Package: elo (via r-universe)

September 10, 2024

Version 3.0.2.9000

Title Ranking Teams by Elo Rating and Comparable Methods

Date 2023-10-04

Description A flexible framework for calculating Elo ratings and resulting rankings of any two-team-per-matchup system (chess, sports leagues, 'Go', etc.). This implementation is capable of evaluating a variety of matchups, Elo rating updates, and win probabilities, all based on the basic Elo rating system. It also includes methods to benchmark performance, including logistic regression and Markov chain models.

Depends R (>= 3.6.0), stats

Imports Rcpp, pROC

Suggests knitr, testthat, rmarkdown

VignetteBuilder knitr

License GPL (>= 2)

URL https://github.com/eheinzen/elo, https://cran.r-project.org/package=elo, https://eheinzen.github.io/elo/

BugReports https://github.com/eheinzen/elo/issues

RoxygenNote 7.2.3

LazyData true

LinkingTo Rcpp

Encoding UTF-8

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

RemoteUrl https://github.com/eheinzen/elo

RemoteRef HEAD

RemoteSha 1ce2bba9fa2e91b60d5e947fe4c491f229eba9ba

Contents

`auc.elo` *Calculate AUC on an* `elo.run` *object*

Description

Calculate AUC on an `elo.run` object

Usage

```
## S3 method for class 'elo.run'
auc(object, ..., subset = TRUE)

## S3 method for class 'elo.glm'
auc(object, ..., subset = TRUE)

## S3 method for class 'elo.running'
auc(object, running = TRUE, discard.skipped = FALSE, ..., subset = TRUE)

## S3 method for class 'elo.markovchain'
```

```
auc(object, ..., subset = TRUE)

## S3 method for class 'elo.winpct'
auc(object, ..., subset = TRUE)

## S3 method for class 'elo.colley'
auc(object, ..., subset = TRUE)
```

Arguments

object	An object of class elo.run.
...	Other arguments (not used at this time).
subset	(optional) A vector of indices on which to calculate
running	logical, denoting whether to use the running predicted values.
discard.skipped	Logical, denoting whether to ignore the skipped observations in the calculation

Value

The AUC of the predicted Elo probabilities and the actual win results.

References

Adapted from code here: https://stat.ethz.ch/pipermail/r-help/2005-September/079872.html

See Also

pROC::auc, elo.run.

---

elo *The Elo Package*

Description

An implementation of Elo ratings for general use in 'R'.

Functions

Listed below are the most useful functions available in elo:

elo.prob: Calculate the probability that team A beats team B.

elo.update: Calculate the update value for a given Elo matchup.

elo.calc: Calculate post-update Elo values.

elo.run: Calculate Elos for a series of matches.

score: Create a 1/0/0.5 win "indicator" based on two teams' scores.

Data

tournament: Mock data for examples.

References

Elo, A. E. 1978. The Rating of Chess Players, Past and Present. New York: Arco.

Examples

```
library(elo)
```

`elo.calc` *Post-update Elo values*

Description

Calculate post-update Elo values. This is vectorized.

Usage

```
elo.calc(wins.A, ...)

## Default S3 method:
elo.calc(wins.A, elo.A, elo.B, k, ..., adjust.A = 0, adjust.B = 0)

## S3 method for class 'formula'
elo.calc(formula, data, na.action, subset, k = NULL, ...)
```

Arguments

`wins.A`	Numeric vector of wins by team A.
`...`	Other arguments (not in use at this time).
`elo.A`, `elo.B`	Numeric vectors of elo scores.
`k`	A constant k-value (or a vector, where appropriate).
`adjust.A`, `adjust.B`	Numeric vectors to adjust `elo.A` and `elo.B` by.
`formula`	A formula. See the help page for formulas for details.
`data`	A `data.frame` in which to look for objects in `formula`.
`na.action`	A function which indicates what should happen when the data contain NAs.
`subset`	An optional vector specifying a subset of observations.

Value

A data.frame with two columns, giving the new Elo values after each update.

See Also

elo.prob, elo.update, elo.model.frame

Examples

```
elo.calc(c(1, 0), c(1500, 1500), c(1500, 1600), k = 20)

dat <- data.frame(wins.A = c(1, 0), elo.A = c(1500, 1500),
                  elo.B = c(1500, 1600), k = c(20, 20))
elo.calc(wins.A ~ elo.A + elo.B + k(k), data = dat)
```

---

elo.colley *Compute a Colley matrix model for a matchup.*

Description

Compute a Colley matrix model for a matchup.

Usage

```
elo.colley(
  formula,
  data,
  family = "binomial",
  weights,
  na.action,
  subset,
  k = 1,
  ...,
  running = FALSE,
  skip = 0
)
```

Arguments

formula	A formula. See the help page for formulas for details.
data	A data.frame in which to look for objects in formula.
family	Argument passed to glm.
weights	A vector of weights. Note that these weights are used in the Colley matrix creation, but not the regression.
na.action	A function which indicates what should happen when the data contain NAs.
subset	An optional vector specifying a subset of observations.
k	The fraction of a win to be assigned to the winning team. See "details".
...	Argument passed to glm.

`running` Logical, denoting whether to calculate "running" projected probabilities. If true, a model is fit for group 1 on its own to predict group 2, then groups 1 and 2 to predict 3, then groups 1 through 3 to predict 4, etc. Groups are determined in `formula`. Omitting a group term re-runs a glm model to predict each observation (a potentially time-consuming operation!)

`skip` Integer, denoting how many groups to skip before fitting the running models. This is helpful if groups are small, where glm would have trouble converging for the first few groups. The predicted values are then set to 0.5 for the skipped groups.

Details

See the vignette for details on this method. The differences in assigned scores (from the coefficients of the Colley matrix regression) are fed into a logistic regression model to predict wins or (usually) a linear model to predict margin of victory. In this setting, 'k' indicates the fraction of a win to be assigned to the winning team (and the fraction of a loss to be assigned to the losing team); setting `k = 1` (the default) emits the "Bias Free" ranking method presented by Colley. It is also possible to adjust the regression by setting the second argument of `adjust()`. As in `elo.glm`, the intercept represents the home-field advantage. Neutral fields can be indicated using the `neutral()` function, which sets the intercept to 0.

References

Colley W.N. Colley's Bias Free College Football Ranking Method: The Colley Matrix Explained. 2002.

See Also

`glm`, `summary.elo.colley`, `score`, `mov`, `elo.model.frame`

Examples

```
elo.colley(score(points.Home, points.Visitor) ~ team.Home + team.Visitor, data = tournament,
  subset = points.Home != points.Visitor)
```

---

elo.glm *Compute a (usually logistic) regression model for a series of matches.*

Description

Compute a (usually logistic) regression model for a series of matches.

Usage

```
elo.glm(
  formula,
  data,
  family = "binomial",
  weights,
  na.action,
  subset,
  ...,
  running = FALSE,
  skip = 0
)
```

Arguments

formula	A formula. See the help page for formulas for details.
data	A data.frame in which to look for objects in formula.
family	Argument passed to glm.
weights	Argument passed to glm.
na.action	A function which indicates what should happen when the data contain NAs.
subset	An optional vector specifying a subset of observations.
...	Argument passed to glm.
running	Logical, denoting whether to calculate "running" projected probabilities. If true, a model is fit for group 1 on its own to predict group 2, then groups 1 and 2 to predict 3, then groups 1 through 3 to predict 4, etc. Groups are determined in formula. Omitting a group term re-runs a glm model to predict each observation (a potentially time-consuming operation!)
skip	Integer, denoting how many groups to skip before fitting the running models. This is helpful if groups are small, where glm would have trouble converging for the first few groups. The predicted values are then set to 0.5 for the skipped groups.

Details

The formula syntax is the same as other elo functions. A data.frame of indicator variables is built, where an entry is 1 if a team is home, 0 if a team didn't play, and -1 if a team is a visitor. Anything passed to adjust() in formula is also put in the data.frame. A glm model is then run to predict wins or margin of victory.

With this setup, the intercept represents the home-field advantage. Neutral fields can be indicated using the neutral() function, which sets the intercept to 0.

Note that any weights specified in players() will be ignored.

This is essentially the Bradley-Terry model.

Value

An object of class `c("elo.glm", "glm")`. If `running==TRUE`, the class `"elo.glm.running"` is prepended.

References

https://en.wikipedia.org/wiki/Bradley

See Also

`glm`, `summary.elo.glm`, `score`, `mov`, `elo.model.frame`

Examples

```
data(tournament)
elo.glm(score(points.Home, points.Visitor) ~ team.Home + team.Visitor, data = tournament,
  subset = points.Home != points.Visitor)
elo.glm(mov(points.Home, points.Visitor) ~ team.Home + team.Visitor, data = tournament,
  family = "gaussian")
```

---

elo.markovchain *Compute a Markov chain model for a series of matches.*

Description

Compute a Markov chain model for a series of matches.

Usage

```
elo.markovchain(
  formula,
  data,
  family = "binomial",
  weights,
  na.action,
  subset,
  k = NULL,
  ...,
  running = FALSE,
  skip = 0
)
```

Arguments

formula	A formula. See the help page for formulas for details.
data	A data.frame in which to look for objects in formula.
family	Argument passed to glm.
weights	A vector of weights. Note that these weights are used in the Markov Chain model, but not the regression.
na.action	A function which indicates what should happen when the data contain NAs.
subset	An optional vector specifying a subset of observations.
k	The probability that the winning team is better given that they won. See details.
...	Argument passed to glm.
running	Logical, denoting whether to calculate "running" projected probabilities. If true, a model is fit for group 1 on its own to predict group 2, then groups 1 and 2 to predict 3, then groups 1 through 3 to predict 4, etc. Groups are determined in formula. Omitting a group term re-runs a glm model to predict each observation (a potentially time-consuming operation!)
skip	Integer, denoting how many groups to skip before fitting the running models. This is helpful if groups are small, where glm would have trouble converging for the first few groups. The predicted values are then set to 0.5 for the skipped groups.

Details

See the vignette for details on this method. The probabilities we call 'k' purely for convenience. The differences in assigned scores (from the stationary distribution pi) are fed into a logistic regression model to predict wins or (usually) a linear model to predict margin of victory. It is also possible to adjust the regression by setting the second argument of adjust(). As in elo.glm, the intercept represents the home-field advantage. Neutral fields can be indicated using the neutral() function, which sets the intercept to 0.

Note that by assigning probabilities in the right way, this function emits the Logistic Regression Markov Chain model (LRMC).

References

Kvam, P. and Sokol, J.S. A logistic regression/Markov chain model for NCAA basketball. Naval Research Logistics. 2006. 53; 788-803.

See Also

glm, summary.elo.markovchain, score, mov, elo.model.frame

Examples

```
elo.markovchain(score(points.Home, points.Visitor) ~ team.Home + team.Visitor, data = tournament,
  subset = points.Home != points.Visitor, k = 0.7)

elo.markovchain(mov(points.Home, points.Visitor) ~ team.Home + team.Visitor, family = "gaussian",
  data = tournament, k = 0.7)
```

`elo.model.frame` *Interpret formulas in* `elo` *functions*

Description

A helper function to create the `model.frame` for many `elo` functions.

Usage

```
elo.model.frame(
  formula,
  data,
  na.action,
  subset,
  k = NULL,
  ...,
  required.vars = "elos",
  warn.k = TRUE,
  ncol.k = 1,
  ncol.elos = 2
)
```

Arguments

`formula`	A formula. See the help page for formulas for details.
`data`	A `data.frame` in which to look for objects in `formula`.
`na.action`	A function which indicates what should happen when the data contain NAs.
`subset`	An optional vector specifying a subset of observations.
`k`	A constant k-value (or a vector, where appropriate).
`...`	Other arguments (not in use at this time).
`required.vars`	One or more of `c("wins", "elos", "k", "group", "regress")`, denoting which variables are required to appear in the final model.frame.
`warn.k`	Should a warning be issued if `k` is specified as an argument and in `formula`?
`ncol.k`	How many columns (`NCOL`) should `k` have?
`ncol.elos`	How many Elo columns are expected?

See Also

`elo.run`, `elo.calc`, `elo.update`, `elo.prob`

elo.mov *Create a "margin of victory" column*

Description

Create a "margin of victory" based on two teams' scores

Usage

```
mov(score.A, score.B = 0)
```

Arguments

score.A	Numeric; the score of the first team. Alternatively, this can be a pre-computed margin of victory which will get compared to 0.
score.B	Numeric; the score of the second team; default is 0, in case score.A is already a margin of victory..

Value

An object with class "elo.mov", denoting score.A = score.B.

See Also

score

Examples

```
mov(12, 10)
mov(10, 10)
mov(10, 12)
```

elo.mse *Calculate the mean square error*

Description

Calculate the mean square error (Brier score) for a model.

Usage

```
mse(object, ..., subset = TRUE)

brier(object, ..., subset = TRUE)

## S3 method for class 'elo.run'
mse(object, ..., subset = TRUE)

## S3 method for class 'elo.glm'
mse(object, ..., subset = TRUE)

## S3 method for class 'elo.running'
mse(object, running = TRUE, discard.skipped = FALSE, ..., subset = TRUE)

## S3 method for class 'elo.markovchain'
mse(object, ..., subset = TRUE)

## S3 method for class 'elo.winpct'
mse(object, ..., subset = TRUE)

## S3 method for class 'elo.colley'
mse(object, ..., subset = TRUE)
```

Arguments

`object`	An object
`...`	Other arguments (not used at this time).
`subset`	(optional) A vector of indices on which to calculate
`running`	logical, denoting whether to use the running predicted values.
`discard.skipped`	Logical, denoting whether to ignore the skipped observations in the calculation

Details

Even though logistic regressions don't use the MSE on the y=0/1 scale, it can still be informative. Note that the S3 method is `mse`.

`elo.prob` *Elo probability*

Description

Calculate the probability that team A beats team B. This is vectorized.

Usage

```
elo.prob(elo.A, ...)

## Default S3 method:
elo.prob(elo.A, elo.B, ..., elos = NULL, adjust.A = 0, adjust.B = 0)

## S3 method for class 'formula'
elo.prob(formula, data, na.action, subset, ..., elos = NULL)

## S3 method for class 'elo.multiteam.matrix'
elo.prob(elo.A, ..., elos = NULL)
```

Arguments

elo.A, elo.B	Numeric vectors of elo scores, or else vectors of teams.
...	Other arguments (not in use at this time).
elos	An optional named vector containing Elo ratings for all teams in formula or elo.A and elo.B.
adjust.A, adjust.B	Numeric vectors to adjust elo.A and elo.B by.
formula	A formula. See the help page for formulas for details.
data	A data.frame in which to look for objects in formula.
na.action	A function which indicates what should happen when the data contain NAs.
subset	An optional vector specifying a subset of observations.

Details

Note that formula can be missing the wins.A component. If present, it's ignored by elo.model.frame.

Value

A vector of Elo probabilities.

See Also

elo.update, elo.calc, elo.model.frame

Examples

```
elo.prob(1500, 1500)
elo.prob(c(1500, 1500), c(1500, 1600))

dat <- data.frame(wins.A = c(1, 0), elo.A = c(1500, 1500),
                  elo.B = c(1500, 1600), k = c(20, 20))
elo.prob(~ elo.A + elo.B, data = dat)

## Also works to include the wins and k:
elo.prob(wins.A ~ elo.A + elo.B + k(k), data = dat)
```

```
## Also allows teams
elo.prob(c("A", "B"), c("C", "C"), elos = c(A = 1500, B = 1600, C = 1500))
```

`elo.run` *Calculate running Elos for a series of matches.*

Description

Calculate running Elos for a series of matches.

Usage

```
elo.run(
  formula,
  data,
  na.action,
  subset,
  k = NULL,
  initial.elos = NULL,
  ...,
  prob.fun = elo.prob,
  update.fun = elo.update,
  verbose = TRUE
)
```

Arguments

`formula`	A formula. See the help page for formulas for details.
`data`	A `data.frame` in which to look for objects in `formula`.
`na.action`	A function which indicates what should happen when the data contain NAs.
`subset`	An optional vector specifying a subset of observations.
`k`	A constant k-value (or a vector, where appropriate).
`initial.elos`	An optional named vector containing initial Elo ratings for all teams in `formula`. If a single (unnamed) value is supplied, that value is applied to all teams. `NULL` (the default) sets all Elos to 1500.
`...`	Other arguments (not used at this time).
`prob.fun`	A function with at least 4 arguments: elo.A, elo.B, adjust.A, and adjust.B. It should return a predicted probability that team A wins. The values passed in will be scalars, and a scalar is expected as output.
`update.fun`	A function with at least 6 arguments: the same as `elo.update.default`. The function takes in the Elos, the win indicator, k, and any adjustments, and returns a value by which to update the Elos. The values passed in will be scalars, and a scalar is expected as output.
`verbose`	Should a message be issued when R is used (over C++)?

Details

elo.run is run two different ways: the first (default) uses C++ and may be up to 50 times faster, while the second (when prob.fun or update.fun are specified) uses R but also supports custom update functions. Prefer the first unless you really need a custom update function.

Value

An object of class "elo.run" or class "elo.run.regressed".

See Also

score, elo.run.helperselo.run helpers, elo.calc, elo.update, elo.prob, elo.model.frame.

Examples

```
data(tournament)
elo.run(score(points.Home, points.Visitor) ~ team.Home + team.Visitor,
        data = tournament, k = 20)

# Create non-constant 'k'
elo.run(score(points.Home, points.Visitor) ~ team.Home + team.Visitor +
        k(20*log(abs(points.Home - points.Visitor) + 1)), data = tournament)

# Adjust Elo for, e.g., home-field advantage
elo.run(score(points.Home, points.Visitor) ~ adjust(team.Home, 30) + team.Visitor,
        data = tournament, k = 20)

tournament$home.field <- 30
elo.run(score(points.Home, points.Visitor) ~ adjust(team.Home, home.field) + team.Visitor,
        data = tournament, k = 20)

# Regress the Elos back toward 1500 at the end of the half-season
elo.run(score(points.Home, points.Visitor) ~ adjust(team.Home, 30) +
        team.Visitor + regress(half, 1500, 0.2), data = tournament, k = 20)
```

---

elo.run.helpers *Helper functions for* elo.run

Description

as.matrix converts an Elo object into a matrix of running Elos. These are the Elos at the time of grouping, but before any regression takes place.

Usage

```
## S3 method for class 'elo.run'
as.matrix(x, ...)

## S3 method for class 'elo.run.regressed'
as.matrix(x, ...)

## S3 method for class 'elo.run'
as.data.frame(x, ...)

final.elos(x, ...)

## S3 method for class 'elo.run'
final.elos(x, ...)

## S3 method for class 'elo.run.regressed'
final.elos(x, regressed = FALSE, ...)
```

Arguments

x	An object of class "elo.run" or class "elo.run.regressed".
...	Other arguments (Not in use at this time).
regressed	Logical, denoting whether to use the post-regressed (TRUE) or pre-regressed (FALSE) final Elos. Note that TRUE only makes sense when the final Elos were regressed one last time (i.e., if the last element of the regress()) vector yields TRUE).

Details

as.data.frame converts the "elos" component of an object from elo.run into a data.frame.

final.elos is a generic function to extract the last Elo per team.

Value

A matrix, a data.frame, or a named vector.

See Also

elo.run

Examples

```
e <- elo.run(score(points.Home, points.Visitor) ~ team.Home + team.Visitor + group(week),
             data = tournament, k = 20)
head(as.matrix(e))
str(as.data.frame(e))
final.elos(e)
```

`elo.run.multiteam` *Calculate running Elos for a series of multi-team matches.*

Description

Calculate running Elos for a series of multi-team matches.

Usage

```
elo.run.multiteam(
  formula,
  data,
  na.action,
  subset,
  k = NULL,
  initial.elos = NULL,
  ...
)
```

Arguments

`formula`	A one-sided formula with a `multiteam()` object. See also the the help page for formulas for details.
`data`	A `data.frame` in which to look for objects in `formula`.
`na.action`	A function which indicates what should happen when the data contain NAs.
`subset`	An optional vector specifying a subset of observations.
`k`	A constant k-value (or a vector, where appropriate).
`initial.elos`	An optional named vector containing initial Elo ratings for all teams in `formula`. If a single (unnamed) value is supplied, that value is applied to all teams. `NULL` (the default) sets all Elos to 1500.
`...`	Other arguments (not used at this time).

Details

This is like `elo.run` (and in fact it runs `elo.run` in the background). The formula takes a `multiteam()` object, which assumes that teams "win" in a well-ordered ranking. It assumes that the first place team beats all other teams, that the second place team loses to the first but beats the others, etc. In that regard, `elo.run.multiteam` reduces to `elo.run` when the number of teams (`ncol(multiteam())`) is 2

However, this is less flexible than `elo.run`, because (1) there cannot be ties; (2) it does not accept adjustments; and (3) k is constant within a "game"

Examples

```
data(tournament.multiteam)
elo.run.multiteam(~ multiteam(Place_1, Place_2, Place_3, Place_4),
                  data = tournament.multiteam, subset = -28, k = 20)
```

elo.update *Elo updates*

Description

Calculate the update value for a given Elo matchup. This is used in elo.calc, which reports the post-update Elo values. This is vectorized.

Usage

```
elo.update(wins.A, ...)

## Default S3 method:
elo.update(wins.A, elo.A, elo.B, k, ..., adjust.A = 0, adjust.B = 0)

## S3 method for class 'formula'
elo.update(formula, data, na.action, subset, k = NULL, ...)
```

Arguments

wins.A	Numeric vector of wins by team A.
...	Other arguments (not in use at this time).
elo.A, elo.B	Numeric vectors of elo scores.
k	A constant k-value (or a vector, where appropriate).
adjust.A, adjust.B	Numeric vectors to adjust elo.A and elo.B by.
formula	A formula. See the help page for formulas for details.
data	A data.frame in which to look for objects in formula.
na.action	A function which indicates what should happen when the data contain NAs.
subset	An optional vector specifying a subset of observations.

Value

A vector of Elo updates.

See Also

elo.prob, elo.calc, elo.model.frame

Examples

```
elo.update(c(1, 0), c(1500, 1500), c(1500, 1600), k = 20)

dat <- data.frame(wins.A = c(1, 0), elo.A = c(1500, 1500),
                  elo.B = c(1500, 1600), k = c(20, 20))
elo.update(wins.A ~ elo.A + elo.B + k(k), data = dat)
```

`elo.winpct` *Compute a (usually logistic) regression based on win percentage for a series of matches.*

Description

Compute a (usually logistic) regression based on win percentage for a series of matches.

Usage

```
elo.winpct(
  formula,
  data,
  family = "binomial",
  weights,
  na.action,
  subset,
  ...,
  running = FALSE,
  skip = 0
)
```

Arguments

`formula`	A formula. See the help page for formulas for details.
`data`	A `data.frame` in which to look for objects in `formula`.
`family`	Argument passed to `glm`.
`weights`	A vector of weights. Note that these are used in calculating wins and losses but not in the regression.
`na.action`	A function which indicates what should happen when the data contain NAs.
`subset`	An optional vector specifying a subset of observations.
`...`	Argument passed to `glm`.
`running`	Logical, denoting whether to calculate "running" projected probabilities. If true, a model is fit for group 1 on its own to predict group 2, then groups 1 and 2 to predict 3, then groups 1 through 3 to predict 4, etc. Groups are determined in `formula`. Omitting a group term re-runs a glm model to predict each observation (a potentially time-consuming operation!)
`skip`	Integer, denoting how many groups to skip before fitting the running models. This is helpful if groups are small, where glm would have trouble converging for the first few groups. The predicted values are then set to 0.5 for the skipped groups.

Details

Win percentages are first calculated. Anything passed to `adjust()` in `formula` is also put in the data.frame. A `glm` model is then run to predict wins or margin of victory.

With this setup, the intercept represents the home-field advantage. Neutral fields can be indicated using the `neutral()` function, which sets the intercept to 0.

See Also

`glm`, `summary.elo.winpct`, `score`, `mov`, `elo.model.frame`

Examples

```
elo.winpct(score(points.Home, points.Visitor) ~ team.Home + team.Visitor, data = tournament,
  subset = points.Home != points.Visitor)

elo.winpct(mov(points.Home, points.Visitor) ~ team.Home + team.Visitor, data = tournament,
  family = "gaussian")
```

---

favored.elo *Classify teams that are favored to win*

Description

Classify teams that are favored to win

Usage

```
favored(x, ..., subset = TRUE)

## S3 method for class 'elo.run'
favored(x, ..., subset = TRUE)

## S3 method for class 'elo.glm'
favored(x, ..., subset = TRUE)

## S3 method for class 'elo.running'
favored(x, running = TRUE, discard.skipped = FALSE, ..., subset = TRUE)

## S3 method for class 'elo.markovchain'
favored(x, ..., subset = TRUE)

## S3 method for class 'elo.winpct'
favored(x, ..., subset = TRUE)

## S3 method for class 'elo.colley'
favored(x, ..., subset = TRUE)
```

```
## Default S3 method:
favored(x, p.A, ...)
```

Arguments

x	An object from elo.run or elo.glm, or for the default method a vector representing wins.A.
...	Other arguments (not used at this time).
subset	(optional) A vector of indices on which to calculate
running	logical, denoting whether to use the running predicted values.
discard.skipped	Logical, denoting whether to ignore the skipped observations in the calculation
p.A	A vector of predicted win probabilities.

---

fitted.elo *Extract model values*

Description

Extract model values from elo functions.

Usage

```
## S3 method for class 'elo.run'
fitted(object, ...)

## S3 method for class 'elo.run'
residuals(object, ...)

## S3 method for class 'elo.running'
fitted(object, running = TRUE, ...)

## S3 method for class 'elo.glm'
fitted(object, ...)

## S3 method for class 'elo.markovchain'
fitted(object, ...)

## S3 method for class 'elo.winpct'
fitted(object, ...)

## S3 method for class 'elo.colley'
fitted(object, ...)
```

Arguments

object	An object.
...	Other arguments
running	logical, denoting whether to use the running predicted values.

Value

A vector of fitted values. For running values, it has an additional attribute denoting to which group (i.e., which model) the prediction belongs

---

players *Details on* elo *formulas and the specials therein*

Description

Details on elo functions and the special functions allowed in them to change functions' behaviors.

Usage

```
players(..., weights = NULL)

multiteam(...)

k(x, y = NULL)

adjust(x, adjustment)

regress(x, to, by, regress.unused = TRUE)

group(x)

neutral(x)
```

Arguments

...	Vectors to be coerced to character, which comprise of the players of a team.
weights	A vector giving the weights of Elo updates for the players in .... Ignored for elo.glm.
x, y	A vector.
adjustment	A single value or a vector of the same length as x: how much to adjust the Elos in x.
to	Numeric: what Elo to regress to. Can be a single value or named vector the same length as the number of teams.
by	Numeric: by how much should Elos be regressed toward to.
regress.unused	Logical: whether to continue regressing teams which have stopped playing.

Details

In the functions in this package, `formula` is usually of the form `wins.A ~ elo.A + elo.B`, where `elo.A` and `elo.B` are vectors of Elos, and `wins.A` is between 0 and 1, denoting whether team A (Elo A) won or lost (or something between). `elo.prob` also allows `elo.A` and `elo.B` to be character or factors, denoting which team(s) played. `elo.run` requires `elo.A` to be a vector of teams or a players matrix from `players()` (sometimes denoted by `"team.A"`), but `elo.B` can be either a vector of teams or players matrix (`"team.B"`) or else a numeric column (denoting a fixed-Elo opponent). `elo.glm` requires both to be a vector of teams or players matrix. `elo.markovchain` requires both to be a vector of teams.

`formula` accepts six special functions in it:

`k()` allows for complicated Elo updates. For constant Elo updates, use the `k =` argument instead of this special function. Note that `elo.markovchain` uses this function (or argument) as a convenient way of specifying transition probabilities. `elo.colley` uses this to indicate the fraction of a win to be assigned to the winning team.

`adjust()` allows for Elos to be adjusted for, e.g., home-field advantage. The second argument to this function can be a scalar or vector of appropriate length. This can also be used in `elo.glm` and `elo.markovchain` as an adjuster to the logistic regressions.

`regress()` can be used to regress Elos back to a fixed value after certain matches. Giving a logical vector identifies these matches after which to regress back to the mean. Giving any other kind of vector regresses after the appropriate groupings (see, e.g., `duplicated(..., fromLast = TRUE)`). The other three arguments determine what Elo to regress to (`to =`), by how much to regress toward that value (`by =`), and whether to continue regressing teams which have stopped playing (`regress.unused`, default = `TRUE`).

`group()` is used to group matches (by, e.g., week). For `elo.run`, Elos are not updated until the group changes. It is also fed to `as.matrix.elo.run`, giving the number of rows to return. to produce only certain rows of matrix output. It also determines how many models to run (and on what data) for `elo.glm` and `elo.markovchain` when `running=TRUE`.

`neutral()` is used in `elo.glm` and `elo.markovchain` to determine the intercept. In short, the intercept is `1 - neutral()`, denoting home-field advantage. Therefore, the column passed should be 0 (denoting home-field advantange) or 1 (denoting a neutral game). If omitted, all matches are assumed to have home field advantage.

`players()` is used for multiple players on a team contributing to an overall Elo. The Elo updates are then assigned based on the specified weights. The weights are ignored in `elo.glm`.

`multiteam()` is used for matchups consisting of multiple teams and is only valid in `elo.run.multiteam`.

`predict.elo` *Make Predictions on an* `elo` *Object*

Description

Make Predictions on an `elo` Object

Usage

```
## S3 method for class 'elo.run'
predict(object, newdata, ...)

## S3 method for class 'elo.run.regressed'
predict(object, newdata, regressed = FALSE, ...)

## S3 method for class 'elo.run.multiteam'
predict(object, newdata, ...)

## S3 method for class 'elo.glm'
predict(object, newdata, type = "response", ...)

## S3 method for class 'elo.running'
predict(object, newdata, running = TRUE, ...)

## S3 method for class 'elo.markovchain'
predict(object, newdata, ...)

## S3 method for class 'elo.colley'
predict(object, newdata, ...)

## S3 method for class 'elo.winpct'
predict(object, newdata, ...)
```

Arguments

object	An model from which to get predictions.
newdata	A new dataset containing the same variables as the call that made object. If missing, the predicted win probabilities from object will be returned.
...	Other arguments.
regressed	See the note on final.elos.
type	See predict.glm
running	logical, denoting whether to use the running predicted values. Only makes sense if newdata is missing.

Details

Note that the "elo.glm.running" objects will use a model fit on all the data to predict.

Value

A vector of win probabilities.

Examples

```
data(tournament)
```

```
t1 <- head(tournament, -3)
t2 <- tail(tournament, 3)
results <- elo.run(score(points.Home, points.Visitor) ~ team.Home + team.Visitor,
                   data = t1, k = 20)
predict(results)
predict(results, newdata = t2)

results <- elo.glm(score(points.Home, points.Visitor) ~ team.Home + team.Visitor, data = t1,
  subset = points.Home != points.Visitor)
predict(results)
predict(results, newdata = t2)

results <- elo.markovchain(score(points.Home, points.Visitor) ~ team.Home + team.Visitor, data = t1,
  subset = points.Home != points.Visitor, k = 0.7)
predict(results)
predict(results, newdata = t2)

results <- elo.colley(score(points.Home, points.Visitor) ~ team.Home + team.Visitor, data = t1,
  subset = points.Home != points.Visitor)
predict(results)
predict(results, newdata = t2)

results <- elo.winpct(score(points.Home, points.Visitor) ~ team.Home + team.Visitor, data = t1,
  subset = points.Home != points.Visitor, k = 0.7)
predict(results)
predict(results, newdata = t2)
```

`rank.teams` *Rank teams*

Description

Extract the rankings from Elo objects.

Usage

```
rank.teams(object, ties.method = "min", ...)

## S3 method for class 'elo.run'
rank.teams(object, ties.method = "min", ...)

## S3 method for class 'elo.run.regressed'
rank.teams(object, ties.method = "min", regressed = FALSE, ...)

## S3 method for class 'elo.glm'
rank.teams(object, ties.method = "min", ...)

## S3 method for class 'elo.markovchain'
rank.teams(object, ties.method = "min", ...)
```

```
## S3 method for class 'elo.winpct'
rank.teams(object, ties.method = "min", ...)

## S3 method for class 'elo.colley'
rank.teams(object, ties.method = "min", ...)
```

Arguments

object	An object.
ties.method	Passed to rank.
...	Other arguments
regressed	Passed to final.elos.

---

score *Create a 1/0/0.5 win "indicator"*

Description

Create a 1/0/0.5 win "indicator" based on two teams' scores, and test for "score-ness".

Usage

```
score(score.A, score.B)

is.score(x)
```

Arguments

score.A	Numeric; the score of the first team (whose wins are to be denoted by 1).
score.B	Numeric; the score of the second team (whose wins are to be denoted by 0).
x	An R object.

Value

For score, a vector containing 0, 1, and 0.5 (for ties). For is.score, TRUE or FALSE depending on whether all values of x are between 0 and 1 (inclusive).

See Also

score

Examples

```
score(12, 10)
score(10, 10)
score(10, 12)
```

summary.elo *Summarize an* elo *Object*

Description

Summarize an elo Object

Usage

```
## S3 method for class 'elo.run'
summary(object, ...)

## S3 method for class 'elo.glm'
summary(object, ...)

## S3 method for class 'elo.markovchain'
summary(object, ...)

## S3 method for class 'elo.colley'
summary(object, ...)

## S3 method for class 'elo.winpct'
summary(object, ...)
```

Arguments

object	An object to summarize.
...	Other arguments

Value

A summary of object.

See Also

favored, auc.elo.run, mse

Examples

```
summary(elo.run(score(points.Home, points.Visitor) ~ team.Home + team.Visitor,
  data = tournament, k = 20))
summary(elo.glm(score(points.Home, points.Visitor) ~ team.Home + team.Visitor,
  data = tournament))
mc <- elo.markovchain(score(points.Home, points.Visitor) ~ team.Home + team.Visitor,
  data = tournament, subset = points.Home != points.Visitor, k = 0.7)
summary(mc)
co <- elo.colley(score(points.Home, points.Visitor) ~ team.Home + team.Visitor,
  data = tournament, subset = points.Home != points.Visitor)
```

```
summary(co)
wp <- elo.winpct(score(points.Home, points.Visitor) ~ team.Home + team.Visitor,
  data = tournament, subset = points.Home != points.Visitor, k = 0.7)
summary(wp)
```

`tournament` `tournament`*: Mock data for examples*

Description

A fake dataset containing results from "animal-ball" matches.

Format

A data frame with 56 observations on the following 4 variables:

`team.Home` The home team for the match

`team.Visitor` The visiting team for the match

`points.Home` Number of points scored by the home team

`points.Visitor` Number of points scored by the visiting team

`week` Week Number

`half` The half of the season in which the match was played

Examples

```
data(tournament)
str(tournament)
```

`tournament.multiteam` `tournament.multiteam`*: Mock data for examples*

Description

A fake dataset containing results from "animal-ball" matches.

Format

A data frame with 56 observations on the following 4 variables:

`week` Week Number

`half` The half of the season in which the match was played

`Place_1` The first-place team

`Place_2` The second-place team

`Place_3` The third-place team

`Place_4` The fourth-place team

Index