5-Star Trails?

Predicting the rating and popularity of America’s National Park Trails

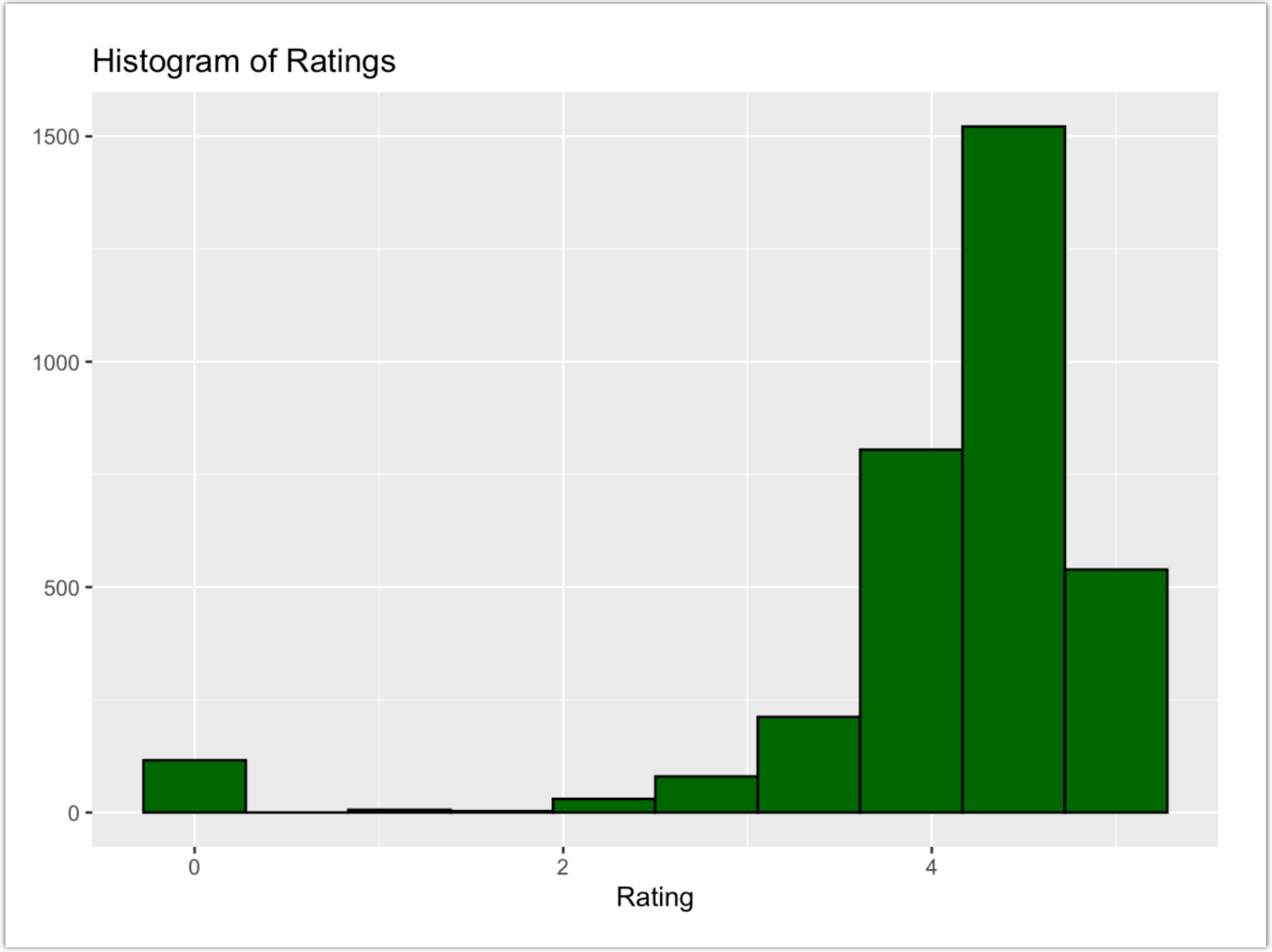
The Dataset

- Data on National Park trails was collected from the website Alltrails.com, which catalogs data on trails worldwide and provides a forum for people to discuss their hikes, add their photos, etc.
- The dataset includes all of the trail's information from this Alltrails, including both quantitative and qualitative information.
- Quantitative data includes the length of the trail, elevation gained, number of reviews given to the trail on the website, latitude & longitude.
- Qualitative data includes the park the trail is in, the type of route (loop/out and back?), the difficulty rating, usage level, and even some features and activities associated with the trail (Dog-friendly, wild-flowers, hiking, birding, etc)

Example: Angels Landing Trail

trail_id	name	area_name	city_name	state_name	popularity	length	elevation_gain	difficulty_rating
10006571	Angels Landing Trail	Zion National Park	Springdale	Utah	84.6229	6598.294	492.8616	5

route_type	visitor_usage	avg_rating	num_reviews	features	activities	latitude	longitude
out and back	4	5	3903	[‘dogs-no’, ‘partially-paved’, ‘river’, ‘views’, ‘wild-flowers’, ‘wildlife’]	[‘hiking’, ‘nature-trips’, ‘trail-running’]	37.25928	-112.9517



Our Response Variable:

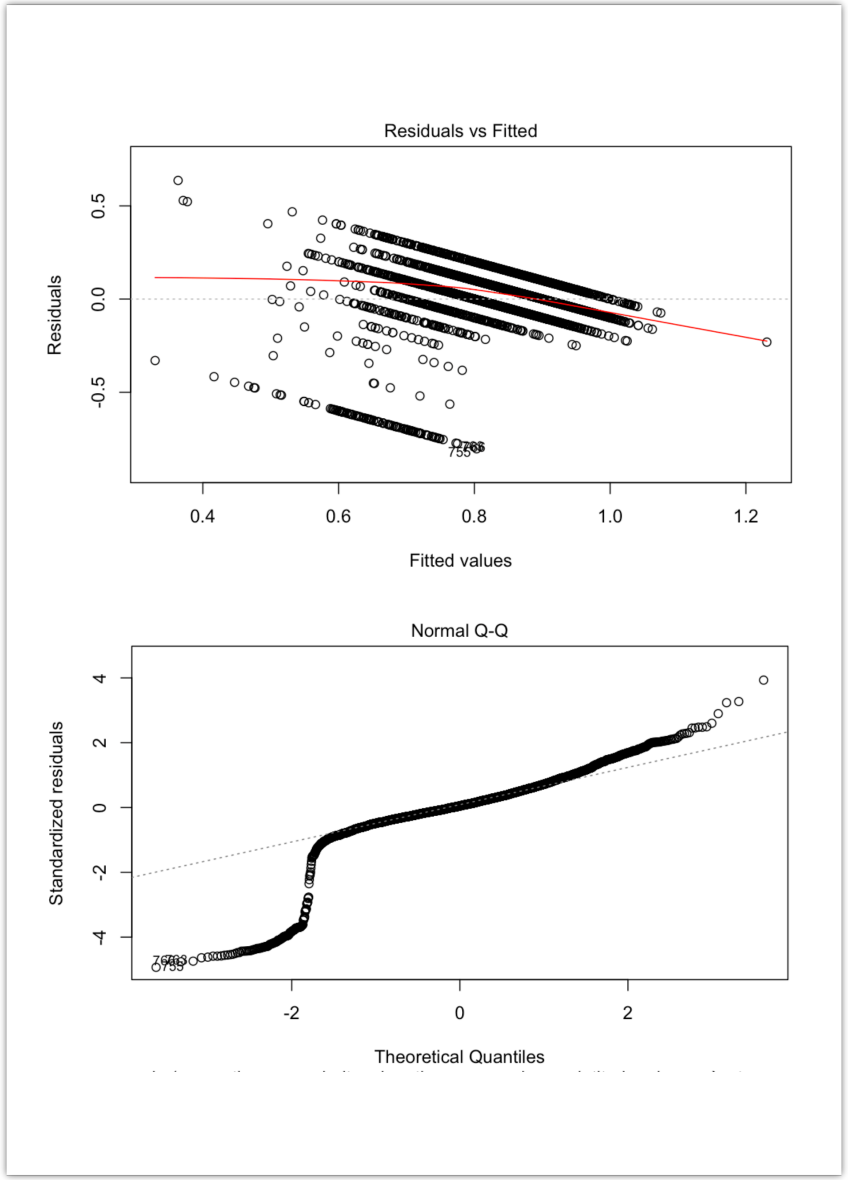
- Star Rating, which is measured by the stars granted to each trail by the users of the Alltrails website. Can you see the issue that I missed?

Data Treatment

- For the overall dataset used in all models, I removed the State, City, Name, and trail_id before prediction.
- The features and activities were separated out into their own categories, and a 0 or 1 based on if it is part of the trail (similar to one-hot-encoding)
- Popularity, Length, Elevation Gain, Rating, and Number of Reviews all got normalized
- Difficulty Rating and Visitor Usage were recoded into factors (1="Easy"(difficulty)/"Light"(usage) , 2="Moderate", etc.)

First thought – Regression!

- The logical first choice for this data was to try a regression.
- After checking for collinearity (there were none that were obvious to remove, although number of reviews and popularity showed about a .8 correlaton), I ran a backwards step selection on the model using the Akaike Information criterion (AIC).
- I also added an addition term using the log of popularity – I found no other terms that would be useful.
- Model: RSE = .1633, meaning we missed predicting by about .8 stars! The adjusted R^2 of the model came out to .2565, so we're not reducing the error very well with this model.
- Notes:
  - The AIC removed longitude, but kept latitude which was a significant predictor at p = .002.
  - Popularity and number of reviews were the most significant predictors here based on the AIC.
  - Higher difficulty and lower usage led to a higher rating on average.

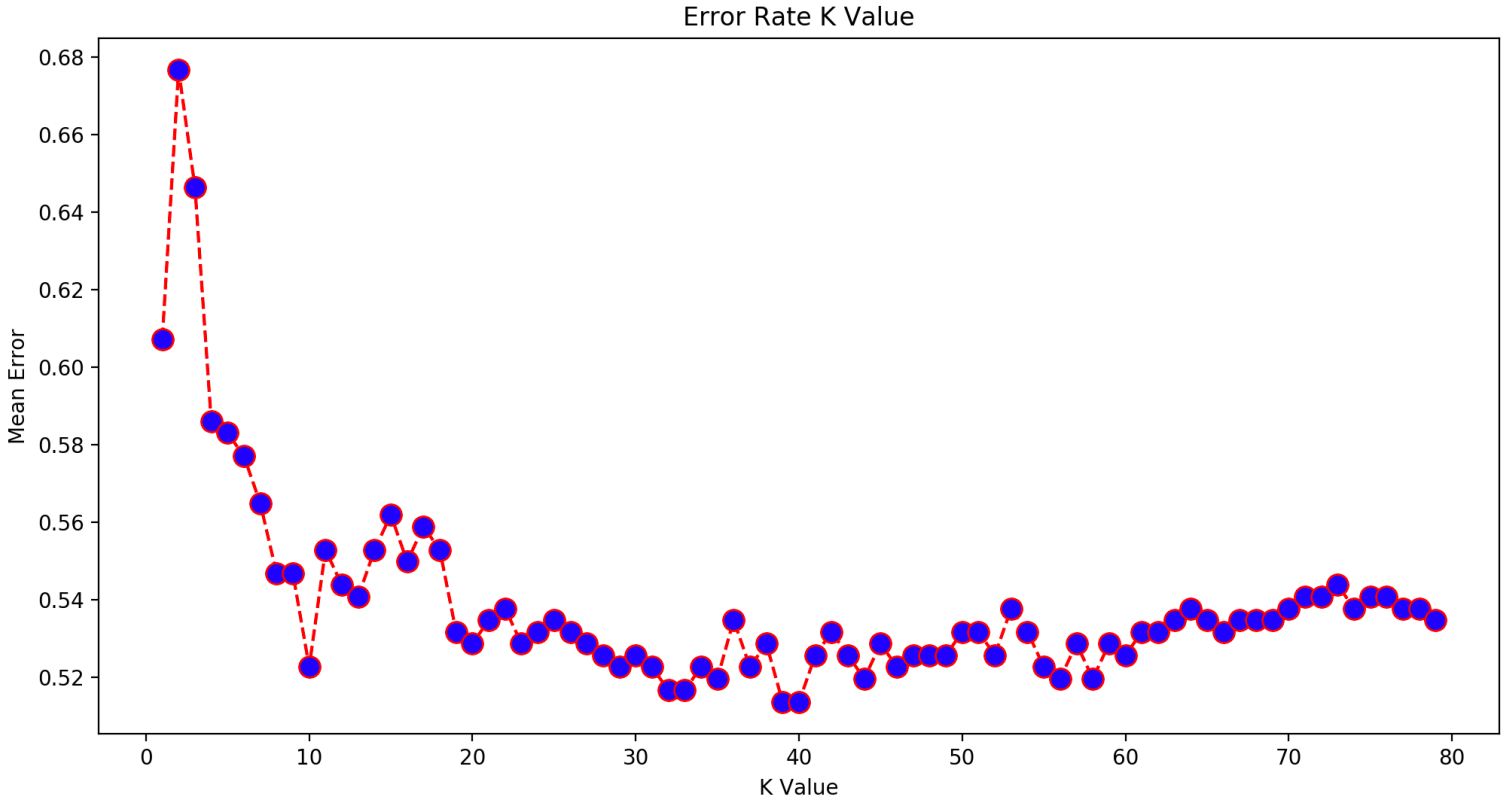


Issues with Rating

- After checking the residuals plots, I noticed something I had missed before – it's not continuous like I had thought, but rather discrete and finite.
- Specifically, the rating only measures between 0 and 5 by the half star – giving us only 10 possible outcomes for our rating (there were no trails rated at .5 stars)

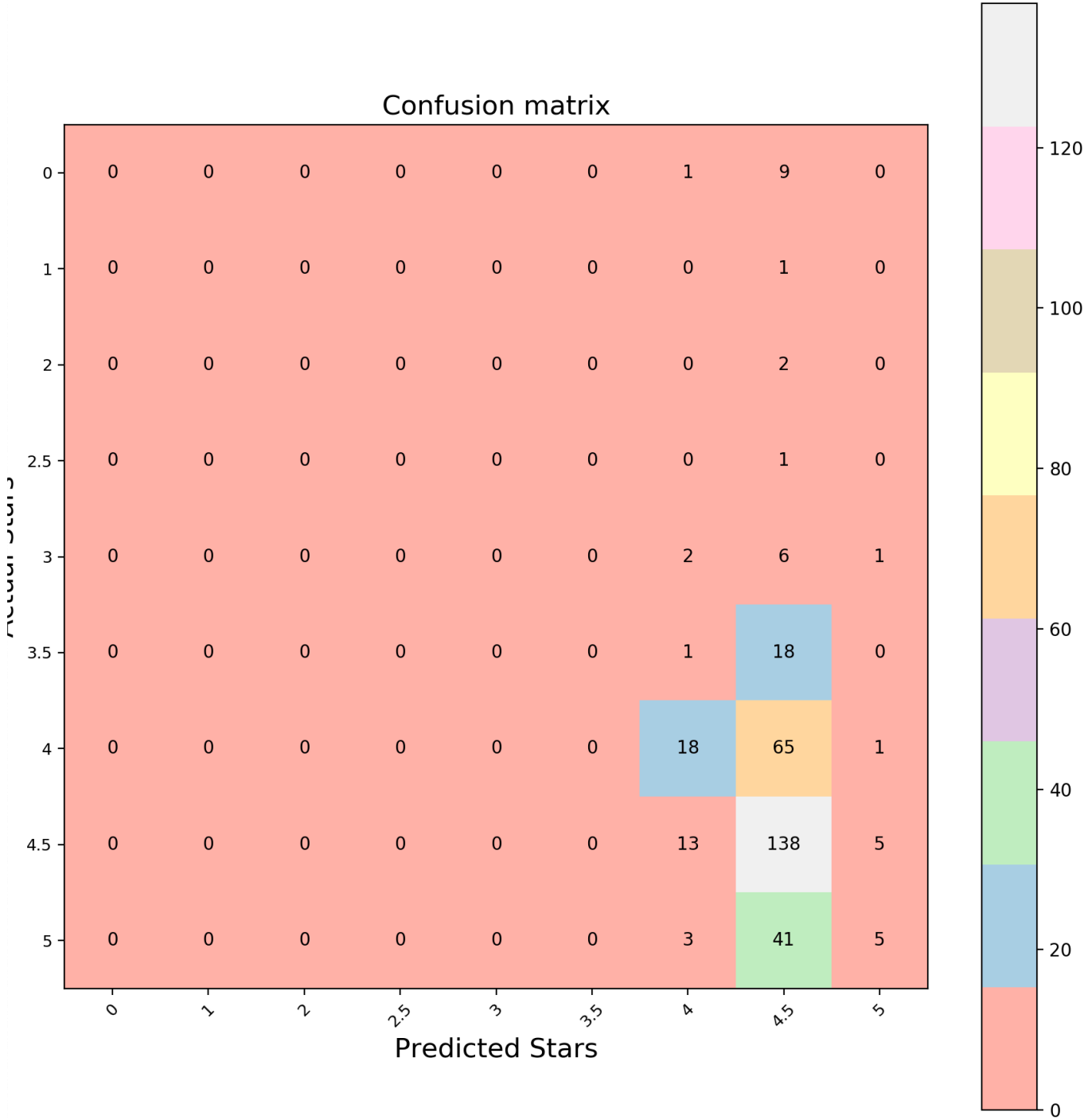
New Goal?

- After figuring that out, I became interested in whether or not it makes more sense to train various models on this data predicting as a category ("1 star", "2.5 stars", etc) or still as a discrete numerical measure.
- I ended up fitting 5 models – a K-nearest neighbors model on the category of stars, an XGBoost model and Random Forest model on the numerical measure, and 2 Neural Nets – one for each.
- To make sure my measures were the same after each (to allow comparisons), I forced each model to give one output predicting the star value, and then measured the accuracy of that prediction on the labels.
  - Categorical models picked the most likely category of star rating
  - Numerical models predicted the number and then rounded to the nearest half-star
- Each model used the same seed and 90/10 test/train split



First Categorical Model: KNN

- After fitting a KNN classifier, I ran it through a loop to see how many neighbors achieved the lowest error
- We see that the lowest mean error rate is at k=40

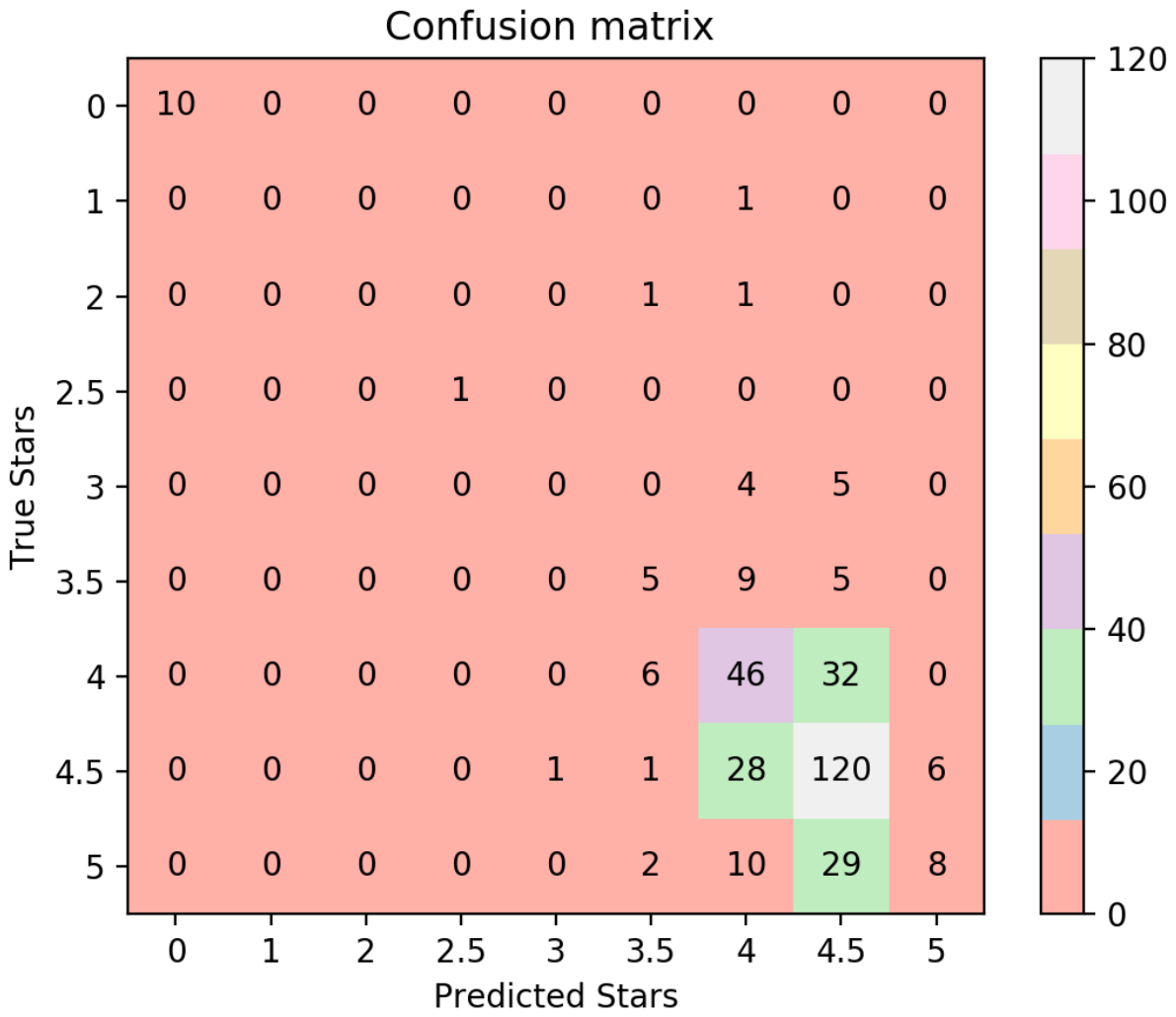


KNN Confusion Matrix

- Using the best KNN model, at k=40
- The classifier correctly predicts 161/331 observations, giving us an accuracy rate of **48.6%**
- Unfortunately, given k=40, the classifier has difficulty classifying anything as other than 4.5 stars

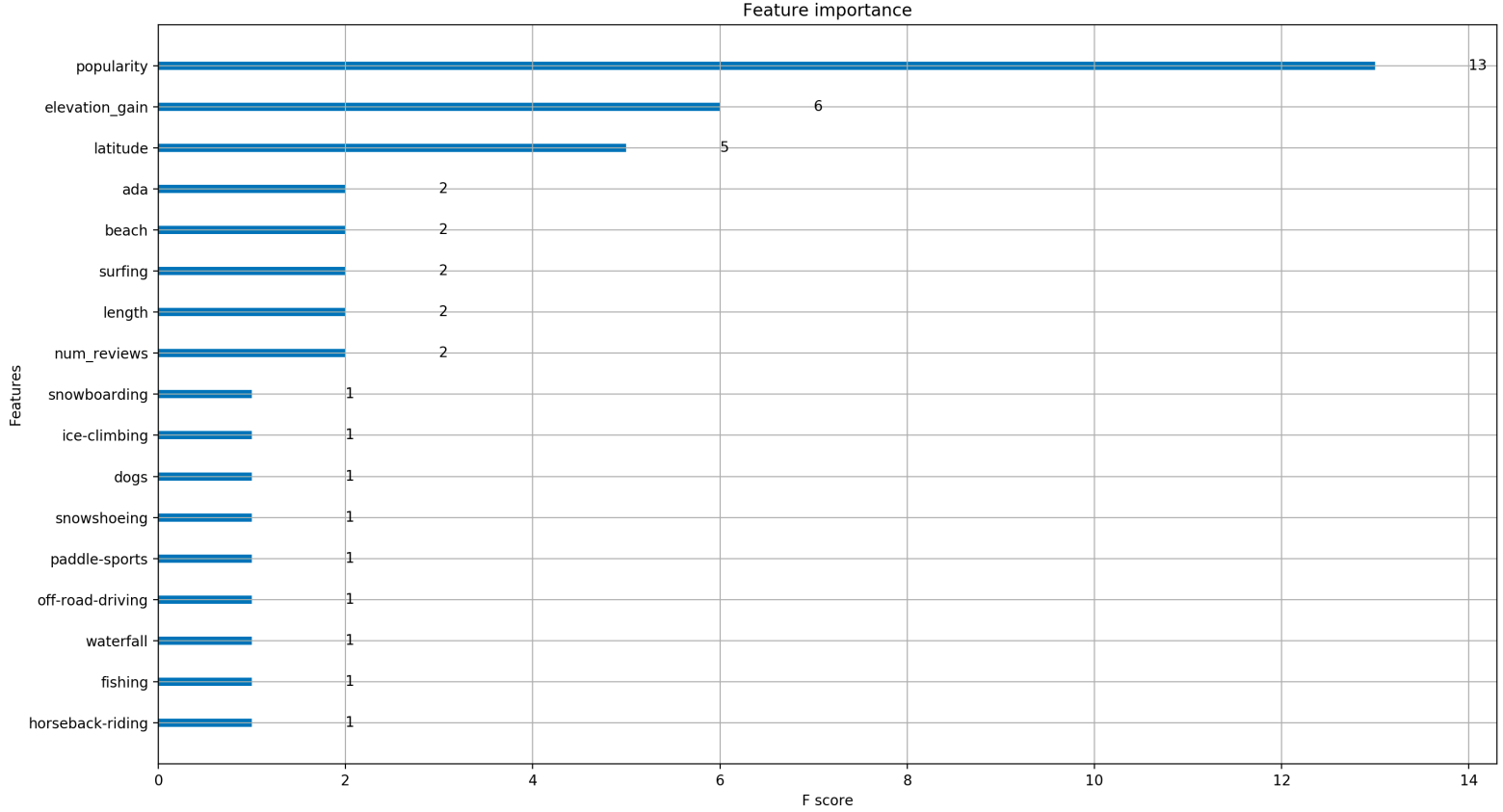
Random Forest Model

- Next, I fit a Random Forest model on the data to see how well it could predict the number of stars.
- The model was fit using 1000 decision trees and then finding the optimal model from the average of those.
- The baseline prediction would be about 4.1 stars, which would have an error from the actual dataset of about .57 stars.
- After fitting the model, our error drops to having an error of only .06 stars – a great improvement!



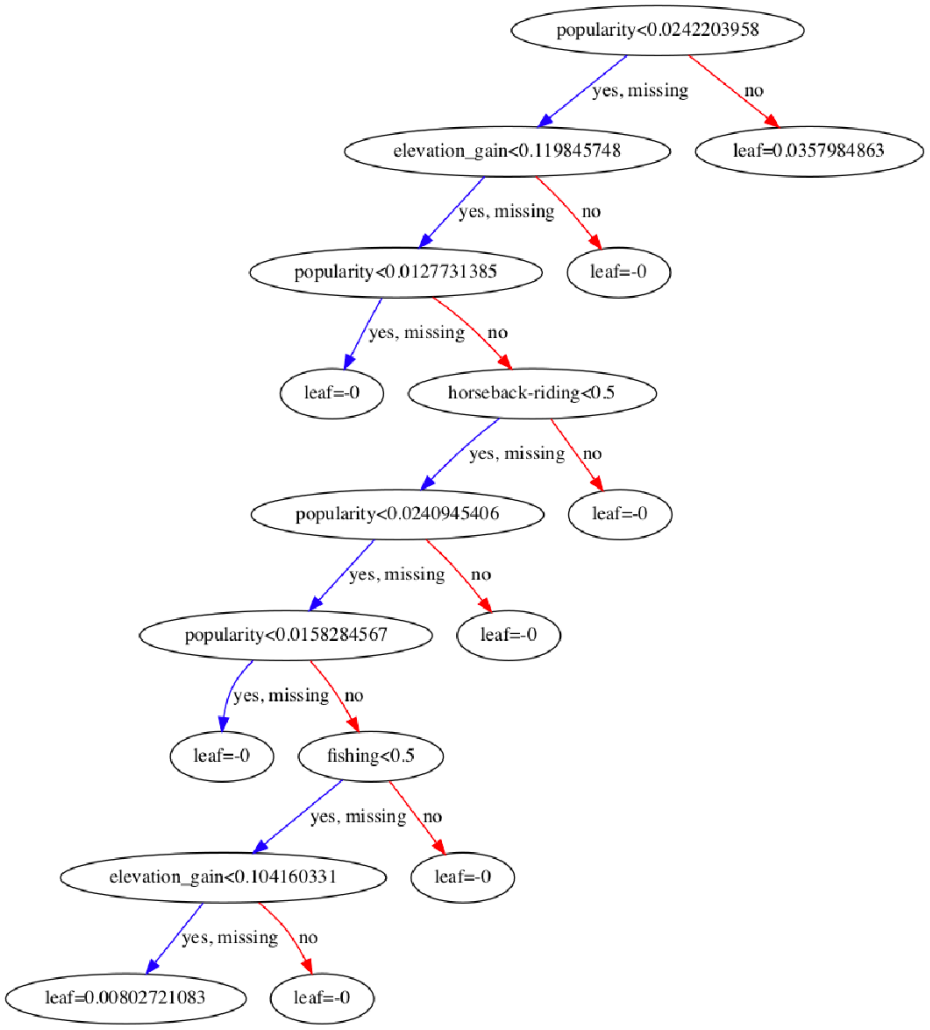
Random Forest Confusion Matrix

- Turning the Random Forest predictions into a confusion matrix to compare with the other models, we get these results.
- The model correctly predicts 190/331 observations, giving us an accuracy rate of **57.4%**
- This model does a better job of separating the classes as well as having a better overall accuracy
- While the overall error was only .06, this is largely due to predicting nearly 57% perfectly, reducing the overall error term.



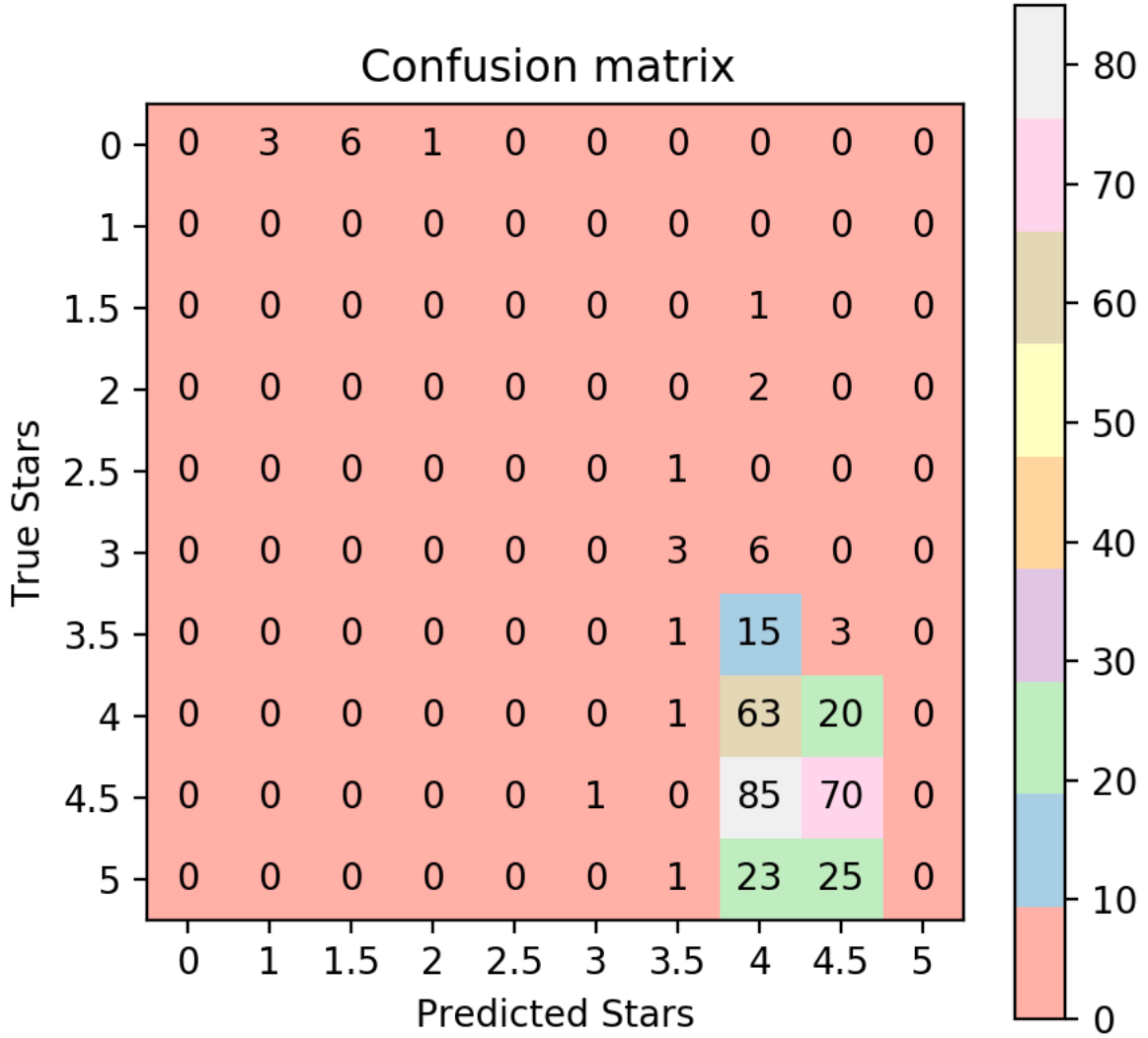
XGBoost Model

- Using XG Boost, we can figure out what features are most important for prediction
- Popularity is highly useful to the model, which was also reflected in our regression earlier
- Elevation gain and latitude are our other more important predictors
- Some like surfing, snowboarding rate highly herebecause they are rare



Example tree from XGBoost

- There seems to be some issues with the 'leaf' values that I can't quite iron out, however it shows where the tree made the splits.
- It used popularity quite a bit, which makes sense based on the feature importance it was given

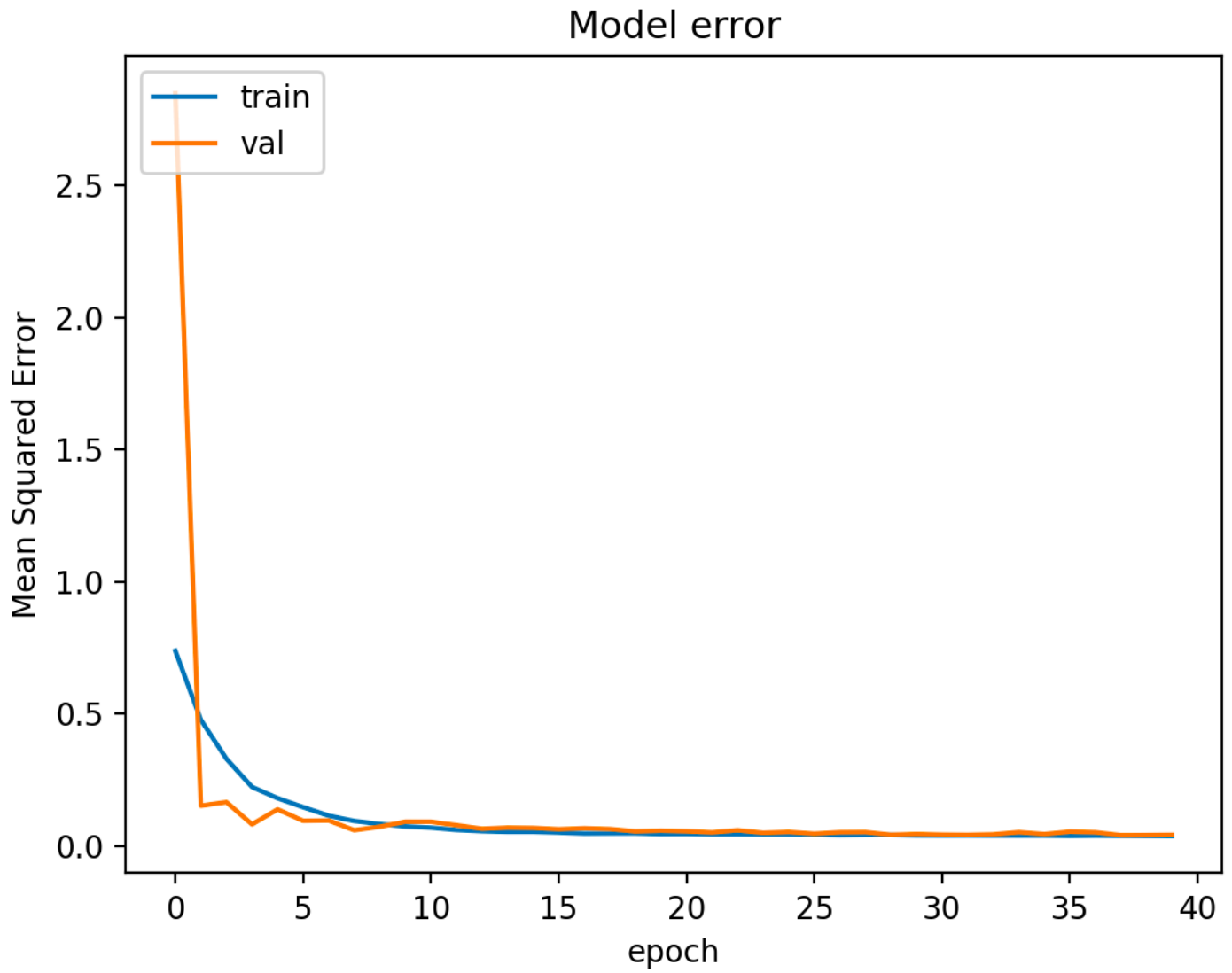


XGBoost Confusion Matrix

- This confusion matrix comes from the Random Forest method and the previously described method for comparing the values (rounding the star prediction and comparing the number right)
- The model correctly predicts 134/331 observations, giving us an accuracy rate of **40.5%**
- This model does a better job of separating the classes

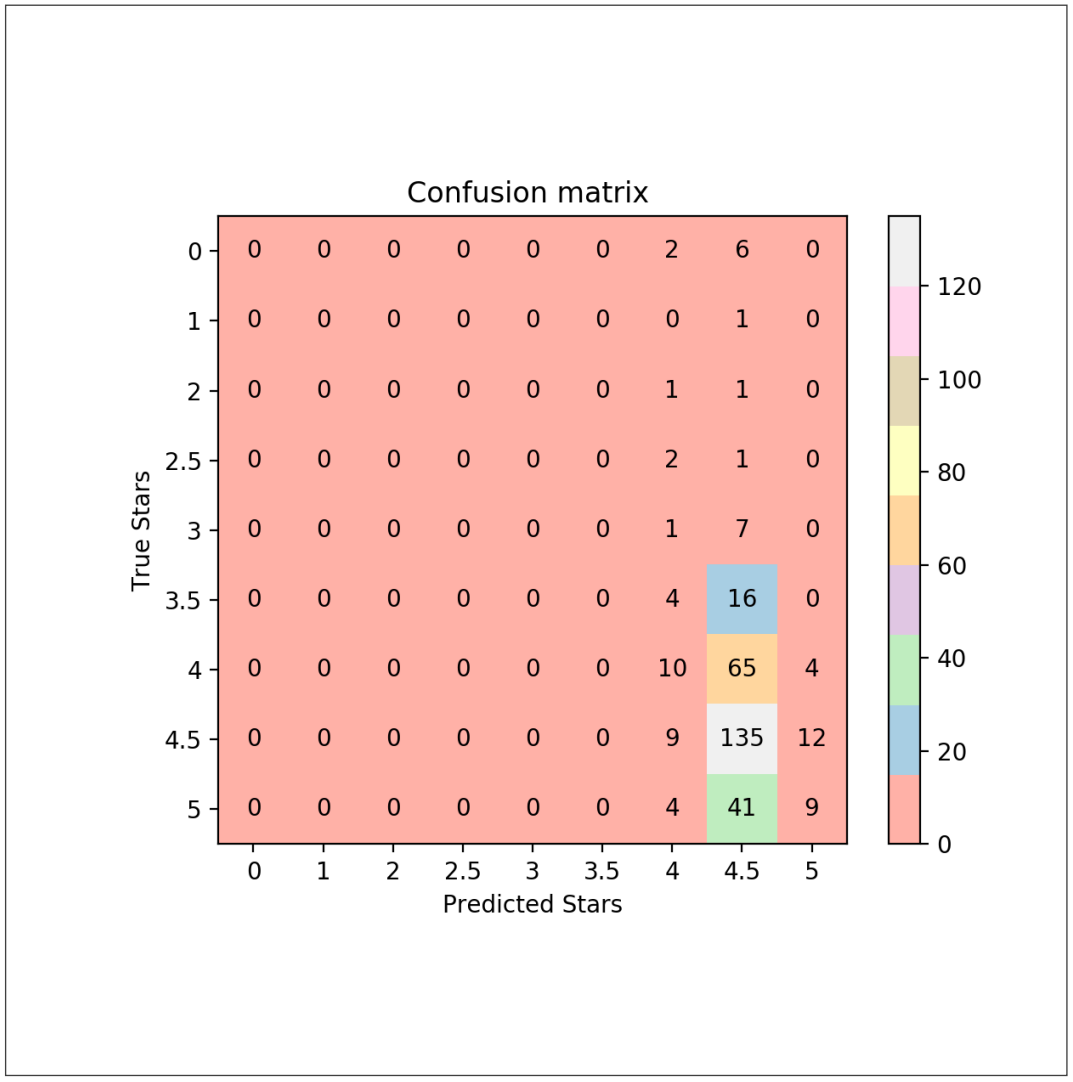
Moving on to the Neural Networks

- Each Neural network will be largely the same, except for the output layer (10 outputs vs. 1 output)
- It will consist of an input layer (65 dimensions), two fully-connected hidden layers of 75 neurons each, and then an output layer (10 with a softmax activation for categories, and 1 output with no activation for the numerical.
- Each layer (except the output layer) uses batch normalization and the ReLU activation function, with a 40% dropout between layers to help prevent overfitting.
- After each epoch, the model is validated on a 20% of the data that was separated out before. At the end of the fitting, the model is tested on another separate 10% of the data.



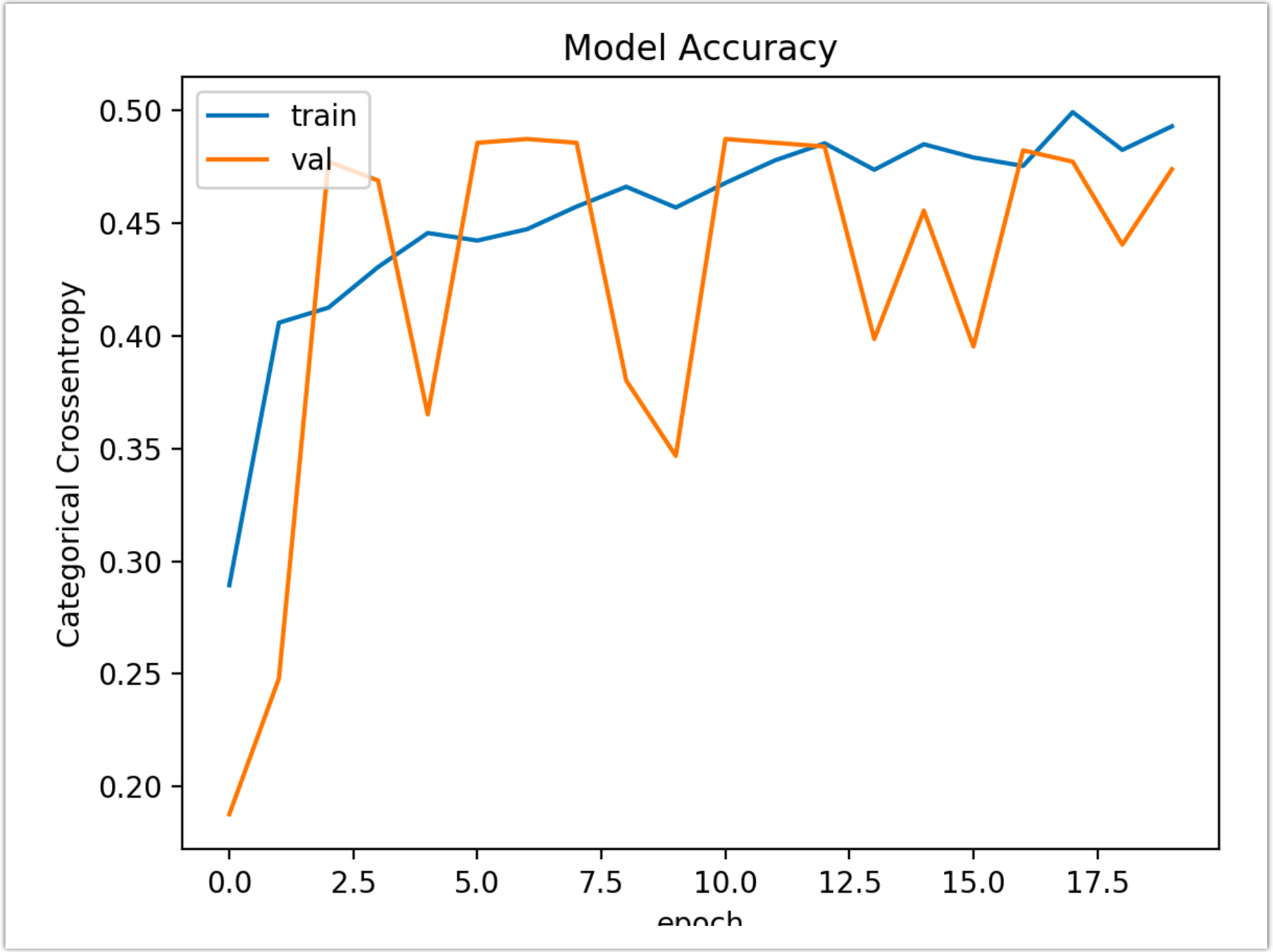
Numerical Model Error

- Oddly, the validation accuracy seems to start out lower than the training data, and stabilizes to it's best value early
- Training accuracy drops quickly, then slowly over time (normal)
- Test accuracy on this model achieved a mean squared error of .044, which lines up with the validation accuracy



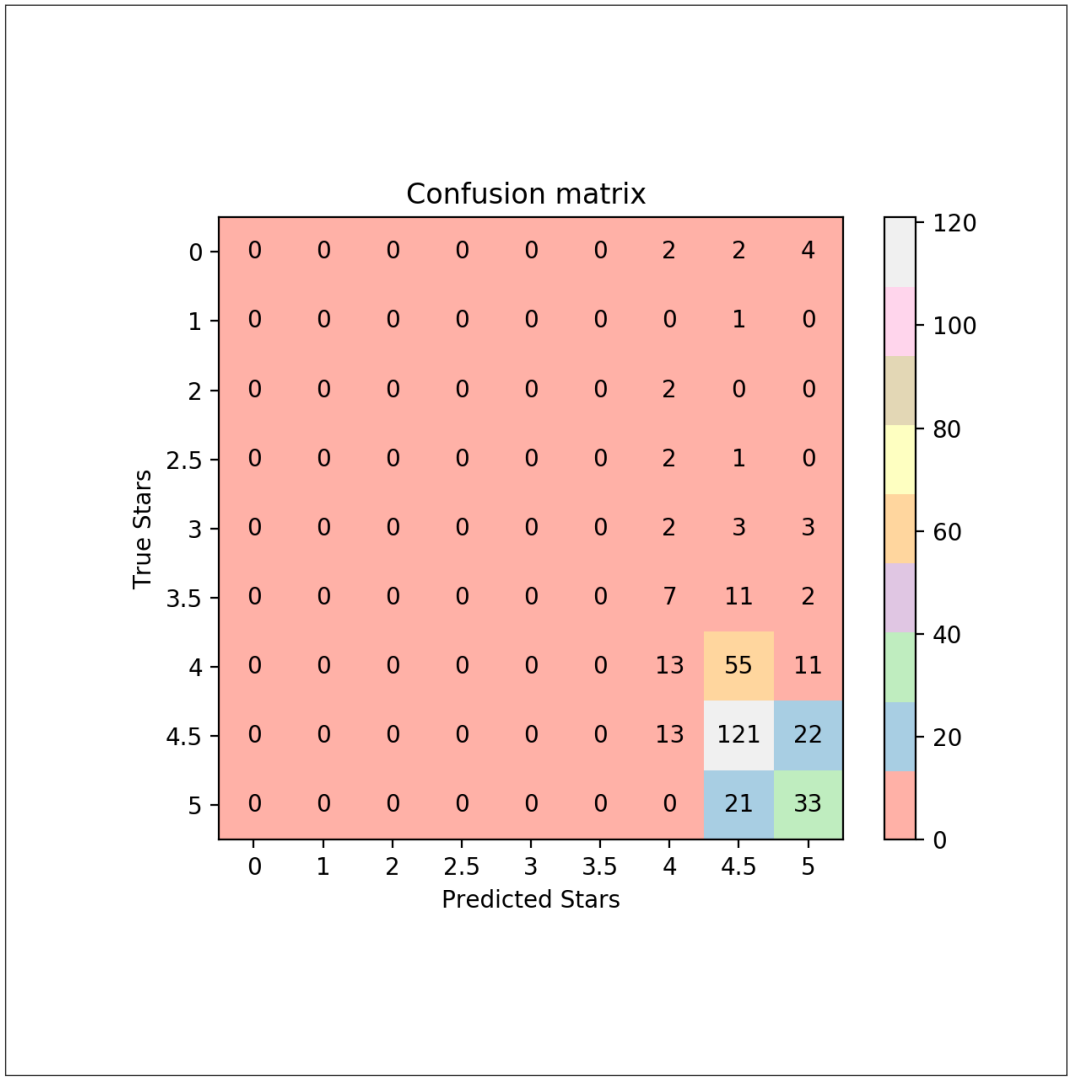
"Numerical" Neural Net

- The model has difficulty predicting anything other than 4.5 stars, so it does not perform much better than the ground truth.
- Overall accuracy here is only 144/331, or **43.5%**
- Ground truth here would actually be a bit better (146/331), so this model could use a bit more thought and tweaking.



Categorical Model Accuracy

- Validation accuracy moves around quite a bit, even after testing other learning rates
- Training accuracy improves over time normally
- Test accuracy on this model achieved a cross-entropy accuracy of of .46, similar to the validation.



"Categorical" Neural Net

- Overall accuracy here (which is likely wrong) is only 154/331, or **46.5%** (this lines up with our crossentropy number as well)
- This too is clustered down around the higher star counts, similar to the KNN and Numerical NN model. It doesn't predict anything to have a star rating of less than 4 stars.

Results

- The KNN classifier correctly predicts 161/331 observations, an accuracy rate of **48.6%**
- The Random Forest model correctly predicts 190/331 observations, an accuracy rate of **57.4%**
- The model correctly predicts 134/331 observations, giving us an accuracy rate of **40.5%**
- Overall accuracy on the Numerical Neural Net was 144/331, or **43.5%**
- Overall accuracy on the Categorical Neural Net was154/331, or **46.5%**

Analysis

- So when it comes to predicting the rating of trails from this data, the Random Forest Model performs the best.
- The KNN (categorical) did second best.
- Both Neural Nets did not perform well, I think there are definitely ways to improve those, possibly by tweaking the learning rate a bit more and perhaps adding a layer. The "Numerical" neural net especially could use another look, it actually did a bit worse than if it predicted the most popular star rating (4.5) for every observation, but did better than random guessing which would be roughly 10% accuracy.

Final Thoughts/ Other ideas for this dataset

- Alltrails.com allows users to upload images to their website - it could be interesting to scrape just a few of those images and train a CNN on the 'quality' of the trail, or potentially find out other information just from user-uploaded images.
- I would like to do the same analysis for a smaller subset of these predictors – as seen by the XGBoost feature rankings and the original regression, popularity was easily the most important variable. Perhaps these models would work better without that dominating the models so much.

Thoughts/Questions?