CS506 Midterm Report

1. Introduction:

To begin with, I got the idea from an essay written by Nicholas Chu on the Internet, and I will post the link at the end as the citation part. It let me know how to deal with the numerical data and textual data. Also, give me some choices of model, and I try some of them to select the several models that perform the best accuracy.

2. Adding Feature

To enhance the predictive power of the model, I engineered a range of features from the existing dataset. This step was critical in capturing important aspects of the data, such as review helpfulness, sentiment, temporal patterns, and textual information. Below is a detailed breakdown of the transformations and additional features created.

Numeric Features:

(1) Adding 'Helpfulness' feature

The HelpfulnessNumerator and HelpfulnessDenominator fields were transformed into a helpfulness (HelpfulnessNumerator / HelpfulnessDenominator), capturing the perceived helpfulness of a review.

- Assumption**:** I hypothesized that more helpful reviews could correlate with extreme ratings (either very positive or very negative).
- Special Trick: I handled cases where the denominator was zero by setting the ratio to zero, which was effective in preventing divide-by-zero errors and capturing information from such reviews.

However, in the end I decided to drop this feature, because I found that when I increased the max_feature in the TF-IDF. This may because as the max_feature increases, there will be more duplication information in the features.

(2) Temporal Features

I converted the Time column (initially a Unix timestamp) to a datetime format and extracted year, month, and day of the week as separate features. This allows the model to recognize any seasonal or day-of-week patterns in the reviews. For example, reviews during certain periods may tend to have higher or lower ratings.

But I found that it will decrease the accuracy of the test, so I am not using this in my final prediction. This may also be the same reason as the Helpfulness feature.

Textual Features:

(1) Sentiment Analysis

To capture the overall sentiment of each review, I applied a simple sentiment analysis using the TextBlob library. This provided a sentiment polarity score for each review's text, ranging from -1 (very negative) to +1 (very positive). Reviews with higher sentiment scores are likely to be associated with higher star ratings, while lower scores may indicate dissatisfaction.

- Special Trick: For simplicity and interpretability, I classified sentiment as positive or negative. I added it as a binary feature, which proved to enhance predictive power without increasing model complexity significantly. Also, I found that the complement NB model / MultinomialNB model both can't deal with the negative value in the Sentiment feature, so I increased every value of Sentiment by 1. Then the range of the Sentiment is [0,2].

(2) TF-IDF Transformation on Textual Data

The `Text` and `Summary` columns contain valuable information about the content of each review. I only consider the Text columns to do feature engineering because I think if I do both Summary and Text it will cause a lot of duplicate information. To transform these textual fields into features suitable for modeling, I applied **TF-IDF (Term Frequency-Inverse Document Frequency) vectorization** with bigrams on Text columns, capturing word pairs to enrich context and meaning. **TF-IDF (Term Frequency-Inverse Document Frequency)** is a numerical statistic used to represent the importance of words within a document relative to a collection of documents. **Bigrams** are pairs of consecutive words in a text, used to capture phrases and contextual information beyond single words. After trying many different sizes of max_feature, I limited the number of features to 50000 for `Text` to keep the dataset manageable, avoid overfitting and decrease computation complexity.

- Special Trick: By recognizing common word pairs, bigrams can improve models by preserving phrases like "not good" or "highly recommended," which convey specific sentiment or meaning.

Combining the Features:

- **Assumption:** The sentiment scores complement the textual data, providing an additional signal that pure text analysis might miss.
- hstack is used to combine the sparse matrix of TF-IDF features with the dense matrix of numerical features. hstack stands for "horizontal stack" and allows concatenating matrices side-by-side, so each row will contain both the TF-IDF features and the Sentiment features..

3. Model Selection and Ensemble

To create a robust predictive model, I focused on two traditional machine learning algorithms known for their effectiveness with text and categorical data: Logistic Regression and Complement Naive Bayes. I choose these two models because they do relatively good results in Nicholas' paper. Additionally, these models were selected based on their complementary strengths in handling high-dimensional TF-IDF features and mitigating class imbalance issues, which are often present in text-based classification tasks.

Logistic Regression:

Logistic Regression was chosen primarily due to its suitability for high-dimensional data, such as text transformed through TF-IDF. Key reasons for its selection included:

- Logistic Regression provides coefficient weights for each feature, making it possible to interpret the influence of specific words or features on the predicted star rating.
- I also try many solvers in the Logistic Regression model such as saga and liblinear. Finally, I choose saga because it is optimized for large datasets and can handle both L1 and L2 regularization, which could be useful for sparse TF-IDF features, and it has best performance in accuracy.
- Logistic Regression works well with TF-IDF features, capturing the influence of both word frequency and uniqueness across reviews, which is critical for handling large vocabularies in text data.
- I set the max iteration of Logistic Regression to 300, so the computation time will reduce.

Complement Naive Bayes

Complement Naive Bayes (CNB) was selected as the second model in the ensemble due to its strength in handling imbalanced classes:

- CNB is an extension of the standard Naive Bayes classifier, designed specifically to handle class imbalances by adjusting feature weights for each class. This characteristic helps ensure that minority classes, such as low-frequency star ratings, contribute meaningfully to the overall predictions. For this dataset, I found that almost 80% people give 4 and 5 ratings. In that case, it indicates that a lot of people tend to give a high rating, so the dataset may be imbalanced.
- Naive Bayes models, in general, are popular for text classification due to their simplicity and effectiveness with word frequencies, making them a strong fit for this competition's data.

Ensemble Strategy

To leverage the strengths of both models, I implemented a Voting Classifier with soft voting:

- **Soft Voting**: Soft voting was chosen over hard voting to take advantage of the probabilistic predictions from each model. In soft voting, the classifier computes the average predicted probability for each class across all models, resulting in more nuanced and accurate final predictions.
- **Balancing Strengths**: Logistic Regression and CNB have different strengths – Logistic Regression excels with the TF-IDF-transformed text, while CNB provides an edge with imbalanced class data. By combining them in an ensemble, I aimed to achieve higher accuracy than either model could achieve independently.

Citation

Chu, Nicholas. “Amazon Review Rating Prediction with NLP.” *Medium*, Data Science Lab Spring 2021, 7 May 2021, medium.com/data-science-lab-spring-2021/amazon-review-rating-prediction-with-nlp-28a4acdd4352. Accessed 28 Oct. 2024.