Package ‘survcompare’

January 22, 2024

Title Compares Cox and Survival Random Forests to Quantify Nonlinearity

Version 0.1.2

Date 2024-01-19

Description Performs repeated nested cross-validation for Cox Proportionate Hazards, Cox Lasso, Survival Random Forest, and their ensemble. Returns internally validated concordance index, time-dependent area under the curve, Brier score, calibration slope, and statistical testing of non-linear ensemble outperforming the baseline Cox model. In this, it helps researchers to quantify the gain of using a more complex survival model, or justify its redundancy. Equally, it shows the performance value of the non-linear and interaction terms, and may highlight the need of further feature transformation. Further details can be found in Shamsutdinova, Stamate, Roberts, & Stahl (2022) ``Combining Cox Model and Tree-Based Algorithms to Boost Performance and Preserve Interpretability for Health Outcomes'' <doi:10.1007/978-3-031-08337-2_15>, where the method is described as Ensemble 1.

License GPL (>= 3)

Encoding UTF-8

Depends R (>= 4.1), survival (>= 3.0)

Imports stats, timeROC, caret, glmnet, randomForestSRC

RoxygenNote 7.2.3

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Maintainer Diana Shamsutdinova <`diana.shamsutdinova.github@gmail.com`>

Config/testthat/edition 3

NeedsCompilation no

Author Diana Shamsutdinova [aut, cre] (<`https://orcid.org/0000-0003-2434-3641`>), Daniel Stahl [aut] (<`https://orcid.org/0000-0001-7987-6619`>)

Repository CRAN

Date/Publication 2024-01-22 17:20:02 UTC

R topics documented:

`cox_calibration_stats` *Calibration stats of a fitted Cox PH model*

Description

Computes calibration alpha and slope for a fitted coxph model in the data.

Crowson, C. S., Atkinson, E. J., & Therneau, T. M. (2016). Assessing calibration of prognostic risk scores. Statistical methods in medical research, 25(4), 1692-1706.

https://journals.sagepub.com/doi/pdf/10.1177/0962280213497434

Usage

```
cox_calibration_stats(cox_model, test_data)
```

Arguments

`cox_model`	fitted cox model, namely, coxph() object
`test_data`	test data, should be a data frame with "time" and "event" columns for survival outcome

Value

c(calibration alpha, calibration slope)

`linear_beta` *Auxiliary function for simulatedata functions*

Description

Auxiliary function for simulatedata functions

Usage

```
linear_beta(df)
```

Arguments

`df`	data

`predict.survensemble` *Predicts event probability for a fitted survensemble*

Description

predict.survensemble

Usage

```
## S3 method for class 'survensemble'
predict(object, newdata, fixed_time, oob = FALSE, ...)
```

Arguments

`object`	trained survensemble model
`newdata`	test data
`fixed_time`	time for which probabilities are computed
`oob`	TRUE/FALSE , default is FALSE, if out of bag predictions are to be made from SRF
`...`	other parameters to pass

Value

matrix of predictions for observations in newdata by times

`print.survcompare` *Print survcompare object*

Description

Print survcompare object

Usage

```
## S3 method for class 'survcompare'
print(x, ...)
```

Arguments

`x`	output object of the survcompare function
`...`	additional arguments to be passed

Value

x

`print.survensemble` *Prints trained survensemble object*

Description

Prints trained survensemble object

Usage

```
## S3 method for class 'survensemble'
print(x, ...)
```

Arguments

`x`	survensemble object
`...`	additional arguments to be passed

Value

x

`print.survensemble_cv` *Prints survensemble_cv object*

Description

Prints survensemble_cv object

Usage

```
## S3 method for class 'survensemble_cv'
print(x, ...)
```

Arguments

`x`	survensemble_cv object
`...`	additional arguments to be passed

Value

x

`simulate_crossterms` *Simulated sample with survival outcomes with non-linear and cross-term dependencies*

Description

Simulated sample with exponentially or Weibull distributed time-to-event; log-hazard depends non-linearly on risk factors, and includes cross-terms.

Usage

```
simulate_crossterms(
  N = 300,
  observe_time = 10,
  percentcensored = 0.75,
  randomseed = NULL,
  lambda = 0.1,
  distr = "Exp",
  rho_w = 1,
  drop_out = 0.3
)
```

Arguments

`N`	sample size, 300 by default
`observe_time`	study's observation time, 10 by default
`percentcensored`	expected number of non-events by observe_time, 0.75 by default (i.e. event rate is 0.25)
`randomseed`	random seed for replication
`lambda`	baseline hazard rate, 0.1 by default
`distr`	time-to-event distribution, "Exp" for exponential (default), "W" for Weibull
`rho_w`	shape parameter for Weibull distribution, 0.3 by default
`drop_out`	expected rate of drop out before observe_time, 0.3 by default

Value

data frame; "time" and "event" columns describe survival outcome; predictors are "age", "sex", "hyp", "bmi"

Examples

```
mydata <- simulate_crossterms()
head(mydata)
```

`simulate_linear` *Simulated sample with survival outcomes with linear dependencies*

Description

Simulated sample with exponentially or Weibull distributed time-to-event; log-hazard (lambda parameter) depends linearly on risk factors.

Usage

```
simulate_linear(
  N = 300,
  observe_time = 10,
  percentcensored = 0.75,
  randomseed = NULL,
  lambda = 0.1,
  distr = "Exp",
  rho_w = 1,
  drop_out = 0.3
)
```

Arguments

`N`	sample size, 300 by default
`observe_time`	study's observation time, 10 by default
`percentcensored`	expected number of non-events by observe_time, 0.75 by default (i.e. event rate is 0.25)
`randomseed`	random seed for replication
`lambda`	baseline hazard rate, 0.1 by default
`distr`	time-to-event distribution, "Exp" for exponential (default), "W" for Weibull
`rho_w`	shape parameter for Weibull distribution, 0.3 by default
`drop_out`	expected rate of drop out before observe_time, 0.3 by default

Value

data frame; "time" and "event" columns describe survival outcome; predictors are "age", "sex", "hyp", "bmi"

Examples

```
mydata <- simulate_linear()
head(mydata)
```

`simulate_nonlinear` *Simulated sample with survival outcomes with non-linear dependencies*

Description

Simulated sample with exponentially or Weibull distributed time-to-event; log-hazard (lambda parameter) depends non-linearly on risk factors.

Usage

```
simulate_nonlinear(
  N = 300,
  observe_time = 10,
  percentcensored = 0.75,
  randomseed = NULL,
  lambda = 0.1,
  distr = "Exp",
  rho_w = 1,
  drop_out = 0.3
)
```

Arguments

`N`	sample size, 300 by default
`observe_time`	study's observation time, 10 by default
`percentcensored`	expected number of non-events by observe_time, 0.75 by default (i.e. event rate is 0.25)
`randomseed`	random seed for replication
`lambda`	baseline hazard rate, 0.1 by default
`distr`	time-to-event distribution, "Exp" for exponential (default), "W" for Weibull
`rho_w`	shape parameter for Weibull distribution, 0.3 by default
`drop_out`	expected rate of drop out before observe_time, 0.3 by default

Value

data frame; "time" and "event" columns describe survival outcome; predictors are "age", "sex", "hyp", "bmi"

Examples

```
mydata <- simulate_nonlinear()
head(mydata)
```

`srf_survival_prob_for_time` *Internal function to compute survival probability by time from a fitted survival random forest*

Description

Internal function to compute survival probability by time from a fitted survival random forest

Usage

```
srf_survival_prob_for_time(rfmodel, df_to_predict, fixed_time, oob = FALSE)
```

Arguments

`rfmodel`	pre-trained survsrf_train model
`df_to_predict`	test data
`fixed_time`	at which event probabilities are computed
`oob`	TRUE/FALSE use out-of-bag prediction

Value

output list: output$train, test, testaverage, traintaverage, time

Examples

```
df <- simulate_nonlinear()
#params<- c("age", "hyp", "bmi")
#s <- survsrf_train(df, params)
#p <- survsrf_predict(s, df, 5)
```

`summary.survcompare` *Summary of survcompare results*

Description

Summary of survcompare results

Usage

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

Arguments

`object`	output object of the survcompare function
`...`	additional arguments to be passed

Value

object

`summary.survensemble` *Prints summary of a trained survensemble object*

Description

Prints summary of a trained survensemble object

Usage

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

Arguments

`object`	survensemble object
`...`	additional arguments to be passed

Value

object

summary.survensemble_cv

Prints a summary of survensemble_cv object

Description

Prints a summary of survensemble_cv object

Usage

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

Arguments

`object`	survensemble_cv object
`...`	additional arguments to be passed

Value

object

survcompare

Cross-validates and compares Cox Proportionate Hazards and Survival Random Forest models

Description

The function performs a repeated nested cross-validation for

1. Cox-PH (survival package, survival::coxph) or Cox-Lasso (glmnet package, glmnet::cox.fit)
2. Ensemble of the Cox model and Survival Random Forest (randomForestSRC::rfsrc)
3. Survival Random Forest on its own, if train_srf = TRUE

The same random seed for the train/test splits are used for all models to aid fair comparison; and the performance metrics are computed for the tree models including Harrel's c-index, time-dependent AUC-ROC, time-dependent Brier Score, and calibration slope. The statistical significance of the performance differences between Cox-PH and Cox-SRF Ensemble is tested and reported.

The function is designed to help with the model selection by quantifying the loss of predictive performance (if any) if Cox-PH is used instead of a more complex model such as SRF which can capture non-linear and interaction terms, as well as non-proportionate hazards. The difference in performance of the Ensembled Cox and SRF and the baseline Cox-PH can be viewed as quantification of the non-linear and cross-terms contribution to the predictive power of the supplied predictors.

Cross-validates and compares Cox Proportionate Hazards and Survival Random Forest models

Usage

```
survcompare(
  df_train,
  predict_factors,
  predict_time = NULL,
  randomseed = NULL,
  useCoxLasso = FALSE,
  outer_cv = 3,
  inner_cv = 3,
  srf_tuning = list(),
  return_models = FALSE,
  repeat_cv = 2,
  train_srf = FALSE
)
```

Arguments

`df_train`	training data, a data frame with "time" and "event" columns to define the survival outcome
`predict_factors`	list of column names to be used as predictors
`predict_time`	prediction time of interest. If NULL, 0.90th quantile of event times is used
`randomseed`	random seed for replication
`useCoxLasso`	TRUE / FALSE, for whether to use regularized version of the Cox model, FALSE is default
`outer_cv`	k in k-fold CV
`inner_cv`	k in k-fold CV for internal CV to tune survival random forest hyper-parameters
`srf_tuning`	list of tuning parameters for random forest: 1) NULL for using a default tuning grid, or 2) a list("mtry"=c(...), "nodedepth" = c(...), "nodesize" = c(...))
`return_models`	TRUE/FALSE to return the trained models; default is FALSE, only performance is returned
`repeat_cv`	if NULL, runs once, otherwise repeats several times with different random split for CV, reports average of all
`train_srf`	TRUE/FALSE for whether to train SRF on its own, apart from the CoxPH->SRF ensemble. Default is FALSE as there is not much information in SRF itself compared to the ensembled version.

Value

outcome = list(data frame with performance results, fitted Cox models, fitted SRF)

Author(s)

Diana Shamsutdinova <`diana.shamsutdinova.github@gmail.com`>

Examples

```
df <-simulate_nonlinear(100)
srf_params <- list("mtry" = c(2), "nodedepth"=c(25), "nodesize" =c(15))
mysurvcomp <- survcompare(df, names(df)[1:4], srf_tuning = srf_params, outer_cv = 2, inner_cv =2)
summary(mysurvcomp)
```

`survcoxlasso_train` *Trains CoxLasso, using cv.glmnet(s="lambda.min")*

Description

Trains CoxLasso, using cv.glmnet(s="lambda.min")

Usage

```
survcoxlasso_train(
  df_train,
  predict.factors,
  inner_cv = 5,
  fixed_time = NaN,
  retrain_cox = FALSE,
  verbose = FALSE
)
```

Arguments

`df_train`	data frame with the data, "time" and "event" should describe survival outcome
`predict.factors`	list of the column names to be used as predictors
`inner_cv`	k in k-fold CV for lambda tuning
`fixed_time`	not used here, for internal use
`retrain_cox`	whether to re-train coxph on non-zero predictors; FALSE by default
`verbose`	TRUE/FALSE prints warnings if no predictors in Lasso

Value

fitted CoxPH object with coefficient of CoxLasso or re-trained CoxPH with non-zero CoxLasso if retrain_cox = FALSE or TRUE

survcox_cv *Cross-validates Cox or CoxLasso model*

Description

Cross-validates Cox or CoxLasso model

Usage

```
survcox_cv(
  df,
  predict.factors,
  fixed_time = NaN,
  outer_cv = 3,
  repeat_cv = 2,
  randomseed = NULL,
  return_models = FALSE,
  inner_cv = 3,
  useCoxLasso = FALSE
)
```

Arguments

`df`	data frame with the data, "time" and "event" for survival outcome
`predict.factors`	list of predictor names
`fixed_time`	at which performance metrics are computed
`outer_cv`	k in k-fold CV, default 3
`repeat_cv`	if NULL, runs once, otherwise repeats CV
`randomseed`	random seed
`return_models`	TRUE/FALSE, if TRUE returns all CV objects
`inner_cv`	k in the inner loop of k-fold CV, default is 3; only used if CoxLasso is TRUE
`useCoxLasso`	TRUE/FALSE, FALSE by default

Value

list of outputs

Examples

```
df <- simulate_nonlinear()
coxph_cv <- survcox_cv(df, names(df)[1:4])
summary(coxph_cv)
```

survcox_predict *Computes event probabilities from a trained cox model*

Description

Computes event probabilities from a trained cox model

Usage

```
survcox_predict(trained_model, newdata, fixed_time, interpolation = "constant")
```

Arguments

`trained_model`	pre-trained cox model of coxph class
`newdata`	data to compute event probabilities for
`fixed_time`	at which event probabilities are computed
`interpolation`	"constant" by default, can also be "linear", for between times interpolation for hazard rates

Value

returns matrix(nrow = length(newdata), ncol = length(fixed_time))

survcox_train *Trains CoxPH using survival package, or trains CoxLasso (cv.glmnet, lambda.min), and then re-trains survival:coxph on non-zero predictors*

Description

Trains CoxPH using survival package, or trains CoxLasso (cv.glmnet, lambda.min), and then re-trains survival:coxph on non-zero predictors

Usage

```
survcox_train(
  df_train,
  predict.factors,
  fixed_time = NaN,
  useCoxLasso = FALSE,
  retrain_cox = FALSE,
  inner_cv = 5
)
```

Arguments

`df_train`	data, "time" and "event" should describe survival outcome
`predict.factors`	list of the column names to be used as predictors
`fixed_time`	target time, NaN by default; needed here only to re-align with other methods
`useCoxLasso`	TRUE or FALSE
`retrain_cox`	if useCoxLasso is TRUE, whether to re-train coxph on non-zero predictors, FALSE by default
`inner_cv`	k in k-fold CV for training lambda for Cox Lasso, only used for useCoxLasso = TRUE

Value

fitted CoxPH or CoxLasso model

`survensemble_cv` *Cross-validates predictive performance for Ensemble 1*

Description

Cross-validates predictive performance for Ensemble 1

Usage

```
survensemble_cv(
  df,
  predict.factors,
  fixed_time = NaN,
  outer_cv = 3,
  inner_cv = 3,
  repeat_cv = 2,
  randomseed = NULL,
  return_models = FALSE,
  useCoxLasso = FALSE,
  srf_tuning = list(),
  oob = TRUE
)
```

Arguments

`df`	data frame with the data, "time" and "event" for survival outcome
`predict.factors`	list of predictor names
`fixed_time`	at which performance metrics are computed
`outer_cv`	k in k-fold CV, default 3

`inner_cv`	kk in the inner look of kk-fold CV, default 3
`repeat_cv`	if NULL, runs once (or 1), otherwise repeats CV
`randomseed`	random seed
`return_models`	TRUE/FALSE, if TRUE returns all CV objects
`useCoxLasso`	TRUE/FALSE, default is FALSE
`srf_tuning`	list of tuning parameters for random forest: 1) NULL for using a default tuning grid, or 2) a list("mtry"=c(...), "nodedepth" = c(...), "nodesize" = c(...))
`oob`	TRUE/FALSE use out-of-bag predictions while tuning instead of cross-validation, TRUE by default

Value

list of outputs

Examples

```
df <- simulate_nonlinear()
ens_cv <- survensemble_cv(df, names(df)[1:4])
summary(ens_cv)
```

`survensemble_train` *Fits an ensemble of Cox-PH and Survival Random Forest (SRF) with internal CV to tune SRF hyperparameters.*

Description

Details: the function trains Cox model, then adds its out-of-the-box predictions to Survival Random Forest as an additional predictor to mimic stacking procedure used in Machine Learning and reduce over-fitting. #' Cox model is fitted to .9 data to predict the rest .1 for each 1/10s fold; these out-of-the-bag predictions are passed on to SRF

Usage

```
survensemble_train(
  df_train,
  predict.factors,
  fixed_time = NaN,
  inner_cv = 3,
  randomseed = NULL,
  srf_tuning = list(),
  fast_version = TRUE,
  oob = TRUE,
  useCoxLasso = FALSE,
  var_importance_calc = 1
)
```

Arguments

`df_train`	data, "time" and "event" describe survival outcome
`predict.factors`	list of the column names to be used as predictors
`fixed_time`	for which the performance is maximized
`inner_cv`	number of inner cycles for model tuning
`randomseed`	random seed
`srf_tuning`	list of mtry, nodedepth and nodesize, to use default supply empty list()
`fast_version`	TRUE/FALSE, TRUE by default
`oob`	FALSE/TRUE, TRUE by default
`useCoxLasso`	FALSE/TRUE, FALSE by default
`var_importance_calc`	FALSE/TRUE, TRUE by default

Value

trained object of class survensemble

`survival_prob_km` *Calculates survival probability estimated by Kaplan-Meier survival curve Uses polynomial extrapolation in survival function space, using poly(n=3)*

Description

Calculates survival probability estimated by Kaplan-Meier survival curve Uses polynomial extrapolation in survival function space, using poly(n=3)

Usage

```
survival_prob_km(df_km_train, times, estimate_censoring = FALSE)
```

Arguments

`df_km_train`	event probabilities (!not survival)
`times`	times at which survival is estimated
`estimate_censoring`	FALSE by default, if TRUE, event and censoring is reversed (for IPCW calculations)

Value

vector of survival probabilities for time_points

survsrf_cv *Cross-validates SRF model*

Description

Cross-validates SRF model

Usage

```
survsrf_cv(
  df,
  predict.factors,
  fixed_time = NaN,
  outer_cv = 3,
  repeat_cv = 2,
  randomseed = NULL,
  return_models = FALSE,
  inner_cv = 3,
  srf_tuning = list(),
  oob = TRUE
)
```

Arguments

`df`	data frame with the data, "time" and "event" for survival outcome
`predict.factors`	list of predictor names
`fixed_time`	at which performance metrics are computed
`outer_cv`	k in k-fold CV, default 3
`repeat_cv`	if NULL, runs once, otherwise repeats CV
`randomseed`	random seed
`return_models`	TRUE/FALSE, if TRUE returns all CV objects
`inner_cv`	k in the inner loop of k-fold CV for SRF hyperparameters tuning, default is 3
`srf_tuning`	list of tuning parameters for random forest: 1) NULL for using a default tuning grid, or 2) a list("mtry"=c(...), "nodedepth" = c(...), "nodesize" = c(...))
`oob`	TRUE/FALSE use out-of-bag prediction accuracy while tuning instead of cross-validation, TRUE by default

Value

list of outputs

Examples

```
df <- simulate_nonlinear()
srf_cv <- survsrf_cv(df, names(df)[1:4])
summary(srf_cv)
```

`survsrf_predict` *Predicts event probability for a fitted SRF model*

Description

Predicts event probability for a fitted SRF model randomForestSRC::rfsrc. Essentially a wrapper of srf_survival_prob_for_time.

Usage

```
survsrf_predict(trained_model, newdata, fixed_time, oob = FALSE)
```

Arguments

`trained_model`	trained model
`newdata`	test data
`fixed_time`	time for which probabilities are computed
`oob`	TRUE/FALSE use out-of-bag predictions while tuning instead of cross-validation, default is TRUE and is faster

Value

returns vector of predictions (or matrix if fixed_time is a vector of times)

`survsrf_train` *Fits randomForestSRC, with tuning by mtry, nodedepth, and nodesize. Underlying model is by Ishwaran et al(2008) https://www.randomforestsrc.org/articles/survival.html Ishwaran H, Kogalur UB, Blackstone EH, Lauer MS. Random survival forests. The Annals of Applied Statistics. 2008;2:841–60.*

Description

Fits randomForestSRC, with tuning by mtry, nodedepth, and nodesize. Underlying model is by Ishwaran et al(2008) https://www.randomforestsrc.org/articles/survival.html Ishwaran H, Kogalur UB, Blackstone EH, Lauer MS. Random survival forests. The Annals of Applied Statistics. 2008;2:841–60.

Usage

```
survsrf_train(
  df_train,
  predict.factors,
  fixed_time = NaN,
  inner_cv = 3,
  randomseed = NULL,
  srf_tuning = list(),
  fast_version = TRUE,
  oob = TRUE,
  verbose = FALSE
)
```

Arguments

`df_train`	data, "time" and "event" should describe survival outcome
`predict.factors`	list of the column names to be used as predictors
`fixed_time`	time at which performance is maximized
`inner_cv`	k in k-fold CV for model tuning
`randomseed`	random seed
`srf_tuning`	list of mtry, nodedepth and nodesize, default is NULL
`fast_version`	TRUE/FALSE, TRUE by default
`oob`	TRUE/FALSE use out-of-bag predictions while tuning SRF instead of cross-validation, default is TRUE and is faster
`verbose`	TRUE/FALSE, FALSE by default

Value

output = list(beststats, allstats, model)

`survsrf_tune` *Internal function to tune SRF model, in nested CV loop*

Description

Internal function to tune SRF model, in nested CV loop

Usage

```
survsrf_tune(
  df_tune,
  predict.factors,
  inner_cv = 3,
  fixed_time = NaN,
  randomseed = NULL,
  mtry = c(3, 4, 5),
  nodesize = c(10, 20, 50),
  nodedepth = c(100),
  verbose = FALSE,
  oob = TRUE
)
```

Arguments

`df_tune`	data frame
`predict.factors`	predictor names
`inner_cv`	k in k-fold CV, applied if oob=FALSE
`fixed_time`	NaN
`randomseed`	random seed
`mtry`	tuning parameter
`nodesize`	at which event probabilities are computed
`nodedepth`	tuning parameter
`verbose`	FALSE
`oob`	TRUE/FALSE use out-of-bag predictions while tuning instead of cross-validation, default is TRUE and is faster

Value

output=list(modelstats, bestbrier, bestauc, bestcindex)

`surv_brierscore` *Calculates time-dependent Brier Score*

Description

Calculates time-dependent Brier Scores for a vector of times. Calculations are similar to that in: https://scikit-survival.readthedocs.io/en/stable/api/generated/sksurv.metrics.brier_score.html#sksurv.metrics.brier_score https://github.com/sebp/scikit-survival/blob/v0.19.0.post1/sksurv/metrics.py#L524-L644 The function uses IPCW (inverse probability of censoring weights), computed using the Kaplan-Meier survival function, where events are censored events from train data

Usage

```
surv_brierscore(
  y_predicted_newdata,
  df_brier_train,
  df_newdata,
  time_points,
  weighted = TRUE
)
```

Arguments

`y_predicted_newdata`	computed event probabilities
`df_brier_train`	train data
`df_newdata`	test data for which brier score is computed
`time_points`	times at which BS calculated
`weighted`	TRUE/FALSE for IPWC to use or not

Value

vector of time-dependent Brier Scores for all time_points

`surv_validate` *Computes performance statistics for a survival data given the predicted event probabilities*

Description

Computes performance statistics for a survival data given the predicted event probabilities

Usage

```
surv_validate(
  y_predict,
  predict_time,
  df_train,
  df_test,
  weighted = TRUE,
  alpha = "logit"
)
```

Arguments

`y_predict`	probabilities of event by predict_time (matrix=observations x times)
`predict_time`	times for which event probabilities are given
`df_train`	train data, data frame
`df_test`	test data, data frame
`weighted`	TRUE/FALSE, for IPWC
`alpha`	calibration alpha as mean difference or from logistic regression

Value

data.frame(T, AUCROC, Brier Score, Scaled Brier Score, C_score, Calib slope, Calib alpha)

Index