Gavin Schatz

Midterm Report

Introduction

To predict the star ratings of Amazon movies based on user reviews, I employed a Logistic Classification algorithm to model the likelihood of ratings from 1 to 5. Unfortunately, I faced last-minute issues that prevented me from submitting my output for part 1 of the competition. Nonetheless, I hope my detailed explanation of the algorithm and the steps taken to predict the ratings will be validated as part of the evaluation rubric. I will outline the processes involved in loading, preprocessing, training, testing, and evaluating the model.

Preprocessing

The first step in running the algorithm on Colab was to import necessary libraries, including Random Forest, Lasso, TfidfVectorizer, and PCA. I encountered an initial challenge loading the train and test files, which I resolved by mounting them from Google Drive. This was more efficient than manual uploads, as runtime crashes due to high RAM usage necessitated repeated uploads. I loaded the training and testing datasets, checked their shapes, and addressed class imbalance by splitting the training set into two groups: one with the majority class (Score = 5) and another with the remaining scores. I downsampled the majority class to 50% of its original size and visualized the new score distribution, which still skewed towards higher scores, particularly 5.0 and 4.0. This visualization prompted me to consider whether the disparity reflected customer satisfaction or sampling bias. I printed the first few rows of both datasets to confirm the data structure and ran descriptive statistics on the training set to extract essential

metrics. I created a function called add_features_to(df) to calculate the helpfulness ratio and handled missing values in the Helpfulness column, as well as any empty strings in the Text and Summary columns. I planned to use TfidfVectorizer to convert text into numerical features, ensuring a consistent dataset for model training and evaluation.

Sample + Split and Feature Selection

After adding features to X_train and X_submission, I used train_test_split to separate the training and testing subsets, dropping the 'Score' column and assigning it to Y. I filled missing values in the 'Text' column with empty strings and initialized a TfidfVectorizer to convert textual data into a sparse matrix. I fitted the vectorizer to the training data, transforming the text for modeling. To scale the combined features, I applied StandardScalar() while keeping the data sparse. I included error handling to load existing Lasso model results from a pickle file, retraining the model if loading fails. After fitting the model, I extracted coefficients to identify important features based on non-zero values and saved the results for future analysis. Finally, I applied Principal Component Analysis (PCA) to the Lasso-selected features, initializing PCA with 50 components. After fitting PCA to the scaled training data, I prepared to transform the test set, ensuring consistency in the model's performance on unseen data.

Model Creation

Next, I focused on creating a logistic regression model using cross-validation for performance evaluation. I defined a function, cross_validate_kfold, to split the dataset into training and testing sets using K-fold cross-validation. Within this function, I fitted the logistic regression model to the training data and predicted on the test set, calculating the mean squared error (MSE) for each

fold. After evaluating all folds, I identified the best-performing model based on the lowest MSE and proceeded to train the model using the entire training dataset with PCA features. After fitting the model, I predicted outcomes on the test set and calculated the accuracy score, providing a comprehensive view of the model's effectiveness on both training and unseen data.

Model Evaluation

I then evaluated my logistic regression model on the testing set, calculating and printing the accuracy score, which was approximately 0.525. To visualize the model's performance, I created a confusion matrix that displayed how well the predictions aligned with the actual labels. I used a heatmap to illustrate the confusion matrix, normalizing values to better represent the proportions of correct and incorrect classifications. This matrix highlighted the true positives and false positives for each class, allowing me to identify areas of strong performance and areas needing improvement, guiding my future adjustments to the model.

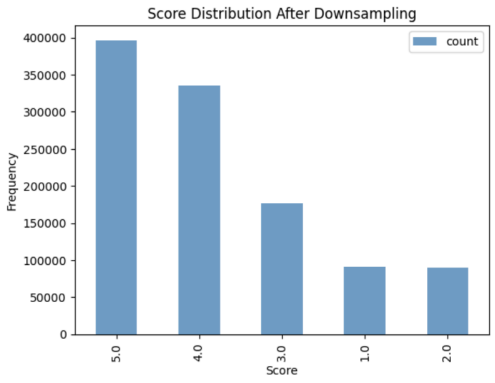
Improvements going forward

In the last code segment, I attempted to implement a strategy to preprocess the submission data, focusing on encoding the 'ProductId' feature. I started by fitting a LabelEncoder to the 'ProductId' column from the training dataset, ensuring it was treated as a string to avoid issues with integer labels. The preprocessing function was designed to check for unseen labels in the submission data and apply the encoder, replacing any unknown labels with a placeholder value. This approach aimed to maintain consistency and integrity in the data before making predictions. Unfortunately, I tried to execute this process but was unsuccessful before time ran out, which hindered my ability to create a submission file. This experience underscored the importance of robust data preprocessing in the modeling workflow.

Figures

1. Score Distribution

```
train.csv shape is  (1697533, 9)
test.csv shape is  (212192, 2)
Score distribution after downsampling:
Score
5.0    396581
4.0    335228
3.0    176082
1.0     91190
2.0     89678
Name: count, dtype: int64
```



2. CV Kfold Output

```
Fold 1 MSE: 1.0967
Fold 2 MSE: 1.0839
Fold 3 MSE: 1.0886
Fold 4 MSE: 1.0863
Fold 5 MSE: 1.0867
Best Mean Squared Error across folds: 1.0839
Logistic Regression best MSE: 1.0838874805589234
Test set accuracy: 0.5251552224549029
```

3. Heat map of accuracy

```
Accuracy on testing set =  0.5251552224549029
```