# Package 'ale'

October 19, 2023

**Title** Interpretable Machine Learning and Statistical Inference with Accumulated Local Effects (ALE)

**Version** 0.2.0

**Description** Accumulated Local Effects (ALE) were initially developed as a model-agnostic approach for global explanations of the results of black-box machine learning algorithms. ALE has a key advantage over other approaches like partial dependency plots (PDP) and SHapley Additive exPlanations (SHAP): its values represent a clean functional decomposition of the model. As such, ALE values are not affected by the presence or absence of interactions among variables in a mode. Moreover, its computation is relatively rapid. This package rewrites the original code from the 'ALEPlot' package for calculating ALE data and it completely reimplements the plotting of ALE values. It also extends the original ALE concept to add bootstrap-based confidence intervals and ALE-based statistics that can be used for statistical inference. For more details, see Okoli, Chitu. 2023. "Statistical Inference Using Machine Learning and Classical Techniques Based on Accumulated Local Effects (ALE)." arXiv. <arXiv:2310.09877>. <doi:10.48550/arXiv.2310.09877>.

**License** GPL-2

**Language** en-ca

**Encoding** UTF-8

**RoxygenNote** 7.2.3

**Suggests** ALEPlot, gbm, gridExtra, knitr, mgcv, nnet, readr, rmarkdown, testthat (>= 3.0.0)

**VignetteBuilder** knitr

**Imports** assertthat, broom, dplyr, ellipsis, grDevices, labeling, methods, purrr, stats, stringr, tidyr, yaImpute

**Depends** R (>= 3.5.0), ggplot2

**URL** https://github.com/Tripartio/ale

**BugReports** https://github.com/Tripartio/ale/issues

**Config/testthat/edition** 3

**LazyData** true

**NeedsCompilation** no

**Author** Chitu Okoli [aut, cre] (<https://orcid.org/0000-0001-5574-7572>),
Dan Apley [cph] (The current code for calculating ALE interaction values is copied with frew changes from Dan Apley's ALEPlot package. We gratefully acknowledge his open-source contribution. However, he was not directly involved with this ale package.)

**Maintainer** Chitu Okoli <Chitu.Okoli@skema.edu>

**Repository** CRAN

**Date/Publication** 2023-10-19 21:30:05 UTC

# R topics documented:



---

`ale` *Create and return ALE data, statistics, and plots*

---

## Description

`ale` is the central function that manages the creation of ALE data and plots for one-way ALE. For two-way interactions, see `ale_ixn`. This function calls `ale_core` (a non-exported function) that manages the ALE data and plot creation in detail. For details, see the introductory vignette for this package or the details and examples below.

### Custom predict function

The calculation of ALE requires modifying several values of the original `data`. Thus, `ale` needs direct access to a `predict` function that work on `model`. By default, `ale` uses a generic default `predict` function of the form `predict(model_object, new_data)` with the default prediction type of 'response'. If, however, the desired prediction values are not generated with that format, the user must specify what they want. Most of the time, the only modification needed is to change the prediction type to some other value by setting the `pred_type` argument (e.g., to 'prob' to generated classification probabilities). But if the desired predictions need a different function signature, then the user must create a custom prediction function and pass it to `pred_fun`. The requirements for this custom function are:

- It must take two arguments and nothing else: `object` (a model) and `newdata` (a dataframe or compatible table type). These argument names are according to the R convention for the generic stats::predict function.
- It must return a vector of numeric values as the prediction.

You can see an example below of a custom prediction function.

**Note:** `survival` models probably do not need a custom prediction function but `y_col` must be set to the name of the binary event column and `pred_type` must be set to the desired prediction type.

**ALE statistics**

For details about the ALE-based statistics (ALED, ALER, NALED, and NALER), see `vignette("ale-statistics")`.

**About the `ale` package**

Accumulated Local Effects (ALE) were initially developed as a model-agnostic approach for global explanations of the results of black-box machine learning algorithms. ALE has a key advantage over other approaches like partial dependency plots (PDP) and SHapley Additive exPlanations (SHAP): its values represent a clean functional decomposition of the model. As such, ALE values are not affected by the presence or absence of interactions among variables in a mode. Moreover, its computation is relatively rapid. This package rewrites the original code from the 'ALEPlot' package for calculating ALE data and it completely reimplements the plotting of ALE values. It also extends the original ALE concept to add bootstrap-based confidence intervals and ALE-based statistics that can be used for statistical inference. For more details, see Okoli, Chitu. 2023. "Statistical Inference Using Machine Learning and Classical Techniques Based on Accumulated Local Effects (ALE)." arXiv. https://arxiv.org/abs/2310.09877.

## Usage

```
ale(
  data,
  model,
  x_cols = NULL,
  y_col = NULL,
  ...,
  output = c("plots", "data", "stats"),
  pred_fun = function(object, newdata) {
     stats::predict(object = object, newdata =
    newdata, type = pred_type)
 },
  pred_type = "response",
  x_intervals = 100,
  boot_it = 0,
  seed = 0,
  boot_alpha = 0.05,
  boot_centre = "mean",
  relative_y = "median",
  y_type = NULL,
  median_band = 0.05,
  rug_sample_size = 500,
  min_rug_per_interval = 1,
  ale_xs = NULL,
  ale_ns = NULL,
  silent = FALSE
)
```

## Arguments

| | |
|---|---|
| data | dataframe. Dataset from which to create predictions for the ALE. |
| model | model object. Model for which ALE should be calculated. May be any kind of R object that can make predictions from data. |
| x_cols | character. Vector of column names from data for which one-way ALE data is to be calculated (that is, simple ALE without interactions). If not provided, ALE will be created for all columns in data except y_col. |
| y_col | character length 1. Name of the outcome target label (y) variable. If not provided, ale will try to detect it automatically. For non-standard models, y_col should be provided. For survival models, set y_col to the name of the binary event column; in that case, pred_type should also be specified. |
| ... | not used. Inserted to require explicit naming of subsequent arguments. |
| output | character in c('plots', 'data', 'stats'). Vector of types of results to return. 'plots' will return an ALE plot; 'data' will return the source ALE data; 'stats' will return ALE statistics. Each option must be listed to return the specified component. By default, all are returned. |
| pred_fun, pred_type | function,character length 1. pred_fun is a function that returns a vector of predicted values of type pred_type from model on data. See details. |
| x_intervals | positive integer length 1. Maximum number of intervals on the x-axis for the ALE data for each column in x_cols. The number of intervals that the algorithm generates might eventually be fewer than what the user specifies if the data values for a given x value do not support that many intervals. |
| boot_it | non-negative integer length 1. Number of bootstrap iterations for the ALE values. If boot_it = 0 (default), then ALE will be calculated on the entire dataset with no bootstrapping. |
| seed | integer length 1. Random seed. Supply this between runs to assure that identical random ALE data is generated each time |
| boot_alpha | numeric length 1 from 0 to 1. Alpha for percentile-based confidence interval range for the bootstrap intervals; the bootstrap confidence intervals will be the lowest and highest (1 - 0.05) / 2 percentiles. For example, if boot_alpha = 0.05 (default), the intervals will be from the 2.5 and 97.5 percentiles. |
| boot_centre | character length 1 in c('mean', 'median'). When bootstrapping, the main estimate for ale_y is considered to be boot_centre. Regardless of the value specified here, both the mean and median will be available. |
| relative_y | character length 1 in c('median', 'mean', 'zero'). The ale_y values will be adjusted relative to this value. 'median' is the default. 'zero' will maintain the default of ALEPlot::ALEPlot, which is not shifted. |
| y_type | character length 1. Datatype of the y (outcome) variable. Must be one of c('binary', 'numeric', 'multinomial', 'ordinal'). Normally determined automatically; only provide for complex non-standard models that require it. |
| median_band | numeric length 1 from 0 to 1. Alpha for "confidence interval" range for printing bands around the median for single-variable plots. The band range will be the median value of y ± median_band. |

`rug_sample_size`, `min_rug_per_interval`
: single non-negative integer length 1. Rug plots are normally down-sampled otherwise they are too slow. `rug_sample_size` specifies the size of this sample. To prevent down-sampling, set to `Inf`. To suppress rug plots, set to 0. When down-sampling, the rug plots maintain representativeness of the data by guaranteeing that each of the `x_intervals` intervals will retain at least `min_rug_per_interval` elements; usually set to just 1 or 2.

`ale_xs`, `ale_ns`
: list of ale_x and ale_n vectors. If provided, these vectors will be used to set the intervals of the ALE x axis for each variable. By default (NULL), the function automatically calculates the ale_x intervals. `ale_xs` is normally used in advanced analyses where the ale_x intervals from a previous analysis are reused for subsequent analyses (for example, for full model bootstrapping; see the `model_bootstrap` function).

`silent`
: logical length 1, default FALSE. If TRUE, do not display any non-essential messages during execution (such as progress bars). Regardless, any warnings and errors will always display.

## Details

ale_core.R

Core functions for the ale package: ale, ale_ixn, and ale_core

## Value

list with elements `data`, `plots`, and `stats` as requested in the `output` argument. Each of these is a list named by the x variables with the respective values for each variable. In addition, the return object recapitulates several elements that were passed as arguments that apply to all the x variables for the ALE calculation.

## Author(s)

Chitu Okoli <Chitu.Okoli@skema.edu>

## References

Okoli, Chitu. 2023. "Statistical Inference Using Machine Learning and Classical Techniques Based on Accumulated Local Effects (ALE)." arXiv. https://arxiv.org/abs/2310.09877.

## See Also

Useful links:

- https://github.com/Tripartio/ale
- Report bugs at https://github.com/Tripartio/ale/issues

## Examples

```
diamonds
set.seed(0)
diamonds_sample <- diamonds[sample(nrow(diamonds), 1000), ]

# Split the dataset into training and test sets
# https://stackoverflow.com/a/54892459/2449926
set.seed(0)
train_test_split <- sample(
  c(TRUE, FALSE), nrow(diamonds_sample), replace = TRUE, prob = c(0.8, 0.2)
)
diamonds_train <- diamonds_sample[train_test_split, ]
diamonds_test <- diamonds_sample[!train_test_split, ]


# Create a GAM model with flexible curves to predict diamond price
# Smooth all numeric variables and include all other variables
# Build model on training data, not on the full dataset.
gam_diamonds <- mgcv::gam(
  price ~ s(carat) + s(depth) + s(table) + s(x) + s(y) + s(z) +
    cut + color + clarity,
  data = diamonds_train
)
summary(gam_diamonds)


# Simple ALE without bootstrapping
ale_gam_diamonds <- ale(diamonds_test, gam_diamonds)



# Plot the ALE data
gridExtra::grid.arrange(grobs = ale_gam_diamonds$plots, ncol = 2)

# Bootstrapped ALE
# This can be slow, since bootstrapping runs the algorithm boot_it times

# Create ALE with 100 bootstrap samples
ale_gam_diamonds_boot <- ale(diamonds_test, gam_diamonds, boot_it = 100)

# Bootstrapped ALEs print with confidence intervals
gridExtra::grid.arrange(grobs = ale_gam_diamonds_boot$plots, ncol = 2)


# If the predict function you want is non-standard, you may define a
# custom predict function. It must return a single numeric vector.
custom_predict <- function(object, newdata) {
  predict(object, newdata, type = 'link', se.fit = TRUE)$fit
}

ale_gam_diamonds_custom <- ale(
  diamonds_test, gam_diamonds,
```

```
  pred_fun = custom_predict
)

# Plot the ALE data
gridExtra::grid.arrange(grobs = ale_gam_diamonds_custom$plots, ncol = 2)
```

---

ale_ixn *Create and return ALE interaction data, statistics, and plots*

---

## Description

This is the central function that manages the creation of ALE data and plots for two-way ALE interactions. For simple one-way ALE, see `ale`. See documentation there for functionality shared between both functions. For details, see the introductory vignette for this package or the details and examples below.

For the plots, `n_y_quant` is the number of quantiles into which to divide the predicted variable (y). The middle quantiles are grouped specially:

- The middle quantile is the `median_band` confidence interval around the median. This middle quantile is special because it generally represents no meaningful interaction.
- The quantiles above and below the middle are extended from the borders of the middle quantile to the regular borders of the other quantiles.

There will always be an odd number of quantiles: the special middle quantile plus an equal number of quantiles on each side of it. If n_y_quant is even, then a middle quantile will be added to it. If n_y_quant is odd, then the number specified will be used, including the middle quantile.

## Usage

```
ale_ixn(
  data,
  model,
  x1_cols = NULL,
  x2_cols = NULL,
  y_col = NULL,
  ...,
  output = c("plots", "data"),
  pred_fun = function(object, newdata) {
     stats::predict(object = object, newdata =
    newdata, type = pred_type)
 },
  pred_type = "response",
  x_intervals = 100,
  relative_y = "median",
  y_type = NULL,
```

```
  median_band = 0.05,
  rug_sample_size = 500,
  min_rug_per_interval = 1,
  ale_xs = NULL,
  n_x1_int = 20,
  n_x2_int = 20,
  n_y_quant = 10,
  silent = FALSE
)
```

## Arguments

| | |
|---|---|
| `data` | See documentation for `ale` |
| `model` | See documentation for `ale` |
| `x1_cols, x2_cols` | character. Vectors of column names from `data` for which two-way interaction ALE data is to be calculated. ALE data will be calculated for each x1 column interacting with each x2 column. x1_cols can be of any standard datatype (logical, factor, or numeric) but x2_cols can only be numeric. If `ixn` is TRUE, then both values must be provided. |
| `y_col` | See documentation for `ale` |
| `...` | not used. Inserted to require explicit naming of subsequent arguments. |
| `output` | See documentation for `ale` |
| `pred_fun, pred_type` | See documentation for `ale` |
| `x_intervals` | See documentation for `ale` |
| `relative_y` | See documentation for `ale` |
| `y_type` | See documentation for `ale` |
| `median_band` | See documentation for `ale` |
| `rug_sample_size, min_rug_per_interval` | See documentation for `ale` |
| `ale_xs` | See documentation for `ale` |
| `n_x1_int, n_x2_int` | positive scalar integer. Number of intervals for the x1 or x2 axes respectively for interaction plot. These values are ignored if x1 or x2 are not numeric (i.e, if they are logical or factors). |
| `n_y_quant` | positive scalar integer. Number of intervals over which the range of y values is divided for the colour bands of the interaction plot. See details. |
| `silent` | See documentation for `ale` |

## Value

list of ALE interaction data tibbles and plots. The list has two levels of depth:

- The first level is named by the x1 variables.

- Within each x1 variable list, the second level is named by the x2 variables.
- Within each x1-x2 list element, the data or plot is returned as requested in the `output` argument.

## Examples

```
diamonds
set.seed(0)
diamonds_sample <- diamonds[sample(nrow(diamonds), 1000), ]

# Split the dataset into training and test sets
# https://stackoverflow.com/a/54892459/2449926
set.seed(0)
train_test_split <- sample(
  c(TRUE, FALSE), nrow(diamonds_sample), replace = TRUE, prob = c(0.8, 0.2)
)
diamonds_train <- diamonds_sample[train_test_split, ]
diamonds_test <- diamonds_sample[!train_test_split, ]


# Create a GAM model with flexible curves to predict diamond price
# Smooth all numeric variables and include all other variables
# Build model on training data, not on the full dataset.
gam_diamonds <- mgcv::gam(
  price ~ s(carat) + s(depth) + s(table) + s(x) + s(y) + s(z) +
    cut + color + clarity,
  data = diamonds_train
)
summary(gam_diamonds)

# ALE two-way interactions
ale_ixn_gam_diamonds <- ale_ixn(diamonds_test, gam_diamonds)



# Print interaction plots
ale_ixn_gam_diamonds$plots |>
  purrr::walk(\(.x1) {  # extract list of x1 ALE outputs
    gridExtra::grid.arrange(grobs = .x1, ncol = 2)  # plot all x1 plots
  })
```

---

census *Census Income*

---

## Description

Census data that indicates, among other details, if the respondent's income exceeds $50,000 per year. Also known as "Adult" dataset.

## Usage

```
census
```

## Format

A tibble with 32,561 rows and 15 columns:

**higher_income** TRUE if income > $50,000

**age** continuous

**workclass** Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked

**fnlwgt** continuous. "A proxy for the demographic background of the people: 'People with similar demographic characteristics should have similar weights'" For more details, see https://www.openml.org/search?type=d

**education** Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool

**education_num** continuous

**marital_status** Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse

**occupation** Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces

**relationship** Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried

**race** White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black

**sex** Female, Male

**capital_gain** continuous

**capital_loss** continuous

**hours_per_week** continuous

**native_country** United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinidad&Tobago, Peru, Hong, Holland-Netherlands



## Source

Becker,Barry and Kohavi,Ronny. (1996). Adult. UCI Machine Learning Repository. https://doi.org/10.24432/C5XW20.

model_bootstrap *model_bootstrap.R*

## Description

Execute full model bootstrapping with ALE calculation on each bootstrap run

## Usage

```
model_bootstrap(
  data,
  model_call_string,
  ...,
  boot_it = 100,
  seed = 0,
  boot_alpha = 0.05,
  boot_centre = "mean",
  output = c("ale", "model_stats", "model_coefs"),
  ale_options = list(),
  tidy_options = list(),
  glance_options = list(),
  silent = FALSE
)
```

## Arguments

| | |
|---|---|
| data | dataframe. Dataset that will be bootstrapped. |
| model_call_string | character. Character string of the full call for the model, except that the data option must be left out. The data option will be replaced with the data argument. |
| ... | not used. Inserted to require explicit naming of subsequent arguments. |
| boot_it | integer from 0 to Inf. Number of bootstrap iterations. If boot_it = 0, then the model is run as normal once on the full data with no bootstrap. |
| seed | integer. Random seed. Supply this between runs to assure identical bootstrap samples are generated each time on the same data. |
| boot_alpha | numeric. The confidence level for the bootstrap confidence intervals is 1 - boot_alpha. For example, the default 0.05 will give a 95% confidence interval, that is, from the 2.5% to the 97.5% percentile. |
| boot_centre | See documentation for ale |
| output | character vector. Which types of bootstraps to calculate and return:<br>• 'ale': Calculate and return bootstrapped ALE data and plot.<br>• 'model_stats': Calculate and return bootstrapped overall model statistics.<br>• 'model_coefs': Calculate and return bootstrapped model coefficients. |

- 'boot_data': Return full data for all bootstrap iterations. This data will always be calculated because it is needed for the bootstrap averages. By default, it is not returned except if included in this `output` argument.

`ale_options, tidy_options, glance_options`
: list of named arguments. Arguments to pass to the `ale`, `broom::tidy`, or `broom::glance` functions, respectively, beyond (or overriding) the defaults.

`silent`
: See documentation for `ale`

## Details

No modelling results, with or without ALE, should be considered reliable without being bootstrapped. For large datasets with clear separation between training and testing samples, `ale` bootstraps the ALE results of the test data. However, when a dataset is too small to be subdivided into training and test sets, then the entire model should be bootstrapped. That is, multiple models should be trained, one on each bootstrap sample. The reliable results are the average results of all the bootstrap models, however many there are. For details, see the vignette on small datasets or the details and examples below.

`model_bootstrap` automatically carries out full-model bootstrapping suitable for small datasets. Specifically, it:

- Creates multiple bootstrap samples (default 100; the user can specify any number);
- Creates a model on each bootstrap sample;
- Calculates model overall statistics, variable coefficients, and ALE values for each model on each bootstrap sample;
- Calculates the mean, median, and lower and upper confidence intervals for each of those values across all bootstrap samples.

## Value

list with tibbles of the following elements (depending on values requested in the `output` argument:

- model_stats: bootstrapped results from `broom::glance`
- model_coefs: bootstrapped results from `broom::tidy`
- ale: bootstrapped ALE results
  - data: ALE data (see `ale` for details about the format)
  - stats: ALE statistics. The same data is duplicated with different views that might be variously useful. The column
    - by_term: statistic, estimate, conf.low, median, mean, conf.high. ("term" means variable name.) The column names are compatible with the `broom` package. The confidence intervals are based on the `ale` function defaults; they can be changed with the `ale_options` argument. The estimate is the median or the mean, depending on the `boot_centre` argument.
    - by_statistic: term, estimate, conf.low, median, mean, conf.high.
    - estimate: term, then one column per statistic Provided with the default estimate. This view does not present confidence intervals.
  - plots: ALE plots (see `ale` for details about the format)

- boot_data: full bootstrap data (not returned by default)
- other values: the `boot_it`, `seed`, `boot_alpha`, and `boot_centre` arguments that were originally passed are returned for reference.

## References

Okoli, Chitu. 2023. "Statistical Inference Using Machine Learning and Classical Techniques Based on Accumulated Local Effects (ALE)." arXiv. https://arxiv.org/abs/2310.09877.

## Examples

```
# attitude dataset
attitude

## ALE for general additive models (GAM)
## GAM is tweaked to work on the small dataset.
gam_attitude <- mgcv::gam(rating ~ complaints + privileges + s(learning) +
                            raises + s(critical) + advance,
                          data = attitude)
summary(gam_attitude)

# Full model bootstrapping
# Only 3 bootstrap iterations for a rapid example; default is 100
# Increase value of boot_it for more realistic results
mb_gam <- model_bootstrap(
  attitude,
  'mgcv::gam(rating ~ complaints + privileges + s(learning) +
              raises + s(critical) + advance)',
  boot_it = 3
)

# Model statistics and coefficients
mb_gam$model_stats
mb_gam$model_coefs


# Plot ALE
gridExtra::grid.arrange(grobs = mb_gam$ale$plots, ncol = 2)
```

---

`var_cars` *Multi-variable transformation of the mtcars dataset.*

---

## Description

This is a transformation of the `mtcars` dataset from R to produce a small dataset with each of the fundamental datatypes: logical, factor, ordered, integer, and double. Most of the transformations are obvious, but two are noteworthy:

- For the unordered factor, the country of the car manufacturer is obtained based on the row names of `mtcars`. This `var_cars` version does not have row names.
- For the ordered factor, gears 3, 4, and 5 are encoded as 'three', 'four', and 'five', respectively. The text labels make it explicit that the variable is ordinal, yet the number names make the order crystal clear.

Here is the original description of the `mtcars` dataset:

The data was extracted from the 1974 *Motor Trend* US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models).

## Usage

```
var_cars
```

## Format

A tibble with 32 observations on 12 variables.

**mpg** `double`: Miles/(US) gallon

**cyl** `integer`: Number of cylinders

**disp** `double`: Displacement (cu.in.)

**hp** `double`: Gross horsepower

**drat** `double`: Rear axle ratio

**wt** `double`: Weight (1000 lbs)

**qsec** `double`: 1/4 mile time

**vs** `logical`: Engine (0 = V-shaped, 1 = straight)

**am** `logical`: Transmission (0 = automatic, 1 = manual)

**gear** `ordered`: Number of forward gears

**carb** `integer`: Number of carburetors

**country** `factor`: Country of car manufacturer

## Note

Henderson and Velleman (1981) comment in a footnote to Table 1: 'Hocking (original transcriber)'s noncrucial coding of the Mazda's rotary engine as a straight six-cylinder engine and the Porsche's flat engine as a V engine, as well as the inclusion of the diesel Mercedes 240D, have been retained to enable direct comparisons to be made with previous analyses.'

## References

Henderson and Velleman (1981), Building multiple regression models interactively. *Biometrics*, **37**, 391–411.

# Index