CS506 Midterm Writeup

Data Preprocessing and Feature Engineering

To prepare the data, I focused on handling missing values that could impact the results. For numeric data, I replaced missing entries with zeros, and for text data, I used empty strings to avoid problems during the conversion process.

- Sentiment Analysis: I created a simple score to capture the overall tone of each review by counting the number of positive and negative words. This way, the model could identify if a review leaned towards being positive or negative.

- Length of Review: I calculated the word count for each review and summary since longer reviews often mean more detailed opinions.

- Time-Based Features: I extracted the year and month from each review to see if there were any patterns related to the timing of the review.

- Helpfulness Metrics: I checked how "helpful" each review was by dividing the number of helpful votes by the total votes. This helped gauge how users perceived the usefulness of the reviews.

Text Vectorization with TF-IDF

I needed to turn the text into numbers so that the model could analyze it. For this, I used the TF-IDF technique, which gives more weight to significant words while ignoring common ones like "the" or "and." I set a limit of 5,000 words to keep the model size manageable without losing important information.

- I set a max feature limit of 5,000 to strike a balance between capturing important words and keeping the model size manageable. Common English words like “the” or “and” were ignored to focus more on meaningful terms.

Selecting Numeric Features

I chose a few key numerical features to add context to the text data:

- Helpfulness Numerator & Denominator:asic metrics that show how useful people found the reviews.

- Review and Summary Lengths: The number of words in the review and summary.

- Sentiment Score: An extra feature to measure the tone of each review.

- Year and Month: To see if there were any seasonal trends in the reviews.

These numeric features added context and worked alongside the text features to help our model understand the data better.

Standardizing the Numbers
Since I was using numeric features, I made sure they were all on a similar scale through standardization. This helps the models learn more efficiently and keeps training consistent.

Model Training and Tuning
I first tried using a Random Forest Classifier because it's good with mixed data types like text and numbers. But it turned out to be slow, so I switched to Logistic Regression. It was faster and still gave good results with the text-based features.

- Hyperparameter Tuning: I adjusted the number of iterations to help the Logistic Regression model converge. However, increasing this number didn't improve the accuracy much, so I decided not to go higher.

Assumptions

I assumed the chosen words for sentiment analysis would reflect the overall emotions in the reviews. It might help to add more domain-specific words in the future.

Methods Not Covered in Class

Sentiment Score
For the sentiment score I just did a very simple version of what was explained in this article
https://www.alpha-sense.com/blog/engineering/sentiment-score/
The main differences were that I gave it a list of positive and negative words whereas they would use advanced NLP in order to understand the context of each sentence and assign it a sentiment score

Hstack
I used hstack to combine the numerical and text features into a single matrix. This allowed the model to treat both types of data as one. There's a similar method called FeatureUnion, but I preferred hstack for its simplicity in combining sparse matrices.
https://stackoverflow.com/questions/16473042/numpy-vstack-vs-column-stack/65177565#65177565

Logistic Regression

At first, I considered using a Random Forest Classifier, which works well with complex data. But I ended up choosing Logistic Regression since it's efficient and handles high-dimensional data, like the TF-IDF vectors, quite well. I used the 'saga' solver for handling large-scale datasets. Even after increasing the max iterations to 1,000, I noticed only a slight improvement in accuracy, so I stuck with 200 iterations to save computing time.

https://scikit-learn.org/1.5/modules/generated/sklearn.linear_model.LogisticRegression.html