Dana Yim
CS506
Professor Galletti

Midterm Writeup

Table of Contents:

1) Thought Process and Hypothesis Made to Get to the Final Algorithm

**side note: I learned that asking the right questions are crucial. I tried to focus on why I made the following hypothesis while working on improving my algorithm.*

To study the fields of train.csv file, I asked myself the following questions:

- For each field, what value types are used especially in the Text and Summary columns? In what format are the reviews written?
    - This thinking process was necessary because it made sense to me that in order to implement new features to make the starter code better, I had to have a clear idea on their characteristics. The columns used one or more sentences describing the user's experience with movies.
- Out of the Summary column and Text column, which one would provide more valuable information?
    - My hypothesis = Text column would provide more valuable information since it has more word data that are tied to the Score. Therefore, if I write an algorithm that selects the words from the Summary column and use it as features, it will improve the prediction accuracy.
        - In case the algorithm does not select the keywords that I think are helpful, I also added some **keywords** manually.
- Is using all the rows in the train.csv file efficient?
    - My hypothesis = No, because it will only increase time complexity and space complexity. Since there are rows in which the helpfulness is equal to zero, we can remove them (insignificant data). My hypothesis is that filtering the rows where the helpfulness is greater than 0.5 would significantly decrease the run time of my algorithm and improve the accuracy.
- What other features can I add?
    - I decided to add **review length** as a feature, reasoning that longer reviews may contain more detailed sentiments, which could correlate with **Score**.
- What models should I use?
    - To decide on the models to use, it was important to understand the nature of the dataset. Given the structure of the data (text-based features and numerical scores), I hypothesized that ensemble models like **Gradient Boosting** and **Random Forest** would handle the complexity of the features well. In addition, **Logistic Regression** was included as a simpler model that could serve as a baseline. Finally, **XGBoost** was

chosen as the meta model for stacking due to its reputation for handling structured data efficiently.

2) Final Algorithm Description

Data Loading and Preprocessing: The dataset is first loaded from Google Drive using pandas. The train.csv file contains the reviews, scores, and helpfulness information, while the test.csv file contains the data for which predictions need to be made. The preprocessing begins with handling missing data, particularly within the Helpfulness column. Helpfulness is calculated by dividing the HelpfulnessNumerator by the HelpfulnessDenominator, with any missing values replaced by zero. This step ensures the dataset is complete and ready for feature extraction.

Feature Engineering – Adding Custom Features: The function add_features_to was designed to dynamically extract important features from the dataset, focusing on the Text column, which I hypothesized would provide the most valuable information related to the star ratings.

First, I filtered the dataset to only include rows where Helpfulness was greater than 0.5. This decision was based on the assumption that more helpful reviews are more likely to contain valuable insights that can correlate with the review score. I used CountVectorizer to extract the top 10 most frequent keywords from these helpful reviews. These keywords are then added as binary features to the dataset, indicating whether a review contains each keyword.

For added robustness, I manually selected keywords like 'love', 'great', 'recommend', and 'worst' based on their potential to convey sentiment that impacts the review score. These keywords were also added as binary features, further refining the feature space. Additionally, I created a review_length feature, which measures the number of words in each review, under the assumption that longer reviews may convey more detailed sentiments.

Data Saving and Reuse: To avoid redundant computation in subsequent runs, I saved the processed training and testing datasets to Google Drive as X_train.csv and X_submission.csv. These files contain the newly engineered features. If these files already exist, the code directly loads them, reducing runtime by skipping the feature extraction process.

Train-Test Splitting: After preprocessing, I split the dataset into training and testing sets using train_test_split. The training set is used to fit the models, while the test set is used for evaluation. A 75-25% split was used to ensure sufficient data for both training and testing.

Base Models *(citations are included in the next section of this writeup)*: The next stage involves training three base models: **Gradient Boosting Classifier**, **Random Forest Classifier**, and **Logistic Regression**. These models were chosen because of their diverse approaches to classification:

Dana Yim
CS506
Professor Galletti

- **Gradient Boosting Classifier** is a powerful boosting model that sequentially improves weak learners by focusing on errors made by previous models. I used 50 estimators for balance between performance and runtime.
- **Random Forest Classifier** is an ensemble of decision trees that reduces overfitting by averaging the predictions of multiple trees. Again, 50 estimators were used to keep the model lightweight yet effective.
- **Logistic Regression** provides a simpler linear model that serves as a baseline for comparison with the more complex ensemble methods.

For each model, I performed **3-fold cross-validation** to obtain out-of-sample predictions, which are used as input to the next stage of the algorithm: stacking.

Stacking Predictions: Stacking is a method of combining multiple models to create a meta-model that can make better predictions by learning from the outputs of the base models. In this algorithm, I used the cross-validated predictions from **Gradient Boosting**, **Random Forest**, and **Logistic Regression** as new features. These features are then passed to the meta-model, which is an **XGBoost Classifier**. XGBoost was chosen due to its ability to handle structured data and its efficiency in learning complex relationships from multiple inputs.

To create the stacked feature set, I took the class probabilities output by each base model for all five possible star ratings (1 to 5). These probabilities were stored as new features (gb_1, gb_2, ..., lr_5), with each base model contributing five probability features. This gives the meta-model a rich set of inputs from which it can learn how to best combine the predictions from the base models.

Meta Model: The meta-model, **XGBoost Classifier**, was trained on the stacked predictions from the base models. It uses gradient boosting, but instead of raw features, it learns from the predictions of the base models, making it more robust and accurate. To align with the model's requirements, I adjusted the labels of the target variable (star ratings) to start from 0 instead of 1.

Evaluation: After training, I evaluated the stacking model by predicting the test set using the trained meta-model. The **accuracy score** was calculated as the primary metric for performance evaluation, and a **confusion matrix** was plotted to visualize the performance across all five star-rating categories. The confusion matrix highlights where the model is making accurate predictions and where it struggles, providing insights for further tuning.

Final Predictions for Submission: For the final submission, I used the **Gradient Boosting Classifier** to predict the star ratings of the test dataset, X_submission. These predictions were saved to a CSV file and submitted as the final result for evaluation.

Model Optimization: While developing this algorithm, I made conscious efforts to minimize the runtime by reducing the number of estimators in the base models and optimizing the selection of features. By filtering the dataset to only include helpful reviews and manually selecting important

Dana Yim
CS506
Professor Galletti
keywords, I was able to reduce unnecessary computations and improve prediction accuracy without compromising performance.

3) Tools Implemented & Citing Sources

- The algorithm employs several machine learning libraries, including:
    - **scikit-learn**: for train-test splitting, cross-validation, and evaluation metrics such as accuracy score and confusion matrix.

      https://scikit-learn.org/stable/

    - **XGBoost**: for implementing the meta model in the stacking classifier, as it is known for its speed and performance with large datasets.

      https://xgboost.readthedocs.io/en/stable/

    - **pandas** and **numpy**: for data manipulation and feature extraction.

      https://pandas.pydata.org/

      https://numpy.org/

    - **matplotlib** and **seaborn**: for visualizing model performance with confusion matrices.

      https://matplotlib.org/

      https://seaborn.pydata.org/

    - **Google Colab & Google Drive**: for data storage and execution of the algorithm in a cloud environment.