Package: ovlytics (via r-universe)

August 30, 2024

Title Functions and Algorithms for Volleyball Analytics

Version 0.2.9

Description Analytical functions for volleyball analytics, to be used in conjunction with the datavolley and peranavolley packages.

URL https://ovlytics.openvolley.org, https://github.com/openvolley/ovlytics

BugReports https://github.com/openvolley/ovlytics/issues

Imports assertthat, cowplot, datavolley (>= 1.0.2), dplyr (>= 1.0.0), forcats, ggplot2, grid, gt, htmltools, MASS, methods, ovdata, paletteer, patchwork, purrr, reactable, reactablefmtr, rlang, scales, stringr, tidyr, viridisLite

Suggests covr, shiny, testthat

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.2.1

Roxygen list(markdown = TRUE)

Remotes openvolley/datavolley, openvolley/ovdata

Repository https://openvolley.r-universe.dev

RemoteUrl https://github.com/openvolley/ovlytics

RemoteRef HEAD

RemoteSha 9aab15a671b39955cbda9eac0efc441c1716a593

Contents

`attack_eff` *Various skill performance indicators*

Description

- attack_eff: (number of kills - number of errors and blocked attacks) / (number of attacks)
- serve_eff: (number of aces and positive serves - number of errors and poor serves) / (number of serves)
- reception_eff: (number of perfect and positive passes - number of errors and overpasses) / (number of passes)

Usage

```
attack_eff(evaluation, skill)

serve_eff(evaluation, skill)

reception_eff(evaluation, skill)
```

Arguments

`evaluation`	character: vector of skill evaluations ("Winning attack", "Error", etc)
`skill`	character: (optional) vector of skill values ("Attack", "Block", etc). If provided, it will be used to filter the `evaluation` vector to the elements corresponding to the correct skill. If not provided, all elements of `evaluation` will be used

Value

A numeric scalar

Examples

```
## Not run:
 library(dplyr)
 x <- ovdata_example("mlafin_braslovce_nkbm", as = "parsed")
 plays(x) %>% dplyr::filter(skill == "Attack") %>% group_by(player_name) %>%
   dplyr::summarize(N_attacks = n(), att_eff = attack_eff(evaluation))

## End(Not run)
```

`ovlytics` **ovlytics**

Description

Analytical functions for volleyball analytics, to be used in conjunction with the datavolley and peranavolley packages.

`ov_augment_plays` *Add some extra columns to a plays object*

Description

Add some extra columns to a plays object

Usage

```
ov_augment_plays(
  x,
  to_add = c("receiving_team", "touch_summaries", "setters"),
  rotation = "SHM",
  use_existing = TRUE
)
```

Arguments

`x`	data.frame: the `plays` data.frame as returned by `datavolley::read_dv()` or `peranavolley::pv_read()`
`to_add`	character: columns to add • "receiving_team" adds the columns "receiving_team" (team name) and "receiving_team_id"

- "winners" adds the columns "set_won_by", "set_won_by_id" (the name and ID of the team that won the current set), "match_won_by", "match_won_by_id" (the name and ID of the team that won the current match), "team_won_set" and "team_won_match" (did the team making the action in the current row win the set/match), and "home_sets_won" and "visiting_sets_won" (the number of sets won by the home and visiting teams)
- "touch_summaries" adds a number of columns named "ts_*" that summarize a team touch (e.g. columns "ts_pass_quality", "ts_pass_evaluation_code" give the pass quality and pass evaluation code of the reception or dig associated with a given team touch). "touch_summaries" also adds a column named `freeball_over`, which disambiguates the action of putting a freeball over the net from the action of digging such a ball. Many scouts code both of these as a "Freeball". The `freeball_over` column will be `TRUE` if it was a freeball being put over the net, and `FALSE` otherwise (including freeball digs). Freeballs over and freeball digs will still both have "Freeball" in their `skill` column
- "setters" adds the columns "home_setter_id", "visiting_setter_id" (the player IDs of the home and visiting setter on court), and "setter_id", "setter_position", and "setter_front_back" (the player ID and position of the setter from the team performing the current action)
- "followed" adds the columns "followed_timeout", "followed_technical_timeout", and "followed_sub"
- "player_role" add the column "player_role" which gives the role (outside, middle, opposite, setter) for the active player on each row of `x`. This assumes a standard rotation as specified by `rotation`. Note that `player_role` does NOT include libero, although this can be inferred from the `meta` component of a full datavolley object
- "all" is a shortcut for all of the above

`rotation` string: (only relevant when `to_add` includes "player_role") either "SHM" (assume a setter-hitter-middle rotation order, i.e. outside hitter is in position 2 when the setter is in 1), or "SMH" (setter-middle-hitter)

`use_existing` logical: if `TRUE` and all of the columns associated with a given `to_add` choice are already present in `x`, they won't be re-generated

Value

`x` with the extra columns added

`ov_create_history_table` *Create a prior table from a dvw or a directory of dvw files*

Description

Create a prior table from a dvw or a directory of dvw files

Usage

```
ov_create_history_table(
  dvw,
  play_phase = c("Reception", "Transition"),
  attack_by = "code",
  setter_position_by = "rotation",
  exclude_attacks = c("PR"),
  normalize_parameters = TRUE
)
```

Arguments

dvw	string: path to one or more datavolley files, a list of one or more datavolley objects, or a directory containing datavolley files
play_phase	character: one or both of "Reception", "Transition"
attack_by	string: either "code", "zone", "tempo" or "setter call"
setter_position_by	string: either "rotation", or "front_back"
exclude_attacks	character: vector of attack codes to exclude
normalize_parameters	logical: reduce the prior parameter values

Value

A list, currently with one component named "prior_table"

Examples

```
## use this file to create the priors
hist_dvw <- ovdata_example("190301_kats_beds")
history_table <- ov_create_history_table(dvw = hist_dvw, attack_by = "setter call",
                                    setter_position_by = "front_back")

## use it on another file (here, the same file for demo purposes)
## usually the history would be from a reference set of previous matches

dvw <- ovdata_example("190301_kats_beds")
setter <- ov_simulate_setter_distribution(dvw = dvw, play_phase = "Reception", n_sim = 100,
                                attack_by = "setter call", attack_options = "use_history",
                                    setter_position_by = "front_back",
                                    history_table = history_table, filter_sim = TRUE)

## plot the results
ov_plot_ssd(setter, overlay_set_number = TRUE)
ov_plot_distribution(setter)
```

ov_example_file *Example DataVolley files provided as part of the ovlytics package*

Description

Example DataVolley files provided as part of the ovlytics package

Usage

```
ov_example_file(choice = "190301_kats_beds")
```

Arguments

choice	string: which data file to return? • "190301_kats_beds" - a match between GKS Katowice and MKS Bedzin during the 2018/19 Polish Plus Liga

Value

path to the file

Examples

```
myfile <- ov_example_file()
x <- dv_read(myfile)
summary(x)
```

ov_heatmap_kde *Kernel density estimates for volleyball heatmaps*

Description

Kernel density estimates for volleyball heatmaps

Usage

```
ov_heatmap_kde(
  x,
  y,
  N = NULL,
  resolution = "coordinates",
  bw,
  n,
  court = "full",
  auto_flip = FALSE
)
```

Arguments

x	: either a numeric vector of x-locations, or a three-column data.frame or matrix with columns x, y, and optionally N. If x is a grouped tibble, the kernel density estimates will be calculated separately for group
y	numeric: (unless x is a data.frame or matrix) a numeric vector of y-locations
N	numeric: (unless x is a data.frame or matrix) a numeric vector of counts associated with each location (the corresponding location was observed N times)
resolution	string: the resolution of the locations, either "coordinates" or "subzones"
bw	numeric: a vector of bandwidths to use in the x- and y-directions (see MASS::kde2d()). If not provided, default values will be used based on the location resolution
n	integer: (scalar or a length-2 integer vector) the number of grid points in each direction. If not provided, 60 points in the x-direction and 60 (for half-court) or 120 points in the y-direction will be used
court	string: "full" (generate the kernel density estimate for the full court) or "lower" or "upper" (only the lower or upper half of the court)
auto_flip	logical: if TRUE, and court is either "lower" or "upper", then locations corresponding to the non-selected half of the court will be flipped. This might be appropriate if, for example, the heatmap represents attack end locations that were scouted with coordinates (because these aren't necessarily all aligned to the same end of the court by default)

Value

A data.frame with columns x, y, and density

Examples

```
library(ggplot2)
library(datavolley)

## Example 1 - by coordinates
## generate some fake coordinate data
Na <- 20
set.seed(17)
px <- data.frame(x = c(runif(Na, min = 0.4, max = 1.2), runif(Na, min = 2, max = 3)),
                 y = c(runif(Na, min = 4.5, max = 6.6), runif(Na, min = 4.9, max = 6.6)))

## plot as points
ggplot(px, aes(x, y)) + ggcourt(labels = NULL, court = "upper") +
       geom_point(colour = "dodgerblue")

## or as a heatmap
hx <- ov_heatmap_kde(px, resolution = "coordinates", court = "upper")
ggplot(hx, aes(x, y, fill = density)) +
       scale_fill_distiller(palette = "Purples", direction = 1, labels = NULL,
                            name = "Attack\ndensity") +
       geom_raster() + ggcourt(labels = NULL, court = "upper")
```

```
## Example 2 - by subzones, with data from two attackers
## generate some fake data
Na <- 20
set.seed(17)
px <- data.frame(zone = sample(c(1, 5:9), Na * 2, replace = TRUE),
                 subzone = sample(c("A", "B", "C", "D"), Na * 2, replace = TRUE),
                 attacker = c(rep("Attacker 1", Na), rep("Attacker 2", Na)))

## convert to x, y coordinates
px <- cbind(px, dv_xy(zones = px$zone, end = "upper", subzone = px$subzone))

## plot as tiles
library(dplyr)
ggplot(count(px, attacker, x, y), aes(x, y, fill = n)) + geom_tile() +
       facet_wrap(~attacker) + ggcourt(labels = NULL, court = "upper")

## or as a heatmap, noting that we group the data by attacker first
hx <- ov_heatmap_kde(group_by(px, attacker), resolution = "subzones", court = "upper")
ggplot(hx, aes(x, y, fill = density)) + facet_wrap(~attacker) +
       scale_fill_distiller(palette = "Purples", direction = 1, labels = NULL,
                            name = "Attack\ndensity") +
       geom_raster() + ggcourt(labels = NULL, court = "upper")
```

`ov_infer_player_roles` *Infer the role of each player*

Description

Infer the role of each player

Usage

```
ov_infer_player_roles(
  x,
  target_team,
  method,
  fall_back = TRUE,
  setter_tip_codes = c("PP")
)
```

Arguments

`x`	: a datavolley object (as returned by `datavolley::dv_read()`), a list of datavolley objects, or the `plays` component of a datavolley object
`target_team`	string or function: team to report on. If this is a function, it should return `TRUE` when passed the target team name

method	string: "meta" (rely on player metadata), "SHM" (assume a setter-hitter-middle rotation order), "SMH" (setter-middle-hitter), or "data" (figure out positions from scouting data). Method "meta" is the default if a datavolley object or list of objects is provided
fall_back	logical: if TRUE and method is "meta" and x is a single datavolley object BUT player roles are not provided in the DataVolley file metadata section, fall back to method="data"
setter_tip_codes	character: vector of attack combination codes that correspond to setter tips

Value

A data.frame

Examples

```
x <- ovdata_example("mlafin_braslovce_nkbm", as = "parsed")
## guess roles according to the actions that the players made
rx <- ov_infer_player_roles(x, target_team = "Nova KBM Branik", method = "data")
```

---

ov_plot_distribution *Court plot of a real and simulated setter distribution*

Description

Court plot of a real and simulated setter distribution

Usage

```
ov_plot_distribution(
  ssd,
  label_setters_by = "id",
  font_size = 11,
  title_wrap = NA,
  output = "plot"
)
```

Arguments

ssd	simulated setter distribution output as returned by ov_simulate_setter_distribution()
label_setters_by	string: either "id" or "name"
font_size	numeric: font size
title_wrap	numeric: if non-NA, use strwrap() to break the title into lines of this width
output	string: either "plot" or "list"

Examples

```
dvw <- ovdata_example("190301_kats_beds")
setter <- ov_simulate_setter_distribution(dvw = dvw, play_phase = c("Reception", "Transition"),
                                          n_sim = 100, attack_by = "code")
ov_plot_distribution(setter)
```

`ov_plot_history_table` *Plot the prior table*

Description

Plot the prior table

Usage

```
ov_plot_history_table(history_table, team, setter_id)
```

Arguments

`history_table`	data.frame: the `prior_table` component of the object returned by `ov_create_history_table()`
`team`	string: team name
`setter_id`	string: setter_id

Examples

```
hist_dvw <- ovdata_example("190301_kats_beds")
history_table <- ov_create_history_table(dvw = hist_dvw, attack_by = "tempo",
                                                         setter_position_by = "front_back",
                                                         normalize_parameters = FALSE)
team = unique(history_table$prior_table$team)[1]
setter_id = unique(history_table$prior_table$setter_id)[4]
ov_plot_history_table(history_table, team, setter_id)
```

`ov_plot_sequence_distribution`
Plot a simulated setter distribution sequence

Description

Plot a simulated setter distribution sequence

Usage

```
ov_plot_sequence_distribution(
  ssd,
  label_setters_by = "id",
  font_size = 11,
  title_wrap = NA,
  split_set = FALSE,
  output = "plot"
)
```

Arguments

ssd	simulated setter distribution output as returned by ov_simulate_setter_distribution()
label_setters_by	string: either "id" or "name"
font_size	numeric: font size
title_wrap	numeric: if non-NA, use strwrap() to break the title into lines of this width
split_set	boolean: if TRUE, separate the distribution sequence by set
output	string: either "plot" or "list"

Examples

```
dvw <- ovdata_example("190301_kats_beds")
ssd <- ov_simulate_setter_distribution(dvw = dvw, play_phase = c("Reception"),
                                       n_sim = 100, attack_by = "zone",
                                       setter_position_by = "front_back")
ov_plot_sequence_distribution(ssd)
```

---

ov_plot_ssd *Plot a simulated setter distribution*

Description

Plot a simulated setter distribution

Usage

```
ov_plot_ssd(
  ssd,
  overlay_set_number = FALSE,
  label_setters_by = "name",
  font_size = 11
)
```

Arguments

ssd	simulated setter distribution output as returned by ov_simulate_setter_distribution()
overlay_set_number	boolean: if TRUE, overlay set number and score in the plot
label_setters_by	string: either "id" or "name"
font_size	numeric: font size

Examples

```
dvw <- ovdata_example("190301_kats_beds")
setter <- ov_simulate_setter_distribution(dvw = dvw,
                                          n_sim = 150, attack_by = "zone")
ov_plot_ssd(setter, overlay_set_number = TRUE)
```

---

ov_print_history_table

Print the prior table

Description

Print the prior table

Usage

```
ov_print_history_table(history_table, team, setter_id)
```

Arguments

history_table	data.frame: the prior_table component of the object returned by ov_create_history_table()
team	string: team name
setter_id	string: setter_id

Examples

```
hist_dvw <- ovdata_example("190301_kats_beds")
history_table <- ov_create_history_table(dvw = hist_dvw, attack_by = "zone")
team = history_table$prior_table$team[1]
setter_id = history_table$prior_table$setter_id[1]
ov_print_history_table(history_table, team, setter_id)
```

ov_print_rate_table *Print the rate table*

Description

Print the rate table

Usage

```
ov_print_rate_table(ssd, team, setter_id)
```

Arguments

ssd	simulated setter distribution output as returned by ov_simulate_setter_distribution()
team	string: team name
setter_id	string: setter_id

Examples

```
dvw <- ovdata_example("190301_kats_beds")
system.time({
  ssd <- ov_simulate_setter_distribution(dvw = dvw, play_phase = "Reception",
                                         n_sim = 500, setter_position_by = "front_back")
  team <- ssd$raw_data$meta$teams$team[1]
 setter_id <- ssd$raw_data$meta$players_h$player_id[which(ssd$raw_data$meta$players_h$role == "setter")][1]
  ov_print_rate_table(ssd, team, setter_id)
})
```

ov_season_table *Create a summary table of a team's matches in a season*

Description

Create a summary table of a team's matches in a season

Usage

```
ov_season_table(xl, target_team, target_team_id, show_by = "match date")
```

Arguments

xl	list: list of datavolley objects (each as returned by datavolley::dv_read()
target_team	string: the name of the target team. Only one of target_team or target_team_id is required
target_team_id	string: the team ID of the target team. Ignored if target_team has been provided
show_by	string: either "match date" (show each match according to its date) or "filename" (show each match according to its filename. This might be useful if the match dates are being parsed incorrectly by datavolley::dv_read())

Value

A tibble with columns "Opponent", "Date" (or "File"), "Result", "Set scores", and one column for sets 1 to 5

Examples

```
## trivial example of a single-match "season"
library(datavolley)
x <- dv_read(dv_example_file())
ov_season_table(list(x), target_team = home_team(x))
```

---

ov_setter_repetition *Tabulate setter repeat patterns*

Description

Note: analysis is done on the basis of attack actions, and simply assumes that the setter on court made the set.

Usage

```
ov_setter_repetition(
  x,
  setter_id,
  setter_name,
  exclude_attacks = c("PP", "PR", "P2"),
  exclude_negative_reception = TRUE,
  exclude_highballs = FALSE
)
```

Arguments

x	data.frame: the plays data.frame as returned by datavolley::read_dv() or peranavolley::pv_read()
setter_id	string: (optional) the player ID of the setter to analyze (or provide setter_name). If neither setter_id nor setter_name are provided, all setters will be analyzed separately, and collated results returned
setter_name	string: (optional) the name of the setter to analyze (ignored if setter_id is provided). If neither setter_id nor setter_name are provided, all setters will be analyzed separately, and collated results returned
exclude_attacks	character: vector of attack codes to exclude
exclude_negative_reception	logical: if TRUE, exclude attacks following poor reception (likely to be out-of-system and therefore might not represent attacks on which the setter had genuine options)
exclude_highballs	logical: if TRUE, exclude highball attacks (likely to be out-of-system and therefore might not represent attacks on which the setter had genuine options)

Value

A data.frame with columns "team", "setter_name", "setter_id", "player_name", "player_id", "category", "opportunities", "repeats", "repeat%"

Examples

```
x <- plays(ovdata_example("190301_kats_beds", as = "parsed"))
set_reps <- ov_setter_repetition(x, setter_name = "Lukas Tichacek")

library(ggplot2)
ggplot(set_reps, aes(x = player_name, y = `repeat%`)) + geom_col() +
    geom_text(aes(x = player_name, label = paste0("N=", opportunities)),
              angle = 90, y = 100, hjust = 1, inherit.aes = FALSE) +
    facet_wrap(~category) +
    theme_bw() +
    theme(axis.text.x = element_text(angle = 60, vjust = 1, hjust = 1)) +
    labs(x = NULL, y = "Repeat percentage")
```

---

ov_simulate_multiple_setter_distribution

Simulate a Bayesian Bandit choice for a given set of probabilities and a number of points for multiple games

---

Description

Simulate a Bayesian Bandit choice for a given set of probabilities and a number of points for multiple games

Usage

```
ov_simulate_multiple_setter_distribution(
  list_dv,
  play_phase = c("Reception", "Transition"),
  n_sim = 500,
  priors = list(name = "beta", par1 = 1, par2 = 1),
  epsilon = 1,
  filter_sim = FALSE,
  attack_options = "use_data",
  setter_position_by = "rotation",
  history_table = NULL,
  attack_by = "code",
  exclude_attacks = c("PR"),
  rotation = "SHM",
  shiny_progress = NULL
)
```

Arguments

list_dv	list: list of datavolley object as returned by datavolley::dv_read()
play_phase	character: one or both of "Reception", "Transition"
n_sim	integer: number of simulations
priors	numeric: prior distribution of the kill rate for the different attacking options
epsilon	numeric: reward size
filter_sim	logical:
attack_options	string: either "use_data" or "use_history"
setter_position_by	string: either "rotation" or "front_back"
history_table	list: (only if attack_options is "use_history") the object returned by ov_create_history_table()
attack_by	string: either "code", "zone", "tempo", "setter call", "attacker_name", "player_role"
exclude_attacks	character: vector of attack codes to exclude
rotation	string: (only relevant when attack_by is "player_role") either "SHM" (assume a setter-hitter-middle rotation order), or "SMH" (setter-middle-hitter)
shiny_progress	numeric: an optional two-element vector. If not NULL or NA, shiny::setProgress() calls will be made during simulation with values in this range

See Also

ov_simulate_setter_distribution()

Examples

```
list_dv <- list(dv_read(ovdata_example("190301_kats_beds")))
system.time({
```

```
  mssd <- ov_simulate_multiple_setter_distribution(list_dv = list_dv, play_phase = "Reception",
                n_sim = 100, setter_position_by = "front_back")
})
```

`ov_simulate_setter_distribution` *Simulate a Bayesian Bandit choice for a given set of probabilities and a number of points*

Description

Simulate a Bayesian Bandit choice for a given set of probabilities and a number of points

Usage

```
ov_simulate_setter_distribution(
  dvw,
  play_phase = c("Reception", "Transition"),
  n_sim = 500,
  priors = list(name = "beta", par1 = 1, par2 = 1),
  epsilon = 1,
  filter_sim = FALSE,
  attack_options = "use_data",
  setter_position_by = "rotation",
  history_table = NULL,
  attack_by = "code",
  exclude_attacks = c("PR"),
  rotation = "SHM",
  shiny_progress = NULL
)
```

Arguments

`dvw`	string or datavolley: a datavolley object as returned by `datavolley::dv_read()` or a path to datavolley file
`play_phase`	character: one or both of "Reception", "Transition"
`n_sim`	integer: number of simulations
`priors`	numeric: prior distribution of the kill rate for the different attacking options
`epsilon`	numeric: reward size
`filter_sim`	logical:
`attack_options`	string: either "use_data" or "use_history"
`setter_position_by`	string: either "rotation" or "front_back"
`history_table`	list: (only if `attack_options` is "use_history") the object returned by `ov_create_history_table()`
`attack_by`	string: either "code", "zone", "tempo", "setter call", "attacker_name", "player_role"

exclude_attacks
: character: vector of attack codes to exclude

rotation
: string: (only relevant when attack_by is "player_role") either "SHM" (assume a setter-hitter-middle rotation order), or "SMH" (setter-middle-hitter)

shiny_progress
: numeric: an optional two-element vector. If not NULL or NA, shiny::setProgress() calls will be made during simulation with values in this range

See Also

ov_create_history_table()

Examples

```
dvw <- ovdata_example("190301_kats_beds")
system.time({
  ssd <- ov_simulate_setter_distribution(dvw = dvw, play_phase = "Reception",
                                         n_sim = 100, attack_by = "setter call",
                                         setter_position_by = "front_back")
})
```

---

ov_sort_attack_codes *Sort DataVolley attack codes*

Description

Sort DataVolley attack codes

Usage

```
ov_sort_attack_codes(ac, by = "XV", na.last = NA)
```

Arguments

ac
: character: character vector of attack codes to sort

by
: string: method to use, currently only "XV" (any other value will default back to using sort without modification). "XV" will place X and V codes first (in numerical order, with each X preceding its matching V) then everything else in alphabetical order after that

na.last
: logical: passed to sort

Value

Sorted character vector

Examples

```
ov_sort_attack_codes(c("V5", "V1", "X6", "CF", "X5"))

## Not run:
  ## sorting might be useful for controlling the plot order when facetting
  ##  a `ggplot` by attack code
  mydata$attack_code <- factor(mydata$attack_code,
            levels = ov_sort_attack_codes(unique(na.omit(mydata$attack_code))))
  ggplot(mydata, ...) + facet_wrap(~attack_code)

## End(Not run)
```

`ov_table_mssd` *Table of a simulated multi-game setter distribution sequence*

Description

Table of a simulated multi-game setter distribution sequence

Usage

```
ov_table_mssd(mssd, label_setters_by = "name", team = NULL, nrows = 50)
```

Arguments

`mssd`	simulated multi-game setter distribution output as returned by ov_simulate_multiple_setter_distrib
`label_setters_by`	string: either "id" or "name"
`team`	NULL or string: if non-NULL, show sequence just for this team name
`nrows`	integer: number of rows per page in the table

Examples

```
## Not run:
 list_dv <- list(dv_read(ovdata_example("190301_kats_beds"))) # would normally be multiple games
 mssd <- ov_simulate_multiple_setter_distribution(list_dv = list_dv,
             play_phase = c("Reception", "Transition"), attack_by = "player_role",
             n_sim = 100, setter_position_by = "front_back")

 res <- ov_table_mssd(mssd, team = "GKS Katowice")

## End(Not run)
```

Index