Predicting College Football Wins

Michael Omelchenko

12/4/2020

Abstract

Machine learning algorithms and statistics have vastly changed the way sports are played and coached. The development of sabermetrics, which was used by Billy Beane to select a team for the Oakland Athletics[2], revolutionized the way statistics were used in sports. Since then, statistical analysis has become mainstream across all major sports.

In this project, we examined season statistics associated with college football. In particular, we used data from the 2019 college football season to predict the win percentage of a team and examined the importance of the predictors on the model.

Being able to predict a teams success, and the importance of the variables leading to their success, is important because it allows coaches and players to determine which areas they need to focus on most. Furthermore, in college football players are recruited to play for a college from high school. Determining which predictors are most important will allow coaches to focus more of their attention on areas where these recruits shine, ultimately improving their teams chance for success in the future.

Using some of the machine learning techniques learned throughout the class, such as Linear Regression, Regularization, Random Forest, and Knn, we found an ensemble Root Mean Squared Error (RMSE) of 0.058 on the test set. We were also able to determine that in both the linear regression model and the random forest model defensive statistics appeared to be more important than offensive statistics.

Introduction

To predict the number of wins a college football team has based on their statistics, we will be using a dataset from Jeff Gallini that was posted to Kaggle[4]. First, we will visualize and clean the data. Next we will use predictors such as Yards/Play Allowed, Pass Yards/Game, and others to train our machine learning algorithm on a portion of the dataset (the training set). Then, we will use the remaining portion of the partitioned dataset (the test set) to evaluate the performance of our machine learning algorithm. In order to evaluate our algorithms, we will be using the Root Mean Squared Error (RMSE) for comparison. Various machine learning algorithms such as linear regression, knn, random forest, and regularization will be used. These methods will be combined into an equally weighted ensemble that will be used for the final prediction.

Methods/Analysis

This section will highlight techniques and analysis used throughout our project. This includes downloading the data, data cleaning, data visualization, and machine learning models.

Downloading Data

To start, we need to load all of the libraries that will be used throughout this project. The readr and RCurl package will be used to download the csv file from a GitHub repository. The tidyverse package will be used for data wrangling and manipulation. The stringr package will be used for parsing and detecting strings using a regex pattern. The caret package will be used for the various machine learning techniques mentioned previously. Finally, the knitr, kableExtra, and data.table packages will be used to make formatted tables. The packages will be installed if not previously installed by the user before loading the libraries.

```r
#installing required packages and loading libraries
if(!require(readr)) install.packages("readr", repos = "http://cran.us.r-project.org")
if(!require(RCurl)) install.packages("RCurl", repos = "http://cran.us.r-project.org")
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(stringr)) install.packages("stringr", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(knitr)) install.packages("knitr", repos = "http://cran.us.r-project.org")
if(!require(kableExtra)) install.packages("kableExtra", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")


library(readr)
library(RCurl)
library(tidyverse)
library(stringr)
library(caret)
library(knitr)
library(kableExtra)
library(data.table)
```

Next we need to download the data. This data was provided by Kaggle user Jeff Gallini[4], but was uploaded to a GitHub repository with the files for this analysis and to ensure a backup of the data in case it was taken down. Downloading the data from the repository and inspecting it shows that the data has 130 entries with 146 columns.

```r
#downloading dataset
url <- getURL("https://raw.githubusercontent.com/mjchenko/college_football/main/CFB2019.csv")
cfb_data <- read_csv(file = url, skip = 0, col_names = TRUE)

tab <- data_frame("Number of Rows" = nrow(cfb_data),
                  "Number of Columns" = ncol(cfb_data))

kable(tab, caption = "Size of Dataset", booktabs = T,
      linesep = "", digits = 3) %>% kable_styling(latex_options = c("striped", "hold_position"))
```

Table 1: Size of Dataset

Number of Rows	Number of Columns
130	146

Cleaning Data

The first thing we notice is that the column Team, has the team name but also the conference in parenthesis. Also, we notice that some of these columns are very similar, such as Offensive Yards and Offensive Yards per Play. Furthermore, some of the columns have not been standardized for comparison; instead they are season totals. In order to do the comparison we will need to select stats that are standardized, such as per game or per attempt stats. We will begin our data cleaning with these steps.

As mentioned above, the first column has both the team and the conference. All but two entries follow this general format: TeamName (ConferenceName). Therefore, we will use str_split to separate at the parenthesis and then fix the two entries that do not follow this pattern. We will also convert the conference into a factor since each team belongs to a particular conference.

```r
#two entries do not follow the pattern.  Miami (OH) (MAC) and Miami (FL) (ACC)
#adding conference to the data
conf <- str_split(cfb_data$Team, pattern = "[()]", simplify = TRUE)[,2]
conf <- str_replace(conf, pattern = "FL", replacement = "ACC")
conf <- str_replace(conf, pattern = "OH", replacement = "MAC")
cfb_data <- cfb_data %>% add_column(.after = 'Team', conf = conf)
cfb_data$conf <- cfb_data$conf <- as.factor(cfb_data$conf)

#splitting first entry to only be the team name
teams <- str_trim(str_split(cfb_data$Team, pattern = "[\\(]", simplify = TRUE)[,1], side = "right")
#two entries are both Miami so we need to differentiate
teams[58:59] <- c("Miami (FL)", "Miami (OH)")
#mutate the team names to get rid of conference
cfb_data <- cfb_data %>% mutate(Team = teams)
```

There is also a column that has both wins and losses in the same column separated by a dash (W-L). Again, we will use the stringr library to split the data into two separate columns and then transform these character strings into numeric form. Since we have already split the columns into their own separate columns, we can remove the original W-L column. Because every team does not play the same number of games, we will create a win percentage column to account for the differences in games played. This column will ultimately be what we are trying to predict.

```r
#splitting W-L column into two difference columns
wins <- str_split(cfb_data$`Win-Loss`, pattern = "-", simplify = TRUE)[,1]
wins <- as.numeric(wins)
losses <- str_split(cfb_data$`Win-Loss`, pattern = "-", simplify = TRUE)[,2]
losses <- as.numeric(losses)
cfb_data <- cfb_data %>%
  add_column(.after = 'conf', wins = wins, losses = losses) %>%
  select(-'Win-Loss')

#creating the win percentage column
cfb_data <- cfb_data %>% mutate(win_perc = wins/Games)
```

Examining the data we see that there are a lot of columns that have Ranked statistics (for example the Offensive Rank). These columns will not aid in our machine learning algorithm because they are based on totals (total yards for Offensive Rank). This is not a good comparison because the number of games each team plays is not the same due to conference and scheduling differences. Therefore, we will remove these columns from our dataset.

```r
#removing columns that contain the pattern rank
ind <- str_detect(colnames(cfb_data), pattern = "Rank")
cfb_data <- cfb_data[!ind]
```

As mentioned earlier, we need to select statistics that are standardized such as per game, per attempt, etc. This way differences in number of games played gets taken into account. Below we select (or in some cases create) these statistics for our analysis. We also rename the columns so that they are consistent and easier to use.

```r
#mutating to per game or per attempt stats
cfb_data <- cfb_data %>%
    mutate(int_per_attempt = `Interceptions Thrown.y`/`Pass Attempts`,
        pass_attempt_per_game = `Pass Attempts`/Games,
        pass_completion_per_game = `Pass Completions`/Games,
        pass_attempt_per_game_allowed = `Opp Pass Attempts`/Games,
        pass_completion_per_game_allowed = `Opp Completions Allowed`/Games,
        rush_td_per_game = `Rushing TD`/Games,
        rush_td_per_game_allowed = `Opp Rush Touchdowns Allowed`/Games,
        pass_td_per_game = `Pass Touchdowns`/Games,
        pass_td_per_game_allowed = `Opp Pass TDs Allowed`/Games,
        ko_return_td_per_game =`Kickoff Return Touchdowns`/Games,
        ko_return_td_per_game_allowed = `Opp Kickoff Return Touchdowns Allowed`/Games,
        fmb_rec_per_game = `Fumbles Recovered`/Games,
        fmb_loss_per_game = `Fumbles Lost`/Games
         )

#selecting stats we want to keep
cfb_data <- cfb_data %>%
  select(Team,
        conf,
        win_perc,
        wins,
        losses,
        off_yards_per_play = `Off Yards/Play`,
        yards_per_play_allowed = `Yards/Play Allowed`,
        off_yards_per_game = `Off Yards per Game`,
        yards_per_game_allowed = `Yards Per Game Allowed`,
        percent_conv_4th = `4th Percent`,
        percent_conv_4th_allowed = `Opponent 4th Percent`,
        yards_per_kickoff_return_allowed = `Avg Yards per Kickoff Return Allowed`,
        yards_per_kickoff_return = `Avg Yard per Kickoff Return`,
        pass_yards_per_game = `Pass Yards Per Game`,
        pass_yards_per_attempt = `Pass Yards/Attempt`,
        pass_yards_per_completion = `Yards/Completion`,
        pass_yards_per_game_allowed = `Pass Yards Per Game Allowed`,
        penalty_yards_per_game = `Penalty Yards Per Game`,
        yards_per_punt_return = `Avg Yards Per Punt Return`,
        yards_per_punt_return_allowed = `Avg Yards Allowed per Punt Return`,
        percent_rz_points_allowed = `Redzone Points Allowed`,
        percent_rz_points = `Redzone Points`,
        yards_per_rush_allowed = `Yds/Rush Allowed`,
        rush_yards_per_game_allowed = `Rush Yards Per Game Allowed`,
        rush_yards_per_game = `Rushing Yards per Game`,
        yards_per_rush = `Yards/Rush`,
        sacks_per_game = `Average Sacks per Game`,
        points_per_game_allowed = `Avg Points per Game Allowed`,
        points_per_game = `Points Per Game`,
        percent_conv_3rd = `3rd Percent`,
        turnover_margin_per_game = `Avg Turnover Margin per Game`,
        int_per_attempt,
        pass_attempt_per_game,
        pass_completion_per_game,
        pass_attempt_per_game_allowed,
        pass_completion_per_game_allowed,
        rush_td_per_game,
        rush_td_per_game_allowed,
        pass_td_per_game,
        pass_td_per_game_allowed,
        ko_return_td_per_game,
        ko_return_td_per_game_allowed,
        fmb_rec_per_game,
        fmb_loss_per_game
        )
```

Data Exploration and Visualization

Now that we have our data cleaned we are ready to do some exploration and analysis.

Correlations

Before starting this section, we want to stress that correlation is not causation. When using correlation we need to be mindful of making spurious correlations. We also need to be mindful because correlation measures linear association. Thus, outliers can have a large impact on correlation[7]. In a dataset as small as ours (130 observations), outliers could have a big effect on correlation. We also need to be mindful of confounding: when a predictor effect both the outcome (win_perc) and another dependent variable[7].

Now that we have our data cleaned we are ready to do some exploration and analysis. Firstly, we want to see which statistics are most highly correlated with win_percentage. We would expect offensive statistics to be postively correlated (i.e. more offensive yards corresponds with more wins). We can see this in the table below.

```
#making a data table to show highest positive correlations
y <- cfb_data$win_perc
x <- cfb_data %>% select(-Team, - conf, -win_perc, - wins, - losses)

tab <- data_frame(Predictor = rownames(cor(x,y)), Correlation = cor(x,y)) %>%
  slice_max(order_by = Correlation, n = 10)

kable(tab, caption = "Offensive Effects", booktabs = T,
      linesep = "", digits = 3) %>% kable_styling(latex_options = c("striped", "hold_position"))
```

Table 2: Offensive Effects

Predictor	Correlation
points_per_game	0.7650084
percent_conv_3rd	0.6562478
off_yards_per_play	0.6485452
pass_yards_per_attempt	0.6393519
off_yards_per_game	0.6167474
sacks_per_game	0.5877946
turnover_margin_per_game	0.5596805
rush_td_per_game	0.5265560
percent_rz_points	0.5050004
pass_td_per_game	0.4740514

From the table we also notice that several defensive statistics are positively correlated as well. These include sacks per game and turnover margin per game. This intuitively makes sense because the more sacks and more turnovers you get per game the better your defense is, and the more likely you are to win.

We would also expect many defensive stats to be negatively correlated (i.e. more yards allowed to the opposing team corresponds with less wins).

```r
#making a table to show highest negative correlations
tab <- data_frame(Predictor = rownames(cor(x,y)), Correlation = cor(x,y)) %>%
  slice_min(order_by = Correlation, n = 10)

kable(tab, caption = "Defensive Effects", booktabs = T,
      linesep = "", digits = 3) %>% kable_styling(latex_options = c("striped", "hold_position"))
```

Table 3: Defensive Effects

Predictor	Correlation
points_per_game_allowed	-0.7628700
yards_per_play_allowed	-0.6731990
rush_td_per_game_allowed	-0.6632737
yards_per_game_allowed	-0.6559694
rush_yards_per_game_allowed	-0.6433369
yards_per_rush_allowed	-0.6225143
pass_td_per_game_allowed	-0.5287332
int_per_attempt	-0.4176751
percent_conv_4th_allowed	-0.3559097
percent_rz_points_allowed	-0.3556676

We notice that nearly all of the negatively correlated predictors are defensive categories that correspond with failing to stop the oposing teams offense. However, there is one offensive predictor, interceptions thrown per attempt, that is negatively correlated. Again this makes sense because an interception thrown corresponds to a failed offensive drive; therefore, as more interceptions are thrown per pass attempt, the higher the chance a team has at losing.

Lastly, there are several statistics we included in our original cleaned data that do not seem to correlate strongly with win percentage.

```r
#making a table that shows variables that dont appear to have correlation with win #percentage
tab <- data_frame(predictor = rownames(cor(x,y)),
                  abs_correlation = abs(cor(x,y))) %>%
  slice_min(order_by = abs_correlation, n = 5)
colnames(tab) <- c("Predictor", "Absolute Value of Correlation")

kable(tab, caption = "Least Correlated to Wins", booktabs = T,
      linesep = "", digits = 3) %>% kable_styling(latex_options = c("striped", "hold_position"))
```

Table 4: Least Correlated to Wins

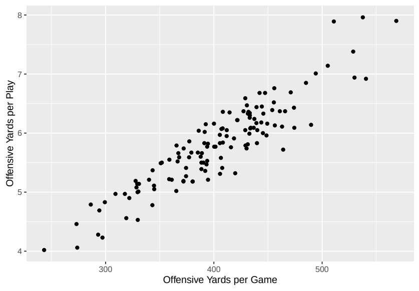
Predictor	Absolute Value of Correlation
ko_return_td_per_game_allowed	0.02393935
penalty_yards_per_game	0.02856942
yards_per_punt_return_allowed	0.04563721
pass_completion_per_game	0.06318917
yards_per_punt_return	0.07628595

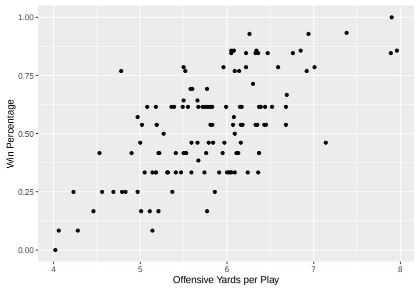
Three out of five of these predictors are special teams plays. Although special team plays can often turn the momentum of a game, it appears that over the course of the season most special team plays do not have a large impact on the number of games won. Two surprising predictors that do not show high correlation with win percentage are pass completion per game and penalty yards per game. These are surprising because if you have more completions per game you would expect the offense to be better. Additionally, a team with many penalty yards are often construed as "undisciplined" in the media.

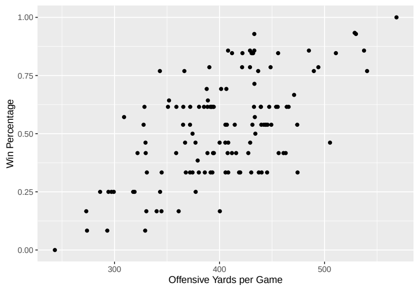
Confounding

As mentioned above, we need to be mindful of confounding. In the tables, we noticed that even after cleaning, several of the variables are still very similar. For example, offensive yards per play and offensive yards per game. There is a good chance these variables are confounded, i.e. that offensive yards per play affects both offensive yards per game and win percentage. We will check this visually below.

```r
#plotting both offensive yards per game and offensive yards per play to show confounding
ggplot(data = cfb_data, aes(x = off_yards_per_game, y = off_yards_per_play)) +
  geom_point() + xlab("Offensive Yards per Game") + ylab("Offensive Yards per Play")

ggplot(data = cfb_data, aes(x = off_yards_per_play, y = win_perc)) +
  geom_point() + xlab("Offensive Yards per Play") + ylab("Win Percentage")

ggplot(data = cfb_data, aes(x = off_yards_per_game, y = win_perc)) +
  geom_point() + xlab("Offensive Yards per Game") + ylab("Win Percentage")
```

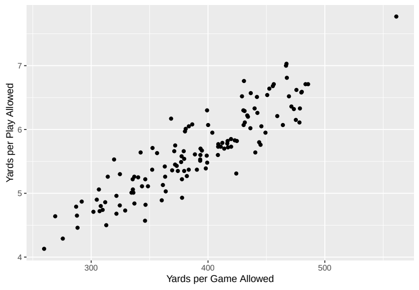


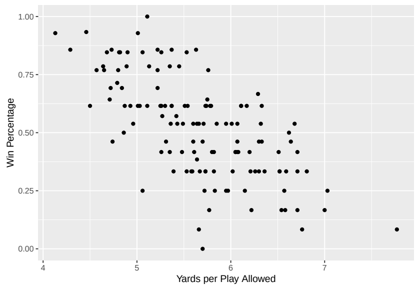


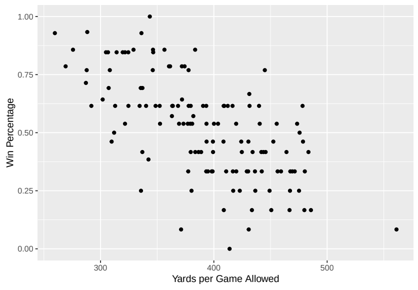


Looking at the graphs above, we see a clear relationship between offensive yards per game and offensive yards per play. Both of these predictors similarly effect win percentage. We see a similar effect for yards per game allowed and yards per play allowed.

```r
#plotting both offensive yards per game allowed and offensive yards per play allowed to show confoundin
ggplot(data = cfb_data, aes(x = yards_per_game_allowed, y = yards_per_play_allowed)) +
  geom_point() + xlab("Yards per Game Allowed") + ylab("Yards per Play Allowed")

ggplot(data = cfb_data, aes(x = yards_per_play_allowed, y = win_perc)) +
  geom_point() + xlab("Yards per Play Allowed") + ylab("Win Percentage")

ggplot(data = cfb_data, aes(x = yards_per_game_allowed, y = win_perc)) +
  geom_point() + xlab("Yards per Game Allowed") + ylab("Win Percentage")
```







When it comes to confounding there is a balance that needs to be struck. According to Zhongheng Zhang, some researches suggest including as many covariates as possible since incorporating a larger number of predictors will make independent associations more reliable^{1}0. However, Zhang also goes on to say that overfitting can occur when too many variables are introduced into a model that already performs well^{1}0.

Therefore, we will need to strike a balance between model performance, and how many predictors we have in the model. In order to do this, we will further clean the data by removing variables that are very similar while leaving enough variables for machine learning to occur.

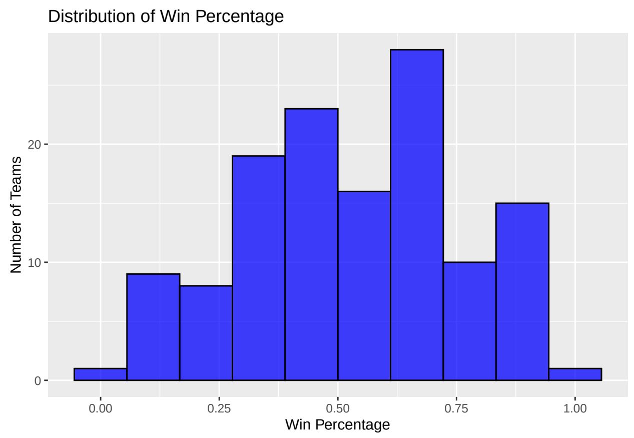
```r
#cleaning data by removing variables that are very similar
cfb_data <- cfb_data %>%
  select(-off_yards_per_game, -yards_per_game_allowed, )
```

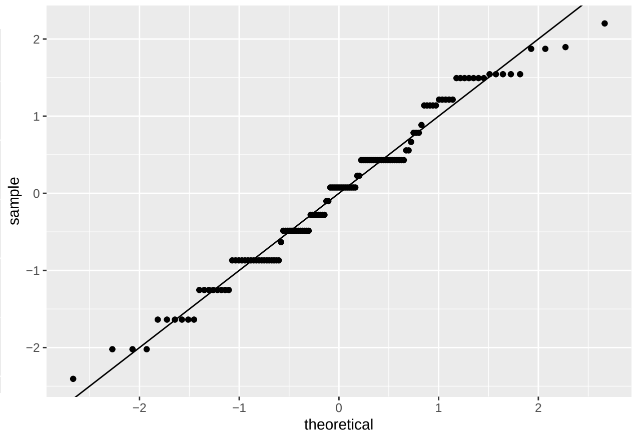
Data Visualization

Now that we explored some of predictors correlation to win percentage, let's see if visualization gives any additional insights.

To begin, we explore the distribution of win percentage. In doing so, we see that the distribution appears to almost be normal. However, because there are so few data points it is hard to determine from the histogram. Instead, we make a qq-plot with the scaled values of win percentage and see that the distribution does appear to be normal. This makes intuitive sense since there are a few very good teams, a few very bad teams, and a group of teams in the middle.

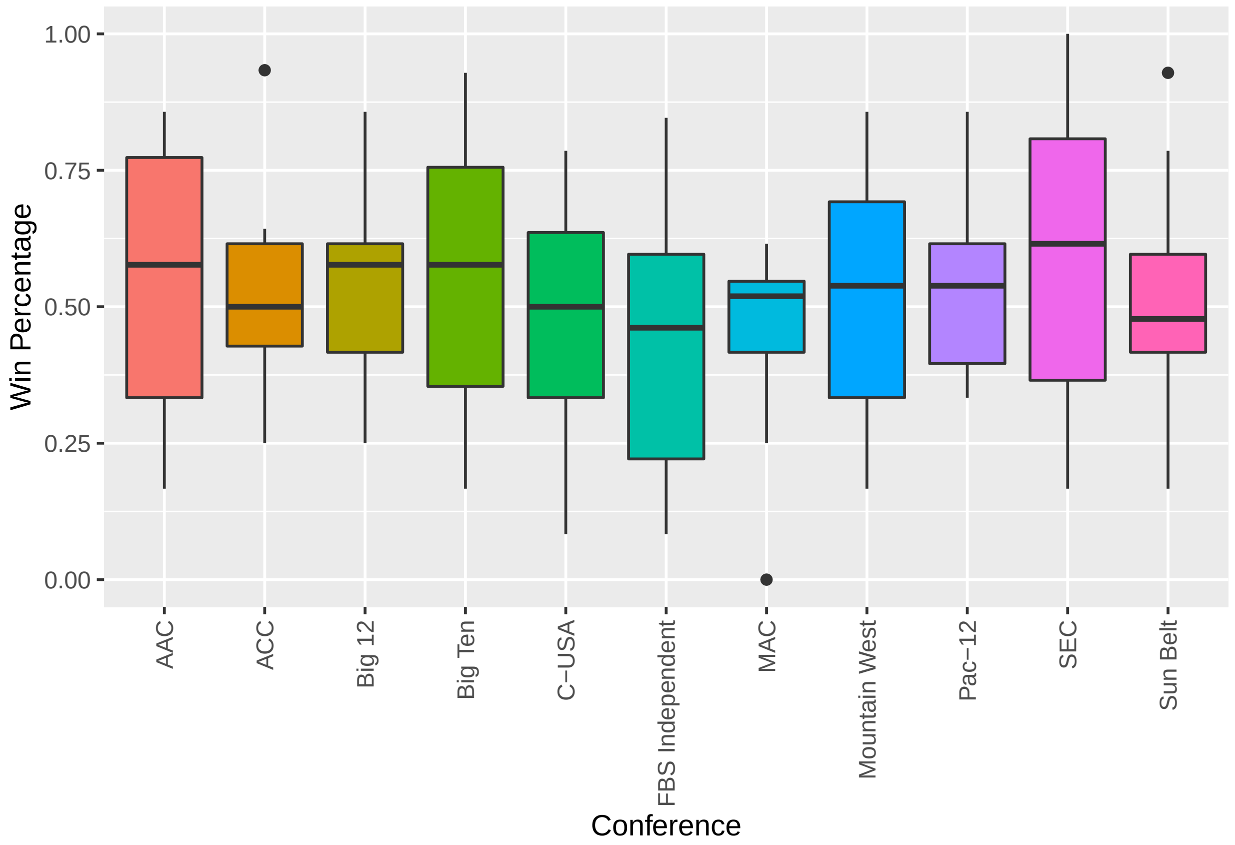
```r
#histogram showing distribution of win percentage
cfb_data %>% ggplot(aes(x = win_perc)) +
  geom_histogram(bins = 10, alpha = 0.75, col = "black", fill = "blue") +
  xlab("Win Percentage") + ylab("Number of Teams") +
  ggtitle("Distribution of Win Percentage")

#qq plot showing data appears to be normal
ggplot(data = cfb_data, aes(sample = scale(win_perc))) + geom_qq() + geom_abline()
```





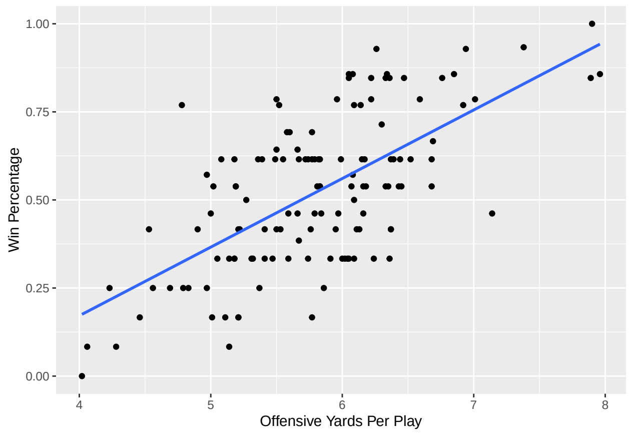
Grouping by conference, we see that most conferences have a median around 0.5 and two conferences have outliers. The conference could be a useful predictor in our model because there are differences in the inner quantile ranges between conferences. These differences may be due to some conferences having better teams than others. For example, the SEC and BIG Ten are considered by many to be the best conferences. You can see that there upper quantile range is higher than other conferences suggesting more of their teams have a higher win percentage. However, because these differences appear to be small we will remove it from the dataset when we partition the data.

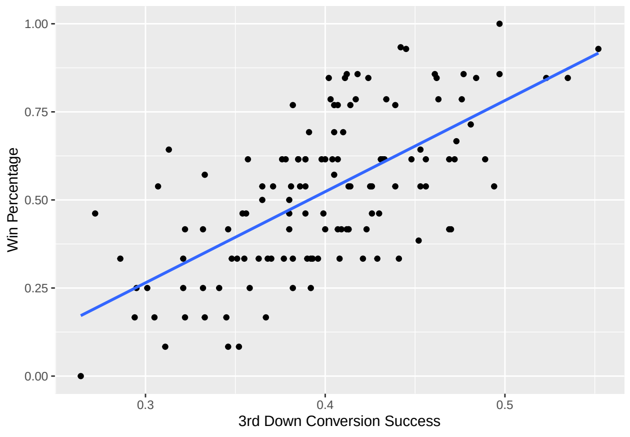
```r
#boxplot showing win percentage for difference conferences
cfb_data %>% ggplot(aes(x = conf, y = win_perc, fill = conf)) +
  geom_boxplot() +
  xlab("Conference") + ylab("Win Percentage") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) +
  theme(legend.position = "none")
```



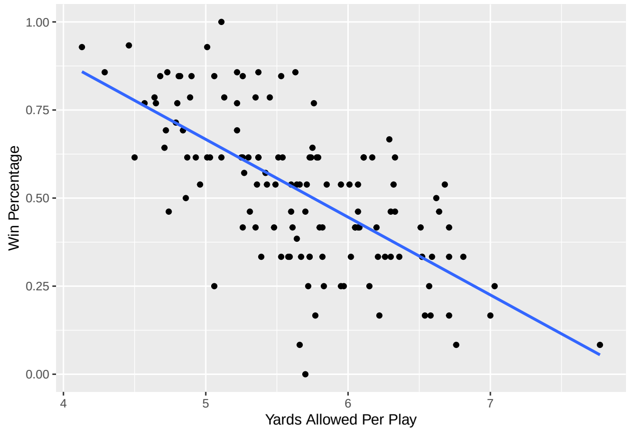
Visualizing some of the predictors with the highest correlation reveals that the relationship between win percentage and several statistics appear to be linear. This suggests that linear regression may be a good algorithm to predict a teams success. This will be examined in further detail later.

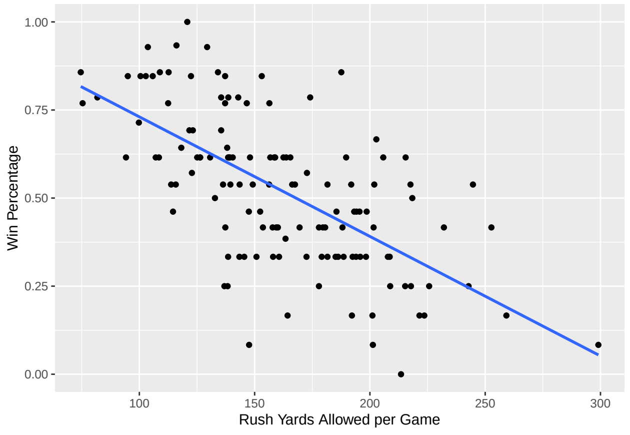
```r
#plot win percentage vs offensive yards per play
cfb_data %>% ggplot(aes(x = off_yards_per_play, y = win_perc)) +
  geom_point() + xlab("Offensive Yards Per Play") + ylab("Win Percentage") +
  geom_smooth(method = "lm", se = FALSE)

#plot win percentage vs 3rd down conversion
cfb_data %>% ggplot(aes(x = percent_conv_3rd, y = win_perc)) +
  geom_point() + xlab("3rd Down Conversion Success") + ylab("Win Percentage") +
  geom_smooth(method = "lm", se = FALSE)
```





```
#plot win percentage vs yards per play allowed
cfb_data %>% ggplot(aes(x = yards_per_play_allowed, y = win_perc)) +
  geom_point() + xlab("Yards Allowed Per Play") + ylab("Win Percentage") +
  geom_smooth(method = "lm", se = FALSE)

#plot win percentage vs rush yards per game allowed
cfb_data %>% ggplot(aes(x = rush_yards_per_game_allowed, y = win_perc)) +
  geom_point() + xlab("Rush Yards Allowed per Game") + ylab("Win Percentage") +
  geom_smooth(method = "lm", se = FALSE)
```





Evaluation Method and Algorithms

Evaluation Method

In order to evaluate how well our algorithms do, we will use the RMSE to compare their success. Here win percentage ranges from 0 to 1. Most teams play around 13 games. Achieving an RMSE of 0.077 would correspond to being able to predict a teams number of wins within one game. This will be used as the benchmark goal of our machine learning model.

$$RMSE = \sqrt{\frac{\sum_{u,i}(y_{u,i} - yhat_{u,i})^2}{N}}$$

where:

- $y_{u,i}$ is the rating for movie i by user u
- $yhat_{u,i}$ is the rating for movie i by user u

Here we create a function that calculates the RMSE that we will use while to evaluate how well our model does at predicting ratings.

```
#RMSE function used for evaluations
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Algorithms

As mentioned previously, our dataset is small with only 130 entries. Once we split the data our training set will be even smaller ~ 100 entries. Because of this, we need to consider a way to address this with our machine learning algorithms. According to Ahmed El Deeb and Jyoti Prakash Maheswari[3,6], some good approaches to machine learning on small datasets are cleaning the data, using machine learning ensembles, using regularization if possible, and using models robust to outliers. Below we will attempt to use several of these techniques to come up with a machine learning algorithm that can predict the winning percentage a team will have.

Partitioning the Data

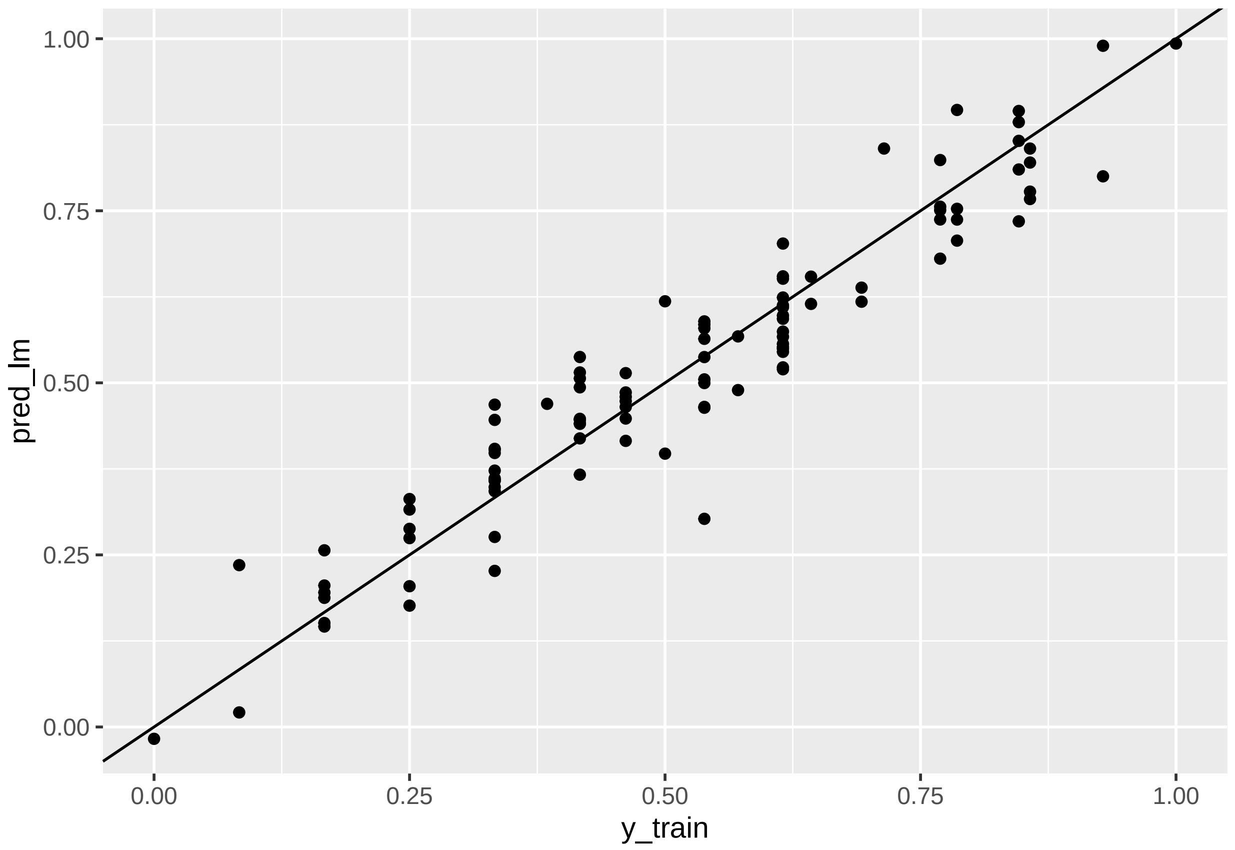
Because our dataset is small, we want to have as much data as possible to train our machine learning algorithm with. However, we also want enough data to test the ability of our machine learning algorithm. In order to do this, create a training set that includes 80 percent of our data. This will leave 20 percent of our data to test against.

```
#partition data into train, and test sets
set.seed(1, sample.kind="Rounding")
test_ind <- createDataPartition(y = cfb_data$win_perc, times = 1, p = 0.2, list = FALSE)
test_set <- cfb_data[test_ind,]
train_set <- cfb_data[-test_ind,]

y_train <- train_set$win_perc
x_train <- train_set %>% select(-win_perc, -Team, -wins, - losses, -conf)

y_test <- test_set$win_perc
x_test <- test_set %>% select(-win_perc, -Team, -wins, - losses, -conf)
```

Linear Regression

Our first approach will be to use linear regression to fit the data. Here we need to be sure to set the intercept at 0 because it wouldnt make sense for a team to have a negative win percentage. Plotting the results we see that linear regression does a fairly well job at predicting the win percentage of the teams. Calculating the RMSE we see that we get an RMSE of approximately 0.06. In a 13 game season, this corresponds to being able to predict how many games a team will win within +/- 0.78 games.

```
#fit linear regression model
fit_lm <- train(x_train, y_train, method = "lm", tuneGrid = data.frame("intercept" = 0))

# predict win percentage based on model and calculate RMSE
pred_lm <- predict(fit_lm)
RMSE_lm <- RMSE(y_train,pred_lm)
RMSE_tab <- data_frame("Method" = "Linear Regression", "RMSE" = RMSE_lm)

#plot how well predictions fit linear model
df <- data_frame(y_train, pred_lm)
df %>% ggplot(aes(x = y_train, y = pred_lm)) + geom_point() + geom_abline()
```



```r
#show table of results
kable(RMSE_tab, caption = "Model RMSEs (Training)", booktabs = T, linesep = "", digits = 3) %>%
  kable_styling(latex_options = c("striped", "hold_position"))
```

Table 5: Model RMSEs (Training)

Method	RMSE
Linear Regression	0.066

To look closer at the data, we will take a look at the variable importance according to the model. As seen in the table below, 8 of the top 10 most important variables are defensive statistics.

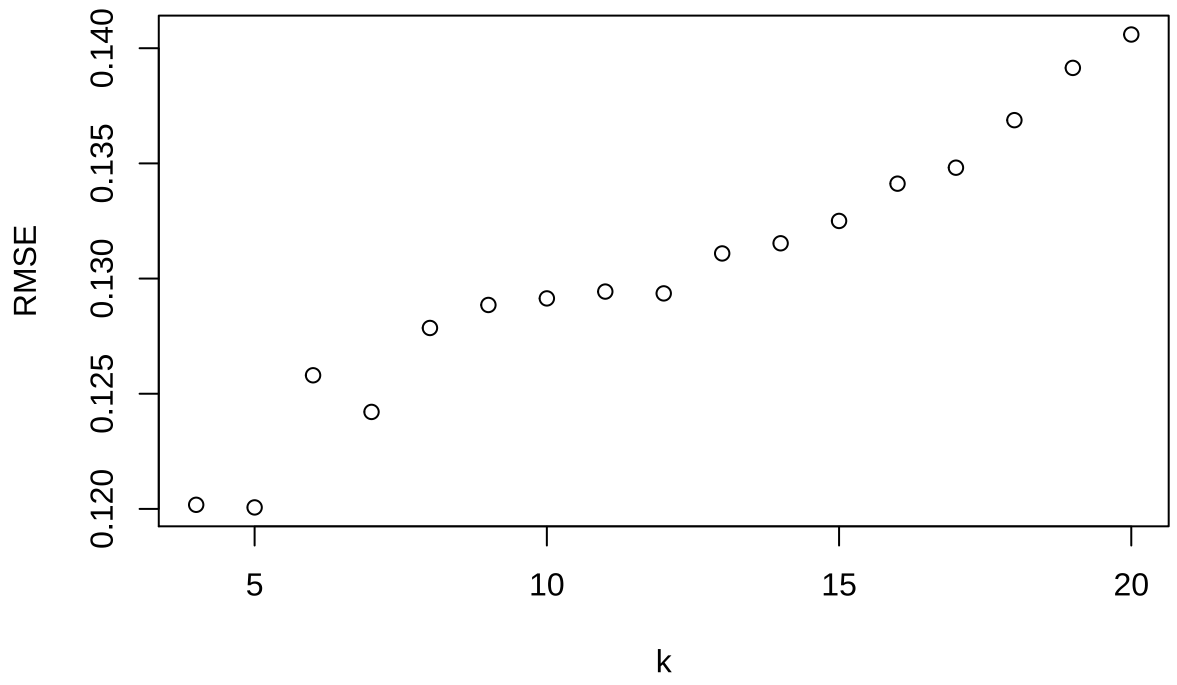
```r
#variable importance from linear model
varImp(fit_lm)
```

```
## lm variable importance
##
##   only 20 most important variables shown (out of 37)
##
##                                 Overall
## pass_attempt_per_game_allowed  100.00
## yards_per_rush_allowed          90.57
## rush_yards_per_game_allowed     87.36
## points_per_game_allowed         87.02
## yards_per_play_allowed          85.24
## pass_yards_per_game_allowed     72.73
## points_per_game                 61.48
## off_yards_per_play                59.85
## pass_yards_per_attempt            57.42
## ko_return_td_per_game_allowed     56.49
## yards_per_rush                    54.46
## sacks_per_game                    52.49
## percent_conv_4th_allowed          51.92
## pass_td_per_game_allowed          51.45
## rush_yards_per_game               50.02
## fmb_loss_per_game                 47.68
## rush_td_per_game_allowed          42.12
## pass_td_per_game                  35.34
## yards_per_punt_return             33.48
## pass_attempt_per_game             33.40
```

K Nearest Neighbors

Now that we explored linear regression, we will use K Nearest Neighbors (KNN). Because our data set is small, we are able to use this machine learning approach. However, if our dataset gets larger in the future it may be difficult to implement due to the computational requirements of calculating distance to neighboring points. According to Naresh Kumar at "The Professional Point" KNN becomes difficult as the number of dimensions get higher because it becomes difficult for the algorithm to calculate the distance in each dimension[5]. However, for now while our dataset is small knn can be used. We need to optimize the parameter "k". As k gets larger we are performing more smoothing and as k gets smaller we are smoothing fewer and fewer points (overfitting the model when k is 1, i.e. each individual point). To choose this parameter we use multiple values and then select the one with the lowest RMSE.

```r
#create values of k
ks = seq(4,20)

#Calculate RMSE for each value of k
RMSE_knn <- sapply(ks, function(k){
  fit_knn <- train(x_train, y_train, method = "knn", tuneGrid = data.frame("k" = k))
  pred_knn <- predict(fit_knn)
  RMSE(y_train, pred_knn)
  })

#plotting RMSE vs k and selecting best value
plot(x = ks, y = RMSE_knn, xlab = "k", ylab = "RMSE")
```



```r
ind <- which.min(RMSE_knn)
k_best <- ks[ind]
```

According to Dhilip Subramanian, a typical way of picking k is to take the square root of the number of predictors[9]. From the graph, we see that this appears to be the case. We have 37 predictors in our training data set which corresponds to a k of approximately 6. From our plot, we see that this rule of thumb holds true. Going forward, we will use the best k obtained above when choosing k for the test set. Compared to linear regression we see that knn performed worse.

```r
#calculating the predictions from that result for use later
fit_knn <- train(x_train, y_train, method = "knn", tuneGrid = data.frame("k" = k_best))
pred_knn <- predict(fit_knn)

#calculate RMSE of best knn
RMSE_knn <- RMSE(y_train, pred_knn)
RMSE_tab <- RMSE_tab %>% add_row("Method" = "KNN", "RMSE" = RMSE_knn)

#show results in table
kable(RMSE_tab, caption = "Model RMSEs (Training)", booktabs = T, linesep = "", digits = 3) %>%
  kable_styling(latex_options = c("striped", "hold_position"))
```

Table 6: Model RMSEs (Training)

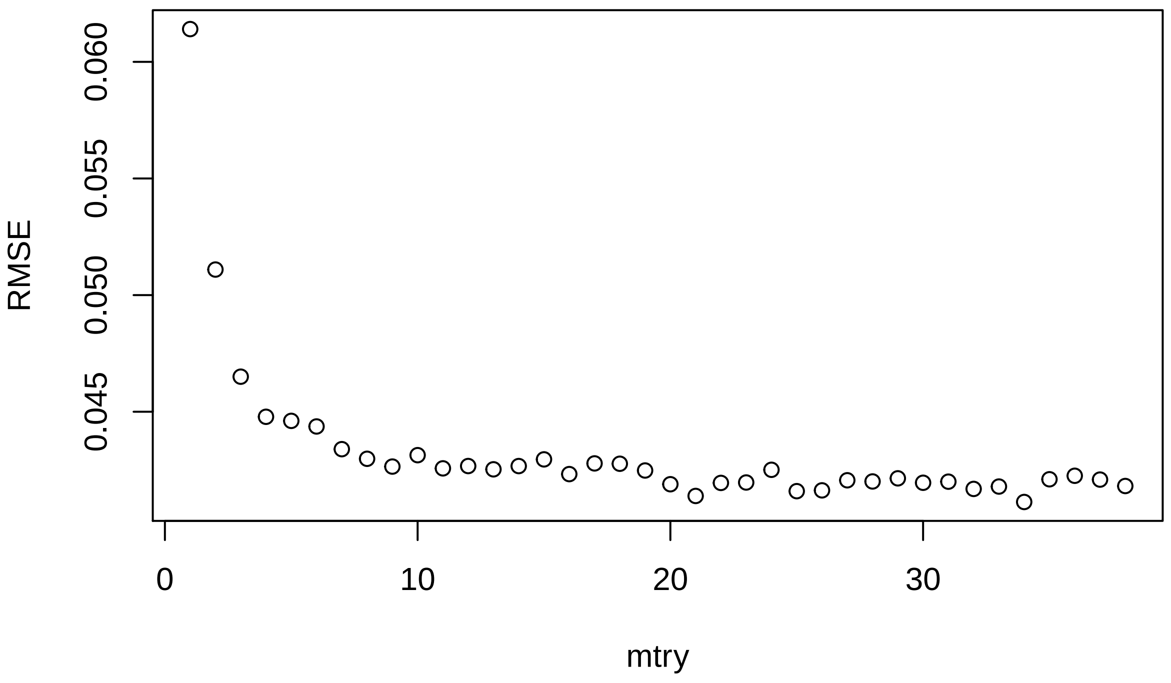
Method	RMSE
Linear Regression	0.066
KNN	0.120

Random Forest

Similarly to knn, random forest is another method we can implement while our dataset is small. As our dataset gets larger random forest becomes more computationally taxing on our computer. However, for now, we can create a large number of trees to compensate for not having a big dataset. Here we aim to optimize the parameter mtry, which corressponds to the number of variables for splitting at each node in the tree.

Some advantages of random forest include: being robust to correlated predictors, it can be used to select predictors by looking at the variable importance of the model[8].

```r
# fit the model with 38 (number of variables we have) different values of mtry
m <- seq(1,38)
RMSE_rf <- sapply(m, function(x){
  fit_rf <- train(x_train, y_train, method = "rf",
                  tuneGrid = data.frame("mtry" = x))
  pred_rf <- predict(fit_rf)
  RMSE(y_train, pred_rf)
  })
#plot to see which is best
plot(x = m, y = RMSE_rf, xlab = "mtry", ylab = "RMSE")
```



```r
mtry_best <- which.min(RMSE_rf)

# fit model with best mtry
fit_rf <- train(x_train,y_train, method = "rf", tuneGrid = data.frame("mtry" = mtry_best),
                  nodesize = 2, ntree = 2000)

pred_rf <- predict(fit_rf)
#Calculating RMSE based on predictions
RMSE_rf <- RMSE(y_train, pred_rf)

#create table to see top predictors
tab <- data_frame(rownames(fit_rf$finalModel$importance), fit_rf$finalModel$importance) %>%
  arrange(desc(fit_rf$finalModel$importance[,"IncNodePurity"])) %>% slice(n = 1:10)
names(tab) <- c("Predictor", "Importance")

kable(tab, caption = "Variable Importance", booktabs = T, linesep = "", digits = 3) %>%
  kable_styling(latex_options = c("striped", "hold_position"))
```

Table 7: Variable Importance

Predictor	Importance
points_per_game_allowed	1.23649618
points_per_game	1.22611929
pass_yards_per_attempt	0.30371816
rush_td_per_game_allowed	0.29794642
percent_conv_3rd	0.24127219
yards_per_play_allowed	0.23565455
sacks_per_game	0.16847166
yards_per_rush_allowed	0.15790776
rush_yards_per_game_allowed	0.10365187
off_yards_per_play	0.08197098

We see several of the same predictors as in the linear regression model including points per game allowed, rush yards per game allowed, and others. Similarly, we see that the majority (in this case 6 to 4) are defensive stats.

Comparing the RMSE to linear regression and knn we see that random forest performed better than both other models.

```r
#show results in table
RMSE_tab <- RMSE_tab %>% add_row("Method" = "Random Forest", "RMSE" = RMSE_rf)
kable(RMSE_tab, caption = "Model RMSEs (Training)", booktabs = T, linesep = "", digits = 3) %>%
  kable_styling(latex_options = c("striped", "hold_position"))
```

Table 8: Model RMSEs (Training)

Method	RMSE
Linear Regression	0.066
KNN	0.120
Random Forest	0.039

Regularization

According to UC Bussiness analytics school, as the number of predictors increase, we are more likely to capture features that have some multicollinearity, which causes higher variability in our fit coefficients[1]. Regularized regression helps to penalize parameters with large effects but small sample size. Ridge regression, also known as L2 regression, adds a second order penalty to the objective function[1]. Lasso regression applies an L1 pentalty to the objective funciton. We can see these in the formula below:

Ridge Regression Goal:

$minimize\{SSE + \lambda \sum_j^p B_j^2\}$

Lasso Regression Goal: $minimize\{SSE + \lambda \sum_j^p |B_j|\}$

Implementing the regularized regression through the use of the glmnet method. Here we center and scale the predictors as well as exclude near zero variance and zero variance predictors. Here we are trying to optimize alpha and lambda. An alpha of 0 corresponds to lasso regression, while an alpha of 1 corresponds to ridge regression. Lambda is the size of the penalty applied.

```r
#use cross validation
train_control <- trainControl(method = "cv")
#select a variety of alphas and lambdas to test
grid <- expand.grid(alpha = seq(0,1,0.1), lambda = seq(0.0001, 1, length = 50))

#fit regularized model
fit_reg <- train(x_train, y_train, method = "glmnet", preProc = c("center", "scale", "nzv", "zv"), tune

#selecting the best alpha and lambda parameters from th model
a_best <- fit_reg$bestTune$alpha
l_best <- fit_reg$bestTune$lambda

#fit regularized model with best alpha and lambda
grid <- expand.grid(alpha = a_best, lambda = l_best)
fit_reg <- train(x_train, y_train, method = "glmnet", preProc = c("center", "scale", "nzv", "zv"), tune

#predict with optimized parameters and calculate RMSE
pred_reg <- predict(fit_reg)
RMSE_reg <- RMSE(y_train, pred_reg)

#a table showing the best alpha and lambda
df <- data_frame("alpha" = a_best, "lambda" = l_best, "RMSE" = RMSE_reg)
kable(df, caption = "Best Tuning Parameters", booktabs = T, linesep = "", digits = 3) %>%
  kable_styling(latex_options = c("striped", "hold_position"))
```

Table 9: Best Tuning Parameters

alpha	lambda	RMSE
0.8	0.021	0.086

After cross validation, we see that our best tuning parameters provide an RMSE of 0.086. Comparing this to the other methods, we see that it performed worse than linear regression and random forest but better than knn. Since it performed worse than linear regression, it might indicate that our linear regression model overfit the data.

```r
#show RMSE results in table
RMSE_tab <- RMSE_tab %>% add_row("Method" = "Regularization", "RMSE" = RMSE_reg)
kable(RMSE_tab, caption = "Model RMSEs (Training)", booktabs = T, linesep = "", digits = 3) %>%
  kable_styling(latex_options = c("striped", "hold_position"))
```

Table 10: Model RMSEs (Training)

Method	RMSE
Linear Regression	0.066
KNN	0.120
Random Forest	0.039
Regularization	0.086

Ensemble

Because our dataset is small, one of the things we can do to make our model more robust is use an ensemble of machine learning algorithms[6]. Here we will use an equally weighted ensemble.

```r
# calculate predictions based on equal weight to each model
pred_ensemble <- (pred_knn + pred_rf + pred_lm + pred_reg)/4

#calculate RMSE
RMSE_ensemble <- RMSE(y_train, pred_ensemble)

#show results in table
RMSE_tab <- RMSE_tab %>% add_row("Method" = "Ensemble", "RMSE" = RMSE_ensemble)
kable(RMSE_tab, caption = "Model RMSEs (Training)", booktabs = T, linesep = "", digits = 3) %>%
  kable_styling(latex_options = c("striped", "hold_position"))
```

Table 11: Model RMSEs (Training)

Method	RMSE
Linear Regression	0.066
KNN	0.120
Random Forest	0.039
Regularization	0.086
Ensemble	0.069

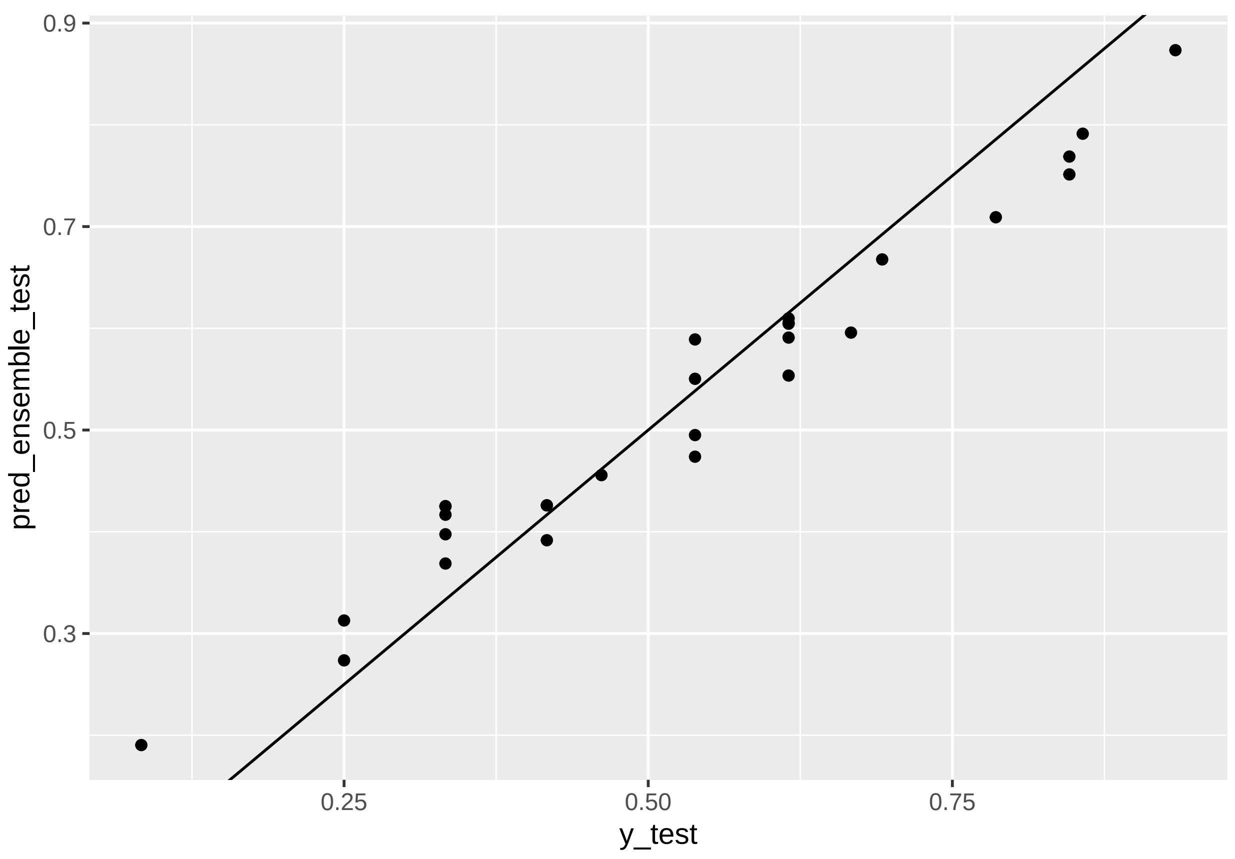
As you can see, our ensemble performs nearly as well as linear regression while having the benefit of being more robust to outliers and multicollinearity in our small dataset. This will be the method we use for our test_set.

Testing our Model

As mentioned above, we will be using the ensemble model to make our final predictions on the test set. We will use the best tuning parameters obtained from tuning on our training set. Below we perform these techniques, first fitting each model and getting that particular models predictions, and then using an equal weight average for the final prediction.

```r
#fit linear regression model on test set
fit_lm_test <- train(x_test, y_test, method = "lm", tuneGrid = data.frame("intercept" = 0))
# predict win percentage based on lm
pred_lm_test <- predict(fit_lm_test)


#fit knn on test set with best k
fit_knn_test <- train(x_test, y_test, method = "knn", tuneGrid = data.frame("k" = k_best))
#predict win percentage based on knn
pred_knn_test <- predict(fit_knn_test)


# fit rf on test set with best mtry
fit_rf_test <- train(x_test, y_test, method = "rf", tuneGrid = data.frame("mtry" = mtry_best),
                  nodesize = 2, ntree = 2000)
#predict win percentage based on rf
pred_rf_test <- predict(fit_rf_test)
```

```
# fit regularized model on test set with best alpha and lambda
#selecting the best alpha and lambda parameters from th model
grid <- expand.grid(alpha = a_best, lambda = l_best)
fit_reg_test <- train(x_test, y_test, method = "glmnet", preProc = c("center", "scale", "nzv", "zv"), tu
#predict win percentage bsed on regularization
pred_reg_test <- predict(fit_reg_test)


#final ensemble prediction for win percentage
pred_ensemble_test <- (pred_lm_test + pred_knn_test + pred_rf_test + pred_reg_test)/4

#final ensemble RMSE
RMSE_ensemble_test <- RMSE(y_test, pred_ensemble_test)

#making a table
RMSE_tab_test <- data_frame("Method" = "Ensemble", "RMSE" = RMSE_ensemble_test)

#plot how well predictions fit the data
df <- data_frame(y_test, pred_ensemble_test)
df %>% ggplot(aes(x = y_test, y = pred_ensemble_test)) + geom_point() + geom_abline()
```



```
#show table of results
kable(RMSE_tab_test, caption = "Final Model RMSE (Test Set)", booktabs = T, linesep = "", digits = 3) %
  kable_styling(latex_options = c("striped", "hold_position"))
```

Table 12: Final Model RMSE (Test Set)

Method	RMSE
Ensemble	0.058

Conclusion

Overall, we see that our ensemble model performed quite well, achieving and RMSE of 0.058. We were able to do better than our goal of 0.077 which coressponded to predicting a win percentage within 1 game in a 13 game season. Additionally, by examining the importance of predictors within the different algorithm, we were able determine that defensive predictors appear to be slightly more significant than offensive predictors. We saw this in both the linear regression model where 8 of the top 10 predictors were defensive statistics, and in the random forest model where 6 of the top 10 predictors were defensive statistics. This would be important to know for coaches, because they could put more of an emphasis on recruiting players who are highly rated on defense. Additionally, they could focus on defense more in practice.

Future work includes scraping more data from the NCAA website so that we have more observations to train and test against. This data would be from previous seasons, so it would be interesting to see if there is a time effect on the model and if the importance of offensive or defensive predictors change throughout time. Also, future work could explore models such as quantile-quantile regression which have been suggested to be more robust to outliers in smaller datasets.

References

1) Boehmke, Bradley. Regularized Regression, UC Business Analytics R Programming Guide, 2020, uc-r.github.io/regularized_regression.

2) Brocklebank, John. “The Truth about Sabermetrics.” Samford University, Nov. 2018, www.samford.edu/sports-analytics/fans/2018/The-Truth-about-Sabermetrics.

3) Deeb, Ahmed El. “What to Do with ‘Small’ Data?” Medium, Rants on Machine Learning, 14 Oct. 2015, medium.com/rants-on-machine-learning/what-to-do-with-small-data-d253254d1a89.

4) Gallini, Jeff. “College Football Team Stats 2019-2020.” Kaggle, 15 Jan. 2020, www.kaggle.com/jeffgallini/college-football-team-stats-2019.

5) Kumar, Naresh. Advantages and Disadvantages of KNN Algorithm in Machine Learning, 2019, theprofessionalspoint.blogspot.com/2019/02/advantages-and-disadvantages-of-knn.html.

6) Maheswari, Jyoti Prakash. “Breaking the Curse of Small Data Sets in Machine Learning: Part 2.” Medium, Towards Data Science, 27 Apr. 2019, towardsdatascience.com/breaking-the-curse-of-small-data-sets-in-machine-learning-part-2-894aa45277f4.

7) No Author. Cautions about Correlation and Regression , University of Kentucky, 2020, www.uky.edu/~kdbrad2/EPE557/Notes/Chapter%202%20C.pdf.

8) Saraswat, Manish. “Practical Tutorial on Random Forest and Parameter Tuning in R Tutorials & Notes: Machine Learning.” HackerEarth, 2020, www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/tutorial-random-forest-parameter-tuning-r/tutorial/.

9) Subramanian, Dhilip. “A Simple Introduction to K-Nearest Neighbors Algorithm.” Medium, Towards Data Science, 3 Jan. 2020, towardsdatascience.com/a-simple-introduction-to-k-nearest-neighbors-algorithm-b3519ed98e.

10) Zhang, Zhongheng. “Too Much Covariates in a Multivariable Model May Cause the Problem of Overfitting.” Journal of Thoracic Disease, Pioneer Bioscience Publishing Company, Sept. 2014, www.ncbi.nlm.nih.gov/pmc/articles/PMC4178069/.