Package 'literanger'

September 22, 2024

Title Random Forests for Multiple Imputation Based on 'ranger'

Version 0.1.1

Description An updated implementation of R package 'ranger' by Wright et al, (2017) <doi:10.18637/jss.v077.i01> for training and predicting from random forests, particularly suited to high-dimensional data, and for embedding in 'Multiple Imputation by Chained Equations' (MICE) by van Buuren (2007) <doi:10.1177/0962280206074463>. Ensembles of classification and regression trees are currently supported. Sparse data of class 'dgCMatrix' (R package 'Matrix') can be directly analyzed. Conventional bagged predictions are available alongside an efficient prediction for MICE via the algorithm proposed by Doove et al (2014) <doi:10.1016/j.csda.2013.10.025>. Survival and probability forests are not supported in the update, nor is data of class 'gwaa.data' (R package 'GenABEL'); use the original 'ranger' package for these analyses.

Depends R (>= 3.6.0)

License GPL-3

Encoding UTF-8

BugReports https://gitlab.com/stephematician/literanger/-/issues

URL https://gitlab.com/stephematician/literanger

Suggests Matrix (>= 1.5.3), testthat (>= 3.0.0), tibble (>= 3.2.1)

Imports stats

LinkingTo cpp11 (>= 0.4.7), Rcereal (>= 1.3.2)

RoxygenNote 7.3.1

NeedsCompilation yes

Author Stephen Wade [aut, cre] (<https://orcid.org/0000-0002-2573-9683>), Marvin N Wright [ctb]

Maintainer Stephen Wade <stephematician@gmail.com>

Repository CRAN

Date/Publication 2024-09-22 21:30:05 UTC

Contents

`predict.literanger` *Literanger prediction*

Description

'literanger' provides different types of prediction that may be used in multiple imputation algorithms with random forests. The usual prediction is the 'bagged' prediction, the most frequent value (or the mean) of the in-bag samples in a terminal node. Doove et al (2014) propose a prediction that better matches the predictive distribution as needed for multiple imputation; take a random draw from the observations in the terminal node from a randomly drawn tree in the forest for each predicted value needed. Alternatively, the usual most-frequent-value or mean of the in-bag responses can be used as in missForest (Stekhoven et al, 2014) or miceRanger https://cran.r-project.org/package=miceRanger and missRanger https://cran.r-project.org/package=missRanger.

Usage

```
## S3 method for class 'literanger'
predict(
  object,
  newdata = NULL,
  prediction_type = c("bagged", "inbag", "nodes"),
  seed = 1L + sample.int(n = .Machine$integer.max - 1L, size = 1),
  n_thread = 0,
  verbose = FALSE,
  ...
)
```

Arguments

`object`	A trained random forest `literanger` object.
`newdata`	Data of class `data.frame`, `matrix`, or `dgCMatrix` (Matrix), for the latter two; must have column names; all predictors named in `object$predictor_names` must be present.
`prediction_type`	Name of the prediction algorithm; "bagged" is the most-frequent value among in-bag samples for classification, or the mean of in-bag responses for regression; "inbag" predicts by drawing one in-bag response from a random tree for each row; "nodes" (currently unsupported) returns the node keys (ids) of the terminal node from every tree for each row.

seed	Random seed, an integer between 1 and .Machine$integer.max. Default generates the seed from R, set to 0 to ignore the R seed and use a C++ std::random_device.
n_thread	Number of threads. Default is determined by system, typically the number of cores.
verbose	Show computation status and estimated runtime.
...	Ignored.

Details

Forests trained by literanger retain information about the in-bag responses in each terminal node, thus facilitating efficient predictions within a variation on multiple imputation proposed by Doove et al (2014). This type of prediction can be selected by setting prediction_type="inbag", or the usual prediction for classification and regression forests, the most-frequent-value and mean of in bag samples respectively, is given by setting prediction_type="bagged".

A list is returned. The values item contains the predicted classes or values (classification and regression forests, respectively). Factor levels are returned as factors with the levels as per the original training data.

Compared to the original package ranger, literanger excludes certain features:

- Probability, survival, and quantile regression forests.
- Support for class gwaa.data.
- Standard error estimation.

Value

Object of class literanger_prediction with elements:

values Predicted (drawn) classes/value for classification and regression.

tree_type Number of trees.

seed The seed supplied to the C++ library.

Author(s)

stephematician stephematician@gmail.com, Marvin N Wright (original ranger package)

References

- Doove, L. L., Van Buuren, S., & Dusseldorp, E. (2014). Recursive partitioning for missing data imputation in the presence of interaction effects. *Computational Statistics & Data Analysis*, 72, 92-104. doi:10.1016/j.csda.2013.10.025.
- Stekhoven, D.J. and Buehlmann, P. (2012). MissForest–non-parametric missing value imputation for mixed-type data. *Bioinformatics*, 28(1), 112-118. doi:10.1093/bioinformatics/btr597.
- Wright, M. N., & Ziegler, A. (2017a). ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R. *Journal of Statistical Software*, 77, 1-17. doi:10.18637/jss.v077.i01.

See Also

train

Examples

```
## Classification forest
train_idx <- sample(nrow(iris), 2/3 * nrow(iris))
iris_train <- iris[ train_idx, ]
iris_test  <- iris[-train_idx, ]
rf_iris <- train(data=iris_train, response_name="Species")
pred_iris_bagged <- predict(rf_iris, newdata=iris_test,
                            prediction_type="bagged")
pred_iris_inbag  <- predict(rf_iris, newdata=iris_test,
                            prediction_type="inbag")
# compare bagged vs actual test values
table(iris_test$Species, pred_iris_bagged$values)
# compare bagged prediction vs in-bag draw
table(pred_iris_bagged$values, pred_iris_inbag$values)
```

---

read_literanger *De-serialize random forest*

Description

Read the random forest from a file or connection using light-weight serialization for C++ objects.

Usage

```
read_literanger(file, verbose = TRUE, ...)
```

Arguments

file	A connection or the name of a file containing a serialized literanger object.
verbose	Show additional serialization information (not implemented).
...	Further arguments passed to readRDS().

Details

This function uses 'cereal' light-weight serialization to read a literanger object (random forest) from a file or connection. The file is usually the result of a call to write_literanger(). The random forest returned can be used for prediction immediately upon return, and does not require the original training data or training environment.

Value

A literanger random forest object

Author(s)

stephematician <stephematician@gmail.com

See Also

write_literanger() readRDS

---

train *Train forest using ranger for multiple imputation algorithms.*

Description

'literanger' trains random forests for use in multiple imputation problems via an adaptation of the 'ranger' R package. ranger is a fast implementation of random forests (Breiman, 2001) or recursive partitioning, particularly suited for high dimensional data (Wright et al, 2017a). literanger supports prediction used in algorithms such as "Multiple Imputation via Chained Equations" (Van Buuren, 2007).

Usage

```
train(
  data = NULL,
  response_name = character(),
  predictor_names = character(),
  x = NULL,
  y = NULL,
  case_weights = numeric(),
  classification = NULL,
  n_tree = 10,
  replace = TRUE,
  sample_fraction = ifelse(replace, 1, 0.632),
  n_try = NULL,
  draw_predictor_weights = numeric(),
  names_of_always_draw = character(),
  split_rule = NULL,
  max_depth = 0,
  min_split_n_sample = 0,
  min_leaf_n_sample = 0,
  unordered_predictors = NULL,
  response_weights = numeric(),
  n_random_split = 1,
  alpha = 0.5,
  min_prop = 0.1,
  seed = 1L + sample.int(n = .Machine$integer.max - 1L, size = 1),
  save_memory = FALSE,
  n_thread = 0,
  verbose = FALSE
)
```

Arguments

data	Training data of class data.frame, matrix, or dgCMatrix (Matrix), for the latter two; must have column names.
response_name	Name of response (dependent) variable if data was provided.
predictor_names	Names of predictor (independent) variables if data was provided; default is all variables that are not the response.
x	Predictor data (independent variables), alternative interface to data and response_name.
y	Response vector (dependent variable), alternative interface to data and response_name.
case_weights	Weights for sampling of training observations. Observations with larger weights will be selected with higher probability in the bootstrap (or sub-sampled) samples for each tree.
classification	Set to TRUE to grow a classification forest if the response is numeric (including if data is a matrix), else, a regression forest is grown.
n_tree	Number of trees (default 10).
replace	Sample with replacement to train each tree.
sample_fraction	Fraction of observations to sample to train each tree. Default is 1 for sampling with replacement and 0.632 for sampling without replacement. For classification, this can be a vector of class-specific values.
n_try	Number of variables (predictors) to draw that are candidates for splitting each node by. Default is the (rounded down) square root of the number of predictors. Alternatively, a single argument function returning an integer, given the number of predictors.
draw_predictor_weights	For predictor-drawing weights shared by all trees; a numeric vector of *non-negative* weights for each predictor. For tree-specific predictor-drawing weights; a list of size n_tree containing (non-negative) vectors with length equal to the number of predictors.
names_of_always_draw	Character vector with predictor (variable) names to be selected *in addition* to the n_try predictors drawn as candidates to split by.
split_rule	Splitting rule. For classification estimation "gini", "extratrees" or "hellinger" with default "gini". For regression "variance", "extratrees", "maxstat" or "beta" with default "variance".
max_depth	Maximal tree depth. A value of NULL or 0 (the default) corresponds to unlimited depth, 1 to tree stumps (1 split per tree).
min_split_n_sample	Minimal number of in-bag samples a node must have in order to be split. Default 1 for classification and 5 for regression.
min_leaf_n_sample	Minimum number of in-bag samples in a leaf node.

unordered_predictors
: Handling of unordered factor predictors. One of "ignore", "order" and "partition". For the "extratrees" splitting rule the default is "partition" for all other splitting rules "ignore".

response_weights
: Classification only: Weights for the response classes (in order of the factor levels) in the splitting rule e.g. cost-sensitive learning. Weights are also used by each tree to determine majority vote.

n_random_split
: "extratrees" split metric only: Number of random splits to consider for each candidate splitting variable, default is 1.

alpha
: "maxstat" splitting rule only: Significance threshold to allow splitting, default is 0.5, must be in the interval `(0,1]`.

min_prop
: "maxstat" splitting rule only: Lower quantile of covariate distribution to be considered for splitting, default is 0.1, must be in the interval `[0,0.5]`.

seed
: Random seed, an integer between 1 and `.Machine$integer.max`. Default generates the seed from `R`, set to `0` to ignore the `R` seed and use a C++ `std::random_device`.

save_memory
: Use memory saving (but slower) splitting mode. Warning: This option slows down the tree growing, use only if you encounter memory problems.

n_thread
: Number of threads. Default is determined by system, typically the number of cores.

verbose
: Show computation status and estimated runtime.

Details

literanger trains classification and regression forests using the original Random Forest (Breiman, 2001) or extremely randomized trees (Geurts et al, 2006) algorithms. The trained forest retains information about the in-bag responses in each terminal node, thus facilitating a variation on the algorithm for multiple imputation with random forests proposed by Doove et al (2014). This algorithm should match the predictive distribution more closely than using predictive mean matching.

The default split metric for classification trees is the Gini impurity, which can be extended to use the extra-randomized trees rule (Geurts et al, 2006). For binary responses, the Hellinger distance metric may be used instead (Cieslak et al, 2012).

The default split metric for regression trees is the estimated variance, which can be extended to include the extra-randomized trees rule, too. Alternatively, the beta log-likelihood (Wright et al, 2017b) or maximally selected rank statistics (Wright et al, 2019) are available.

When the `data` and `response_name` arguments are supplied the response variable is identified by its corresponding column name. The type of response may be used to determine the type of tree. If the response is a factor then classification trees are used. If the response is numeric then regression trees are used. The `classification` argument can be used to override the default tree type when the response is numeric. Alternatively, use `x` and `y` arguments to specify response and predictor; this can avoid conversions and save memory. If memory usage issues persist, consider setting `save_memory=TRUE` but be aware that this option slows down the tree growing.

The `min_split_n_sample` rule can be used to control the minimum number of in-bag samples required to split a node; thus, as in the original algorithm, nodes with fewer samples than `min_split_n_sample` are possible. To put a floor under the number of samples per node, the `min_leaf_n_sample` argument is used.

When drawing candidate predictors for splitting a node on, the predictors identified by `names_of_always_draw` are included *in addition* to the `n_try` predictors that are randomly drawn. Another way to modify the way predictors are selected is via the `draw_predictor_weights` argument, which are normalised and interpreted as probabilities when drawing candidates. The weights are assigned *in the order they appear in the data*. Weights assigned by `draw_predictor_weights` to variables in `names_of_always_draw` are ignored. The usage of `draw_predictor_weights` can increase the computation times for large forests.

Unordered-factor predictors can be handled in 3 different ways by using `unordered_predictors`:

- For "ignore" all factors are regarded ordered;
- For "partition" all possible 2-partitions are considered for splitting.
- For "order" and 2-class classification the factor levels are ordered by their proportion falling in the second class, for regression by their mean response, as described in Hastie et al. (2009), chapter 9.2.4. For multi-class classification the factor levels are ordered by the first principal component of the weighted covariance matrix of the contingency table (Coppersmith et al, 1999).

The use of "order" is recommended, as it computationally fast and can handle an unlimited number of factor levels. Note that the factors are only reordered once and not again in each split.

Compared to the original package ranger, literanger excludes certain features:

- Formula interface.
- Probability, survival, and quantile regression forests.
- Support for class gwaa.data.
- Measures of variable importance.
- Regularisation of importance.
- Access to in-bag data via R.
- Support for user-specified hold-out data.

Value

Object of class `literanger` with elements:

`predictor_names` The names of the predictor variables in the model.

`names_of_unordered` The names of predictors that are unordered.

`tree_type` The type of tree in the forest.

`n_tree` The number of trees that were trained.

`n_try` The number of predictors drawn as candidates for each split.

`split_rule` The name of the split metric used.

`max_depth` The maximum allowed depth of a tree in the forest.

`min_metric_decrease` The minimum decrease in the metric for an acceptable split (equal to negative @p alpha for maximally selected rank statistics, else zero).

`min_split_n_sample` The minimum number of in-bag samples in a node prior to splitting.

`min_leaf_n_sample` The minimum number of in-bag samples in a leaf node.

`seed` The seed supplied to the C++ library.

`oob_error` The misclassification rate or the mean square error using out-of-bag samples.

`cpp11_ptr` An external pointer to the trained forest. DO NOT MODIFY.

`response_values` Classification only: the values of the response in the order they appear in the data.

`response_levels` Classification only: the labels for the response in the order they appear in the data.

Author(s)

stephematician stephematician@gmail.com, Marvin N Wright (original ranger package)

References

- Breiman, L. (2001). Random forests. *Machine Learning*, 45, 5-32. doi:10.1023/A:1010933404324.
- Cieslak, D. A., Hoens, T. R., Chawla, N. V., & Kegelmeyer, W. P. (2012). Hellinger distance decision trees are robust and skew-insensitive. *Data Mining and Knowledge Discovery*, 24, 136-158. doi:10.1007/s1061801102221.
- Coppersmith, D., Hong, S. J., & Hosking, J. R. (1999). Partitioning nominal attributes in decision trees. *Data Mining and Knowledge Discovery*, 3, 197-217. doi:10.1023/A:1009869804967.
- Doove, L. L., Van Buuren, S., & Dusseldorp, E. (2014). Recursive partitioning for missing data imputation in the presence of interaction effects. *Computational Statistics & Data Analysis*, 72, 92-104. doi:10.1016/j.csda.2013.10.025.
- Geurts, P., Ernst, D., & Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, 63, 3-42. doi:10.1007/s1099400662261.
- Hastie, T., Tibshirani, R., Friedman, J. H., & Friedman, J. H. (2009). The elements of statistical learning: data mining, inference, and prediction (Vol. 2). New York: Springer. doi:10.1007/9780387216065.
- Van Buuren, S. (2007). Multiple imputation of discrete and continuous data by fully conditional specification. *Statistical Methods in Medical Research*, 16(3), 219-242. doi:10.1177/0962280206074463.
- Weinhold, L., Schmid, M., Wright, M. N., & Berger, M. (2019). A random forest approach for modeling bounded outcomes. *arXiv preprint*, arXiv:1901.06211. doi:10.48550/arXiv.1901.06211.
- Wright, M. N., & Ziegler, A. (2017a). ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R. *Journal of Statistical Software*, 77, 1-17. doi:10.18637/jss.v077.i01.
- Wright, M. N., Dankowski, T., & Ziegler, A. (2017b). Unbiased split variable selection for random survival forests using maximally selected rank statistics. *Statistics in medicine*, 36(8), 1272-1284. doi:10.1002/sim.7212.

See Also

`predict.literanger`

Examples

```
## Classification forest with default settings
train(data=iris, response_name="Species")

## Prediction
train_idx <- sample(nrow(iris), 2/3 * nrow(iris))
iris_train <- iris[train_idx, ]
iris_test <- iris[-train_idx, ]
rg_iris <- train(data=iris_train, response_name="Species")
pred_iris <- predict(rg_iris, newdata=iris_test)
table(iris_test$Species, pred_iris$values)
```

`write_literanger` *Serialize random forest*

Description

Write a random forest to a file or connection using light-weight serialization for C++ objects.

Usage

```
write_literanger(object, file, verbose = TRUE, ...)
```

Arguments

`object`	A trained random forest `literanger` object.
`file`	A connection or the name of the file where the `literanger` object will be saved.
`verbose`	Show additional serialization information (not implemented).
`...`	Further arguments passed to `saveRDS()`.

Details

This function uses 'cereal' light-weight serialization to write a literanger object (random forest) to a file or connection. The file can be read in via `read_literanger()` and used for prediction with no requirement for the original training data.

Author(s)

stephematician stephematician@gmail.com

See Also

`read_literanger()` saveRDS

Examples

```
## Classification forest
train_idx <- sample(nrow(iris), 2/3 * nrow(iris))
iris_train <- iris[ train_idx, ]
iris_test  <- iris[-train_idx, ]
rf_iris <- train(data=iris_train, response_name="Species")
file <- tempfile()
write_literanger(rf_iris, file)
rf_copy <- read_literanger(file)
pred_bagged <- predict(rf_copy, newdata=iris_test, prediction_type="bagged")
```

Index