# Package ‘FACT’

March 16, 2023

**Type** Package

**Title** Feature Attributions for ClusTering

**Version** 0.1.0

**Description** We present 'FACT' (Feature Attributions for ClusTering), a framework for unsupervised interpretation methods that can be used with an arbitrary clustering algorithm. The package is capable of re-assigning instances to clusters (algorithm agnostic), preserves the integrity of the data and does not introduce additional models. 'FACT' is inspired by the principles of model-agnostic interpretation in supervised learning. Therefore, some of the methods presented are based on 'iml', a R Package for Interpretable Machine Learning by Christoph Molnar, Giuseppe Casalicchio, and Bernd Bischl (2018) <doi:10.21105/joss.00786>.

**License** LGPL-3

**BugReports** `https://github.com/henrifnk/FACT/issues`

**Imports** checkmate, data.table, ggplot2, gridExtra, prediction, R6, iml

**Encoding** UTF-8

**Suggests** testthat (>= 3.0.0), caret, covr, knitr, mlr3, mlr3cluster, rmarkdown, FuzzyDBScan, factoextra, patchwork, spelling

**Config/testthat/edition** 3

**RoxygenNote** 7.2.2

**Language** en-US

**NeedsCompilation** no

**Author** Henri Funk [aut, cre],
Christian Scholbeck [aut, ctb],
Giuseppe Casalicchio [aut, ctb]

**Maintainer** Henri Funk <Henri.Funk@stat.uni-muenchen.de>

**Repository** CRAN

**Date/Publication** 2023-03-16 11:30:05 UTC

# R topics documented:



---

`ClustPredictor` *Clustering Predictor Object*

---

## Description

A `ClustPredictor` object holds any unsupervised clustering algorithm and the data to be used for analyzing the model. The interpretation methods in the `FACT` package need the clustering algorithm to be wrapped in a `ClustPredictor` object.

## Details

A Cluster Predictor object is a container for the unsupervised prediction model and the data. This ensures that the clustering algorithm can be analyzed in a robust way. The Model inherits from iml::Predictor Object and adjusts this Object to contain unsupervised Methods.

## Super class

`iml::Predictor` -> `ClustPredictor`

## Public fields

`type character(1)`
Either partition for cluster assignments or prob for soft labels. Can be decided by chosen by the user when initializing the object. If `NULL`, it checks the the dimensions of `y`.

`cnames character`
Is `NULL`, if hard labeling is used. If soft labels are used, column names of `y` are being transferred.

## Methods

### Public methods:

- `ClustPredictor$new()`
- `ClustPredictor$clone()`

**Method** `new()`**:** Create a ClustPredictor object

*Usage:*

```
ClustPredictor$new(
  model = NULL,
  data = NULL,
  predict.function = NULL,
  y = NULL,
  batch.size = 1000,
  type = NULL
)
```

*Arguments:*

`model` any
: The trained clustering algorithm. Recommended are models from `mlr3cluster`. For other clustering algorithms predict functions need to be specified.

`data` data.frame
: The data to be used for analyzing the prediction model. Allowed column classes are: numeric, factor, integer, ordered and character

`predict.function` function
: The function to assign newdata. Only needed if `model` is not a model from `mlr3cluster`. The first argument of `predict.fun` has to be the model, the second the `newdata`:

  `function(model, newdata)`

`y` any
: A integer vector representing the assigned clusters or a data.frame representing the soft labels per cluster assigned in columns.

`batch.size` `numeric(1)`
: The maximum number of rows to be input the model for prediction at once. Currently only respected for SMART.

`type` `character(1))`
: This argument is passed to the prediction function of the model. For soft label predictions, use `type="prob"`. For hard label predictions, use `type="partition"`. Consult the documentation or definition of the clustering algorithm you use to find which type options you have.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

## Examples

```
require(factoextra)
require(FuzzyDBScan)
multishapes <- as.data.frame(multishapes[, 1:2])
eps = c(0, 0.2)
pts = c(3, 15)
res <- FuzzyDBScan$new(multishapes, eps, pts)
res$plot("x", "y")
# create hard label predictor
```

```
predict_part = function(model, newdata) model$predict(new_data = newdata, cmatrix = FALSE)$cluster
ClustPredictor$new(res, as.data.frame(multishapes), y = res$clusters,
                        predict.function = predict_part, type = "partition")
# create soft label predictor
predict_prob = function(model, newdata) model$predict(new_data = newdata)
ClustPredictor$new(res, as.data.frame(multishapes), y = res$results,
                               predict.function = predict_prob, type = "prob")
```

---

`create_predict_fun` *Create a generic prediction function*

---

## Description

Create the algorithms prediction function.

## Usage

```
create_predict_fun(model, task, predict.fun = NULL, type = NULL)

## S3 method for class 'Learner'
create_predict_fun(model, task, predict.fun = NULL, type = NULL)
```

## Arguments

| | |
|---|---|
| `model` | any<br>An arbitrary trained clustering algorithm. |
| `task` | `character(1)`<br>Should be clustering in this case. To be extended... |
| `predict.fun` | `function`<br>The function to assign newdata. Only needed if `model` is not a model from `mlr3cluster`. The first argument of `predict.fun` has to be the model, the second the `newdata`:<br>`function(model, newdata)`<br>To be extended for more methods. |
| `type` | `character(1)`<br>For soft label predictions, `type="prob"`. For hard label predictions, `type="partition"`. Consult the documentation or definition of the clustering algorithm you use to find which type options you have. |

## Value

A unified cluster assignment function for either hard or soft labels.

## Methods (by class)

- `create_predict_fun(Learner)`: Create a predict function for algorithms from `mlr3cluster` `evaluate_class` *Evaluate Class*

### Description

Calculation of binary similarity metric based on confusion matrix.

### Usage

```
evaluate_class(actual, predicted, metric = "f1")

calculate_confusion(actual, predicted)
```

### Arguments

| | |
|---|---|
| `actual` | `numeric`<br>initial cluster assignments |
| `predicted` | `numeric`<br>cluster assignments of permuted data |
| `metric` | `character(1)`<br>binary score metric |

### Value

A binary score for each of the clusters and the number of instances.

### Functions

- `calculate_confusion()`: Calculate confusion matrix

`IDEA` *Idea - Isolated Effect on Assignment*

### Description

`IDEA` with a soft label predictor (sIDEA)
tacks changes the soft label of being assigned to each existing cluster throughout a (multidimensional) feature space `IDEA` with a hard label predictor (hIDEA)
tacks changes the soft label of being assigned to each existing cluster throughout a (multidimensional) feature space

## Details

IDEA for soft labeling algorithms (sIDEA) indicates the soft label that an observation $\mathbf{x}$ with replaced values $\tilde{\mathbf{x}}_S$ is assigned to the k-th cluster. IDEA for hard labeling algorithms (hIDEA) indicates the cluster assignment of an observation $\mathbf{x}$ with replaced values $\tilde{\mathbf{x}}_S$.

The global IDEA is denoted by the corresponding data set X:

$$\text{sIDEA}_X(\tilde{\mathbf{x}}_S) = \left( \frac{1}{n} \sum_{i=1}^{n} \text{sIDEA}^{(1)}_{\mathbf{x}^{(i)}}(\tilde{\mathbf{x}}_S), \ldots, \frac{1}{n} \sum_{i=1}^{n} \text{sIDEA}^{(k)}_{\mathbf{x}^{(i)}}(\tilde{\mathbf{x}}_S) \right)$$

where the c-th vector element is the average c-th vector element of local sIDEA functions. The global hIDEA corresponds to:

$$\text{hIDEA}_X(\tilde{\mathbf{x}}_S) = \left( \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_1(\text{hIDEA}_{\mathbf{x}^{(i)}}(\tilde{\mathbf{x}}_S)), \ldots, \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_k(\text{hIDEA}_{\mathbf{x}^{(i)}}(\tilde{\mathbf{x}}_S)) \right)$$

where the c-th vector element is the fraction of hard label reassignments to the c-th cluster.

## Public fields

`predictor` ClustPredictor
: The object (created with `ClustPredictor$new()`) holding the cluster algorithm and the data.

`feature` `(character or list)`
: Features/ feature sets to calculate the effect curves.

`method` `character(1)`
: The IDEA method to be used.

`mg` `DataGenerator`
: A `MarginalGenerator` object to sample and generate the pseudo instances.

`results` `data.table`
: The IDEA results.

`noise.out` any
: Indicator for the noise variable.

## Active bindings

`type` function
: Detect the type in the predictor

## Methods

### Public methods:

- `IDEA$new()`
- `IDEA$plot()`
- `IDEA$plot_globals()`
- `IDEA$clone()`

**Method** `new()`: Create an IDEA object.

*Usage:*

`IDEA$new(predictor, feature, method = "g+l", grid.size = 20L, noise.out = NULL)`

*Arguments:*

`predictor` ClustPredictor
  The object (created with `ClustPredictor$new()`) holding the cluster algorithm and the data.

`feature` `(character or list)`
  For which features do you want importance scores calculated. The default value of `NULL` implies all features. Use a named list of character vectors to define groups of features for which joint importance will be calculated.

`method` `character(1)`
  The `IDEA` method to be used. Possible choices for the method are:
  `"g+l"` (default): store global and local `IDEA` results
  `"local"`: store only local `IDEA` results
  `"global"`: store only global `IDEA` results
  `"init_local"`: store only local `IDEA` results and additional reference for the observations initial assigned cluster.
  `"init_g+l"` store global and local `IDEA` results and additional reference for the observations initial assigned cluster.

`grid.size` `(numeric(1) or NULL)`
  size of the grid to replace values. If grid size is given, an equidistant grid is create. If `NULL`, values are calculated at all present combinations of feature values.

`noise.out` `any`
  Indicator for the noise variable. If not NULL, noise will be excluded from the effect estimation.

*Returns:* (data.frame)
Values for the effect curves:
One row per grid per instance for each local idea estimation. If `method` includes global estimation, one additional row per grid point.

**Method** `plot()`: Plot an IDEA object.

*Usage:*

`IDEA$plot(c = NULL)`

*Arguments:*

`c` indicator for the cluster to plot. If `NULL`, all clusters are plotted.

*Returns:* (ggplot)
A ggplot object that depends on the `method` chosen.

**Method** `plot_globals()`: Plot the global sIDEA curves of all clusters.

*Usage:*

`IDEA$plot_globals(mass = NULL)`

*Arguments:*

`mass` between 0 and 1. The percentage of local `IDEA` curves to plot a certainty interval.

*Returns:* (ggplot)
A ggplot object.

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

## See Also

iml::FeatureEffects, iml::FeatureEffects

## Examples

```
# load data and packages
require(factoextra)
require(FuzzyDBScan)
multishapes = as.data.frame(multishapes[, 1:2])
# Set up an train FuzzyDBScan
eps = c(0, 0.2)
pts = c(3, 15)
res = FuzzyDBScan$new(multishapes, eps, pts)
res$plot("x", "y")
# create soft label predictor
predict_prob = function(model, newdata) model$predict(new_data = newdata)
predictor = ClustPredictor$new(res, as.data.frame(multishapes), y = res$results,
                                    predict.function = predict_prob, type = "prob")
# Calculate `IDEA` global and local for feature "x"
idea_x = IDEA$new(predictor = predictor, feature = "x", grid.size = 5)
idea_x$plot_globals(0.5) # plot global effect of all clusters with 50 percent of local mass.
```

---

SMART SMART - *Scoring Metric after Permutation*

---

## Description

`SMART` estimates the importance of a feature to the clustering algorithm by measuring changes in cluster assignments by scoring functions after permuting selected feature. Cluster-specific `SMART` indicates the importance of specific clusters versus the remaining ones, measured by a binary scoring metric. Global `SMART` assigns importance scores across all clusters, measured by a multi-class scoring metric. Currently, `SMART` can only be used for hard label predictors.

## Details

Let $M \in \mathbb{N}_0^{k \times k}$ denote the multi-cluster confusion matrix and $M_c \in \mathbb{N}_0^{2 \times 2}$ the binary confusion matrix for cluster c versus the remaining clusters. `SMART` for feature set S corresponds to:

$$\text{Multi-cluster scoring:} \quad \text{SMART}(X, \tilde{X}_S) = h_{\text{multi}}(M) \text{Binary scoring:} \quad \text{SMART}(X, \tilde{X}_S) = \text{AVE}(h_{\text{binary}}(M_1), \ldots, h_{\text{binary}}($$

where AVE averages a vector of binary scores, e.g., via micro or macro averaging. In order to reduce variance in the estimate from shuffling the data, one can shuffle t times and evaluate the distribution of scores. Let $\tilde{X}_S^{(t)}$ denote the t-th shuffling iteration for feature set S. The `SMART` point estimate is given by:

$$\overline{\text{SMART}}(X, \tilde{X}_S) = \psi \left( \text{SMART}(X, \tilde{X}_S^{(1)}), \ldots, \text{SMART}(X, \tilde{X}_S^{(t)}) \right)$$

where $\psi$ extracts a sample statistic such as the mean or median or quantile.

## Public fields

`avg` (`character(1)` or `NULL`)
: `NULL` is calculating cluster-specific (binary) metrics. `"micro"` summarizes binary scores to a global score that treats each instance in the data set with equal importance. `"macro"` summarizes binary scores to a global score that treats each cluster with equal importance.

`metric character(1)`
: The binary similarity metric used.

`predictor` ClustPredictor
: The object (created with `ClustPredictor$new()`) holding the cluster algorithm and the data.

`data.sample` data.frame
: The data, including features and cluster soft/ hard labels.

`sampler` any
: Sampler from the `predictor` object.

`features (character or list)`
: Features/ feature sets to calculate importance scores.

`n.repetitions (numeric(1))`
: How often is the shuffling of the feature repeated?

`results (data.table)`
: A data.table containing the results from `SMART` procedure.

## Methods

### Public methods:

- `SMART$new()`
- `SMART$print()`
- `SMART$plot()`
- `SMART$clone()`

**Method** `new()`**:** Create a SMART object

*Usage:*

```
SMART$new(
  predictor,
  features = NULL,
  metric = "f1",
  avg = NULL,
  n.repetitions = 5
)
```

*Arguments:*

`predictor` ClustPredictor
: The object (created with `ClustPredictor$new()`) holding the cluster algorithm and the data.

`features (character or list)`
: For which features do you want importance scores calculated. The default value of `NULL` implies all features. Use a named list of character vectors to define groups of features for which joint importance will be calculated.

`metric character(1)`
: The binary similarity metric used. Defaults to `f1`, where F1 Score is used. Other possible binary scores are `"precision"`, `"recall"`, `"jaccard"`, `"folkes_mallows"` and `"accuracy"`.

`avg (character(1)` or `NULL)`
: Either `NULL`, `"micro"` or `"macro"`. Defaults to `NULL` is calculating cluster-specific (binary) metrics. `"micro"` summarizes binary scores to a global score that treats each instance in the data set with equal importance. `"macro"` summarizes binary scores to a global score that treats each cluster with equal importance. For unbalanced clusters, `"macro"` is more recommendable.

`n.repetitions (numeric(1))`
: How often should the shuffling of the feature be repeated? The higher the number of repetitions the more stable and accurate the results become.

*Returns:* (data.frame)
data.frame with the results of the feature importance computation. One row per feature with the following columns: For global scores:

- importance.05 (5% quantile of importance values from the repetitions)
- importance (median importance)
- importance.95 (95% quantile) and the permutation.error (median error over all repetitions). For cluster specific scores each column indicates for a different cluster.

**Method** `print()`**:** Print a `SMART` object

*Usage:*

```
SMART$print()
```

*Returns:* `character`
Information about `predictor`, `data`, `metric`, and `avg` and head of the `results`.

**Method** `plot()`**:** plots the similarity score results of a `SMART` object.

*Usage:*

```
SMART$plot(log = FALSE, single_cl = NULL)
```

*Arguments:*

`log` `logical(1)`
Indicator weather results should be logged. This can be useful to distinguish the importance if similarity scores are all close to 1.

`single_cl` `character(1)`
Only used for cluster-specific scores (`avg = NULL`). Should match one of the cluster names. In this case, importance scores for a single cluster are plotted.

*Details:* The plot shows the similarity per feature. For global scores: When `n.repetitions` in `SMART$new` was larger than 1, then we get multiple similarity estimates per feature. The similarity are aggregated and the plot shows the median similarity per feature (as dots) and also the 90%-quantile, which helps to understand how much variance the computation has per feature. For cluster-specific scores: Stacks the similarity estimates of all clusters per feature. Can be used to achieve a global estimate as a sum of cluster-wise similarities.

*Returns:* ggplot2 plot object

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

*Usage:*

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

*Arguments:*

`deep` Whether to make a deep clone.

## See Also

iml::FeatureImp

SMART

SMART

## Examples

```
# load data and packages
require(factoextra)
require(FuzzyDBScan)
multishapes = as.data.frame(multishapes[, 1:2])
# Set up an train FuzzyDBScan
eps = c(0, 0.2)
pts = c(3, 15)
res = FuzzyDBScan$new(multishapes, eps, pts)
res$plot("x", "y")
# create hard label predictor
predict_part = function(model, newdata) model$predict(new_data = newdata, cmatrix = FALSE)$cluster
predictor = ClustPredictor$new(res, as.data.frame(multishapes), y = res$clusters,
                               predict.function = predict_part, type = "partition")
# Run SMART globally
macro_f1 = SMART$new(predictor, n.repetitions = 50, metric = "f1", avg = "macro")
macro_f1 # print global SMART
macro_f1$plot(log = TRUE) # plot global SMART
# Run cluster specific SMART
classwise_f1 = SMART$new(predictor, n.repetitions = 50, metric = "f1")
```

```
macro_f1 # print regional SMART
macro_f1$plot(log = TRUE) # plot regional SMART
```

# Index