Package 'bagged.outliertrees'

May 7, 2026

Maintainer Rafael Santos <rafael.jpsantos@outlook.com>

Title Robust Explainable Outlier Detection Based on OutlierTree

Version 1.0.0

Description

Bagged OutlierTrees is an explainable unsupervised outlier detection method based on an ensemble implementation of the existing OutlierTree procedure (Cortes, 2020). This implementation takes advantage of bootstrap aggregating (bagging) to improve robustness by reducing the possible masking effect and subsequent high variance (similarly to Isolation Forest), hence the name ``Bagged OutlierTrees". To learn more about the base procedure OutlierTree (Cortes, 2020), please refer to <doi:10.48550/arXiv.2001.00636>.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Imports outliertree, dplyr, doSNOW, parallel, foreach, rlist, data.table

Depends R (>= 3.5.0)

URL https://github.com/RafaJPSantos/bagged.outliertrees

BugReports https://github.com/RafaJPSantos/bagged.outliertrees/issues

NeedsCompilation no

Author Rafael Santos [aut, cre]

Repository CRAN

Date/Publication 2021-07-06 09:00:02 UTC

Contents

`bagged.outliertrees` *Bagged OutlierTrees*

Description

Fit Bagged OutlierTrees ensemble model to normal data with perhaps some outliers.

Usage

```
bagged.outliertrees(
  df,
  ntrees = 100L,
  subsampling_rate = 0.25,
  max_depth = 4L,
  min_gain = 0.01,
  z_norm = 2.67,
  z_outlier = 8,
  pct_outliers = 0.01,
  min_size_numeric = 25L,
  min_size_categ = 50L,
  categ_split = "binarize",
  categ_outliers = "tail",
  numeric_split = "raw",
  cols_ignore = NULL,
  follow_all = FALSE,
  gain_as_pct = TRUE,
  nthreads = parallel::detectCores()
)
```

Arguments

`df`	Data Frame with normal data that might contain some outliers. See details for allowed column types.
`ntrees`	Controls the ensemble size (i.e. the number of OutlierTrees or bootstrapped training sets). A large value is always recommended to build a robust and stable ensemble. Should be decreased if training is taking too much time.
`subsampling_rate`	Sub-sampling rate used for bootstrapping. A small rate results in smaller bootstrapped training sets, which should not suffer from the masking effect. This parameter should be adjusted given the size of the training data (perhaps a smaller value for large training data and conversely).
`max_depth`	Maximum depth of the trees to grow. Can also pass zero, in which case it will only look for outliers with no conditions (i.e. takes each column as a 1-d distribution and looks for outliers in there independently of the values in other columns).

`min_gain` Minimum gain that a split has to produce in order to consider it (both in terms of looking for outliers in each branch, and in considering whether to continue branching from them). Note that default value for GritBot is 1e-6, with `gain_as_pct = FALSE`, but it's recommended to pass higher values (e.g. 1e-1) when using `gain_as_pct = FALSE`.

`z_norm` Maximum Z-value (from standard normal distribution) that can be considered as a normal observation. Note that simply having values above this will not automatically flag observations as outliers, nor does it assume that columns follow normal distributions. Also used for categorical and ordinal columns for building approximate confidence intervals of proportions.

`z_outlier` Minimum Z-value that can be considered as an outlier. There must be a large gap in the Z-value of the next observation in sorted order to consider it as outlier, given by (z_outlier - z_norm). Decreasing this parameter is likely to result in more observations being flagged as outliers. Ignored for categorical and ordinal columns.

`pct_outliers` Approximate max percentage of outliers to expect in a given branch.

`min_size_numeric`
Minimum size that branches need to have when splitting a numeric column. In order to look for outliers in a given branch for a numeric column, it must have a minimum of twice this number of observations.

`min_size_categ` Minimum size that branches need to have when splitting a categorical or ordinal column. In order to look for outliers in a given branch for a categorical, ordinal, or boolean column, it must have a minimum of twice this number of observations.

`categ_split` How to produce categorical-by-categorical splits. Options are:

- `"binarize"` : Will binarize the target variable according to whether it's equal to each present category within it (greater/less for ordinal), and split each binarized variable separately.
- `"bruteforce"` : Will evaluate each possible binary split of the categories (that is, it evaluates 2^n potential splits every time). Note that trying this when there are many categories in a column will result in exponential computation time that might never finish.
- `"separate"` : Will create one branch per category of the splitting variable (this is how GritBot handles them).

`categ_outliers` How to look for outliers in categorical variables. Options are:

- `"tail"` : Will try to flag outliers if there is a large gap between proportions in sorted order, and this gap is unexpected given the prior probabilities. Such criteria tends to sometimes flag too many uninteresting outliers, but is able to detect more cases and recognize outliers when there is no single dominant category.
- `"majority"` : Will calculate an equivalent to z-value according to the number of observations that do not belong to the non-majority class, according to formula '(n-n_maj)/(n * p_prior) < 1/z_outlier^2'. Such criteria tends to miss many interesting outliers and will only be able to flag outliers in large sample sizes. This is the approach used by GritBot.

`numeric_split` How to determine the split point in numeric variables. Options are:

- `"mid"` : Will calculate the midpoint between the largest observation that goes to the '<=' branch and the smallest observation that goes to the '>' branch.
- `"raw"` : Will set the split point as the value of the largest observation that goes to the '<=' branch.

This doesn't affect how outliers are determined in the training data passed in `df`, but it does affect the way in which they are presented and the way in which new outliers are detected when using `predict`. `"mid"` is recommended for continuous-valued variables, while `"raw"` will provide more readable explanations for counts data at the expense of perhaps slightly worse generalizability to unseen data.

`cols_ignore` Vector containing columns which will not be split, but will be evaluated for usage in splitting other columns. Can pass either a logical (boolean) vector with the same number of columns as `df`, or a character vector of column names (must match with those of `df`). Pass `NULL` to use all columns.

`follow_all` Whether to continue branching from each split that meets the size and gain criteria. This will produce exponentially many more branches, and if depth is large, might take forever to finish. Will also produce a lot more spurious outiers. Not recommended.

`gain_as_pct` Whether the minimum gain above should be taken in absolute terms, or as a percentage of the standard deviation (for numerical columns) or shannon entropy (for categorical columns). Taking it in absolute terms will prefer making more splits on columns that have a large variance, while taking it as a percentage might be more restrictive on them and might create deeper trees in some columns. For GritBot this parameter would always be `FALSE`. Recommended to pass higher values for `min_gain` when passing `FALSE` here. Not that when `gain_as_pct = FALSE`, the results will be sensitive to the scales of variables.

`nthreads` Number of parallel threads to use when fitting the model.

Value

An object with the fitted model that can be used to detect more outliers in new data.

References

- GritBot software: https://www.rulequest.com/gritbot-info.html
- Cortes, David. "Explainable outlier detection through decision tree conditioning." arXiv preprint arXiv:2001.00636 (2020).

See Also

predict.bagged.outliertrees print.bagged.outlieroutputs hypothyroid

Examples

```
library(bagged.outliertrees)

### example dataset with interesting outliers
data(hypothyroid)

### fit a Bagged OutlierTrees model
model <- bagged.outliertrees(hypothyroid,
  ntrees = 10,
  subsampling_rate = 0.5,
  z_outlier = 6,
  nthreads = 1
)

### use the fitted model to find outliers in the training dataset
outliers <- predict(model,
  newdata = hypothyroid,
  min_outlier_score = 0.5,
  nthreads = 1
)

### print the top-10 outliers in human-readable format
print(outliers, outliers_print = 10)
```

---

hypothyroid *Hypothyroid*

Description

Hypothyroid

Usage

```
hypothyroid
```

Format

An object of class `data.frame` with 2772 rows and 23 columns.

---

predict.bagged.outliertrees *Predict method for Bagged OutlierTrees*

Description

Predict method for Bagged OutlierTrees

Usage

```
## S3 method for class 'bagged.outliertrees'
predict(
  object,
  newdata,
  min_outlier_score = 0.95,
  nthreads = parallel::detectCores(),
  ...
)
```

Arguments

`object`	A Bagged OutlierTrees object as returned by `bagged.outliertrees`.
`newdata`	A Data Frame in which to look for outliers according to the fitted model.
`min_outlier_score`	Minimum outlier score to use when finding outliers.
`nthreads`	Number of threads to use when predicting.
`...`	No use.

Value

Will return a list of lists with the outliers and their information (each row is an entry in the first list, with the same names as the rows in the input data frame), which can be printed into a human-readable format after-the-fact through functions `print`.

See Also

bagged.outliertrees print.bagged.outlieroutputs

Examples

```
library(bagged.outliertrees)

### example dataset with interesting outliers
data(hypothyroid)

### fit a Bagged OutlierTrees model
model <- bagged.outliertrees(hypothyroid,
  ntrees = 10,
  subsampling_rate = 0.5,
  z_outlier = 6,
  nthreads = 1
)

### use the fitted model to find outliers in the training dataset
outliers <- predict(model,
  newdata = hypothyroid,
  min_outlier_score = 0.5,
  nthreads = 1
```

```
)

### print the top-10 outliers in human-readable format
print(outliers, outliers_print = 10)
```

`print.bagged.outlieroutputs`
Print outliers in human-readable format

Description

Pretty-prints outliers as output by the `predict` function from a Bagged OutlierTrees model (as generated by function `bagged.outliertrees`).

Usage

```
## S3 method for class 'bagged.outlieroutputs'
print(x, outliers_print = 15, ...)
```

Arguments

`x`	Outliers as returned by predict method on an object from `bagged.outliertrees`.
`outliers_print`	Maximum number of outliers to print.
`...`	No use.

Value

The same input `x` that was passed (as `invisible`).

See Also

bagged.outliertrees predict.bagged.outliertrees

Examples

```
library(bagged.outliertrees)

### example dataset with interesting outliers
data(hypothyroid)

### fit a Bagged OutlierTrees model
model <- bagged.outliertrees(hypothyroid,
  ntrees = 10,
  subsampling_rate = 0.5,
  z_outlier = 6,
  nthreads = 1
)

### use the fitted model to find outliers in the training dataset
```

```
outliers <- predict(model,
  newdata = hypothyroid,
  min_outlier_score = 0.5,
  nthreads = 1
)

### print the top-10 outliers in human-readable format
print(outliers, outliers_print = 10)
```

Index