CS506 Midterm Kaggle Dataset Analysis

Emmanouil Kritharakis
Boston University, Boston, Massachusetts, Kaggle Username: KrithMan

Abstract

The competition data-set contains reviews of different movies and their corresponding rating scores. The goal of this midterm is to analyze the features of each movie such as summary, text review, review helpfulness and so on to predict how a movie will be rated given those features. In this report, I first conduct feature engineering by adding in features based on existing data, then we use techniques like logistic regression and random forest to make predictions.

Preliminary Analysis

The data-set contains a set of features describing the user ids, movie product ids, and their respective reviews of the movie in a score out of 5 containing some review text. It seems that the most influential data column that can affect the score is the reviews that the reviewers make, and their helpfulness factor. Since if an individual were to make a positive review but the helpfulness factor is low, then the movie can still receive a low score. On the contrary, if the user were to review the movie highly and the helpfulness factor is also high, then the movie is expected to be reviewed positively. This means that in feature extraction the significance of the text and summary data columns must be taken into context, in addition to the helpfulness column and both the user IDs and productIDs specifying which users are more likely to make accurate reviews as well as which products are more likely to be reviewed. Furthermore, as we can notice from the figures in the data exploration file both productIDs and userIDs do not appear in the dataset only one time. In particular, there are products as well as users who have been reviewed or reviewed respectively over 2000 times! The repetitive format of products and users attests that both columns play an important role in the review score. Moreover, it should be mentioned from figures in the data exploration file that the mean length of the textual columns (Text & Summary) is between 150-200 words for text and 6 for summary. Hence, we can come to the following conclusion. The combination of them builds a corpus big enough to be able to vectorize it and keep the most frequent and useful words for our model later.

Feature extraction

Before any models are built, we want to extract and emphasize the key aspects of this data-set to produce the best possible data for analysis. To do that, several techniques were used.

TFIDF vectorization of Textual Info (Text & Summary)

In the dataset itself, there are two columns of text data that in some way describes how people reacted to the movie. These text reactions seem crucial to how a movie will be rated, as the more positive people react, the more highly the movie will be rated. This means we want to do some kind of sentiment analysis on the text. Taking the steps with chronological order, first I fill the NaN values of ‘Summary’ and ‘Text’ columns with empty strings. The information of not giving a ‘Summary’ or a ‘Text’ is still an information. Afterwards, I combine them into one column named ‘Textual_info’ since my final goal is to vectorize them

and extract the most frequent and useful words for the combination of them. After creation of ‘Textual_info’, I ran through a few crucial steps to clean the textual data. In particular, I lowercase the text, remove all special characters or numbers, URLs and leave only the words. In succession, with the help of TFIDF vectorizer I removed the stopwords and kept only the words with minimum appearance on the textual data of 2 times. This choice made my corpus concise. Now, it is important to clarify the correct use of TFIDF vectorization. Since we have both train and test data in the same dataframe, it is tempting to use “fit_transform” to the whole column but the final result will be overfitted as the final corpus will be based on the combination of train and test data. To solve this problem, I fitted the TFIDF vectorizer with only the trained columns of ‘Textual_info’ and transformed the TFIDF vectorizer both to train and test ‘Textual_info’ columns. The inspiration of this technique was taken by homework 3 exercise 2(c).

One hot encoding vectorization of UserId & ProductId

In the dataset itself, there are two columns representing the product being reviewed and the user who reviewed it. As described in the preliminary analysis, the same users and products are being utilized over train and test dataset multiple times (over 2000). To take advantage of these two columns, I utilized a technique discussed by instructor Saurav vara prasad chennuri when he replied on homework 2 about one hot encoding the 'Title' column of Titanic dataset. Therefore, I carefully use the one hot encoder from sklearn library for ‘ProductID’ and ‘UserID’. Carefully usage of one hot encoder means that as in TFIDF technique fit should be done in train
dataset only, the same should be done with one hot encoding. Hence, I fitted the one hot encoder in train columns ‘UserID’ and ‘ProductID’ and transformed it to both train and test ones. It is important to mention here that if a userID or a productID is included only on the test set I i have inform the encoder to ignore it (pass all zero values) in order to be working properly otherwise it will raise an error since the encoder do not know how to handle the unknown userID or productID.

Helpfulness

Both ‘HelpfulnessNumerator’ and ‘HelpfulnessDenominator’ columns play an important role in the final score as it is shown in the data exploration file and described in the preliminary analysis. In order to take advantage of both of them I combined them creating the ‘Helpfulness’ column as a fraction of ‘HelpfulnessNumerator’ and ‘HelpfulnessDenominator’ respectively. Making sure that I do not add any NaN values in case of ‘HelpfulnessDenominator’ being zero, I assign 0 to fill NaN values.

Features Explored but not added

At this point it should be mentioned that I explored other features such as ‘Average score per ProductId’ or ‘Average score per UserId’ but their contribution to my model prediction was not that helpful. Furthermore, before trying the TFIDF vectorization I chose a different sentiment analysis tool called VADER, a lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media, and is used to generate predictions on how a viewer reacted to certain movies. The tool generates three separate values, a positive sentiment, a neutral sentiment, and a negative sentiment. Nevertheless, TFIDF vectorization gave me better final accuracy scores so I decided not to keep the Vader analysis.

Analysis Methods

This section aims to go over all of the different methods I have tried. The results of which will be presented in the next section.

Random Forest

Random forests are often used in classification problems because of its ability to avoid over-fitting, accommodate large datasets, and are favored among many traditional machine learning tasks. A random forest consists of multiple decision trees, with each tree answering true or false questions comparing certain features at a threshold. The decision trees then all vote on their respective choices, and the decision with the most votes wins. In my case, due to the large number of generated feature columns I decided to reduce them by a dimensionality reduction technique SVD working in the pipeline with the model. The total number of components chosen were 100. To reduce the concurrent processing time, sklearn speeds up the process by toggling the *n_jobs* flag to −1, signaling that all cores should be used for training. This is possible since each decision tree seems a subset of all the data and is only concerned with a subset of features, which allows for each training session to happen independently of each other.

Logistic Regression

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. We can call a Logistic Regression a Linear Regression model but the Logistic Regression uses a more complex cost function, this cost function can be defined as the 'Sigmoid function' or also known as the 'logistic function' instead of a linear function.The hypothesis of logistic regression tends it to limit the cost function between 0 and 1. Therefore linear functions fail to represent it as it can have a value greater than 1 or less than 0 which is not possible as per the hypothesis of logistic regression. In my case, I tried to increase the maximum number of iterations (*max_iter*) to 500 and which gave me the highest score in Kaggle competition (67.58%) but for reproducibility issues I used the default max iterations of 100.

Model evaluation

To evaluate my model, I used a well known technique named cross validation which we learn during the lectures of CS506. I implement the 'K Fold' method with 10 splits over the train dataset. The interesting and novel part is that I kept the model with the highest accuracy over the 10 splits for my later predictions over the submission data. My inspiration over this idea came from this article:
https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6
It should be mentioned that I used the cross validation method only on the logistic regression model since the random forest required a really long time and it was not easy to reproduce by the graders.

Results

Method	Random Forest	Logistic Regression
Accuracy(%)	58.15%	66.81%

Out of these two methods tried, logistic regression performed better than random forest. This may have happened as a result of logistic regression being able to better fit these features and having a sentiment "threshold" for each text allowed the model to better fit the data entries. As we can see through the confusion matrices of those two methods, even though random forest is capable to recognize better the 5 star reviews which are larger in number in the dataset, logistic regression is able to overall identify all kinds of reviews (1 to 5) especially the lowest and the greatest.