Project Part One: UFC Fight Predictor

Cary Dean Turner and Tony Kim

10/21/2020

Part One: The Data

The dataset we are working with came from Kaggle.com and contains information on ~4400 UFC mixed martial arts fights. The data was originally scraped directly from ufcstats.com, leading us to believe that it is likely very reliable, professionally collected data.

In the dataset, each observation is one UFC fight between two fighters. By convention, in UFC fights, one fighter is designated "Red" and the other "Blue", and are identified as such by the color of their gloves. In each observation we have several statistics for each of the two fighters, such as their reach, height, weight, age, career average successful striking percentage, career average takedown attempts and percentage, total wins, total losses, total number of draws, total rounds fought in their career, the Las Vegas gambling odds for each fighter, the outcome of the match, and more.

Our choice of binary variable to predict was obvious: the outcome of the match. In the dataset there are only two possible outcomes: either the Red fighter wins or the Blue fighter wins (there are no draws in the dataset). Instead of using Blue or Red as the outcome, we created an indicator variable `red_win` that is set to 1 if the Red fighter won and 0 if the Blue fighter won. This choice was somewhat arbitrary, although by convention, the favored fighter is Red, so it made sense to us to make our outcome variable in terms of Red.

Our continuous variable of interest was somewhat of a less obvious choice, however. There are several continuous variables in the dataset (almost every statistic for each fighter is continuous), but ultimately the one which seemed the most interesting to us is the Las Vegas gambling odds assigned to the Red fighter.

The UFC (and MMA more broadly) is a notoriously difficult sport to predict. Not only is it an incredibly young professional sport, but just by its very nature is chaotic and unconventional. Put two world class fighters in a cage with nothing but 4oz gloves to protect themselves and anything can happen. If we are able to come up with some reliable results, we may very well be the first people to create an accurate predictor of mixed martial arts bouts. Additionally, if we are able to accurately predict odds of a win, we could potentially identify when the gambling odds over- or under-favor a particular.

Part Two: Cleaning

The original dataset is quite robust, containing 127 variables. However, many of these variables are missing for nearly all fights. For this reason, we had to pare down our covariates significantly. We also removed several variable which are recorded *during* the match, such as successful takedown attempts or significant strikes in the match. The reason for removing these variables is that we wanted to make a predictive algorithm that is *useful*. If our outcome predictor relies on using statistics that are collected within the match we are trying to predict the outcome of, then that means we can only predict the outcome after the match has already happened! This is obviously a useless predictor, and would not be of much interest to anybody. Other variables were removed simply because they were superfluous, and seemed completely irrelevant. Some of these variables included the date and location of the fight, as well as the gender and weight class (since any particular fight will be within one weight class and between two fighters of the same gender).

Part Three: Transformations and Exploration

Perhaps the most significant transformation of variables that we made was to take every statistic of each fighter and combine it into a single statistic capturing the difference between the fighters. For example, instead of looking at the height of the Red fighter and the height of the Blue fighter *separately*, we combine them into one variable by subtracting Blue fighter's height from the Red fighter's heigh and call it `R_height_adv` (Red height advantage). We repeated this process for every statistic, always using the same form: $Red_stat - Blue_stat$, so that everything is in terms of the Red fighter.

We also added additional variables such as the win-loss ratio for each fighter, calculated as $\frac{wins+1}{losses+1}$ (adding 1 to the number of losses to ensure we're not dividing by zero, and adding 1 to the number of wins to even it out). We also calculated for each fighter the proportion of their wins that were by KO or submission, as opposed to wins by judge's decision, and created ratios similar to the win-loss ratio above.

We stayed away from doing log transformations due to the fact that, because nearly all our predictors are in the form of $Red - Blue$, we have many negative and zero values. Note: We did first try logging the original variables (i.e., Red and Blue separately), and the logged versions had almost no association with either of our outcome variables, so we decided to nix the idea. We considered squaring our predictors and looking for associations with the outcome, but it didn't ultimately make much sense because so many of our variables have negative values which we want to stay negative. We instead decided to cube all our predictors, but alas, none of the cubic versions of the predictors have very strong associations with either of our outcomes. Ultimately, we decided to use the cubic terms, as well as two-way interactive terms, as additional predictors in our models, allowing the regularization methods to pick out the most important ones.

Through our analysis, we found that age difference, current win streak difference, win-loss ratio difference, and career losses difference to be the predictors most strongly correlated with our continuous outcome variable `R_odds`. We didn't find many strong associations with our binary outcome variable `red_win`, but the strongest are age difference, successful takedown difference, current win streak difference, and career losses difference.

Part Four: Predictions

Classification Model

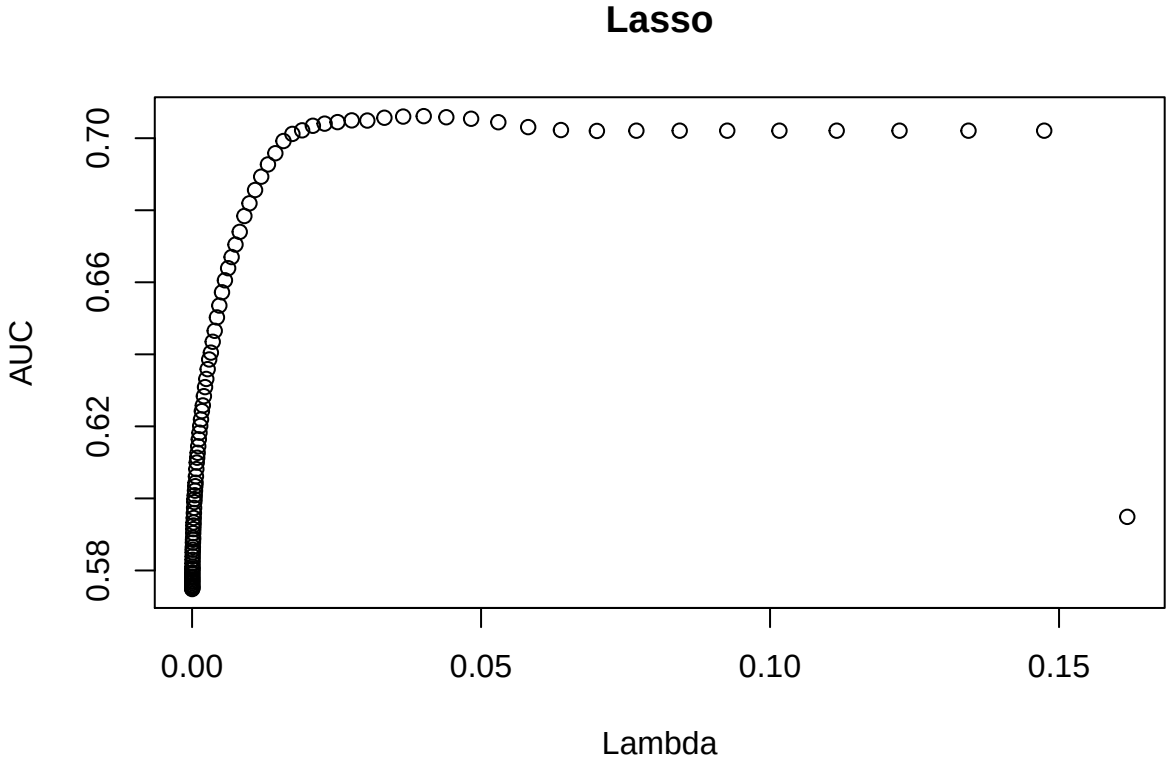
Since we are predicting the outcome of a fight (win or loss), there isn't any particular reason to treat Type I errors and type II errors differently. In other words, predicting a loss as a win isn't any worse than predicting a win as a loss. For this reason, maximizing classification accuracy seems to be the most reasonable objective function.

To determine a baseline prediction error to advance from, we simply calculated the percentage of matches which resulted in a win for the Red fighter, since any classifier could perform at least that well by predicting a win for Red every time. We found this baseline value to be 0.5753589. From this benchmark, we can say that any classifier which has a test accuracy greater than 58% is at least somewhat useful, but ideally we would like to shoot for test accuracy of 60% or greater. Of course, anything better than this would be amazing, but given the fact that MMA fights are notoriously unpredictable, I don't think it's likely we will achieve a very high accuracy.

In training our model, we opted to go with the "kitchen sink" method. i.e., throw in all our variables, transformed variables, and two way interactions, and let the regularization process suss out which variables are the most important. As for the type of regularization used, we trained both lasso and ridge models using 100 different values of lambda, and picked the value of lambda for each which maximized estimated AUC, where the AUC was estimated using 10-fold cross validation. After performing the cross-validation, we had two classification models: a ridge model and a lasso model. The lasso model slightly out-performed the ridge model, with a CV AUC of 0.70, versus 0.65 for the ridge model. Based on these cross-validation metrics, we are estimating that our model will have an AUC of ~0.70 when computed on our test set.

One interesting thing to note is that, when evaluating the models on the full training set, the ridge model actually outperformed the lasso model by almost every metric, even though the lasso performed better in cross-validation. Our theory to explain this is that because the ridge doesn't zero out any coefficients, it

likely has higher variance and could perhaps be overfitting to the training data. However, because the lasso model zeroed out nearly all of the ~600 predictors, it almost certainly has much lower variance, although almost certainly higher bias. In this case, it seems like the additional bias added by lasso was outweighed by the reduction in variance.




```
## # A tibble: 9 x 3
##   Metric               Lasso  Ridge
##   <chr>                <dbl>  <dbl>
## 1 Accuracy             0.650 0.666
## 2 0-1 Loss             0.350 0.334
## 3 Sensitivity          0.779 0.920
## 4 Specificity          0.476 0.322
## 5 Precision            0.668 0.648
## 6 Type I Error Rate    0.524 0.678
## 7 Type II Error Rate   0.221 0.0804
## 8 False Discovery Rate 0.332 0.352
## 9 Training AUC         0.706 0.738
```

Regression Model

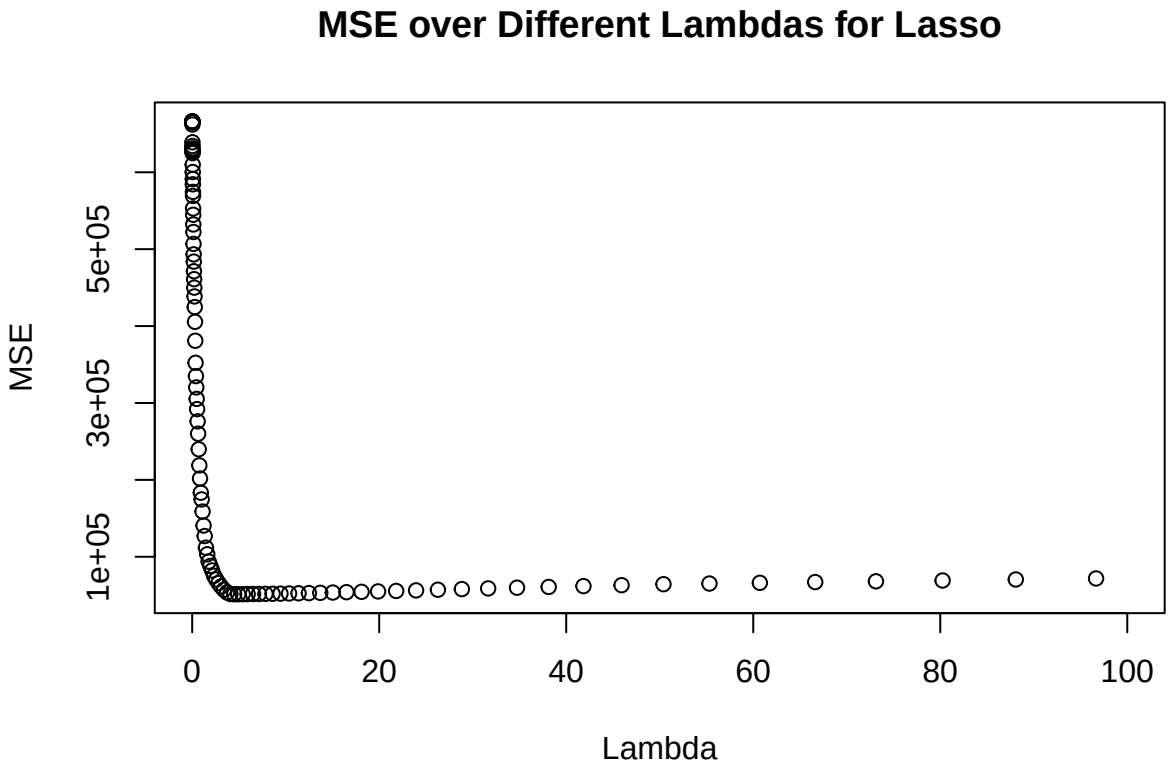
We tried various methods of regression to predict the red fighter's odds (R_odds), and our best model was a Lasso regression model that took in all columns except red_win as covariates (the red_win of the match would have given away the odds that we were trying to predict for in the first place). In addition to those columns, it took in all two-way interaction terms between columns and cubic transforms of numeric variables. This model is the most superior in terms of root mean squared error; whereas the baseline of guessing the sample mean for every datapoint had a training RMSE of 268 and the linear regression model without regularization and transforms gave a cross-validation RMSE of 228, this particular Lasso model is able to give the only cross-validation RMSE below 227 (the cross validations all used k=10 folds). It is not the biggest improvement, but the fact that an improvement exists makes sense because Lasso is able to cut terms that do not contribute much and to retain the terms that truly matter.

```
##                                                 Models     RMSE
## 1                             Baseline Training Error 268.0049
## 2      CV Error for Linear Regression w/o Transforms 228.0628
## 3       CV Error for Linear Regression w/ Transforms 311.1833
## 4 Training Error for Linear Regression w/ Transforms 219.4963
## 5 CV Error for Ridge w/o Transforms and Interactions 227.6853
## 6  CV Error for Ridge w/ Transforms and Interactions 256.7685
## 7 CV Error for Lasso w/o Transforms and Interactions 227.3848
## 8  CV Error for Lasso w/ Transforms and Interactions 226.4476
```

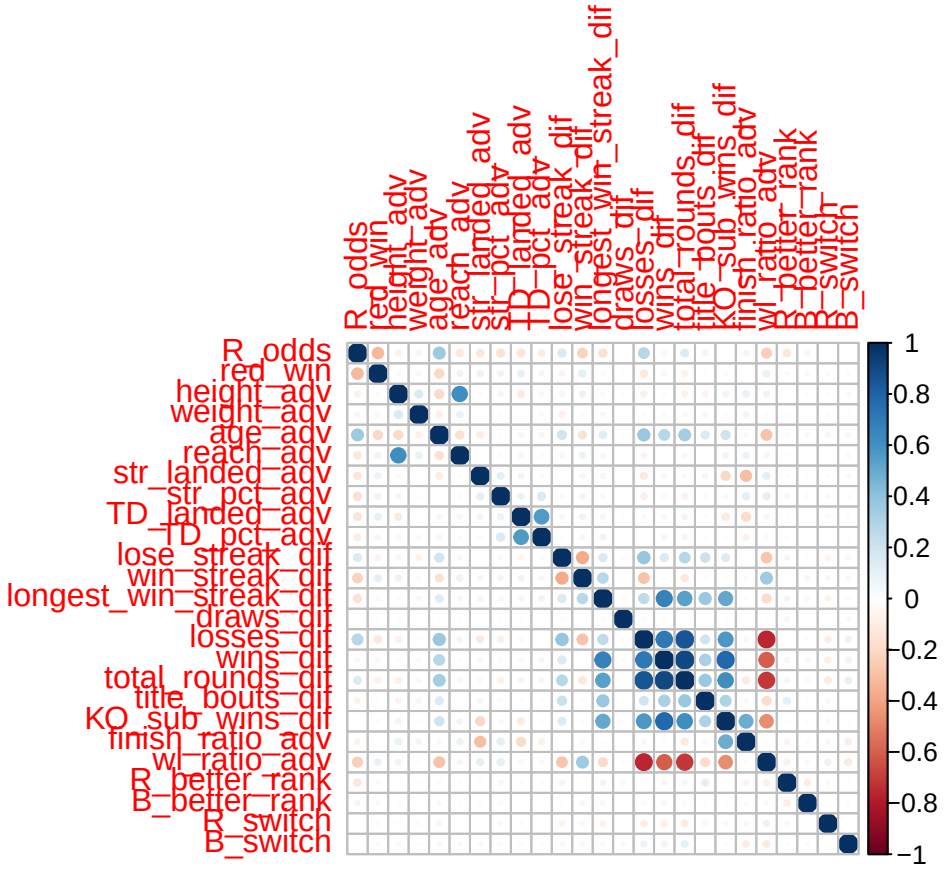
One particularly interesting observation to note is how Lasso is the only technique here that seems to do better with the inclusion of transform/interaction terms. This suggests that other methods are vulnerable to the overfitting that comes with the inclusion of more covariates, but Lasso in particular is able to cut out variables while retaining the particular cubic-transform or two-way-interaction terms that have value. In fact, when we observe the particular coefficients that Lasso kept, we could see the logic behind the model's retaining key interaction terms (ex. losses_dif:total_rounds_dif where the interaction between two fighters' loss record and how quickly they won or lost each match may give important information) and cubic transform terms (ex. height_adv^3; intuitively, a height advantage of 20 inches vs 10 inches would make such a huge difference in a fight that it cannot be explained by a mere linear relationship).

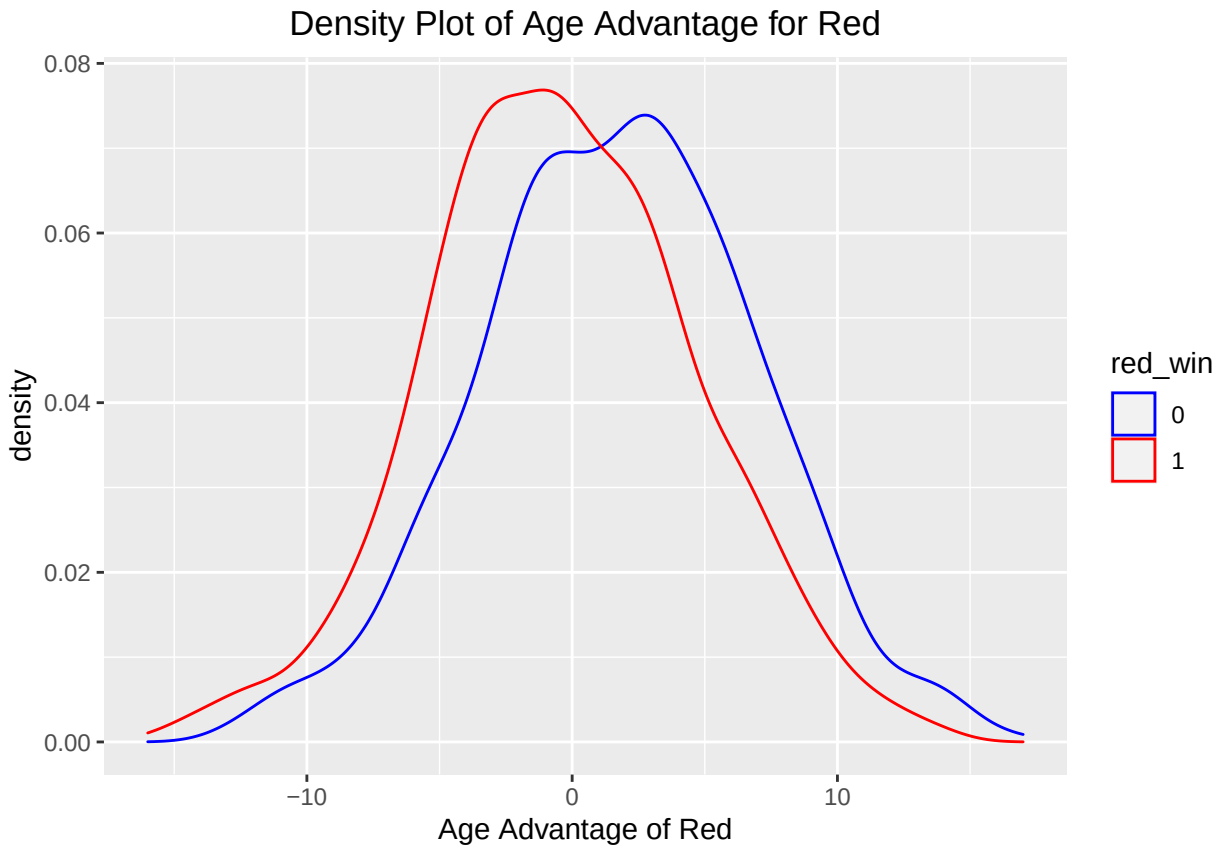
In terms of comparing these various models by bias and variance, we start out with the baseline, which gives a high bias and low variance. This is because the perpetual guessing of the sample mean would give values that are far off from the true individual values. At the same time, taking the sample mean would not differ too much given another dataset of a similarly hefty size. The linear regression model without any regularization, transforms, and interactions gives back significantly less bias with a better attempt at explaining the trend, but more variance than a baseline that constantly selects the same value. When we add on the cubic transforms to this non-regularized regression, the bias would decrease (with better fit to the training data), but the variance would increase due to overfitting. In fact, as we see in the table above, the validation error of this non-regularized model with cubic transforms skyrockets (showcasing poor generalization and big variance), while its low training error simultaneously shows that the model fit the training data very well. Both ridge and lasso add a layer of regularization that decreases overfitting and thus decreases variance, while potentially increasing bias due to the extra regularization term that limits optimization. At the same time, the fact that our best-performing model introduces cubic transforms and two-way interactions in addition to the regularization means that the model is able to cut down on this bias, since these more complex terms can potentially explain the data better.

In terms of a test RMSE estimate for predicting R_odds, there are two factors at play to consider. The first is that the validation errors are generally underestimates of the test error (as discussed in lecture) due to the process of model selection and the possibility that the particular data made the model look better. On the other hand, since our number of folds k=10 is a relatively small number, the cross validation could certainly overestimate the test error since the use of less data in each fold may introduce more bias. In the end, since there is a large number of covariates that may have made the model fit too closely to the training set, it is safer to assume that the test error will be slightly higher than our validation error. Thus, keeping all of these factors into consideration, we estimate the test RMSE at around 229 – about 2 points higher than the cross-validation error.

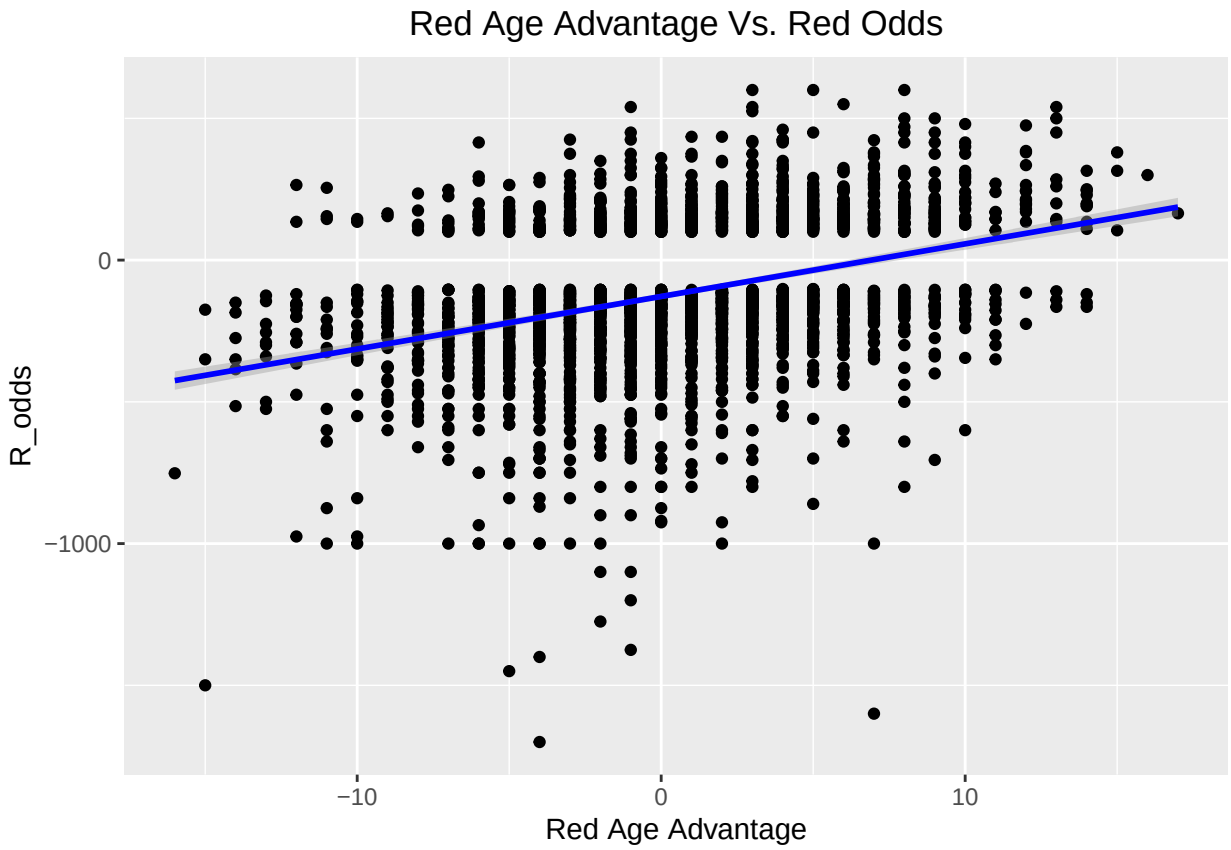



```
## corrplot 0.84 loaded
```

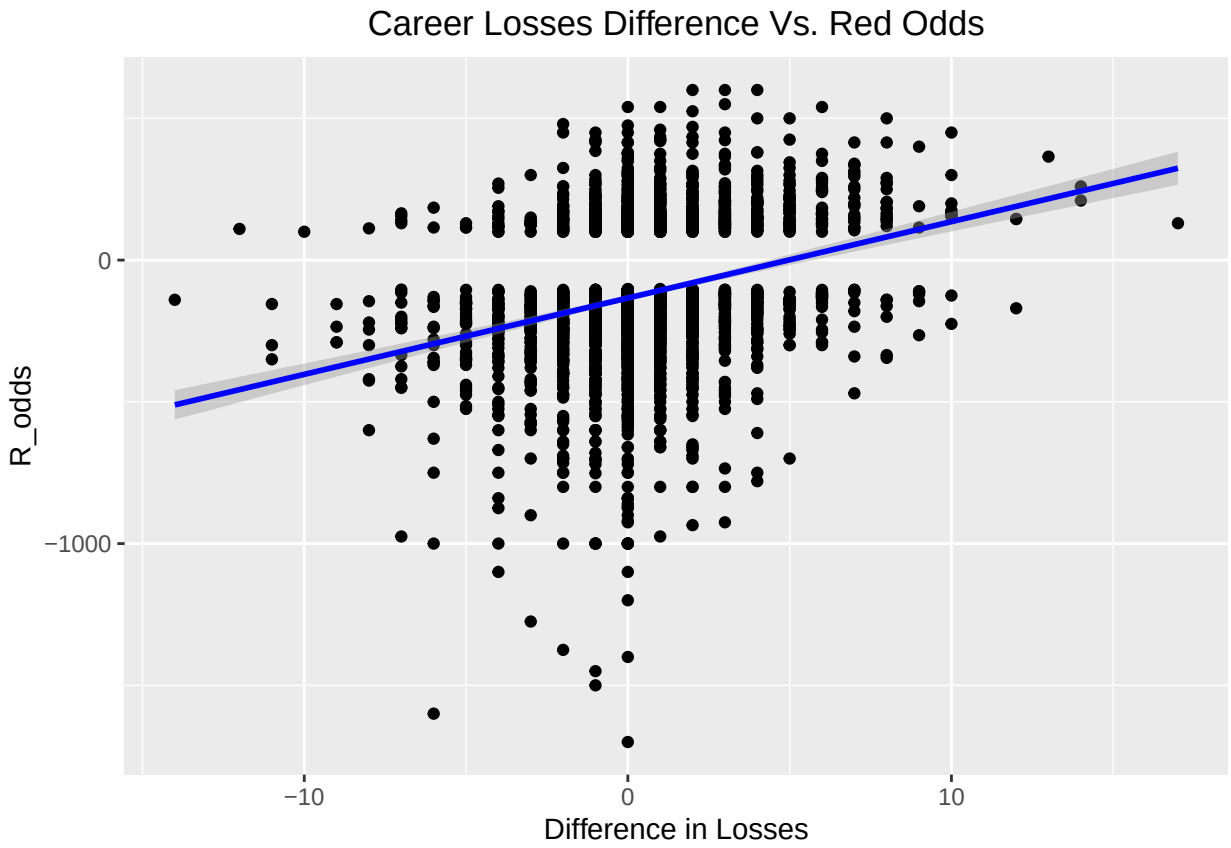





```
## `geom_smooth()` using formula 'y ~ x'
```




```
## `geom_smooth()` using formula 'y ~ x'
```


Career Losses Difference Vs. Red Odds
R_odds
0
−1000
−10
0
10
Difference in Losses

```
#####################################################
#                       HEAD                        #
#####################################################

# File Name: ufc_fight_predictor.R
# Authors: Cary Turner and Tony Kim
# Description: Data stuff
#####################################################

library(tidyverse)

master <- read.csv('ufc-master.csv')


# Create an indicator set to 1 if red fighter won and 0 otherwise.
# There were no draws in this data set.
master$red_win <- ifelse(master$Winner == 'Red', 1, 0)
names(master)


# Create a vector of all variables we want to keep
cols <- c(
  'R_odds',
  'B_odds',
  'title_bout',
  'weight_class',
  'no_of_rounds',
  'B_current_lose_streak',
  'B_current_win_streak',
  'B_draw',
  'B_longest_win_streak',
  'B_losses',
  'B_wins',
  'B_Stance',
  'B_total_rounds_fought',
  'B_total_title_bouts',
  'B_win_by_KO.TKO',
  'B_win_by_Submission',
  'B_Weight_lbs',
  'B_avg_SIG_STR_landed',
  'B_avg_SIG_STR_pct',
  'B_avg_SUB_ATT',
  'B_avg_TD_landed',
  'B_avg_TD_pct',
  'B_Height_cms',
  'B_Reach_cms',
  'B_age',
  'R_current_lose_streak',
  'R_current_win_streak',
  'R_draw',
  'R_longest_win_streak',
```

```
  'R_losses',
  'R_wins',
  'R_Stance',
  'R_total_rounds_fought',
  'R_total_title_bouts',
  'R_win_by_KO.TKO',
  'R_win_by_Submission',
  'R_Weight_lbs',
  'R_avg_SIG_STR_landed',
  'R_avg_SIG_STR_pct',
  'R_avg_SUB_ATT',
  'R_avg_TD_landed',
  'R_avg_TD_pct',
  'R_Height_cms',
  'R_Reach_cms',
  'R_age',
  'better_rank',
  'finish_round',
  'total_fight_time_secs',
  'red_win'
)



# Update data frame to include only relevant variables
fights <- master[,cols]

# Get rid of NA values
fights <- na.omit(fights)


# Subtract Blue height, weight, reach, and age from Red's
# to get Red's height, weight, reach, and age advantage
fights$height_adv <- fights$R_Height_cms - fights$B_Height_cms
fights$weight_adv <- fights$R_Weight_lbs - fights$B_Weight_lbs
fights$reach_adv <- fights$R_Reach_cms - fights$B_Reach_cms
fights$age_adv <- fights$R_age - fights$B_age

# Do same for avg strikes, and takedowns landed/percentage, and for submission
 attempts
fights$str_landed_adv <- fights$R_avg_SIG_STR_landed -
 fights$B_avg_SIG_STR_landed
fights$str_pct_adv <- fights$R_avg_SIG_STR_pct - fights$B_avg_SIG_STR_pct
fights$TD_landed_adv <- fights$R_avg_TD_landed - fights$B_avg_TD_landed
fights$TD_pct_adv <- fights$R_avg_TD_pct - fights$B_avg_TD_pct
fights$sub_att_adv <- fights$R_avg_SUB_ATT - fights$B_avg_SUB_ATT

fights$lose_streak_dif <- fights$R_current_lose_streak -
 fights$B_current_lose_streak
fights$win_streak_dif <- fights$R_current_win_streak -
 fights$B_current_win_streak
```

```r
fights$longest_win_streak_dif <- fights$R_longest_win_streak -
 fights$B_longest_win_streak
fights$draws_dif <- fights$R_draw - fights$B_draw
fights$losses_dif <- fights$R_losses - fights$B_losses
fights$wins_dif <- fights$R_wins - fights$B_wins
fights$total_rounds_dif <- fights$R_total_rounds_fought -
 fights$B_total_rounds_fought
fights$title_bouts_dif <- fights$R_total_title_bouts -
 fights$B_total_title_bouts

# Determine number of wins by KO/TKO or submission for each fighter
fights$R_wins_by_KO_sub <- fights$R_win_by_KO.TKO + fights$R_win_by_Submission
fights$B_wins_by_KO_sub <- fights$B_win_by_KO.TKO + fights$B_win_by_Submission

# Determine number of wins by decision for each fighter.
# Wins by decision = wins - wins by KO/TKO or submission
fights$R_wins_by_dec <- fights$R_wins - fights$R_wins_by_KO_sub
fights$B_wins_by_dec <- fights$B_wins - fights$B_wins_by_KO_sub

# Get rid of nonsensical negative values
fights <- fights %>%
  filter(R_wins_by_dec >= 0, B_wins_by_dec >= 0)

# Calculate the ratio of wins by KO/TKO and submission (finishes) to wins by
 decision.
# We add one to each to avoid zero division.
fights$R_finish_dec_ratio <- (fights$R_wins_by_KO_sub + 1) /
 (fights$R_wins_by_dec + 1)
fights$B_finish_dec_ratio <- (fights$B_wins_by_KO_sub + 1) /
 (fights$B_wins_by_dec + 1)

# Calculate overall win loss ratio for each fighter. Add one again to avoid
 zero division.
fights$R_wl_ratio <- (fights$R_wins + 1) / (fights$R_losses + 1)
fights$B_wl_ratio <- (fights$R_wins + 1) / (fights$B_losses + 1)

# Difference between number of KO/submission finishes
fights$KO_sub_wins_dif <- fights$R_wins_by_KO_sub - fights$B_wins_by_KO_sub

# Finish ratio difference between fighters
fights$finish_ratio_adv <- fights$R_finish_dec_ratio -
 fights$B_finish_dec_ratio

# Win-loss ratio difference between fighters
fights$wl_ratio_adv <- fights$R_wl_ratio - fights$B_wl_ratio

# Create variables for different stances of the fighters
fights$R_switch <- ifelse(fights$R_Stance == 'Switch', 1, 0)
fights$B_switch <- ifelse(fights$B_Stance == 'Switch', 1, 0)

# Create indicator for each fighter's relative ranking
```

```r
fights$R_better_rank <- ifelse(fights$better_rank == 'Red', 1, 0)
fights$B_better_rank <- ifelse(fights$better_rank == 'Blue',  1, 0)



# Create vector with names of all variables we are keeping for analysis
keep <- c(
  'R_odds',
  'red_win',
  'title_bout',
  'weight_class',
  'height_adv',
  'weight_adv',
  'age_adv',
  'reach_adv',
  'str_landed_adv',
  'str_pct_adv',
  'TD_landed_adv',
  'TD_pct_adv',
  'lose_streak_dif',
  'win_streak_dif',
  'longest_win_streak_dif',
  'draws_dif',
  'losses_dif',
  'wins_dif',
  'total_rounds_dif',
  'title_bouts_dif',
  'KO_sub_wins_dif',
  'finish_ratio_adv',
  'wl_ratio_adv',
  'R_better_rank',
  'B_better_rank',
  'R_switch',
  'B_switch'
)


############### Split Data Into Test and Train Sets ###################

# Split into train and test sets, using 80/20 split
set.seed(138)
in.test <- sample(nrow(fights), nrow(fights)/5)

# Sanity check
length(in.test)
dim(fights)[1]*0.2

# Assign train and test sets
test <- fights[in.test,keep]
train <- fights[-in.test,keep]
```

```
############## Initial Data Exploration ###############

# Indices of non-numeric variables
non_numeric <- c(3,4)

# cube predictors (would square them but we want negatives to stay negative)
cubed.vars <- train[,-c(2, 3, 4, 24, 25, 26, 27)]^3
cubed.vars$red_win <- train$red_win

# Check for correlation between variables and outputs
library(corrplot)
C <- cor(train[,-non_numeric])
corrplot(C, method = 'circle')
C

# Check for correlation between cubic vars and outputs
C.cubic <- cor(cubed.vars)
corrplot(C.cubic, method = 'circle')

colors <- c('1' = 'red', '0' = 'blue')

# Plot the density of all variables, separated by cases where red wins vs. blue
 wins
for (var in names(train[,-non_numeric])) {
  title <- cat('Density of',var)
  plot <- ggplot(train) +
    geom_density(aes(x = train[,var], color = as.factor(red_win))) +
    scale_color_manual(values = colors) +
    labs(x = var, color = 'red_win', title = title) +
    theme(plot.title = element_text(hjust = 0.5))
  print(plot)
  Sys.sleep(2)
}

# Scatter plot of each var vs. R_odds
for (var in names(train[,-non_numeric])) {
  title <- cat('Red Odds Vs.',var)
  plot <- ggplot(train, aes(x = train[,var], y = train$R_odds)) +
    geom_point() +
    geom_smooth(method = 'lm', color = 'blue', se = FALSE) +
    labs(x = var, y = 'R_odds', title = title) +
    theme(plot.title = element_text(hjust = 0.5))
  print(plot)
  Sys.sleep(2)
}



############### Predictive Modeling #################
```

```
library(ROCR)
library(glmnet)
library(cvTools)
library(boot)

# Function to compute the AUC of a binary classifier
compute_auc <- function(p, labels) {
  pred <- prediction(p, labels)
  auc <- performance(pred, 'auc')
  auc <- unlist(slot(auc, 'y.values'))
  auc
}

# Function to plot the ROC curve of a binary classifier
plot_roc <- function(p, labels, model_name) {
  pred <- prediction(p, labels)
  perf <- performance(pred,"tpr","fpr")
  plot(perf, col="black", main = paste("ROC for", model_name))
}

# Calculates accuracy of our logistic regression classifier
# to be used when running cross-validation
cost <- function(r, pi) mean(abs(r-pi) < 0.5)

# Build logistic regression model using all variables. This is our base
# model that we will judge other models against.
glm.base <- glm(red_win ~ ., data = train, family = 'binomial')

# Compute cross-validation error using the base model
base.err <- cv.glm(train, glm.base, cost, K = 10)
base.err[3]

glm.base.prob <- predict(glm.base, train, type='response')
glm.base.pred <- rep(0, length(train$red_win))
glm.base.pred[(glm.base.prob > 0.55)] <- 1
mean(glm.base.pred == train$red_win)

# Compute AUC on training set
base.train.auc <- compute_auc(predict(glm.base, data = train, type =
 'response'), train$red_win)
base.train.auc



################## Ridge and Lasso Models for Binary Outcome
 ####################

### Predicting wins/losses ###

# Remove binary outcome variable (red_win) from cubed.vars data frame
cubed.vars <- cubed.vars[,-21]
```

```
# Change names of all columns to reflect that they are cubed.
cubed.cols <- rep('', dim(cubed.vars)[2])
for (i in 1:length(names(cubed.vars))) {
  cubed.cols[i] <- paste(names(cubed.vars)[i], '^3', sep='')
}


# Create model matrix with all cubed terms and all two way interactions
form <- red_win ~ . + .^2
x <- model.matrix(form, data=train)

# Use column bind to add all cubed variables and our outcome variable (red_win)
x <- cbind(train$red_win, x, cubed.vars)
x <- x[,-2]
colnames(x)[colnames(x) == 'train$red_win'] <- 'red_win'

# Recast it as a model matrix
x.train <- model.matrix(red_win ~ ., x)

# Create vector of our outcome variable
y.train <- train$red_win

### Ridge Regression
ridge.lr <- cv.glmnet(x.train, y.train, type.measure='auc', alpha=0,
                      family='binomial', standardize=TRUE, seed=1)
plot(ridge.lr$lambda, ridge.lr$cvm, xlab='Lambda', ylab='AUC', main='Ridge')
ridge.lambda <- ridge.lr$lambda.min
ridge.auc <- max(ridge.lr$cvm)

# Make predictions on full training set (using lambda that maximizes AUC)
ridge.lr.pred <- predict(ridge.lr, s=ridge.lr$lambda.min, newx=x.train,
 type='response')

# Plot ROC curve for model using AUC-maximizing lambda
plot_roc(ridge.lr.pred, train$red_win, paste('Ridge (Lambda =
 ',ridge.lr$lambda.min,')', sep=''))
ridge.train.auc <- compute_auc(ridge.lr.pred, train$red_win)
ridge.train.auc

# Transform probabilities into actual predicted outcomes, using 0.55 as our
 threshold
ridge.lr.pred <- ifelse(ridge.lr.pred > 0.5, 1, 0)

# Create confusion matrix
ridge.conf <- table(ridge.lr.pred, train$red_win)
ridge.conf

# Calculate classification metrics
TP <- ridge.conf[2,2]
TN <- ridge.conf[1,1]
```

```
FP <- ridge.conf[2,1]
FN <- ridge.conf[1,2]
n <- TP + TN + FP + FN
accuracy <- (TP + TN) / n
zo.loss <- (FP + FN) / n
TPR <- TP / (FN + TP)
FPR <- FP / (TN + FP)
TNR <- TN / (TN + FP)
FNR <- FN / (FN + TP)
precision <- TP / (TP + FP)
false.discovery <- FP / (TP + FP)

Ridge <- c(accuracy, zo.loss, TPR, TNR, precision, FPR, FNR, false.discovery,
 ridge.train.auc)



### Lasso
lasso.lr <- cv.glmnet(x.train, y.train, type.measure='auc', alpha=1,
                      family='binomial', standardize=TRUE, seed=1)
plot(lasso.lr$lambda, lasso.lr$cvm, xlab='Lambda', ylab='AUC', main='Lasso')
lasso.lambda <- lasso.lr$lambda.min
lasso.auc <- max(lasso.lr$cvm)

# Make predictions on full training set (using AUC-maximizing lambda)
lasso.lr.pred <- predict(lasso.lr, s=lasso.lr$lambda.min, newx=x.train,
 type='response')

# Plot ROC curve for Lasso model using AUC-maximizing lambda
plot_roc(lasso.lr.pred, train$red_win, paste('Lasso (Lambda =
 ',lasso.lr$lambda.min,')', sep=''))
lasso.train.auc <- compute_auc(lasso.lr.pred, train$red_win)
lasso.train.auc

# Transform probabilities into actual predicted outcomes, using 0.55 as our
 threshold
lasso.lr.pred <- ifelse(lasso.lr.pred > 0.5, 1, 0)

# Create confusion matrix
lasso.conf <- table(lasso.lr.pred, train$red_win)
lasso.conf

# Compute classification metrics
TP <- lasso.conf[2,2]
TN <- lasso.conf[1,1]
FP <- lasso.conf[2,1]
FN <- lasso.conf[1,2]
n <- TP + TN + FP + FN
accuracy <- (TP + TN) / n
zo.loss <- (FP + FN) / n
TPR <- TP / (FN + TP)
```

```
FPR <- FP / (TN + FP)
TNR <- TN / (TN + FP)
FNR <- FN / (FN + TP)
precision <- TP / (TP + FP)
false.discovery <- FP / (TP + FP)

Lasso <- c(accuracy, zo.loss, TPR, TNR, precision, FPR, FNR, false.discovery,
 lasso.train.auc)
Metric <- c('Accuracy', '0-1 Loss', 'Sensitivity', 'Specificity', 'Precision',
            'Type I Error Rate', 'Type II Error Rate', 'False Discovery Rate',
             'Training AUC')

# Create data frame containing all metrics for both models
model.metrics.train <- tibble(Metric, Lasso, Ridge)
model.metrics.train

# Coefficients for both models
coef(lasso.lr, s='lambda.min')
coef(ridge.lr, s='lambda.min')


################### Predicting Fighter Odds ######################

# take away red_win column because it would give what we're predicting for away
train_odds <- select(train, -(red_win))
test_odds <- select(test, -(red_win))

# baseline RMSE
rmse_baseline <- sqrt(mean((train_odds$R_odds - mean(train_odds$R_odds)) ^ 2))
print(paste("Baseline RMSE of always predicting sample mean:",
            rmse_baseline))

# make the non-numeric weight_class a categorical variable
train_odds$weight_class <- factor(train_odds$weight_class)
test_odds$weight_class <- factor(test_odds$weight_class)

# fit vanilla regression model
fit <- lm(R_odds ~ ., data=train_odds)
val_error_wo_transforms <-
  cvFit(fit, data=train_odds, y=train_odds$R_odds, K=10, seed=1)$cv
val_error_wo_transforms

# divide train data into X and Y
# will be useful for Ridge and Lasso especially
X.train_odds <- select(train_odds, -(R_odds))
Y.train_odds <- train_odds$R_odds
# cubic transform on all numeric variables
cubed_X.train_odds <-
  select(X.train_odds,
         -c(title_bout, weight_class, R_better_rank, B_better_rank,
            R_switch, B_switch))^3
```

```
# rename cubed column names to avoid confusion
new_col_names <- rep(NA, ncol(cubed_X.train_odds))
for (i in 1:length(new_col_names)) {
  new_col_names[i] <- paste(colnames(cubed_X.train_odds)[i], '^3', sep='')
}
colnames(cubed_X.train_odds) <- new_col_names

# column-bind cube-transformed data to predict
transformed_data_odds <- cbind(Y.train_odds, X.train_odds, cubed_X.train_odds)
fit_transform <- lm(Y.train_odds ~ ., data=transformed_data_odds)
val_error_w_transforms <-
  cvFit(fit_transform, data=transformed_data_odds,
        y=transformed_data_odds$Y.train_odds, K=10, seed=1)$cv
# error is going way up with cubic transform; it might be overfitting
val_error_w_transforms

# so let's predict on training set and measure training error
pred <- predict(fit_transform, transformed_data_odds)
rmse_transform_train <- sqrt(mean((transformed_data_odds$Y.train_odds
                            -pred) ^ 2))
# in fact, training error is very low
print(paste("Training RMSE of transformed data:",
            rmse_transform_train))

# shoots way up for interaction variables; on top of that, getting additional
# error saying invalid rank for using cvFit

fit_inter <- lm(R_odds ~ . + .:., data=train_odds)
val_error_inter <-
  cvFit(fit_inter, data=train_odds, y=train_odds$R_odds, K=5, seed=1)$cv
val_error_inter


# Create model matrix with all two way interactions
form_inter <- R_odds ~ . + .^2
X.train_odds_matrix_inter <- model.matrix(form_inter, data=train_odds)
# column-bind cubic terms to the interaction terms
data_w_cubes_inter <- cbind(X.train_odds_matrix_inter, cubed_X.train_odds,
 Y.train_odds)
# ultimate model matrix with interactions and transforms
X.train_odds_matrix_inter <- model.matrix(Y.train_odds ~ .,
 data=data_w_cubes_inter)

# 10 fold cross validation for Ridge with all interactions and transforms
fm.ridge_inter <- cv.glmnet(X.train_odds_matrix_inter,
                            Y.train_odds, type.measure='mse',
                            alpha = 0, seed=1, standardized=TRUE)
# Value of lambda that gives minimum MSE
fm.ridge_inter$lambda.min
i <- which(fm.ridge_inter$lambda == fm.ridge_inter$lambda.min)
# minimum RMSE
```

```
rmse_ridge_inter <- sqrt(fm.ridge_inter$cvm[i])
print(rmse_ridge_inter)

# 10 fold cross validation for Ridge w/o interactions and transforms
form <- R_odds ~ .
X.train_odds_matrix <- model.matrix(form, data=train_odds)
fm.ridge <- cv.glmnet(X.train_odds_matrix, Y.train_odds, type.measure='mse',
                      alpha = 0, seed=1, standardize=TRUE)
# Value of lambda that gives minimum MSE
fm.ridge$lambda.min
plot(fm.ridge$lambda, fm.ridge$cvm, xlab='Lambda', ylab='MSE', main='Ridge',
 xlim=c(0,1000))
i <- which(fm.ridge$lambda == fm.ridge$lambda.min)
# minimum RMSE
rmse_ridge_wo_inter <- sqrt(fm.ridge$cvm[i])
print(rmse_ridge_wo_inter)

# 10 fold cross validation for Lasso with all interactions and transforms
fm.lasso_inter <- cv.glmnet(X.train_odds_matrix_inter,
                            Y.train_odds, type.measure='mse',
                            alpha = 1, seed=1, standardize=TRUE)
# Value of lambda that gives minimum MSE
fm.lasso_inter$lambda.min
i <- which(fm.lasso_inter$lambda == fm.lasso_inter$lambda.min)
rmse_lasso_inter <- sqrt(fm.lasso_inter$cvm[i])
# minimum RMSE
print(rmse_lasso_inter)
plot(fm.lasso_inter$lambda, fm.lasso_inter$cvm, xlab='Lambda', ylab='MSE',
     main='MSE over Different Lambdas for Lasso', xlim=c(0,100))
# lowest RMSE

# 10 fold cross validation for Lasso w/o interactions and transforms
fm.lasso <- cv.glmnet(X.train_odds_matrix, Y.train_odds, type.measure='mse',
                      alpha = 1, seed=1, standardize=TRUE)
# Value of lambda that gives minimum MSE
fm.lasso$lambda.min
i <- which(fm.lasso$lambda == fm.lasso$lambda.min)
# minimum RMSE
rmse_lasso_wo_inter <- sqrt(fm.lasso$cvm[i])
print(rmse_lasso_wo_inter)

coef(fm.lasso_inter, s='lambda.min')
coef(fm.lasso, s='lambda.min')
coef(fm.ridge_inter, s='lambda.min')
coef(fm.ridge, s='lambda.min')

# create table of all RMSE
RMSE <- c(rmse_baseline, val_error_wo_transforms, val_error_w_transforms,
                 rmse_transform_train, rmse_ridge_wo_inter, rmse_ridge_inter,
                 rmse_lasso_wo_inter, rmse_lasso_inter)
Models <- c("Baseline Training Error",
```

```
            "CV Error for Linear Regression w/o Transforms",
            "CV Error for Linear Regression w/ Transforms",
            "Training Error for Linear Regression w/ Transforms",
            "CV Error for Ridge w/o Transforms and Interactions",
            "CV Error for Ridge w/ Transforms and Interactions",
            "CV Error for Lasso w/o Transforms and Interactions",
            "CV Error for Lasso w/ Transforms and Interactions")
rmse_errors_df <- data.frame(Models, RMSE)
rmse_errors_df
```