Research about adequacy of Elo based models applied to snooker match results. Contains a novel approach (EloBeta) targeted for sport results with variable “best of N” format.

**Prologue**

For many years I’ve been following snooker as a sport. It has it all: hypnotic beauty of smart play, elegance of cue strikes and psychological tension of competition. The one thing I don’t like that much is its *ranking system*.

Generally speaking, **current snooker ranking is based on player accomplishments in tournaments (events)** with different “weights” for different tournaments. Long time ago, it just used World Championships. Then, after more events had emerged, there was a table of points player could earn for winning at certain stage of tournament. Now it has the form of “rolling” sum of prize money player won during (roughly) past two calendar years.

This system has two main advantages: it is **simple** (win more money -> rise in rankings) and **predictabile** (want to get certain ranking -> win certain amount of money, other things being equal). The problem is that **this type of rankings doesn’t account the strength (skill, form) of player’s opponents**. The usual counter-argument for this is that if player reached high stage of tournament then he/she is “strong” at this moment of time (“weak players don’t win tournaments”). Well, it does sound quite convincing. However, in snooker, as in any sport, the role of chance should be taken into account: if player is “weaker” it doesn’t mean that he can’t ever win in a match with “stronger” player. It means that this happens less often then the other way around. Here where [Elo model](https://en.wikipedia.org/wiki/Elo_rating_system) comes into play.

The idea behind Elo model is that each player is associated with numerical rating. The assumption is that a result of a game between two players can be predicted based on difference of their ratings: more value indicates more probability of “stronger” (with higher rating) player to win. **Elo ranking is based on current player “strength”** derived by wins against other players. This eliminates main disadvantage of current official ranking system. It is also capable of updating player rating during tournament to numerically react to player’s strong tournament performance.

Having some practical experience with Elo ratings, I think it can do well in snooker too. However, there is one obstacle: *it is devised for competitions with uniform type of games*. Yes, there are some variations to account for home field advantage in football or first move advantage in chess (both by adding fixed amount of rating points to “less advantageous” player). In snooker, however, matches are played in the “best of \(N\)” format: the first one to win \(n = \frac{N + 1}{2}\) frames wins a match. We will also call this format “\(n\) to win”.

Intuitively, winning a “10 to win” match (final of major tournament) should be harder for “weaker” player then “4 to win” match (first rounds of current Home Nations tournaments). This is taken into account by my proposed **EloBeta model**.

To celebrate actual start of 2018/19 snooker season I decided to write this post in which **I explore adequacy of both Elo and EloBeta models on snooker match results**. Note that the goal is not to build models for forecasting and gambling purposes but for assessing players “strength” and creating “fair” ranking.

The idea of using Elo rating in snooker is not new at all. There are works on this topic, for example:

* Snooker Analyst provides “Elo inspired” (more like [Bradley–Terry model](https://en.wikipedia.org/wiki/Bradley-Terry_model)) rating system based on the idea of updating rating based on difference between “actual frames won” and “expected frames won”. This approach is a little bit questionable. Surely, more frame difference should indicate more difference in strength, however, achieving that is not player’s goal. In snooker aim is “just” to win match, i.e. get certain amount of frame wins before the opponent.
* Possibly, there are other works that I’ve missed. I will highly appreciate any information on this topic.

**Overview**

This post is intended for both R users interested in Elo ratings and snooker analysts and fans. All experiments are written with intention to be reproducible. All code is hidden under spoilers (text appears after clicking on its summary, usually beginning with “Code for …”). It has commentaries and uses tidyverse packages, so it might be an interesting read for R users and programmers.

This post is organized as follows:

* **Models** describes *Elo* and *EloBeta* models with their R implementations.
* **Experiment** describes details and intentions of computations: which *data* and *methodology* were used (and why) and what are the *results*.
* **Exploration of EloBeta ratings** has application results of EloBeta model to actual snooker data. This section is written more for Snooker fans than for R enthusiasts.

We will need the following setup:

Code for setup

# Data wrangling packages

suppressPackageStartupMessages(library(dplyr))

library(tidyr)

library(purrr)

# Visualization package

library(ggplot2)

# Package for ratings

suppressPackageStartupMessages(library(comperank))

theme\_set(theme\_bw())

# Shouldn't be needed. Added just in case.

set.seed(20180703)

**Models**

Both models are based on the following assumptions:

1. There is a fixed set of players which should be ranked from “strongest” (first place) to “weakest” (last place).
2. Ranking is done by associating player \(i\) with numerical rating \(r\_i\): a number indicating the “strength” of player (more value -> “stronger”).
3. The more difference in player ratings before the match the less favorable is “weaker” player to win it.
4. Ratings are updated after every match based on its result and the ratings before it.
5. Winning against “stronger” opponent should lead to bigger increase in rating than winning against “weaker” opponent. The opposite should be true for losing.

**Elo**

Code for Elo model

#' @details This function is vectorized by all its arguments. Also usage of

#' `...` is crucial to allow supplying unrelated arguments in the future.

#'

#' @return A probability of player 1 (rating `rating1`) wins in a match with

#' player 2 (rating `rating2`). Here difference in ratings directly affects

#' the outcome.

elo\_win\_prob <- function(rating1, rating2, ksi = 400, ...) {

norm\_rating\_diff <- (rating2 - rating1) / ksi

1 / (1 + 10^norm\_rating\_diff)

}

#' @return A rating function for Elo model that can be supplied to

#' `comperank::add\_iterative\_ratings()`.

elo\_fun\_gen <- function(K, ksi = 400) {

function(rating1, score1, rating2, score2) {

comperank::elo(rating1, score1, rating2, score2, K = K, ksi = ksi)[1, ]

}

}

Elo model updates ratings by the following steps:

* **Compute probability** (*before* the match) **of certain player to win the match**. Probability of one player (we will call him/her “first”) with “identifier” \(i\) and rating \(r\_i\) winning against the other player (“second”) with “identifier” \(j\) and rating \(r\_j\) is equal to

\[Pr(r\_i , r\_j) = \frac{1}{1 + 10^{(r\_j – r\_i)/400}}\]

This way of computing probability is aligned with third model assumption.

Difference normalization by 400 is a mathematical way to say which difference is considered “big”. This can be replaced by a model parameter \(\xi\), however this only affects the spread of future ratings and is often an overkill. Number 400 is fairly standard in chess world.

In general approach probability is equal to \(L(r\_j – r\_i)\) where \(L(x)\) is some strictly increasing function with values from 0 to 1. We will use logistic curve to compute winning probability.

**Obtain match result** \(S\). In basic model it is 1 if first player wins (second player loses), 0.5 in case of draw and 0 if second player wins.

* **Update ratings**:
  + \(\delta = K \cdot (S – Pr(r\_i , r\_j))\). This is the value by which ratings will change. It introduces the “K factor” (only model parameter). Less \(K\) (ratings being equal) means less change in ratings – model is more conservative, i.e. more wins is needed to “prove” increase in “strength”. On the other hand, more \(K\) means more “trust” to the current results than current ratings. *Choosing “optimal” \(K\) is a way to produce “good” ranking system*.
  + \(r\_i^{(new)} = r\_i + \delta\), \(r\_j^{(new)} = r\_j – \delta\).

**Notes**:

* As one can see from rating update formulas, the sum of ratings for all ranked players doesn’t change over time: rating increase of one rating can be only done by taking this amount from another player.
* Players without any matches played are associated with initial rating 0. The usual value is 1500, however I don’t see any other reason except psychological for this. With previous note, using 0 means that sum of all ratings will always be 0, which is kind of beautiful.
* It is needed some matches to be played in order for rating to represent player’s “strength”. This introduces a problem: newly added players start with rating 0 which is almost surely not the lowest among current players. In other words, newcomers are considered to be “stronger” than some other players. This should be dealt with by external procedures of rating updates when introducing new player: maybe he/she should start with some low rating while compensating overall sum decrease by uniform increasing of other players’ ratings.
* Why this procedure makes sense? In case of equal ratings \(\delta\) always equals \(0.5 \cdot K\). Let’s say, for example, that \(r\_i = 0\) and \(r\_j = 400\). It means that probability of first player winning is \(\frac{1}{1 + 10} \approx 0.0909\), i.e. he/she will win 1 out of 11 matches.
  + In case of win, he/she will be awarded with approximately \(0.909 \cdot K\) increase, which is more than in case of equal ratings.
  + If he/she is defeated, then rating is decreased by approximately \(0.0909 \cdot K\), which is less than in case of equal ratings.

This shows that Elo model is aligned with fifth model assumption: winning against “stronger” opponent leads to bigger increase in rating than winning against “weaker” opponent and vice versa.

Of course, Elo model has its (fairly high-level) [practical issues](https://en.wikipedia.org/wiki/Elo_rating_system#Practical_issues). However, the most important for our research is that “it thinks” that all matches are played in uniform conditions. It means, that match length isn’t taken into account: winning in “4 to win” match is rewarded the same as winning in “10 to win” match. That is where EloBeta comes into play.

**EloBeta**

Code for EloBeta model

#' @details This function is vectorized by all its arguments.

#'

#' @return A probability of player 1 (rating `rating1`) wins in a match with

#' player 2 (rating `rating2`). Match is played until either player wins

#' `frames\_to\_win` frames. Here difference in ratings directly affects

#' the probability of winning one frame.

elobeta\_win\_prob <- function(rating1, rating2, frames\_to\_win, ksi = 400, ...) {

prob\_frame <- elo\_win\_prob(rating1 = rating1, rating2 = rating2, ksi = ksi)

# Probability that first player wins `frames\_to\_win` frames sooner than second

# player based on probability of first player to win one frame `prob\_frame`.

# Frames are treated as independent games.

pbeta(prob\_frame, frames\_to\_win, frames\_to\_win)

}

#' @return Match result in terms of player 1 win: 1 if he/she wins, 0.5 in case

#' of a draw, and 0 if he/she loses.

get\_match\_result <- function(score1, score2) {

# There are no ties in snooker but this handles general case

near\_score <- dplyr::near(score1, score2)

dplyr::if\_else(near\_score, 0.5, as.numeric(score1 > score2))

}

#' @return A rating function for EloBeta model that can be supplied to

#' `comperank::add\_iterative\_ratings()`.

elobeta\_fun\_gen <- function(K, ksi = 400) {

function(rating1, score1, rating2, score2) {

prob\_win <- elobeta\_win\_prob(

rating1 = rating1, rating2 = rating2,

frames\_to\_win = pmax(score1, score2), ksi = ksi

)

match\_result <- get\_match\_result(score1, score2)

delta <- K \* (match\_result - prob\_win)

c(rating1 + delta, rating2 - delta)

}

}

In Elo model difference in ratings directly affects the outcome probability of winning the whole match. The main idea behind EloBeta model is to **make difference in ratings directly affect the outcome of one frame** and **actually compute the probability of a player winning \(n\) frames before his opponent**.

The question is: how to compute this probability? Turns out, this is one of the oldest task in the history of probability theory and has its own name – [Problem of points](https://en.wikipedia.org/wiki/Problem_of_points). Using its notation, the outcome probability equals to:

\[  
P(n, n) = \sum\limits\_{j = n}^{2n-1}{{{2n-1} \choose {j}} p^j (1-p)^{2n-1-j}}  
\]

Here \(P(n, n)\) is a probability of first player to win in “\(n\) to win” match; \(p\) is his/her probability to win one frame (\(1-p\) – opponents probability). With this approach one assumes that **result of frames inside a match is independent one from another**. This is arguable, but necessary assumption (in terms of this model).

Is there a way to compute this faster? It turns out, that the answer is yes. After hours of formula wrangling, practical experiments and internet search I found the following [property](https://dlmf.nist.gov/8.17#E5) of [regularized incomplete beta function](https://en.wikipedia.org/wiki/Beta_function#Incomplete_beta_function) \(I\_x(a, b)\). By substituting \(m = k,~ n = 2k – 1\) into that property and changing \(k\) into \(n\) we obtain that \(P(n, n) = I\_p(n, n)\).

This is also very good news for R users, because \(I\_p(n, n)\) can be computed as simply as pbeta(p, n, n). **Note** that the general case probability of winning \(n\) frames before opponent wins \(m\) can also be computed as \(I\_p(n, m)\) and pbeta(p, n, m) respectively. This introduces the reach field of *updating winning probabilities during the mach*.

So the procedure of updating ratings within EloBeta model is as follows (given ratings \(r\_i, r\_j\), number of frames \(n\) to win in match and match outcome \(S\), as in Elo model):

* **Compute probability of first player to win the frame**: \(p = Pr(r\_i , r\_j) = \frac{1}{1 + 10^{(r\_j – r\_i)/400}}\).
* **Compute probability of this player to win the match**: \(Pr^{Beta}(r\_i, r\_j) = I\_p(n, n)\). For example, if \(p\) equals 0.4 then probability to win “4 to win” match drops to 0.29 and with “18 to win” to 0.11.
* **Update ratings**:
  + \(\delta = K \cdot (S – Pr^{Beta}(r\_i , r\_j))\).
  + \(r\_i^{(new)} = r\_i + \delta\), \(r\_j^{(new)} = r\_j – \delta\).

**Note** that, as difference in ratings affects probability of winning one frame (not the whole match), we should expect lower optimal \(K\): part of \(\delta\)’s value comes from amplifying effect of \(Pr^{Beta}(r\_i, r\_j)\).

The idea of computation match result based on probability of winning one frame is not very novel. I was very glad to notice that results of our computations match. However, I didn’t find any sources of incorporating this into Elo updating procedure. I will highly appreciate any information on this topic.

Simulating backgammon players ELO Rating

Winning prob. = 1 - (1 / (10 ^ ((YOU - HIM) \* SQRT(ML) / 2000) + 1))

(As an aside, I don’t think there is a strong theoretical reason for the SQRT(ML) in that formula. With the complications of the doubling cube I very much doubt that the relationship of the probability winning to match length is as simple as that. But it’s a reasonable empirical approximation; I might blog more about that another time.)

Now, here’s how I made that animation. In the R code below, first, I load up some functionality and the Google font I use for this blog. Then I define a function that estimates the probability of a player winning using the FIBS formula above.

library(dplyr)

library(showtext) *# for fonts*

library(RColorBrewer)

library(directlabels) *# for labels on contour lines*

library(ggplot2)

library(scales)

library(forecast) *# for auto.arima later on*

font.add.google("Poppins", "myfont")

showtext.auto()

*#==================helper functions=====================*

fibs\_p **<-** **function**(a, b, ml){

*# function to determine the expected probability of player a winning a bachgammon match*

*# against player b of length ml, with a and b representing their FIBS Elo ratings*

tmp **<-** 1 **-** (1 **/** (10 **^** ((a **-** b) **\*** **sqrt**(ml) **/** 2000) **+** 1))

**return**(tmp)

}

The strategy to create the animation is simple:

* Create all the combinations of two players’ Elo ratings.
* Loop through 23 possible match lengths, calculating the probabilities of A winning for each combination of Elo ratings at that match length
* Draw a still image heatmap for each of those 23 match lengths using ggplot2
* Compile the images into a single animated Gif using [ImageMagick](http://studio.imagemagick.org/script/index.php).

*#================heat maps showing probability of winning*

*# create a matrix of players A and B's possible Elo ratings*

a **<-** b **<-** seq(from **=** 1000, to **=** 2000, by **=** 5)

mat **<-** expand.grid(a, b) **%>%**

rename(a **=** Var1, b **=** Var2)

*# create a folder to hold the images and navigate to it*

dir.create("tmp0002")

owd **<-** setwd("tmp0002")

*# cycle through a range of possible match lengths, drawing a plot for each*

matchlengths **<-** seq(from **=** 1, to **=** 23, by **=** 1)

res **<-** 150

**for**(i **in** 1**:length**(matchlengths)){

df1 **<-** mat **%>%** mutate(probs **=** fibs\_p(a **=** a, b **=** b, ml **=** matchlengths[i]))

p1 **<-** ggplot(df1, aes(x **=** a, y **=** b, z **=** probs)) **+**

geom\_tile(aes(fill **=** probs)) **+**

theme\_minimal(base\_family **=** "myfont") **+**

scale\_fill\_gradientn(colours **=** brewer.pal(10, "Spectral"), limits **=** **c**(0, 1)) **+**

scale\_colour\_gradientn(colours **=** "black") **+**

stat\_contour(aes(colour **=** ..level..), binwidth **=** .1) **+** *# force contours to be same distance each image*

labs(x **=** "Player A Elo rating", y **=** "Player B Elo rating") **+**

theme(legend.position **=** "none") **+**

coord\_equal() **+**

ggtitle(paste0("Probability of Player A winning a match to ", matchlengths[i]))

png(paste0(letters[i], ".png"), res **\*** 5, res **\*** 5, res **=** res, bg **=** "white")

print(direct.label(p1)) *# direct labels used to label the contour points*

dev.off()

}

*# compile the images into a single animated Gif with ImageMagick. Note that*

*# the {animation} package provides R wrappers to do this but they often get*

*# confused (eg clashes with Windows' convert) and it's easier to do it explicitly*

*# yourself by sending an instruction to the system:*

system('"C:\\Program Files\\ImageMagick-6.9.1-Q16\\convert" -loop 0 -delay 150 \*.png "EloProbs.gif"')

*# move the asset over to where needed for the blog*

file.copy("EloProbs.gif", "../../img/0002-EloProbs.gif", overwrite **=** **TRUE**)

*# clean up*

setwd(owd)

unlink("tmp0002", recursive **=** **TRUE**)

## Actual ratings varying, “true” rating constant

The estimated probability of winning against a certain opponent at a given match length might be of interest to backgammon players (I’m surprised it’s not referred to more often), but its most common use is embedded in the calculation of how much each player’s Elo rating changes after a match is decided. That formula is well explained by FIBS and a little involved so I won’t spell it out here, but you can see it in action in the function I provide later in this post. A key point for our purposes is that as the win/loss of a match is a random event, players’ Elo ratings which are based on those events are random variables. Even if we grant the existence of a “real” value for each player, it is unobservable, and can only be estimated by their actual Elo rating in a tournament or internet forum.

In fact, my original motivation for this blog post was to see how much Elo ratings fluctuate due to randomness of individual games. In a later post I’ll have a more complex and realistic simulation, but the chart below shows what happens to a player’s Elo ratings over time in the following situation:

* Only two players
* They only ever play 5 point matches, and they play 10,000 of them
* Player A’s chance of winning the 5 point match is 0.6, and this does not change over time (ie no player is improving in skill relative to the other)

The results are shown in the chart below, and in this unrealistically simple scenario there’s more variation than I’d realised there would be. Player A’s actual Elo rating fluctuates fairly wildly around the true value (which can be calculated as 1578.75), hitting 1650 several times and dropping nearly all the way to 1500 at one point.

Here’s the code for that simulation, including my R function that provides FIBS-style Elo ratings, adjusted for experience as set out by FIBS.

*#=================determine change of Elo rating ================*

fibs\_scores **<-** **function**(a, b, winner **=** "a", ml **=** 1, axp **=** 500, bxp **=** 500){

*# a is Elo rating of player A before match*

*# b is Elo rating of player B before match*

*# ml is match length*

*# axp is total match lengths (experience) of player a until this match*

*# bxp is total match lengths (experience) of player b until this match*

*# see http://www.fibs.com/ratings.html for formulae*

*# calculate experience-correction multipliers:*

multa **<-** ifelse(axp **<** 400, 5 **-** ((axp **+** ml) **/** 100), 1)

multb **<-** ifelse(bxp **<** 400, 5 **-** ((axp **+** ml) **/** 100), 1)

*# probability of A winning, using fibs\_p function defined earlier:*

winproba **<-** fibs\_p(a **=** a, b **=** b, ml **=** ml)

*# match value (points to be distributed between the two players):*

matchvalue **<-** 4 **\*** **sqrt**(ml)

*# who gets them?:*

**if**(winner **==** "b"){

a **<-** a **-** matchvalue **\*** winproba **\*** multa

b **<-** b **+** matchvalue **\*** winproba **\*** multb

} **else** {

a **<-** a **+** matchvalue **\*** (1 **-** winproba) **\*** multa

b **<-** b **-** matchvalue **\*** (1**-** winproba) **\*** multb

}

**return**(**list**(a **=** a, b **=** b, axp **=** axp **+** ml, bxp **=** bxp **+** ml))

}

*# test against "baptism by fire" example at http://www.fibs.com/ratings.html*

*# should be 1540.95:*

**round**(fibs\_scores(a **=** 1500, b **=** 1925, ml **=** 7, axp **=** 0, bxp **=** 10000, winner **=** "a")**$**a, 2)

*#================simulations of how long it takes for scores to stablise================*

*# two people playing eachother 5 point games with 0.6 chance of winning*

A **<-** B **<-** data\_frame(rating **=** 1500, exp **=** 0)

timeseries **<-** data\_frame(A **=** A**$**rating, B **=** B**$**rating)

set.seed(123) *# for reproducibility*

**for** (i **in** 1**:**10000){

result **<-** fibs\_scores(a **=** A**$**rating, b **=** B**$**rating,

winner **=** ifelse(runif(1) **>** 0.4, "a", "b"),

axp **=** A**$**exp, bxp **=** B**$**exp, ml **=** 5)

timeseries[i, "A"] **<-** A**$**rating **<-** result**$**a

timeseries[i, "B"] **<-** B**$**rating **<-** result**$**b

A**$**exp **<-** result**$**axp

B**$**exp **<-** result**$**bxp

}

timeseries**$**A **<-** ts(timeseries**$**A)

timeseries**$**B **<-** ts(timeseries**$**B)

timeseries**$**time **<-** 1**:**nrow(timeseries)

*# theoretical value:*

tv **<-** (**log**(1**/** 0.4 **-** 1) **/** **log**(10) **\*** 2000 **/** **sqrt**(5) **+** 3000 ) **/** 2

svg("../img/0002-elo-rating.svg", 8, 5)

ggplot(timeseries, aes(x **=** time, y **=** A)) **+**

geom\_line(colour **=** "grey50") **+**

theme\_minimal(base\_family **=** "myfont") **+**

scale\_y\_continuous("Player A's Elo rating") **+**

geom\_hline(yintercept **=** tv, colour **=** "blue") **+**

ggtitle("Elo rating of Player A in two player, constant skill simulation") **+**

scale\_x\_continuous("\nNumber of matches played", label **=** comma) **+**

annotate("text", y **=** tv **-** 10, x **=** **max**(timeseries**$**time) **-** 100,

label **=**"Theoretical\nvalue", colour **=** "blue",

family **=** "myfont", size **=** 3)

dev.off()

In this situation, the resulting player’s Elo rating bouncing around at random is quite well modelled by an autoregressive AR(1) time series model with an intercept at the “true” level of the player’s skill. With an autoregression parameter of around 0.98, the rating at any point in time is obviously very closely related to the previous rating; almost but not quite a random walk. Showing how and why would make this post too long, but it’s a useful factoid to note for future work when we come to more realistic and complex simulations of Elo ratings on FIBS.

**Experiment**

The experiment has several goals. Based on snooker data:

* Derive best value of “K factor” for both models.
* Study stability of models in terms of prediction probability accuracy.
* Study the effect of using “invitational” events on ratings.
* Produce “best” rating history of all professional players from 2017/18 season.

**Data**

Code for data creation

# Function to split cases between "train", "validation", and "test" types

split\_cases <- function(n, props = c(0.5, 0.25, 0.25)) {

breaks <- n \* cumsum(head(props, -1)) / sum(props)

id\_vec <- findInterval(seq\_len(n), breaks, left.open = TRUE) + 1

c("train", "validation", "test")[id\_vec]

}

pro\_players <- snooker\_players %>% filter(status == "pro")

# Matches only between pro players.

pro\_matches\_all <- snooker\_matches %>%

# Using actually happened matches

filter(!walkover1, !walkover2) %>%

# Filter matches only between pro players

semi\_join(y = pro\_players, by = c(player1Id = "id")) %>%

semi\_join(y = pro\_players, by = c(player2Id = "id")) %>%

# Add 'season' column

left\_join(

y = snooker\_events %>% select(id, season), by = c(eventId = "id")

) %>%

# Ensure arranging by end date

arrange(endDate) %>%

# Prepare for widecr format

transmute(

game = seq\_len(n()),

player1 = player1Id, score1, player2 = player2Id, score2,

matchId = id, endDate, eventId, season,

# Compute match type ("train", "validation", or "test") with 50/25/25 split

matchType = split\_cases(n())

) %>%

# Convert to widecr format

as\_widecr()

# Matches only between pro players in not invitational events (which by

# quantity is dominated by Championship League).

pro\_matches\_off <- pro\_matches\_all %>%

anti\_join(

y = snooker\_events %>% filter(type == "Invitational"),

by = c(eventId = "id")

)

# Split confirmation

get\_split <- . %>% count(matchType) %>% mutate(share = n / sum(n))

# This should give 50/25/25 split (train/validation/test).

pro\_matches\_all %>% get\_split()

## # A tibble: 3 x 3

## matchType n share

##

## 1 test 1030 0.250

## 2 train 2059 0.5

## 3 validation 1029 0.250

# This gives different split because invitational events aren't spread evenly

# during season. However, this way matches are split based on the same

# \_\_time\_\_ breaks as in `pro\_matches\_all`. This ensures that matches with same

# type represent identical \_\_time periods\_\_.

pro\_matches\_off %>% get\_split()

## # A tibble: 3 x 3

## matchType n share

##

## 1 test 820 0.225

## 2 train 1810 0.497

## 3 validation 1014 0.278

# Grid for K factor

k\_grid <- 1:100

We will use snooker data from comperank package. The original source is [snooker.org](http://www.snooker.org/) site. Results are taken from the following matches:

* Match is **from 2016/17 or 2017/18 seasons**.
* Match is a **part of “professional” snooker event**. That is:
  + It has “Invitational”, “Qualifying”, or “Ranking” type. We will also differ two sets of matches: “all matches” (from all these events) and “official matches” (not from invitational events). There are two main reasons behind it:
    - In invitational events not all players are given opportunity to change their ratings (which isn’t a clearly bad thing under Elo and EloBeta models).
    - Belief that players “take seriously” only official ranking events. **Note** that most of “Invitational” events are from “Championship League” which, I think, are treated by most players as practice with opportunity to win money, i.e. “not very seriously”. Their presence can affect outcome ratings.

Besides “Championship League”, other invitational events are: “2016 China Championship”, both “Champion of Champions”, both “Masters”, “2017 Hong Kong Masters”, “2017 World games”, “2017 Romanian Masters”.

* + It describes traditional snooker (not 6 Red or Power Snooker) between individual players (not teams).
  + Both genders can take part (not only men or women).
  + Players of all ages can take part (not only seniors or under 21).
  + It is not “Shoot-Out” as those events are treated differently in snooker.org database.
* Match **actually happened**: its result is due to actual play from both players.
* Match **is between two professionals** (pros). List of professionals is taken as for 2017/18 season (131 players). This seems like the most controversial decision, as removing “pro-ama” (“ama” for “amateur”) and “ama-ama” matches leads to “closing eyes” on pros’ losses to amateurs, and thus giving unfair advantage to those pros. I think this choice is necessary to ensure absence of [rating inflation](https://en.wikipedia.org/wiki/Elo_rating_system#Ratings_inflation_and_deflation) which will happen if matches with amateurs are taken into account. Another possibility would be to study pros and amas together, but this seems unreasonable to me within this research. Professional’s loss to amateur is treated as loss of opportunity to increase rating.

The final numbers of used matches are 4118 for “all matches” and 3644 for “official matches” (62.9 and 55.6 matches per player respectively).

**Methodology**

Code for methodology functions

#' @param matches A `longcr` or `widecr` object with column `matchType` (with

#' type of match for the experiment: "train", "validation", or "test").

#' @param test\_type A type of match to be used for computing goodness of fit.

#' For experiment correctness, all matches with this type should happen later

#' than all other ("warm-up") matches. This means having bigger values in

#' `game` column.

#' @param k\_vec Vector of "K factor" values to compute goodness of fit.

#' @param rate\_fun\_gen Function that, given "K factor" value, returns rating

#' function that can be supplied to `comperank::add\_iterative\_ratings()`.

#' @param get\_win\_prob Function to compute rating probability based on

#' ratings of players (`rating1`, `rating2`) and number of frames needed to

#' win in a match (`frames\_to\_win`). \_\_Note\_\_ that it should be vectorized by

#' all its arguments.

#' @param initial\_ratings Initial ratings in format ready for

#' `comperank::add\_iterative\_ratings()`.

#'

#' @details This function computes:

#' - History of iterative ratings after arranging `matches` by `game` column.

#' - For matches with type equals to `test\_type`:

#' - Probability of player 1 winning.

#' - Match result in terms of player 1 win: 1 if he/she wins, 0.5 in case of

#' a draw, and 0 if he/she loses.

#' - Goodness of fit in the form of RMSE: square root of mean square error,

#' where "error" is difference between predicted probability and match result.

#'

#' @return A tibble with columns 'k' for "K factor" and 'goodness' for RMSE

#' goodness of fit.

compute\_goodness <- function(matches, test\_type, k\_vec, rate\_fun\_gen,

get\_win\_prob, initial\_ratings = 0) {

cat("\n")

map\_dfr(k\_vec, function(cur\_k) {

# Track execution

cat(cur\_k, " ")

matches %>%

arrange(game) %>%

add\_iterative\_ratings(

rate\_fun = rate\_fun\_gen(cur\_k), initial\_ratings = initial\_ratings

) %>%

left\_join(y = matches %>% select(game, matchType), by = "game") %>%

filter(matchType %in% test\_type) %>%

mutate(

# Number of frames needed to win in a match

framesToWin = pmax(score1, score2),

# Probability of player 1 winning a match with `frame\_to\_win` frames

# needed to win.

winProb = get\_win\_prob(

rating1 = rating1Before, rating2 = rating2Before,

frames\_to\_win = framesToWin

),

result = get\_match\_result(score1, score2),

squareError = (result - winProb)^2

) %>%

summarise(goodness = sqrt(mean(squareError)))

}) %>%

mutate(k = k\_vec) %>%

select(k, goodness)

}

#' A wrapper for `compute\_goodness()` to be used with design matrix data.

compute\_goodness\_wrap <- function(matches\_name, test\_type, k\_vec,

rate\_fun\_gen\_name, win\_prob\_fun\_name,

initial\_ratings = 0) {

matches\_tbl <- get(matches\_name)

rate\_fun\_gen <- get(rate\_fun\_gen\_name)

get\_win\_prob <- get(win\_prob\_fun\_name)

compute\_goodness(

matches\_tbl, test\_type, k\_vec, rate\_fun\_gen, get\_win\_prob, initial\_ratings

)

}

#' Function to perform experiment.

#'

#' @param test\_type Vector of values for `test\_type` for `compute\_goodness()`.

#' @param rating\_type Names of rating models.

#' @param data\_type Suffixes of data types.

#' @param k\_vec,initial\_ratings Values for `compute\_goodnes()`

#'

#' @details This function generates design matrix and computes multiple values

#' of goodness of fit for different combinations of rating and data types. For

#' this to work, variables with the following combinations of names should be

#' created in the global environment:

#' - "pro\_matches\_" + `` + `` for matches data.

#' - `` + "\_fun\_gen" for rating function generators.

#' - `` + "\_win\_prob" for functions that compute win probability.

#'

#' @return A tibble with columns:

#' - \_\_testType\_\_ : Test type identifier.

#' - \_\_ratingType\_\_ : Rating type identifier.

#' - \_\_dataType\_\_ : Data type identifier.

#' - \_\_k\_\_ : Value of "K factor".

#' - \_\_goodness\_\_ : Value of goodness of fit.

do\_experiment <- function(test\_type = c("validation", "test"),

rating\_type = c("elo", "elobeta"),

data\_type = c("all", "off"),

k\_vec = k\_grid,

initial\_ratings = 0) {

crossing(

testType = test\_type, ratingType = rating\_type, dataType = data\_type

) %>%

mutate(

dataName = paste0("pro\_matches\_", testType, "\_", dataType),

kVec = rep(list(k\_vec), n()),

rateFunGenName = paste0(ratingType, "\_fun\_gen"),

winProbFunName = paste0(ratingType, "\_win\_prob"),

initialRatings = rep(list(initial\_ratings), n()),

experimentData = pmap(

list(dataName, testType, kVec,

rateFunGenName, winProbFunName, initialRatings),

compute\_goodness\_wrap

)

) %>%

unnest(experimentData) %>%

select(testType, ratingType, dataType, k, goodness)

}

To find “optimal” value of \(K\) we will use the even grid \(K = 1, 2, …, 100\). Accounting for greater values seems to be unreasonable which is confirmed by the experiment. The following procedure is used to find it:

* For every \(K\):
  + **Compute history of iterative ratings** of certain model based on certain data set. It means that ratings of players would be known before every match. This is done with add\_iterative\_ratings() function from comperank package. This procedure corresponds to “live ratings” which update after every match.
  + Based on data, starting from a certain (distant from the beginning) moment in time, **compute goodness of model fit**. We will use [RMSE](https://en.wikipedia.org/wiki/Root-mean-square_deviation) between probability of first player to win (computed based on model) and match result. That is \(RMSE = \sqrt{\frac{1}{|T|} \sum\limits\_{t \in T}{(S\_t – P\_t)^2}}\), where \(T\) – indices of used matches, \(|T|\) – number of used matches, \(S\_t\) – result of match for first player, \(P\_t\) – probability of first player to win the match (computed based on model). Not including matches from the beginning of data is needed for ratings to “catch up” to “current strength” from initial ratings.
* The value of \(K\) with **stable minimal RMSE** is said to be optimal. Here by “stable” we mean that small RMSE values is present in some neighborhood of optimal \(K\) (will be controlled not very strictly by looking at graphs). Values of RMSE lower 0.5 (value for “model” with constant 0.5 probability) will be considered a success.

As one of the goals is to study stability of models, data will be split into three subsets: “train”, “validation” and “test”. They are ordered in time, i.e. any “train”/“validation” match has ending time earlier than any “validation”/“test” match. I decided to do actual split in 50/25/25 proportion for “all matches”. Split of “official matches” is done by removing from “all matches” invitational events. It gives split not totally in desired proportion, but rather 49.7/27.8/22.5. However, this approach ensures that matches with same type in both match data represent identical **time periods**.

Experiment will be performed for all combinations of the following variables:

* **Type of model**: Elo and EloBeta.
* **Type of match data**: “All matches” and “official matches”.
* **Experiment type**: “Validation” (“validation” matches are used for computing RMSE after “warming up” on “train” matches) and “Test” (“test” matches are used after “warming up” on both “train” and “validation” ones).

**Results**

Code for doing experiment

pro\_matches\_validation\_all <- pro\_matches\_all %>% filter(matchType != "test")

pro\_matches\_validation\_off <- pro\_matches\_off %>% filter(matchType != "test")

pro\_matches\_test\_all <- pro\_matches\_all

pro\_matches\_test\_off <- pro\_matches\_off

# Takes some time to run

experiment\_tbl <- do\_experiment()

Code for producing results of experiment

cap\_first <- function(x) {

paste0(toupper(substring(x, 1, 1)), substring(x, 2))

}

plot\_data <- experiment\_tbl %>%

unite(group, ratingType, dataType) %>%

mutate(

testType = cap\_first(testType),

groupName = recode(

group, elo\_all = "Elo, all matches", elo\_off = "Elo, official matches",

elobeta\_all = "EloBeta, all matches",

elobeta\_off = "EloBeta, official matches"

),

# Ensure preferred order. This is needed because sorting of strings will

# give "Elo, all matches", "EloBeta, all matches", "EloBeta, official

# matches", and "Elo, official matches" as, apperently, non-letters are

# ignored while sorting.

groupName = factor(groupName, levels = unique(groupName))

)

compute\_optimal\_k <- . %>% group\_by(testType, groupName) %>%

slice(which.min(goodness)) %>%

ungroup()

compute\_k\_labels <- . %>% compute\_optimal\_k() %>%

mutate(label = paste0("K = ", k)) %>%

group\_by(groupName) %>%

# If optimal K within future facet is on the right, it needs a little

# adjustment to the right. If on the left - full and a little adjustment to

# the left.

mutate(hjust = - (k == max(k)) \* 1.1 + 1.05) %>%

ungroup()

plot\_experiment\_results <- function(results\_tbl) {

ggplot(results\_tbl) +

geom\_hline(

yintercept = 0.5, colour = "#AA5555", size = 0.5, linetype = "dotted"

) +

geom\_line(aes(k, goodness, colour = testType)) +

geom\_vline(

data = compute\_optimal\_k,

mapping = aes(xintercept = k, colour = testType),

linetype = "dashed", show.legend = FALSE

) +

geom\_text(

data = compute\_k\_labels,

mapping = aes(k, Inf, label = label, hjust = hjust),

vjust = 1.2

) +

facet\_wrap(~ groupName) +

scale\_colour\_manual(

values = c(Validation = "#377EB8", Test = "#FF7F00"),

guide = guide\_legend(

title = "Experiment", reverse = TRUE,

override.aes = list(size = 4)

)

) +

labs(

x = "K factor", y = "Goodness of fit (RMSE)",

title = "Best goodness of fit of Elo and EloBeta models are almost equal",

subtitle = paste0(

'Using official matches (without invitational events) gives more ',

'stable results.\n',

'All optimal K values from test experiment (with longer "warm up") are',

' lower than from validation experiment.'

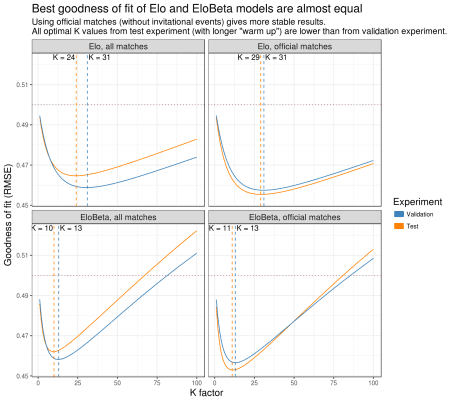
)

) +

theme(title = element\_text(size = 14), strip.text = element\_text(size = 12))

}

plot\_experiment\_results(plot\_data)



From the experiment we can make the following conclusions:

* As it was expected, optimal \(K\) values for EloBeta are lower than for Elo.
* Using official matches (without invitational events) gives more stable results (“Validation” and “Test” results differ less). This shouldn’t be considered as a point that professionals take invitational events not seriously. Probably, this is due to quality of match results from “Championship League”: it has rather unpredictable “3 to win” format and tight schedule.
* Change in RMSE for optimal \(K\) is not substantial. That is, RMSE didn’t change drastically after computing optimal \(K\) in “Validation” and applying it in “Test” experiment. Moreover, on “official matches” it even decreased.
* All optimal K values from test experiment (with longer “warm up”) are lower than from validation experiment. This may be *the result* of longer “warm up” or *just a feature* of particular data.
* Best goodness of Elo and EloBeta fits are almost the same. Both are stable and below 0.5. Data for “official matches” (as they demonstrate stable behavior) is presented below. As results don’t differ that much, we will round optimal \(K\) to a factor of 5: for Elo model it is 30 and for EloBeta – 10.

| **Group** | **Optimal K** | **RMSE** |
| --- | --- | --- |
| Elo, all matches | 24 | 0.465 |
| Elo, official matches | 29 | 0.455 |
| EloBeta, all matches | 10 | 0.462 |
| EloBeta, official matches | 11 | 0.453 |

Based on these results, I am inclined to conclude that **Elo model with \(K = 30\) and EloBeta model with \(K = 10\) can be usefully applied to officially ranked snooker matches**. However, EloBeta model accounts for different “\(n\) to win” matches, so it should be preferred over Elo.

**Exploration of EloBeta ratings**

The following results are computed based on “official matches” with EloBeta model (\(K = 10\)). All possible conclusions shouldn’t be viewed as personal to any player.

**Top 16 by the end of 2017/18**

Code for 2017/18 top 16

# Helper function

gather\_to\_longcr <- function(tbl) {

bind\_rows(

tbl %>% select(-matches("2")) %>% rename\_all(funs(gsub("1", "", .))),

tbl %>% select(-matches("1")) %>% rename\_all(funs(gsub("2", "", .)))

) %>%

arrange(game)

}

# Extract best "K factor" value

best\_k <- experiment\_tbl %>%

filter(testType == "test", ratingType == "elobeta", dataType == "off") %>%

slice(which.min(goodness)) %>%

pull(k)

#!!! Round to "pretty" number as it doesn't affect result that much!!!

best\_k <- round(best\_k / 5) \* 5

# Compute ratings at the end of the data

elobeta\_ratings <- rate\_iterative(

pro\_matches\_test\_off, elobeta\_fun\_gen(best\_k), initial\_ratings = 0

) %>%

rename(ratingEloBeta = rating\_iterative) %>%

arrange(desc(ratingEloBeta)) %>%

left\_join(

y = snooker\_players %>% select(id, playerName = name), by = c(player = "id")

) %>%

mutate(rankEloBeta = order(ratingEloBeta, decreasing = TRUE)) %>%

select(player, playerName, ratingEloBeta, rankEloBeta)

elobeta\_top16 <- elobeta\_ratings %>%

filter(rankEloBeta <= 16) %>%

mutate(

rankChr = formatC(rankEloBeta, width = 2, format = "d", flag = "0"),

ratingEloBeta = round(ratingEloBeta, 1)

)

Top 16 by EloBeta model at the end of 2017/18 season looks like this (official data is also taken from snooker.org site):

| **Player** | **EloBeta rank** | **EloBeta rating** | **Official rank** | **Official rating** | **EloBeta rank increase** |
| --- | --- | --- | --- | --- | --- |
| Ronnie O’Sullivan | **1** | 128.8 | **2** | 905 750 | **1** |
| Mark J Williams | **2** | 123.4 | **3** | 878 750 | **1** |
| John Higgins | **3** | 112.5 | **4** | 751 525 | **1** |
| Mark Selby | **4** | 102.4 | **1** | 1 315 275 | **-3** |
| Judd Trump | **5** | 92.2 | **5** | 660 250 | **0** |
| Barry Hawkins | **6** | 83.1 | **7** | 543 225 | **1** |
| Ding Junhui | **7** | 82.8 | **6** | 590 525 | **-1** |
| Stuart Bingham | **8** | 74.3 | **13** | 324 587 | **5** |
| Ryan Day | **9** | 71.9 | **16** | 303 862 | **7** |
| Neil Robertson | **10** | 70.6 | **10** | 356 125 | **0** |
| Shaun Murphy | **11** | 70.1 | **8** | 453 875 | **-3** |
| Kyren Wilson | **12** | 70.1 | **9** | 416 250 | **-3** |
| Jack Lisowski | **13** | 68.8 | **26** | 180 862 | **13** |
| Stephen Maguire | **14** | 63.7 | **17** | 291 025 | **3** |
| Mark Allen | **15** | 63.7 | **12** | 332 450 | **-3** |
| Yan Bingtao | **16** | 61.6 | **23** | 215 125 | **7** |

Some observations:

* Current official #1 Mark Selby is ranked 3 places lower in EloBeta. This might be a sign that current distribution of prize money doesn’t quite reflect efforts needed to win them (on average).
* Most “underrated” players according to official ranking are Jack Lisowski (astonishing 13 place difference), Ryan Day and Yan Bingtao (both have 7 place difference).
* Stuart Bingham is ranked 5 positions higher by EloBeta probably because he didn’t play for six month due to WPBSA ban. His EloBeta rating didn’t change during this period but in official rating he lost points because of its “rolling” nature. This case demonstrates one important differences between two approaches: official system is good to account for “not playing” and EloBeta is good to account for “playing”.
* Judd Trump and Neil Robertson are ranked the same under both methods.
* With EloBeta model Allister Carter (officially ranked #11), Anthony McGill (#14) and Luca Brecel (#15) are not in top 16. Instead, Jack Lisowski (#26), Yan Bingtao (#23) and Stephen Maguire (#17) are in.

Here is an **example of predictions Based on EloBeta model**. The probability of 16-th player (Yan Bingtao) win one frame in a match against first player (Ronnie O’Sullivan) equals to 0.404. In “4 to win” match it drops to 0.299, in “10 to win” – 0.197 and in World Championship final “18 to win” – 0.125. In my opinion, these numbers might be close to reality.

**Collective evolution of EloBeta ratings**

Code for rating evolution

# Helper data

seasons\_break <- ISOdatetime(2017, 5, 2, 0, 0, 0, tz = "UTC")

# Compute evolution of ratings

elobeta\_history <- pro\_matches\_test\_off %>%

add\_iterative\_ratings(elobeta\_fun\_gen(best\_k), initial\_ratings = 0) %>%

gather\_to\_longcr() %>%

left\_join(y = pro\_matches\_test\_off %>% select(game, endDate), by = "game")

# Generate plot

plot\_all\_elobeta\_history <- function(history\_tbl) {

history\_tbl %>%

mutate(isTop16 = player %in% elobeta\_top16$player) %>%

ggplot(aes(endDate, ratingAfter, group = player)) +

geom\_step(data = . %>% filter(!isTop16), colour = "#C2DF9A") +

geom\_step(data = . %>% filter(isTop16), colour = "#22A01C") +

geom\_hline(yintercept = 0, colour = "#AAAAAA") +

geom\_vline(

xintercept = seasons\_break, linetype = "dotted",

colour = "#E41A1C", size = 1

) +

geom\_text(

x = seasons\_break, y = Inf, label = "End of 2016/17",

colour = "#E41A1C", hjust = 1.05, vjust = 1.2

) +

scale\_x\_datetime(date\_labels = "%Y-%m") +

labs(

x = NULL, y = "EloBeta rating",

title = paste0(

"Most of current top 16 established at the end of 2016/17 season"

),

subtitle = paste0(

"Winning of event is well noticable as rapid increase without ",

"descrease at the end."

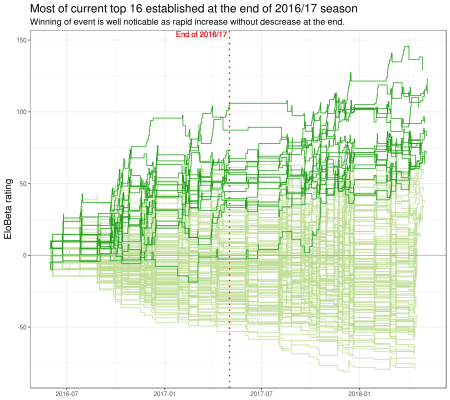
)

) +

theme(title = element\_text(size = 14))

}

plot\_all\_elobeta\_history(elobeta\_history)



**Evolution of EloBeta top 16**

Code for rating evolution of top 16

# Compute plot data

top16\_rating\_evolution <- elobeta\_history %>%

# Using `inner\_join` to leave only players from `elobeta\_top16`

inner\_join(y = elobeta\_top16 %>% select(-ratingEloBeta), by = "player") %>%

# Leave games only from 2017/18 season

semi\_join(

y = pro\_matches\_test\_off %>% filter(season == 2017), by = "game"

) %>%

mutate(playerLabel = paste(rankChr, playerName))

# Generate plot

plot\_top16\_elobeta\_history <- function(elobeta\_history) {

ggplot(elobeta\_history) +

geom\_step(aes(endDate, ratingAfter, group = player), colour = "#22A01C") +

geom\_hline(yintercept = 0, colour = "#AAAAAA") +

geom\_rug(

data = elobeta\_top16,

mapping = aes(y = ratingEloBeta), sides = "r"

) +

facet\_wrap(~ playerLabel, nrow = 4, ncol = 4) +

scale\_x\_datetime(date\_labels = "%Y-%m") +

labs(

x = NULL, y = "EloBeta rating",

title = "Rating evolution for EloBeta top 16 (as of 2017/18 end)",

subtitle = paste0(

"Ronnie O'Sullivan and Mark J Williams did very well in 2017/18 ",

"season.\n",

"As did Jack Lisowski: rise from negative rating to place 13."

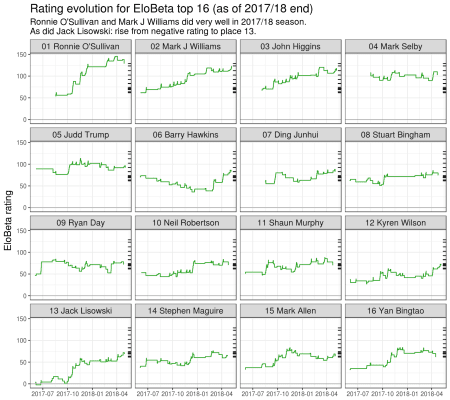
)

) +

theme(title = element\_text(size = 14), strip.text = element\_text(size = 12))

}

plot\_top16\_elobeta\_history(top16\_rating\_evolution)



**Conclusions**

* Solving “Problem of points” in R is as easy as pbeta(p, n, m).
* EloBeta model is a modification of Elo for “best of \(N\)” (or “\(n\) to win”) matches. It can make predictions for different match length.
* Elo model with \(K = 30\) and EloBeta model with \(K = 10\) can be usefully applied to officially ranked snooker matches.
* Snooker related:
  + Most “underrated” players from EloBeta top 16 according to official ranking (as of end of 2017/18 season) are Jack Lisowski, Ryan Day and Yan Bingtao.
  + Season 2017/18 was very productive for Ronnie O’Sullivan, Mark J Williams and Jack Lisowski.

sessionInfo()

sessionInfo()

## R version 3.4.4 (2018-03-15)

## Platform: x86\_64-pc-linux-gnu (64-bit)

## Running under: Ubuntu 16.04.4 LTS

##

## Matrix products: default

## BLAS: /usr/lib/openblas-base/libblas.so.3

## LAPACK: /usr/lib/libopenblasp-r0.2.18.so

##

## locale:

## [1] LC\_CTYPE=ru\_UA.UTF-8 LC\_NUMERIC=C

## [3] LC\_TIME=ru\_UA.UTF-8 LC\_COLLATE=ru\_UA.UTF-8

## [5] LC\_MONETARY=ru\_UA.UTF-8 LC\_MESSAGES=ru\_UA.UTF-8

## [7] LC\_PAPER=ru\_UA.UTF-8 LC\_NAME=C

## [9] LC\_ADDRESS=C LC\_TELEPHONE=C

## [11] LC\_MEASUREMENT=ru\_UA.UTF-8 LC\_IDENTIFICATION=C

##

## attached base packages:

## [1] methods stats graphics grDevices utils datasets base

##

## other attached packages:

## [1] bindrcpp\_0.2.2 comperank\_0.1.0 comperes\_0.2.0 ggplot2\_2.2.1

## [5] purrr\_0.2.5 tidyr\_0.8.1 dplyr\_0.7.6 kableExtra\_0.9.0

##

## loaded via a namespace (and not attached):

## [1] Rcpp\_0.12.17 highr\_0.7 pillar\_1.2.3

## [4] compiler\_3.4.4 plyr\_1.8.4 bindr\_0.1.1

## [7] tools\_3.4.4 digest\_0.6.15 gtable\_0.2.0

## [10] evaluate\_0.10.1 tibble\_1.4.2 viridisLite\_0.3.0

## [13] pkgconfig\_2.0.1 rlang\_0.2.1 cli\_1.0.0

## [16] rstudioapi\_0.7 yaml\_2.1.19 blogdown\_0.6

## [19] xfun\_0.2 httr\_1.3.1 stringr\_1.3.1

## [22] xml2\_1.2.0 knitr\_1.20 hms\_0.4.2

## [25] grid\_3.4.4 rprojroot\_1.3-2 tidyselect\_0.2.4

## [28] glue\_1.2.0 R6\_2.2.2 rmarkdown\_1.10

## [31] bookdown\_0.7 readr\_1.1.1 magrittr\_1.5

## [34] backports\_1.1.2 scales\_0.5.0 htmltools\_0.3.6

## [37] assertthat\_0.2.0 rvest\_0.3.2 colorspace\_1.3-2

## [40] labeling\_0.3 utf8\_1.1.4 stringi\_1.2.3

## [43] lazyeval\_0.2.1 munsell\_0.5.0 crayon\_1.3.4