URSS: Bradley-Terry Models in Cricket

Peter Matthews

07/07/2021

Introduction

This project aims to investigate the Bradley-Terry class of models for pairwise comparisons, and to apply them to cricket. I aim to formulate and fit a model - using ball-by-ball data in ODI matches from 2018 to 2021 - that estimates the wicket-taking ability of each bowler and the wicket-saving ability of each batsman, and to formulate another model that estimates the run-scoring and run-saving ability of each batsman and bowler, respectively. Some cricket-specific extensions of the models I look at are to account for the different ease of taking wickets and scoring runs at various stages of the match; *incorporating home advantage; comparing a batsman's ability against seam vs against spin; allowing for abilities to vary smoothly over time; and allowing player abilities to vary in each venue*. The primary application I'm interested in are to form rankings of cricket players in a more principled manner to the ICC's official rankings, The paradigm can be used also to optimise the players to come in at each situation. Import the packages needed for the project: tidyverse; fs to deal with files; BradleyTerry2 and gnm for fitting Bradley-Terry models.

```
library(tidyverse)
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.0     v dplyr   1.0.5
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
library(fs)
library(BradleyTerry2)
library(gnm)
```

Data

The ODI Matches folder - downloaded from https://cricsheet.org/downloads/ - contains a csv file for each ODI played after March 2004 (Matches from 1st Jan 2018 are in "ODIs since 2018"). In each row in each file corresponds to 1 ball in the match, with the following attributes - match_id, season, start_date, venue, innings, ball, batting_team, bowling_team, striker, non_striker, bowler, runs_off_bat, extras, wides, noballs, byes, legbyes, penalty, wicket_type, player_dismissed, other_wicket_type, other_player_dismissed. We only care about matches since 2018 and are interested in striker, bowler, runs_off_bat, wicket_type. For now ignore extras and run outs but I'll keep innings and ball numbers to add as covariates later.

```
files <- fs::dir_ls("ODIs since 2018")
files <- map(files,
             function(path){
               match <- read.csv(path,
                        colClasses = c("numeric", "character", "Date", "character", "numeric",  "numeric",  "char
acter",    "character",    "character",    "character",    "character",    "integer",  "integer",  "integer",  "i
nteger",  "integer",  "integer",  "integer",  "character",    "character",    "character",    "character"
))
  match$bowler <- paste(match$bowler, "Bowl")
  match$striker <- paste(match$striker, "Bat")## have to specify classes so that R can correctly bind rows
  return(match)             })
full_data <- bind_rows(files)
model_data <- filter(full_data, extras == 0, wicket_type != "run out") ## Ignore balls with runs not credited to
 batsman and wickets not credited to bowler
model_data$wicket <- model_data$wicket_type != ""
#for innings phase


model_data <- model_data %>% select(striker, bowler, runs_off_bat, wicket, innings, ball) ##keeping innings and b
all numbers as covariates later, when that happens will have to process them more here
eligbowl <- model_data %>%                      ##filter for bowlers who have balled more than 150 balls
    group_by(bowler) %>%
    summarise(wickets = sum(wicket), survivals = sum(!wicket)) %>%
    filter(wickets > 10 & survivals > 250)

eligbat <- model_data %>%                     ## and batsmen who have faced more than 150
    group_by(striker) %>%
    summarise(wickets = sum(wicket), survivals = sum(!wicket)) %>%
    filter(wickets > 10 & survivals > 250)

output <- model_data %>% filter(bowler %in% eligbowl$bowler & striker %in% eligbat$striker)


head(model_data)
```

```
##           striker               bowler runs_off_bat wicket innings ball
## 1 H Masakadza Bat Shakib Al Hasan Bowl            1  FALSE       1  0.1
## 2   CR Ervine Bat Shakib Al Hasan Bowl            0  FALSE       1  0.3
## 3   CR Ervine Bat Shakib Al Hasan Bowl            0   TRUE       1  0.4
## 4  BRM Taylor Bat Shakib Al Hasan Bowl            2  FALSE       1  0.5
## 5  BRM Taylor Bat Shakib Al Hasan Bowl            0  FALSE       1  0.7
## 6  BRM Taylor Bat Shakib Al Hasan Bowl            0  FALSE       1  0.8
```

Data Cleaning for First Model

Relatively easy thing to get started, for now we can fit the wickets model as a simple Bradley-Terry with an order effect on bowlers (reflecting how hard it is to take a wicket, compared to just surviving one ball)

Here is a function to put the model data into the format that models can be fitted with. For the wicket model BradleyTerry2 "prefers" data to be summarised for each pair of batsman and bowler.

```
DataToBinomial <- function(model_data){
  binom <- model_data %>%
    group_by(bowler, striker) %>%
    summarise(wickets = sum(wicket), survivals = sum(!wicket))

  ## after this point could be reused when including covariates (need to change first part to group table differe
ntly)
  binom$bowl <- factor(binom$bowler, levels = unique(c(binom$bowler, binom$striker, "Average")))
  binom$bat <- factor(binom$striker, levels = unique(c(binom$bowler, binom$striker, "Average")))
  binom$bowl <- data.frame(player = binom$bowl, batting = 0)
  binom$bat <- data.frame(player = binom$bat, batting = 1)
  binom <- binom %>% select(bowl, bat, wickets, survivals)
  return(binom)
}

BTdata <- DataToBinomial(model_data)
```

```
## `summarise()` has grouped output by 'bowler'. You can override using the `.groups` argument.
```

```
## Adding missing grouping variables: `bowler`
```

Function to add a prior, with each player having bowled 38 balls to and faced 38 balls from "Average" player, with 1 wicket and 37 survivals (average not out balls per wicket across the dataset). Adding this made the model a lot more stable.

```
add_prior <- function(model_data, wickets = 1, survivals = 37){
  ## There is gonna be a much nicer purr implementation for doing it but this works for now
  bowl.prior = model_data[1: length(unique(model_data$bowl$player)),] ## creates a dataframe the right size to ad
d stuff to.
  bat.prior = model_data[1: length(unique(model_data$bat$player)), ]

  bowl.prior$bowler = unique(model_data$bowl$player)
  bowl.prior$bowl$player = unique(model_data$bowl$player)
  bowl.prior$bowl$batting = 0
  bowl.prior$bat$player = factor("Average")
  bowl.prior$bat$batting = 1
  bowl.prior$wickets = wickets
  bowl.prior$survivals = survivals

  bat.prior$bat$player = unique(model_data$bat$player)
  bat.prior$bowl$player = factor("Average")
  bat.prior$bowl$batting = 0
  bat.prior$bowler = factor("Average")
  bat.prior$bat$batting = 1
  bat.prior$wickets = wickets
  bat.prior$survivals = survivals

  return(rbind(model_data, bowl.prior, bat.prior))
}

BTdata <- add_prior(BTdata)
```

First Wicket-Based Model

The initial baby wicket model that just has an order effect ν for batting and no other covariates. Unlike the runs models it can be fitted just with BradleyTerry2, using maximum-likelihood-estimation and does not require gnm. Takes about 3 mins to run on my laptop. The coefficients calculated are log-abilities, so with $log(\mu_i)$ as the batting ability of player i and $log(\lambda_j)$ as the bowling ability of player j, the model is

$$logit(i\ survives\ j) = log(\nu) + log(\mu_i) - log(\lambda_j)$$

$$\mathbb{P}(i\ survives\ j) = \frac{\nu\mu_i}{\nu\mu_i + \lambda_j}$$

```
FirstWickModel <- BTm(outcome = cbind(wickets, survivals), player1 = bowl, player2 = bat,
data = BTdata, id = "player", formula = ~ player + batting, refcat = "Average")
```

The model seems to give semi reasonable results, a lot of players who don't bowl or bat that much rise to the tip so prior probably needs to be strengthened a bit. $log(\nu)\ = 3.967$ implies that the average batsman gets out to the average bowler once every 53 balls, which is pretty high but within the realms of what you would expect. Based on the results pretty sure the "average" bowler (log-ability = 0) of this model is comparitively worse than the true average bowler of the data.

```
sort(FirstWickModel$coefficients, decreasing = T) ##need to figure out how to split into batting and bowling
```

A hypothetical you could ask the model is long would Joe Root survive against himself?

```
nu = 3.967145852
bowl = 0.629774970
bat = 0.550226006  ##had to manually look these up because R doesn't like indexing things with spaces, need to fi
x.

## 1 / Probability of getting out each ball
1 + exp(nu + bat - bowl)
```

```
## [1] 49.79349
```

Wickets model with phase of match

The first extension to the model will split each innings into 10 over blocks as factors, (e.g. 1st Inn Overs 1-10, 1st Inn Overs 11-20, … , 2nd Inn Overs 41-50). There is a separate batting effect ν_k for each phase of the match, k. So the model becomes

$$logit(i\ survives\ j \mid phase\ k) = log(\nu_k) + log(\mu_i) - log(\lambda_j)$$

$$\mathbb{P}(i\ survives\ j \mid phase\ k) = \frac{\nu_k\mu_i}{\nu_k\mu_i + \lambda_j}$$

Below are functions to fit the second model.

```
DataToModel2 <- function(model_data){
  eligbowl <- model_data %>%                      ##filter for bowlers who have balled more than 150 balls
    group_by(bowler) %>%
    summarise(wickets = sum(wicket), survivals = sum(!wicket)) %>%
    filter(wickets > 10 & survivals > 150)

  eligbat <- model_data %>%                     ## and batsmen who have faced more than 150
    group_by(striker) %>%
    summarise(wickets = sum(wicket), survivals = sum(!wicket)) %>%
    filter(wickets > 10 & survivals > 150)

  output <- model_data %>% filter(bowler %in% eligbowl$bowler & striker %in% eligbat$striker)
  output$phase <- output$ball %/% 10 + 5*(output$innings - 1) + 1

  output <- output %>%                    ## and batsmen who have faced more than 150
    group_by(bowler, striker, phase) %>%
    summarise(wickets = sum(wicket), survivals = sum(!wicket))

  output$ball <- factor(output$bowler, levels = unique(c(output$bowler, output$striker, "Average")))
  output$bat <- factor(output$striker, levels = unique(c(output$bowler, output$striker, "Average")))
  output$ball <- data.frame(player = output$ball, p11 = 0, p12 = 0, p13 = 0, p14 = 0, p15 = 0, p21 = 0, p22 = 0,
 p23 = 0, p24 = 0, p25 = 0)


  output$bat <- data.frame(player = output$bat, "p11" = as.numeric(output$phase == 1), "p12" = as.numeric(output
$phase == 2), "p13" = as.numeric(output$phase == 3), "p14" = as.numeric(output$phase == 4), "p15" = as.numeric(ou
tput$phase == 5), "p21" = as.numeric(output$phase == 6), "p22" = as.numeric(output$phase == 7), "p23" = as.numeri
c(output$phase == 8), "p24" = as.numeric(output$phase == 9), "p25" = as.numeric(output$phase == 10))
  ## absolutely disgusting implementation, will try to find a better one

  output <- output %>% select(ball, bat, wickets, survivals)
  return(output)
}

M2Data <- DataToModel2(model_data)
```

```
## `summarise()` has grouped output by 'bowler', 'striker'. You can override using the `.groups` argument.
```

```
## Adding missing grouping variables: `bowler`, `striker`
```

Have to also set up a prior.

```
add_prior2 <- function(model_data, wickets = 2, survivals = 75){
  ## There is gonna be a much nicer purr implementation for doing it but this works for now
  bowl.prior = model_data[1: length(unique(model_data$ball$player)),] ## creates a dataframe the right size to ad
d stuff to.
  bat.prior = model_data[1: length(unique(model_data$bat$player)), ]

  bowl.prior$ball$player = unique(model_data$ball$player)

  bowl.prior$bat$player = factor("Average")
  bowl.prior$bat$p11 = 1
  bowl.prior$bat$p12 = 0
  bowl.prior$bat$p13 = 0
  bowl.prior$bat$p14 = 0
  bowl.prior$bat$p15 = 0
  bowl.prior$bat$p21 = 0
  bowl.prior$bat$p22 = 0
  bowl.prior$bat$p23 = 0
  bowl.prior$bat$p24 = 0
  bowl.prior$bat$p25 = 0
  bowl.prior$wickets = wickets
  bowl.prior$survivals = survivals

  bat.prior$bat$player = unique(model_data$bat$player)
  bat.prior$ball$player = factor("Average")

  bat.prior$bat$p11 = 1
  bat.prior$bat$p12 = 0
  bat.prior$bat$p13 = 0
  bat.prior$bat$p14 = 0
  bat.prior$bat$p15 = 0
  bat.prior$bat$p21 = 0
  bat.prior$bat$p22 = 0
  bat.prior$bat$p23 = 0
  bat.prior$bat$p24 = 0
  bat.prior$bat$p25 = 0
  bat.prior$wickets = wickets
  bat.prior$survivals = survivals

  output = bind_rows(model_data, bat.prior, bowl.prior)
  return(output)
}

M2Data1 <- add_prior2(M2Data)
```

```
WickModel2 <- BTm(outcome = cbind(wickets, survivals), player1 = ball, player2 = bat,
data = M2Data1, id = "player", formula = ~ player + p11 + p12 + p13 + p14 + p15 + p21 + p22 + p23 + p24 + p25, re
fcat = "Average")
```

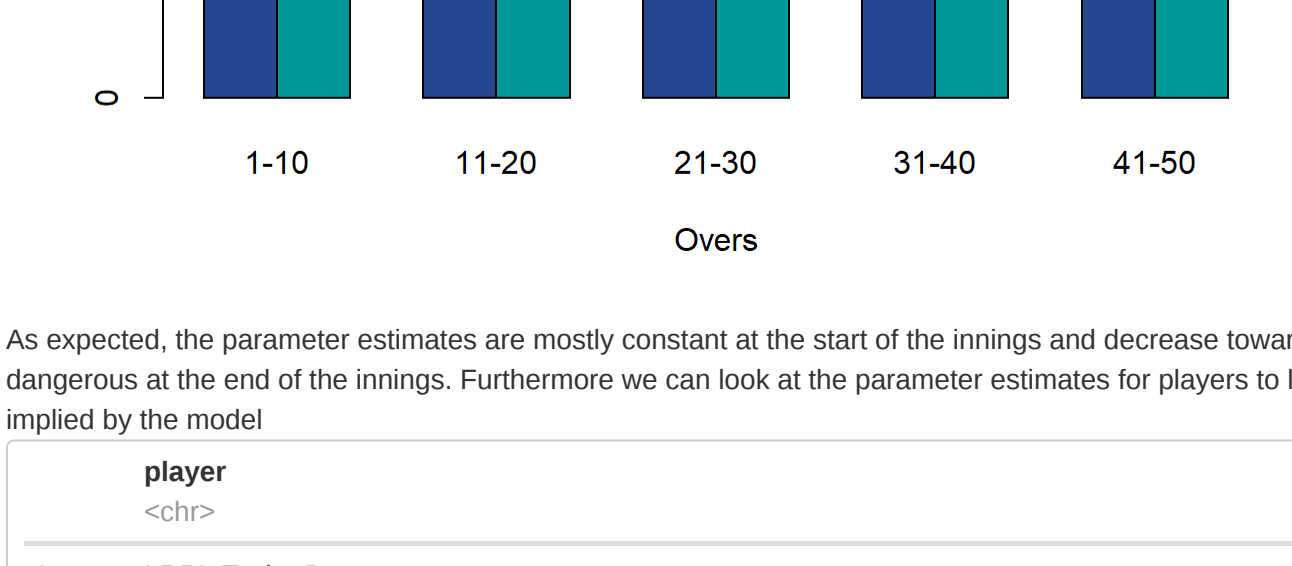
We re-normalise the estimated coefficients such that the average bating ability and average bowling ability are both 0, we can adjust ν_k estimates to compensate. Then we look ν_k estimates to see how they vary by match phase.

```
bowlers <- WickModel2$coefficients[str_detect(names(WickModel2$coefficients), "Bowl")]
batters <- WickModel2$coefficients[str_detect(names(WickModel2$coefficients), "Bat")]
nus <- WickModel2$coefficients[str_detect(names(WickModel2$coefficients), "player", negate = T)]

nus <- nus + mean(batters) - mean(bowlers)
bowlers <- bowlers - mean(bowlers)
batters <- batters - mean(batters)

nu.df <- data.frame(phase = c("1-10", "11-20", "21-30", "31-40", "41-50"), "1st Inning" = nus[1:5], "2nd Inning"
 = nus[6:10])

barplot(t(as.matrix(nu.df[ , 2:3])), beside = T, names.arg = nu.df$phase, main = bquote("Estimate of log(" ~ nu
[k] ~ ") at each phase of the innings"), xlab = "Overs", ylab = bquote("log(" ~ nu[k] ~ ")") , col = c("#24478f",
"#009999"))
legend(12.2, 4, legend = c("1st Innings", "2nd Innings"), fill = c("#24478f", "#009999"))
```



As expected, the parameter estimates are mostly constant at the start of the innings and decrease towards the end, suggesting that batting is more dangerous at the end of the innings. Furthermore we can look at the parameter estimates for players to look at the top batsmen and bowlers as implied by the model

	player <chr>	ability <dbl>
1	LRPL Taylor Bat	0.7564283
2	BA Stokes Bat	0.7308695
3	HE van der Dussen Bat	0.6968541
4	MS Dhoni Bat	0.6783668
5	Haris Sohail Bat	0.6048545
6	SD Hope Bat	0.5747622
7	Mahmudullah Bat	0.5679016
8	KM Jadhav Bat	0.5534316
9	AT Carey Bat	0.5237072
10	Imad Wasim Bat	0.5116972

1-10 of 10 rows

	player <chr>	ability <dbl>
1	Ehsan Khan Bowl	0.7559995
2	S Lamichhane Bowl	0.6243635
3	S Bhari Bowl	0.5566393
4	A Nortje Bowl	0.4363855
5	L Ngidi Bowl	0.4263604
6	JDS Neesham Bowl	0.3938948
7	Aqib Ilyas Bowl	0.3904172
8	N Pokana Bowl	0.3784554
9	LE Plunkett Bowl	0.3783778
10	JA Richardson Bowl	0.3774180

1-10 of 10 rows

Formulating a runs-model

The runs model is formulated in a similar way to (Hamilton 2018). Where the probability of scoring n runs off one ball is proportional to the batting ability to the power of n and the bowling ability to the power of -n

$$P(i\ scores\ 0\ from\ j) \propto \nu_0$$

$$P(i\ scores\ 1\ from\ j) \propto \nu_1\left(\frac{\mu_i}{\lambda_j}\right)$$

$$P(i\ scores\ 2\ from\ j) \propto \nu_2\left(\frac{\mu_i}{\lambda_j}\right)^2$$

$$\dots$$

$$P(i\ scores\ k\ from\ j) \propto \nu_k\left(\frac{\mu_i}{\lambda_j}\right)^k$$

where ν_k is a paramater that accounts for the relative likelihood of each run scoring outcome.

If N_{ijk} is the number of times that batsman i scores k runs off bowler j, then we also have

$$log\left(\mathbb{E}\left[N_{ijk}\right]\right) = log\left(\nu_k\right) + k\left[log(\mu_i) - log(\lambda_j)\right] + a_{ij}$$

where a_{ij} is a nuisance parameter that accounts for both the denominator and the number of balls faced.

So we can format it as a log-linear model and solve using gnm. For identifiability, ν_0 is set to 1 and $\nu_1, \nu_2, \nu_3, \nu_4, \nu_6$ are calculated. Whilst technically possible scoring 5, or 7+ runs off a ball is vanishingly rare so they are ignored.

This was my first attempt at fitting a runs model as an MLE from just the likelihood function, it takes way to long to run, hence the gnm implementation below. I'm keeping it here for posterity.

```
balllik <- function(params, data){

  bat = params["bat"]; one =params["one"]; two =params["two"]
  three = params["three"]; four = params["four"]; six =params["six"]

  bat2 <- data$runs_off_bat * bat
  dot <- as.numeric(data$runs_off_bat == 0)
  one2 <- one * (data$runs_off_bat == 1)
  two2 <- two * (data$runs_off_bat == 2)
  three2 <- three * (data$runs_off_bat == 3)
  four2 <- four * (data$runs_off_bat == 4)
  six2 <- six * (data$runs_off_bat == 6)

  dif <- params[data$striker] - params[data$bowler]

  return(sum( data$runs_off_bat * dif
          + bat2 + dot + one2 + two2 + three2 + four2 + six2 -
            log(1 + exp(one + bat + dif) + exp(two + 2 * (bat + dif))
           + exp(three + 3 * (bat + dif)) + exp(four + 4 * (bat + dif))
           + exp(six + 6 * (bat + dif))
                )))
}

ab0 <- numeric(length(unique(c(model_data$bowler, model_data$striker))) + 6)
names(ab0) <- c(unique(c(model_data$bowler, model_data$striker)), "bat", "one", "two", "three", "four", "six")

cnstrntl <- numeric(length(unique(c(model_data$bowler, model_data$striker))) + 6) - 4
cnstrntl[1] = 0
cnstrntl[length(cnstrntl) - 7] = 0
cnstrntu = -1 * cnstrntl

balllik(ab0, model_data)
```

```
## [1] -208287.5
```

```
runmod <- optim(ab0, balllik, method = "L-BFGS-B", control=list('fnscale'=-1), upper = cnstrntu,  lower = cnstrnt
l, data = model_data)

runmod
```

Here is the gnm implementation of the model that can be fit in a reasonable time (~10-15 mins). The first function creates a matrix where each column represents a players ability and each row vector is the contribution of each player to the model for that row of data. This (as shown above) is equal to the runs scored for the batter and runs conceeded for the bowler, with every other players contribution being zero.

```
makePlayerMatrix <- function(data){
  players <- unique(c(data$bowler, data$striker))
  n <- length(players)

  X <- matrix(0,
              nrow(data),
              n)

  colnames(X) <- players
  for(player in colnames(X)){
    X[data$bowler == player, player] = -1 * data$runs_off_bat[data$bowler == player]
    X[data$striker == player, player] = data$runs_off_bat[data$striker == player]
  }
  return(X)
}
```

This function formats the data in a way that can be fitted by gnm, an example of the format is shown below

```
DataToRunMod <- function(data){
  data <- select(data, -c(wicket, innings, ball))   ### drop the wickets for the runs model
  data <- data[data$runs_off_bat != 5 & data$runs_off_bat != 7, ] ##drop 5s and 7s as they are incredibly rare
  data <- expandCategorical(data, "runs_off_bat")
  data$one = as.numeric(data$runs_off_bat == 1)
  data$two = as.numeric(data$runs_off_bat == 2)
  data$three = as.numeric(data$runs_off_bat == 3)
  data$four = as.numeric(data$runs_off_bat == 4)
  data$six = as.numeric(data$runs_off_bat == 6)
  data$runs_off_bat = as.numeric(as.character(data$runs_off_bat))
  data$Players = makePlayerMatrix(data)
  return(data)
}

RunData <- DataToRunMod(model_data)

rmarkdown::paged_table(head(RunData, 5))
```

Here, count is N_{ijk} and id is a factor that can be fitted as the a_{ij} parameter.

```
RunMod <- gnm(count ~  Players + one + two + three + four + six, eliminate = id, family = "poisson", data = RunDa
ta)
```

Then, like the wicket model we normalise the abilities and adjust the nu_k to maintain the same predictions

```
battersRun <- RunMod$coefficients[str_detect(names(RunMod$coefficients), "Bat")]
battersRun <- battersRun[!is.na(battersRun)]
bowlersRun <- RunMod$coefficients[str_detect(names(RunMod$coefficients), "Bowl")]
bowlersRun <- bowlersRun[!is.na(bowlersRun)]

RunNus <- c(RunMod$coefficients["one"], RunMod$coefficients["two"], RunMod$coefficients["three"],
            RunMod$coefficients["four"], RunMod$coefficients["six"])
RunNus[is.na(RunNus)] = 0

batAdvantage <- mean(battersRun) - mean(bowlersRun)
for(i in 1:5){RunNus[i] = RunNus[i] + i * batAdvantage}
RunNus["six"] <- RunNus["six"] + batAdvantage

battersRun = battersRun - mean(battersRun)
bowlersRun = bowlersRun - mean(bowlersRun)
```

```
batr.df = data.frame(player = str_remove(names(battersRun), "Players"), ability = battersRun)
bowlr.df = data.frame(player = str_remove(names(bowlersRun), "Players"), ability = bowlersRun)
batr.df <- batr.df[order(batr.df$ability, decreasing = T), ]
bowlr.df <- bowlr.df[order(bowlr.df$ability, decreasing = T), ]
rownames(batr.df) <- 1:length(battersRun)
rownames(bowlr.df) <- 1:length(bowlersRun)

rmarkdown::paged_table(head(batr.df, 10))
```

```
rmarkdown::paged_table(head(bowlr.df, 10))
```

```
bowlabilities <- inner_join(bowl.df, bowlr.df, "player", suffix = c("wicket", "run"))
batabilities <- inner_join(bat.df, batr.df, "player", suffix = c("wicket", "run"))
```

Composite Measures

2 measures combining the wicket and runs models are presented. One is a simple arithmetic mean of the log-abilities and the other (effAvg) is consistent with the Average as is usually formed in cricket. The wicket model gives $P(wicket)$ against an average (log-ability = 0) bowler/batter and the runs model can be used to calculate $\mathbb{E}(runs)$ against the average bowler/batter. Effective Average is then calculated as $\frac{\mathbb{E}(runs)}{P(wicket)}$.

```
bowlabilities$probwicket <- 1 / (1 + exp(nus[1] - bowlabilities$abilitywicket))
bowlabilities$xruns <- (exp(RunNus["one"] - bowlabilities$abilityrun)
                        + 2 * exp(RunNus["two"] - 2* bowlabilities$abilityrun)
                        + 3 * exp(RunNus["three"] - 3* bowlabilities$abilityrun)
                        + 4 * exp(RunNus["four"] - 4* bowlabilities$abilityrun)
                        + 6 * exp(RunNus["six"] - 6* bowlabilities$abilityrun))/
                        (1 + exp(RunNus["one"] - bowlabilities$abilityrun)
                        + exp(RunNus["two"] - 2* bowlabilities$abilityrun)
                        + exp(RunNus["three"] - 3* bowlabilities$abilityrun)
                        + exp(RunNus["four"] - 4* bowlabilities$abilityrun)
                        + exp(RunNus["six"] - 6* bowlabilities$abilityrun))

bowlabilities$effAvg <- bowlabilities$xruns / bowlabilities$probwicket

bowlabilities <- bowlabilities[order(bowlabilities$effAvg, decreasing = F), ]
rownames(bowlabilities) <- 1:length(bowlabilities$player)

rmarkdown::paged_table(head(bowlabilities, 10))
```

```
batabilities$probwicket <- 1 / (1 + exp(nus[1] + batabilities$abilitywicket))
batabilities$xruns <- (exp(RunNus["one"] + batabilities$abilityrun)
                        + 2 * exp(RunNus["two"] + 2* batabilities$abilityrun)
                        + 3 * exp(RunNus["three"] + 3* batabilities$abilityrun)
                        + 4 * exp(RunNus["four"] + 4* batabilities$abilityrun)
                        + 6 * exp(RunNus["six"] + 6* batabilities$abilityrun))/
                        (1 + exp(RunNus["one"] + batabilities$abilityrun)
                        + exp(RunNus["two"] + 2* batabilities$abilityrun)
                        + exp(RunNus["three"] + 3* batabilities$abilityrun)
                        + exp(RunNus["four"] + 4* batabilities$abilityrun)
                        + exp(RunNus["six"] + 6* batabilities$abilityrun))

batabilities$effAvg <- batabilities$xruns / batabilities$probwicket
batabilities$averageability <- 0.5 * (batabilities$abilitywicket + batabilities$abilityrun)

batabilities <- batabilities[order(batabilities$effAvg, decreasing = T), ]
rownames(batabilities) <- 1:length(batabilities$player)


rmarkdown::paged_table(head(batabilities, 10))
```