NBA Positions

Matt Kalin

7/2/2019

Introduction

There are five positions in basketball: point guard, shooting guard, small forward, power forward and center. Each position has different roles and responsiblities, so players from different positions will, at least in theory, accumulate different statistics throughout the course of a season. For example, a point guard would get a lot of assists, a shooting guard would attempt a lot of 3 pointers, and a center would get a lot of rebounds. Some players, however, have skills that transcend positions. For example Ben Simmons is a player on my favorite NBA team, the Philadelphia 76ers. Simmons is 6'10, which is tall even by NBA standards, but he plays point guard, which is typically the shortest player on the team. I was wondering if I could train a model that would classify an NBA player's position based on his statistics for a given season.

```r
# packages, functions, constants
library(XML)
library(httr)
library(foreign)
library(nnet)
library(ggplot2)
library(reshape2)
espnPages = 12
bbrefPages = 30
bbrefHeaderRows = (1:4) * 21
seasons = 2019:2010
basketballPositions = c("PG", "SG", "SF", "PF", "C")
minGameApprox = 36
firstCountingStat = "FG"
lastCountingStat = "PTS"

ConvertFactorData = function(df){
  for(i in 1:ncol(df)){
    if(class(df[, i]) == "factor"){
      df[, i] = as.character(df[, i])
    }
    if(class(df[, i]) == "character"){
      tryCatch({
        df[, i] = as.numeric(df[, i])
      }, warning = function(w){
        # do nothing
      })
    }
  }
  return(df)
}
```

I scraped data from two different sites: BasketballReference.com (bbref) and ESPN for the 2010-2019 NBA seasons. I am going to use the box score stats from bbref. The positional listings of the NBA players are from ESPN. I joined the two datasets so I have all the box score stats and the players' positions in the same table.

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(stringr)
cleanupEspnName = function(x){
  # format example: LeBron JamesCLE
  # regular expression is: at the end of the string,  2-3 consecutive capital letters which might be separated by a slash
  str_replace(x, "([A-Z]{2,3}|/)+$", "")
}
bbref.data = data.frame()
espn.data = data.frame()
# pb = txtProgressBar(0, length(seasons), style = 3)
for(j in seasons){
  for(i in 1:bbrefPages){
    bbref.url = paste("https://www.basketball-reference.com/play-index/psl_finder.cgi?request=1&match=single&type=totals&per_minute_base=36&per_poss_base=100&lg_id=NBA&is_playoffs=N&year_min=", j, "&year_max=", j,
                      "&ccomp%5B1%5D=gt&cval%5B1%5D=500&franch_id=&season_start=1&season_end=-1&age_min=0&age_max=99&shoot_hand=&height_min=60&height_max=95&birth_country_is=Y&birth_country=&birth_state=&college_id=&draft_year=&is_active=&debut_yr_nba_start=&debut_yr_nba_end=&is_hof=&is_as=&as_comp=gt&as_val=0&award=&pos_is_g=Y&pos_is_gf=Y&pos_is_f=Y&pos_is_fg=Y&pos_is_fc=Y&pos_is_c=Y&pos_is_cf=Y&qual=&c1stat=&c1comp=&c1val=&c2stat=&c2comp=&c2val=&c3stat=&c3comp=&c3val=&c4stat=&c4comp=&c4val=&c5stat=&c5comp=&c6mult=&c6stat=&order_by=height&order_by_asc=&offset=", 20 * (i - 1), sep = "")
    tryCatch({
      df = readHTMLTable(rawToChar(GET(bbref.url)$content))$stats
      df = ConvertFactorData(df[-c(bbrefHeaderRows), ])
      df$Season = j
    }, error = function(e){
      df <<- data.frame() # blank df
    })
    bbref.data = rbind(bbref.data, df)

  }
  for(i in 1:espnPages){
    espn.url = paste("https://www.espn.com/nba/stats/player/_/season/",
                     j, "/seasontype/2/table/general/sort/avgMinutes/dir/desc",
                     "/page/", i, sep = "")
    espn.stats = readHTMLTable(rawToChar(GET(espn.url)$content))
    tryCatch({
      # player names are stored in the first list element, the rest of the table is in the second element
      df = ConvertFactorData(espn.stats[[2]])
      df$Name = cleanupEspnName(espn.stats[[1]][,2])
    }, error = function(e){
      df <<- data.frame()
    })
    if(nrow(df) == 0){
      break
    }
    df$Season = j
    espn.data = rbind(espn.data, df)
  }
  # setTxtProgressBar(pb, match(j, seasons))
}
# close(pb)
bbref.data$POS = NA
{
  # I had to change these strings in order to make the two datasets consistent with each  other so they can be joined
  espn.data$Name = espn.data$Name %>%
    str_replace_all("JJ", "J.J.") %>%
    str_replace_all("CJ", "C.J.") %>%
    str_replace_all("JR", "J.R.") %>%
    str_replace_all("C.J. McCollum", "CJ McCollum")
  bbref.data$Player = bbref.data$Player %>%
    str_replace_all("Š", "S") %>%
    str_replace_all("ć", "c")
} # name change
bbref.data = bbref.data %>%
  filter(nchar(Player) > 0)
for(i in 1:nrow(bbref.data)){
  name = bbref.data[i, "Player"]
  espn.index = which(espn.data$Name == name &
                       espn.data$Season == bbref.data[i, "Season"])
  if(length(espn.index) > 0){
    bbref.data[i, "POS"] = espn.data[espn.index, "POS"]
  }
}
missing.players = bbref.data[which(is.na(bbref.data$POS)), "Player"]
missing.freq = as.data.frame(table(missing.players))
position.data = bbref.data %>%
  filter(POS %in% basketballPositions) %>%
  mutate(Height = as.numeric(substr(Ht, 1, 1)) * 12 + as.numeric(substr(Ht, 3, nchar(Ht))))
head(position.data)
```

```
##   Rk          Player Season Age  Tm  Lg   Ht  G GS   MP  FG  FGA  2P 2PA 3P 3PA
## 1 12 Dewayne Dedmon   2019  29 ATL NBA 7-0 64 52 1609 259  526 176 309 83 217
## 2 13    Joel Embiid   2019  24 PHI NBA 7-0 64 64 2154 580 1199 501 936 79 263
## 3 18    Serge Ibaka   2019  29 TOR NBA 7-0 74 51 2010 464  877 415 708 49 169
## 4 19   Nikola Jokic   2019  23 DEN NBA 7-0 80 80 2504 616 1206 533 936 83 270
## 5 22       Alex Len   2019  25 ATL NBA 7-0 77 31 1544 320  648 246 444 74 204
## 6 23 Meyers Leonard   2019  26 POR NBA 7-0 61  2  878 132  242  82 131 50 111
##    FT FTA ORB DRB TRB AST STL BLK TOV  PF  PTS   FG%   2P%   3P%  eFG%   FT%
## 1  92 113 105 375 480  90  69  71  84 214  693 0.492 0.570 0.382 0.571 0.814
## 2 522 649 160 711 871 234  46 122 226 211 1761 0.484 0.535 0.300 0.517 0.804
## 3 135 177 156 445 601  99  29 103 114 211 1112 0.529 0.586 0.290 0.557 0.763
## 4 289 352 228 637 865 580 108  55 248 228 1604 0.511 0.569 0.307 0.545 0.821
## 5 140 216 158 266 424  86  27  69  97 200  854 0.494 0.554 0.363 0.551 0.648
## 6  43  51  49 184 233  75  13   9  43 105  357 0.545 0.626 0.450 0.649 0.843
##     TS% POS Height
## 1 0.602   C     84
## 2 0.593   C     84
## 3 0.582   C     84
## 4 0.589   C     84
## 5 0.575   C     84
## 6 0.675   C     84
```

I defined some helper functions to train and test the different classification algorithms.

```r
TrainModel = function(model.fn, formula, data, ...){
  m = model.fn(formula = formula, data = data, ...)
  return(m)
}
ClassPredict = function(model, input.data){
  output.data = predict(model, input.data)
  if(length(intersect(class(output.data), c("matrix", "array", "data.frame"))) > 0){
    output.class = numeric(length = length(nrow(output.data)))
    for (i in 1:nrow(output.data)) {
      output.class[i] = names(which.max(output.data[i, ]))
    }
    return(output.class)
  } else {
    return(output.data)
  }

}
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:httr':
##
##     progress
```

```r
CrossValidation = function(model.list, formula, data, k = 10){
  data.rows = 1:nrow(data)
  folds = createFolds(data.rows, k)
  # model.list is a list with the names of and pointers to different classification models
  data[, names(model.list)] = NA # initialize prediction cols
  all.test.data = data.frame()
  for (i in 1:k) {
    train.data = data[setdiff(data.rows, folds[[i]]), ]
    test.data = data[folds[[i]], ]
    for (m in names(model.list)) {
      model = TrainModel(model.list[[m]], formula, train.data) # gonna have to define wrapper functions for algorithms that require extra arguments
      test.data[, m] = ClassPredict(model, test.data)
    }
    all.test.data = rbind(all.test.data, test.data)
  }
  return(all.test.data)
}
```

I trained and tested several classification models using 10-fold cross validation: Multinomial Log-linear, Decision Tree, K-Nearest Neighbor (for k = 1, 3, 5, 10, 50, 200), and Random Forest. A KNN algorithm with k = 1 is uncommon, but I included it here because I thought maybe it would recognize one of the player's other seasons as its nearest neighbor, and choose the position from that season. Players rarely change positions between seasons, so in theory this would be a fairly accurate algorithm.

```r
counting.cols = match(firstCountingStat, names(position.data)):
  match(lastCountingStat, names(position.data))
for(i in counting.cols){
  position.data[, i] = position.data[, i] / position.data$MP * minGameApprox
  # adjust counting stats to be approximate per game stats
}
position.data$POS2 = relevel(as.factor(position.data$POS), ref = "SG") # relevel so shooting guard is the default position
colnames(position.data) = make.names(names(position.data))
BuildFormula = function(x, y){
  return(as.formula(paste(y, "~", paste0("`", x, "`", collapse = " + "))))
}
input.cols = (match("FG", names(position.data))):(match("TS.", names(position.data))) %>%
  setdiff(match("X3P.", names(position.data)))  %>% # eliminate 3P% because some players did not attempt any 3s and that data is missing
  append(match("Height", names(position.data)))
model.formula = BuildFormula(names(position.data)[input.cols], "POS2")
library(rpart)
rpartWrapper = function(formula, data){
  rpart(formula, data, method = "class")
}
library(nnet)
multinomWrapper = function(formula, data){
  multinom(formula, data, trace = FALSE)
}
KNN10Wrapper = function(formula, data){
  knn3(formula, data = data, k = 10)
}
KNN50Wrapper = function(formula, data){
  knn3(formula, data = data, k = 50)
}
KNN200Wrapper = function(formula, data){
  knn3(formula, data = data, k = 200)
}
KNN5Wrapper = function(formula, data){
  knn3(formula, data = data, k = 5)
}
KNN3Wrapper = function(formula, data){
  knn3(formula, data = data, k = 3)
}
KNN1Wrapper = function(formula, data){
  knn3(formula, data = data, k = 1)
}
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
model.list = list("Multinom" = multinomWrapper, "Decision Tree" = rpartWrapper, "knn1" = KNN1Wrapper, "knn3" = KNN3Wrapper, "knn5" = KNN5Wrapper, "knn10" = KNN10Wrapper, "knn50" = KNN50Wrapper, "knn200" = KNN200Wrapper, "Random Forest" = randomForest)
set.seed(9)
pred.data = CrossValidation(model.list, model.formula, position.data, k = 10)
pred.data = pred.data %>%
  select(Player, Season, Age, Tm, Ht, POS, names(model.list))
# set.seed(6)
pred.data %>%
  slice(sample(1:nrow(pred.data), 10))
```

```
##                   Player Season Age  Tm   Ht POS Multinom Decision Tree knn1
## 1         Brandon Ingram   2018  20 LAL  6-7  SF       SG            SF   SG
## 2         Tyson Chandler   2016  33 PHO  7-0   C        C             C    C
## 3          Dwight Howard   2011  25 ORL 6-10   C        C             C    C
## 4  Giannis Antetokounmpo   2015  20 MIL 6-11  PF        C             C    C
## 5              Ish Smith   2016  27 TOT  6-0  PG       PG            PG   PG
## 6         Rodions Kurucs   2019  20 BRK  6-9  SF       PF            PF   PF
## 7          Kawhi Leonard   2019  27 TOR  6-7  SF       SF            PF   SF
## 8           Brandon Bass   2015  29 BOS  6-8  PF       PF            PF   PF
## 9          Tobias Harris   2017  24 DET  6-8  SF       SF            SF   SF
## 10      Patrick Patterson   2014  24 TOT  6-8  PF       PF            PF    C
##    knn3 knn5 knn10 knn50 knn200 Random Forest
## 1    SG   SG    SG    SF     PF            SF
## 2     C    C     C     C      C             C
## 3     C    C     C     C      C             C
## 4     C    C     C     C      C             C
## 5    PG   PG    PG    PG     PG            PG
## 6     C   PF    PF    SF     SF            PF
## 7    PF   PF    SF    SF     PF            SF
## 8    PF   PF    PF    PF     PF            PF
## 9    PF   SF    SF    SF     SF            SF
## 10    C   PF    PF    SF     SF            PF
```

```r
accuracy.lst = model.list
for (m in names(model.list)) {
  accuracy.lst[[m]] = mean(pred.data$POS == pred.data[, m])
}
accuracy.lst = unlist(accuracy.lst)
accuracy.lst
```

```
##      Multinom Decision Tree          knn1          knn3          knn5
##     0.7391781     0.6893151     0.7435616     0.7468493     0.7315068
##         knn10         knn50        knn200 Random Forest
##     0.7232877     0.6898630     0.6241096     0.8016438
```

All of the classification models had a cross-validation accuracy of between 60% and 75%. The Random Forest performed the best with 74.3% accuracy. The k = 1 knn algorithm performed the worst, so my hunch turned out to be incorrect.

A lot of the time, different models make different classifications for a player's position. I was wondering if the "wisdom of the crowd" principle would apply here; where the classification picked by the most models would outperform any individual model.

```r
pred.data$Majority = NA
for (i in 1:nrow(pred.data)) {
  pred.data[i, "Majority"] = pred.data[i, names(model.list)] %>%
    unlist() %>%
    as.factor() %>%
    summary() %>%
    which.max() %>%
    names()
}
accuracy.lst["Majority"] = mean(pred.data$POS == pred.data$Majority)
accuracy.lst
```

```
##      Multinom Decision Tree          knn1          knn3          knn5
##     0.7391781     0.6893151     0.7435616     0.7468493     0.7315068
##         knn10         knn50        knn200 Random Forest      Majority
##     0.7232877     0.6898630     0.6241096     0.8016438     0.7561644
```

The Majority "model" outperformed most of the other models, but still fell short of the Random Forest in terms of classification accuracy.

```r
pred.data %>%
  filter(Player == "Ben Simmons")
```

```
##        Player Season Age  Tm   Ht POS Multinom Decision Tree knn1 knn3 knn5
## 1 Ben Simmons   2018  21 PHI 6-10  PG       PF             C   PG   PF    C
## 2 Ben Simmons   2019  22 PHI 6-10  PG       PF             C   PG   PF    C
##   knn10 knn50 knn200 Random Forest Majority
## 1    PF    PF     PF            PF       PF
## 2     C    PF     PF            PF        C
```

As I mentioned in the introduction, Ben Simmons is an outlier when it comes to NBA statistics. He plays point guard, but he is 6'10 and gets plenty of rebounds to go with his points and assists. As you can see, all of the models classified him as a center. I guess his height and rebound frequency (and lack of three point attempts) were enough to convince the computer that he is a center.