Package 'interpret'

December 12, 2019

Title Fit Interpretable Machine Learning Models and Explain Blackbox Machine Learning

Version 0.1.24

Date 2019-12-10

Description Package for training interpretable machine learning models and explaining blackbox systems. Historically, the most interpretable machine learning models were not very accurate, and the most accurate models were not very interpretable. Microsoft Research has developed an algorithm called the Explainable Boosting Machine (EBM) which has both high accuracy and interpretability. EBM uses machine learning techniques like bagging and boosting to breathe new life into traditional GAMs (Generalized Additive Models). This makes them as accurate as random forests and gradient boosted trees, and also enhances their intelligibility and editability. Details on the EBM algorithm can be found in the paper by Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad (2015, <doi:10.1145/2783258.2788613>).

URL https://github.com/interpretml/interpret

BugReports https://github.com/interpretml/interpret/issues

License MIT + file LICENSE

Depends R (>= 3.0.0)

NeedsCompilation yes

SystemRequirements C++11

Author Samuel Jenkins [aut],
Harsha Nori [aut],
Paul Koch [aut],
Rich Caruana [aut, cre],
Microsoft Corporation [cph]

Maintainer Rich Caruana <interpretml@outlook.com>

Repository CRAN

Date/Publication 2019-12-12 09:10:08 UTC

R topics documented:

`ebm_classify` *Build an EBM classification model*

Description

Builds a classification model

Usage

```
ebm_classify(
  X,
  y,
  num_outer_bags = 16,
  validation_size = 0.15,
  max_epochs = 2000,
  num_early_stopping_run_length = 50,
  learning_rate = 0.01,
  max_tree_splits = 2,
  min_instances_for_split = 2,
  random_state = 42
)
```

Arguments

`X`	features
`y`	targets
`num_outer_bags`	number of outer bags
`validation_size`	amount of data to use for validation
`max_epochs`	number of boosting rounds
`num_early_stopping_run_length`	how many rounds without improvement before we quit
`learning_rate`	learning rate
`max_tree_splits`	how many splits allowed

`min_instances_for_split`
: number of instances required for a split

`random_state`
: random seed

Value

Returns an EBM model

Examples

```
feature1 <- c(1,2,1,2,1,2,1,2)
feature2 <- c(1,2,1,2,1,2,1,2)
X <- data.frame(feature1, feature2)
y <- c(0,1,1,0,1,0,1,0)
model <- ebm_classify(X, y)
```

---

ebm_feature *Create ebm_feature*

Description

Creates an ebm_feature

Usage

```
ebm_feature(
  n_bins,
  has_missing,
  feature_type
)
```

Arguments

`n_bins`
: count bins

`has_missing`
: has missing

`feature_type`
: feature type

Value

Returns an S3 ebm_feature class that contains information about a single machine learning feature

Examples

```
feature <- ebm_feature(1, FALSE, "ordinal")
```

ebm_feature_combination *Create ebm_feature_combination*

Description

Creates an ebm_feature_combination

Usage

```
ebm_feature_combination(
  feature_indexes
)
```

Arguments

feature_indexes	indexes of the features that compose the feature_combination

Value

Returns an S3 ebm_feature_combination class that contains information about a combination of ebm_feature objects

Examples

```
feature_combination <- ebm_feature_combination(1)
```

ebm_predict_proba *ebm_predict_proba*

Description

Predicts the probabilities of an EBM

Usage

```
ebm_predict_proba(
  model,
  X
)
```

Arguments

model	the model
X	features

Value

returns the probabilities for the predictions

Examples

```
feature1 <- c(1,2,1,2,1,2,1,2)
feature2 <- c(1,2,1,2,1,2,1,2)
X <- data.frame(feature1, feature2)
y <- c(0,1,1,0,1,0,1,0)
model <- ebm_classify(X, y)

proba <- ebm_predict_proba(model, X)
```

---

get_interaction_score *Get Interaction Score*

Description

Get Interaction Score

Usage

```
get_interaction_score(
  ebm_interaction,
  feature_indexes
)
```

Arguments

ebm_interaction	ebm interaction
feature_indexes	feature indexes

Value

Returns an interaction score that indicates the relative benefit of a proposed set of features as inputs to EBM boosting

Examples

```
interaction_ptr <- initialize_interaction_regression(
  list(ebm_feature(1)),
  c(0), c(0), c(0))

interaction_score <- get_interaction_score(interaction_ptr, 0)
```

`initialize_interaction_classification` *Initializes Classification Interaction*

Description

Initializes Classification Interaction

Usage

```
initialize_interaction_classification(
  n_classes,
  features,
  binned_data,
  targets,
  predictor_scores
)
```

Arguments

`n_classes`	count target classes
`features`	features
`binned_data`	binned data
`targets`	targets
`predictor_scores`	predictor scores

Value

Returns an opaque externalptr object that can be passed to the function get_interaction_score to obtain the interaction score for any given set of features

Examples

```
interaction_ptr <- initialize_interaction_classification(
  2,
  list(ebm_feature(1)),
  c(0), c(0), c(0))
```

`initialize_interaction_regression` *Initializes Regression Interaction*

Description

Initializes Regression Interaction

Usage

```
initialize_interaction_regression(
  features,
  binned_data,
  targets,
  predictor_scores
)
```

Arguments

`features`	features
`binned_data`	binned data
`targets`	targets
`predictor_scores`	predictor scores

Value

Returns an opaque externalptr object that can be passed to the function get_interaction_score to obtain the interaction score for any given set of features

Examples

```
interaction_ptr <- initialize_interaction_regression(
  list(ebm_feature(1)),
  c(0), c(0), c(0))
```

Index