This feature engineering approach provided a comprehensive representation of both the numeric and textual characteristics of each review, improving the model's ability to accurately capture sentiment.

Model Development and Optimization Attempts

Attempt 1: Simple Models with Core Features

Using **Random Forest**[3] and **K-Nearest Neighbors**[1] on core metadata features such as `Helpfulness`, `Review Age`, and `Review Length` provided moderate accuracy but lacked the depth to capture sentiment nuances.

- **Observation**: Basic models struggled without text features, confirming the need for richer feature engineering and more sophisticated models.

Attempt 2: Text Features with Dimensionality Reduction

After adding TF-IDF vectors[5] and reducing dimensionality with SVD[6] to 100 components, I explored boosting methods further. Inspired by both lectures and relevant literature on boosting models[7], I selected **XGBoost**[8] and **Gradient Boosting**[9].

Observation: Text features markedly improved sentiment detection, although data preprocessing and training time increased substantially, especially with XGBoost. Dimensionality reduction with SVD helped tame complexity and computation time.

Attempt 3: Stacking Models for Enhanced Prediction

I wanted to find a way that could capture the expressiveness of multiple models, which is what led me to research ensemble methods[11] to increase accuracy. To this end, I researched and implemented Stacking **Ensemble**[12] methods with **XGBoost**[8]**, Random Forest**[3]**, SVC**[10]**, and Gradient Boosting**[9] as base models, with **Logistic Regression**[13] as the meta-learner.

- **Observation**: The stacking ensemble achieved higher accuracy, particularly on minority classes. However, the training time was significantly longer.

Final Attempt: Optimizing the Ensemble Model

Balancing accuracy and runtime, I applied several optimization techniques in the final model:

- **Balanced Downsampling**: The dataset was downsampled to 5,000 records to retain class diversity while reducing training time.
- **Narrowed Parameter Ranges**: RandomizedSearchCV[14] was used to explore the hyperparameter space. This was initialized based on prior experiments, which would reduce search complexity and tuning time.
- **Parallel Processing**: As someone mentioned in the discord chat, using `n_jobs=-1` in Random Forest and XGBoost accelerated model fitting by utilizing the full processing power of my computer, especially during hyperparameter tuning.

Final Model Configuration and Rationale

The final algorithm is a **Stacking Ensemble**[12] with four complementary base models:

- **XGBoost**[8]: Effective at capturing complex relationships, particularly with polarity ratios and core metadata.
- **Random Forest**[3]: Reliable with high-variance features, such as polarity ratios and word counts.
- **SVC**[10]: Good at handling linear separation for middle scores, adding stability to the ensemble.
- **Gradient Boosting**[9]: A lighter alternative to XGBoost, capturing subtler feature interactions in the dataset.

The **Logistic Regression**[14] meta-learner balanced predictions from each model, minimizing overfitting and ensuring stable ensemble behavior. **Stratified Cross-Validation**[15] was applied to maintain balanced accuracy across classes.

Performance Optimization Strategies

- **Balanced Downsampling**: Reducing data to 5,000 rows sped up runtime while maintaining balanced class representation.
- **Feature Scaling and Dimensionality Reduction**: Standard scaling across all features stabilized model performance, while SVD-reduced TF-IDF components provided sentiment information without overwhelming the classifiers.
- **Refined Grid Search Parameters**: Narrowed parameter ranges in RandomizedSearchCV limited search complexity, optimizing both runtime and accuracy.

Observations and Key Insights

1. **Importance of Feature Scaling**: Unscaled features destabilized SVC and tree-based models. Standard scaling was essential to ensure that features such as polarity ratio and review length contributed proportionally.
2. **Sentiment Polarity and Review Length**: Longer reviews often had higher scores, especially when combined with a high polarity ratio. This observation guided the inclusion of **Review Length** and **Polarity Ratio** to capture sentiment more effectively.
3. **Addressing Class Imbalance with Balanced Sampling**: Class imbalance biased predictions toward higher scores. Downsampling helped balance accuracy across all classes, improving recall for lower-frequency classes.
4. **Meta-Learner Choice**: Logistic Regression as the meta-learner minimized overconfidence from individual models, ensuring that the ensemble's predictions were well-balanced and accurate.

Final Evaluation and Results

The final model demonstrated stronger performance across all score classes. **Cross-validation accuracy** indicated high consistency, with marked improvements in minority class recall, as evidenced in the **confusion matrix**. This validated the effectiveness of ensemble stacking combined with balanced sampling and optimized feature engineering.

Confusion Matrix Analysis

The confusion matrix showed fewer misclassifications in minority classes due to the inclusion of sentiment-based features and balanced sampling. This highlights the value of combining both structured and text-derived features for capturing sentiment nuances accurately.

Citations:

1)
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
2)
Parsons 2015
https://rstudio-pubs-static.s3.amazonaws.com/127992_a060e7d374d549998df02fc11ac8c334.html
3)
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
4)
Akshat Biyani 2023
https://careerfoundry.com/en/blog/data-analytics/sentiment-analysis/#:~:text=Sentiment%20analysis%2C%20also%20known%20as%20opinion%20mining%2C%20is,with%20the%20help%20of%3A%20natural%20language%20processing%20%28NLP%29
5)
https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
6)
https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html
7)
https://www.geeksforgeeks.org/boosting-in-machine-learning-boosting-and-adaboost/
8)
https://xgboost.readthedocs.io/en/stable/python/python_intro.html
9)
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
10)
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
11)
Khadka
https://dataaspirant.com/stacking-technique/#:~:text=Stacking%20is%20one%20of%20the%20most%20commonly%20used,more%20accurate%20and%20robust%20model%20by%20combining%20them.
12)
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingClassifier.html
13)
https://www.geeksforgeeks.org/understanding-logistic-regression/
14)
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
15)
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html