Kaggle Regression Competition Report

By: Bryan Kim

Introduction:

For the Regression Competition, we were given a training data set of 380 observations and 30 predictors (excluding ID's column) and a column "Wins", which is the output we were assigned to predict on the test data set.

Steps taken before fitting the model:

1. (Lines 8-9) First, I got rid of the ID's column as it is no use in predicting the number of wins.

2. (Lines 12-59) Then I created and added 21 new predictors to the train and test data sets. 15 of the 21 were created by dividing each predictor by their respective "Opp" predictor. The intuition behind creating these variables was to determine the relative performance of each team against their oppositions. Among the other 6 new predictors: "PassSuccess" and "OppPassSuccess" were created to measure how successfully the teams made a pass, and "RPassSuccess" was a ratio of the two. "YardSuccess" and "OppYardSuccess" were created to measure the average number of yards each team ran per offensive play, and "RYardSuccess" was a ratio of the two.

3. (Lines 62-77) Next, I ran a for loop with the "randomForest" function to find the top 30 predictors of "Wins". To do so I first set a seed so that the results can be replicable, then I initialized an empty vector (choose). I then ran the loop 100 times. In each run I sampled 340 among the 380 observations into a local sub-train data set, ran the "randomForest" function, called the importance element from our "randomForest" model, and obtained each predictor according to the importance ranking using the "%IncMSE" method. I then put all of the chosen top 30 predictors in the globally initialized vector: "choose". Once the for loop has run 100 times, I tallied how many times each predictor was chosen and ranked them based on the number of times they were picked as a "best 30 predictors" in the for loop. After running the for loop, the variable "Wins" seemed to have been erroneously picked by the randomForest() function as one of the top 30 predictors, so I removed it from my 30 predictors.

4. (Lines 79-84) Now I had a list of predictor names, but I needed to find the proper indices of these predictors within the "train" data frame. To do so, I initialized a vector "col_pre" then ran a for loop. In each loop I found the correct index of the desired predictor in the train data frame and added it to "col_pre". I then used this index "col_pre" and created a data frame "toppreds" containing only my desired predictors.

5. (Lines 89-102) Although I had 24 of the top predictors for "Wins", I checked the correlation plot of the predictors and found some predictors that were highly correlated, which would cause overfitting and multicollinearity. Therefore I decided to run the following algorithm on my chosen 24 predictors:

    - Outside the for loop, I first initialized a vector "badpreds" to find and store predictors to drop.

    - Inside the for loop I looked for any correlation between predictors that was higher than 0.88. I picked 0.88 since the R^2 and adjusted R^2 it yielded were satisfactory while training MSE was minimized, but also because I believed any correlation above 0.88 was too high and could lead to issues with multicollinearity.

- I then compared which of the two predictors had a higher ranking according to the random forest importance ranking and dropped the predictor with the lower rank. The intuition behind this step was to pick the predictors that were better according to random forest while keeping all correlations between the predictors lower than the chosen threshold of 0.88.

6. (Lines 103-111) Once the for loop was done, I had a vector "badpreds" full of indexes of the predictors I will drop from "toppreds". I then created a data frame of finalized predictors "finalpreds", and ran it again through a for loop (similar to the one in step 3), to find the indices of these final predictors in the original "train" data frame.

7. (Lines 113-125) Finally, I created a data frame including "Wins" as a numeric response, as well as the final predictors to use, then used the "lm" function to create a model with "Wins" as the response variable and all of the final variables as the predictors. I then outputted the predictions using the model.

Description of final model:

The model I used to predict my output was the multiple linear regression model: `m1 <- lm(Wins~., data=subdata1)`, where "Wins" was the column from train put into "subdata1", and "subdata1" was the data frame I created by including "Wins" and my final predictors. Since I used the lm() function to create my model, it is therefore parametric as the function lm() assumes a set of parameters. Moreover, it is less flexible of a model than other models such as random forest and support vector machines. The final model had a total of 27 predictors which were the following:

```
> predictors
 [1] "FirstDowns"              "InterceptionsThrown"     "OppPassesAttempted"      "OppPassSuccess"          "OppRushingAttempts"
 [6] "OppYardSuccess"          "PassSuccess"             "RFumblesLost"            "RInterceptionsThrown"    "ROffensivePlays"
[11] "RPassesAttempted"        "RPassesCompleted"        "RRushingAttempts"        "RTurnOversLost"          "RYardsGainedPassing"
[16] "RYardSuccess"            "TurnOversLost"           "OppTurnOversLost"        "RFirstDowns"             "RPenaltiesCommitedByTeam"
[21] "RushingAttempts"         "RYardsGainedRushing"     "OppInterceptionsThrown"  "FumblesLost"            "YardsGainedPassing"
[26] "OppYardsGainedRushing"   "PenaltiesCommitedByTeam"
```

The reason I decided to use the lm() function was because using a multiple linear regression was more interpretable. Moreover, since I chose to use 27 predictors of the 51 predictors (including the ones I created) using other more flexible models could have been more susceptible to overfitting.

Discussion:

One strength of this model is that I created 21 new feasible predictors to be considered and included some in my model. Another strengths of this model is that I not only found the "best" predictors using random forest but I also further removed any predictors that were highly correlated; however, I feel that using 0.88 as a threshold was too high and the predictors I ended up using would still have issues of multicollinearity. Also, another possible weaknesses to my model is that some predictors may have been better treated as polynomial and hence simply including them in the lm() function may not have been ideal. In addition, I could have further improved my model by using other tools such as subset selection to find better (and smaller) combinations of predictors to use. Finally, since the data set given had a higher number of observations than predictors, using a more flexible model may have been more appropriate. For model evaluation, I relied on training MSE, adjusted R^2, and R^2. I aimed to find the model with the minimum MSE while maximizing the adjusted R^2, and R^2; this was a somewhat foolish thing to do since the more predictors one has, the higher the R^2, which would have caused more overfitting. Therefore, also checking the MSE on a training and validation set using cross validation may have been better for the evaluation.

Overall, I've enjoyed this competition, but also learned a lot from it. If I could restart this competition, I would have used other evaluation methods such as cross validation and chosen my models based on the structure of the data that was given; for this competition $n > p$ and hence more flexible models would have been better suited.