Package ‘SPlit’

February 14, 2022

Type Package

Title Split a Dataset for Training and Testing

Version 1.1

Date 2022-02-12

Description Procedure to optimally split a dataset for training and testing.

'SPlit' is based on the method of support points, which is independent of modeling methods.

This work is supported by U.S. National Science Foundation grant DMREF-1921873.

License GPL (>= 2)

Imports Rcpp (>= 1.0.4)

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.1.2

Encoding UTF-8

NeedsCompilation yes

Author Akhil Vakayil [aut, cre],
Roshan Joseph [aut, ths],
Simon Mak [aut]

Maintainer Akhil Vakayil <akhilv@gatech.edu>

Repository CRAN

Date/Publication 2022-02-14 08:00:13 UTC

R topics documented:

SPlit-package ................................................................. 2
SPlit ................................................................. 2
subsample ................................................................. 4

Index 5
Description

Split a dataset for training and testing

Details

The package SPlit provides the function SPlit() to optimally split a dataset for training and testing using the method of support points (Mak and Joseph, 2018). Support points is a model-independent method for finding optimal representative points of a distribution. SPlit() attempts to obtain a split in which the distribution of both the training and testing sets resemble the distribution of the dataset. The benefits of SPlit over existing data splitting procedures are detailed in Joseph and Vakayil (2021).

Author(s)

Akhil Vakayil, V. Roshan Joseph, Simon Mak
Maintainer: Akhil Vakayil <akhilv@gatech.edu>

References


Usage

SPlit(
  data,
  splitRatio = 0.2,
  kappa = NULL,
  maxIterations = 500,
  tolerance = 1e-10,
  nThreads
)

**Arguments**

- **data**: The dataset including both the predictors and response(s); should not contain missing values, and only numeric and/or factor column(s) are allowed.

- **splitRatio**: The ratio in which the dataset is to be split; should be in (0, 1) e.g. for an 80-20 split, the splitRatio is either 0.8 or 0.2.

- **kappa**: If provided, stochastic majorization-minimization is used for computing support points using a random sample from the dataset of size = ceiling(kappa * splitRatio * nrow(data)), in every iteration.

- **maxIterations**: The maximum number of iterations before the tolerance level is reached during support points optimization.

- **tolerance**: The tolerance level for support points optimization; measured in terms of the maximum point-wise difference in distance between successive solutions.

- **nThreads**: Number of threads to be used for parallel computation; if not supplied, nThreads defaults to maximum available threads.

**Details**

Support points are defined only for continuous variables. The categorical variables are handled as follows. **SPlit()** will automatically convert a nominal categorical variable with $m$ levels to $m-1$ continuous variables using Helmert coding. Ordinal categorical variables should be converted to numerical columns using a scoring method before using **SPlit()**. For example, if the three levels of an ordinal variable are poor, good, and excellent, then the user may choose 1, 2, and 5 to represent the three levels. These values depend on the problem and data collection method, and therefore, **SPlit()** will not do it automatically. The columns of the resulting numeric dataset are then standardized to have mean zero and variance one. **SPlit()** then computes the support points and calls the provided **subsample()** function to perform a nearest neighbor subsampling. The indices of this subsample are returned.

SPlit can be time consuming for large datasets. The computational time can be reduced by using the stochastic majorization-minimization technique with a trade-off in the quality of the split. For example, setting kappa = 2 will use a random sample, twice the size of the smaller subset in the split, instead of using the whole dataset in every iteration of the support points optimization. Another option for large datasets is to use data twinning (Vakayil and Joseph, 2022) implemented in the R package **twinning**. Twinning is extremely fast, but for small datasets, the results may not be as good as SPlit.

**Value**

Indices of the smaller subset in the split.

**References**


Examples

## 1. An 80-20 split of a numeric dataset
```
X = rnorm(n = 100, mean = 0, sd = 1)
Y = rnorm(n = 100, mean = X^2, sd = 1)
data = cbind(X, Y)
SPlitIndices = SPlit(data, tolerance = 1e-6, nThreads = 2)
dataTest = data[SPlitIndices, ]
dataTrain = data[-SPlitIndices, ]
plot(data, main = "SPlit testing set")
points(dataTest, col = "green", cex = 2)
```

## 2. An 80-20 split of the iris dataset
```
SPlitIndices = SPlit(iris, nThreads = 2)
irisTest = iris[SPlitIndices, ]
irisTrain = iris[-SPlitIndices, ]
```

---

subsample

**Nearest neighbor subsampling**

**Description**

subsample() finds the nearest data points in a dataset to a given set of points as described in Joseph and Vakayil (2021). It uses an efficient kd-tree based algorithm that allows for lazy deletion of a data point from the kd-tree, thereby avoiding the need to rebuild the tree after each query. Please see Blanco and Rai (2014) for details.

**Usage**

subsample(data, points)

**Arguments**

- `data`  
  The dataset; should be numeric.
- `points`  
  The set of query points of the same dimension as the dataset.

**Value**

Indices of the nearest neighbors in the dataset.

**References**


Index

* package
  SPlit-package, 2

SPlit, 2
SPlit-package, 2
subsample, 4