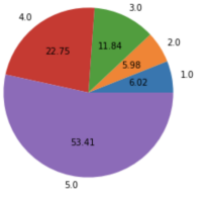
Report

Kaggle Username: JL618

Name: Junyi Li U13632870

Data Preprocessing and Feature Extractions:

At the beginning of each Machine Learning project, it's essential to firstly analyze the structure of the dataframe before processing it in order to gain a better understanding. There are total 139754 entries in the train.csv and 13977 entries in test.csv. However, when running the code data.groupby('Score').size() and visualizing it, they show a significant imbalance of the dataset. The portion of rating=5.0 is larger than other rating in the csv. Also, there are some categorical features in the dataset including the ProductID and UserID.




```
Score
1.0     7361
2.0     7309
3.0    14482
4.0    27818
5.0    65313
dtype: int64
```

```
RangeIndex: 139753 entries, 0 to 139752
Data columns (total 9 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   Id                      139753 non-null  int64
 1   ProductId               139753 non-null  object
 2   UserId                  139753 non-null  object
 3   HelpfulnessNumerator    139753 non-null  int64
 4   HelpfulnessDenominator  139753 non-null  int64
 5   Time                    139753 non-null  int64
 6   Summary                 139752 non-null  object
 7   Text                    139752 non-null  object
 8   Score                   122283 non-null  float64
dtypes: float64(1), int64(4), object(4)
```

```
Id                          0
ProductId                   0
UserId                      0
HelpfulnessNumerator        0
HelpfulnessDenominator      0
Time                        0
Summary                     1
Text                        1
Score                   17470
dtype: int64
```

Here is the detailed pipeline of data preprocessing and extracting features after understanding the dataset:

1. Clean the data, convert Summary and Text to lowercase and remove punctuations and special characters.
2. In order to minimize the influence of the imbalanced data, I decided to downsize the entries of the dateframe with ['Score']=5.0 to 50% of its original amount.
3. Simply fill in NaN values in Summary and Text with an empty string "" to prevent error in further training.
4. Use One-hot(1) Encoding to digitalize all categorical features which are ProductID and UserID, converting categorical variables into a form that could be provided to machine learning algorithms
5. Transform 'HelpfulnessNumerator' and 'HelpfulnessDenominator' into standardized 'Helpful' and 'Unhelpful' features for modeling, while removing the original columns to prevent redundancy with the help of StandardScaler(2).
6. Prepare the text data for modeling by removing list of stopwords with my custom extension and converting the 'Text' and 'Summary' columns into TF-IDF(3) matrices, excluding non-informative characters.
7. Converts the 'Helpful', 'Unhelpful', and 'Time' numerical columns from the DataFrame data into a sparse matrix using spicy, and then stack all processed sparse matrices to their according places.

	Id	ProductId	UserId	Time	Summary	Text	Score	Helpful	Unhelpful
0	195370	1890228583	A3VLX5Z090RQ0V	-1.764415	an unexplained anime review	i was very anxious to see the uncut version of...	2.0	-0.169422	-0.136468
1	1632470	B00BEIYSL4	AUDXDMFM49NGY	1.130690	not great	movie was okaynot great	3.0	-0.235703	-0.136468
2	9771	0767809335	A3LFIA97BUU5IE	-2.128726	technical problem with this dvd	like the dinosaur collectors edition dvd this ...	1.0	-0.036861	4.270759
4	936225	B000B5XOZW	ANM2SCEUL3WL1	-0.736322	herzog the great traveler of both natural and ...	ive always been a great admirer of herzogs oeu...	4.0	-0.169422	-0.274194
6	1213850	B001L8YQ64	A2OFSDJB3MVQ16	-0.107300	disappointing night	saw this film recently on showtime and it was ...	2.0	-0.103142	0.276709

8. Split into Train and Test datasets. Shape of the datasets after splitting:

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

✓ 0.0s

```
((80663, 277393), (8963, 277393), (80663,), (8963,))
```

Model Selection:

Before deciding which genre of models to be used, I need to firstly define the central problem that needs to be address. Since it is a large dataset, and the main goal is to predict the rating based on users' reviews. Overall, it can be either considered as classification problem or regression problem. Therefore, in this case, I choose to use the

ensemble models from scikit-learn library. Ensemble method is a complex model which combines the predictions of several base estimators. I select RandomForest(4) and LightGBM for comparison. After evaluating their performances after first-round training-testing with default parameters, LightGBM stands out from the other by showing the best RMSE score as shown below. LightGBM is a gradient boosting framework that uses tree based learning algorithms(5). According to its documentation, the model LightGBM is optimal for handling categorical features with one-hot encoding, and it's also optimal for sparse features which fits my feature extractions procedures.

```
[100]   valid_0's rmse: 0.85437 valid_0's l2: 0.729948
```

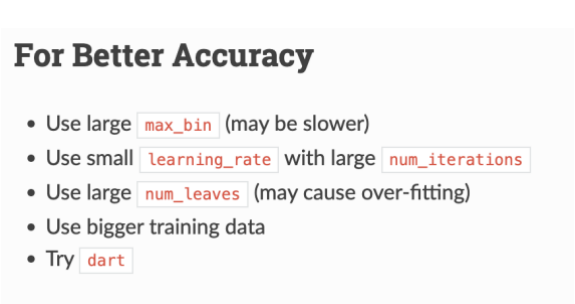
```
Accuracy: 0.5577725079728514
RMSE: 1.3243969616874358
```

```
LGBMRegressor
LGBMRegressor()
```

Tuning Hyperparameters:

I used the most common way which is GridSearch at first but the runtime is too long and crashed due to the limited amount of RAM on my local machine. I think the reason might be there the parameter grid presents too many possible combinations and it requires more time and better computing capabilities. Then I looked up the parameter tuning section from LightGBM official documentation(6). I tried for at least 20 times of different combination of learning_rate and num_iterations with a relatively large num_leaves and finally get the best parameters in the figure shown below. I also incorporate early stopping in training the model to make the training stop until validation score does not improve after 10 rounds in order to reduce runtime. Playing around with parameters is fun but also time-consuming, it takes me about 2 days to finalize the parameter for my model.

```
param_grid = {
    'num_leaves': [100, 200, 300],
    'learning_rate': [0.01, 0.03, 0.1],
    'n_estimators': [100, 300, 500],
    'feature_fraction': [0.8, 0.9],
    'bagging_fraction': [0.7, 0.8],
    'bagging_freq': [5, 8]
}
```




```
best_params = {
    'objective': 'regression',
    'metric': 'rmse',
    'num_leaves': 300,
    'learning_rate': 0.03,
    'n_estimators': 500,
    'feature_fraction': 0.9,
    'bagging_fraction': 0.8,
    'bagging_freq': 5,
    'verbose': -1
}
```

Model Evaluation:

```
Training until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[438]   valid_0's rmse: 0.792914
```

```
LGBMRegressor
LGBMRegressor(bagging_fraction=0.8, bagging_freq=5, feature_fraction=0.9,
              learning_rate=0.03, metric='rmse', n_estimators=500,
              num_leaves=300, objective='regression', verbose=-1)
```

After showing the RMSE calculated from y_test and y_prediction, the final score shows 0.7929 which is a huge improvement from the model in starter_code which gives about 1.4 RMSE score and also a fairly good improvement with default LightGBM model which has a RMSE=0.854.

Challenges and Creativity:

I think there are two main challenges I have encountered during this project. The first one is related to feature extraction. At the beginning stage, I underestimated the importance of feature extraction and assumed how important a complex model might present a really good performance without a proper feature extraction. Therefore, I took too many time focusing on choosing a complex enough model and use multiple ways to tune the hyperparameters. After finding out that all models that are either simple or complex performed similarly with a relatively simple feature extraction procedure, I finally realized the key leading to a better result is through properly and efficiently handling the data.

The other challenge is about tuning parameters after finalizing feature extraction and model selection which I have mentioned above. Running GridSearch is time-consuming and due to the limited RAM of my macbook which is 16gb, sometimes it crashed before completing. Therefore, I have to figure out how to manually select good parameters. To be honest, it even helps me learn lots of useful knowledge with manually tuning parameters. After looking through all pages in the model documentation carefully, now I can capture and understand the relationship between different parameters and how the change of certain parameter would impact the overall performance much better than before.

Another finding after finishing the project is that I think there does not exist an actual boundary to decide user's rating solely based on their review since I believe it's sometimes obvious to distinguish the reviews with rating=1 or rating=5 since they seem to have clearly distinct word choice. However, as for rating within 2,3,4, I think users tend to give a relatively "random" rating with respect to definitely good rating with score=5 and definitely bad rating with score=1. It's kind of hard to generalize their tendency of giving score=2 or 3 or 4. But surprisingly, majority of users chose to give a score=5 to the movies their watched.

I think my creativity in this project is mainly about handling the imbalanced dataset. I used SMOTE technique before but it does not works well for this case with so much text information. Based on this situation with a relatively large majority group, I finally decided to downsize it with certain fraction.

Initially, I want to make my own dict of term frequency with respect to each rating and combine sentiment analysis as well. However, after a simple trial, it does not perform well and even misled the model to predict worse than before. I think the reason is that the code I wrote for building custom dict is relatively simple. I should better use some regression model to handle it.

Reference:

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html(1)

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html(2)

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html(3)

https://scikit-learn.org/stable/modules/ensemble.html (4)

https://github.com/microsoft/LightGBM(5)

https://lightgbm.readthedocs.io/en/latest/Parameters-Tuning.html(6)