

Package ‘fmeffects’

March 7, 2024

Title Model-Agnostic Interpretations with Forward Marginal Effects

Version 0.1.2

Description Create local, regional, and global explanations for any machine learning model with forward marginal effects. You provide a model and data, and 'fmeffects' computes feature effects. The package is based on the theory in: C. A. Scholbeck, G. Casalicchio, C. Molnar, B. Bischl, and C. Heumann (2022) <[arXiv:2201.08837](https://arxiv.org/abs/2201.08837)>.

License LGPL-3

Encoding UTF-8

RoxygenNote 7.2.3

Suggests caret, knitr, mlr3verse, ranger, rmarkdown, rpart

Imports checkmate, data.table, partykit, ggparty, ggplot2, cowplot, R6, testthat

Collate 'ExtrapolationDetector.R' 'FME.R' 'FMEPlot.R'
'NonLinearityMeasure.R' 'Partitioning.R' 'PartitioningCtree.R'
'PartitioningPlot.R' 'PartitioningRpart.R' 'Predictor.R'
'PredictorCaret.R' 'PredictorMLR3.R' 'PredictorParsnip.R'
'Pruner.R' 'S3.R' 'ame.R' 'bikes.R' 'misc.R' 'zzz.R'

URL <https://holgstr.github.io/fmeffects/>,
<https://github.com/holgstr/fmeffects>

BugReports <https://github.com/holgstr/fmeffects/issues>

VignetteBuilder knitr

NeedsCompilation no

Author Holger Löwe [cre, aut],
Christian Scholbeck [aut],
Christian Heumann [rev],
Bernd Bischl [rev],
Giuseppe Casalicchio [rev]

Maintainer Holger Löwe <hbj.loewe@gmail.com>

Depends R (>= 3.5.0)

Repository CRAN

Date/Publication 2024-03-07 20:10:02 UTC

R topics documented:

fmeffects-package	2
ame	3
AverageMarginalEffects	4
bikes	7
came	8
fme	9
ForwardMarginalEffect	11
makePredictor	13
Partitioning	14
PartitioningCtree	16
PartitioningRpart	17
plot.ForwardMarginalEffect	18
plot.Partitioning	18
Predictor	19
PredictorCaret	20
PredictorMLR3	21
PredictorParsnip	22
print.ForwardMarginalEffect	23
print.Partitioning	23
summary.AverageMarginalEffects	24
summary.ForwardMarginalEffect	24
summary.Partitioning	25
Index	26

fmeffects-package *fmeffects*

Description

Computes forward marginal effects (FME) for arbitrary supervised machine learning models. You provide a model and data, and 'fmeffects' gives you feature effects.

Author(s)

Maintainer: Holger Löwe <hbj.loewe@gmail.com>

Authors:

- Christian Scholbeck <christian.scholbeck@stat.uni-muenchen.de>

Other contributors:

- Christian Heumann <christian.heumann@stat.uni-muenchen.de> [reviewer]
- Bernd Bischl <bernd.bischl@stat.uni-muenchen.de> [reviewer]
- Giuseppe Casalicchio <giuseppe.casalicchio@stat.uni-muenchen.de> [reviewer]

See Also

Useful links:

- <https://holgstr.github.io/fmeffects/>
- <https://github.com/holgstr/fmeffects>
- Report bugs at <https://github.com/holgstr/fmeffects/issues>

 ame

Computes AMEs for every feature (or a subset of features) of a model.

Description

This is a wrapper function for `AverageMarginalEffects$new(...)$compute()`. It computes Average Marginal Effects (AME) based on Forward Marginal Effects (FME) for a model. The AME is a simple mean FME and computed w.r.t. a feature variable and a model.

Usage

```
ame(model, data, target, features = NULL, ep.method = "none")
```

Arguments

model	The (trained) model, with the ability to predict on new data. This must be an <code>Learner</code> (<code>mlr3</code>) or <code>train</code> (<code>caret</code>) object.
data	The data used for computing AMEs, must be <code>data.frame</code> or <code>data.table</code> .
target	A string specifying the model's target variable.
features	A named character vector of the names of the feature variables for which AMEs should be computed, together with the desired step sizes.
ep.method	String specifying the method used for extrapolation detection. One of "none" or "envelope". Defaults to "none".

Value

An `AverageMarginalEffects` object, with a field `results` containing a list of summary statistics, including

- `Feature`: The name of the feature.
- `step.size`: The step.size w.r.t. the specified feature.
- `AME`: The Average Marginal Effect for a step of length `step.size` w.r.t. the specified feature.
- `SD`: The standard deviation of FMEs for the specified feature and `step.size`.
- `0.25`: The 0.25-quantile of FMEs for the specified feature and `step.size`.
- `0.75`: The 0.75-quantile of FMEs for the specified feature and `step.size`.
- `n`: The number of observations included for the computation of the AME. This can vary for the following reasons: For categorical features, FMEs are only computed for observations where the original category is not the `step.size` category. For numerical features, FMEs are only computed for observations that are not extrapolation points (if `ep.method` is set to "envelope").

References

Scholbeck, C. A., Casalicchio, G., Molnar, C., Bischl, B., & Heumann, C. (2022). Marginal Effects for Non-Linear Prediction Functions.

Examples

```
# Train a model:

library(mlr3verse)
library(ranger)
data(bikes, package = "fmeffects")
set.seed(123)
row.id = sample(1:nrow(bikes), 100)
task = as_task_regr(x = bikes, id = "bikes", target = "count")
forest = lrn("regr.ranger")$train(task)

# Compute AMEs for all features:
overview = ame(model = forest, data = bikes[row.id, ], target = "count")
summary(overview)

# Compute AMEs for a subset of features with non-default step.sizes:
overview = ame(model = forest,
               data = bikes[row.id, ],
               target = "count",
               features = c(humidity = 0.1, weather = c("clear", "rain")))
summary(overview)

# Extract results:
overview$results
```

AverageMarginalEffects

R6 Class computing Average Marginal Effects (AME) based on Forward Marginal Effects (FME) for a model

Description

The AME is a simple mean FME and computed w.r.t. a feature variable and a model.

Public fields

`predictor` Predictor object
`features` vector of features for which AMEs should be computed
`ep.method` string specifying extrapolation detection method
`results` data.table with AMEs computed
`computed` logical specifying if `compute()` has been run

Methods**Public methods:**

- [AverageMarginalEffects\\$new\(\)](#)
- [AverageMarginalEffects\\$compute\(\)](#)
- [AverageMarginalEffects\\$clone\(\)](#)

Method new(): Create a new AME object.

Usage:

```
AverageMarginalEffects$new(
  model,
  data,
  target,
  features = NULL,
  ep.method = "none"
)
```

Arguments:

`model` The (trained) model, with the ability to predict on new data. This must be an `Learner` (`mlr3`) or `train` (`caret`) object.

`data` The data used for computing AMEs, must be `data.frame` or `data.table`.

`target` A string specifying the model's target variable.

`features` A named character vector of the names of the feature variables for which AMEs should be computed, together with the desired step sizes.

`ep.method` String specifying the method used for extrapolation detection. One of "none" or "envelope". Defaults to "none".

Returns: A new AME object.

Examples:

```
# Train a model:
```

```
library(mlr3verse)
library(ranger)
set.seed(123)
data(bikes, package = "fmeffects")
row.id = sample(1:nrow(bikes), 100)
task = as_task_regr(x = bikes, id = "bikes", target = "count")
forest = lrn("regr.ranger")$train(task)
```

```
# Compute AMEs for all features:
```

```
overview = AverageMarginalEffects$new(
  model = forest,
  data = bikes[row.id, ],
  target = "count")$compute()
summary(overview)
```

```
# Compute AMEs for a subset of features with non-default step.sizes:
```

```
overview = AverageMarginalEffects$new(model = forest,
```

```

data = bikes[row.id, ],
target = "count",
features = c(humidity = 0.1,
weather = c("clear", "rain"))$compute()

summary(overview)

```

Method `compute()`: Computes results, i.e., AMEs including the SD of FMEs, for an AME object.

Usage:

```
AverageMarginalEffects$compute()
```

Returns: An AME object with results.

Examples:

```
# Compute results:
overview$compute()
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
AverageMarginalEffects$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```

## -----
## Method `AverageMarginalEffects$new`
## -----

# Train a model:

library(mlr3verse)
library(ranger)
set.seed(123)
data(bikes, package = "fmeffects")
row.id = sample(1:nrow(bikes), 100)
task = as_task_regr(x = bikes, id = "bikes", target = "count")
forest = lrn("regr.ranger")$train(task)

# Compute AMEs for all features:
overview = AverageMarginalEffects$new(
  model = forest,
  data = bikes[row.id, ],
  target = "count")$compute()
summary(overview)

# Compute AMEs for a subset of features with non-default step.sizes:
overview = AverageMarginalEffects$new(model = forest,
  data = bikes[row.id, ],
  target = "count",
  features = c(humidity = 0.1,

```

```

summary(overview)                                weather = c("clear", "rain"))$compute()

## -----
## Method `AverageMarginalEffects$compute`
## -----

# Compute results:
overview$compute()

```

bikes

Regression data of the usage of rental bikes in Washington D.C., USA

Description

This data set contains information on hourly bike sharing usage in Washington, D.C. for the years 2011-2012. The target variable is count, the total number of bikes lent out to users at a specific time.

Usage

```
data(bikes)
```

Format

An object of class `data.table` (inherits from `data.frame`) with 727 rows and 11 columns.

Details

This data frame contains the following columns:

```

season Season of the year
year   Year; 0=2011, 1=2012
month  Month of the year
holiday If a day is a public holiday (y/n)
weekday Day of the week
workingday If a day is a working day (y/n)
weather Weather situation
temp   Temperature in degrees celsius
humidity Humidity (relative)
windspeed Windspeed in miles per hour
count  Total number of bikes lent out to users

```

Source

This is a subset of the original data, which can be found on the [OpenML](#) database (ID = 42712).

References

Fanaee-T, Hadi, and Gama, Joao, "Event labeling combining ensemble detectors and background knowledge", *Progress in Artificial Intelligence* (2013): pp. 1-15, Springer Berlin Heidelberg, doi:10.1007/s13748-013-0040-3.

Vanschoren, Joaquin, et al. "OpenML: networked science in machine learning." *ACM SIGKDD Explorations Newsletter* 15.2 (2014): 49-60.

came

Computes a partitioning for a ForwardMarginalEffect

Description

This is a wrapper function that creates the correct subclass of `Partitioning`. It computes feature subspaces for semi-global interpretations of FMEs via recursive partitioning (RP).

Usage

```
came(
  effects,
  number.partitions = NULL,
  max.sd = Inf,
  rp.method = "ctree",
  tree.control = NULL
)
```

Arguments

<code>effects</code>	A <code>ForwardMarginalEffect</code> object with FMEs computed.
<code>number.partitions</code>	The exact number of partitions required. Either <code>number.partitions</code> or <code>max.sd</code> can be specified.
<code>max.sd</code>	The maximum standard deviation required in each partition. Among multiple partitionings with this criterion identified, the one with lowest number of partitions is selected. Either <code>number.partitions</code> or <code>max.sd</code> can be specified.
<code>rp.method</code>	One of "ctree" or "rpart". The RP algorithm used for growing the decision tree. Defaults to "ctree".
<code>tree.control</code>	Control parameters for the RP algorithm. One of " <code>ctree.control(...)</code> " or " <code>rpart.control(...)</code> ". #' @return <code>Partitioning</code> Object with identified feature subspaces.

References

Scholbeck, C. A., Casalicchio, G., Molnar, C., Bischl, B., & Heumann, C. (2022). Marginal Effects for Non-Linear Prediction Functions.

Examples

```
# Train a model and compute FMEs:

library(mlr3verse)
library(ranger)
data(bikes, package = "fmeffects")
task = as_task_regr(x = bikes, id = "bikes", target = "count")
forest = lrn("regr.ranger")$train(task)
effects = fme(model = forest, data = bikes, target = "count", feature = "temp",
              step.size = 1, ep.method = "envelope")

# Find a partitioning with exactly 3 subspaces:
subspaces = came(effects, number.partitions = 3)

# Find a partitioning with a maximum standard deviation of 20, use `rpart`:
library(rpart)
subspaces = came(effects, max.sd = 20, rp.method = "rpart")

# Analyze results:
summary(subspaces)
plot(subspaces)

# Extract results:
subspaces$results
subspaces$tree
```

fme

Computes FMEs.

Description

This is a wrapper function for `FME$new(...)$compute()`. It computes forward marginal effects (FMEs) for a specified change in feature values.

Usage

```
fme(
  model,
  data,
  target,
  feature,
  step.size,
  ep.method = "none",
  compute.nlm = FALSE,
  nlm.intervals = 1
)
```

Arguments

model	The (trained) model, with the ability to predict on new data. This must be an <code>Learner (mlr3)</code> or <code>train (caret)</code> object.
data	The data used for computing FMEs, must be <code>data.frame</code> or <code>data.table</code> .
target	A string specifying the model's target variable.
feature	A character vector of the names of the feature variables affected by the step. For numerical steps, this must have length 1 or 2. For categorical steps, this must have length 1.
step.size	A numeric vector of the step lengths in the features affected by the step. For numerical steps, this must have length 1 or 2. For categorical steps, this is the name of the reference category.
ep.method	String specifying the method used for extrapolation detection. One of "none" or "envelope". Defaults to "none".
compute.nlm	Compute NLMs for FMEs for numerical steps. Defaults to FALSE.
nlm.intervals	Number of intervals for computing NLMs. Results in longer computing time but more accurate approximation of NLMs. Defaults to 1.

Value

FME Object with FMEs computed.

References

Scholbeck, C. A., Casalicchio, G., Molnar, C., Bischl, B., & Heumann, C. (2022). Marginal Effects for Non-Linear Prediction Functions.

Examples

```
# Train a model:

library(mlr3verse)
library(ranger)
data(bikes, package = "fmeffects")
forest = lrn("regr.ranger")$train(as_task_regr(x = bikes, id = "bikes", target = "count"))

# Compute FMEs:
effects = fme(model = forest, data = bikes, target = "count", feature = "temp",
              step.size = 1, ep.method = "envelope")

# Analyze results:
summary(effects)
plot(effects)

# Extract results:
effects$results
```

ForwardMarginalEffect *R6 Class representing a forward marginal effect (FME)*

Description

The FME is a forward difference in prediction due to a specified change in feature values.

Public fields

feature vector of features
 predictor Predictor object
 step.size vector of step sizes for features specified by "feature"
 data.step the data.table with the data matrix after the step
 ep.method string specifying extrapolation detection method
 compute.nlm logical specifying if NLM should be computed
 nlm.intervals number of intervals for computing NLMs
 step.type "numerical" or "categorical"
 extrapolation.ids vector of observation ids classified as extrapolation points
 results data.table with FMEs and NLMs computed
 ame Average Marginal Effect (AME) of observations in results
 anlm Average Non-linearity Measure (ANLM) of observations in results
 computed logical specifying if compute() has been run

Methods

Public methods:

- [ForwardMarginalEffect\\$new\(\)](#)
- [ForwardMarginalEffect\\$compute\(\)](#)
- [ForwardMarginalEffect\\$plot\(\)](#)
- [ForwardMarginalEffect\\$clone\(\)](#)

Method `new()`: Create a new ForwardMarginalEffect object.

Usage:

```
ForwardMarginalEffect$new(
  predictor,
  feature,
  step.size,
  ep.method = "none",
  compute.nlm = FALSE,
  nlm.intervals = 1
)
```

Arguments:

predictor Predictor object.
 feature Feature vector.
 step.size Vector of step sizes.
 ep.method String specifying extrapolation detection method.
 compute.nlm Compute NLM with FMEs? Defaults to FALSE.
 nlm.intervals How many intervals for NLM computation. Defaults to 1.

Returns: A new ForwardMarginalEffect object.

Examples:

```

# Train a model:

library(mlr3verse)
library(ranger)
data(bikes, package = "fmeffects")
forest = lrn("regr.ranger")$train(as_task_regr(x = bikes, id = "bikes", target = "count"))

# Create an `ForwardMarginalEffect` object:
effects = ForwardMarginalEffect$new(makePredictor(forest, bikes, "count"),
  feature = c("temp", "humidity"),
  step.size = c(1, 0.01),
  ep.method = "envelope")
  
```

Method `compute()`: Computes results, i.e., FME (and NLMs) for non-extrapolation points, for a ForwardMarginalEffect object.

Usage:

```
ForwardMarginalEffect$compute()
```

Returns: A ForwardMarginalEffect object with results.

Examples:

```

# Compute results:
effects$compute()
  
```

Method `plot()`: Plots results, i.e., FME (and NLMs) for non-extrapolation points, for an FME object.

Usage:

```
ForwardMarginalEffect$plot(with.nlm = FALSE, jitter = c(0, 0))
```

Arguments:

`with.nlm` Plots NLMs if computed, defaults to FALSE.

`jitter` Jitters data. A two-dimensional numeric vector, corresponds to "width" and "height". See `?ggplot2::geom_jitter` for details. Not available if `step.type` is categorical. Defaults to no jittering, i.e., `c(0, 0)`.

Examples:

```

# Compute results:
effects$plot()
  
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
ForwardMarginalEffect$new(clone(deep = FALSE))
```

Arguments:

```
deep Whether to make a deep clone.
```

Examples

```
## -----
## Method `ForwardMarginalEffect$new`
## -----

# Train a model:

library(mlr3verse)
library(ranger)
data(bikes, package = "fmeffects")
forest = lrn("regr.ranger")$train(as_task_regr(x = bikes, id = "bikes", target = "count"))

# Create an `ForwardMarginalEffect` object:
effects = ForwardMarginalEffect$new(makePredictor(forest, bikes, "count"),
                                   feature = c("temp", "humidity"),
                                   step.size = c(1, 0.01),
                                   ep.method = "envelope")

## -----
## Method `ForwardMarginalEffect$compute`
## -----

# Compute results:
effects$compute()

## -----
## Method `ForwardMarginalEffect$plot`
## -----

# Compute results:
effects$plot()
```

makePredictor

User-friendly function to create a [Predictor](#).

Description

A wrapper function that creates the correct subclass of Predictor by automatically from model. Can be passed to the constructor of FME.

Usage

```
makePredictor(model, data, target)
```

Arguments

```
model      the (trained) model, with the ability to predict on new data.
data       the data used for computing FMEs, must be data.frame or data.table.
target     a string specifying the target variable.
```

Examples

```
# Train a model:

library(mlr3verse)
data(bikes, package = "fmeffects")
task = as_task_regr(x = bikes, id = "bikes", target = "count")
forest = lrn("regr.ranger")$train(task)

# Create the predictor:
predictor = makePredictor(forest, bikes, "count")

# This instantiated an object of the correct subclass of `Predictor`:
class(predictor)
```

 Partitioning

R6 Class representing a partitioning

Description

This is the abstract superclass for partitioning objects like [PartitioningCtree](#) and [PartitioningRpart](#). A Partitioning contains information about feature subspaces with conditional average marginal effects (cAME) computed for ForwardMarginalEffect objects.

Public fields

```
object  a ForwardMarginalEffect object with results computed
method  the method for finding feature subspaces
value   the value of method
results descriptive statistics of the resulting feature subspaces
tree    the tree representing the partitioning, a party object
tree.control control parameters for the RP algorithm
computed logical specifying if compute() has been run
```

Methods**Public methods:**

- `Partitioning$new()`
- `Partitioning$compute()`
- `Partitioning$plot()`
- `Partitioning$clone()`

Method `new()`: Create a Partitioning object

Usage:

```
Partitioning$new(...)
```

Arguments:

... Partitioning cannot be initialized, only its subclasses

Method `compute()`: Computes the partitioning, i.e., feature subspaces with more homogeneous FMEs, for a ForwardMarginalEffect object.

Usage:

```
Partitioning$compute()
```

Returns: An Partitioning object with results.

Examples:

```
# Compute results for an arbitrary partitioning:
# subspaces$compute()
```

Method `plot()`: Plots results, i.e., a decision tree and summary statistics of the feature subspaces, for an Partitioning object after `$compute()` has been run.

Usage:

```
Partitioning$plot()
```

Examples:

```
# Plot an arbitrary partitioning:
# subspaces$plot()
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
Partitioning$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
## -----
## Method `Partitioning$compute`
## -----

# Compute results for an arbitrary partitioning:
```

```
# subspaces$compute()

## -----
## Method `Partitioning$plot`
## -----

# Plot an arbitrary partitioning:
# subspaces$plot()
```

PartitioningCtree *PartitioningCtree*

Description

This task specializes [Partitioning](#) for the ctree algorithm for recursive partitioning.

It is recommended to use [came\(\)](#) for construction of Partitioning objects.

Super class

[fmeffects::Partitioning](#) -> PartitioningCtree

Methods

Public methods:

- [PartitioningCtree\\$new\(\)](#)
- [PartitioningCtree\\$clone\(\)](#)

Method [new\(\)](#): Create a new PartitioningCtree object.

Usage:

```
PartitioningCtree$new(object, method, value, tree.control = NULL)
```

Arguments:

`object` an FME object with results computed.

`method` the method for finding feature subspaces.

`value` the value of method.

`tree.control` control parameters for the RP algorithm.

Method [clone\(\)](#): The objects of this class are cloneable with this method.

Usage:

```
PartitioningCtree$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

PartitioningRpart	<i>PartitioningRpart</i>
-------------------	--------------------------

Description

This task specializes [Partitioning](#) for the rpart algorithm for recursive partitioning.

It is recommended to use [came\(\)](#) for construction of Partitioning objects.

Super class

`fmeffects::Partitioning` -> PartitioningRpart

Methods

Public methods:

- [PartitioningRpart\\$new\(\)](#)
- [PartitioningRpart\\$clone\(\)](#)

Method `new()`: Create a new PartitioningRpart object.

Usage:

```
PartitioningRpart$new(object, method, value, tree.control = NULL)
```

Arguments:

`object` An FME object with results computed.

`method` The method for finding feature subspaces.

`value` The value of method.

`tree.control` Control parameters for the RP algorithm.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
PartitioningRpart$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

plot.ForwardMarginalEffect
Plots an ForwardMarginalEffect object.

Description

Plots an ForwardMarginalEffect object.

Usage

```
## S3 method for class 'ForwardMarginalEffect'  
plot(x, ...)
```

Arguments

x object of class ForwardMarginalEffect.
... additional arguments affecting the summary produced.

plot.Partitioning *Plots an FME Partitioning.*

Description

Plots an FME Partitioning.

Usage

```
## S3 method for class 'Partitioning'  
plot(x, ...)
```

Arguments

x object of class Partitioning.
... additional arguments affecting the summary produced.

Predictor	<i>R6 Class representing a predictor</i>
-----------	--

Description

This is the abstract superclass for predictor objects like [PredictorMLR3](#) and [PredictorCaret](#). A Predictor contains information about an ML model's prediction function and training data.

Public fields

`model` The (trained) model, with the ability to predict on new data.

`X` A `data.table` with feature and target variables.

`feature.names` A character vector with the names of the features in `X`.

`feature.types` A character vector with the types (numerical or categorical) of the features in `X`.

Methods

Public methods:

- [Predictor\\$new\(\)](#)
- [Predictor\\$clone\(\)](#)

Method `new()`: Create a Predictor object

Usage:

```
Predictor$new(...)
```

Arguments:

... Predictor cannot be initialized, only its subclasses

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
Predictor$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

PredictorCaret

PredictorCaret

Description

This task specializes [Predictor](#) for caret regression models. The model is assumed to be a c("train", "train.formula").

It is recommended to use [makePredictor\(\)](#) for construction of Predictor objects.

Super class

[fmeffects::Predictor](#) -> PredictorCaret

Methods

Public methods:

- [PredictorCaret\\$new\(\)](#)
- [PredictorCaret\\$predict\(\)](#)
- [PredictorCaret\\$clone\(\)](#)

Method `new()`: Create a new PredictorCaret object.

Usage:

```
PredictorCaret$new(model, data, target)
```

Arguments:

`model` train, train.formula object.

`data` The data used for computing FMEs, must be data.frame or data.table.

`target` A string specifying the target variable.

Method `predict()`: Predicts on an observation "newdata".

Usage:

```
PredictorCaret$predict(newdata)
```

Arguments:

`newdata` The feature vector for which the target should be predicted.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
PredictorCaret$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

PredictorMLR3

PredictorMLR3

Description

This task specializes [Predictor](#) for mlr3 models. The model is assumed to be a `LearnerRegr` or `LearnerClassif`.

It is recommended to use [makePredictor\(\)](#) for construction of Predictor objects.

Super class

`fmeffects::Predictor` -> PredictorMLR3

Methods

Public methods:

- [PredictorMLR3\\$new\(\)](#)
- [PredictorMLR3\\$predict\(\)](#)
- [PredictorMLR3\\$clone\(\)](#)

Method `new()`: Create a new PredictorMLR3 object.

Usage:

```
PredictorMLR3$new(model, data, target)
```

Arguments:

`model` `LearnerRegr` or `LearnerClassif` object.

`data` The data used for computing FMEs, must be `data.frame` or `data.table`.

`target` A string specifying the target variable.

Method `predict()`: Predicts on an observation "newdata".

Usage:

```
PredictorMLR3$predict(newdata)
```

Arguments:

`newdata` The feature vector for which the target should be predicted.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
PredictorMLR3$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

PredictorParsnip *PredictorParsnip*

Description

This task specializes [Predictor](#) for parsnip models. The model is assumed to be a `model_fit` object.

It is recommended to use [makePredictor\(\)](#) for construction of Predictor objects.

Super class

`fmeffects::Predictor` -> PredictorParsnip

Methods

Public methods:

- [PredictorParsnip\\$new\(\)](#)
- [PredictorParsnip\\$predict\(\)](#)
- [PredictorParsnip\\$clone\(\)](#)

Method `new()`: Create a new PredictorParsnip object.

Usage:

```
PredictorParsnip$new(model, data, target)
```

Arguments:

`model` `model_fit` object.

`data` The data used for computing FMEs, must be `data.frame` or `data.table`.

`target` A string specifying the target variable.

Method `predict()`: Predicts on an observation "newdata".

Usage:

```
PredictorParsnip$predict(newdata)
```

Arguments:

`newdata` The feature vector for which the target should be predicted.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
PredictorParsnip$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

```
print.ForwardMarginalEffect
```

Prints an ForwardMarginalEffect object.

Description

Prints an ForwardMarginalEffect object.

Usage

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

Arguments

x object of class ForwardMarginalEffect.
... additional arguments affecting the summary produced.

```
print.Partitioning      Prints an FME Partitioning.
```

Description

Prints an FME Partitioning.

Usage

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

Arguments

x object of class Partitioning.
... additional arguments affecting the summary produced.

```
summary.AverageMarginalEffects
```

Prints summary of an AverageMarginalEffects object.

Description

Prints summary of an AverageMarginalEffects object.

Usage

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

Arguments

object	object of class AverageMarginalEffects.
...	additional arguments affecting the summary produced.

```
summary.ForwardMarginalEffect
```

Prints summary of an ForwardMarginalEffect object.

Description

Prints summary of an ForwardMarginalEffect object.

Usage

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

Arguments

object	object of class ForwardMarginalEffect.
...	additional arguments affecting the summary produced.

summary.Partitioning *Prints summary of an FME Partitioning.*

Description

Prints summary of an FME Partitioning.

Usage

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

Arguments

object	object of class Partitioning.
...	additional arguments affecting the summary produced.

Index

* datasets

bikes, 7

ame, 3

AverageMarginalEffects, 4

bikes, 7

came, 8

came(), 16, 17

fme, 9

fmeffects (fmeffects-package), 2

fmeffects-package, 2

fmeffects::Partitioning, 16, 17

fmeffects::Predictor, 20–22

ForwardMarginalEffect, 11

makePredictor, 13

makePredictor(), 20–22

Partitioning, 14, 16, 17

PartitioningCtree, 14, 16

PartitioningRpart, 14, 17

plot.ForwardMarginalEffect, 18

plot.Partitioning, 18

Predictor, 13, 19, 20–22

PredictorCaret, 19, 20

PredictorMLR3, 19, 21

PredictorParsnip, 22

print.ForwardMarginalEffect, 23

print.Partitioning, 23

summary.AverageMarginalEffects, 24

summary.ForwardMarginalEffect, 24

summary.Partitioning, 25