

Package ‘Rmach’

July 31, 2024

Title Provides machine learning algorithym
Version 2.0.0.0
Description Provides these algorithms: coefficient finder for regression functions
License GPL (==3)
Encoding UTF-8
Roxygen list(markdown = TRUE)
RoxygenNote 7.3.1
Imports stringr

Contents

best_model	1
calcall	3
calcall_var	4
individual_route	5
knn_Rmach	6
knn_Rmach_cross_validation_k	7
knn_Rmach_cross_validation_train	8
poly_model	9
sample_Rmach-class	11
Index	16

best_model	<i>best_model</i>
------------	-------------------

Description

Returns the best input models. The coefficient of the best model can be found with the poly_model function

Usage

```
best_model(
  inpt_datf,
  Degree,
  Coeff_v = NA,
  Powers = NA,
  Mth_symb,
  Numrtr_v = NA
)
```

Arguments

<code>inpt_datf</code>	is the input dataframe, first column for the x values and second column for the y values
<code>Degree</code>	is a vector containing all the degrees. Each degree represents how many coefficients the model has.
<code>Coeff_v</code>	is a list containing the vector containing the coefficients for each model. The first value of each coefficient vector is always the constant, so it is not linked to any math symbol
<code>Powers</code>	is a list containing all the values associated with the math symbols of <code>mth_symb</code> list for each model. Because you can have multiple models in the function, so <code>Powers</code> is separated with the "-" separator between the different powers values for each model like in the examples
<code>Mth_symb</code>	is a list containing the vector of the different math symbols linked to the coefficients from the second value
<code>Numrtr_v</code>	is a list containing the different numerator values for each math symbol for each model, see examples

Examples

```
print(best_model(inpt_datf=data.frame(mtcars$wt, mtcars$mpg), Degree=c(2, 2), Coeff_v=c("1", "x", "x^2", "y", "y^2"), Powers=c(1, 2, 3, 4, 5), Mth_symb=c("x", "y"), Numrtr_v=c(1, 2, 3, 4, 5))
[1] 2

print(best_model(inpt_datf=data.frame(mtcars$wt, mtcars$mpg), Degree=c(2, 2), Coeff_v=c("1", "x", "x^2", "y", "y^2"), Powers=c(1, 2, 3, 4, 5), Mth_symb=c("x", "y"), Numrtr_v=c(1, 2, 3, 4, 5))
[1] 1

print(best_model(inpt_datf=data.frame(mtcars$wt, mtcars$mpg), Degree=c(2, 2), Coeff_v=c("1", "x", "x^2", "y", "y^2"), Powers=c(1, 2, 3, 4, 5), Mth_symb=c("x", "y"), Numrtr_v=c(1, 2, 3, 4, 5))
[1] 1

print(best_model(inpt_datf=data.frame(mtcars$wt, mtcars$mpg), Degree=c(2, 2), Coeff_v=c("1", "x", "x^2", "y", "y^2"), Powers=c(1, 2, 3, 4, 5), Mth_symb=c("x", "y"), Numrtr_v=c(1, 2, 3, 4, 5))
#' [1] 1
```

calcall

*calcall***Description**

Takes a formula as a character as an input and makes the calculation. Accepts also variables, in this case the part of the formula that contains the variable won't be calculated, but the others part will be as usual.

Usage

```
calcall(inpt)
```

Arguments

`inpt` is the input formula as a character

Examples

```
print(calcall(inpt="ze+(yu*((fgf)-(-12+8-Y+4-T+4+97+a))+tt)"))
```

```
[1] "ze+(yu*(fgf-(-4-Y+4-T+101+a))+tt) "
```

```
print(calcall(inpt="ze+(yu*((fgf)-(-12+8-7+3-67+4+97+1))+tt)"))
```

```
[1] "ze+(yu*(fgf-27+tt)) "
```

```
print(calcall(inpt="ze+(yu*((fgf)+(12*3/2+4))+tt)"))
```

```
[1] "ze+(yu*(fgf+22+tt)) "
```

```
print(calcall(inpt="1+3*2+(-2/-3*-3*((fgf)-(--12-6)+2))+5-3*5"))
```

```
[1] "7+(-2*(fgf-4))+20"
```

```
print(calcall(inpt="1+3*2+(-2/-3*-3*((fgf)-(--12-6)+2))+(-log_e_1_e_2+t+2^3)+m-log_e_1_e_2"))
```

```
[1] "7+(-2*(fgf-4))+(-2+t+8)+m+6-m-12+(e_ii-8+log_im_4-67)-4+(y+2) "
```

```
print(calcall("(6+4*(-(4-5))+3/3"))
```

```
[1] "11"
```

```
print(calcall(inpt="1+3*2+(-2/-3*-3*((fgf)-(--12-6)+2))+(-log_e_1_e_2+t+2^3)+m-log_e_1_e_2"))
```

```
[1] "7+(-2*(fgf-4))+(-2+t+8)+m+6-m-16"
```

```
print(calcall(inpt="(log_5_Z-2-6+5)+-6+2"))
```

```
[1] "(log_5_Z-3)-4"
```

```
print(calcall(inpt="m--2+-5"))
```

```
[1] "m-3"
```

```
print(calcall(inpt="(-2-6)+-6+2"))

[1] "-12"

print(calcall(inpt="m-6"))

[1] "m-6"

print(calcall(inpt="--6"))

[1] "6"
```

calcall_var	<i>calcall_var</i>
-------------	--------------------

Description

Does the same thing as calcall function but calculates the formula that have variables. The values of the variables have to be given in a list of vectors, see examples.

Usage

```
calcall_var(inpt, var_name_v, var_val_l)
```

Arguments

inpt	is the input formula, with the variables
var_name_v	is the vector that contains the variables name in the order of apparition in the formula. If the variable appears multiple times in the formula, it has to be specified in this vector, see examples.
var_val_l	is the list containing the vectors containing the values of each variable, for each point you want to calculate. The vectors has to be given in the same order has the variable in var_name_v.

Examples

```
print(calcall_var(inpt="(6+m*-(4-imp))+3/jp", var_name_v=c("m", "imp", "jp"),
  var_val_l=list(
    c(1:6),
    c(3, 4, 2, 5, 6, 1),
    c(6:1)))

[1] "5.5" "6.6" "0.75" "11" "17.5" "-9"

print(calcall_var(inpt="(6+m*-(4-imp))+3/jp+jp", var_name_v=c("m", "imp", "jp", "jp"),
  var_val_l=list(
    c(1:6),
```

```

c(3, 4, 2, 5, 6, 1),
c(6:1)))

[1] "11.5" "11.6" "4.75" "14" "19.5" "-8"

```

individual_route	<i>individual_route</i>
------------------	-------------------------

Description

From a time serie, allow to get the most common route for each individual at a given depth (time - 1). Access the frequency value as an element from the output vector and the value itself (the path) as a name of its element, see examples.

Usage

```
individual_route(inpt_datf, col_target, id_col, untl_last = 2)
```

Arguments

inpt_datf	is the input time serie as a dataframe
col_target	is the column name or number that refers to the value of each individual
id_col	is the column name or number that refers to the individual (ids)
untl_last	is the depth value

Examples

```

datf_test <- data.frame("id" = c(1, 1, 1, 2, 2, 3, 3, 3, 4, 4, 5, 5, 5, 5),
                        "city" = c("A", "C", "B", "B", "A", "C", "A", "C", "A", "C", "A", "C", "B", "B"))

print(individual_route(inpt_datf = datf_test,
                      col_target = "city",
                      id_col = "id",
                      untl_last = 2))

AC CA BA
2 1 2

print(individual_route(inpt_datf = datf_test,
                      col_target = "city",
                      id_col = "id",
                      untl_last = 3))

ACB AC CAC BA BAA
1 2 1 2 1

```

knn_Rmach	<i>knn_Rmach</i>
-----------	------------------

Description

KNN algorith, see example

Usage

```
knn_Rmach(train, test, k, col_vars_train = c(), col_vars_test = c(), class_col)
```

Arguments

<code>train</code>	is a dataframe with the known individual and their variadbles and classification columns
<code>test</code>	is a dataframe with the new individuals with ich e do not know the class, only the variables
<code>k</code>	is the number of neighbours
<code>col_vars_train</code>	is a vector containing the column names or column numbers of the variables in train, if empty all column are considered as a variable apart from the last one that is considered as the classification column
<code>col_vars_test</code>	is a vector containing the column names or column numbers of the variables in test, if empty all column are considered as a variable
<code>class_col</code>	is the column name or column number of the classification column in train

Examples

```
cur_ids <- round(runif(n = 45, min = 1, max = 150))

vec <- knn_Rmach(train = iris[-cur_ids,],
  test = iris[cur_ids, 1:4],
  col_vars_train = c(1:4),
  col_vars_test = c(1:4),
  class_col = 5,
  k = 3
)

sum(vec == iris[cur_ids, 5]) / 45

[1] 0.9555556
```

```
knn_Rmach_cross_validation_k
      knn_Rmach_cross_validation_k
```

Description

Allow to perform knn with cross validation for the optimal value of k neighbours used, see examples and parameters.

Usage

```
knn_Rmach_cross_validation_k(
  inpt_datf,
  train_prop,
  knn_v = c(),
  n_fold = 5,
  col_vars = c(),
  class_col
)
```

Arguments

<code>inpt_datf</code>	is the input dataset as a ddataframe
<code>train_prop</code>	is the training proportion
<code>knn_v</code>	is a vector containing the values of k neighbours to test
<code>n_fold</code>	is the number of fold used for each value of k, the higher this value is, he more accurate the result will be but the higher the amount of time it will takes
<code>col_vars</code>	is a vector containing the column names or numbers of the variables in the input dataframe
<code>class_col</code>	is the column names or number of the variable to predict in the input dataframe

Examples

```
iris[, 5] <- as.character(iris[, 5])
print(knn_Rmach_cross_validation_k(
  inpt_datf = iris,
  col_vars = c(1:4),
  n_fold = 5,
  knn_v = c(3, 5, 7, 9, 11),
  class_col = 5,
  train_prop = 0.7
))

[1] 0.1303704 0.1303704 0.1333333 0.1288889 0.1259259

# here the optimal k value is 7
```

```
knn_Rmach_cross_validation_train
      knn_Rmach_cross_validation_train
```

Description

Allow to perform knn with cross validation for the optimal value of k neighbours used, see examples and parameters.

Usage

```
knn_Rmach_cross_validation_train(
  inpt_datf,
  train_prop_v = c(),
  k,
  n_fold = 5,
  col_vars = c(),
  class_col
)
```

Arguments

<code>inpt_datf</code>	is the input dataset as a ddataframe
<code>n_fold</code>	is the number of fold used for each value of k, the higher this value is, the more accurate the result will be but the higher the amount of time it will take
<code>col_vars</code>	is a vector containing the column names or numbers of the variables in the input dataframe
<code>class_col</code>	is the column names or number of the variable to predict in the input dataframe
<code>train_prop</code>	is the training proportion
<code>knn_v</code>	is a vector containing the values of k neighbours to test

Examples

```
iris[, 5] <- as.character(iris[, 5])
print(knn_Rmach_cross_validation_train(
  inpt_datf = iris,
  col_vars = c(1:4),
  n_fold = 15,
  k = 7,
  class_col = 5,
  train_prop_v = c(0.7, 0.75, 0.8)
))

[1] 0.1175309 0.1000000 0.1088889

# here the optimal training proportion is 0.7
```


poly_model

Rmach poly_model

Description

Take a datasets of x and y values and a function tha could fit all the data with the missing coefficients, and returns a list containing the coefficients that fit the best the data for a given function, as a vector for the first index, and at the second index, the actual sum of difference between each data point and the function at the same x values.

Usage

```
poly_model (
  inpt_datf,
  degree,
  twk_val = NA,
  sensi_val = twk_val,
  coeff_v = NA,
  powers = NA,
  mth_symb = c("x"),
  numrtr_v = NA
)
```

Arguments

inpt_datf	is the input data as a dataframe, first column is the x values and the second is the y values
degree	is how many coefficients will be involved (each coefficient multiplies either an x to the power of something, an exponential of something or a base something logarithm for a something value)
twk_val	is the value used for finding the best coefficients, it is directly linked to the accuracy of the coefficients, see the description for more information. Defaults to $(\max(yval) - \min(yval)) / n$
sensi_val	is the value from which two variations of a coefficient brings a so small accuracy contribution that the algorythm does not continue to find better coefficients. For example, if i set <code>sensi_val = 0.001</code> , so if coefficients <code>alpha1</code> and <code>beta1</code> brings a total difference between the function and the actual data of 10.8073 and then the algorythm find <code>alpha2</code> and <code>beta1</code> that brings a total difference equal to 10.8066, so the algorythm will stop running. But the coefficients returned will still be the best, that is <code>alpha2</code> and <code>beta1</code>
coeff_v	is a vector containing the original coefficients for the function, so the closest those are from the best one, the fastest the algorythm will compute the best coefficients. The first value of <code>coeff</code> is always the constant.
powers	is a vector containing the exponent, or related value to <code>mth_symb</code> . powers can be a vector if those values are constants or it could be a list of vectors the length of observed individuals, if those values varies like in the examples. Notthat if you use variables in powers (list), each values of a vector from this list has to be at the exact same x coordinates of each observed individuals in the input dataframe. Ex: <code>datf <- data.frame("x"=c(4, 4, 3, 2, 1, 1), "y"=c(1:6))</code> , so vector(s) from


```

[1] 27.359375 -8.140625

[[2]]
[1] 160.2263

print(poly_model(inpt_datf=data.frame(mtcars$wt, mtcars$mpg), degree=1, coeff_v=c(32.5), -
                                numrtr_v=NA))

[[1]]
[1] 19.28125

[[2]]
[1] 148.7625

print(poly_model(inpt_datf=data.frame(mtcars$wt, mtcars$mpg), degree=2, coeff_v=c(32.5, -
                                numrtr_v=NA))

[[1]]
[1] 0.921875 -5.203125 2.000000

[[2]]
[1] 455.6017

```

sample_Rmach-class *v_Rmach_fold*

Description

Allow to create uniform sampling dataset for cross validation, train and test, see examples and variables

Arguments

inpt_datf is the input dataframe
train_prop is the training proportion
n_fold is the number of distinct pair of training and test dataset that will be outputted

Examples

```

lst_test <- v_Rmach_fold(inpt_datf = iris[1:25,],
                        train_prop = 0.7,
                        n_fold = 4)

print(lst_test)

$sample1
An object of class "sample_Rmach"
Slot "train":
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species test_status
4           4.6          3.1          1.5          0.2  setosa           0

```

7	4.6	3.4	1.4	0.3	setosa	0
19	5.7	3.8	1.7	0.3	setosa	0
10	4.9	3.1	1.5	0.1	setosa	0
23	4.6	3.6	1.0	0.2	setosa	0
20	5.1	3.8	1.5	0.3	setosa	0
1	5.1	3.5	1.4	0.2	setosa	0
9	4.4	2.9	1.4	0.2	setosa	0
4.1	4.6	3.1	1.5	0.2	setosa	0
10.1	4.9	3.1	1.5	0.1	setosa	0
7.1	4.6	3.4	1.4	0.3	setosa	0
17	5.4	3.9	1.3	0.4	setosa	0
22	5.1	3.7	1.5	0.4	setosa	0
15	5.8	4.0	1.2	0.2	setosa	0
23.1	4.6	3.6	1.0	0.2	setosa	0
17.1	5.4	3.9	1.3	0.4	setosa	0
15.1	5.8	4.0	1.2	0.2	setosa	0
13	4.8	3.0	1.4	0.1	setosa	0

Slot "test":

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	test_status
7	4.6	3.4	1.4	0.3	setosa	1
12	4.8	3.4	1.6	0.2	setosa	1
10	4.9	3.1	1.5	0.1	setosa	1
7.1	4.6	3.4	1.4	0.3	setosa	1
5	5.0	3.6	1.4	0.2	setosa	1
22	5.1	3.7	1.5	0.4	setosa	1
20	5.1	3.8	1.5	0.3	setosa	1

Slot "train_ids":

```
[1] 3 11 19 22 13 22 16 21 2 25 12 24 11 18 8 18 6 10
```

Slot "test_ids":

```
[1] 13 15 4 12 14 7 8
```

\$sample2

An object of class "sample_Rmach"

Slot "train":

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species	test_status
24	5.1	3.3	1.7	0.5	setosa	0
1	5.1	3.5	1.4	0.2	setosa	0
8	5.0	3.4	1.5	0.2	setosa	0
16	5.7	4.4	1.5	0.4	setosa	0
13	4.8	3.0	1.4	0.1	setosa	0
9	4.4	2.9	1.4	0.2	setosa	0
6	5.4	3.9	1.7	0.4	setosa	0
22	5.1	3.7	1.5	0.4	setosa	0
10	4.9	3.1	1.5	0.1	setosa	0
21	5.4	3.4	1.7	0.2	setosa	0
22.1	5.1	3.7	1.5	0.4	setosa	0
16.1	5.7	4.4	1.5	0.4	setosa	0
15	5.8	4.0	1.2	0.2	setosa	0
5	5.0	3.6	1.4	0.2	setosa	0
15.1	5.8	4.0	1.2	0.2	setosa	0
16.2	5.7	4.4	1.5	0.4	setosa	0
8.1	5.0	3.4	1.5	0.2	setosa	0
10.1	4.9	3.1	1.5	0.1	setosa	0

```
Slot "test":
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species test_status
13           4.8         3.0         1.4         0.1  setosa           1
23           4.6         3.6         1.0         0.2  setosa           1
24           5.1         3.3         1.7         0.5  setosa           1
11           5.4         3.7         1.5         0.2  setosa           1
9            4.4         2.9         1.4         0.2  setosa           1
22           5.1         3.7         1.5         0.4  setosa           1
10           4.9         3.1         1.5         0.1  setosa           1
```

```
Slot "train_ids":
[1] 22 12 19 5 18 6 6 1 19 5 13 15 1 12 10 8 9 6
```

```
Slot "test_ids":
[1] 15 9 22 7 15 2 3
```

```
$sample3
```

```
An object of class "sample_Rmach"
```

```
Slot "train":
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species test_status
19           5.7         3.8         1.7         0.3  setosa           0
5            5.0         3.6         1.4         0.2  setosa           0
7            4.6         3.4         1.4         0.3  setosa           0
1            5.1         3.5         1.4         0.2  setosa           0
15           5.8         4.0         1.2         0.2  setosa           0
22           5.1         3.7         1.5         0.4  setosa           0
14           4.3         3.0         1.1         0.1  setosa           0
23           4.6         3.6         1.0         0.2  setosa           0
11           5.4         3.7         1.5         0.2  setosa           0
11.1         5.4         3.7         1.5         0.2  setosa           0
17           5.4         3.9         1.3         0.4  setosa           0
4            4.6         3.1         1.5         0.2  setosa           0
9            4.4         2.9         1.4         0.2  setosa           0
4.1          4.6         3.1         1.5         0.2  setosa           0
4.2          4.6         3.1         1.5         0.2  setosa           0
12           4.8         3.4         1.6         0.2  setosa           0
5.1          5.0         3.6         1.4         0.2  setosa           0
14.1         4.3         3.0         1.1         0.1  setosa           0
```

```
Slot "test":
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species test_status
22           5.1         3.7         1.5         0.4  setosa           1
1            5.1         3.5         1.4         0.2  setosa           1
19           5.7         3.8         1.7         0.3  setosa           1
4            4.6         3.1         1.5         0.2  setosa           1
5            5.0         3.6         1.4         0.2  setosa           1
24           5.1         3.3         1.7         0.5  setosa           1
22.1         5.1         3.7         1.5         0.4  setosa           1
```

```
Slot "train_ids":
[1] 14 5 2 21 6 19 8 4 9 7 5 2 3 19 3 24 10 17
```

```
Slot "test_ids":
[1] 6 11 9 20 15 5 25
```

```

$sample4
An object of class "sample_Rmach"
Slot "train":
      Sepal.Length Sepal.Width Petal.Length Petal.Width Species test_status
15           5.8         4.0         1.2         0.2  setosa           0
23           4.6         3.6         1.0         0.2  setosa           0
24           5.1         3.3         1.7         0.5  setosa           0
22           5.1         3.7         1.5         0.4  setosa           0
20           5.1         3.8         1.5         0.3  setosa           0
8            5.0         3.4         1.5         0.2  setosa           0
21           5.4         3.4         1.7         0.2  setosa           0
14           4.3         3.0         1.1         0.1  setosa           0
3            4.7         3.2         1.3         0.2  setosa           0
6            5.4         3.9         1.7         0.4  setosa           0
9            4.4         2.9         1.4         0.2  setosa           0
12           4.8         3.4         1.6         0.2  setosa           0
2            4.9         3.0         1.4         0.2  setosa           0
22.1         5.1         3.7         1.5         0.4  setosa           0
8.1          5.0         3.4         1.5         0.2  setosa           0
12.1         4.8         3.4         1.6         0.2  setosa           0
18           5.1         3.5         1.4         0.3  setosa           0
15.1         5.8         4.0         1.2         0.2  setosa           0

Slot "test":
      Sepal.Length Sepal.Width Petal.Length Petal.Width Species test_status
16           5.7         4.4         1.5         0.4  setosa           1
14           4.3         3.0         1.1         0.1  setosa           1
8            5.0         3.4         1.5         0.2  setosa           1
9            4.4         2.9         1.4         0.2  setosa           1
20           5.1         3.8         1.5         0.3  setosa           1
11           5.4         3.7         1.5         0.2  setosa           1
18           5.1         3.5         1.4         0.3  setosa           1

Slot "train_ids":
[1] 8 2 7 24 21 17 14 22 14 23 19 8 21 20 17 7 13 6

Slot "test_ids":
[1] 10 12 17 9 3 19 1

print(lst_test$sample4$train) # to display the 4th train sample

```

```

      Sepal.Length Sepal.Width Petal.Length Petal.Width Species test_status
15           5.8         4.0         1.2         0.2  setosa           0
23           4.6         3.6         1.0         0.2  setosa           0
24           5.1         3.3         1.7         0.5  setosa           0
22           5.1         3.7         1.5         0.4  setosa           0
20           5.1         3.8         1.5         0.3  setosa           0
8            5.0         3.4         1.5         0.2  setosa           0
21           5.4         3.4         1.7         0.2  setosa           0
14           4.3         3.0         1.1         0.1  setosa           0
3            4.7         3.2         1.3         0.2  setosa           0
6            5.4         3.9         1.7         0.4  setosa           0
9            4.4         2.9         1.4         0.2  setosa           0
12           4.8         3.4         1.6         0.2  setosa           0
2            4.9         3.0         1.4         0.2  setosa           0
22.1         5.1         3.7         1.5         0.4  setosa           0

```

sample_Rmach-class

15

8.1	5.0	3.4	1.5	0.2	setosa	0
12.1	4.8	3.4	1.6	0.2	setosa	0
18	5.1	3.5	1.4	0.3	setosa	0
15.1	5.8	4.0	1.2	0.2	setosa	0

Index

`best_model`, [1](#)

`calcall`, [3](#)

`calcall_var`, [4](#)

`individual_route`, [5](#)

`knn_Rmach`, [6](#)

`knn_Rmach_cross_validation_k`, [7](#)

`knn_Rmach_cross_validation_train`,
[8](#)

`poly_model`, [9](#)

`sample_Rmach-class`, [11](#)