XGBoost Regression for Predicting Movie Ratings

Alvito DiStefano

2024-12-01

Introduction

The goal of this project is to provide the foundation for a movie recommendation system by training a regression algorithm to predict movie ratings in the MovieLens 10M Dataset. This dataset contains just over 10 million ratings provided by over 70,000 unique users via movie recommendation service MovieLens. For each movie in the dataset there is a movie ID, movie title with release year, user ID, rating between 0.5 and 5, timestamp (Unix), and a list of genres. No demographic information is included for users. Acknowledgement for this dataset may be found in the Citations section of this report.

Methodology

EDA and Model Selection

The 10M dataset is split into a training and testing set by course-provided code, edx and final_holdout_test, respectively. Edx contains roughly 9 million rows leaving 1 million for final testing. Due to its large size, a smaller subset of edx entitled edx_small was created to facilitate model selection and validation. The size of edx_small frequently varied depending on the use at hand, but in the final version of the code it sits at one tenth the size of the full edx.

After basic checks of data structure and cleanliness were performed, several common regression models were tested upfront. The data's default columns were passed as model features for this purpose (i.e., no feature engineering). All testing was performed with edx_small for expediency. Simpler algorithms such as decision trees performed predictably poorly relative to more complex options. A random forest model proved the most adept at minimizing RMSE, however execution time suffered greatly when the size of edx_small was increased. Ultimately, gradient boosting, and specifically R's xgboost package, provided the best compromise between runtime and performance. Subsequent feature engineering and model tuning focused on the use of xgboost.

Feature creation was informed in equal part by observations gleaned during EDA and the author's own intuitions about the tendencies of movie ratings and raters. In particular, genres were observed to be polarizing predictors of where a movie's rating would tend to fall. Film-Noir movies in edx garnered the most positive reviews of any genre with an average rating of 4.011. Conversely, movies under the Horror genre had an average rating of only 3.270, perhaps due to their tendency to incite feelings of dread and fear in some viewers (but totally not the author of this report, mind you). This observation informed the direction of feature engineering, and ultimately half of the features used were directly tied to genre information.

Feature Engineering

Movie genres are provided in the MovieLens dataset as pipe-separated values (PSV) in one column, which does not readily lend itself to the manipulations necessary for feature creation. To transform genre data into a more useful format, custom function One_Hot_Genrecoder was created to convert from PSV into one-hot encodings. The function dynamically identifies all unique genres encountered and creates a column for each one, assigning the corresponding binary value to indicate if the genre applies to a given movie. Both training and test data must be passed to this function before genre-related features (User_Genre_Average and Movie_Genre_Average) can be created.

The four features used in the final model are described as follows:

- MovieId_rating_means: A feature capturing the average rating of a particular movie, by movie ID. A custom function Add_Rating_Means creates a new column with average movie ratings in the training data by movie and adds it to both training and test data. If a movie ID is present in the test data but not the training data, an alternate value can be specified (e.g., a default value of 0). Leave-one-out encoding is employed to avoid data leakage which could result in overfitting (i.e., the rating for a given row is excluded from the average for that row).
- User_Genre_Average: A feature capturing user preferences for particular movie genres. Average user ratings by genre are aggregated and extracted from training data using the custom function Extract_User_Genre_Ratings. They can then be added as columns to both training and test data using the Add_User_Genre_Ratings function. As with MovieId_rating_means, if a userId is present in the test data that was not in the training data, an alternate value is specified.
- Movie_Genre_Average: A feature capturing the expected movie rating based on the average ratings of movies of the same genre(s). Similar to how the User_Genre_Average feature was created, custom function Extract_Movie_Genre_Ratings extracts average movie ratings by genre from training data. Then Add_Movie_Genre_Ratings takes this as input to create a feature column in training and test data reflecting the expected movie rating based upon the movie's genre(s).
- Nostalgia_factor: A feature capturing the effect of nostalgia, or the time elapsed between movie release and rating timestamp, on movie ratings. It was observed during EDA that the older the movie is relative to the date a review is given, the higher the rating tends to be on average. Custom function Create_Nostalgia_Factor turns this observation into a feature by taking the difference between the movie release year (provided with the title) and the review year (extracted from the timestamp). This "nostalgia factor" is normalized (0 to 1) and then converted to the average rating within that nostalgia factor bucket across all movies.

Cross-Validation and Model Tuning

A straightforward K-fold cross-validation approach is employed for model validation, feature selection, and parameter tuning. The number of folds used varied at different stages of testing, but in its latest iteration, it is set at 4 folds. Fold partitioning is performed via random selection to ensure homogeneous data representation across all folds. Cross-validation is performed by looping through all folds and reserving all but one for model training. RMSE is calculated for each loop and then averaged to give an overall indication of model performance.

The cross-validation loop is encapsulated within a parent for loop used for feature selection and parameter tuning. The number of loops is defined by the user and can be adjusted to perform either random or grid searches. On each loop, cross-validation is performed with the specified features and model parameters, and the best-performing features and parameters (lowest RMSE) are stored to variables. XGBoost model parameters can be selected at random from a pre-defined sample space for random sampling. Since the number of features used was small, a grid search (15 iterations) was the most sensible approach. Once the parent loop determines the best features and parameters, they are passed directly to the final testing round.

Results

During cross-validation it was found that no subset of the features described above outperformed using all four together, which is not a surprising result given how tailored they are. Cross-validation also provided a best-performing set of model parameters for the XGBoost algorithm:

- eta = 0.1,
- max_depth = 4,
- subsample = .6,
- colsample_bytree = 0.6,
- alpha = 100,
- min_child_weight = 1

The best performing parameters tend to favor a reduction in overfitting, particularly the shallow maximum depth of trees and the high alpha regularization value. Feature sampling of 0.6 was also the lowest value within the sample space, possibly indicating that there may be high feature correlation. This would not be hugely surprising given that all features are derived from an average movie rating of one form or another. Factors like individual users' genre preferences are likely to correspond in some capacity with overall genre favorability trends, for example.

The methodology, features, and parameters described above are at last applied to the full edx set to predict the ratings in the final_holdout_test set.

The resulting RMSE is 0.892727

Conclusion

A final RMSE of 0.893 means that the XGBoost model outlined in this report is a firm foundation for the development of a movie recommendation system. It executes quickly for large quantities of data and requires relatively few features to render predictions. While its performance is decent, it could benefit from further fine tuning and experimentation. Since there appears to be a moderate to high amount of feature correlation, reducing dimensionality and introducing new features which are not based on average ratings could yield further improvement. This would also help mitigate the potential for model overfitting.

Citations

F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4, Article 19 (December 2015), 19 pages. DOI=http://dx.doi.org/10.1145/2827872