Package 'Rmach'

June 10, 2024

Title Provides machine learning algorythm

Version 2.0.0.0

Description Provides these algorythms: coefficient finder for regression functions

License GPL (==3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Imports stringr

Contents

`best_model` *best_model*

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

inpt_datf	is the input dataframe, first column for the x values and second column for the y values
Degree	is a vector containing all the degrees. Each degree represents how many coefficients the model has.
Coeff_v	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
Powers	is a list containing all the values associated with the math symbols of mth_symb list for each model. Because you can have multiple models in the function, so Powers is separated with the "-" separator between the different powers values for each model like in the examples
Mth_symb	is a list containing the vector of the different math symbols linked to the coefficients from the second value
Numrtr_v	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] 2

print(best_model(inpt_datf=data.frame(mtcars$wt, mtcars$mpg), Degree=c(2, 2), Coeff_v=c("

[1] 1

print(best_model(inpt_datf=data.frame(mtcars$wt, mtcars$mpg), Degree=c(2, 2), Coeff_v=c("

[1] 1

print(best_model(inpt_datf=data.frame(mtcars$wt, mtcars$mpg), Degree=c(2, 2), Coeff_v=c("

#' [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 wont 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_

[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_

[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` *calcall_var*

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` *common_tracks*

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", "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
```

---

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 sensi_val = 0.001, so if coefficients alpha1 and beta1 brings a total difference between the function and the actual data of 10.8073 and then the algorythm find alpha2 and beta1 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 alpha2 and beta1
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 coeff is always the constant.
powers	is a vector containing the exponent, or related value to mth_symb. 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: datf <- data.frame("x"=c(4, 4, 3, 2, 1, 1), "y"=c(1:6)), so vector(s) from powers that contain varying value must be of length 4. Also, the values are not ascendly sorted, don't worry values are ascendly sorted under the hood, so fill your powers vectors in the intuitive ascendly way
mth_symb	is a vector containing the elemnts linked to the coefficients from the second element. It can be x, e (exp(x)) or log-X (log(x)-base), and their reverse like 1/x. If the numerator varies the element should be entered like tis list/x, list/e or list/log-base. See numrtr_v for the values related to list
numrtr_v	is a vector containing the values for the numerator related to mth_symb if on element is like this: list/x or list/e

Examples

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

                numrtr_v=NA))

[[1]]
[1] 33.234375 -4.265625

[[2]]
[1] 74.78275
```

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

                 numrtr_v=NA))

[[1]]
[1] 31.765625 -3.734375

[[2]]
[1] 80.36228

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

                 numrtr_v=NA))

[[1]]
[1] 32.5 -3.0

[[2]]
[1] 1.067436e+24

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

                 numrtr_v=list(c(length(mtcars$wt):1))))
[[1]]
[1] 19.28125 -0.06250

[[2]]
[1] 35839.44

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

                 numrtr_v=NA))

[[1]]
[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
```

Index