MIDTERM

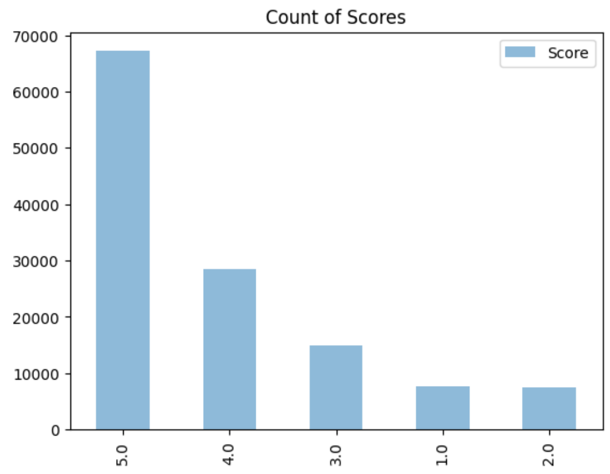
CS 506 - Data Science Tools and Applications
Revathi Vipinachandran

Exploration

The dataset provided is divided into two parts:

- Training Dataset: train.csv of shape `(139753, 9)` which is used to train the model
- Testing Dataset: test.csv of shape `(13976, 2)` which is used to test the accuracy model

The analysis performed on the dataset in starter code has helped me gain useful insights into the dataset. For example:



Based on this visualization we know that the data is skewed as there are way more scores with 5 values as compared to other scores. This can cause the model to be biased which is a problem that needs to be handled for certain ML models. Likewise there are several meaningful visualizations generated in the started_code.ipynb file. (Due to space constraints I am refraining from explaining each visualization).

Feature Extraction:

Topic Generation:
Since we do not have columns that correspond to genres in our dataset, I initially generated 5 topics based on all the data in the text and summary column of the dataset and generated a belongingness score for each record in the Text column with respect to each of the topics. I had used an Latent Dirichlet Allocation model (LDA) for this purpose. An LDA model is a probabilistic model that uses statistical inference to estimate the distribution of topics in a document. But as this method did not provide the required improvement in accuracy, I removed those topics from the list of features.

Sentiment Analysis:
In order to understand and quantize the tone of text in ‘Text’ and ‘Summary’ columns I used sentiment analysis to identify whether a review contains a negative, positive or neutral stance. I used a Text Blob for this purpose. TextBlob is a Python library for processing textual data. The internal architecture of TextBlob for sentiment analysis is based on a Naive Bayes classifier, which is a probabilistic machine learning algorithm. It works by calculating the conditional probability of each feature (word) given the sentiment class and the prior probability of each sentiment class, and then combines them using Bayes' theorem to make a prediction.

Specifically, TextBlob uses a pre-trained classifier that has been trained on a large corpus of movie reviews with both positive and negative labels. During the training phase, the classifier extracts features from the training data, which are then used to estimate the probabilities for each sentiment class. When TextBlob is used for sentiment analysis on new text, it first tokenizes the text into words and then extracts the same features used during training. The classifier then uses these features to calculate the conditional probability of each sentiment class, and the class with the highest probability is assigned to the text.

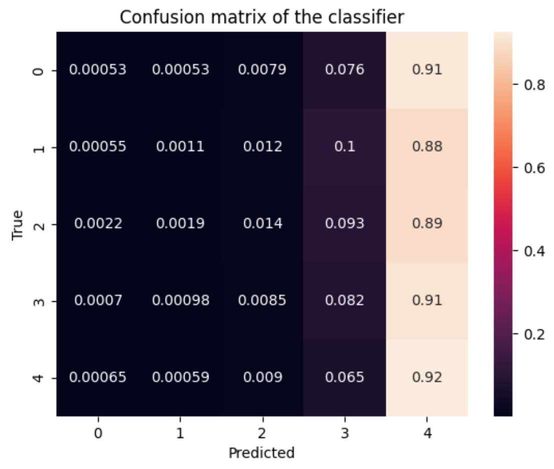
Combined_Review:
This is a column/feature I added to combine the text in ‘Text’ and ‘Summary’ columns. I believe combining the text values in both columns would provide better insight into the sentiments of the product review.

TF-IDF Vectorizer:
I used the TF-IDF Vectorizer to convert the raw text data in the dataframe into a matrix of TF-IDF features. It is a widely used statistical method to evaluate the importance of a word in a document. It tokenizes the text data, then calculates the term frequency for each token, and multiplies it by the inverse document frequency of each token producing a sparse matrix where each row corresponds to each ‘Text’ column value in our dataframe. This has helped me gain insight into the importance of words/terms in each sentence of the ‘Text’ column with respect to the rest of the document and therefore addition of these features has helped my ML model make a more learned prediction. The RMSE produced upon evaluating my model reduced by 0.09713, thus increasing accuracy by a good margin.

Workflow, Decisions and Models:

I have implemented several ML models as part of identifying the best fit for my set of features and the dataset.

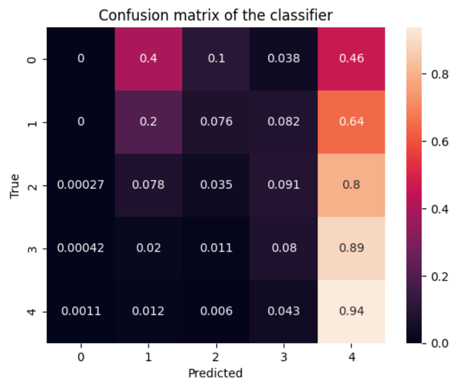
KNeighborsClassifier:
I initially started off with the KNeighborsClassifier Model provided in the starter code. This model and configuration produced an RMSE value of 1.464 and the value did not change much with the addition of new features.



Logistic Regression:
I then worked on a Logistic Regression model for classification. Initially the model didn’t show a stark difference in the accuracy as compared to KNeighborsClassifier although there was improvement. I realized that the issue may be because of the large number of scores with value 5 in the dataset. Therefore

I added ‘class weights’ in order to reduce the bias, and this approach increased accuracy considerably producing the below mentioned results and confusion matrix.

```
Accuracy on testing set =  0.5378279535697249
RMSE on testing set =  1.261978699076022
```



After the above implementation of the logistic regression model, I decided to work on regression models for the same. I realized that regression models could work better given the dataset and objective, as regression models can predict a continuous output value with greater precision as compared to classification models. Regression models can also handle outliers better.

Support Vector Regression:
Support vector machine is one of the powerful regression models. The main idea behind SVR is to find a function that maps the input variables to the output variables, while maximizing the margin between the predicted values and the true values. The margin is defined as the distance between the hyperplane and the closest points from the training data, which are called support vectors. The goal of the algorithm is to find a hyperplane that has the largest margin possible while still accurately predicting the output values. It tries to find the best hyperplane by solving a quadratic optimization problem. When our SVR is trained on our training dataset this high dimensional hyperplane is used to predict the output. The SVR model is trained with a radial basis function (RBF) kernel which is used to train the model on non-linear data. The parameters for the SVR model are set to C=0.7, gamma=0.01, and epsilon=0.01.
In order to find the most suitable C value, I tried implementing the model with different values of C. Below were the RMSE values for various values of C (regularization parameter) with text sentiment and summary sentiment features added, and with TF-IDF Vectorizer added.
When C = 0.5, RMSE =1.055
When C = 0.1, RMSE = 1.165
When C = 0.7, RMSE = 1.0476
When TF-IDF Vectorizer is applied on ‘Text’ column, RMSE = 0.81247

XGBRegressor Model:
Since decision trees are more flexible and can capture non-linear relationships, I decided to use the XGBRegressor model for prediction. An XGBRegressor model is a gradient boosting model that combines multiple decision trees to make predictions. Each decision tree consists of nodes and branches that represent feature splits. The top node of the tree is called the root node, and it represents the entire dataset. The branches of the tree represent different values of a particular feature. The leaves of the tree represent the final decision or prediction made by the model. XGBR works by iteratively adding new trees that correct the errors made by the previous trees. The predictions of each new tree are then added to

the previous tree's predictions to give a new overall prediction. This process is repeated for a specified number of trees or until the error metric reaches a certain threshold. The gradient boosting technique computes the gradients of the objective function with respect to the model's predictions, and then adjusts the model's parameters (i.e., weights and biases) in the direction of the negative gradient. The learning rate controls the step size of the parameter updates, and the maximum depth and subsampling control the complexity of the decision trees.

n_estimators: The number of trees in the ensemble. In this case, it is set to 1000.
learning_rate: This controls the step size at each boosting iteration.
max_depth: This parameter sets the maximum depth of each tree in the ensemble. A deeper tree can capture more complex interactions in the data, but may also overfit. In this case, I set it to 8.
subsample: This is the fraction of the training data that is randomly sampled at each boosting iteration to train the tree. This can help prevent overfitting by introducing randomness into the training process. I have set it to 0.7, which means that 70% of the data is used to train each tree.
colsample_bytree: This is the fraction of the features that are randomly sampled at each node in the tree. This can also help prevent overfitting by introducing randomness into the feature selection process and I have set it to 0.8, which means that 80% of the features are randomly selected at each node.
I decided on the above parameters after experimenting with different values.
Below are the results produced with different sets of features.

Feature Addition and Model	RMSE
XGBRegressor with text_sentiment and summary_sentiment	0.9045
XGBRegressor with text_sentiment, summary_sentiment and TF-IDF Vectorizer applied on ‘Combined_Review’ column	0.78902

Result:

My final code that produced the highest performance is in the file xgbr_tfidf_review.ipynb. With the implementation of the XGBRegressor model, I was able to improve my position from 9 to 5.

Below is the result produced by XGBRegressor with TF-IDF Vectorizer applied on ‘Combined_Review’ column:
RMSE = 0.78902

After multiple iterations of training the model with various parameters and modified features, I was able to achieve an RMSE value of 0.78902 and secure 5th position on the Kaggle Competition.