Package 'outliertree'

January 12, 2020

Type Package

Title Explainable Outlier Detection Through Decision Tree Conditioning

Version 1.1.3

Date 2020-01-10

Author David Cortes

Maintainer David Cortes <david.cortes.rivera@gmail.com>

URL https://github.com/david-cortes/outliertree

BugReports https://github.com/david-cortes/outliertree/issues

Description Will try to fit decision trees that try to ``predict" values for each column based on the values of each other column.
Along the way, each time a split is evaluated, it will take the observations that fall into each branch as a homogeneous
cluster in which it will search for outliers in the 1-d distribution of the column being predicted. Outliers are determined
according to confidence intervals in this 1-d distribution, and need to have a large gap with respect to the next observation
in sorted order to be flagged as outliers. Since outliers are searched for in a decision tree branch, it will know the conditions
that make it a rare observation compared to others that meet the same conditions, and the conditions will always be correlated
with the target variable (as it's being predicted from them). Full procedure is described in Cortes (2020) <arXiv:2001.00636>.
Loosely based on the 'GritBot' <https://www.rulequest.com/gritbot-info.html> software.

License GPL (>= 3)

Imports Rcpp (>= 1.0.1)

Depends R (>= 3.5.0)

LinkingTo Rcpp, Rcereal

LazyData true

RoxygenNote 6.1.1

NeedsCompilation yes

Repository CRAN

Date/Publication 2020-01-12 14:50:02 UTC

R topics documented:

`check.outlierness.bounds` *Check values that could potentially flag an observation as outlier*

Description

Returns, for each numeric/date/timestamp column, a range of values *outside* of which observations could potentially be flagged as being an outlier in some cluster, and for categorical/ordinal/boolean columns, the factor levels that can be flagged as being an outlier in some cluster. If the lower bound is higher than the upper bound, it means any value can potentially be flagged as outlier.

Usage

```
check.outlierness.bounds(outlier_tree_model)
```

Arguments

`outlier_tree_model`
: An Outlier Tree model object as generated by ‘outlier.tree‘.

Value

A list with column as the names and the bounds or categories as values.

extract.training.outliers

Extract outliers found in training data

Description

Extracts outliers from a model generated by ‘outlier.tree‘ if it was passed parameter ‘save_outliers‘ = ‘TRUE‘.

Usage

```
extract.training.outliers(outlier_tree_model)
```

Arguments

`outlier_tree_model`
: An Outlier Tree object as returned by ‘outlier.tree‘.

Value

A data frame with the outliers, which can be pretty-printed by function ‘print‘ from this same package.

hypothyroid

Data about thyroid hormones for anonymous patients

Description

This data contains several obvious outliers from misspellings in data entry. From Garavan Institute. For more details see link at the bottom.

Usage

```
data(hypothyroid)
```

Format

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

Source

http://archive.ics.uci.edu/ml/datasets/thyroid+disease

outlier.tree *Outlier Tree*

Description

Fit Outlier Tree model to normal data with perhaps some outliers.

Usage

```
outlier.tree(df, max_depth = 4, min_gain = 0.01, z_norm = 2.67,
  z_outlier = 8, pct_outliers = 0.01, min_size_numeric = 25,
  min_size_categ = 50, categ_split = "binarize",
  categ_outliers = "tail", cols_ignore = NULL, follow_all = FALSE,
  gain_as_pct = TRUE, save_outliers = FALSE, outliers_print = 10,
  nthreads = parallel::detectCores())
```

Arguments

df	Data Frame with normal data that might contain some outliers. See details for allowed column types.
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'. 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.

`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.

`save_outliers` Whether to store outliers detected in ‘df‘ in the object that is returned. These outliers can then be extracted from the returned object through function ‘extract.training.outliers‘.

`outliers_print` Maximum number of flagged outliers in the training data to print after fitting the model. Pass zero or ‘NULL‘ to avoid printing any. Outliers can be printed

from resulting data frame afterwards through the ‘predict‘ method, or through the ‘print‘ method (on the extracted outliers, not on the model object) if passing ‘save_outliers‘ = ‘TRUE‘.

`nthreads` Number of parallel threads to use. When fitting the model, it will only use up to one thread per column, while for prediction it will use up to one thread per row. The more threads that are used, the more memory will be required and allocated, so using more threads will not always lead to better speed. Can be changed after the object is already initialized.

Details

Explainable outlier detection through decision-tree grouping. Tries to detect outliers by generating decision trees that attempt to "predict" the values of each column based on each other column, testing in each branch of every tried split (if it meets some minimum criteria) whether there are observations that seem too distant from the others in a 1-D distribution for the column that the split tries to "predict" (unlike other methods, this will not generate a score for each observation).

Splits are based on gain, while outlierness is based on confidence intervals. Similar in spirit to the GritBot software developed by RuleQuest research.

Supports columns of types numeric (either as type ‘numeric‘ or ‘integer‘), categorical (either as type ‘character‘ or ‘factor‘ with unordered levels), boolean (as type ‘logical‘), and ordinal (as type ‘factor‘ with ordered levels as checked by ‘is.ordered‘). Can handle missing values in any of them. Can also pass dates/timestamps that will get converted to numeric but shown as dates/timestamps in the output. Offers option to set columns to be used only for generating conditions without looking at outliers in them.

Infinite values will be taken into consideration when the column is used to split another column (that is, +inf will go into the branch that is greater than something, -inf into the other branch), but when a column is the target of the split, they will be taken as missing - that is, it will not report infinite values as outliers.

Value

An object with the fitted model that can be used to detect more outliers in new data, and from which outliers in the training data can be extracted (when passing ‘save_outliers‘ = ‘TRUE‘).

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.outliertree extract.training.outliers hypothyroid unpack.outlier.tree

Examples

```
library(outliertree)
```

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

### fit the model and get a print of outliers
model <- outlier.tree(hypothyroid,
  outliers_print=10,
  save_outliers=TRUE,
  nthreads=1)

### extract outlier info as R list
outliers <- extract.training.outliers(model)
summary(outliers)

### information for row 745
outliers[[745]]

### use custom row names
df.w.names <- hypothyroid
row.names(df.w.names) <- paste0("rownum", 1:nrow(hypothyroid))
outliers.w.names <- predict(model, df.w.names, return_outliers=TRUE)
outliers.w.names[["rownum745"]]
```

`predict.outliertree` *Predict method for Outlier Tree*

Description

Predict method for Outlier Tree

Usage

```
## S3 method for class 'outliertree'
predict(object, newdata, outliers_print = 15,
  return_outliers = TRUE, ...)
```

Arguments

`object`	An Outlier Tree object as returned by ‘outlier.tree‘.
`newdata`	A Data Frame in which to look for outliers according to the fitted model.
`outliers_print`	How many outliers to print. Pass zero or ‘NULL‘ to avoid printing them. Must pass at least one of ‘outliers_print‘ and ‘return_outliers‘.
`return_outliers`	Whether to return the outliers in an R object (otherwise will just print them).
`...`	Not used.

Details

Note that after loading a serialized object from ‘outlier.tree‘ through ‘readRDS‘ or ‘load‘, it will only de-serialize the underlying C++ object upon running ‘predict‘ or ‘print‘, so the first run will be slower, while subsequent runs will be faster as the C++ object will already be in-memory.

Value

If passing ‘return_outliers‘ = ‘TRUE‘, 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‘ and ‘summary‘ (they do the same thing). Otherwise, will not return anything, but will print the outliers if any are detected. Note that, while the object that is returned will display a short summary of only some observations when printing it in the console, it actually contains information for all rows, and can be subsetted to obtain information specific to one row.

See Also

outlier.tree print.outlieroutputs unpack.outlier.tree

Examples

```
library(outliertree)
### random data frame with an obvious outlier
nrows = 100
set.seed(1)
df = data.frame(
    numeric_col1 = c(rnorm(nrows - 1), 1e6),
    numeric_col2 = rgamma(nrows, 1),
    categ_col    = sample(c('categA', 'categB', 'categC'),
        size = nrows, replace = TRUE)
    )

### test data frame with another obvious outlier
nrows_test = 50
df_test = data.frame(
    numeric_col1 = rnorm(nrows_test),
    numeric_col2 = c(-1e6, rgamma(nrows_test - 1, 1)),
    categ_col    = sample(c('categA', 'categB', 'categC'),
        size = nrows_test, replace = TRUE)
)

### fit model on training data
outliers_model = outlier.tree(df, outliers_print=FALSE, nthreads=1)

### find the test outlier
test_outliers = predict(outliers_model, df_test,
    outliers_print = 1, return_outliers = TRUE)

### retrieve the outlier info (for row 1) as an R list
test_outliers[[1]]
```

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

Description

Pretty-prints outliers as output by the ‘predict‘ function from an Outlier Tree model (as generated by function ‘outlier.tree‘), or by ‘extract.training.outliers‘. Same as function ‘summary‘.

Usage

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

Arguments

x	Outliers as returned by predict method on an object from ‘outlier.tree‘.
outliers_print	Maximum number of outliers to print.
only_these_rows	Specific rows to print (either numbers if the row names in the original data frame were null, or the row names they had if non-null). Pass ‘NULL‘ to print information about potentially all rows
...	Not used.

Value

No return value.

See Also

outlier.tree predict.outliertree

Examples

```
### Example re-printing results for selected rows
library(outliertree)
data("hypothyroid")

### Fit model
otree <- outlier.tree(hypothyroid,
  nthreads=1,
  outliers_print=0)

### Store predictions
pred <- predict(otree,
  hypothyroid,
  outliers_print=0,
  return_outliers=TRUE)

### Print stored predictions
### Row 531 is an outlier, but 532 is not
print(pred, only_these_rows = c(531, 532))
```

print.outliertree *Print summary information from Outlier Tree model*

Description

Displays general statistics from a fitted Outlier Tree model (same as 'summary'). For printing the outliers discovered, use function 'print' on the returned outliers (e.g. from 'predict'), not on the model object iself.

Usage

```
## S3 method for class 'outliertree'
print(x, ...)
```

Arguments

x	An Outlier Tree model as produced by function 'outlier.tree'.
...	Not used.

Details

Note that after loading a serialized object from 'outlier.tree' through 'readRDS' or 'load', it will only de-serialize the underlying C++ object upon running 'predict' or 'print', so the first run will be slower, while subsequent runs will be faster as the C++ object will already be in-memory.

summary.outlieroutputs *Print outliers in human-readable format*

Description

Pretty-prints outliers as output by the 'predict' function from an Outlier Tree model (as generated by function 'outlier.tree'), or by 'extract.training.outliers'. Same as function 'print' (see the documentation of 'print' for more information about the parameters).

Usage

```
## S3 method for class 'outlieroutputs'
summary(object, outliers_print = 15, ...)
```

Arguments

object	Outliers as returned by predict method on an object from 'outlier.tree'.
outliers_print	Maximum number of outliers to print.
...	Not used.

See Also

print.outlieroutputs

`summary.outliertree` *Print summary information from Outlier Tree model*

Description

Displays general statistics from a fitted Outlier Tree model (same as ‘print‘). For printing the outliers discovered, use function ‘print‘ on the returned outliers (e.g. from ‘predict‘), not on the model object iself.

Usage

```
## S3 method for class 'outliertree'
summary(object, ...)
```

Arguments

`object`	An Outlier Tree model as produced by function ‘outlier.tree‘.
`...`	Not used.

See Also

print.outliertree

`unpack.outlier.tree` *Unpack Outlier Tree model after de-serializing*

Description

After persisting an outlier tree model object through ‘saveRDS‘, ‘save‘, or restarting a session, the underlying C++ objects that constitute the outlier tree model and which live only on the C++ heap memory are not saved along, thus not restored after loading a saved model through ‘readRDS‘ or ‘load‘.

The model object however keeps serialized versions of the C++ objects as raw bytes, from which the C++ objects can be reconstructed, and are done so automatically after calling ‘predict‘, ‘print‘, or ‘summary‘ on the freshly-loaded object from ‘readRDS‘ or ‘load‘.

But due to R’s environments system (as opposed to other systems such as Python which can use pass-by-reference), they will only be re-constructed in the environment that is calling ‘predict‘, ‘print‘, etc. and not in higher-up environments (i.e. if you call ‘predict‘ on the object from inside different functions, each function will have to reconstruct the C++ objects independently and they will only live within the function that called ‘predict‘).

This function serves as an environment-level unpacker that will reconstruct the C++ object in the environment in which it is called (i.e. if you need to call ‘predict‘ from inside multiple functions, use this function before passing the freshly-loaded model object to those other functions, and then they will not need to reconstruct the C++ objects anymore), in the same way as ‘predict‘ or ‘print‘, but without producing any outputs or messages.

Usage

```
unpack.outlier.tree(model)
```

Arguments

model	An Outlier Tree object as returned by ‘outlier.tree‘, which has been just loaded from a disk file through ‘readRDS‘, ‘load‘, or a session restart.

Value

No return value. Object is modified in-place.

Examples

```
### Warning: this example will generate a temporary .Rds
### file in your temp folder, and will then delete it
library(outliertree)
set.seed(1)
df <- as.data.frame(matrix(rnorm(1000), nrow = 250))
otree <- outlier.tree(df, outliers_print=0, nthreads=1)
temp_file <- file.path(tempdir(), "otree.Rds")
saveRDS(otree, temp_file)
otree2 <- readRDS(temp_file)
file.remove(temp_file)

### will de-serialize inside, but object is short-lived
wrap_predict <- function(model, data) {
    pred <- predict(model, data, outliers_print = 0)
    cat("pointer inside function is this: ")
    print(model$obj_from_cpp$ptr_model)
    return(pred)
}
temp <- wrap_predict(otree2, df)
cat("pointer outside function is this: \n")
print(otree2$obj_from_cpp$ptr_model) ### pointer to the C++ object

### now unpack the C++ object beforehand
unpack.outlier.tree(otree2)
print("after unpacking beforehand")
temp <- wrap_predict(otree2, df)
cat("pointer outside function is this: \n")
print(otree2$obj_from_cpp$ptr_model)
```

Index