Package: TrueSkillThroughTime (via r-universe)

September 9, 2024

Type Package

Title Skill Estimation Based on a Single Bayesian Network

Version 0.1.1

Date 2023-04-26

Maintainer Gustavo Landfried <gustavolandfried@gmail.com>

Description Most estimators implemented by the video game industry cannot obtain reliable initial estimates nor guarantee comparability between distant estimates. TrueSkill Through Time solves all these problems by modeling the entire history of activities using a single Bayesian network allowing the information to propagate correctly throughout the system. This algorithm requires only a few iterations to converge, allowing millions of observations to be analyzed using any low-end computer. The core ideas implemented in this project were developed by Dangauthier P, Herbrich R, Minka T, Graepel T (2007). ``Trueskill through time: Revisiting the history of chess.'' <https://dl.acm.org/doi/10.5555/2981562.2981605>.

License GPL (>= 3)

Imports hash, methods, stats

Encoding UTF-8

URL https://github.com/glandfried/TrueSkillThroughTime.R

BugReports https://github.com/glandfried/TrueSkillThroughTime.R/issues

RoxygenNote 7.2.1

LazyData false

Depends R (>= 2.10)

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

RemoteUrl https://github.com/glandfried/trueskillthroughtime.r

RemoteRef HEAD

RemoteSha 07565e93b7f0943a4efe0a8d2fbcb8ef0aba0e26

Contents

`Game` *Game*

Description

Game class

Usage

```
Game(teams, result = vector(), p_draw = P_DRAW)

posteriors(g)

## S4 method for signature 'Game'
posteriors(g)
```

Arguments

`teams`	A list of `Player` objects. Each position represents a team, so it must contain a vector of `Player` objects.
`result`	A vector of numbers, with the score obtained by each team, or an empty vector. The default value is an empty vector. In this case, the outcome is defined by the order in which the `teams` list was initialized: the teams appearing firstly in the list defeat those appearing later (no ties). If the list is not empty, it must have the same length as the `teams` list. In this last case, the team with the highest score is the winner, and the teams with the same score are tied.
`p_draw`	A number, the probability of a draw. The default value is `P_DRAW = 0`. A rule of thumb states that the probability of a draw must be initialized with the observed frequency of draws. If in doubt, it is a candidate parameter to be optimized or integrated by the sum rule. It is used to compute the prior probability of the observed result, so its value may affect an eventual model selection task.
`g`	A game object

Value

Game object

Examples

```
a1 = Player(Gaussian(mu=0, sigma=6), beta=1, gamma=0.03)
a2 = Player(); a3 = Player(); a4 = Player()
team_a = c(a1, a2)
team_b = c(a3, a4)
teams = list(team_a, team_b)

g = Game(teams)
post = posteriors(g)
lhs = g@likelihoods
post[[1]][[1]] == lhs[[1]][[1]]*a1@prior
ev = g@evidence
ev == 0.5

ta = c(a1)
tb = c(a2, a3)
tc = c(a4)
teams_3 = list(ta, tb, tc)
result = c(1, 0, 0)
g3 = Game(teams_3, result, p_draw=0.25)
```

Gaussian *Gaussian*

Description

Gaussian class

Usage

```
Gaussian(mu = 0, sigma = 1)

Pi(N)

## S4 method for signature 'Gaussian'
Pi(N)

Tau(N)

## S4 method for signature 'Gaussian'
Tau(N)

forget(N, gamma, t)

## S4 method for signature 'Gaussian,numeric,numeric'
forget(N, gamma, t)
```

```
isapprox(N, M, tol = 1e-04)

## S4 method for signature 'Gaussian,Gaussian,numeric'
isapprox(N, M, tol = 1e-04)

## S4 method for signature 'Gaussian,Gaussian'
e1 + e2

## S4 method for signature 'Gaussian,Gaussian'
e1 - e2

## S4 method for signature 'Gaussian,Gaussian'
e1 * e2

## S4 method for signature 'Gaussian,Gaussian'
e1 / e2

## S4 method for signature 'Gaussian,Gaussian'
e1 == e2

## S4 method for signature 'Player'
performance(a)
```

Arguments

mu	A number, the mean of the Gaussian distribution.
sigma	A number, the standar deviation of the Gaussian distribution.
N	A Gaussian object
gamma	The dynamic factor, the dynamic uncertainty
t	The elapsed time
M	A Gaussian object
tol	The tolerance threshold for comparitions
e1	A Gaussian object
e2	A Gaussian object
a	A Gaussian object

Value

Gaussian object

Examples

```
N01 = Gaussian(0,1); N12 = Gaussian(mu = 1, sigma = 2)
N06 = Gaussian(); Ninf = Gaussian(0,Inf)
N01 * Ninf == N01
N01 * N12
N01 / N12
```

```
N01 + N12
N01 - N12
Pi(N12) == 1/(N12@sigma^2)
Tau(N12) == N12@mu/(N12@sigma^2)
Nnew = forget(N = N01, gamma = 0.01, t = 100)
isapprox(Nnew, Gaussian(N01@mu,sqrt(N01@sigma^2+100*(0.01^2))), tol=1e-6)
```

`History-class` *History*

Description

History class

Arguments

`composition`	A list of list of player's names (id). Each position of the list is a list that represents the teams of a game, so the latter must contain vectors of names representing the composition of each team in that game.
`results`	A list of numeric vectors, representing the outcome of each game. It must have the same length as the `composition` list or be an empty list. The default value is an empty list. When the list is empty, the outcomes of the games are inferred by the order of the teams in the `composition` list: the teams appearing firstly in the list defeat those appearing later (no ties). When the list is not empty, each vector of the list must represents the score of each team in the game. The team with the highest score is the winner, and the teams with the same score are tied.
`times`	A numeric vector, the timestamp of each game. It must have the same length as the `composition` list or be an empty list. The default value is an empty list. When the list is empty, all players' games are separated by a single timestamp, so a single dynamic uncertainty `gamma` will be added between games. When the list is not empty, the amount of dynamic uncertainty will depend on the difference (measured in timestamps) that each player has between games. In addition, the order of the timestamps determines the reading order of the `composition` and `results` lists. If a player appears more than once in the same timestamp, no dynamic uncertainty will be added between those games.
`priors`	A hash object, a dictionary of `Player` objects indexed by the players' name (id). Used to create players with special values. The default value is an empty hash. In this case, one `Player` object for each unique name in the `composition` list is automatically initialized using the values of the parameters `mu`, `sigma`, `beta`, and `gamma`. The names that appear in the hash are the only ones that will be initialized with special values.
`mu`	A number, the prior mean. The deafult value is: `MU = 0`.
`sigma`	A number, the prior standar deviation. The deafult value is: `SIGMA = 6`.

`beta`	A number, the standard deviation of the performance. The default value is: `BETA = 1`. The parameter `beta` acts as the scale of the estimates. A real difference of one `beta` between two skills is equivalent to 76% probability of winning.
`gamma`	A number, the amount of uncertainty (standar deviation) added to the estimates between events. The default value is: `GAMMA = 0.03`.
`p_draw`	A number, the probability of a draw. The default value is `P_DRAW = 0`. A rule of thumb states that the probability of a draw must be initialized with the observed frequency of draws. If in doubt, it is a candidate parameter to be optimized or integrated by the sum rule. It is used to compute the prior probability of the observed result, so its value may affect an eventual model selection task.
`epsilon`	A number, the convergence threshold. Used to stop the convergence procedure. The default value is `EPSILON = 1e-6`.
`iterations`	A number, the maximum number of iterations for convergence. Used to stop the convergence procedure. The default value is `ITERATIONS = 30`.

Value

History object

Fields

`size` A number, the amount of games.

`batches` A vector of `Batch` objects. Where the games that occur at the same timestamp live.

`agents` A hash, a dictionary indexed by the players' name (id).

`time` A boolean, indicating whether the history was initialized with timestamps or not.

`mu` A number, the default prior mean in this particular `History` object

`sigma` A number, the default prior standard deviation in this particular `History` object

`beta` A number, the default standar deviation of the performance in this particular `History` object

`gamma` A number, the default dynamic uncertainty in this particular `History` object

`p_draw` A number, the probability of a draw in this particular `History` object

`h_epsilon` A number, the convergence threshold in this particular `History` object

`h_iterations` A number, the maximum number of iterations for convergence in this particular `History` object

Methods

`convergence(epsilon = NA, iterations = NA, verbose = TRUE)`

`initialize( composition, results = list(), times = c(), priors = hash(), mu = MU, sigma = SIGMA, beta = BETA,`

`learning_curves()`

`log_evidence()`

Examples

```
c1 = list(c("a"),c("b"))
c2 = list(c("b"),c("c"))
c3 = list(c("c"),c("a"))
composition = list(c1,c2,c3)
h = History(composition, gamma=0.0)

trueskill_learning_curves = h$learning_curves()
ts_a = trueskill_learning_curves[["a"]]
ts_a[[1]]$N; ts_a[[2]]$N
ts_a[[1]]$t; ts_a[[2]]$t
h$convergence()
trueskillThrougTime_learning_curves = h$learning_curves()
ttt_a = trueskillThrougTime_learning_curves[["a"]]
ttt_a[[1]]$N; ttt_a[[2]]$N
ttt_a[[1]]$t; ttt_a[[2]]$t

## Not run:
# Synthetic example
library(hash)
N = 100
skill <- function(experience, middle, maximum, slope){
return(maximum/(1+exp(slope*(-experience+middle)))) }
target = skill(seq(N), N/2, 2, 0.075)
opponents = rnorm(N,target,0.5)
composition = list(); results = list(); times = c(); priors = hash()
for(i in seq(N)){composition[[i]] = list(c("a"), c(toString(i)))}
for(i in
seq(N)){results[[i]]=if(rnorm(1,target[i])>rnorm(1,opponents[i])){c(1,0)}else{c(0,1)}}
for(i in seq(N)){times = c(times,i)}
for(i in seq(N)){priors[[toString(i)]] = Player(Gaussian(opponents[i],0.2))}
h = History(composition, results, times, priors, gamma=0.1)
h$convergence(); lc_a = h$learning_curves()$a; mu = c()
for(tp in lc_a){mu = c(mu,tp[[2]]@mu)}
plot(target)
lines(mu)

# Plotting learning curves

# First solve your own example. Here is a dummy one.
agents <- c("a", "b", "c", "d", "e")
composition <- list()
for (i in 1:500) {
 who = sample(agents, 2)
 composition[[i]] <- list(list(who[1]), list(who[2]))
}
h <- History(composition = composition, gamma = 0.03, sigma = 1.0)
h$convergence(iterations=6)

# Then plot some learning curves
lc <- h$learning_curves()
colors <- c(rgb(0.2,0.2,0.8), rgb(0.2,0.8,0.2), rgb(0.8,0.2,0.2))
```

```
colors_alpha <- c(rgb(0.2,0.2,0.8,0.2), rgb(0.2,0.8,0.2,0.2), rgb(0.8,0.2,0.2,0.2))
plot(0,0, xlim = c(0, 500), ylim = c(-1, 1), xlab = "t", ylab = "skill", type = "n")
for (i in 1:3) {
  agent <- agents[i]
  t <- c(); mu <- c(); sigma <- c()
  for(x in lc[[agent]]){
    t <- c(t, x$t )
    mu <- c(mu, x$N@mu)
    sigma <- c(sigma, x$N@sigma)
  }
  lines(t, mu, col = colors[i], lwd = 2, type = "l")
 polygon(c(t, rev(t)), c(mu + sigma, rev(mu - sigma)), col = colors_alpha[i], border = NA)
}
legend("topright", legend = agents[1:3], col = colors, lwd = 2)


## End(Not run)
```

lc_print *Print list of Gaussian using the python and julia syntax*

Description

Print list of Gaussian using the python and julia syntax

Usage

```
lc_print(lc.a)
```

Arguments

lc.a	List of Gaussians

Value

No return value, print lists of Gaussian using the python and julia syntax

Player *Player*

Description

Player class

Usage

```
Player(prior = Nms, beta = BETA, gamma = GAMMA)

performance(a)
```

Arguments

`prior`	A Gaussian object, the prior belief distribution of the skills. The default value is: `Nms = Gaussian(mu = 0, sigma = 6)`.
`beta`	A number, the standard deviation of the performance. The default value is: `BETA = 1`. The parameter `beta` acts as the scale of the estimates. A real difference of one `beta` between two skills is equivalent to 76% probability of winning.
`gamma`	A number, the amount of uncertainty (standar deviation) added to the estimates at each time step. The default value is: `GAMMA = 0.03`.
`a`	A Player object

Value

Player object

Examples

```
a1 = Player(prior = Gaussian(0,6), beta = 1, gamma = 0.03);
a2 = Player()
a1@gamma == a2@gamma
N = performance(a1)
N@mu == a1@prior@mu
N@sigma == sqrt(a1@prior@sigma^2 + a1@beta^2)
```

Index