

Package: unifiedml (via r-universe)

May 11, 2026

Type Package

Title Unified Interface for Machine Learning Models

Version 0.3.0

Date 2026-05-05

Maintainer T. Moudiki <thierry.moudiki@gmail.com>

Description Provides a unified R6-based interface for various machine learning models with automatic interface detection, consistent cross-validation, model interpretations via numerical derivatives, and visualization. Supports both regression and classification tasks with any model function that follows R's standard modeling conventions (formula or matrix interface).

License MIT + file LICENSE

URL <https://github.com/Techtonique/unifiedml>

BugReports <https://github.com/Techtonique/unifiedml/issues>

Depends R (>= 3.5.0), doParallel, R6, foreach

Imports Rcpp (>= 1.1.0)

Suggests testthat (>= 3.0.0), knitr, rmarkdown, glmnet, randomForest, e1071, covr, spelling, MASS, nnet, caret

VignetteBuilder knitr

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Config/testthat/edition 3

Repository <https://techtonique.r-universe.dev>

Date/Publication 2026-05-11 01:54:06 UTC

RemoteUrl <https://github.com/Techtonique/unifiedml>

RemoteRef HEAD

RemoteSha eb5faa3666048090e2ea582b9b1b9b2d009bf784

Contents

unifiedml-package	2
benchmark	2
cross_val_score	4
extract_probabilities	6
formula_to_matrix	7
matrix_to_formula	8
Model	9
train_test_split	12

Index	13
--------------	-----------

unifiedml-package	<i>Unified Interface for Machine Learning Models</i>
-------------------	------------------------------------------------------

Description

Provides a unified R6-based interface for various machine learning models with automatic interface detection, consistent cross-validation, model interpretations via numerical derivatives, and visualization. Supports both regression and classification tasks with any model function that follows R's standard modeling conventions (formula or matrix interface).

Package Content

Index: This package was not yet installed at build time.

Maintainer

T. Moudiki <thierry.moudiki@gmail.com>

Author(s)

T. Moudiki [aut, cre]

benchmark	<i>Benchmark Multiple Models with Cross-Validation and Model-Specific Parameters</i>
-----------	--------------------------------------------------------------------------------------

Description

Perform k-fold cross-validation on a list of models, using model-specific parameters. Supports verbose messages and a progress bar.

Usage

```
benchmark(
  models,
  X,
  y,
  cv = 5L,
  scoring = NULL,
  params = NULL,
  cl = NULL,
  show_progress = FALSE,
  verbose = TRUE
)
```

Arguments

<code>models</code>	A named list of <code>Model\$new(...)</code> objects to benchmark.
<code>X</code>	A data frame or matrix of predictors.
<code>y</code>	A vector of outcomes (factor for classification, numeric for regression).
<code>cv</code>	Integer, number of cross-validation folds (default 5).
<code>scoring</code>	Scoring metric: "rmse", "mae", "accuracy", or "f1" (default: auto-detected based on task)
<code>params</code>	Optional named list of lists, each sublist containing extra arguments to pass to the corresponding model's <code>fit()</code> call. Names must match models.
<code>cl</code>	Optional number of clusters for parallel processing
<code>show_progress</code>	Logical, whether to show a progress bar (default TRUE).
<code>verbose</code>	Logical, whether to print messages about each model (default TRUE).

Value

A list containing the CV scores for each model.

Examples

```
## Not run:
library(randomForest)

X <- iris[, 1:4]
y <- iris$Species

models <- list(
  glm = Model$new(caret::train),
  rf = Model$new(randomForest::randomForest),
  xgb = Model$new(caret::train)
)

params <- list(
  glm = list(method = "glmnet",
```

```

        tuneGrid = data.frame(alpha = 0, lambda = 0.01),
        trControl = trainControl(method = "none")),
  rf = list(ntree = 150),
  xgb = list(method = "xgbTree",
            tuneGrid = data.frame(nrounds = 150, max_depth = 3, eta = 0.3,
                                  gamma = 0, colsample_bytree = 1,
                                  min_child_weight = 1, subsample = 1),
            trControl = trainControl(method = "none"))
)

results <- benchmark(models, X, y, cv = 5, params = params,
                    show_progress = TRUE, verbose = TRUE)
print(results)

## End(Not run)

```

cross_val_score

Cross-Validation for Model Objects

Description

Perform k-fold cross-validation with consistent scoring metrics across different model types. The scoring metric is automatically selected based on the detected task type.

Usage

```

cross_val_score(
  model,
  X,
  y,
  cv = 5,
  scoring = NULL,
  show_progress = TRUE,
  verbose = TRUE,
  cl = NULL,
  seed = 123,
  fit_params = NULL,
  predict_params = NULL
)

```

Arguments

model	A Model object
X	Feature matrix or data.frame
y	Target vector (type determines regression vs classification)
cv	Number of cross-validation folds (default: 5)

scoring	Scoring metric: "rmse", "mae", "accuracy", "f1", or a custom function with signature <code>function(true, pred)</code> returning a scalar. Default: auto-detected based on task type.
show_progress	Whether to show progress bar (default: TRUE) in sequential mode
verbose	logical flag enabling verbose messages (default: TRUE) in parallel mode
c1	Optional number of clusters for parallel processing. If using <code>c1</code> for parallel execution, custom scoring functions must be self-contained (no dependencies on the calling environment).
seed	Reproducibility seed
fit_params	A list of additional arguments passed to <code>model\$fit()</code>
predict_params	A list of additional arguments passed to <code>model\$predict()</code>

Value

Vector of cross-validation scores for each fold

Examples

```
## Not run:
library(glmnet)
X <- matrix(rnorm(100), ncol = 4)
y <- 2*X[,1] - 1.5*X[,2] + rnorm(25) # numeric -> regression

mod <- Model$new(glmnet::glmnet)
(cv_scores <- cross_val_score(mod, X, y, cv = 5)) # auto-uses RMSE
mean(cv_scores) # Average RMSE

cross_val_score(mod, X, y,
  fit_params = list(alpha = 0, lambda = 0.1),
  predict_params = list(type = "response"))

cross_val_score(mod, X, y,
  fit_params = list(alpha = 0.5, lambda = 0.1),
  predict_params = list(type = "response"))

# Custom scoring: R-squared
r2 <- function(true, pred) {
  ss_res <- sum((true - pred)^2)
  ss_tot <- sum((true - mean(true))^2)
  1 - ss_res / ss_tot
}

(cv_scores4 <- cross_val_score(mod, X, y, cv = 5, scoring = r2))
mean(cv_scores4) # Average R2

# Classification with accuracy scoring
data(iris)
X_class <- iris[, 1:4]
y_class <- iris$Species # factor -> classification
mod2 <- Model$new(e1071::svm)
```

```
(cv_scores2 <- cross_val_score(mod2, X_class, y_class, cv = 5)) # auto-uses accuracy
mean(cv_scores2) # Average accuracy

iris_bin <- iris[iris$Species != "virginica", ]
X_bin <- iris_bin[, 1:4]
y_bin <- droplevels(iris_bin$Species)
(cv_scores3 <- cross_val_score(mod2, X_bin, y_bin, cv = 3,
scoring="f1", fit_params=list(kernel="polynomial")))
mean(cv_scores3) # Average F1

## End(Not run)
```

`extract_probabilities` *Extract probability predictions from any R model in a standardised format*

Description

Extract probability predictions from any R model in a standardised format

Usage

```
extract_probabilities(model, X, y_train = NULL, verbose = FALSE)
```

Arguments

<code>model</code>	Fitted model object (any class)
<code>X</code>	Feature matrix or data.frame for predictions
<code>y_train</code>	Optional training labels used to name output columns
<code>verbose</code>	Print diagnostic information (default: FALSE)

Value

Numeric matrix of shape `n_samples x n_classes` with column names as class labels. Attributes `extraction_method` and `model_class` record how predictions were obtained.

formula_to_matrix	<i>Convert a formula-based model to a matrix interface</i>
-------------------	------------------------------------------------------------

Description

Wraps a model function that expects formula + data so it can be called with a plain X (data.frame or matrix) and y (vector). Factors in X are preserved; special column names are safely backtick-quoted in the generated formula so they survive the formula parser.

Usage

```
formula_to_matrix(fit_func, predict_func = stats::predict)
```

Arguments

fit_func	A model-fitting function whose first two arguments are formula and data (e.g. <code>lm</code> , <code>glm</code>).
predict_func	A prediction function with signature <code>function(model, newdata, ...)</code> . Defaults to <code>stats::predict</code> .

Value

A named list with two elements:

`fit(X, y, weights, ...)` Fits the model. X is a data.frame (or coercible matrix), y is the response vector. Extra arguments are forwarded to `fit_func`.

`predict(model, newdata, ...)` Generates predictions. `newdata` must have the same columns as the X used in `fit`. Extra arguments are forwarded to `predict_func`.

Examples

```
lm_matrix <- formula_to_matrix(lm)
X <- data.frame(wt = mtcars$wt, hp = mtcars$hp, cyl = factor(mtcars$cyl))
y <- mtcars$mpg
model <- lm_matrix$fit(X, y)
lm_matrix$predict(model, X[1:5, ])
```

matrix_to_formula	<i>Convert a matrix-based model to a formula interface</i>
-------------------	------------------------------------------------------------

Description

Wraps a model function that expects a numeric matrix X and a response vector y (like `glmnet::glmnet`) so it can be called with the familiar `formula + data` interface. The formula is expanded via `model.matrix`, which handles factor dummy-coding, interactions, and inline transformations automatically.

Usage

```
matrix_to_formula(
  fit_func,
  predict_func = function(model, newX, ...) stats::predict(model, newdata = newX, ...)
)
```

Arguments

<code>fit_func</code>	A model-fitting function whose first two positional arguments are x (numeric matrix) and y (response vector), e.g. <code>glmnet::glmnet</code> .
<code>predict_func</code>	A prediction function with signature <code>function(model, newX, ...)</code> where <code>newX</code> is a numeric matrix. Defaults to a thin wrapper around <code>stats::predict</code> that passes <code>newdata</code> as <code>newx</code> .

Value

A named list with two elements:

`fit(formula, data, ...)` Fits the model. The formula is expanded with `model.matrix`; the intercept column is dropped before passing to `fit_func` (add it back via `...` if your model needs it). Extra arguments are forwarded to `fit_func`.

`predict(model, newdata, ...)` Generates predictions. `newdata` is expanded with the same `model.matrix` terms captured at fit time. Extra arguments are forwarded to `predict_func`.

Examples

```
## Not run:
glmnet_formula <- matrix_to_formula(
  fit_func = glmnet::glmnet,
  predict_func = function(model, newX, ...) {
    glmnet::predict.glmnet(model, newx = newX, s = 0.01, ...)
  }
)
model <- glmnet_formula$fit(mpg ~ wt + hp + factor(cyl), data = mtcars)
glmnet_formula$predict(model, newdata = mtcars[1:5, ])

## End(Not run)
```

Description

Provides a consistent interface for various machine learning models in R, with automatic detection of formula vs matrix interfaces, built-in cross-validation, model interpretability, and visualization.

An R6 class that provides a unified interface for regression and classification models with automatic interface detection, cross-validation, and interpretability features. The task type (regression vs classification) is automatically detected from the response variable type.

Public fields

`model_fn` The modeling function (e.g., `glmnet::glmnet`, `randomForest::randomForest`)

`fitted` The fitted model object

`task` Type of task: "regression" or "classification" (automatically detected)

`X_train` Training features matrix

`y_train` Training target vector

Methods**Public methods:**

- `Model$new()`
- `Model$fit()`
- `Model$predict()`
- `Model$predict_proba()`
- `Model$print()`
- `Model$summary()`
- `Model$plot()`
- `Model$clone_model()`
- `Model$clone()`

Method `new()`: Initialize a new Model

Usage:

```
Model$new(model_fn)
```

Arguments:

`model_fn` A modeling function (e.g., `glmnet`, `randomForest`, `svm`)

Returns: A new Model object

Method `fit()`: Fit the model to training data

Automatically detects task type (regression vs classification) based on the type of the response variable `y`. Numeric `y` -> regression, factor `y` -> classification.

Usage:

```
Model$fit(X, y, ...)
```

Arguments:

X Feature matrix or data.frame

y Target vector (numeric for regression, factor for classification)

... Additional arguments passed to the model function

Returns: self (invisible) for method chaining

Method predict(): Generate predictions from fitted model

Usage:

```
Model$predict(X, ...)
```

Arguments:

X Feature matrix for prediction

... Additional arguments passed to predict function

Returns: Vector of predictions

Method predict_proba(): Predict probabilities from fitted model

Usage:

```
Model$predict_proba(X)
```

Arguments:

X Feature matrix for prediction

Method print(): Print model information

Usage:

```
Model$print()
```

Returns: self (invisible) for method chaining

Method summary(): Compute numerical derivatives and statistical significance

Uses finite differences to compute approximate partial derivatives for each feature, providing model-agnostic interpretability.

Usage:

```
Model$summary(h = 0.01, alpha = 0.05)
```

Arguments:

h Step size for finite differences (default: 0.01)

alpha Significance level for p-values (default: 0.05)

Details: The method computes numerical derivatives using central differences.

Statistical significance is assessed using t-tests on the derivative estimates across samples.

Returns: A data.frame with derivative statistics (invisible)

Method plot(): Create partial dependence plot for a feature

Visualizes the relationship between a feature and the predicted outcome while holding other features at their mean values.

Usage:

```
Model$plot(feature = 1, n_points = 100)
```

Arguments:

feature Index or name of feature to plot

n_points Number of points for the grid (default: 100)

Returns: self (invisible) for method chaining

Method `clone_model()`: Create a deep copy of the model

Useful for cross-validation and parallel processing where multiple independent model instances are needed.

Usage:

```
Model$clone_model()
```

Returns: A new Model object with same configuration

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

Usage:

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

Arguments:

deep Whether to make a deep clone.

Author(s)

Your Name

Examples

```
# Regression example with glmnet
library(glmnet)
X <- matrix(rnorm(100), ncol = 4)
y <- 2*X[,1] - 1.5*X[,2] + rnorm(25) # numeric -> regression

mod <- Model$new(glmnet::glmnet)
mod$fit(X, y, alpha = 0, lambda = 0.1)
mod$summary()
predictions <- mod$predict(X)

# Classification example
data(iris)
iris_binary <- iris[iris$Species %in% c("setosa", "versicolor"), ]
X_class <- as.matrix(iris_binary[, 1:4])
y_class <- droplevels(iris_binary$Species) # factor -> classification

mod2 <- Model$new(e1071::svm)
mod2$fit(X_class, y_class, kernel = "radial")
predictions <- mod2$predict(X_class)
mod2$predict_proba(X_class)
```

train_test_split *Split data into training and test sets*

Description

Randomly splits a feature matrix or data.frame and its corresponding response vector into training and test subsets.

Usage

```
train_test_split(X, y, test_size = 0.2, seed = NULL)
```

Arguments

X	A matrix or data.frame of features.
y	A vector of responses (numeric or factor). Must have the same number of rows as X.
test_size	Proportion of observations to use as the test set. A number in (0, 1). Default is 0.2 (80/20 split).
seed	An optional integer random seed for reproducibility. If NULL (default) the current RNG state is used.

Value

A named list with four elements:

X_train	Training features (same type as X).
X_test	Test features (same type as X).
y_train	Training response.
y_test	Test response.

Examples

```
# matrix input
X <- iris[, 1:4]
y <- iris$Species
d <- unifiedml::train_test_split(X, y, test_size = 0.3, seed = 42)
dim(d$X_train) # 105 x 4
dim(d$X_test)  # 45 x 4

# data.frame input
d2 <- unifiedml::train_test_split(iris[, 1:4], iris$Species, test_size = 0.2)
is.data.frame(d2$X_train) # TRUE
```

Index

- * **package**
 - unifiedml-package, [2](#)
- benchmark, [2](#)
- cross_val_score, [4](#)
- extract_probabilities, [6](#)
- formula_to_matrix, [7](#)
- matrix_to_formula, [8](#)
- Model, [9](#)
- model.matrix, [8](#)
- train_test_split, [12](#)
- unifiedml (unifiedml-package), [2](#)
- unifiedml-package, [2](#)