Package ‘BTdecayLasso’

January 20, 2025

Type Package

Title Bradley-Terry Model with Exponential Time Decayed Log-Likelihood and Adaptive Lasso

Version 0.1.1

Description We utilize the Bradley-Terry Model to estimate the abilities of teams using paired comparison data. For dynamic approximation of current rankings, we employ the Exponential Decayed Log-likelihood function, and we also apply the Lasso penalty for variance reduction and grouping. The main algorithm applies the Augmented Lagrangian Method described by Masarotto and Varin (2012) <doi:10.1214/12-AOAS581>.

Imports optimx, ggplot2, stats

License GPL (>= 2)

LazyData true

RoxygenNote 7.2.3

NeedsCompilation no

Author Yunpeng Zhou [aut, cre], Jinfeng Xu [aut]

Maintainer Yunpeng Zhou <u3514104@connect.hku.hk>

Repository CRAN

Date/Publication 2023-12-07 11:40:02 UTC

Contents

boot.BTdecayLasso — *Compute the standard deviation of Bradley-Terry decay Lasso model by bootstrapping*

Description

Bootstrapping is done assuming that Maximum Likelihood's estimation reflects the true abilities. Same level of Lasso penalty "lambda" should be applied in different simulation models for Lasso induced estimation.

Usage

```
boot.BTdecayLasso(
  dataframe,
  ability,
  lambda,
  boot = 100,
  weight = NULL,
  decay.rate = 0,
  fixed = 1,
  thersh = 1e-05,
  max = 100,
  iter = 100
)
```

Arguments

`dataframe`	Generated using `BTdataframe` given raw data.
`ability`	A column vector of teams ability, the last row is the home parameter. The row number is consistent with the team's index shown in dataframe. It can be generated using `BTdataframe` given raw data.
`lambda`	The amount of Lasso penalty induced, only a single scalar is accepted in bootstrapping.
`boot`	Amount of simulations.
`weight`	Weight for Lasso penalty on different abilities.
`decay.rate`	The exponential decay rate. Usually ranging from (0, 0.01), A larger decay rate weights more importance to most recent matches and the estimated parameters reflect more on recent behaviour.
`fixed`	A teams index whose ability will be fixed as 0. The worstTeam's index can be generated using `BTdataframe` given raw data.
`thersh`	Threshold for convergence
`max`	Maximum weight for w_{ij} (weight used for Adaptive Lasso).
`iter`	Number of iterations used in L-BFGS-B algorithm.

Details

100 times of simulation will be done by default, user can adjust the numbers of simulation by input of boot. However, bootstrapping process is time consuming and usually 1000 time of simulations is enough to provide a stable result.

More detailed description of "lambda", "penalty" and "weight" are documented in `BTdecayLasso`.

summary() function follows S3 method can be applied to view the outputs.

Value

A list with class "boot" contain Lasso and Hybrid Lasso's bootstrapping's mean and standard deviation.

`Lasso`	Lasso bootstrapping's result. A three column matrix where first column is the original estimation, the second column is bootstrapping mean and the last column is the bootstrapping standard deviation
`HYBRID.Lasso`	HYBRID Lasso bootstrapping's result. A three column matrix where the first column is the original estimation, the second column is bootstrapping mean and the last column is the bootstrapping standard deviation

References

Masarotto, G. and Varin, C.(2012) The Ranking Lasso and its Application to Sport Tournaments. *The Annals of Applied Statistics* **6** 1949–1970.

Zou, H. (2006) The adaptive lasso and its oracle properties. *J.Amer.Statist.Assoc* **101** 1418–1429.

See Also

`BTdataframe` for dataframe initialization, `BTdecayLasso` for detailed description

---

`BTdataframe` *Dataframe initialization*

Description

Dataframe initialization

Usage

```
BTdataframe(dataframe, home = TRUE)
```

Arguments

`dataframe`	Raw dataframe input, an example data "NFL2010" is attached in package for reference The raw data is a dataframe with 5 columns. First column is home teams. Second column is away teams. Third column is the number of wins of home teams (if home team defeats away team, record 1 here, 0 otherwise). Fourth column is the number of wins of away teams (if home team defeats away team, record 0 here, 1 otherwise). Fifth column is a scalar of time when the match is played until now (Time lag). Any time scale can be used here. "NFL2010" applies the unit of day.
`home`	Whether home effect will be considered, the default is TRUE.

Details

Initial the raw dataframe and return an un-estimated ability vector and the worst team who loses most.

Note that even if the tournament does not have any home team or away team, you can still provide the match results according to the description above regardless of who is at home and who is away. By selecting the home = FALSE, We duplicate the dataset, switch the home, away teams and also the home, away match results. Then this dataset will be attached to the original dataset and all home and away win's number will be divided by 2. MLE estimation of home effect is proved to be an exact 0.

The elimination of home effect by duplicating the original dataset will be less efficient than eliminating the home parameter directly in iterations. Since most games such as football, basketball have home effect and this method provides an idea of handling the case where some games have home effect and some games are played on neutral place, this method is applied here.

Value

`dataframe`	dataframe for Bradley-Terry run
`ability`	Initial ability vector for iterations
`worstTeam`	The worst team whose ability can be set as 0 during any model's run

---

`BTdecay` *Bradley-Terry Model with Exponential Decayed weighted likelihood*

Description

Exponential decay rate is applied to the likelihood function to achieve a better track of current abilities. When "decay.rate" is setting as 0, this is a standard Bradley-Terry Model whose estimated parameters are equivalent to package "BradleyTerry2". Further detailed description is attached in `BTdecayLasso`.

Usage

```
BTdecay(dataframe, ability, decay.rate = 0, fixed = 1, iter = 100)
```

Arguments

`dataframe`	Generated using `BTdataframe` given raw data.
`ability`	A column vector of teams ability, the last row is the home parameter. The row number is consistent with the team's index shown in dataframe. It can be generated using `BTdataframe` given raw data.
`decay.rate`	The exponential decay rate. Usually ranging from (0, 0.01), A larger decay rate weights more importance to most recent matches and the estimated parameters reflect more on recent behaviour.
`fixed`	A teams index whose ability will be fixed as 0. The worstTeam's index can be generated using `BTdataframe` given raw data.
`iter`	Number of iterations used in L-BFGS-B algorithm.

Details

The standard Bradley-Terry Model defines the winning probability of i against j,

$$P(Y_{ij} = 1) = \frac{\exp(\tau h_{ij}^{t_k} + \mu_i - \mu_j)}{1 + \exp(\tau h_{ij}^{t_k} + \mu_i - \mu_j)}$$

τ is the home parameter and μ_i is the team i's ability score. h_{ij} takes 1 if team i is at home, -1 otherwise. Given, a complete tournament's result. The objective likelihood function with an exponential decay rate is,

$$\sum_{k=1}^{n} \sum_{i<j} \exp(-\alpha t_k) \cdot (y_{ij}(\tau h_{ij}^{t_k} + \mu_i - \mu_j) - \log(1 + \exp(\tau h_{ij}^{t_k} + \mu_i - \mu_j)))$$

where n is the number of matches, α is the exponential decay rate and y_{ij} takes 0 if i is defeated by j, 1 otherwise. t_k is the time lag (time until now). This likelihood function is optimized using L-BFGS-B method with package **optimr** and summary() function with S3 method can be applied to view the outputs.

Value

List with class "BT" contains estimated abilities and convergent code, 0 stands for convergence reaches, 1 stands for convergence not reaches. If 1 is returned, we suggest that decay rate should be set lower. Bradley-Terry model fails to model the situation when a team wins or loses in all matches. If a high decay rate is considered, a team who only loses or wins 1 matches long time ago will also causes the same problem.

`ability`	Estimated ability scores
`convergence`	0 stands for convergent, 1 stands for not convergent
`decay.rate`	Decay rate of this model

Examples

```
##Initializing Dataframe
x <- BTdataframe(NFL2010)
```

```
##Standard Bradley-Terry Model optimization
y <- BTdecay(x$dataframe, x$ability, decay.rate = 0, fixed = x$worstTeam)
summary(y)

##Dynamic approximation of current ability scores using exponential decayed likelihood.
##If we take decay.rate = 0.005
##Match happens one month before will weight exp(-0.15)=0.86 on log-likelihood function
z <- BTdecay(x$dataframe, x$ability, decay.rate = 0.005, fixed = x$worstTeam)
summary(z)
```

`BTdecayLasso` *Bradley-Terry Model with Exponential Decayed weighted likelihood and Adaptive Lasso*

Description

Bradley-Terry model is applied for paired comparison data. Teams' ability score is estimated by maximizing log-likelihood function.

To achieve a better track of current abilities, we apply an exponential decay rate to weight the log-likelihood function. The most current matches will weight more than previous matches. Parameter "decay.rate" in most functions of this package is used to set the amount of exponential decay rate. decay.rate should be non-negative and the appropriate range of it depends on time scale in original dataframe. (see `BTdataframe` and parameter "dataframe"'s definition of fifth column) For example, a unit of week with a "decay.rate" 0.007 is equivalent to the unit of day with "decay.rate" 0.001. Usually, for sports matches, if we take the unit of day, it's ranging from 0 to 0.01. The higher choice of "decay.rate", the better track of current teams' ability with a side effect of higher variance.

If "decay.rate" is too large, for example "0.1" with a unit of day, $\exp(-0.7) = 0.50$. Only half weight will be add to the likelihood for matches played one week ago and $\exp(-3.1) = 0.05$ suggests that previous matches took place one month ago will have little effect. Therefore, Only a few matches are accounted for ability's estimation. It will lead to a very high variance and uncertainty. Since standard Bradley-Terry model can not handle the case where there is a team who wins or loses all matches, such estimation may not provide convergent results. Thus, if our estimation provides divergent result, an error will be returned and we suggest user to chose a smaller "decay.rate" or adding more match results into the same modeling period.

By default, the Adaptive Lasso is implemented for variance reduction and team's grouping. Adaptive Lasso is proved to have good grouping property. Apart from adaptive lasso, user can define own weight for different Lasso constraint $|\mu_i - \mu_j|$ where μ_i is team i's ability.

Also by default, the whole Lasso path will be run. Similar to package "glmnet", user can provide their own choice of Lasso penalty "lambda" and determine whether the whole Lasso path will be run (since such run is time-consuming). However, we suggest that if user is not familiar with the actual relationship among lambda, the amount of penalty, the amount of shrinkage and grouping effect, a whole Lasso path should be run and selection of an appropriate lambda is done by AIC or BIC criteria using `BTdecayLassoC` (since this model is time related, cross-validation method cannot be applied). Also, users can use `BTdecayLassoF` to run with a specific Lasso penalty ranging from 0 to 1 (1 penalty means all estimators will shrink to 0).

Two sets of estimated abilities will be given, the biased Lasso estimation and the HYBRID Lasso's estimation. HYBRID Lasso estimation solves the restricted Maximum Likelihood optimization based on the group determined by Lasso's estimation (Different team's ability will converges to the same value if Lasso penalty is added and these teams' ability is setting to be equal as a restriction).

In addition, summary() using S3 method can be applied to view the outputs.

Usage

```
BTdecayLasso(
  dataframe,
  ability,
  lambda = NULL,
  weight = NULL,
  path = TRUE,
  decay.rate = 0,
  fixed = 1,
  thersh = 1e-05,
  max = 100,
  iter = 100
)
```

Arguments

`dataframe`	Generated using `BTdataframe` given raw data.
`ability`	A column vector of teams ability, the last row is the home parameter. The row number is consistent with the team's index shown in dataframe. It can be generated using `BTdataframe` given raw data.
`lambda`	The amount of Lasso penalty induced. The input should be a positive scalar or a sequence.
`weight`	Weight for Lasso penalty on different abilities.
`path`	whether the whole Lasso path will be run (plot.BTdecayLasso is enabled only if path = TRUE)
`decay.rate`	A non-negative exponential decay rate. Usually ranging from (0, 0.01), A larger decay rate weights more importance to most recent matches and the estimated parameters reflect more on recent behaviour.
`fixed`	A teams index whose ability will be fixed as 0. The worstTeam's index can be generated using `BTdataframe` given raw data.
`thersh`	Threshold for convergence used for Augmented Lagrangian Method.
`max`	Maximum weight for w_{ij} (weight used for Adaptive Lasso)
`iter`	Number of iterations used in L-BFGS-B algorithm.

Details

According to `BTdecay`, the objective likelihood function to be optimized is,

$$\sum_{k=1}^{n}\sum_{i<j}\exp(-\alpha t_k)\cdot(y_{ij}(\tau h_{ij}^{t_k}+\mu_i-\mu_j)-\log(1+\exp(\tau h_{ij}^{t_k}+\mu_i-\mu_j)))$$

The Lasso constraint is given as,

$$\sum_{i<j} w_{ij} \, |\mu_i - \mu_j| \leq s$$

where w_{ij} are predefined weight. For Adaptive Lasso, $\left| w_{ij} = 1/(\mu_i^{MLE} - \mu_j^{MLE}) \right|$.

Maximize this constraint objective function is equivalent to minimizing the following equation,

$$-l(\mu, \tau) + \lambda \sum_{i<j} w_{ij} |\mu_i - \mu_j|$$

Where $-l(\mu, \tau)$ is taking negative value of objective function above. Increase "lambda" will decrease "s", their relationship is monotone. Here, we define "penalty" as $1 - s/\max(s)$. Thus, "lambda" and "penalty" has a positive correlation.

Value

`ability`	Estimated ability scores with user given lambda
`likelihood`	Negative likelihood of objective function with user given lambda
`df`	Degree of freedom with user given lambda(number of distinct μ)
`penalty`	$s/max(s)$ with user given lambda
`Lambda`	User given lambda
`ability.path`	if path = TRUE, estimated ability scores on whole Lasso path
`likelihood.path`	if path = TRUE, negative likelihood of objective function on whole Lasso path
`df.path`	if path = TRUE, degree of freedom on whole Lasso path(number of distinct μ)
`penalty.path`	if path = TRUE, $s/max(s)$ on whole Lasso path
`Lambda.path`	if path = TRUE, Whole Lasso path
`path`	Whether whole Lasso path will be run
`HYBRID.ability.path`	If path = TRUE, the whole path of evolving of HYBRID ability
`HYBRID.likelihood.path`	if path = TRUE, the whole path of HYBRID likelihood

References

Masarotto, G. and Varin, C.(2012) The Ranking Lasso and its Application to Sport Tournaments. *The Annals of Applied Statistics* **6** 1949–1970.

Zou, H. (2006) The adaptive lasso and its oracle properties. *J.Amer.Statist.Assoc* **101** 1418–1429.

See Also

`BTdataframe` for dataframe initialization, `plot.swlasso`, `plot.wlasso` are used for Lasso path plot if path = TRUE in this function's run

Examples

```
##Initializing Dataframe
x <- BTdataframe(NFL2010)

##The following code runs the main results
##Usually a single lambda's run will take 1-20 s
##The whole Adaptive Lasso run will take 5-20 min

##BTdecayLasso run with exponential decay rate 0.005 and
##lambda 0.1, use path = TRUE if you want to run whole LASSO path
y1 <- BTdecayLasso(x$dataframe, x$ability, lambda = 0.1, path = FALSE,
                   decay.rate = 0.005, fixed = x$worstTeam)
summary(y1)

##Defining equal weight
##Note that comparing to Adaptive weight, the user defined weight may not be
##efficient in groupiing. Therefore, to run the whole Lasso path
##(evolving of distinct ability scores), it may take a much longer time.
##We recommend the user to apply the default setting,
##where Adaptive Lasso will be run.

n <- nrow(x$ability) - 1
w2 <- matrix(1, nrow = n, ncol = n)
w2[lower.tri(w2, diag = TRUE)] <- 0

##BTdecayLasso run with exponential decay rate 0.005 and with a specific lambda 0.1
y2 <- BTdecayLasso(x$dataframe, x$ability, lambda = 0.1, weight = w2,
                   path = FALSE, decay.rate = 0.005, fixed = x$worstTeam)

summary(y2)
```

`BTdecayLassoC` *Bradley-Terry Model with Exponential Decayed weighted likelihood and weighted Lasso with AIC or BIC criteria*

Description

Model selection via AIC or BIC criteria. For Lasso estimators, the degree of freedom is the number of distinct groups of estimated abilities.

Usage

```
BTdecayLassoC(
  dataframe,
  ability,
  weight = NULL,
  criteria = "AIC",
  type = "HYBRID",
  model = NULL,
  decay.rate = 0,
  fixed = 1,
  thersh = 1e-05,
  iter = 100,
  max = 100
)
```

Arguments

`dataframe`	Generated using `BTdataframe` given raw data.
`ability`	A column vector of teams ability, the last row is the home parameter. The row number is consistent with the team's index shown in dataframe. It can be generated using `BTdataframe` given raw data.
`weight`	Weight for Lasso penalty on different abilities
`criteria`	"AIC" or "BIC"
`type`	"HYBRID" or "LASSO"
`model`	An Lasso path object with class wlasso or swlasso. If NULL, the whole lasso path will be run.
`decay.rate`	The exponential decay rate. Usually ranging from (0, 0.01), A larger decay rate weights more importance to most recent matches and the estimated parameters reflect more on recent behaviour.
`fixed`	A teams index whose ability will be fixed as 0. The worstTeam's index can be generated using `BTdataframe` given raw data.
`thersh`	Threshold for convergence
`iter`	Number of iterations used in L-BFGS-B algorithm.
`max`	Maximum weight for w_{ij} (weight used for Adaptive Lasso)

Details

This function is usually run after the run of whole Lasso path. "model" parameter is obtained by whole Lasso pass's run using `BTdecayLasso`. If no model is provided, this function will run Lasso path first (time-consuming).

Users can select the information score added to HYBRID Lasso's likelihood or original Lasso's likelihood. ("HYBRID" is recommended)

summary() function can be applied to view the outputs.

Value

`Score`	Lowest AIC or BIC score
`Optimal.degree`	The degree of freedom where lowest AIC or BIC score is achieved
`Optimal.ability`	The ability where lowest AIC or BIC score is achieved
`ability`	Matrix contains all abilities computed in this algorithm

`Optimal.lambda`	The lambda where lowest score is attained
`Optimal.penalty`	The penalty (1- s/$\max(s)$) where lowest score is attained
`type`	Type of model selection method
`decay.rate`	Decay rate of this model

References

Masarotto, G. and Varin, C.(2012) The Ranking Lasso and its Application to Sport Tournaments. *The Annals of Applied Statistics* **6** 1949–1970.

Zou, H. (2006) The adaptive lasso and its oracle properties. *J.Amer.Statist.Assoc* **101** 1418–1429.

See Also

`BTdataframe` for dataframe initialization, `BTdecayLasso` for obtaining a whole Lasso path

---

`BTdecayLassoF` *Bradley-Terry Model with Exponential Decayed weighted likelihood and Adaptive Lasso with a given penalty rate*

Description

This function provides a method to computed the estimated abilities and lambda given an intuitive fixed Lasso penalty rate. Since in Lasso method, the selection of lambda varies a lot with respect to different datasets. We can keep the consistency of amount of Lasso penalty induced in different datasets from different period by setting a fixed Lasso penalty rate "penalty". Please refer to `BTdecayLasso` for the definition of "penalty" and its relationship with "lambda".

Usage

```
BTdecayLassoF(
  dataframe,
  ability,
  penalty,
  decay.rate = 0,
  fixed = 1,
  thersh = 1e-05,
  max = 100,
  iter = 100
)
```

Arguments

`dataframe`	Generated using `BTdataframe` given raw data.
`ability`	A column vector of teams ability, the last row is the home parameter. It can be generated using `BTdataframe` given raw data. The row number is consistent with the team's index shown in dataframe. It can be generated using `BTdataframe` given raw data.
`penalty`	The amount of Lasso penalty induced (1-s/max(s)) where is the sum of Lasso penalty part.
`decay.rate`	The exponential decay rate. Usually ranging from (0, 0.01), A larger decay rate weights more importance to most recent matches and the estimated parameters reflect more on recent behaviour.
`fixed`	A teams index whose ability will be fixed as 0. The worstTeam's index can be generated using `BTdataframe` given raw data.
`thersh`	Threshold for convergence
`max`	Maximum weight for w_{ij} (weight used for Adaptive Lasso)
`iter`	Number of iterations used in L-BFGS-B algorithm.

Details

The estimated ability given fixed penalty $p = 1 - s/\max(s)$ where s is the sum of Lasso penalty part. When p = 0, this model is reduced to a standard Bradley-Terry Model. When p = 1, all ability scores are shrinking to 0.

The parameter "penalty" should be ranging from 0.01 to 0.99 due to the iteration's convergent error.

summary() function can be applied to view the outputs.

Value

The list with class "BTF" contains estimated abilities and other parameters.

`ability`	Estimated ability scores
`df`	Degree of freedom (number of distinct μ)
`penalty`	Amount of Lasso Penalty
`decay.rate`	Exponential decay rate
`lambda`	Corresponding Lasso lambda given penalty rate

References

Masarotto, G. and Varin, C.(2012) The Ranking Lasso and its Application to Sport Tournaments. *The Annals of Applied Statistics* **6** 1949–1970.

Zou, H. (2006) The adaptive lasso and its oracle properties. *J.Amer.Statist.Assoc* **101** 1418–1429.

See Also

`BTdataframe` for dataframe initialization, `BTdecayLasso` for detailed description

---

`NFL2010` *The 2010 NFL Regular Season*

Description

A dataframe containing all match results with 5 columns

Usage

```
NFL2010
```

Format

A dataframe containing all match results with 5 columns

home.team Team who plays at home

away.team Team who plays away

home.win Take "1" if home team wins

away.win Take "1" if away team wins

date Number of days until now

`plot.swlasso` *Plot the Lasso path*

Description

Plot the whole lasso path run by BTdecayLasso() with given lambda and path = TRUE

Usage

```
##S3 method for class "swlasso"
```

Arguments

`x`	Object with class "swlasso"
`...`	Further arguments pass to or from other methods

`plot.wlasso` *Plot the Lasso path*

Description

Plot the whole lasso path run by BTdecayLasso() with lambda = NULL and path = TRUE

Usage

```
##S3 method for class "wlasso"
```

Arguments

`x`	Object with class "wlasso"
`...`	Further arguments pass to or from other methods

Index