Package 'abclass'

July 22, 2025

Title Angle-Based Large-Margin Classifiers

Version 0.4.0

Description Multi-category angle-based large-margin classifiers.
See Zhang and Liu (2014) <doi:10.1093/biomet/asu017> for details.

Depends R (>= 3.5.0)

Imports Rcpp, parallel, stats

LinkingTo Rcpp, RcppArmadillo

Suggests Matrix, Rglpk, qpmadr, tinytest

Copyright Eli Lilly and Company

License GPL (>= 3)

URL https://wwenjie.org/abclass,
https://github.com/wenjie2wang/abclass

BugReports https://github.com/wenjie2wang/abclass/issues

Encoding UTF-8

RoxygenNote 7.2.1

NeedsCompilation yes

Author Wenjie Wang [aut, cre] (ORCID: <https://orcid.org/0000-0003-0363-3180>),
Eli Lilly and Company [cph]

Maintainer Wenjie Wang <wang@wwenjie.org>

Repository CRAN

Date/Publication 2022-09-18 04:36:04 UTC

Contents

`abclass-package` *Multi-Category Angle-Based Large-Margin Classifiers*

Description

This package provides implementations of the multi-category angle-based classifiers (Zhang & Liu, 2014) with the large-margin unified machines (Liu, et al., 2011) for high-dimensional data.

References

Zhang, C., & Liu, Y. (2014). Multicategory Angle-Based Large-Margin Classification. *Biometrika*, 101(3), 625–640.

Liu, Y., Zhang, H. H., & Wu, Y. (2011). Hard or soft classification? large-margin unified machines. *Journal of the American Statistical Association*, 106(493), 166–177.

`abclass` *Multi-Category Angle-Based Classification*

Description

Multi-category angle-based large-margin classifiers with regularization by the elastic-net or group-wise penalty.

Usage

```
abclass(
  x,
  y,
  intercept = TRUE,
  weight = NULL,
  loss = c("logistic", "boost", "hinge-boost", "lum"),
  control = list(),
  ...
)

abclass.control(
  lambda = NULL,
  alpha = 1,
```

```
  nlambda = 50L,
  lambda_min_ratio = NULL,
  grouped = TRUE,
  group_weight = NULL,
  group_penalty = c("lasso", "scad", "mcp"),
  dgamma = 1,
  lum_a = 1,
  lum_c = 1,
  boost_umin = -5,
  maxit = 100000L,
  epsilon = 1e-04,
  standardize = TRUE,
  varying_active_set = TRUE,
  verbose = 0L,
  ...
)
```

Arguments

x	A numeric matrix representing the design matrix. No missing valus are allowed. The coefficient estimates for constant columns will be zero. Thus, one should set the argument `intercept` to `TRUE` to include an intercept term instead of adding an all-one column to `x`.
y	An integer vector, a character vector, or a factor vector representing the response label.
intercept	A logical value indicating if an intercept should be considered in the model. The default value is `TRUE` and the intercept is excluded from regularization.
weight	A numeric vector for nonnegative observation weights. Equal observation weights are used by default.
loss	A character value specifying the loss function. The available options are `"logistic"` for the logistic deviance loss, `"boost"` for the exponential loss approximating Boosting machines, `"hinge-boost"` for hybrid of SVM and AdaBoost machine, and `"lum"` for largin-margin unified machines (LUM). See Liu, et al. (2011) for details.
control	A list of control parameters. See `abclass.control()` for details.
...	Other control parameters passed to `abclass.control()`.
lambda	A numeric vector specifying the tuning parameter *lambda*. A data-driven *lambda* sequence will be generated and used according to specified `alpha`, `nlambda` and `lambda_min_ratio` if this argument is left as `NULL` by default. The specified `lambda` will be sorted in decreasing order internally and only the unique values will be kept.
alpha	A numeric value in [0, 1] representing the mixing parameter *alpha*. The default value is `1.0`.
nlambda	A positive integer specifying the length of the internally generated *lambda* sequence. This argument will be ignored if a valid `lambda` is specified. The default value is `50`.

`lambda_min_ratio`
: A positive number specifying the ratio of the smallest lambda parameter to the largest lambda parameter. The default value is set to `1e-4` if the sample size is larger than the number of predictors, and `1e-2` otherwise.

`grouped`
: A logicial value. Experimental flag to apply group penalties.

`group_weight`
: A numerical vector with nonnegative values representing the adaptive penalty factors for the specified group penalty.

`group_penalty`
: A character vector specifying the name of the group penalty.

`dgamma`
: A positive number specifying the increment to the minimal gamma parameter for group SCAD or group MCP.

`lum_a`
: A positive number greater than one representing the parameter a in LUM, which will be used only if `loss = "lum"`. The default value is `1.0`.

`lum_c`
: A nonnegative number specifying the parameter c in LUM, which will be used only if `loss = "hinge-boost"` or `loss = "lum"`. The default value is `1.0`.

`boost_umin`
: A negative number for adjusting the boosting loss for the internal majorization procedure.

`maxit`
: A positive integer specifying the maximum number of iteration. The default value is `10^5`.

`epsilon`
: A positive number specifying the relative tolerance that determines convergence. The default value is `1e-4`.

`standardize`
: A logical value indicating if each column of the design matrix should be standardized internally to have mean zero and standard deviation equal to the sample size. The default value is `TRUE`. Notice that the coefficient estimates are always returned on the original scale.

`varying_active_set`
: A logical value indicating if the active set should be updated after each cycle of coordinate-majorization-descent algorithm. The default value is `TRUE` for usually more efficient estimation procedure.

`verbose`
: A nonnegative integer specifying if the estimation procedure is allowed to print out intermediate steps/results. The default value is `0` for silent estimation procedure.

Value

The function `abclass()` returns an object of class `abclass` representing a trained classifier; The function `abclass.control()` returns an object of class abclass.control representing a list of control parameters.

References

Zhang, C., & Liu, Y. (2014). Multicategory Angle-Based Large-Margin Classification. *Biometrika*, 101(3), 625–640.

Liu, Y., Zhang, H. H., & Wu, Y. (2011). Hard or soft classification? large-margin unified machines. *Journal of the American Statistical Association*, 106(493), 166–177.

Examples

```
library(abclass)
set.seed(123)

## toy examples for demonstration purpose
## reference: example 1 in Zhang and Liu (2014)
ntrain <- 100 # size of training set
ntest <- 100  # size of testing set
p0 <- 5       # number of actual predictors
p1 <- 5       # number of random predictors
k <- 5        # number of categories

n <- ntrain + ntest; p <- p0 + p1
train_idx <- seq_len(ntrain)
y <- sample(k, size = n, replace = TRUE)         # response
mu <- matrix(rnorm(p0 * k), nrow = k, ncol = p0) # mean vector
## normalize the mean vector so that they are distributed on the unit circle
mu <- mu / apply(mu, 1, function(a) sqrt(sum(a ^ 2)))
x0 <- t(sapply(y, function(i) rnorm(p0, mean = mu[i, ], sd = 0.25)))
x1 <- matrix(rnorm(p1 * n, sd = 0.3), nrow = n, ncol = p1)
x <- cbind(x0, x1)
train_x <- x[train_idx, ]
test_x <- x[- train_idx, ]
y <- factor(paste0("label_", y))
train_y <- y[train_idx]
test_y <- y[- train_idx]

## Regularization through ridge penalty
control1 <- abclass.control(nlambda = 5, lambda_min_ratio = 1e-3,
                            alpha = 1, grouped = FALSE)
model1 <- abclass(train_x, train_y, loss = "logistic",
                  control = control1)
pred1 <- predict(model1, test_x, s = 5)
table(test_y, pred1)
mean(test_y == pred1) # accuracy

## groupwise regularization via group lasso
model2 <- abclass(train_x, train_y, loss = "boost",
                  grouped = TRUE, nlambda = 5)
pred2 <- predict(model2, test_x, s = 5)
table(test_y, pred2)
mean(test_y == pred2) # accuracy
```

`coef.abclass` *Coefficient Estimates of A Trained Angle-Based Classifier*

Description

Extract coefficient estimates from an `abclass` object.

Usage

```
## S3 method for class 'abclass'
coef(object, selection = c("cv_1se", "cv_min", "all"), ...)
```

Arguments

`object`	An object of class `abclass`.
`selection`	An integer vector for the indices of solution path or a character value specifying how to select a particular set of coefficient estimates from the entire solution path. If the specified `abclass` object contains the cross-validation results, one may set `selection` to `"cv_min"` (or `"cv_1se"`) for the estimates giving the smallest cross-validation error (or the set of estimates resulted from the largest *lambda* within one standard error of the smallest cross-validation error). The entire solution path will be returned in an array if `selection = "all"` or no cross-validation results are available in the specified `abclass` object.
`...`	Other arguments not used now.

Value

A matrix representing the coefficient estimates or an array representing all the selected solutions.

Examples

```
## see examples of `abclass()`.
```

---

coef.supclass *Coefficient Estimates of A Trained Sup-Norm Classifier*

Description

Extract coefficient estimates from an `supclass` object.

Usage

```
## S3 method for class 'supclass'
coef(object, selection = c("cv_1se", "cv_min", "all"), ...)
```

Arguments

`object`	An object of class `supclass`.
`selection`	An integer vector for the indices of solution or a character value specifying how to select a particular set of coefficient estimates from the entire solution path. If the specified `supclass` object contains the cross-validation results, one may set `selection` to `"cv_min"` (or `"cv_1se"`) for the estimates giving the smallest cross-validation error (or the set of estimates resulted from the largest

lambda within one standard error of the smallest cross-validation error). The entire solution path will be returned in an array if `selection = "all"` or no cross-validation results are available in the specified `supclass` object.

`...` Other arguments not used now.

Value

A matrix representing the coefficient estimates or an array representing all the selected solutions.

Examples

```
## see examples of `supclass()`.
```

`cv.abclass` *Tune Angle-Based Classifiers by Cross-Validation*

Description

Tune the regularization parameter for an angle-based large-margin classifier by cross-validation.

Usage

```
cv.abclass(
  x,
  y,
  intercept = TRUE,
  weight = NULL,
  loss = c("logistic", "boost", "hinge-boost", "lum"),
  control = list(),
  nfolds = 5L,
  stratified = TRUE,
  alignment = c("fraction", "lambda"),
  refit = FALSE,
  ...
)
```

Arguments

`x` A numeric matrix representing the design matrix. No missing valus are allowed. The coefficient estimates for constant columns will be zero. Thus, one should set the argument `intercept` to `TRUE` to include an intercept term instead of adding an all-one column to `x`.

`y` An integer vector, a character vector, or a factor vector representing the response label.

`intercept` A logical value indicating if an intercept should be considered in the model. The default value is `TRUE` and the intercept is excluded from regularization.

weight	A numeric vector for nonnegative observation weights. Equal observation weights are used by default.
loss	A character value specifying the loss function. The available options are "logistic" for the logistic deviance loss, "boost" for the exponential loss approximating Boosting machines, "hinge-boost" for hybrid of SVM and AdaBoost machine, and "lum" for largin-margin unified machines (LUM). See Liu, et al. (2011) for details.
control	A list of control parameters. See abclass.control() for details.
nfolds	A positive integer specifying the number of folds for cross-validation. Five-folds cross-validation will be used by default. An error will be thrown out if the nfolds is specified to be less than 2.
stratified	A logical value indicating if the cross-validation procedure should be stratified by the response label. The default value is TRUE to ensure the same number of categories be used in validation and training.
alignment	A character vector specifying how to align the lambda sequence used in the main fit with the cross-validation fits. The available options are "fraction" for allowing cross-validation fits to have their own lambda sequences and "lambda" for using the same lambda sequence of the main fit. The option "lambda" will be applied if a meaningful lambda is specified. The default value is "fraction".
refit	A logical value or a named list specifying if and how a refit for those selected predictors should be performed. The default valie is FALSE.
...	Other control parameters passed to abclass.control().

Value

An S3 object of class cv.abclass.

`cv.supclass` *Tune Sup-Norm Classifiers by Cross-Validation*

Description

Tune the regularization parameter lambda for a sup-norm classifier by cross-validation.

Usage

```
cv.supclass(
  x,
  y,
  model = c("logistic", "psvm", "svm"),
  penalty = c("lasso", "scad"),
  start = NULL,
  control = list(),
  nfolds = 5L,
  stratified = TRUE,
  ...
)
```

Arguments

x	A numeric matrix representing the design matrix. No missing valus are allowed. The coefficient estimates for constant columns will be zero. Thus, one should set the argument `intercept` to `TRUE` to include an intercept term instead of adding an all-one column to `x`.
y	An integer vector, a character vector, or a factor vector representing the response label.
model	A charactor vector specifying the classification model. The available options are `"logistic"` for multi-nomial logistic regression model, `"psvm"` for proximal support vector machine (PSVM), `"svm"` for multi-category support vector machine.
penalty	A charactor vector specifying the penalty function for the sup-norms. The available options are `"lasso"` for sup-norm regularization proposed by Zhang et al. (2008) and `"scad"` for supSCAD regularization proposed by Li & Zhang (2021).
start	A numeric matrix representing the starting values for the quadratic approximation procedure behind the scene.
control	A list with named elements.
nfolds	A positive integer specifying the number of folds for cross-validation. Five-folds cross-validation will be used by default. An error will be thrown out if the `nfolds` is specified to be less than 2.
stratified	A logical value indicating if the cross-validation procedure should be stratified by the response label. The default value is `TRUE` to ensure the same number of categories be used in validation and training.
...	Other arguments passed to `supclass`.

Value

An S3 object of class `cv.supclass`.

`et.abclass` *Tune Angle-Based Classifiers by ET-Lasso*

Description

Tune the regularization parameter for an angle-based large-margin classifier by the ET-Lasso method (Yang, et al., 2019).

Usage

```
et.abclass(
  x,
  y,
  intercept = TRUE,
  weight = NULL,
```

```
  loss = c("logistic", "boost", "hinge-boost", "lum"),
  control = list(),
  nstages = 2,
  refit = list(lambda = 1e-06),
  ...
)
```

Arguments

x	A numeric matrix representing the design matrix. No missing valus are allowed. The coefficient estimates for constant columns will be zero. Thus, one should set the argument `intercept` to `TRUE` to include an intercept term instead of adding an all-one column to `x`.
y	An integer vector, a character vector, or a factor vector representing the response label.
intercept	A logical value indicating if an intercept should be considered in the model. The default value is `TRUE` and the intercept is excluded from regularization.
weight	A numeric vector for nonnegative observation weights. Equal observation weights are used by default.
loss	A character value specifying the loss function. The available options are `"logistic"` for the logistic deviance loss, `"boost"` for the exponential loss approximating Boosting machines, `"hinge-boost"` for hybrid of SVM and AdaBoost machine, and `"lum"` for largin-margin unified machines (LUM). See Liu, et al. (2011) for details.
control	A list of control parameters. See `abclass.control()` for details.
nstages	A positive integer specifying for the number of stages in the ET-Lasso procedure. By default, two rounds of tuning by random permutations will be performed as suggested in Yang, et al. (2019).
refit	A logical value indicating if a new classifier should be trained using the selected predictors. This argument can also be a list with named elements, which will be passed to `abclass.control()` to specify how the new classifier should be trained.
...	Other control parameters passed to `abclass.control()`.

References

Yang, S., Wen, J., Zhan, X., & Kifer, D. (2019). ET-Lasso: A new efficient tuning of lasso-type regularization for high-dimensional data. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (pp. 607–616).

predict.abclass *Prediction by A Trained Angle-Based Classifier*

Description

Predict class labels or estimate conditional probabilities for the specified new data.

Usage

```
## S3 method for class 'abclass'
predict(
  object,
  newx,
  type = c("class", "probability"),
  selection = c("cv_1se", "cv_min", "all"),
  ...
)
```

Arguments

object	An object of class abclass.
newx	A numeric matrix representing the design matrix for predictions.
type	A character value specifying the desired type of predictions. The available options are "class" for predicted labels and "probability" for class conditional probability estimates.
selection	An integer vector for the solution indices or a character value specifying how to select a particular set of coefficient estimates from the entire solution path for prediction. If the specified object contains the cross-validation results, one may set selection to "cv_min" (or "cv_1se") for using the estimates giving the smallest cross-validation error (or the set of estimates resulted from the largest *lambda* within one standard error of the smallest cross-validation error) or prediction. The prediction for the entire solution path will be returned in a list if selection = "all" or no cross-validation results are available in the specified object.
...	Other arguments not used now.

Value

A vector representing the predictions or a list containing the predictions for each set of estimates along the solution path.

Examples

```
## see examples of `abclass()`.
```

`predict.supclass` *Predictions from A Trained Sup-Norm Classifier*

Description

Predict class labels or estimate conditional probabilities for the specified new data.

Usage

```
## S3 method for class 'supclass'
predict(
  object,
  newx,
  type = c("class", "probability"),
  selection = c("cv_1se", "cv_min", "all"),
  ...
)
```

Arguments

`object`	An object of class `abclass`.
`newx`	A numeric matrix representing the design matrix for predictions.
`type`	A character value specifying the desired type of predictions. The available options are `"class"` for predicted labels and `"probability"` for class conditional probability estimates.
`selection`	An integer vector for the solution indices or a character value specifying how to select a particular set of coefficient estimates from the entire solution path for prediction. If the specified `object` contains the cross-validation results, one may set `selection` to `"cv_min"` (or `"cv_1se"`) for using the estimates giving the smallest cross-validation error (or the set of estimates resulted from the largest *lambda* within one standard error of the smallest cross-validation error) or prediction. The prediction for the entire solution path will be returned in a list if `selection = "all"` or no cross-validation results are available in the specified `object`.
`...`	Other arguments not used now.

Value

A vector representing the predictions or a list containing the predictions for each set of estimates.

Examples

```
## see examples of `supclass()`.
```

supclass *Multi-Category Classifiers with Sup-Norm Regularization*

Description

Experimental implementations of multi-category classifiers with sup-norm penalties proposed by Zhang, et al. (2008) and Li & Zhang (2021).

Usage

```
supclass(
  x,
  y,
  model = c("logistic", "psvm", "svm"),
  penalty = c("lasso", "scad"),
  start = NULL,
  control = list(),
  ...
)

supclass.control(
  lambda = 0.1,
  adaptive_weight = NULL,
  scad_a = 3.7,
  maxit = 50,
  epsilon = 1e-04,
  shrinkage = 1e-04,
  warm_start = TRUE,
  standardize = TRUE,
  verbose = 0L,
  ...
)
```

Arguments

x	A numeric matrix representing the design matrix. No missing valus are allowed. The coefficient estimates for constant columns will be zero. Thus, one should set the argument `intercept` to `TRUE` to include an intercept term instead of adding an all-one column to `x`.
y	An integer vector, a character vector, or a factor vector representing the response label.
model	A charactor vector specifying the classification model. The available options are `"logistic"` for multi-nomial logistic regression model, `"psvm"` for proximal support vector machine (PSVM), `"svm"` for multi-category support vector machine.

`penalty` A charactor vector specifying the penalty function for the sup-norms. The available options are `"lasso"` for sup-norm regularization proposed by Zhang et al. (2008) and `"scad"` for supSCAD regularization proposed by Li & Zhang (2021).

`start` A numeric matrix representing the starting values for the quadratic approximation procedure behind the scene.

`control` A list with named elements.

`...` Optional control parameters passed to the `supclass.control()`.

`lambda` A numeric vector specifying the tuning parameter *lambda*. The default value is `0.1`. Users should tune this parameter for a better model fit. The specified lambda will be sorted in decreasing order internally and only the unique values will be kept.

`adaptive_weight` A numeric vector or matrix representing the adaptive penalty weights. The default value is `NULL` for equal weights. Zhang, et al. (2008) proposed two ways to employ the adaptive weights. The first approach applies the weights to the sup-norm of coefficient estimates, while the second approach applies element-wise multiplication to the weights and coefficient estimates inside the sup-norms. The first or second approach will be applied if a numeric vector or matrix is specified, respectively. The adaptive weights are supported for lasso penalty only.

`scad_a` A positive number specifying the tuning parameter *a* in the SCAD penalty.

`maxit` A positive integer specifying the maximum number of iteration. The default value is `50` as suggested in Li & Zhang (2021).

`epsilon` A positive number specifying the relative tolerance that determines convergence. The default value is `1e-4`.

`shrinkage` A nonnegative tolerance to shrink estimates with sup-norm close enough to zero (within the specified tolerance) to zeros. The default value is `1e-4`. ## @param ridge_lambda The tuning parameter lambda of the ridge penalty used to ## set the (first set of) starting values.

`warm_start` A logical value indicating if the estimates from last lambda should be used as the starting values for the next lambda. If `FALSE`, the user-specified starting values will be used instead.

`standardize` A logical value indicating if a standardization procedure should be performed so that each column of the design matrix has mean zero and standardization

`verbose` A nonnegative integer specifying if the estimation procedure is allowed to print out intermediate steps/results. The default value is `0` for silent estimation procedure.

Details

For the multinomial logistic model or the proximal SVM model, this function utilizes the function `quadprog::solve.QP()` to solve the equivalent quadratic problem; For the multi-class SVM, this function utilizes GNU GLPK to solve the equivalent linear programming problem via the package Rglpk. It is recommended to use a recent version of GLPK.

References

Zhang, H. H., Liu, Y., Wu, Y., & Zhu, J. (2008). Variable selection for the multicategory SVM via adaptive sup-norm regularization. *Electronic Journal of Statistics*, 2, 149–167.

Li, N., & Zhang, H. H. (2021). Sparse learning with non-convex penalty in multi-classification. *Journal of Data Science*, 19(1), 56–74.

Examples

```
library(abclass)
set.seed(123)

## toy examples for demonstration purpose
## reference: example 1 in Zhang and Liu (2014)
ntrain <- 100 # size of training set
ntest <- 1000 # size of testing set
p0 <- 2       # number of actual predictors
p1 <- 2       # number of random predictors
k <- 3        # number of categories

n <- ntrain + ntest; p <- p0 + p1
train_idx <- seq_len(ntrain)
y <- sample(k, size = n, replace = TRUE)         # response
mu <- matrix(rnorm(p0 * k), nrow = k, ncol = p0) # mean vector
## normalize the mean vector so that they are distributed on the unit circle
mu <- mu / apply(mu, 1, function(a) sqrt(sum(a ^ 2)))
x0 <- t(sapply(y, function(i) rnorm(p0, mean = mu[i, ], sd = 0.25)))
x1 <- matrix(rnorm(p1 * n, sd = 0.3), nrow = n, ncol = p1)
x <- cbind(x0, x1)
train_x <- x[train_idx, ]
test_x <- x[- train_idx, ]
y <- factor(paste0("label_", y))
train_y <- y[train_idx]
test_y <- y[- train_idx]

## regularization with the supnorm lasso penalty
options("mc.cores" = 1)
model <- supclass(train_x, train_y, model = "psvm", penalty = "lasso")
pred <- predict(model, test_x)
table(test_y, pred)
mean(test_y == pred) # accuracy
```

Index