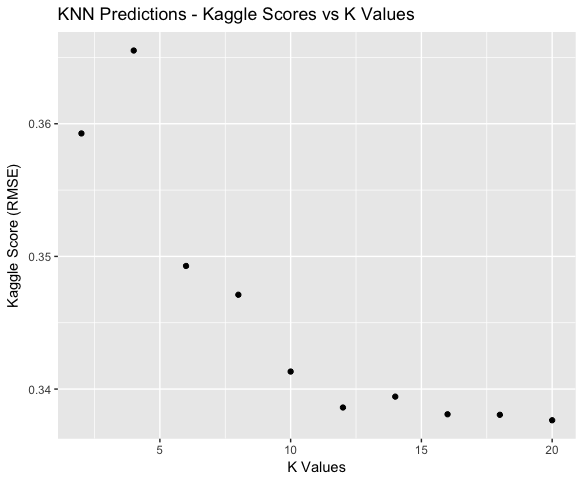
**Method Details**

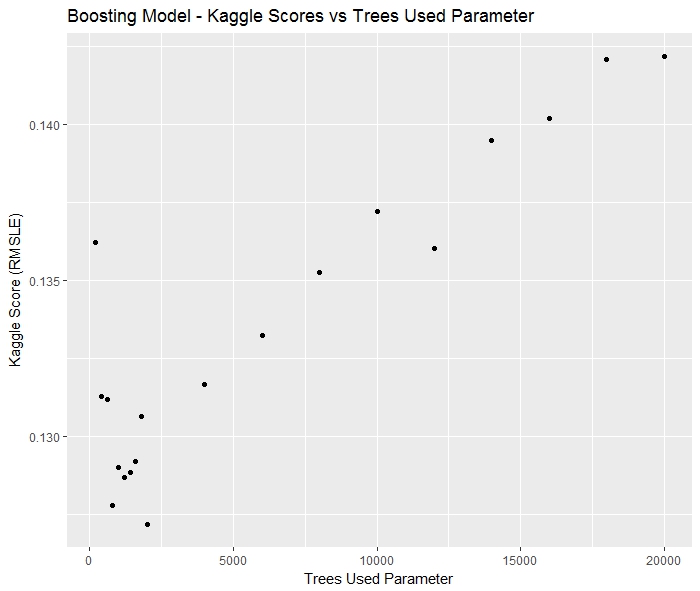
As a quick precursor, the only measure of fit at hand for our group models/data was Kaggle’s model fit score (RMSE) that was returned to us upon upload of a .CSV file of predicted response figures. The Kaggle score index is 0 to 1, where a score of 0 equates to a perfectly fitting model, while 1 would be a terrible fit. We unfortunately were only able to work with this measure because our test set lacked a response variable. Therefore, on all the graphs below, the y-axis will represent Kaggle score instead of MSE.

I had the responsibility of running KNN models, GAM, and testing some boosting models for this project. For the KNN model, I had written a loop for the model that created 10 models. The sole discrepancy between these models was the size of the tuning parameter k. This initial run created 10 models starting with k=2 up to k=20, outputting a model at every increment of 2 (i.e. k=2, k=4, k=6…, k=20). I then created the plot below using the “ggplot2” function to represent the scores I received from Kaggle regarding the respective fits of these 10 models:

****

This plot hinted to me that perhaps I should not have not stopped increasing the value of k at 20, as k=20 yielded the lowest error score out of all of these 10 models. I then used the same loop as before to run 6 more KNN models from k=20 to k=50, this time outputting a model every 5 values of k (i.e. k=20, k=25, k=30… k=50). I chose not to include this graph as these new models showed me that k=20 indeed yields the lowest error score for this data. Every KNN model with a value of k greater than 20 ended up returning a higher error score than 20. The KNN models with values of k greater than 20 yielded error scores that leveled off 0.34. KNN ran pretty consistently for me overall.

For the boosting model, the tuning parameters that we tweaked were the shrinkage rate and the number of trees that were constructed. I initially ran a default shrinkage rate or learning rate with 5000 and 10000 trees. These models performed pretty well as they yielded error scores all around 0.185. However, Devansh ended up running boosting models with a larger variety of which out performed my models. His best model attained an error score of 0.12717, with 2000 trees. This turned out to be our best model overall. The graph for this boosting model and final tuning parameter choice can be found below:



This final boosting model that we went with ended up out performing our random tree models, which at one point we thought were going to be our most accurate models. However, there was only about a 0.01 difference between the boosting and random trees model error scores.

I also used GAM to help determine which variables our final model should include. The GAM run returned “Overall Quality of the Home”, “Year Built”, “Ground Floor Living Area” and “Lot Area” as being the most important variables to the final model, which seems fairly intuitive. Many, many variables appeared to be routinely unimportant to our model, including some rather obscure porch-related features, basement features, lot-boarding road features, and pool features. Overall, we didn’t run into too many issues implementing tests or models on our data. Most of the issues we interfaced with this project were related to processing the data and getting all columns are variables to a point where they were work-able with.