Movielens Capstone

Sarah Sorlien

2024-04-05

Overview

The assignment is to create a machine learning model to predict movie ratings using a specific subset of the movielens data set. We are provided with large dataset for training and testing the model, as well as a smaller dataset, final_holdout_test.The goal of the model is to minimize RMSE yielded by the comparison of the model's predictions of the final_holdout_test ratings to the actual ratings found in the final_holdout_test data.

The following report details my steps of exploratory data analysis, data transformation, model selection and results. The successful model used the XGBoost (Extreme Gradient Boosting) library. When used to predict the ratings of the final_holdout_test set, this model yielded an RMSE of 0.8623448.

Methods and Analysis

Exploratory Data Analysis

We are provided with large dataset for training and testing the model, edx, which contains 9000055 observations of six variables as well as a smaller dataset, final_holdout_test, containing 999999 observations of 6 variables to be used only for the final assessment of the model. The variables included integers for a unique userId and movieId, a numeric rating between 0 and 5 in 0.5 increments, and a timestamp indicating the time and date the review was posted to the system. Additionally there were two chr variables: title, containing the title and year the movie premiered and genres, which contained the various genres the movie fell into each separated by a "|" character. Movies could fall in one or many genres.

Initial data exploration revealed this dataset contained no NA values so there was not an initial requirment to manage NA values or impute values in the dataframe. It is a tidy dataset.

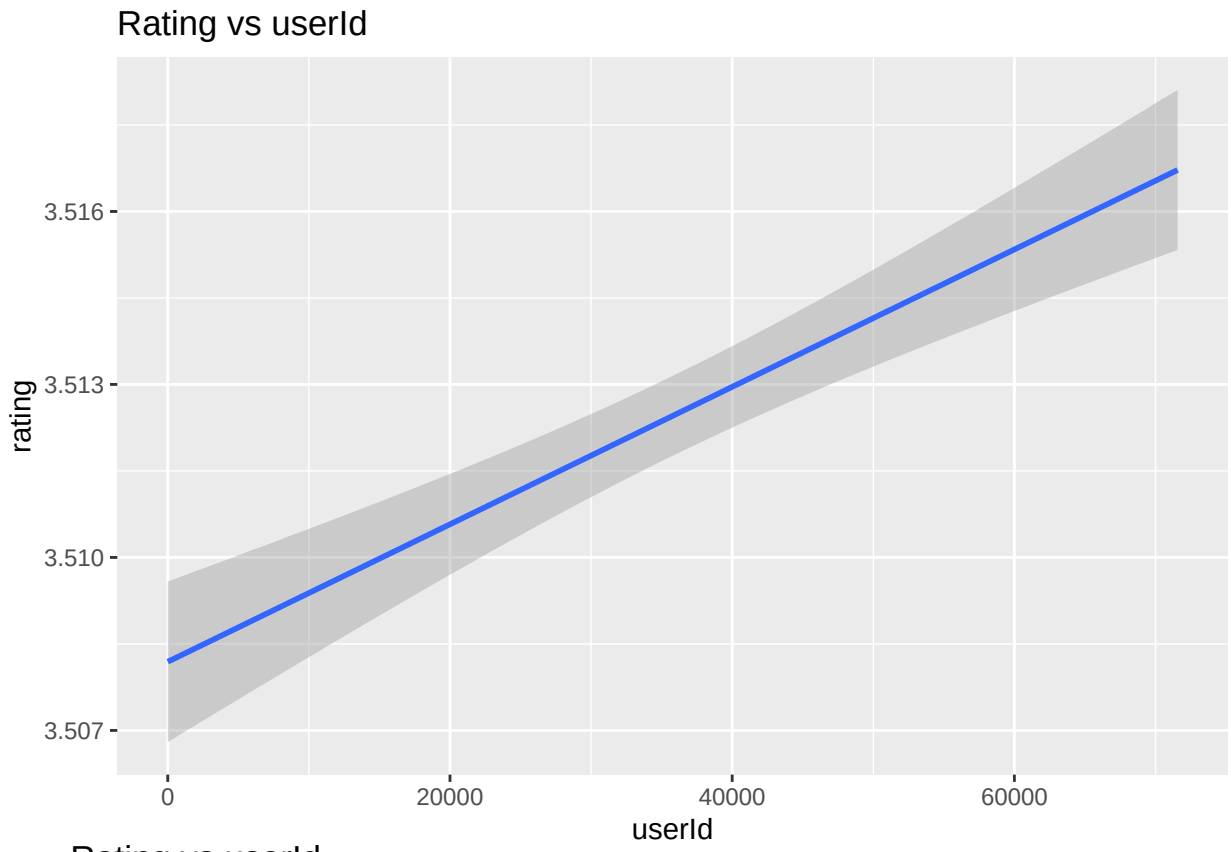
```
load("edx.Rdata")
any(is.na(edx))
```

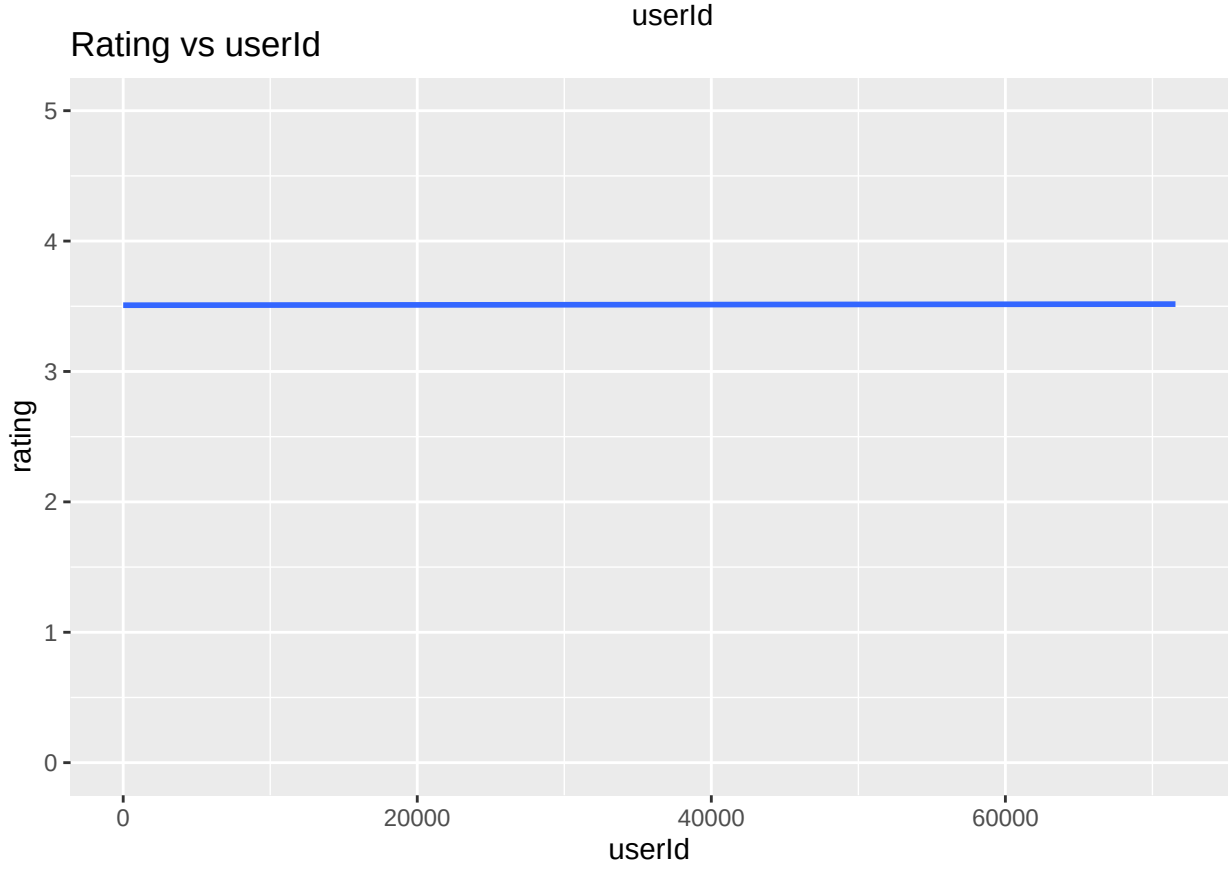
```
## [1] FALSE
```

I also got some sense of the distribution of the ratings in the edx dataset.

```
##   mean_rating median_rating min_rating max_rating sd_rating
## 1    3.512465             4        0.5          5  1.060331
```

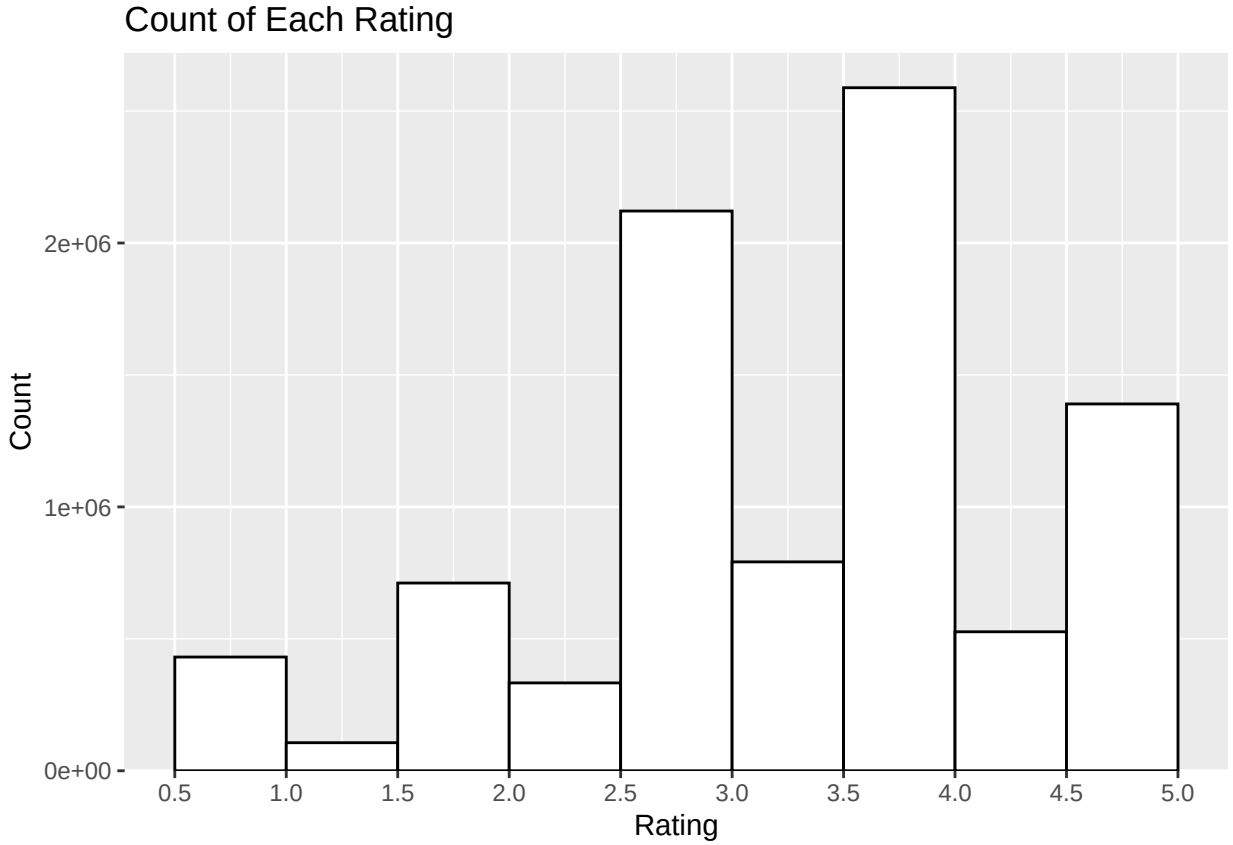
I then looked at the information I could get from the data without much transformation. I examined the relationships between rating vs userId and rating vs movieId. While the initial graphs looked dramatic, it was clear that the range of ratings is very small. To demonstrate this, here are two graphs of the same data, illustrating the importance of having the y axis start at zero. I have used the userID vs rating graph to illustrate this as it had the wider range of rating values of the two.





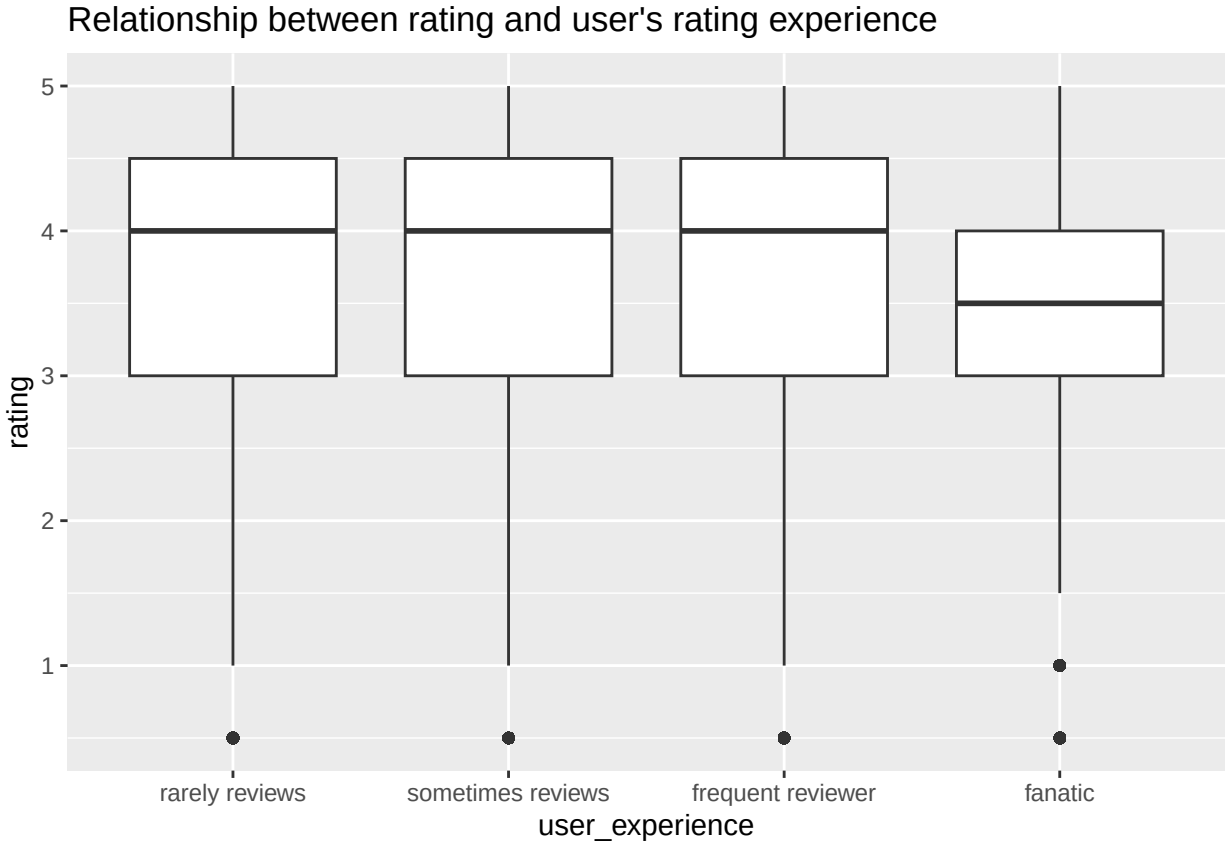
At this point I wanted to see what the distribution of the ratings were overall using a histogram to visualize

the data.



As the ratings distribution was skewed, I considered that there might be a difference in the distribution of ratings per user and ratings per movie in different subgroups of users and movies. This was investigated by looking at the distribution of the quartiles of each of these measures, and distinct differences were noted.I chose to characterize the similar quartile results as factors.

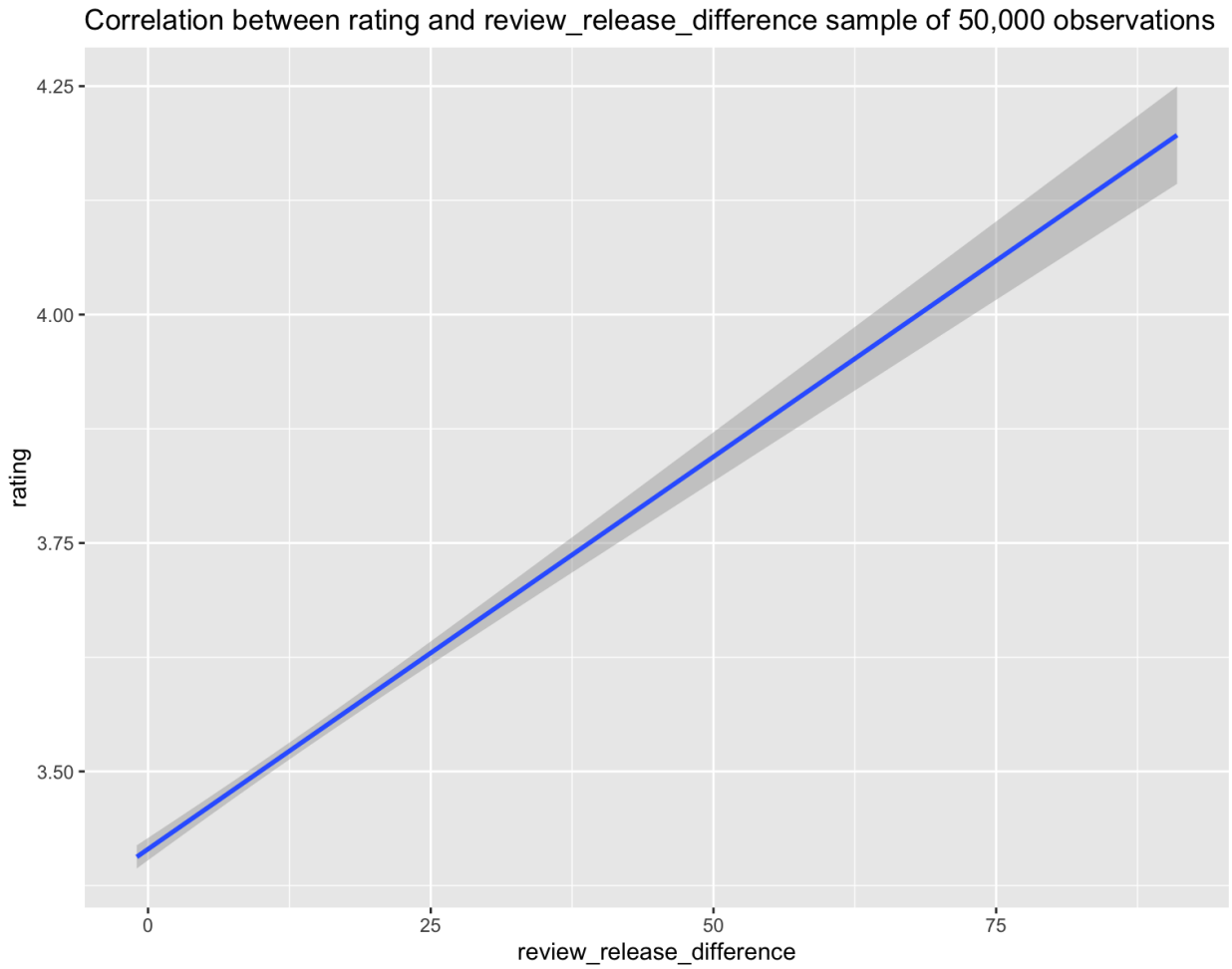
In fact there were differences as seen in the box and whisker charts here of the 4 subgroups of users ratings. As this looked like a useful feature, I differentiated between reviewers who were reviewing "fanatics" who tended to give lower ratings and those who were "not fanatics". This was added to the preprocessed data as a factor in a column marked user_experience. There were similar results for the movies and these were divided into 3 factors with a similar method and labeled movie_experience.



Additionally, I added a column for median user rating.

```
## # A tibble: 5 x 4
##   userId rating title                          user_median_rating
##    <int>  <dbl> <chr>                                      <dbl>
## 1  32620    3.5 Down and Derby (2005)                        4
## 2  40976    3   Africa addio (1966)                          3.5
## 3  59269    2   Rockin' in the Rockies (1945)                3
## 4  60713    2   Won't Anybody Listen? (2000)                 3
## 5  64621    2.5 Confess (2005)                               3
```

At this point I wanted to maximize the amount of information I could get from the few variables I had to work with. The timestamp contained many pieces of information. Obtaining the year would allow me to look at the relationship between date of release and date of review, for example. To get date of the movie release, I did need to extract that to its own column from the title and coerce it to an int. Visualization performed on a subset of the edx data showed this difference could be a useful feature for the model and the review_release_difference column was added to the preprocessed data. There was a positive corellation between the ratings and the length of time between the movie's release and the year of the review. This made sense in the context that better movies might have more staying power over the years.



I did not initially make use of other information in the timestamp, however, after several models under performed, I also added columns for the review_month, review_day_of_week and review_part_of_day (day vs night) to the preprocessed data. The code for this is clearly commented in the accompanying R file in the preprocess_data() function. I did not look at summary statistics for these columns, as the model took into account their contribution and used a reasonable amount of resources. This would be an option if I needed to reduce run time by reducing redundant features or features with Near Zero variance.

The final information to get from the data was the effect of the genres. This information was not standardized in any way as any number of genres could be applied to a movie. Converting all the possible combinations to factors seemed unwieldy and a way to introduce NA values when using the model on a new data set that may not have all of the specific combinations in the test set.

I elected to use a list of all unique genres in the training set to simplify the preprocessing function. Hard coding like this can cause issues with future use on a new data set, but it is useful for consistency in the model. In addition to using the code below, transform genres using a "-" rather than an underscore to allow proper processing.

```
#find all genre combinations
unique_genres <- edx %>%
  select(genres) %>%
  distinct() %>%
  arrange(genres)
#dim(unique_genres) # There were 797 distinct combinations of genres.
#Instead I built a list of the individual genres
```

```
split_genres <- strsplit(edx$genres, "\\|")
genres_list <- unique(unlist(split_genres))
genres_list
```

```
##  [1] "Comedy"             "Romance"            "Action"
##  [4] "Crime"              "Thriller"           "Drama"
##  [7] "Sci-Fi"             "Adventure"          "Children"
## [10] "Fantasy"            "War"                "Animation"
## [13] "Musical"            "Western"            "Mystery"
## [16] "Film-Noir"          "Horror"             "Documentary"
## [19] "IMAX"               "(no genres listed)"
```

In preprocessing, I converted each of these to a column labeled with the name of the genre from the list and a logical variable indicating if the movie was associated with that genre. This is called one hot encoding and is a useful way to manage character based variables in a machine learning model. Once this encoding was performed, I could come up with a mean_rating_by_genres column that took the average of all the mean ratings of each genre with which the movie was associated. In this way a film that was Action|Adventure, for example, would have the mean rating that was half the mean rating for the Action genre alone plus half the mean rating for Adventure genre alone.

When I had these transformations, I combined them into a function for preprocessing to apply identically to edx and final_holdout_test. When it came time to process the final_holdout_test set, however, it became apparent that my code produced NA values in the mean_rating_by_genres column when the test set had no movies with a genre from the training set. As these would not be used in any calculations (the logical variable for that unused genre would be false), I considered handling the NA values by substituting 0. I decided ultimately to use the mean of all ratings instead lest the 0 cause a problem down the line.

After deleting the title, timestamp and genres columns from the preprocessed data, I had 33 features and the target column, rating, to use for training and testing the models. All of these transformations are included in the function preprocess_data() in the movielens_capstone.R file submitted.

```
## [1] "Preprocessed edx dataset has 9000055 observations of 34 variables."
```

```
## [1] "Preprocessed final_holdout_test dataset has 999999 observations of 34 variables."
```

Model Selection

The goal of the predictive model was to minimize RMSE when predicting ratings from the final_holdout_test data. My target was RMSE < 0.86490 to maximize the points I could earn on this venture. The first model I tried was a simple naive model. After dividing the edx data into a training and test set. I used the mean of all the ratings in the training set to predict the rating in the test set. As anticipated this was not a great predictor.

```
## [1] "The RMSE for the naive model is  1.06033134095774"
```

```
## [1] "The Accuracy within tolerance is  0.375559260471186"
```

I then attempted several other models on my preprocessed data from edx. This was a real Goldilocks experience.

I started with a random forest model using the ranger library as it is reported to be faster than rf (Random Forest) from the caret library. It did provide an (inadequate) result in a reasonable amount of time without cross validation. I attempted to run the model with cross validation and tuning to improve the performance and 24 hours later it was still running. It was too resource intensive to be useful. As I have a fairly powerful laptop (MacBook Pro, M3 8 core, 38 RAM) I gave it another try with parallel processing. (library: doParallel). Again, it was just too resource intensive to be practical.

Another type of model useful for recommendation engines is a Collaborative Filtering type model. The recommenderlab library provides several of these models that combine like users, for example, to generate

recommendations. With this library I looked at User Based Collaborative Filtering (UBCF) and Item Based Collaborative Filtering (ICBF) models. In this case the item was the movieId. I was able to train both models with a minimum of features: rating, userId, and movieId, with RMSE results > 1. This model required data transformation to a RealMatrix for the model production and there were numerous issues with getting my data to comply with the model requirements. I will need to learn more before using this model again, but did abandon it due to the performance.

I also worked with the glmnet library. These models are regularized linear or logistic models that can be tuned and cross validated. The glmnet model without tuning or cross validation yielded an RMSE of >1. With tuning parameters and cross validation, I let the model train for 48 hours until I basically ran out of memory and my laptop made a sound that may have been a sigh.

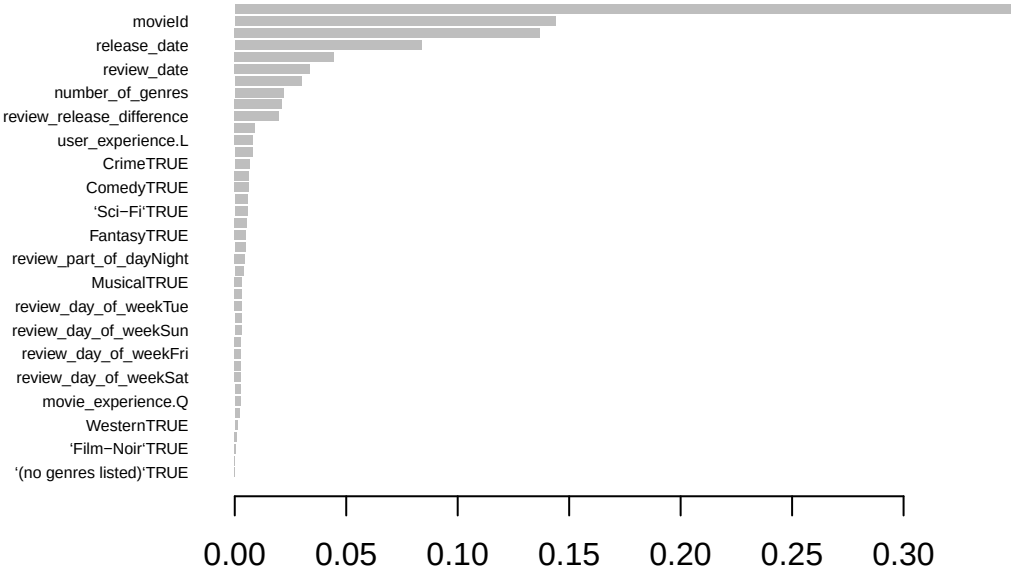
I then turned to the XGBoost library for an Extreme Gradient Boosting model. This is a type of decision tree model that does serial iterations of the tree, learning from its past errors. There are tuning parameters to avoid overfitting. It is incredibly easy to implement. It does require one hot coding for chr variables, but I understand there are versions where the model accomplishes this on its own. In my case, I had already structured my data in this way. This model proved to be the best choice of models for the project based on resource usage and accuracy.

Results

An XGBoost (Extreme Gradient Boosting) was used to predict the ratings in the final_holdout_test set. Training and testing the model on the preprocessed edx dataframe suggested this was the most accurate of all models I had tested. The RMSE results of the training model are noted in the movielens_capstone.R file attached. While these results of the model on the preprocessed edx data had not reached my RMSE target, I decided to proceed with assessing the model on the final_holdout test set and move on.

While training the model, the initial 1000 iterations showed the model was still improving with each iteration. It is very satisfying to watch as it prints out the results of the iterations at regular intervals. I tuned it by running again with 10,000 iterations and it was still improving. Ultimately setting iterations at 20K yielded the best model. The total human scale time of training the model on the preprocessed edx data was almost 2 hours.

From the final model, you can get an understanding of the contributions of the various features to the prediction. The plot below shows that the movie itself was the most important determinant of rating in this model.



Running the xgboost model on the final_holdout_test the model predicted ratings with an RMSE of 0.8623448

which was below the target.

Conclusion

This report describes the successful development of a supervised machine learning model using the XGBoost library to predict movie ratings from tabular data derived from the movielens dataset.

Exploratory data analysis suggested possible features for the model, many of which required the preprocessing and transformation of the original data. This was accomplished with a function, preprocess_data(), that could be applied to the training data (edx) as well as the final_holdout_test data or any future dataframe generated from movielens with the same columns for each observation.

The features with apparent non linear distributions, ratings vs counts of reviews per movie or per user, were grouped into quantiles with similar distributions and transformed into ordered factors. Using log transformations of these data was not evaluated, and would be appropriate to test to refine the model in the future. Similarly I opted to use median rather than mean per user rating as a feature, so the results from these two different methods could be compared in the future.

Other avenues to attempt improvement of the model,aside from those already mentioned, would include seeking more information from the movielens dataset such as user's geographical location, for example, to use as predictors. Combining models is another option to improve user predictions, as is digging deeper into the User Based Collaborative Filtering and other models in recommenderlab.

Genres were transformed with one hot encoding and average rating_per_genres across all genres linked to a movieId were calculated. As this could result in NA values if a test set did not contain all genres in the larger training set, a method for handling the NA values was required when pre processing the data.

Extreme Gradient Boosting (XGBoost) is a decision tree model that produces an ensemble of weak prediction models. It has a reputation of high performance and scalability. While it did take over an hour to run on my MacBook Pro M3 laptop, it was less resource intensive than either the random forest model or glmnet model on the preprocessed edx dataset. The XGBoost model was also much easier to implement and more accurate than the recommenderlab user or item based collaborative filtering models (UBCF, IBCF).

In the end, the XGBoost model was successful, yielding an RMSE of 0.8623448 on the final_holdout_test data.

References:

`Introduction to Data Science, Rafael A. Irizarry, 2019`
`R for Data Science, 2nd edition,Hadley Wickham, Mine Çetinkaya-Rundel, and Garrett Grolemund`

https://r4ds.hadley.nz

`Introduction to Extreme Gradient Boosting in Exploratory, Medium blog post, Kan Nishida 2017`

https://blog.exploratory.io/introduction-to-extreme-gradient-boosting-in-exploratory-7bbec554ac7

`Documentation for recommenderlab:`

https://cran.r-project.org/web/packages/recommenderlab/recommenderlab.pdf

`Documentation for XGBoost:`

https://xgboost.readthedocs.io/en/stable/

`Documentation for ranger:`

https://www.rdocumentation.org/packages/ranger/versions/0.16.0/topics/ranger

`An Introduction to glmnet:`

https://glmnet.stanford.edu/articles/glmnet.html