Package ‘fmeffects’

June 8, 2024

Title  Model-Agnostic Interpretations with Forward Marginal Effects

Version  0.1.3


License  LGPL-3

Encoding  UTF-8

RoxygenNote  7.2.3

Suggests  caret, furrr, future, hexbin, knitr, mlr3verse, parallelly, ranger, rmarkdown, rpart, tidymodels

Imports  checkmate, cli, 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' 'PredictorLM.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>
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>
ame

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, features = NULL, ep.method = "none")

Arguments

model
	The (trained) model, with the ability to predict on new data. This must be a
train.formula (tidymodels), Learner (mlr3), train (caret), lm or glm object.

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

features
	If not NULL, a named list of the names of the feature variables for which AMEs
should be computed, together with the desired step sizes. For numeric features,
the step size must be a single number. For categorial features, the step size must
be a character vector of category names that is a subset of the levels of the factor
variable.

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


Examples

# Train a model:
library(mlr3verse)
library(ranger)
data(bikes, package = "fmeffects")
set.seed(123)
task = as_task_regr(x = bikes, id = "bikes", target = "count")
forest = lrn("regr.ranger")$train(task)

# Compute AMEs for all features:
## Not run:
overview = ame(model = forest, data = bikes)
summary(overview)

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

# Extract results:
overview$results

## End(Not run)
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, features = NULL, ep.method = "none")

Arguments:

- model: The (trained) model, with the ability to predict on new data. This must be a train.formula (tidymodels), Learner (mlr3), train (caret), lm or glm object.
- data: The data used for computing AMEs, must be data.frame or data.table.
- features: If not NULL, a named list of the names of the feature variables for which AMEs should be computed, together with the desired step sizes. For numeric features, the step size must be a single number. For categorial features, the step size must be a character vector of category names that is a subset of the levels of the factor variable.
- 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")
task = as_task_regr(x = bikes, id = "bikes", target = "count")
forest = lrn("regr.ranger")$train(task)

# Compute AMEs for all features:
\dontrun{
overview = AverageMarginalEffects$new(
  model = forest,
  data = bikes)$compute()
summary(overview)
}

# Compute AMEs for a subset of features with non-default step.sizes:
overview = AverageMarginalEffects$new(model = forest,
data = bikes,
features = list(humidity = 0.1,
weather = c("clear", "rain")))$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:
\dontrun{
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
library(mlr3verse)
library(ranger)
set.seed(123)
data(bikes, package = "fmeffects")
task = as_task_regr(x = bikes, id = "bikes", target = "count")
forest = lrn("regr.ranger")$train(task)

# Compute AMEs for all features:
## Not run:
overview = AverageMarginalEffects$new(model = forest, 
data = bikes)$compute()
summary(overview)

# Compute AMEs for a subset of features with non-default step.sizes:
overview = AverageMarginalEffects$new(model = forest, 
data = bikes, 
features = list(humidity = 0.1, 
weather = c("clear", "rain")))$compute()

summary(overview)
## End(Not run)

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

# Compute results:
## Not run:
overview$compute()
## End(Not run)

---

**bikes**

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

**Description**

This data set contains information on daily 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 day.

**Usage**

data(bikes)

**Format**

An object of class `data.frame` with 731 rows and 10 columns.
Details

This data frame contains the following columns:

- **season**: Season of the year
- **year**: Year; 0=2011, 1=2012
- **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

The original data can be found on the UCI database (ID = 275).

References


---

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

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

- **effects**: A ForwardMarginalEffect object with FMEs computed.
- **number.partitions**: The exact number of partitions required. Either `number.partitions` or `max.sd` can be specified.
- **max.sd**: 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 `number.partitions` or `max.sd` can be specified.
- **rp.method**: One of "ctree" or "rpart". The RP algorithm used for growing the decision tree. Defaults to "ctree".
- **tree.control**: Control parameters for the RP algorithm. One of "ctree.control(...)" or "rpart.control(...)".

Value

Partitioning Object with identified feature subspaces.

References


Examples

```r
# 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, features = list("temp" = 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 = 200, 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,
  features,
  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 a `train.formula` (tidymodels), Learner (mlr3), train (caret), lm or glm object.
- `data`: The data used for computing FMEs, must be data.frame or data.table.
- `features`: A named list with the feature name(s) and step size(s). The list names should correspond to the names of the feature variables affected by the step. The list must exclusively contain either numeric or categorical features, but not a combination of both. Numeric features must have a number as step size, categorical features 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.

Details
If one or more numeric features are passed to the `features` argument, FMEs are computed as

\[
FME_{x,h_S} = f(x + h_S, x_{-S}) - f(x)
\]

where \(h_S\) is the step size vector and \(x_{-S}\) the other features. If one or more categorical features are passed to `features`,

\[
FME_{x,c_J} = f(c_J, x_{-J}) - f(x)
\]

where \(c_J\) is the set of selected reference categories in `features` and \(x_{-J}\) the other features.
Value

ForwardsMarginalEffect object with the following fields:

- `ame` average marginal effect (AME).
- `anlm` average non-linearity measure (NLM).
- `extrapolation.ids` observations that have been identified as extrapolation points and not included in the analysis.
- `data.step` a data.table of the feature matrix after the step has been applied.
- `results` a data.table of the individual FMEs (and NLMs, if applicable) for all observations that are not extrapolation points.

References


Examples

```r
# Train a model:
library(mlr3verse)
library(ranger)
data(bikes, package = "fmeffects")
forest = lrn("regr.ranger")$train(as_task_regr(x = bikes, target = "count"))

# Compute FMEs for a numerical feature:
effects = fme(model = forest, data = bikes, features = list("temp" = 1), ep.method = "envelope")

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

# Extract results:
effects$results

# Compute the AME for a categorial feature:
fme(model = forest, data = bikes, features = list("weather" = "rain"))$ame
```

---

**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, features, ep.method = "none", compute.nlm = FALSE, nlm.intervals = 1)`

**Arguments:**

- **predictor**: Predictor object.
- **features**: A named list with the feature name(s) and step size(s).
- **ep.method**: String specifying extrapolation detection method.
- **compute.nlm**: String specifying extrapolation detection method.
- **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, target = "count"))

# Create an `ForwardMarginalEffect` object:
effects = ForwardMarginalEffect$new(makePredictor(forest, bikes),
features = list("temp" = 1, "humidity" = 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, bins = 40, binwidth = NULL)

*Arguments:*
with.nlm  Plots NLMs if computed, defaults to FALSE.
bins  Numeric vector giving number of bins in both vertical and horizontal directions. Applies only to univariate or bivariate numeric effects. See `ggplot2::stat_summary_hex()` for details.
binwidth  Numeric vector giving bin width in both vertical and horizontal directions. Overrides bins if both set. Applies only to univariate or bivariate numeric effects. See `ggplot2::stat_summary_hex()` for details.

*Examples:*
# Compute results:
effects$plot()

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

*Usage:*
ForwardMarginalEffect$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, target = "count"))

# Create an `ForwardMarginalEffect` object:
effects = ForwardMarginalEffect$new(makePredictor(forest, bikes),
    features = list("temp" = 1, "humidity" = 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)
```

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.
```
Examples

# Train a model:

```r
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:

```r
predictor = makePredictor(forest, bikes)
```

# This instantiated an object of the correct subclass of 'Predictor':

```r
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:*

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

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:
depth Whether to make a deep clone.

Examples

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

# Plot an arbitrary partitioning:
# subspaces$plot()
```
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:
deeplWhether to make a deep clone.
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. See the method $plot() in ForwardMarginalEffect() for details.
...
      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.
**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.
- **target** A character vector with the name of the target variable.
- **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.
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)

Arguments:
model train, train.formula object.
data The data used for computing FMEs, must be data.frame or data.table.

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.
Description

This task specializes Predictor for lm and lmlm-type models. The model is assumed to be a lm. It is recommended to use makePredictor() for construction of Predictor objects.

Super class

fmeffects::Predictor -> PredictorLM

Methods

Public methods:

• PredictorLM$new()
• PredictorLM$predict()
• PredictorLM$clone()

Method new(): Create a new PredictorCaret object.

Usage:
PredictorLM$new(model, data)

Arguments:
model train, train.formula object.
data The data used for computing FMEs, must be data.frame or data.table.

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

Usage:
PredictorLM$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:
PredictorLM$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.
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)

Arguments:
model LearnerRegr or LearnerClassif object.
data The data used for computing FMEs, must be data.frame or data.table.

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.
Description

This task specializes \texttt{Predictor} for \texttt{parsnip} models. The model is assumed to be a \texttt{model_fit} object.

It is recommended to use \texttt{makePredictor()} for construction of Predictor objects.

Super class

\texttt{fmeffects::Predictor} -> \texttt{PredictorParsnip}

Methods

Public methods:

- \texttt{PredictorParsnip$new()}
- \texttt{PredictorParsnip$predict()}
- \texttt{PredictorParsnip$clone()}

Method \texttt{new()}: Create a new \texttt{PredictorParsnip} object.

\textit{Usage:}
\begin{verbatim}
PredictorParsnip$new(model, data)
\end{verbatim}

\textit{Arguments:}

- \texttt{model} \texttt{model_fit} object.
- \texttt{data} The data used for computing FMEs, must be data.frame or data.table.

Method \texttt{predict()}: Predicts on an observation "newdata".

\textit{Usage:}
\begin{verbatim}
PredictorParsnip$predict(newdata)
\end{verbatim}

\textit{Arguments:}

- \texttt{newdata} The feature vector for which the target should be predicted.

Method \texttt{clone()}: The objects of this class are cloneable with this method.

\textit{Usage:}
\begin{verbatim}
PredictorParsnip$clone(deep = FALSE)
\end{verbatim}

\textit{Arguments:}

- \texttt{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

```r
## 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, 5
bikes, 7
came, 8
came(), 17, 18
fme, 10
fmeffects (fmeffects-package), 2
fmeffects-package, 2
fmeffects::Partitioning, 17, 18
fmeffects::Predictor, 21–24
ForwardMarginalEffect, 11
ForwardMarginalEffect(), 19
ggplot2::stat_summary_hex(), 13
makePredictor, 14
makePredictor(), 21–24
Partitioning, 15, 17, 18
PartitioningCtree, 15, 17
PartitioningRpart, 15, 18
plot.ForwardMarginalEffect, 19
plot.Partitioning, 19
Predictor, 14, 20, 21–24
PredictorCaret, 20, 21
PredictorLM, 22
PredictorMLR3, 20, 23
PredictorParsnip, 24
print.ForwardMarginalEffect, 25
print.Partitioning, 25
summary.AverageMarginalEffects, 26
summary.ForwardMarginalEffect, 26
summary.Partitioning, 27