Package ‘stressor’

May 1, 2024

Type Package

Title Algorithms for Testing Models under Stress

Version 0.2.0

Description Traditional model evaluation metrics fail to capture model
performance under less than ideal conditions. This package employs
techniques to evaluate models ``under-stress''. This includes testing
models' extrapolation ability, or testing accuracy on specific
sub-samples of the overall model space. Details describing stress-testing
methods in this package are provided in
Haycock (2023) <doi:10.26076/2am5-9f67>. The other primary contribution of
this package is provided to R users access to the 'Python' library 'PyCaret'
<https://pycaret.org/> for quick and easy access to auto-tuned
machine learning models.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.2.3

SystemRequirements python(>=3.8.10)

Suggests knitr, rmarkdown, ggplot2, mlbench, testthat (>= 3.0.0)

Config/testthat/edition 3

Imports reticulate, stats, dplyr

VignetteBuilder knitr

Depends R (>= 3.5)

NeedsCompilation no

Author Sam Haycock [aut, cre],
Brennan Bean [aut],
Utah State University [cph, fnd],
Thermo Fisher Scientific Inc. [fnd]

Maintainer Sam Haycock <haycock.sam@outlook.com>

Repository CRAN

Date/Publication 2024-05-01 04:00:03 UTC

R topics documented:

`boston` *Boston Housing Data*

Description

A subset of data from the Housing data for 506 census tracts of Boston from the 1970 Census. Original data set can be found in the mlbench package.

Usage

```
data(boston)
```

Format

A data.frame with 506 rows and 13 columns:

cmedv corrected median value of owner-occupied homes in USD 1000's

crim per capita crime rate by town

zn proportion of residential land zoned for lots over 25,000 sq.ft

indus proportion of non-retail business acres per town

nox nitric oxides concentration (parts per 10 million)

rm average number of rooms per dwelling

age proportion of owner-occupied units built prior to 1940

dis weighted distances to five Boston employment centres

rad index of accessibility to radial highways

tax full-value property-tax rate per USD 10,000

ptratio pupil-teacher ratio by town

chas Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)

lstat percentage of lower status of the population

Source

mlbench package

`create_groups` *Create Groups for CV*

Description

Create groups for the data by separating them either into 10 fold cross-validation, LOO cross-validation, or k-means grouping.

Usage

```
create_groups(
  formula,
  data,
  n_folds = 10,
  k_mult = NULL,
  repl = FALSE,
  grouping_formula = NULL
)
```

Arguments

`formula` A formula object that specifies the model to be fit.

`data` The data that will be separated into each group.

`n_folds` An integer value defaulted to 10 fold cross-validation. If NULL uses Leave One Out(LOO) instead.

`k_mult` When specified, this is passed onto the cv_cluster to fit the data into k_groups.

`repl` A Boolean value defaulted to ‘FALSE‘, change to ‘TRUE‘ when replicates need to be included in the same group.

`grouping_formula`
A formula object that specifies how the groups will be gathered.

Details

If ‘k_mult‘ is specified as an integer, the formula object will be used to help determine the features specified by the user. This will be passed to the cv_cluster function, which takes a scaled matrix of features.

This function is called by the cv methods as it forms the groups necessary to perform the cross-validation. If you want to use this, it is a nice function that separates the ‘data‘ into groups for training and testing.

Value

A vector of the length equal to number of rows of data.frame from the data argument.

Examples

```
# data generation
lm_data <- data_gen_lm(1000)

# 10 Fold CV group
create_groups(Y ~ ., lm_data)

# Spatial CV
create_groups(Y ~ ., lm_data, n_folds = 10, k_mult = 5)

# LOO CV group
create_groups(Y ~ ., lm_data, n_folds = NULL)
```

`create_virtualenv` *Create ‘Python‘ Virtual Environment*

Description

Allows the user to create a stressor ‘python‘ environment with ‘PyCaret‘ installed in the environment. This function assumes that you have properly installed ‘python‘. We recommend version 3.8.10. It uses existing stressor environments.

Usage

```
create_virtualenv(python = Sys.which("python"), delete_env = FALSE)
```

Arguments

`python`	Defaults to your install of ‘python‘. We prefer version 3.8.10. This is assuming that you installed python from python.org. Currently ‘Anaconda‘ installations of python are not implemented.
`delete_env`	Boolean value to indicate if the environments need to be deleted.

Details

To install ‘python‘, it is recommended using ‘python‘ version 3.8.10 from python.org. This is the same version recommended by ‘PyCaret‘, as it is the most stable. Users have reported troubles using the ‘Anaconda‘ distribution of ‘python‘.

For MacOS and Linux Users note that in order to run this package, ‘LightGBM‘ package on python requires the install of an additional compiler ‘cmake‘ and the ‘libomp‘ (Open Multi-Processing interface). Troubleshoot link from the ‘LightGBM‘documentation here.

Value

A message indicating which environment is being used.

Troubleshoot

If ‘python‘ is not being found properly, trying setting the ‘RETICULATE_PYTHON‘ to blank string. Also ensure that you do not have other ‘python‘ objects in your environment.

Also note that on some instances that a warning message may be displayed as to which version of ‘python‘ is being used.

Examples

```
create_virtualenv()
```

cv *Cross Validation*

Description

This is the core of cross-validation- both standard and using k-mean groups. This method is called by other cv methods of classes.

Usage

```
cv(
  object,
  data,
  n_folds = 10,
  k_mult = NULL,
  repl = FALSE,
  grouping_formula = NULL
)

## S3 method for class 'lm'
cv(
  object,
  data,
  n_folds = 10,
  k_mult = NULL,
  repl = FALSE,
  grouping_formula = NULL
)

## S3 method for class 'mlm_stressor'
cv(
  object,
  data,
  n_folds = 10,
  k_mult = NULL,
  repl = FALSE,
  grouping_formula = NULL
)

## S3 method for class 'reg_asym'
cv(
  object,
  data,
  n_folds = 10,
  k_mult = NULL,
  repl = FALSE,
  grouping_formula = NULL
)

## S3 method for class 'reg_sine'
cv(
  object,
  data,
  n_folds = 10,
  k_mult = NULL,
  repl = FALSE,
  grouping_formula = NULL
)
```

Arguments

`object` One of the four objects that is accepted: mlm_stressor, reg_sine, reg_asym, or lm.

`data` A data.frame object that contains all the entries to be cross-validated on.

`n_folds` An integer value for the number of folds defaulted to 10. If NULL, it will run LOO cross-validation.

`k_mult` Used to specify if k-means clustering is to be used, defaulted to NULL.

`repl` A Boolean value defaulted to ‘FALSE‘, change to ‘TRUE‘ when replicates need to be included in the same group.

`grouping_formula`
A formula object that specifies how the groups will be gathered.

Value

If the object is of class mlm_stressor, then a data.frame will be returned. Otherwise, a vector of the predictions will be returned.

Methods (by class)

- `cv(lm)`: Cross-Validation for lm
- `cv(mlm_stressor)`: Cross-Validation for mlm_stressor
- `cv(reg_asym)`: Cross-Validation for reg_asym
- `cv(reg_sine)`: Cross-Validation for reg_sine

Examples

```
# lm example
lm_test <- data_gen_lm(20)
lm <- lm(Y ~ ., lm_test)
cv(lm, lm_test, n_folds = 2)


lm_test <- data_gen_lm(20)
create_virtualenv()
mlm_lm <- mlm_regressor(Y ~ ., lm_test)
cv(mlm_lm, lm_test, n_folds = 2)

# Asymptotic example
asym_data <- data_gen_asym(10)
asym_fit <- reg_asym(Y ~ ., asym_data)
cv(asym_fit, asym_data, n_folds = 2)

# Sine example
sine_data <- data_gen_sine(10)
sine_fit <- reg_sine(Y ~ ., sine_data)
cv(sine_fit, sine_data, n_folds = 2)
```

cv_cluster *Spatial Cluster-Based Partitions for Cross-Validation*

Description

This function creates cluster-based partitions of a sample space based on k-means clustering. Included in the function are algorithms that attempt to produce clusters of roughly equal size.

Usage

```
cv_cluster(features, k, k_mult = 5, ...)
```

Arguments

features	A scaled matrix of features to be used in the clustering. Scaling usually done with scale and should not include the predictor variable.
k	The number of partitions for k-fold cross-validation.
k_mult	k*k_mult determines the number of subgroups that will be created as part of the balancing algorithm.
...	Additional arguments passed to kmeans as needed.

Details

More information regarding spatial cross-validation can be found in Robin Lovelace's explanation of spatial cross-validation in his textbook.

Value

An integer vector that is number of rows of features with indices of each group.

Examples

```
# Creating a matrix of predictor variables
x_data <- base::scale(data_gen_lm(30)[, -1])
groups <- cv_cluster(x_data, 5, k_mult = 5)
groups
```

cv_core *Cross Validation Function*

Description

This is the machinery to run cross validation. It subsets the test and train set based on the groups it receives.

Usage

```
cv_core(object, data, t_groups, ...)
```

Arguments

`object` Currently ‘"reg_sine", "reg_asym", "lm", "mlm_stressor"‘ objects are accepted.

`data` A data.frame object that has the same formula that was fitted on the data.

`t_groups` The groups for cross validation: standard cross validation, LOO cross_validation, or spatial cross validation.

`...` Additional arguments that are passed to the predict function.

Value

Either a vector of predictions for ‘"reg_sine", "reg_asym", "lm"‘ and a data frame for ‘"mlm_stressor"‘.

Examples

```
# lm example
lm_test <- data_gen_lm(20)
lm <- lm(Y ~ ., lm_test)
cv(lm, lm_test, n_folds = 2)


lm_test <- data_gen_lm(20)
create_virtualenv()
mlm_lm <- mlm_regressor(Y ~ ., lm_test)
cv(mlm_lm, lm_test, n_folds = 2)

# Asymptotic example
asym_data <- data_gen_asym(10)
asym_fit <- reg_asym(Y ~ ., asym_data)
cv(asym_fit, asym_data, n_folds = 2)

# Sine example
sine_data <- data_gen_sine(10)
sine_fit <- reg_sine(Y ~ ., sine_data)
cv(sine_fit, sine_data, n_folds = 2)
```

data_gen_asym *Data Generation Asymptotic*

Description

Creates a synthetic data set for an additive asymptotic model. See the details section for clarification.

Usage

```
data_gen_asym(
  n,
  weight_mat = matrix(rlnorm(10), nrow = 2, ncol = 5),
  y_int = 0,
  resp_sd = 1,
  window = 1e-05,
  ...
)
```

Arguments

n	The number of observations for each parameter.
weight_mat	The parameter coefficients, where each column represents the coefficients and is two rows as each additive equation contains two parameters. Defaulted to be 10 random numbers from the log-normal distribution. The second row of the matrix needs to be positive.
y_int	The y-intercept term of the additive model.
resp_sd	The standard deviation of the epsilon term to be added for noise.
window	Used to determine for any given X variable to get you within distance to capture the asymptotic behavior.
...	Additional arguments that are not currently implemented.

Details

Observations are generated from the following model:

$$y = \sum_{i=1}^{n} -\alpha_i e^{-\beta_i \cdot x_i} + y_{int}$$

Where ‘n‘ is the number of parameters to be used, α_i’s are the scaling parameter and the β_i’s are the weights associated with each x_i. With the y_{int} being where it crosses the y-axis.

Value

A data.frame object with the n rows and the response variable with the number of parameters being equal to the number of columns from the weight matrix.

Examples

```
# Generates 10 observations
asym_data <- data_gen_asym(10)
asym_data
```

data_gen_lm *Data Generation for Linear Regression*

Description

Creates a synthetic data set for an additive linear model. See details for clarification.

Usage

```
data_gen_lm(n, weight_vec = rep(1, 5), y_int = 0, resp_sd = 1, ...)
```

Arguments

n The number of observations for each parameter.

weight_vec The parameter coefficients where each entry represents the coefficients for the additive linear model.

y_int The y-intercept term of the additive model.

resp_sd The standard deviation of the epsilon term to be added for noise.

... Additional arguments that are not currently implemented.

Details

Observations are generated from the following model:

$$y = \sum_{i=1}^{n} \alpha_i \cdot x_i + y_{int}$$

Where ‘n‘ is the number of parameters to be used and the α_i’s are the weights associated with each x_i. With the y_{int} being where it crosses the y-axis.

Value

A data.frame object with the n rows and the response variable with the number of parameters being equal to the number of columns from the weight matrix.

Examples

```
# Generates 10 observations
lm_data <- data_gen_lm(10)
lm_data
```

`data_gen_sine` *Data Generation for Sinusoidal Regression*

Description

Creates a synthetic data set for an additive sinusoidal regression model. See the details section for clarification.

Usage

```
data_gen_sine(
  n,
  weight_mat = matrix(rnorm(15), nrow = 3, ncol = 5),
  y_int = 0,
  resp_sd = 1,
  ...
)
```

Arguments

`n` The number of observations for each parameter.

`weight_mat` The parameter coefficients, where each column represents the coefficients and is three rows as each additive equation contains three parameters. Defaulted to be 15 random numbers from the normal distribution.

`y_int` The y-intercept term of the additive model.

`resp_sd` The standard deviation of the epsilon term to be added for noise.

`...` Additional arguments that are not currently implemented.

Details

Observations are generated from the following model:

$$y = \sum_{i=1}^{n} \alpha_i \; \sin\left(\beta_i(x_i - \gamma_i)))\right) + y_{int}$$

Where ‘n‘ is the number of parameters to be used, α_i’s are the amplitude of each sine wave, β_i’s are the periods for each sine wave and indirectly the weight on each x_i, and the γ_i’s are the phase shift associated with each sine wave. With the y_{int} being where it crosses the y-axis.

Value

A data.frame object with the n rows and the response variable with the number of parameters being equal to the number of columns from the weight matrix.

Examples

```
# Generates 10 observations
sine_data <- data_gen_sine(10)
sine_data
```

dist_cent Distance to Center

Description

Calculates the distance from center of the matrix of predictor variables using a euclidean distance, or the average of all x-dimensions.

Usage

```
dist_cent(formula, data)
```

Arguments

formula	A formula object.
data	A data.frame object.

Details

Formula used to calculate the center point:

$$\bar{x} = \frac{1}{N} \sum_{j=1}^{N} x_{ij}$$

Where $\bar{x}$ is a vector of the center of the x-dimensions, N is the number of rows in the matrix, and x_{ij} is the i, j^{th} entry in the matrix.

Value

A vector of distances from the center.

Examples

```
data <- data_gen_lm(10)
dist <- dist_cent(Y ~ ., data)
dist
```

kappa_class *Kappa function*

Description

A function to calculate the Kappa of binary classification.

Usage

```
kappa_class(confusion_matrix)
```

Arguments

`confusion_matrix`
A matrix or table that is the confusion matrix.

Value

A numeric value representing the kappa value.

mlm_classification *Fit Machine Learning Classification Models*

Description

Through the PyCaret module from ‘python‘, this function fits many machine learning models simultaneously without requiring any ‘python‘ programming on the part of the user. This function is specifically designed for the classification models fitted by ‘PyCaret‘.

Usage

```
mlm_classification(
  formula,
  train_data,
  fit_models = c("ada", "et", "lightgbm", "dummy", "lr", "rf", "ridge", "knn", "dt",
    "gbc", "svm", "lda", "nb", "qda"),
  sort_v = c("Accuracy", "AUC", "Recall", "Precision", "F1", "Kappa", "MCC"),
  n_models = 9999,
  seed = NULL,
  ...
)
```

Arguments

`formula` The classification formula, as a formula object.

`train_data` A data.frame object that includes data to be trained on.

`fit_models` A character vector with all the possible Machine Learning classifiers that are currently being fit, the user may specify a subset of them using a character vector.

ada	AdaBoost Classifier
dt	Decision Tree Classifier
dummy	Dummy Classifier
et	Extra Trees Classifier
gbc	Gradient Boosting Classifier
knn	K Neighbors Classifier
lda	Linear Discriminant Analysis
lightgbm	Light Gradient Boosting Machine
lr	Logistic Regression
nb	Naive Bayes
qda	Quadratic Discriminant Analysis
rf	Random Forest Classifier
ridge	Ridge Classifier
svm	SVM - Linear Kernel

`sort_v` A character vector indicating what to sort the tuned models on.

`n_models` An integer value defaulted to a large integer value to return all possible models.

`seed` An integer value to set the seed of the ‘python‘ environment. Default value is set to ‘NULL‘.

`...` Additional arguments passed onto mlm_init.

Details

‘PyCaret‘ is a ‘python‘ module where machine learning models can be fitted with little coding by the user. The pipeline that ‘PyCaret‘ uses is a setup function to parameterize the data that is easy for all the models to fit on. Then the compare models function is executed, which fits all the models that are currently available. This process takes less than five minutes for data.frame objects that are less than 10,000 rows.

Value

A list object where the first entry is the models fitted and the second is the initial predictive accuracy on the random test data. Returns as two classes ‘"mlm_stressor"‘ and ‘"classifier"‘.

Examples

```
lm_test <- data_gen_lm(20)
binary_response <- sample(c(0, 1), 20, replace = TRUE)
lm_test$Y <- binary_response
```

```
mlm_class <- mlm_classification(Y ~ ., lm_test)
```

mlm_init *Compare Machine Learning Models*

Description

Through the PyCaret module from ‘python‘, this function fits many machine learning models simultaneously without requiring any ‘python‘ programming on the part of the user. This is the core function to fitting the initial models. This function is the backbone to fitting all the models.

Usage

```
mlm_init(
  formula,
  train_data,
  fit_models,
  sort_v = NULL,
  n_models = 9999,
  classification = FALSE,
  seed = NULL,
  ...
)
```

Arguments

`formula` The regression formula or classification formula. This formula should be linear.

`train_data` A data.frame object that includes data to be trained on.

`fit_models` A character vector with all the possible Machine Learning regressors that are currently being fit. The user may specify a subset of them using a character vector.

ada	AdaBoost Regressor
br	Bayesian Ridge
dt	Decision Tree Regressor
dummy	Dummy Regressor
en	Elastic Net
et	Extra Trees Regressor
gbr	Gradient Boosting Regressor
huber	Huber Regressor
knn	K Neighbors Regressor
lar	Least Angle Regression
lasso	Lasso Regression
lightgbm	Light Gradient Boosting Machine
llar	Lasso Least Angle Regression
lr	Linear Regression

omp	Orthogonal Matching Pursuit
par	Passive Aggressive Regressor
rf	Random Forest Regressor
ridge	Ridge Regression

If classification is set to ‘TRUE‘, these models can be used depending on user. These are the default values for classification:

ada	AdaBoost Classifier
dt	Decision Tree Classifier
dummy	Dummy Classifier
et	Extra Trees Classifier
gbc	Gradient Boosting Classifier
knn	K Neighbors Classifier
lda	Linear Discriminant Analysis
lightgbm	Light Gradient Boosting Machine
lr	Logistic Regression
nb	Naive Bayes
qda	Quadratic Discriminant Analysis
rf	Random Forest Classifier
ridge	Ridge Classifier
svm	SVM - Linear Kernel

`sort_v` A character vector indicating what to sort the tuned models on. Default value is ‘NULL‘.

`n_models` A defaulted integer to return the maximum number of models.

`classification` A Boolean value tag to indicate if classification methods should be used.

`seed` An integer value to set the seed of the python environment. Default value is set to ‘NULL‘.

`...` Additional arguments passed to the setup function in ‘PyCaret‘.

Details

The formula should be linear. However, that does not imply a linear fit. The formula is a convenient way to separate predictor variables from explanatory variables.

‘PyCaret‘ is a ‘python‘ module where machine learning models can be fitted with little coding by the user. The pipeline that ‘PyCaret‘ uses has a setup function to parameterize the data that is easy for all the models to fit on. Then compare models function is executed which fits all the models that are currently available. This process takes less than five minutes for data.frame objects that are less than 10,000 rows.

Value

A list object that contains all the fitted models and the CV predictive accuracy. With a class attribute of ‘"mlm_stressor"‘.

Examples

```
lm_test <- data_gen_lm(20)
create_virtualenv()
mlm_lm <- mlm_regressor(Y ~ ., lm_test)
```

mlm_refit *Refit Machine Learning Models*

Description

Refits models fitted in the mlm_init, and returns the predictions.

Usage

```
mlm_refit(mlm_object, train_data, test_data, classification = FALSE)
```

Arguments

`mlm_object` A ‘"mlm_stressor"‘ object.

`train_data` A data.frame object used for refitting excludes the test data. Can be ‘NULL‘ to allow for predictions to be used on the current model.

`test_data` A data.frame object used for predictions.

`classification` A Boolean value used to represent if classification methods need to be used to refit the data.

Value

A matrix with the predictions of the various machine learning methods.

Examples

```
lm_train <- data_gen_lm(20)
train_idx <- sample.int(20, 5)
train <- lm_train[train_idx, ]
test <- lm_train[-train_idx, ]
create_virtualenv()
mlm_lm <- mlm_regressor(Y ~ ., lm_train)
mlm_refit(mlm_lm, train, test, classification = FALSE)
```

mlm_regressor *Fit Machine Learning Regressor Models*

Description

Through the PyCaret module from ‘python‘, this function fits many machine learning models simultaneously with without requiring any ‘python‘ programming on the part of the user. This function is specifically designed for the regression models.

Usage

```
mlm_regressor(
  formula,
  train_data,
 fit_models = c("ada", "et", "lightgbm", "gbr", "lr", "rf", "ridge", "knn", "dt",
   "dummy", "lar", "br", "huber", "omp", "lasso", "en", "llar", "par"),
  sort_v = c("MAE", "MSE", "RMSE", "R2", "RMSLE", "MAPE"),
  n_models = 9999,
  seed = NULL,
  ...
)
```

Arguments

`formula` A linear formula object.

`train_data` A data.frame object that includes data to be trained on.

`fit_models` A character vector with all the possible Machine Learning regressors that are currently being fit. The user may specify a subset of them using a character vector.

ada	AdaBoost Regressor
br	Bayesian Ridge
dt	Decision Tree Regressor
dummy	Dummy Regressor
en	Elastic Net
et	Extra Trees Regressor
gbr	Gradient Boosting Regressor
huber	Huber Regressor
knn	K Neighbors Regressor
lar	Least Angle Regression
lasso	Lasso Regression
lightgbm	Light Gradient Boosting Machine
llar	Lasso Least Angle Regression
lr	Linear Regression
omp	Orthogonal Matching Pursuit
par	Passive Aggressive Regressor
rf	Random Forest Regressor
ridge	Ridge Regression

`sort_v`	A character vector indicating what to sort the tuned models on.
`n_models`	An integer value defaulted to a large integer value to return all possible models.
`seed`	An integer value to set the seed of the ‘python‘ environment. Default value is set to ‘NULL‘.
`...`	Additional arguments passed onto mlm_init.

Details

‘PyCaret‘ is a ‘python‘ module where machine learning models can be fitted with little coding by the user. The pipeline that ‘PyCaret‘ uses is a setup function to parameterize the data that is easy for all the models to fit on. Then the compare models function is executed, which fits all the models that are currently available. This process takes less than five minutes for data.frame objects that are less than 10,000 rows.

Value

A list object where the first entry is the models fitted and the second is the initial predictive accuracy on the random test data. Returns as two classes ‘"mlm_stressor"‘ and ‘"regressor"‘.

Examples

```
 lm_test <- data_gen_lm(20)
 create_virtualenv()
 mlm_lm <- mlm_regressor(Y ~ ., lm_test)
```

`predict` *Prediction Methods for Various Models*

Description

Predict values on ‘mlm_stressor‘, ‘reg_asym‘, or ‘reg_sine‘ objects. This expands the predict function.

Usage

```
## S3 method for class 'mlm_stressor'
predict(object, newdata, train_data = NULL, ...)

## S3 method for class 'reg_asym'
predict(object, newdata, ...)

## S3 method for class 'reg_sine'
predict(object, newdata, ...)
```

Arguments

`object` A ‘mlm_stressor‘, ‘reg_asym‘, or ‘reg_sine‘ object.

`newdata` A data.frame object that is the data to be predicted on.

`train_data` A data.frame object defaulted to ‘NULL‘. This is only used when an ‘mlm_stressor‘ object needs to be refitted.

`...` Extending the predict function default. In this case, it is ignored.

Value

A data.frame of predictions if ‘mlm_stressor‘ object or vector of predicted values.

Examples

```
# mlm_stressor example
lm_test <- data_gen_lm(20)
create_virtualenv()
mlm_lm <- mlm_regressor(Y ~ ., lm_test)
predict(mlm_lm, lm_test)

# Asymptotic Examples
asym_data <- data_gen_asym(10)
asym_fit <- reg_asym(Y ~ ., asym_data)
predict(asym_fit, asym_data)
# Sinusoidal Examples
sine_data <- data_gen_sine(10)
sine_fit <- reg_sine(Y ~ ., sine_data)
predict(sine_fit, sine_data)
```

`python_avail` *Check if ‘Python‘ is Available*

Description

A function that allows examples to run when appropriate.

Usage

```
python_avail()
```

Value

A Boolean value is returned.

Examples

```
python_avail()
```

reg_asym *Asymptotic Regression*

Description

A simple example of asymptotic regression that is in the form of $y = -e^{-x}$ and is the sum of multiple of these exponential functions with a common intercept term.

Usage

```
reg_asym(
  formula,
  data,
  method = "BFGS",
  init_guess = rep(1, ncol(data) * 2 - 1),
  ...
)
```

Arguments

`formula` A formula object to describe the relationship.

`data` The response and predictor variables.

`method` The method that is passed to the optim function. By default, it is the BFGS method which uses a gradient.

`init_guess` The initial parameter guesses for the optim function. By default, it is all ones.

`...` Additional arguments passed to the optim function.

Value

A "reg_asym" object is returned which contains the results from the optim function that was returned.

Examples

```
asym_data <- data_gen_asym(10)
reg_asym(Y ~ ., asym_data)
```

reg_sine *Sinusoidal Regression*

Description

A simple example of sinusoidal regression that is in the form of $y = asin(b(x - c))$ and is the sum of of multiple of these sine functions with a common intercept term.

Usage

```
reg_sine(
  formula,
  data,
  method = "BFGS",
  init_guess = rep(1, ncol(data) * 3 - 2),
  ...
)
```

Arguments

formula	A formula object to describe the relationship.
data	The response and predictor variables.
method	The method that is passed to the optim function. By default, it is the BFGS method which uses a gradient.
init_guess	The initial parameter guesses for the optim function. By default, it is all ones.
...	Additional arguments passed to the optim function.

Value

A "reg_sine" object is returned which contains the results from the optim function that was returned.

Examples

```
sine_data <- data_gen_sine(10)
reg_sine(Y ~ ., sine_data)
```

`rmse` *Root Mean Squarred Error (RMSE)*

Description

A function to calculate the RMSE.

Usage

```
rmse(predicted, observed)
```

Arguments

`predicted`	A data.frame or vector object that is the same number of rows or length as the length of observed values.
`observed`	A vector of the observed results.

`score` *Score Function for Metrics*

Description

A score function takes the observed and predicted values and returns a vector or data.frame of the various metrics that are reported from ‘PyCaret‘. For regression, the following metrics are available: ‘RMSE‘, ‘MAE‘, ‘MSE‘, ‘R2‘, ‘RMSLE‘, and ‘MAPE‘. For classification, the following metrics are available:‘Accuracy‘, ‘AUC‘, ‘Recall‘, ‘Prec.‘, ‘F1‘, ‘MCC‘, and ‘Kappa‘.

Usage

```
score(observed, predicted, ...)
```

Arguments

`observed`	A vector of the observed results.
`predicted`	A data.frame or vector object that is the same number of rows or length as the length of observed values.
`...`	Arguments passed on to `score_classification`, `score_regression` `metrics` A character vector of the metrics to be fitted. This is defaulted to be the metrics from ‘PyCaret‘.

Value

A matrix with the various metrics reported.

Examples

```
lm_data <- data_gen_lm(100)
indices <- split_data_prob(lm_data, .2)
train <- lm_data[!indices,]
test <- lm_data[indices,]
model <- lm(Y ~ ., train)
pred_lm <- predict(model, test)
score(test$Y, pred_lm)
```

score_classification *Score Function for Binary Classification*

Description

This function takes the observed and predicted values and computes metrics that are found in ‘PyCaret‘ such as: ‘Accuracy‘, ‘AUC‘, ‘Recall‘, ‘Prec.‘, ‘F1‘, ‘MCC‘, and ‘Kappa‘.

Usage

```
score_classification(
  observed,
  predicted,
  metrics = c("Accuracy", "AUC", "Recall", "Prec.", "F1", "MCC", "Kappa")
)
```

Arguments

`observed`	A vector of the observed results.
`predicted`	A data.frame or vector object that is the same number of rows or length as the length of observed values.
`metrics`	A character vector of the metrics to be fitted. This is defaulted to be the metrics from ‘PyCaret‘.

Value

A vector or data.frame of the methods and metrics.

score_regression *Score Function for Regression*

Description

This function takes the observed and predicted values and computes metrics that are found in ‘PyCaret‘ such as: ‘RMSE‘, ‘MAE‘, ‘MSE‘, ‘R2‘, ‘RMSLE‘, and ‘MAPE‘.

Usage

```
score_regression(
  observed,
  predicted,
  metrics = c("RMSE", "MAE", "MSE", "R2", "RMSLE", "MAPE")
)
```

Arguments

`observed`	A vector of the observed results.
`predicted`	A data.frame or vector object that is the same number of rows or length as the length of observed values.
`metrics`	A character vector of the metrics to be fitted. This is defaulted to be the metrics from ‘PyCaret‘.

Value

A vector or data.frame of the methods and metrics.

split_data_prob *Create Train Index Set*

Description

This function takes in a data.frame object and the training size and returns a logical vector indicating which entries to include.

Usage

```
split_data_prob(data, test_prop)
```

Arguments

`data`	A data.frame object used to determine the length of the vector.
`test_prop`	A numeric that is between zero and one that represents the proportion of observations to be included in the test data.

Value

A logical vector is returned that is the same length as the number of rows of the data.

Examples

```
lm_data <- data_gen_lm(10)
indices <- split_data_prob(lm_data, .8)
train <- lm_data[indices, ]
test <- lm_data[!indices, ]
```

thinning *Thinning Algorithm for Models with Predict Function*

Description

Fits various train size and test sizes.

Usage

```
thinning(
  model,
  data,
  max = 0.95,
  min = 0.05,
  iter = 0.05,
  classification = FALSE
)
```

Arguments

model	A model that is currently of class type "reg_sine", "reg_asym", "lm", or "mlm_stressor".
data	A data frame with all the data.
max	A numeric value in (0, 1] and greater than ‘min‘, defaulted to .95.
min	A numeric value in (0, 1) and less than ‘max‘, defaulted to .05.
iter	A numeric value to indicate the step size, defaulted to .05.
classification	A Boolean value defaulted ‘FALSE‘, used for ‘mlm_classification‘.

Value

A list of objects, where the first element is the RMSE values at each iteration and the second element is the predictions.

Examples

```
lm_data <- data_gen_lm(1000)
lm_model <- lm(Y ~ ., lm_data)
thin_results <- thinning(lm_model, lm_data)
```

Index