Report for CS506 Kaggle Competition

Yujia (Stella) Zhai

1 Data Preprocessing

1.1 Natural Language Processing Features

To enhance the dataset with text-based features, I utilized various natural language processing techniques on the `Summary` and `Text` fields. These features aim to capture sentiment, lexical, and semantic information embedded within the review content.

Sentiment Analysis Using the VADER (Valence Aware Dictionary and sEntiment Reasoner) sentiment analysis tool from the `nltk` library, I calculated the compound sentiment score for both the `Summary` and `Text` fields. The compound score, which represents the overall sentiment polarity, helps capture the sentiment expressed in the review text. VADER is particularly effective for analyzing social media content and short texts, making it well-suited for this review-based dataset.

Positive and Negative Word Counts I applied the `nltk` opinion lexicon to identify and count positive and negative words within the `Summary` and `Text` fields. Specifically, I computed the number of positive and negative words for each field, providing a direct measurement of sentiment at the lexical level. This lexicon-based approach serves as a straightforward method to quantify sentiment-laden language within each review.

Special Character Counts Additionally, I calculated the number of exclamation marks (!) and question marks (?) in the `Summary` and `Text` fields. These punctuation marks often convey emphasis, excitement, or uncertainty, contributing to the overall tone and sentiment of the review. The count of these characters provides further granularity in understanding the sentiment intensity.

TF-IDF Vectorization To capture semantic and contextual information in the text, I used Term Frequency-Inverse Document Frequency (TF-IDF) vectorization from the `scikit-learn` library on both the `Summary` and `Text` fields. For the main review `Text`, I extracted 1,000

TF-IDF features, while for the `Summary`, I generated 200 features. TF-IDF helps to highlight the most distinctive terms across the dataset, giving the model contextual word importance.

Comment Length As a final feature, I computed the length of each review comment by counting the number of characters in the `Text` field. This feature provides an indicator of the verbosity or detail level of each review, which may correlate with review informativeness or sentiment strength.

These natural language processing features serve as critical inputs, helping to transform raw text data into informative, quantifiable measures that are useful for subsequent modeling.

1.2 Other Features

Beyond natural language processing features, several other features were engineered to enhance the dataset:

Helpfulness Ratio I calculated a `Helpfulness` score for each review by dividing the `HelpfulnessNumerator` by the `HelpfulnessDenominator`. This feature provides a measure of perceived helpfulness by other users, with missing values filled with zero.

Target Encoding For both `ProductId` and `UserId`, I used target encoding to compute the average score associated with each ID. Additionally, I included counts of reviews per product and per user, representing activity and popularity.

Time Features Using the review timestamps, I extracted the `Year`, `Month`, `Day`, and `Hour` of each review. These temporal features capture seasonal or time-dependent trends that may impact review behavior.

Normalization Finally, I normalized all numerical features, including helpfulness scores, sentiment scores, and various counts, using `StandardScaler` from `scikit-learn`. This ensures that all features contribute proportionately, which is crucial for model stability and efficiency.

Parallel Processing To speed up the computation of sentiment scores and other text-based features, I utilized parallel processing with the `joblib` library. This approach allowed us to efficiently compute features for large datasets by distributing the workload across multiple CPU cores, significantly reducing processing time. I displayed progress using `tqdm`, ensuring visibility into computation progress. But usually this doesn't work well in Jupyter Notebook. So I recommend putting data preprocessing in a separate python script.

2 Model Selection and Evaluation

For model selection, I employed LightGBM due to its efficiency and high performance in handling large datasets. I used the `LGBMClassifier` with manually tuned hyperparameters. Although Optuna is an effective tool for automated hyperparameter optimization, I opted for manual tuning due to time constraints.

The best-performing configuration included: `n_estimators`=500, `max_depth`=12, `num_leaves`=100, `learning_rate`=0.1, `reg_lambda`=10, `reg_alpha`=10, `subsample`=0.8, and `colsample_bytree`=0.9. This setup yielded optimal results on the validation set.

I achieved 78.38% on training set and 71.01% on test set. I have tried to avoid overfitting by using `reg_lambda`=10, `reg_alpha`=10, `subsample`=0.8, and `colsample_bytree`=0.9. But I have to admit that tree-based models are prone to overfitting, especially when the dataset is complex and high-dimensional.

During model development, I observed an issue of class imbalance in the dataset. However, since the final evaluation metric is `accuracy` rather than `multi_logloss`, I chose not to apply `class_Iight` adjustments, focusing instead on optimizing overall accuracy.