CS 506 – Spring 2021 Midterm Competition Report

Zhi Li, mikelili@bu.edu, GitHub ID: milesway

1. Introduction

This competition aims to predict the movie review score based on the information collected on Amazon Movie Reviews. This competition's side goal is to get in touch with Data Science workflow, processing techniques, and decision-making involved during the process.

2. Preliminary Analysis

The preliminary analysis starts overall structural analysis and aims to answer three questions: what's the goal, what do I have, what's the possible way.

2.1 The goal is pretty clear. This competition aims to predict the star rating associated with user reviews from Amazon Movie Reviews using the available features.

2.2 The final result should be a CSV file that contained 30,000 prediction rows; each row has a corresponding review ID and its score. Therefore, the main goal, or the only goal, is to predict the score.

2.3 The data set is relatively complicated. There are three files provided from the handout:

2.3.1 A train.csv contains 1,697,533 unique reviews from Amazon Movie Reviews, with their associated star ratings and metadata. Each review has nine features inside of it, includes

- ProductId - unique identifier for the product
- UserId - unique identifier for the user
- HelpfulnessNumerator - number of users who found the review helpful
- HelpfulnessDenominator - number of users who indicated whether they found the review helpful
- Score - a rating between 1 and 5
- Time - timestamp for the review
- Summary - a brief summary of the review
- Text - a text of the review
- Id - a unique identifier associated with a review

Among all of those features, the most useful are Id, Summary, and the review Text. Some reviews will have the missing entry that may need to handle during the processing. Also, the amount of data provided is huge. With more than 1.6 million entries, an efficiency algorithm must be applied to reasonably control the running time.

2.3.2 A test.csv contains a table with 300,000 unique reviews. The table's format has two columns:

- Id: corresponds to a review in train.csv for which need predict a score
- Score: the star rating scores required to predict.

2.3.3 A sample.csv contains an example submission for the structure of the output.

3. Feature extraction

Feature extraction requires a deep look at the dataset. I realized that in the dataset, the column of 'Score', 'Id', 'Summary', 'Text', are the only object we are focusing on. As for 'ProductId', 'UserId', 'HelpfulnessNumerator', 'HelpfulnessDenominator', 'Time', are something nice to have, but should not be in the mainstream of the model learning process. The 'HelpfulnessNumerator', 'HelpfulnessDenominator', firstly sounds like a good option to go. However, it doesn't have a direct relationship with the score itself, since this can only reflect the persuasive of this review for the general public, after the review is completed and published. The best way of using the feature is to use the Id as the index, apply the model on both summary and text, and predict the score using the model. It should be noticed that the most of score have the score of 5 and there is 300,000 missing entry for the score in the dataset, which counts 18% of the total, while the Summary and Text only has 6 and 62 missing, which are both less than 1%.

4. The working flow

4.1 Import data

The first step of all the work is to import the dataset. For this, the Panda is well-known as the most powerful tool in python. It not only has the intuitive method to access/modify the CSV file, but also can keep its size light, which is critical for such a large dataset I am currently working on.

4.2 Cleaning data

After importing the data, I, first, tokenize words, which splits a large sample of text into words; second, use Regular expression operations to remove non-characters; third, changing all characters to lower case; firth, remove stopwords and lemmatize the remaining words. I do this just on the text column only since applying summary does not improve the score but takes a much longer time to run. After processing, instead of replacing the text column, I decided to add a new column named clean_review to store it as another feature, so that I still have access to the original text if needed.

4.3 Add more feature (*removed*)

For the next step, I decided to add more feature by using a module called TextBlob, which is well known as sentiment analysis. This will apply directly to the clean_review column to calculate the sentiment score, which will be close to 0 if the word is objective. And add this as another feature to be analyzed. However, the sentiment score involves deep learning, which is prohibited according to the competition rule. I finally removed this feature.

I also tried an alternative way by using the SentimentIntensiveAnalyzer powered by NTLK, which also could find the compound sentiment score, and add it as a feature. However, this is also related to deep learning and does not improve the score, so it is removed from the final version.

4.4 Creating checkpoint and test set

The next step is to create the current work checkpoint by storing all data into a local CSV file, so for the next time after changing the method, I can read it directly from the local file, which would be faster than doing the re-processing again.
With the train_test_split provided by the SKlearn module, I split the current data set into the train set and test set with a 3/4 and 1/4 ratio.

4.5 Core technique

I creativity using two approaches to calculate my final score.

4.5.1 The first approach requires three steps. First, I vectorize words using CountVectorizer, which is one of the helpful functions in SKlearn. This will transform the clean_review feature into vectors, stored in an array-shaped matrix to minimize the storage space. However, I choose to set the feature to 8000 since it will provide enough the feature that the classifier can analyze in the next step. I apply this both on the train and test set. Second, I train a Random Forest Classifier with grid search to tune hyperparameters by using the Sklearn toolkit. The best way to think about hyperparameters is like the settings of an algorithm that can be adjusted to optimize performance, just as we might turn the knobs of an AM radio to get a clear signal. I spent days adjusting the hyperparameters and finalize the best one for the current model. Third, I used the model to predict the score.

4.5.2 The second approach is similar to the first one, but instead of passing in the vector, I used the regular data frame to train the model.

4.5.3 The last step is to combine the result. By giving two results different weights, I created 11 different scores for the final model validation.

5. model validation

For the model validation, I first compare the sole score produced by method 1 with the true answer produced by the model. By calculating the mean squared error, I found out that method 1 did not perform that well independently. It gives 1.5775... as the final score in the best scenario. Method 2 performed the same, with the best final score of 1.11248... (the closer to 1, the better). However, the combination of Method 1 and 2 with the weight of 50% on both gives the best result of 0.9982... And became the final submission file.

6. Conclusion and thought

The combination of the two methods gives a surprisingly good result. However, the cost is noticeable. The entire program requires more than 6 hours to complete on a 20 cores machine, while taking all 128G of the system memory almost instantly after vectorized the text. Each run was suffering both for the computer and myself. And such a huge running cost limited me in the existing frame, finding it difficult to jump out of the quagmire and brace for new algorithms such as logistic regression. In the future similar project, I should learn this from this competition, start working from small-scale data, identify the most efficient method, and then focus on that method to avoid wasting time change model.