CS 506 Midterm Report

The objective of this data science competition was to predict the star rating associated with user reviews from Amazon Movie Reviews. First, let's break down what the code implemented means, and then go through my process to identify the patterns, other attempts, etc. I observed and attempted in order to increase the accuracy of my prediction.

We use pandas to load two datasets from CSV files. The trainingSet contains labeled data for training the model, while the testingSet contains the data for which predictions will be made (typically without labels). The shape function is called to print out the dimensions of both datasets, providing insight into how many rows (samples) and columns (features) each dataset contains.

We also have an add features function that does the computing for the Helpfulness ratio by dividing HelpfulnessNumerator by HelpfulnessDenominator. The Time column is converted to a DateTime format, and new features like Review_Year, Review_Month, and Review_Day are extracted from it. Additionally, the Summary and Text columns are cleaned by filling missing values with empty strings. The function calculates the length and word count of both the Summary and Text, creating new features such as Summary_length, Text_length, Summary_word_count, and Text_word_count.

We then preprocess the data for model training. The train_data is created by filtering rows in the trainingSet where the Score is not missing. test_data is created by merging the testingSet with the trainingSet (excluding the Score column) based on the Id column, ensuring that the testing set has the same structure. The Combined_Text column is created by concatenating the Summary and Text columns for both the training and testing datasets. This combined column will later be used for text vectorization.

The combined text data (from both training and testing sets) is vectorized using TfidfVectorizer, which transforms the text into numerical feature vectors. The max_features parameter limits the vocabulary size to 5000, and ngram_range=(1, 2) allows both unigrams (single words) and bigrams (pairs of consecutive words) to be considered. After fitting the vectorizer to the combined text data, the transformed data is split back into tfidf_train and tfidf_test to be used for training and testing, respectively. This was researched using ChatGPT.

We then look at the numeric features, including columns such as Helpfulness, Text_length, and Review_Year. Any missing values in these columns are filled with zeros to prevent errors during model training. The resulting numeric features are stored in X_train_numeric and X_test_numeric for training and testing. Here, the numeric features are converted into sparse matrices using csr_matrix to save memory and computational resources, as sparse matrices store only non-zero values. Then, the TF-IDF features and the numeric features are combined using hstack. This combination results in two matrices, X_train_combined and X_test_combined, which hold all the feature information for both text and numeric data. The target variable (y_train) is extracted from the Score column in the train_data.

The dataset is split into training and validation sets using train_test_split, with 25% of the data reserved for validation. We reduce the dimensionality with TruncatedSVD. The reduced data is used to train a RandomForestClassifier. The classifier is fit on the reduced training set, and predictions are made on the validation set (X_valid_reduced). Finally, the accuracy of the Random Forest model is printed.

The final step trains the Random Forest model on the full training data (X_train_combined) after reducing its dimensionality using SVD. Once the model is trained, predictions are made on the reduced test set (X_test_reduced).

Essentially, the process of my code was to calculate the accuracy of the model for this smaller set, and then go ahead with the full set and produce the submission.csv file.

Also in terms of process, before I landed on the Random Forest Classifier, I had tried regression and Naive Bayes on a smaller dataset as well to test the accuracy score of those. I opted for a smaller dataset to optimize the time I spent evaluating the success of these models because we are working with a very large dataset. I finally noticed that the Random Forest Classifier produced the highest accuracy score. My code no longer shows the code for regression and Naive Bayes, however, I aggregated the accuracy scores of all 3 models on the smaller set of data into a list, and chose the “max.” I noticed it was the Random Forest Classifier and I then worked on making this model the most accurate I could.

The biggest thing that I noticed was when using pandas to load the dataset in the beginning of my code, I also used a matplotlib plot to see the distribution of scores in our dataset. As you can see from the plot, there is a significant amount of “5”s in our data that could skew the learning of the model. This is detrimental to the data because the model will use the most common “class” as the safest prediction. It’ll perform really well for the reviews that are 5s, however, with a larger dataset, it’ll fail more with 1-4 stars which may lead to a lower accuracy score. I attempted to use SMOTE which helps us with over/undersampling, however, this did not improve my accuracy score much. The idea behind this is that if we help the imbalance in the data by oversampling the minority or undersampling the majority, we can get a better prediction. I could not make this work, unfortunately. The danger of this,

Even though there isn’t an explicit handling for this in the code, it is worth noting that because I chose Random Forest Classifier, I am less at risk of overfitting my data because of the way it works. It works with different patterns in the data because each decision tree makes its prediction, and the model leverages all of these decisions.

Looking more into this, Naive Bayes and regression may have smaller accuracy scores because of this. Naive Bayes is less flexible with how you can preprocess the data to account for class imbalances and can tend to be more biased toward the majority. Regression is also a bad choice in terms of imbalance in the data because decision boundaries are prone to be skewed and thresholds are hard to get “correct.” Relationships that are complex are also harder to emulate. Same with KNN - I first tried it as I saw it in the starter code; however, we know that KNN is “sensitive” to noise, and with a large dataset with numerical and text data it would be hard to make good decisions. Increasing the accuracy would also take increasing the number of neighbors to a very large number, which would not work for a huge dataset and the code would lose efficiency. I also wanted to use more features.

Because I also noticed this imbalance, I thought it was also important to “balance” out skews in the *features* part of the data. I wanted to use more features, including numerical (date, for example) but also text data (reviews). It was important to preprocess this data to make it work together. TF-IDF was used to make sure that the data was weighted in a way; words that were very frequently used were “down-weighted” so they wouldn’t skew the model too much. Numerical outliers will not affect the model as well. This is a part of Latent Semantic Analysis (LSA), which we spoke about in class to give more importance

I also used TruncatedSVD for the model beforehand. As we spoke about in class as well, Singular Value Decomposition is used to handle datasets with large dimensionality. I wanted to understand the patterns of the data for the model, without the large dataset reducing the efficiency of the model.

Note: I ran my code again and tried to make sure that within my terminal, there was nothing different between the CSV file I submitted to Kaggle vs the new file I created – I didn’t change anything however it stated there were a few lines different. I’m not sure why this is, as the only lines I changed were my comments to make sure this didn’t happen. I’m not sure why this happened at all - I hope this is okay.

Resources:

CS 506 lectures
https://www.ibm.com/topics/random-forest#:~:text=Random%20forest%20is%20a%20commonly,Decision%20trees
https://scikit-learn.org/stable/
https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/
https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.hstack.html
ChatGPT/AI helped me with my print statements and debugging the implementations of my models (Naive Bayes, Random Forest, etc.).