Predict win/lose of each game of March Madness for year 2018

Team Scorpion

June Kim (jkim654), Hoyin Lau (hlau4), Xiaomei Sun (xsun56), Taiga Hasegawa (taigah2), Leo Franco Soto (francst2)

Introduction

The NCAA Division I Men's Basketball Tournament, also known as NCAA March Madness happens every year in spring. It features 68 college basketball teams from Division I level of the National Collegiate Atheletic Association (NCAA) to determine the national championship. March Madness was first created in 1939 by the National Association of Basketball Coaches and was pitched by Harold Olsen. And currently millions of people in America fill out a bracket to correctly predict the outcome of the entire event.

The format of the tournament are in rounds in the following order[1]:

- The First Four
- The First Round (the Round of 64)
- The Second Round (the Round of 32)
- The Regional Semi-finals (participating teams are known popularly as the "Sweet Sixteen")
- The Regional Finals (participating teams are known commonly as the "Elite Eight")
- The National Semi-finals (participating teams are referred to officially as the "Final Four")
- The National Championship

The motivation behind this project is to correctly predict the outcome of a sports game. The ability to accurately predict the win/loss could help sports betting significantly. We will utilize the basketball game data, like the number of 2 pointers/3 pointers in a given game, to draw out useful information and provide insight on the outcome of each game, which is invaluable to sports bettors and viewers who just want to have a fun bet with friends. The data is from Kaggle Machine Learning Competition hosted by Google Cloud and NCAA 2018. It is a historical data collecting from year 1985 to 2018 (i.e. season 2017-2018, since this year's season is 2018-2019)[2].

There are other people who use the same dataset to do different analyses. The most common topics are the overall rating of teams and players, predicting future best team, how different coaches will have impact on the performance of the teams. These are the topics most people have interest with and will spend a lot of time to discuss about. There are lots of very interesting analyses, and they are totally different from the others. They are trying to predict the salary of different players based on their performance in game, how long are they staying, and how well they do from time to time. There is also a prediction of the salary of different coaches based on how well their coaching team performed. Similarly, there are also other people analyze the win and lose rate, but the way we analyze is different from other people. We are using the overall past data, such as scores from the pasts, location, number of three points and free throws and many other variables to predict the win rate, while other people only used the current data or only few variables to predict the win rate. In this project, the goal is was to find the most accurate model to find the winning predictions for NCAA March Madness in year 2018.

Data Exploration

The original data from Kaggle Machine Learning Competitions had 65 variables with 204,861 observations. For this project, we have cleaned the dataset leaving us with 48 variables and 204,861 observations. The links to the cleaned datasets are below:

Training Data (https://drive.google.com/open?id=1henbg-CbVdXcr8jnsLH7JRA0AuWriPoD)

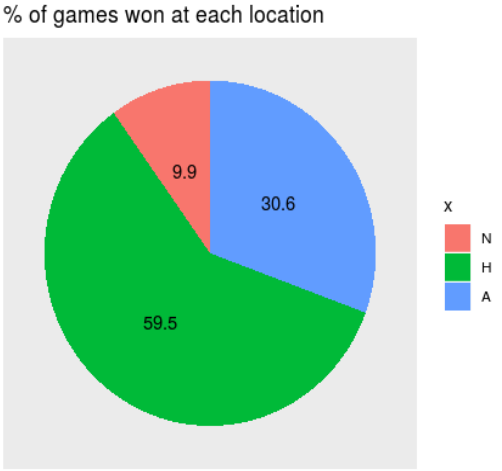
Test Data (https://drive.google.com/file/d/1p6nr-kGy-orNEP8fEn0YjNgdI9Spo9aq/view?usp=sharing)

The training dataset has 204,727 observations and the testing dataset has 134 observations.

The dataset contains information about the regular season before the March Madness and studies on the performance on regular seasons for the teams presented directly influences whether or not they make it to the March Madness itself. Therefore, we have conducted seasonal analysis and overall performance of each team before jumping into NCAA Tour results.

Season Analysis

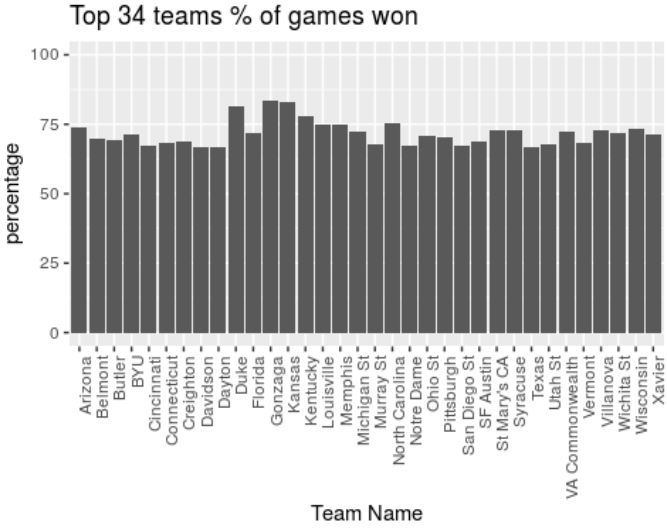
How teams perform during regular seasons affect their chances of getting into the March Madness. In this part, we have analyzed the seasonal performance of the teams that have historically made it into March Madness. First, we explored the relationship between the location of the game with the percentage of wins. The goal here was to find if percentage of wins are associated with the location.

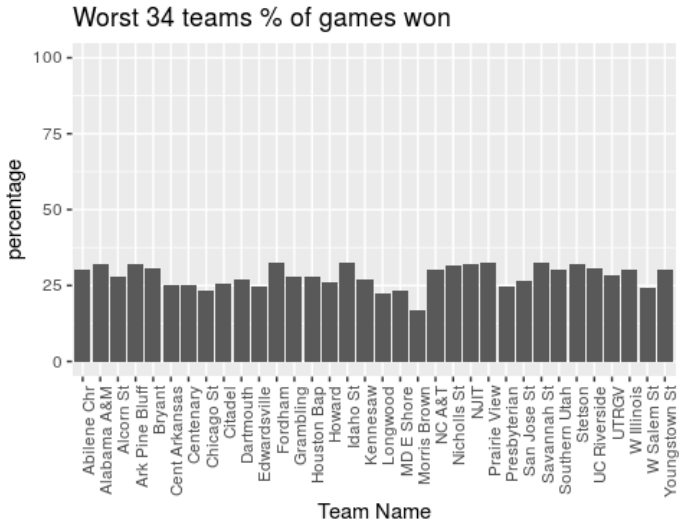


From looking at the graph, it is possible to witness 59.5% of the games won were at home while 30.6% were games won away. Neutral locations seem to be the worst options for teams to win, leaving 9.9% of the games won at this location. From this, we can see that locations may have a big impact in the chance of winning.

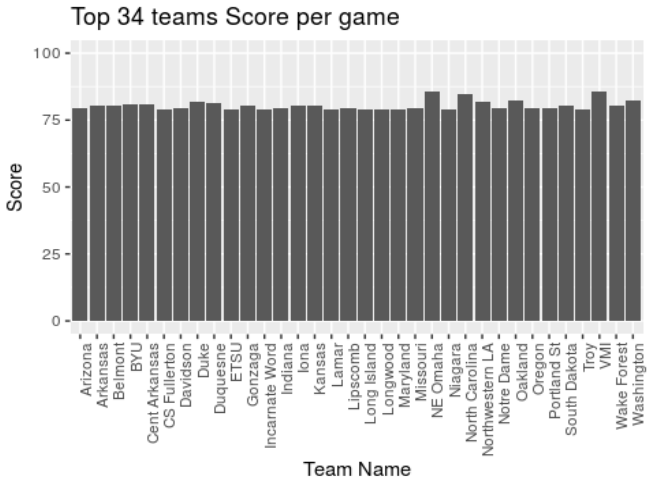
Team Analysis

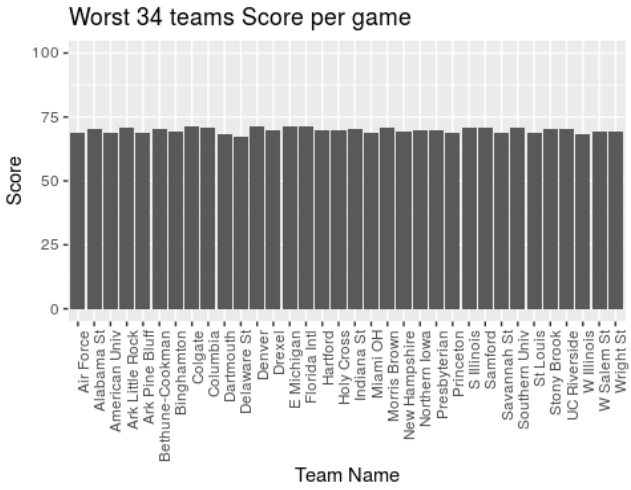
Teams can be evaluated in terms of (1) goals they score in average or (2) win percentage across seasons. First, we considered the win percentage across the regular season. The questions that could be answered from this analysis are: who are the top 34 teams that frequently wins the game in the regular seasons? Are they ones who make it to the March Madness most frequently?





From this exploration, it was possible to see that the top 34 teams with the highest % of games won are one of the most frequent teams playing at NCAA March Madness. For instance, Vilanova won in 2018 and it is one of the top 34 teams. Duke, who is on the top 34 teams, is also very strong team, always making it to the Elite Eight most of the seasons. Moreover, when we look at the worst 34 teams, the winning % of games are in average 25-30%.



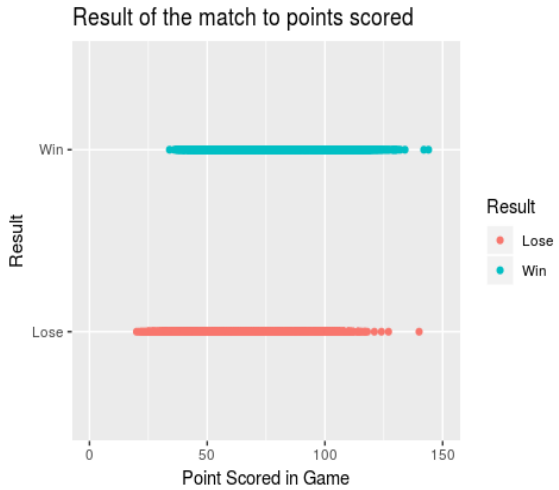


From observing the scores per game of the top 34 highest scoring team per game and those in the bottom 34, we can actually see there are some teams we observed in top % winning teams and bottom % winning teams. It is possible that those who are both in the list of top scoring and % winning would be also in the NCAA March Madness would be the ones making it to the *Final Four*. Just for a side note, the top five frequent winners of NCAA March Madness were: Kentucky, Kansas, North Carolina, Duke and Temple. Since 2015, Gonzaga has been performing well, making it to Elite Eight most of the seasons.

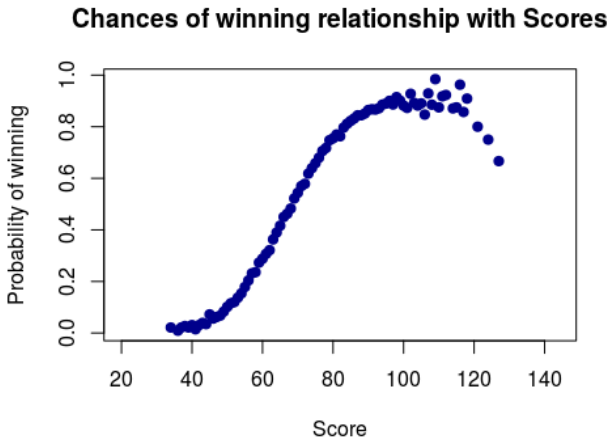
The following table shows some of the teams we have to look at if we are trying to predict the next winner of the March Madness.

Top Performing Teams (Top Scoring & Top Winning %)
Gonzaga
Kansas
Duke
North Carolina
Arizona
BYU
Belmont
Notre Dame
Davidson

Another question we had was to see if points scored are correlated to the game result. To answer this question, we compared the game result relative to the total scores.



From this graph, we can see that winning matches have higher average points scored in the game. However, there are some outliers where it was a lost game, but point scored is very high.

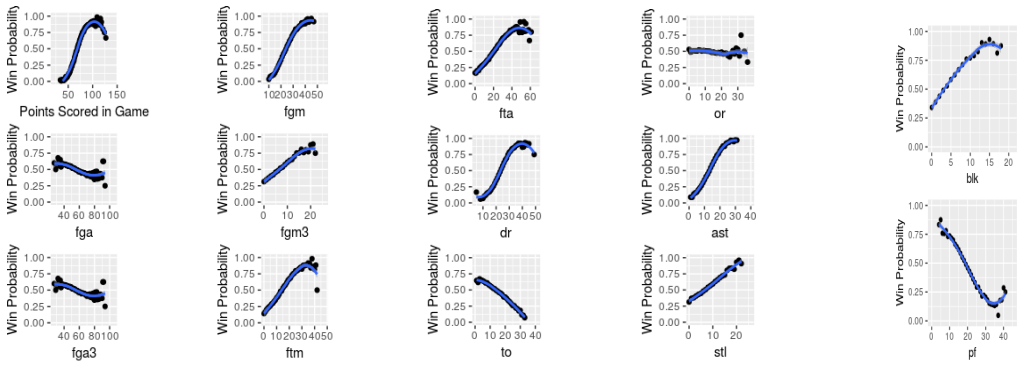


Now, digging into the relationship between points scored vs. probability of winning, we can see that there is a point in which teams can increase the chance of winning. The pinnacle would be between 80-100. After that, the chance of winning slowly drops. So from this, we can see that "high points scored" doesn't necessarily guarantee "winning".

Winning Probability by Basketball Statistics

Also, by exploring different basketball statistics that could be engineered further down the analysis, we can figure out which statistics affect the result of the match. Game statistics are those that may contribute to the probability of winning. Points scores in game (pts), number of assists (ast), number of steals (stl) and number of blocks (blk) are some of the examples of "game statistics" in this report. For the purpose of exploration, we looked at 14 variables:

- pts: points scored in game
- fgm: field goals made
- fgm3: 3-point field goals made
- fga: field goals attempted
- fga3: 3-point field goals attempted
- ftm: free throws made
- fta: free throws attempted
- ast: number of assists
- or: number of offensive rebounds
- dr: number of defensive rebounds
- to: number of turn-over
- stl: number of steals
- blk: number of blocks
- pf: number of personal fouls



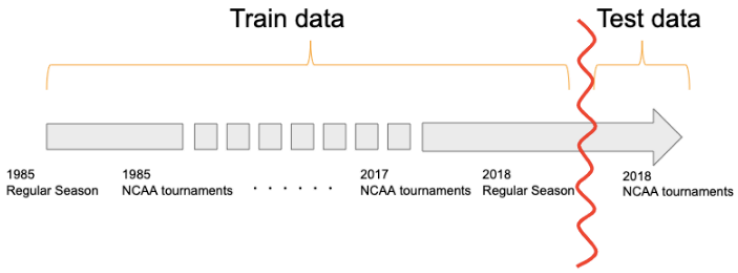
Chance of winning increases as more free throws are attempted (fta), more defensive rebounds (dr) are made, more assists (ast) are made, blocks (blk) are made in the game, steals (stl) are made, more field goals are made (fgm, fgm3) and free throws are made (ftm). However, we can see that *attempts* don't help in the games. It is the accuracy of making the goals that matters as any *attempts* give the opponent the chance to rebound (possibly offensive one). Therefore, those teams who have the most accurate scorer in the team would most likely have higher chances of winning. Moreover, since assists and rebounds take enormous team work, we can infer that top teams that actually make it to the March Madness would have very constructive team.

Data Modeling

Our goal is to get the most accurate predictions of NCAA tournament results in year 2018. To achieve this goal, we tried several machine learning methods, such as logistic regression, random forest, LDA and so on. Data modeling was divided into 3 parts: "Data Cleaning", "Feature Engineering", and "Statistical Modeling".

Data Cleaning

As we had already discussed, this project used many datasets and it was necessary to merge them and extract important features. First let's look through the dataset. The followings are the files we used. The data regarding the matchup before NCAA tournaments in 2018 was used as train data and the one of NCAA tournaments in 2018 was used as test data.



(Train data)

- RegularSeasonCompactResults_Prelim2018.csv: Game-by-game results for many *seasons* of historical data, starting with the 1985 season. For each season, the file includes all games played from daynum 0 through 132. It is important to realize that the "Regular Season" games are simply defined to be all games played on DayNum=132 or earlier (DayNum=132 is Selection Sunday)
- NCAATourneyCompactResults.csv: Game-by-game *NCAA tournament* results for all seasons of historical data since 1985 until 2017.
- RegularSeasonDetailedResults.csv: Team-level box scores for many *regular seasons* of historical data, starting with the 2003 season.
- NCAATourneyDetailedResults.csv: Team-level box scores for many *NCAA tournaments*, starting with the 2003 season until 2017
- MasseyOrdinals_Prelim2018.csv: Rankings (e.g. #1, #2, #3, …, #N) of teams from 2002-2003 *season* to 2018, under a large number of different ranking system methodologies.

- Players_XXXX.csv: Players list from 2010 to 2018. Each player is assigned to team they belonged to.
- Events_XXXX.csv: Play-by-play event logs for almost all games from that season. Elapsed seconds, event type, points are recorded. This data does not include NCAA tournament 2018.

(Test data)

- 2018NCAATourneyCompactResults.csv: Game-by-game *NCAA tournament* results for 2018.
- 2018NCAATourneyDetailedResults.csv: Team-level box scores for *NCAA tournament* 2018

Because Player_XXXX dataset had some strange value: "TEAM", we excluded this from data.

Player Score

We struggled dealing with Player_XXXX data and Event_XXXX data because other data was based on every matchup but these data was based on the player and event. There were two solutions to solve this problem.

1. Aggregate events by each match and merge with other data, using each matchup as ID.
2. Give scores to each event, and aggregate events' score by player and year. Then we can get each player's score in each year. Finally, aggregate players' score by their belonging team.

We thought choice 1 was too overfitting with each matchup and was lack of power to express the ranking of each team in general. Moreover, it was almost identical with Wscores and Lscores in the Compact results dataset. So, we decide to use choice 2 to handle Player and Event data. First, we set scores to each event like below.

- +1: assist, block, steal, reb_off, reb_def, reb_dead, made1_free, made2_tip, made2_dunk, made2_lay, made2_jump, made3_jump
- -1: turnover, foul_pers, foul_teach, miss1_free
- 0: timeout, timeout_tv, sub_in, sub_out, miss2_dunk, miss2_tip, miss2_lay, miss2_jump, miss3_jump

They are all based on our own subjective criteria. So, there might be room for improvement.

(Step 1) Give scores to each event, and aggregate events by player and year (example provided below)

```
####get the score of each player every year####
##give the score to the player in the following way##
player_score=function(Events,Players){
  events=dplyr::full_join(Events,Players,
by=c("Season","EventPlayerID"="PlayerID","EventTeamID"="TeamID"))
  events$events_score=ifelse(events$EventType%in%
c("assist","block","steal","reb_off","reb_def","reb_dead","made1_free",
"made2_dunk","made2_tip","made2_lay","made2_jump","made3_jump"),1, ifelse(events$EventType%in%
c("turnover","foul_pers","foul_tech","miss1_free"),-1,0))
  events=
    group_by(events,PlayerName)%>%
    summarise(TotalScore=sum(events_score))
  return(events)
}
events2010=player_score(Events_2010,Players_2010)
events2011=player_score(Events_2011,Players_2011)

#merge Players file and events
player_score_2010=dplyr::
  full_join(Players_2010,events2010,by="PlayerName")
player_score_2011=dplyr::
  full_join(Players_2011,events2011,by="PlayerName")
```

(Step 2) Aggregate players' score by their belonging team (example provided below)

```
#get the average player score of each team for every year
team_score_2010=group_by(player_score_2010,TeamID)%>%
  summarise(Teamscore=sum(TotalScore))
```

```
team_score_2010$Season=2010
team_score_2010=team_score_2010[-dim(team_score_2010)[1],]


team_score_2011=group_by(player_score_2011,TeamID)%>%
  summarise(Teamscore=sum(TotalScore))
team_score_2011$Season=2011
team_score_2011=team_score_2011[-dim(team_score_2011)[1],]

team_score=rbind(team_score_2010,team_score_2011)
```

Merge data

Now the data are separated and so we need to combine them. First RegularSeasonCompactResults and NCAATourneyCompactResults were merged vertically because they had same column and no overlap. Merged data was named Compact_train. Likewise, RegularSeasonDetailedResults and NCAATourneyDetailedResults were combined and merged data was named Details_train. Then Compact_train and Details_train were joined. Please note that Compact_train and Detail_train do not necessarily have the same day number because Details_train starts from 2003 whereas Compact_train starts with 1985. This is why we used full join in this case to keep the all rows in both datasets. We did the same procedure to test data. We also changed MasseyOrdinals data from long data to wide data because it made it easier to combine with train data (Result_data). We merged twice for winner's rate and loser's rate. Consequently, Column 70T.x~ZAM.x is the rate for WTeam (Winning Team) and column 70T.y~ZAM.y is the rate for LTeam (Losing Team). We wanted to use MasseyOrdinals for test data too but it didn't have rate data for NCAA tournament in 2018. So, we decided to use the rate of DayNum=133 in 2018 as the rate for NCAA tournament in 2018.

Feature Engineering

Part 1

We assumed that adding new features would provide a deeper understanding of a team's performance. That's why we used indexes that were often used for analyzing basketball game. We refered to Kaggle[3] to get the new feature. How we calculated each statistic is shown below.

```
#Points Winning/Losing Team
train$WPts=2*train$WFGM+train$WFGM3+train$WFTM
train$LPts=2*train$LFGM+train$LFGM3+train$LFTM
train$Pts_diff=train$WPts-train$LPts


#Calculate Winning/losing Team Possesion Feature
wPos=train$WFGA+train$WTO+0.44*train$WFTA-train$WOR
lPos=train$LFGA+train$LTO+0.44*train$LFTA-train$LOR
train$Pos_diff=train$WFGA-train$LFGA


#two teams use almost the same number of possessions in a game
#(plus/minus one or two - depending on how quarters end)
#so let's just take the average
train$Pos=(wPos+lPos)/2


#Offensive efficiency (OffRtg) = 100 x (Points / Possessions)
train$WOffRtg=100*(train$WPts/train$Pos)
train$LOffRtg=100*(train$LPts/train$Pos)
train$Off_diff=train$WOffRtg-train$LOffRtg


#Offensive efficiency (OffRtg) = 100 x (Points / Possessions)
train$WDefRtg = train$LOffRtg
train$LDefRtg = train$WOffRtg


#Net Rating = Off.Rtg - Def.Rtg
```

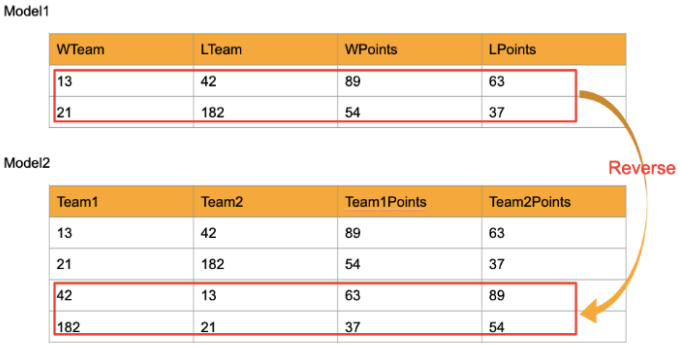
```
train$WNetRtg=train$WOffRtg-train$WDefRtg
train$LNetRtg=train$LOffRtg-train$LDefRtg
train$Net_diff=train$WNetRtg-train$LNetRtg


#Assist Ratio : Percentage of team possessions that end in assists
train$WAstR=100*train$WAst/(train$WFGA + 0.44*train$WFTA+ train$WAst + train$WTO)
train$LAstR=100*train$LAst/(train$LFGA + 0.44*train$LFTA+ train$LAst + train$LTO)
train$AstR_diff=train$WAstR-train$LAstR


#Turnover Ratio: Number of turnovers of a team per 100 possessions used.
#(TO * 100) / (FGA + (FTA * 0.44) + AST + TO)
train$WTOR=100 * train$WTO / (train$WFGA + 0.44*train$WFTA + train$WAst + train$WTO)
train$LTOR=100 * train$LTO / (train$LFGA + 0.44*train$LFTA + train$LAst + train$LTO)
train$TOR_diff=train$WTOR-train$LTOR


#The Shooting Percentage : Measure of Shooting Efficiency (FGA/FGA3, FTA)
train$WTSP=100 * train$WPts / (2 * (train$WFGA + 0.44*train$WFTA))
train$LTSP=100 * train$LPts / (2 * (train$LFGA + 0.44*train$LFTA))
train$TSP_diff=train$WTSP-train$LTSP


#eFG% : Effective Field Goal Percentage adjusting for the fact that 3pt shots are more valuable
train$WeFGP=(train$WFGM + 0.5 *train$WFGM3) / train$WFGA
train$LeFGP=(train$LFGM + 0.5 *train$LFGM3) / train$LFGA
train$eFGP_diff=train$WeFGP-train$LeFGP


#FTA Rate : How good a team is at drawing fouls.
train$WFTAR = train$WFTA / train$WFGA
train$LFTAR = train$LFTA / train$LFGA
train$FTAR_diff=train$WFTAR-train$LFTAR


#OREB% : Percentage of team offensive rebounds
train$WORP = train$WOR / (train$WOR + train$LDR)
train$LORP = train$LOR / (train$WOR + train$LDR)
train$ORP_diff=train$WORP-train$LORP


#DREB% : Percentage of team defensive rebounds
train$WDRP = train$WDR / (train$WDR + train$LOR )
train$LDRP=train$LDR / (train$LDR + train$WOR )
train$DRP_diff=train$WDRP-train$LDRP


#REB% : Percentage of team total rebounds
train$WRP=(train$WDR + train$WOR) / (train$WDR + train$WOR + train$LDR + train$LOR)
train$LRP=(train$LDR + train$LOR) / (train$WDR + train$WOR + train$LDR + train$LOR)
train$RP_diff=train$WRP-train$LRP
```

Because the rate variables were too many, we decided to use only average of them.

```
# use the average ranking
train$Wranking=apply(train[,35:198],1,function(x) mean(x,na.rm=TRUE))
train$Lranking=apply(train[,199:362],1,function(x) mean(x,na.rm=TRUE))
train=train[,c(-9:-362)]
colnames(train)[7]="Loc"
sub_train=train
```

Part 2

Now we have the data like model1 below. This seemed to be good enough to analyze data but we tried to enrich data. If we reverse the winning team and losing team and add this data to the existing data, this will make the data more robust because this will allow for each variable to take more variation. Moreover, in the model1, the outcome is all the same among all samples (i.e. outcome is all win (or lose)) and this will cause the model to return only one value. By enriching data, we were able to avoid those problems.




```
#In this case, winner team is named as TeamID1 and the result is 1
train_1=train
colnames(train_1)[3]="TeamID1"
colnames(train_1)[5]="TeamID2"
colnames(train_1)[4]="Team1_score"
colnames(train_1)[6]="Team2_score"
for(i in c(9,14,17,19,22,25,28,31,34,37,40,43,46)){
  colnames(train_1)[i]=paste0("Team1_",substr(colnames(train_1)[i],
                                              2,nchar(colnames(train_1)[i])))
}
for(i in c(10,15,18,20,23,26,29,32,35,38,41,44,47)){
  colnames(train_1)[i]=paste0("Team2_",substr(colnames(train_1)[i],
                                              2,nchar(colnames(train_1)[i])))
}
winners=train_1
winners$Result=1.0

#In this case, winner team is named as TeamID1 and the result is 1
train_2=train
colnames(train_2)[3]="TeamID2"
colnames(train_2)[5]="TeamID1"
colnames(train_2)[4]="Team2_score"
colnames(train_2)[6]="Team1_score"
for(i in c(9,14,17,19,22,25,28,31,34,37,40,43,46)){
  colnames(train_2)[i]=paste0("Team2_",
                              substr(colnames(train_2)[i],
                                     2,nchar(colnames(train_2)[i])))
}
for(i in c(10,15,18,20,23,26,29,32,35,38,41,44,47)){
  colnames(train_2)[i]=paste0("Team1_",
                              substr(colnames(train_2)[i],
                                     2,nchar(colnames(train_2)[i])))
}
train_2[,c(11,12,16,21,24,27,30,33,36,39,42,45)]=-
  train_2[,c(11,12,16,21,24,27,30,33,36,39,42,45)]
train_2$Loc=ifelse(train_2$Loc=="A","H","A")
losers=train_2
losers$Result=0.0

#Combine them
train=rbind(winners,losers)
```

Next, we combined train data we made with team_score data which we made at player score section.

```
#Combine with team score
train=dplyr::left_join(train,team_score,by=c("Season","TeamID1"="TeamID"))
train=dplyr::left_join(train,team_score,by=c("Season","TeamID2"="TeamID"))

#Drop the variables we don't use
train_X=train[,c(-2,-4,-6,-8,-48)]
train_y=train[,48]
colnames(train_X)[44]="player_score_1"
colnames(train_X)[45]="player_score_2"
```

Part 3

Let's move onto test data. We wanted to make features of test data but we had to be careful when making them. We can't use the features we made in part1 because it will cause data leakage. For example, we can easily know that it's impossible to get the details of matchup, such as winning/ losing points and winning/losing team possession before the match is actually done. So we have to delete the variables that we can't know before the matchup actually starts.

```
#use the average ranking
test$Wranking=apply(test[,35:198],1,function(x) mean(x,na.rm=TRUE))
test$Lranking=apply(test[,199:362],1,function(x) mean(x,na.rm=TRUE))
# remove the value from WFGM~LPF from test data because otherwise it will cause data leakage
test=test[,c(-8:-362)]
```

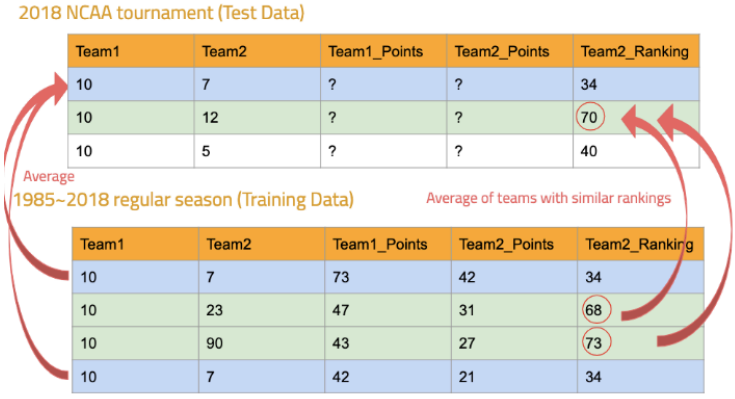
However, we wanted to use the expected values of variables like winning/ losing points and winning/losing team possession as predictors. How can we get the expected value for test data? We developed some algorithm like below.

1. when there was the same match up before

Let's illustrate the following table. The first row in test data is Team10 vs Team7. If we look at the train data, we can find the same matchup in train data. In that case, we take the average of same matchup in training data for every feature like poins, possesion, offensive efficiency and so on. This idea is based on we can expect the same values as the previous same match.

2. when there was no same matchup before

Then how about when there was no same matchup before? In that case, we take an average of features of teams with similar ranking in train data. In the below table, the matchup Team 10 vs Team12 didn't occur in train data and so we took the average of Team 10 vs Team 23 and Team10 vs Team 90 because the opponents' ranking is similar with Team12's ranking.




```
library(class)
#Get the mean of every feature calculated from the previous same pair of match
#If there was not the exactly the same match, we calculate the mean of similar match

create_train_feature=function(dat,colum){
    #Get the average of same matchup
    a=dat%>%group_by(TeamID1,TeamID2)%>%summarise(
      mean=mean(.data[[colnames(dat)[colum]]],na.rm=TRUE))
    #change the colum name
```

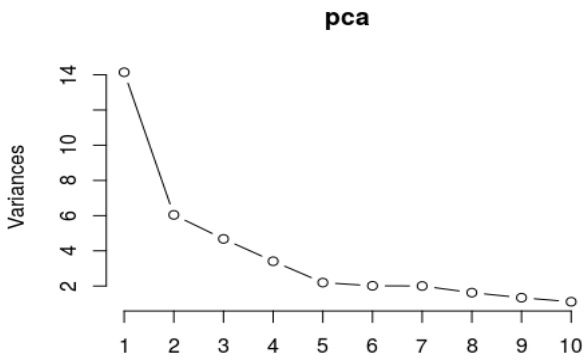
```
    colnames(a)[3]=colnames(sub_train)[colum+4]
    #combine with test data
    test=dplyr::left_join(test,a,by=c("WTeamID"="TeamID1","LTeamID"="TeamID2"))
    #specify where they don't have the same matchup in train data
    naindex=which(is.na(test[colnames(sub_train)[colum+4]]))
    #Losing teamID of this matchup
    LTeamID=test$LTeamID[naindex]
    #Wining teamID of this matchup
    WTeamID=test$WTeamID[naindex]
    #Get the ranking of losing team
    Lranking=sapply(LTeamID, function(x) mean(dat$Team2_ranking[dat$TeamID2==x],na.rm=TRUE))
    knn_target=rep(NA,length(LTeamID))
    count=1
    for(i in WTeamID){
      #list of the team that Team1 had battled in train data
      ranking_list=dat[dat$TeamID1==i,"Team2_ranking"]
      #list of the value of desired variable that Team1 scored in train data
      corresponding_target=dat[dat$TeamID1==i,colnames(dat)[colum]]
      #make the dataframe
      train=data.frame(rank=ranking_list,target=corresponding_target)
      train=drop_na(train)
      #use the k nearest neighbors to get the average of features of teams with similar ranking in
train data
      knn_target[count]=knn(train=train$rank, test =Lranking[count] , cl = train$target, k = 5)
      count=count+1
    }
    count=1
    for(i in naindex){
      test[i,colnames(sub_train)[colum+4]]=knn_target[count]
      count=count+1
    }
    return(test)
}

#apply create_test_feature to the test data
for(i in 5:41){
    test=create_train_feature(dat=train_X,colum =i)
  }
```

We applied the same process as part 2 to test data and enriched the data.

```
#combine testdata with team_score
  test=dplyr::left_join(test,team_score,
                        by=c("Season","WTeamID"="TeamID"))
  test=dplyr::left_join(test,team_score,
                        by=c("Season","LTeamID"="TeamID"))
  test=test[,c(-2,-4,-6)]
  colnames(test)[4]="Loc"

  winner=test
  colnames(winner)[2]="TeamID1"
  colnames(winner)[3]="TeamID2"

  for(i in c(5,7,12,15,17,20,23,26,29,32,35,38,41)){
    colnames(winner)[i]=paste0("Team1_",substr(colnames(winner)[i],
                                               2,nchar(colnames(winner)[i])))
  }
  for(i in c(6,8,13,16,18,21,24,27,30,33,36,39,42)){
    colnames(winner)[i]=paste0("Team2_",substr(colnames(winner)[i],
                                               2,nchar(colnames(winner)[i])))
  }
  winner$Result=1.0
```

```
  loser=test
  colnames(loser)[2]="TeamID2"
  colnames(loser)[3]="TeamID1"
  for(i in c(5,7,12,15,17,20,23,26,29,32,35,38,41)){
    colnames(loser)[i]=paste0("Team2_",substr(colnames(loser)[i],
                                              2,nchar(colnames(loser)[i])))
  }
  for(i in c(6,8,13,16,18,21,24,27,30,33,36,39,42)){
    colnames(loser)[i]=paste0("Team1_",substr(colnames(loser)[i],
                                              2,nchar(colnames(loser)[i])))
  }
  loser[,c(9,10,14,19,22,25,28,31,34,37,40,43)]= -winner[,c(9,10,14,19,22,25,28,31,34,37,40,43)]
  loser$Loc=ifelse(loser$Loc=="A","H","A")
  loser$Result=0.0
  test=rbind(winner,loser)
  colnames(test)[44]="player_score_1"
  colnames(test)[45]="player_score_2"
  test_X=test[,-46]
  test_y=test[,46]
  test_X=test_X[,c(1:4,7:43,5,6,44,45)]

test_X$Team_ranking_diff=
  test_X$Team1_ranking-test_X$Team2_ranking
test_X$player_score_diff=
  test_X$player_score_1-test_X$player_score_2
train_X$Team_ranking_diff=
  train_X$Team1_ranking-train_X$Team2_ranking
train_X$player_score_diff=
  train_X$player_score_1-train_X$player_score_2
```

Part 4



Finally, we tried PCA because the data has many variables and might cause collinearity. We first extracted the numerical variables and then performed PCA. As you can see in the plot, most of the variation was explained after 5th components. We used different number of components in the following Statistical Modeling part and it turned out that there was not much big difference about the choice of number of components after 5.

```
  train_pca=pca$x[,1:9]
  test_pca <- predict(pca, newdata =test_final_for_pca )
  test_pca=data.frame(test_pca[,1:9])
```

Statistical Modeling

Data processing was hard for this dataset and during processing, we used knn to get the adequate predictors in test data. So we can say that data process itself is the statistical modeling and has the power to predict the result of matchup in NCAA tournament in 2018. That is, points difference we predicted between Team1 and Team2 is itself the good indicator of the result. If it's more than zero, it means that Team1 is winning team and if it's less than zero, Team2 is winning team. The accuracy of this simple model was 86.56% and this was quite good.

```
#Pts
  result_from_pts=ifelse(test_final$Pts_diff>0,1,0)
```

Predicted results	0	1
0	58	9
1	9	58

We also tried logistic regression, Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), Random Forest and Neural Network to solve this classification problem (win or lose) and wanted to see applying other statistical methods improved the accuracy or not.

First, we used logistic regression. Accuracy was 82.09%.

```
  #logistic regression
  fit=glm(y~.,data = train_pca,family = binomial(link=logit))
```

Predicted results	0	1
0	55	12
1	12	55

Next, we used LDA, whose accuracy was 80.59%.

```
  #LDA
  dig.lda=lda(train_pca[,1:9],y)
```

Predicted results	0	1
0	54	13
1	13	54

QDA was also applied but it didn't work well.

```
  #QDA
  dig.qda=qda(train_pca[,1:9],y)
```

Predicted results	0	1
0	42	25
1	25	42

Random forest returned the same accuracy as simple model using only points difference. We referred to lecture notes and set the parameter. Number of tree is 1500, which is large enough. Number of variables considered at each split is 3 based on the criteria $\sqrt{p}$. Node size is 1 because this problem is classification. The mean accuracy was 86.57% which looks strong.

```
  #Random Forest
  rf.fit = randomForest(train_final_for_pca, as.factor(y), ntree = 1500, mtry = 5, nodesize = 1,
sampsize = 500)
```

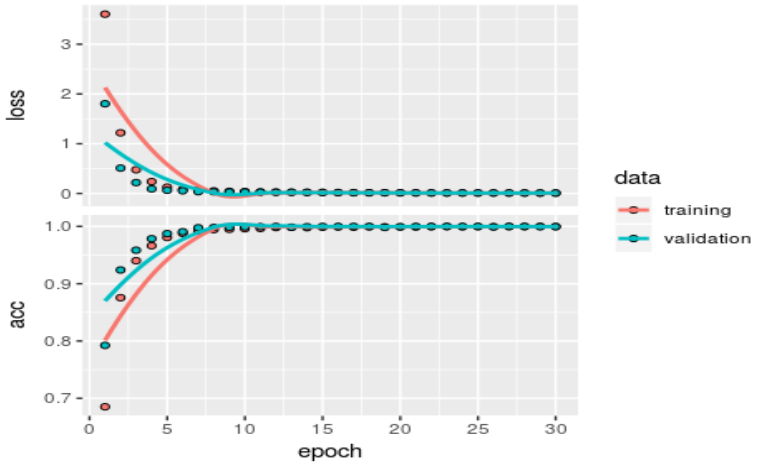
Predicted results	0	1
0	58	9
1	9	58

We also tried Neural Network, especially fully connected layer and drop out. Activation function was relu for the first layer and sigmoid for output layer. Unit size was 32 for the first layer and 1 for output layer. We also used l2 kernel regularizer to avoid overfit.

```
#Neural Network
train_mean=apply(train_final_for_pca, 2, FUN=mean)
train_for_neural=scale(train_final_for_pca,center = train_mean, scale = FALSE)
test_for_neural=scale(test_final_for_pca,center = train_mean, scale = FALSE)

library(keras)
k_clear_session()
model <- keras_model_sequential() %>%
    layer_dense(units = 32, activation = "relu", kernel_regularizer = regularizer_l2(0.001),
                input_shape = dim(train_for_neural)[2]) %>%
    layer_dropout(rate = 0.5) %>%
    layer_dense(units = 1,kernel_regularizer = regularizer_l2(0.001), activation = "sigmoid")

model %>% compile(
    optimizer = optimizer_rmsprop(lr=0.001),
    loss = "binary_crossentropy",
    metrics = c("accuracy")
)
```

We used 20% of train data as validation data and 80% as train data. The accuracy was 76.12% and this was not as good as logistic regression and random forest but there might be room for improvement by changing the architecture of network.

```
set.seed(100)

index <- sample(dim(train_for_neural)[1],10000,replace = FALSE)
x_val=train_for_neural[index,]
x_train=train_for_neural[-index,]

y_val <- y[index]
y_train <- y[-index]

num_epochs <- 30
history=model %>% fit(x_train, y_train,
                epochs = num_epochs, batch_size = 128,validation_split = 0.2)
```




```
results <- model %>% evaluate(test_for_neural, test_final[,48])
results

## $loss
## [1] 3.582345
##
## $acc
## [1] 0.7537314
```

Predicted results	0	1
0	51	17
1	16	50

Conclusion

The best accuracy was achieved by both simple models using Pts difference: point difference between winning team and losing team and random forest. We can consider that this first one is part of the random forest. It is equivalent to the model where only points difference was considered at split in random forest. Consequently, it turned that our algorithm that predicted the expected value of points difference performed very well.

From this analysis, we found the model that could possibly predict the winning teams and the losing teams. Something that we can improve with this analysis is to actually incorporate *seeding* variable (as that is important aspect to consider to see who plays with who) to possibly predict who the real winner was after running through real simulations. At this point, we can see *who* might win.

Sources Cited

[1] https://www.wikipedia.org/wiki/NCAA_Division_I_Men's_Basketball_Tournament

[2] https://www.kaggle.com/c/mens-machine-learning-competition-2018/data

[3] https://www.kaggle.com/lnatml/feature-engineering-with-advanced-stats