Package 'subsemble'

January 24, 2022

Type Package

Title An Ensemble Method for Combining Subset-Specific Algorithm Fits

Version 0.1.0

Date 2022-01-22

Description The Subsemble algorithm is a general subset ensemble prediction method, which can be used for small, moderate, or large datasets. Subsemble partitions the full dataset into subsets of observations, fits a specified underlying algorithm on each subset, and uses a unique form of k-fold cross-validation to output a prediction function that combines the subset-specific fits. An oracle result provides a theoretical performance guarantee for Subsemble. The paper, ``Subsemble: An ensemble method for combining subset-specific algorithm fits" is authored by Stephanie Sapp, Mark J. van der Laan & John Canny (2014) <doi:10.1080/02664763.2013.864263>.

License Apache License (== 2.0)

Depends R (>= 2.14.0), SuperLearner

Suggests arm, caret, class, cvAUC, e1071, earth, gam, gbm, glmnet, Hmisc, ipred, lattice, LogicReg, MASS, mda, mlbench, nnet, parallel, party, polspline, quadprog, randomForest, rpart, SIS, spls, stepPlr

URL https://github.com/ledell/subsemble

BugReports https://github.com/ledell/subsemble/issues

LazyLoad yes

NeedsCompilation no

Author Erin LeDell [cre], Stephanie Sapp [aut], Mark van der Laan [aut]

Maintainer Erin LeDell <oss@ledell.org>

Repository CRAN

Date/Publication 2022-01-24 20:10:02 UTC

R topics documented:

`subsemble-package` *An Ensemble Method for Combining Subset-Specific Algorithm Fits*

Description

The Subsemble algorithm is a general subset ensemble prediction method, which can be used for small, moderate, or large datasets. Subsemble partitions the full dataset into subsets of observations, fits a specified underlying algorithm on each subset, and uses a unique form of k-fold cross-validation to output a prediction function that combines the subset-specific fits. An oracle result provides a theoretical performance guarantee for Subsemble.

Details

Package:	subsemble
Type:	Package
Version:	0.1.0
Date:	2012-01-22
License:	Apache License (== 2.0)

Note

This work was supported in part by the Doris Duke Charitable Foundation Grant No. 2011042.

Author(s)

Authors: Erin LeDell, Stephanie Sapp, Mark van der Laan

Maintainer: Erin LeDell <oss@ledell.org>

References

LeDell, E. (2015) Scalable Ensemble Learning and Computationally Efficient Variance Estimation (Doctoral Dissertation). University of California, Berkeley, USA.
https://github.com/ledell/phd-thesis/blob/main/ledell-phd-thesis.pdf

Stephanie Sapp, Mark J. van der Laan & John Canny. (2014) Subsemble: An ensemble method for combining subset-specific algorithm fits. Journal of Applied Statistics, 41(6):1247-1259.

https://www.tandfonline.com/doi/abs/10.1080/02664763.2013.864263
https://biostats.bepress.com/ucbbiostat/paper313/

See Also

SuperLearner

---

predict.subsemble *Predict method for a 'subsemble' object.*

Description

Obtains predictions on a new data set from a subsemble fit. May require the original data, X, if one of the learner algorithms uses the original data in its predict method.

Usage

```
## S3 method for class 'subsemble'
predict(object, newx, x = NULL, y = NULL, ...)
```

Arguments

object	An object of class 'subsemble', which is returned from the subsemble function.
newx	The predictor variables for a new (testing) data set. The structure should match x.
x	Original data set used to fit object.
y	Original outcome used to fit object.
...	Additional arguments passed to the predict.SL.* functions.

Details

If newx is omitted, the predicted values from object are returned. The learner algorithm needs to have a corresponding prediction function with "predict" prefixed onto the algorithm name (e.g. predict.SL.glm for SL.glm). This should be taken care of by the SuperLearner package.

Value

pred	Predicted values from subsemble fit.
subpred	A data.frame with the predicted values from each sublearner algorithm for the rows in newx. If we have trained M individual models, then there will be M columns.

Author(s)

Erin LeDell <oss@ledell.org>

See Also

subsemble

Examples

```
# See subsemble() function documentation for an example.
```

---

subsemble *An Ensemble Method for Combining Subset-Specific Algorithm Fits*

Description

The Subsemble algorithm partitions the full dataset into subsets of observations, fits a specified underlying algorithm on each subset, and uses a unique form of k-fold cross-validation to output a prediction function that combines the subset-specific fits.

Usage

```
subsemble(x, y, newx = NULL, family = gaussian(),
    learner, metalearner = "SL.glm", subsets = 3, subControl = list(),
    cvControl = list(), learnControl = list(), genControl = list(),
    id = NULL, obsWeights = NULL, seed = 1, parallel = "seq")
```

Arguments

x	The data.frame or matrix of predictor variables.
y	The outcome in the training data set. Must be a numeric vector.
newx	The predictor variables in the test data set. The structure should match x. If missing, uses x for newx.
family	A description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See '?family' for the details of family functions.) Currently allows gaussian() or binomial().
learner	A string or character vector naming the prediction algorithm(s) used to train a model on each of the subsets of x. This uses the learning algorithm API provided by the SuperLearner package, so for example, we could use learner = "SL.randomForest" or learner = c("SL.glm","SL.randomForest"). See the listWrappers function for a full list of available algorithms. If a single learner is provided, the same algorithm will be used on each of the subsets. If a vector of learners is provided, then each algorithm will be applied to each of the subsets (default behavior specified by the learnControl$multiType="crossprod"); or alternatively, the multiple algorithms can be applied to different subsets with learnControl$multiType="divisor".

`metalearner` A string specifying the prediction algorithm used to learn the optimal weighted combination of the sublearners (ie. models learned on subsets of the data.) This uses the API provided by the SuperLearner package, so for example, we could use `metalearner = "SL.glmnet"`. See the `listWrappers` function for a full list of available algorithms.

`subsets` An integer specifying the number of subsets the data should be partitioned into, a vector of subset labels equal to the number of rows of `x`, or a user-specified list of index vectors equal to the number of subsets. If subsets is an integer, you can control how the subsets are partitioned (random shuffle, etc) using the `subControl` argument.

`subControl` A list of parameters to control the data partitioning (subsetting) process. The logical `stratifyCV` list parameter will stratify the data splits by binary outcome (`family=binomial()` only), and defaults to `TRUE`. The logical `shuffle` parameter defaults to `TRUE` to ensure that subsets will be created randomly. If the user explicitly specifies the subsets via the `subsets` argument, that will override any parameters in this list. The last parameter, `supervised`, currently defaults to `NULL` and is a place-holder for option to learn the optimal subsets in a supervised manner. This will be implemented in a future release.

`cvControl` A list of parameters to control the cross-validation process. The `V` parameter is an integer representing the number of cross-validation folds and defaults to 10. Each of the subsets will be divided into `V` cross-validation folds. The other parameters are `stratifyCV` and `shuffle`, which are both logical and default to `TRUE`. See above for descriptions of these parameters.

`learnControl` A list of parameters to control the learning process. Currently, the only parameter is `multiType`, which is only used if there are multiple learners specified by the `learner` argument. The two supported values for `multiType` are `"crossprod"` (the default) and `"divisor"`. The `"crossprod"` type will train each of the learners on each of the subsets. For the `"divisor"` type, the length of the `learners` vector must be a divisor of the number of subsets. If `length(learner)` equals the number of subsets, each learner will be applied to a single subset. If `length(learner)` is a divisor of the number of subsets, then the learners will be repeated as necessary (to equal the number of subsets).

`genControl` A list of general control parameters. Currently, the only parameter is `saveFits`, which defaults to `TRUE`. If set to `FALSE`, then the `subfits` and `metafit` output objects will be set to `NULL`. This can be used if you want to train and test in one step and do not want to waste disk space storing all the models.

`id` Optional cluster identification variable. Passed to the `learner` algorithm.

`obsWeights` Optional observation weights vector. As with `id` above, `obsWeights` is passed to the prediction and screening algorithms, but many of the built in learner wrappers ignore (or can't use) the information. If you are using observation weights, make sure the learner you specify uses the information, or the weights will be ignored.

`seed` A random seed to be set (integer); defaults to 1. If `NULL`, then a random seed will not be set.

`parallel` A character string specifying optional parallelization. Use `"seq"` for sequential computation (the default). Use `"multicore"` to perform the k-fold (internal)

cross-validation step as well as the learning across subsets in parallel over all available cores. Or `parallel` can be a snow cluster object. Both parallel options use the built-in functionality of the core "parallel" package.

Value

`subfits`	A list of predictive models, each of which are fit on a subset of the (rows of) data, `x`. For `learnControl$multiType="crossprod"`, the length of this list is equal to the number of subsets times the number of learners in the `learner` argument. For `learnControl$multiType="divisor"`, the length of this list is equal to the number of subsets.
`metafit`	The predictive model which is learned by regressing `y` on `Z` (see description of `Z` below). The type of model is specified using the `metalearner` argument.
`subpred`	A data.frame with the predicted values from each sublearner algorithm for the rows in `newx`. If we have L unique learners and there are J subsets of data, then there will be L x J columns when `learnControl$multiType=="crossprod"` (default) and J columns when `learnControl$multiType=="divisor"`.
`pred`	A vector containing the predicted values from the subsemble for the rows in `newX`.
`Z`	The Z matrix (the cross-validated predicted values for each sublearner).
`cvRisk`	A numeric vector with the k-fold cross-validated risk estimate for each algorithm in learning library. Note that this does not contain the CV risk estimate for the Subsemble, only the individual models in the library. (Not enabled yet, set to `NULL`.)
`family`	Returns the `family` argument from above.
`subControl`	Returns the `subControl` argument from above.
`cvControl`	Returns the `cvControl` argument from above.
`learnControl`	Returns the `learnControl` argument from above.
`subsets`	The list of subsets, which is a list of vectors of row indicies. The length of this list equals the number of subsets.
`subCVsets`	The list of subsets, further broken down into the cross-validation folds that were used. Each subset (top level list element) is partitioned into V cross-validation folds.
`ylim`	Returns range of `y`.
`seed`	An integer. Returns `seed` argument from above.
`runtime`	An list of runtimes for various steps of the algorithm. The list contains `cv`, `metalearning`, `sublearning` and `total` elements. The `cv` element is the time it takes to create the `Z` matrix (see above). The `metalearning` element is the training time for the metalearning step. The `sublearning` element is a list of training times for each of the models in the ensemble. The time to run the entire `subsemble` function is given in `total`.

Author(s)

Erin LeDell <oss@ledell.org>

References

LeDell, E. (2015) Scalable Ensemble Learning and Computationally Efficient Variance Estimation (Doctoral Dissertation). University of California, Berkeley, USA.
https://github.com/ledell/phd-thesis/blob/main/ledell-phd-thesis.pdf

Stephanie Sapp, Mark J. van der Laan & John Canny. (2014) Subsemble: An ensemble method for combining subset-specific algorithm fits. Journal of Applied Statistics, 41(6):1247-1259
https://www.tandfonline.com/doi/abs/10.1080/02664763.2013.864263
https://biostats.bepress.com/ucbbiostat/paper313/

See Also

listWrappers, SuperLearner

Examples

```
# Load some example data.

library(subsemble)
library(cvAUC)  # >= version 1.0.1
data(admissions)

# Training data.
x <- subset(admissions, select = -c(Y))[1:400,]
y <- admissions$Y[1:400]

# Test data.
newx <- subset(admissions, select = -c(Y))[401:500,]
newy <- admissions$Y[401:500]


# Set up the Subsemble.

learner <- c("SL.randomForest", "SL.glm")
metalearner <- "SL.glm"
subsets <- 2


# Train and test the model.
# With learnControl$multiType="crossprod" (the default),
# we ensemble 4 models (2 subsets x 2 learners).

fit <- subsemble(x = x, y = y, newx = newx, family = binomial(),
                 learner = learner, metalearner = metalearner,
                 subsets = subsets)


# Evaulate the model by calculating AUC on the test set.

auc <- AUC(predictions = fit$pred, labels = newy)
```

```
print(auc)  # Test set AUC is: 0.937


# We can also use the predict method to generate predictions on new data afterwards.

pred <- predict(fit, newx)
auc <- AUC(predictions = pred$pred, labels = newy)
print(auc)  # Test set AUC is: 0.937


# Modify the learnControl argument and train/eval a new Subsemble.
# With learnControl$multiType="divisor",
# we ensemble only 2 models (one for each subset).

fit <- subsemble(x = x, y = y, newx = newx, family = binomial(),
                 learner = learner, metalearner = metalearner,
                 subsets = subsets,
                 learnControl = list(multiType = "divisor"))

auc <- AUC(predictions = fit$pred, labels = newy)
print(auc)  # Test set AUC is: 0.922


# An example using a single learner.
# In this case there are 3 subsets and 1 learner,
# for a total of 3 models in the ensemble.

learner <- c("SL.randomForest")
metalearner <- "SL.glmnet"
subsets <- 3

fit <- subsemble(x = x, y = y, newx = newx, family = binomial(),
                 learner = learner, metalearner = metalearner,
                 subsets = subsets)

auc <- AUC(predictions = fit$pred, labels = newy)
print(auc)  # Test set AUC is: 0.925


# An example using the full data (i.e. subsets = 1).
# Here, we have an ensemble of 2 models (one for each of the 2 learners).
# This is equivalent to the Super Learner algorithm.

learner <- c("SL.randomForest", "SL.glm")
metalearner <- "SL.glm"
subsets <- 1

fit <- subsemble(x = x, y = y, newx = newx, family = binomial(),
                 learner = learner, metalearner = metalearner,
                 subsets = subsets)

auc <- AUC(predictions = fit$pred, labels = newy)
print(auc)  # Test set AUC is: 0.935
```

```
# Multicore subsemble via the "parallel" package.
# To perform the cross-validation and training steps using all available cores,
# use the parallel = "multicore" option.

# More examples and information at: https://github.com/ledell/subsemble
```

Index