Package 'mactivate'

October 13, 2022

Type Package

Title Multiplicative Activation

Version 0.6.6

Date 2021-08-02

Author Dave Zes

Maintainer Dave Zes <zesdave@gmail.com>

Description Provides methods and classes for adding m-activation (``multiplicative activation'') layers to MLR or multivariate logistic regression models. M-activation layers created in this library detect and add input interaction (polynomial) effects into a predictive model. M-activation can detect high-order interactions -- a traditionally non-trivial challenge. Details concerning application, methodology, and relevant survey literature can be found in this library's vignette, ``About.''

License GPL (>= 3)

Depends R (>= 3.5.0)

NeedsCompilation yes

Repository CRAN

Date/Publication 2021-08-02 16:30:02 UTC

R topics documented:

`mactivate-package` *m-activation*

Description

Provides methods and classes for adding m-activation ("multiplicative activation") layers to MLR or multivariate logistic regression models. M-activation layers created in this library detect and add input interaction (polynomial) effects into a predictive model. M-activation can detect high-order interactions – a traditionally non-trivial challenge. Details concerning application, methodology, and relevant survey literature can be found in this library's vignette, "About."

Details

The DESCRIPTION file:

Package:	mactivate
Type:	Package
Title:	Multiplicative Activation
Version:	0.6.6
Date:	2021-08-02
Author:	Dave Zes
Maintainer:	Dave Zes <zesdave@gmail.com>
Description:	Provides methods and classes for adding m-activation ("multiplicative activation") layers to MLR or multivaria
License:	GPL (>=3)
Depends:	R (>= 3.5.0)

Index of help topics:

```
df_hospitals_ortho      Orthopedic Device Sales
f_control_mactivate     Set Fitting Hyperparameters
f_dmss_dW               Calculate Derivative of Cost Function wrt W
f_fit_gradient_01       Fit Multivariate Regression Model with
                        mactivate Using Gradient Descent
f_fit_gradient_logistic_01
                        Fit Logistic Multivariate Regression Model with
                        mactivate Using Gradient Descent
f_fit_hybrid_01         Fit Multivariate Regression Model with
                        mactivate Using Hybrid Method
f_logit_cost            Logistic Cost
f_mactivate             Map Activation Layer and Inputs to Polynomial
                        Model Inputs
mactivate-package       m-activation
predict.mactivate_fit_gradient_01
                        Predict from Fitted Gradient Model
predict.mactivate_fit_gradient_logistic_01
```

```
                        Predict from Fitted Gradient Logistic Model
predict.mactivate_fit_hybrid_01
                        Predict from Fitted Hybrid Model
```

Please make sure to read Details in f_dmss_dW help page before using this library. This package allows the user to extend the usual multivariate regression solution by adding (parallel) multiplicative "activation layers." These activation layers can be very useful for identifying input interactions, and, if the user wishes, transparently test the appropriateness of input transformations. Three functions are provided for fitting data, f_fit_hybrid_01 and f_fit_gradient_01 for a numeric response (usual MLR), and f_fit_gradient_logistic_01 for a binary response (multivariate logistic regresssion). The user is encouraged to consult the "About" vignette as well as the examples available in the respective functions' documentation for details about m-activation and practical examples of implementation.

Author(s)

Dave Zes

Maintainer: Dave Zes <zesdave@gmail.com>

Examples

```
## please see docs for individual functions.
```

---

df_hospitals_ortho *Orthopedic Device Sales*

Description

Sales data of orthopedic device company to client hospitals over almost 2 years. 15 variables, 4703 hospitals. Unit of observation is a unique hospital.

Usage

```
data(df_hospitals_ortho)
```

Format

Variables are:

zip: 'character': Postal code.

hid: 'character': Hospital ID.

city: 'character': Hospital city.

state: 'character': Hospital state.

tot_sales: 'numeric': Total sales to hospital.

tot_knee: 'numeric': Number of knee operations.

tot_hip: 'numeric': Number of hip operations.

beds: 'numeric': Number of beds.

rehab_beds: 'numeric': Number of beds dedicated for rehabilitation.

outpatient_visits: 'numeric': Number of outpatient visits.

adm_costs: 'numeric': Administrative costs ($1000's / yr).

revenue_inpatient: 'numeric': Inpatient revenue.

is_teaching: 'numeric': Is teaching hospital?

has_trauma: 'numeric': Has trauma center?

has_rehab: 'numeric': Offers rehabilitation?

Details

This data frame has attribute 'modelvars' which gives names of numeric model variables.

Source

Data adapted from 'c84.dat' from Statistical Consulting, Javier Cabrera and Andrew McDougall.

References

Statistical Consulting, Javier Cabrera and Andrew McDougall. Springer, Piscataway, NJ, 2002.

Examples

```
data(df_hospitals_ortho)

tail(df_hospitals_ortho)

dim(df_hospitals_ortho)

attr(df_hospitals_ortho, "modelvars")
```

---

f_control_mactivate *Set Fitting Hyperparameters*

Description

Allows user a single function to tune the mactivate fitting algorithms, f_fit_gradient_01, f_fit_hybrid_01, f_fit_gradient_logistic_01.

Usage

```
f_control_mactivate(
param_sensitivity = 10^9,
bool_free_w = FALSE,
w0_seed = 0.1,
max_internal_iter = 500,
w_col_search = "one",
bool_headStart = FALSE,
antifreeze = FALSE,
ss_stop = 10^(-8),
escape_rate = 1.004,
step_size = 1/100,
Wadj = 1/1,
force_tries = 0,
lambda = 0,
tol = 10^(-8))
```

Arguments

param_sensitivity	Large positive scalar numeric.
bool_free_w	Scalar logical. Allow values of W to wander outside [0,1]?
w0_seed	Scalar numeric. Usually in [0,1]. Initial value(s) for multiplicative activation layer, W.
max_internal_iter	Scalar non-negative integer. **Hybrid only**. How many activation descent passes to make before refitting primary effects.
w_col_search	Scalar character. When one, locating W and corresponding coefficients is done (progressively) one column at a time; when all, locating W and corresponding coefficients is done for current column and all previous columns; When alternate, locating W and corresponding coefficients is done (progressively) one column at a time, however, after each column is fitted, an additonal pass is made fitting current column and all previous columns.
bool_headStart	Scalar logical. **Gradient only**. When TRUE, fitting first locates initial primary effects as a "head start" to the subsequent gradient fitting.
antifreeze	Scalar logical. **Hybrid only**. New w/v0.6.5. When FALSE, backwards compatible. When TRUE, prevents hanging (non-convergence) that may rarely occur when input space is highly correlated.
ss_stop	Small positive scalar numeric. Convergence tolerance.
escape_rate	Scalar numeric no less than one and likely no greater than, say, 1.01. Affinity for exiting a column search over W. E.g., if 1, fitting may take a long time. If 1.01, search for each column W will terminate relatively quickly.
step_size	Positive scalar numeric. Initial gradient step size (in both gradient and hybrid fitting algorithms) for all parameters.
Wadj	Positive scalar numeric. Control gradient step size (in both gradient and hybrid fitting algorithms) of W.

force_tries	Scalar non-negative integer. Force a minimum number of fitting recursions.
lambda	Scalar numeric. Ridge regularizer. The actual diagonal loading imposed upon the precision matrix is equal to lambda times its original diagonal. A value of 0 applies no loading; a value of 1 doubles the diagonal values of the precision matrix. This is applied to primary effects only. With gradient MLR fitting, i.e., f_fit_gradient_01, this only applies when arg bool_headStart is set to TRUE (otherwise there'd be nothing to regularize). With hybrid MLR fitting, i.e., f_fit_hybrid_01, this regularization is applied at each LS step (see About vignette). With logistic fitting, this arg **does nothing**. Note that with logistic fitting, we can always add a small amount of white noise to X.
tol	Small positive scalar numeric. **Hybrid only**. Similar to arg ss_stop above, but controls convergence tolerance after both recursions in hybrid fitting have completed.

Details

Fitting a mactivate model to data can/will be dramatically affected by these tuning hyperparameters. On one extreme, one set of hyperparameters may result in the fitting algorithm fruitlessly exiting almost immediately. Another set of hyperparameters may send the fitting algorithm to run and run for hours. While an ideal hyperparameterization will expeditiously fit the data.

Value

Named list to be passed to mact_control arg in fitting functions.

See Also

f_fit_gradient_01, f_fit_hybrid_01, f_fit_gradient_logistic_01.

Examples

```
library(mactivate)

set.seed(777)

d <- 20
N <- 50000

X <- matrix(rnorm(N*d, 0, 1), N, d)

colnames(X) <- paste0("x", I(1:d))

############## primary effect slopes
b <- rep_len( c(-1, 1), d )


ystar <-
```

```
X %*% b +
1 * (X[ , 1]) * (X[ , 2]) * (X[ , 3]) -
1 * (X[ , 2]) * (X[ , 3]) * (X[ , 4]) * (X[ , 5])

Xall <- X

errs <- rnorm(N, 0, 1)
errs <- 3 * (errs - mean(errs)) / sd(errs)

sd(errs)

y <- ystar + errs ### response

yall <- y
Nall <- N



############## hybrid example


### this control setting will exit too quickly
### compare this with example below

xcmact <-
f_control_mactivate(
param_sensitivity = 10^5,
w0_seed           = 0.1,
max_internal_iter = 500,
w_col_search      = "one",
ss_stop           = 10^(-5),
escape_rate       = 1.01,
Wadj              = 1/1,
lambda            = 1/1000,
tol               = 10^(-5)
)


m_tot <- 4

Uall <- Xall

xxnow <- Sys.time()

xxls_out <-
f_fit_hybrid_01(
X = Xall,
y = yall,
m_tot = m_tot,
U = Uall,
m_start = 1,
mact_control = xcmact,
verbosity = 1
```

```
)

cat( difftime(Sys.time(), xxnow, units="mins"), "\n" )

yhatG <- predict(object=xxls_out, X0=Xall, U0=Uall, mcols=m_tot )

sqrt( mean( (yall  -  yhatG)^2 ) )




####################### this control setting should fit
####################### (will take a few minutes)

xcmact <-
f_control_mactivate(
param_sensitivity = 10^10, ### make more sensitive
w0_seed           = 0.1,
max_internal_iter = 500,
w_col_search      = "one",
ss_stop           = 10^(-14), ### make stopping insensitive
escape_rate       = 1.001, #### discourage quitting descent
Wadj              = 1/1,
lambda            = 1/10000,
tol               = 10^(-14) ### make tolerance very small
)


m_tot <- 4

Uall <- Xall

xxnow <- Sys.time()

xxls_out <-
f_fit_hybrid_01(
X = Xall,
y = yall,
m_tot = m_tot,
U = Uall,
m_start = 1,
mact_control = xcmact,
verbosity = 1
)

cat( difftime(Sys.time(), xxnow, units="mins"), "\n" )

yhatG <- predict(object=xxls_out, X0=Xall, U0=Uall, mcols=m_tot )

sqrt( mean( (yall  -  yhatG)^2 ) )
```

```
xxls_out

Xstar <- f_mactivate(U=Uall, W=xxls_out[[ m_tot+1 ]][[ "What" ]])
colnames(Xstar) <- paste0("xstar_", seq(1, m_tot))
Xall <- cbind(Xall, Xstar)

xlm <- lm(yall~Xall)
summary(xlm)
```

`f_dmss_dW` *Calculate Derivative of Cost Function wrt W*

Description

Calculate the first derivative of objective function with respect to W, given data and requisite model parameter values.

Usage

```
f_dmss_dW(U, Xstar, W, yerrs, cc)
```

Arguments

`U`	Numeric matrix, `N` x `d_u` of activation inputs.
`Xstar`	Numeric matrix, `N` x `m`. The "new" polynomial inputs created by applying the activation layer, `W`, to `U`. Accomplished computationally with fun `f_mactivate`.
`W`	Numeric matrix, `d_u` x `m`, the multiplicative activation layer.
`yerrs`	Numeric vector of length `N`. `y` minus `yhat`.
`cc`	Numeric vector of length `m`. Coefficients for `Xstar`.

Details

There is really no need for user to call this function directly; this function is called by the fitting functions in this library. **Important.** Computationally there are (at least) two ways to solve this derivative, one is O(Nd), the other is O(Nd^2) (d is the number of columns in `U`). This function uses the first, computationally less expensive method. It is not an approximation; the simplification occurs simply by dividing out the appropriate partial term rather than taking the full product of terms across `U`. This has a very important implication of which we must be aware: **zeros in `U` may result in division by zero!** This function will handle the errors, but the ultimate consequence of zeros in `U` is that the derivative returned by this function may not be accurate. We should eliminate zeros in `U`. Standardizing `U` is one good solution. If zeros are only present because of "one-hot" indicators (dummies), another possible solution is to substitute -1 for 0 (actually not a bad practice anyway).

Value

Numeric matrix, d_u x m.

See Also

f_mactivate

Examples

```
#######
```

---

f_fit_gradient_01 *Fit Multivariate Regression Model with mactivate Using Gradient Descent*

Description

Use simple gradient descent to locate model parameters, i.e., primary effects, multiplicative effects, and activation parameters, W.

Usage

```
f_fit_gradient_01(
X,
y,
m_tot,
U = NULL,
m_start = 1,
mact_control = f_control_mactivate(),
verbosity = 2)
```

Arguments

X	Numerical matrix, N x d of model inputs. Do not include intercept term.
y	Numerical vector of length N. Model response, or output.
m_tot	Scalar non-negative integer. Total number of columns of activation layer, W, over which to fit.
U	Numerical matrix, N x d_u of model inputs to send to the activation layer, W. The default, NULL, instructs this function to simply use arg X.
m_start	Currently not used.
mact_control	Named list of class control_mactivate_obj as created by fun f_control_mactivate — fitting hyperparameters.
verbosity	Scalar integer.

Details

Please make sure to read Details in f_dmss_dW help page before using this function.

Value

An unnamed list of class mactivate_fit_gradient_01 of length m_tot + 1. Each node is a named list containing fitted parameter estimates. The first top-level node of this object contains parameter estimates when fitting 'primary effects' only (W has no columns), the second, parameter estimates for fitting with 1 column of W, and so on.

See Also

Essentially equivalent to, but likely slower than: f_fit_hybrid_01. See f_fit_gradient_logistic_01 for logistic data (binomial response).

Examples

```
xxnow <- Sys.time()

library(mactivate)

set.seed(777)


d <- 4
N <- 2000

X <- matrix(rnorm(N*d, 0, 1), N, d) ####

colnames(X) <- paste0("x", I(1:d))

############# primary effects
b <- rep_len( c(-1/2, 1/2), d )



###########

xxA <- (X[ , 1]+1/3) * (X[ , 2]-1/3)
#xxA <- (X[ , 1]+0/3) * (X[ , 2]-0/3)


ystar <-
X %*% b +
2 * xxA


m_tot <- 4
##############
```

```
xs2 <- "y ~ . "


xtrue_formula <- eval(parse(text=xs2))

xnoint_formula <- eval(parse(text="y ~ . - xxA"))



yerrs <- rnorm(N, 0, 3)

y <- ystar + yerrs

## y <- (y - mean(y)) / sd(y)


########## standardize X
Xall <- t( ( t(X) - apply(X, 2, mean) ) / apply(X, 2, sd) )
yall <- y
Nall <- N


####### fold index
xxfoldNumber <- rep_len(1:2, N)

ufolds <- sort(unique(xxfoldNumber)) ; ufolds


############### predict
############### predict


dfx <- data.frame("y"=yall, Xall, xxA)

tail(dfx)



################### incorrectly fit LM: no interactions

xlm <- lm(xnoint_formula , data=dfx)
summary(xlm)
yhat <- predict(xlm, newdata=dfx)
sqrt( mean( (yall - yhat)^2 ) )



################### correctly fit LM
xlm <- lm(xtrue_formula, data=dfx)
```

```
summary(xlm)
yhat <- predict(xlm, newdata=dfx)
sqrt( mean( (yall - yhat)^2 ) )




################ fit using gradient m-activation
######

m_tot <- 4


xcmact_gradient <-
f_control_mactivate(
param_sensitivity = 10^11,
bool_free_w       = TRUE,
w0_seed           = 0.05,
w_col_search      = "alternate",
bool_headStart    = TRUE,
ss_stop           = 10^(-12), ###
escape_rate       = 1.02,  #### 1.0002,
Wadj              = 1/1,
force_tries       = 0,
lambda            = 0
)

#### Fit

Uall <- Xall

head(Uall)

xthis_fold <- ufolds[ 1 ]

xndx_test <- which( xxfoldNumber %in% xthis_fold )
xndx_train <- setdiff( 1:Nall, xndx_test )

X_train <- Xall[ xndx_train, , drop=FALSE ]
y_train <- yall[ xndx_train ]
U_train <- Uall[ xndx_train, , drop=FALSE ]


xxls_out <-
f_fit_gradient_01(
X = X_train,
y = y_train,
m_tot = m_tot,
U = U_train,
m_start = 1,
mact_control = xcmact_gradient,
verbosity = 0
```

```
)



######### check test error

U_test <- Uall[ xndx_test, , drop=FALSE ]
X_test <- Xall[ xndx_test, , drop=FALSE ]
y_test <- yall[ xndx_test ]


yhatTT <- matrix(NA, length(xndx_test), m_tot+1)

for(iimm in 0:m_tot) {
    yhat_fold <- predict(object=xxls_out, X0=X_test, U0=U_test, mcols=iimm )
    yhatTT[ , iimm + 1 ] <- yhat_fold
}

errs_by_m <- NULL
for(iimm in 1:ncol(yhatTT)) {
    yhatX <- yhatTT[ , iimm]
    errs_by_m[ iimm ] <- sqrt(mean( (y_test - yhatX)^2 ))
    cat(iimm, "::", errs_by_m[ iimm ])
}


##### plot test RMSE vs m

plot(0:(length(errs_by_m)-1), errs_by_m, type="l", xlab="m", ylab="RMSE Cost")




##################
xtrue_formula_use <- xtrue_formula


xlm <- lm(xnoint_formula , data=dfx[ xndx_train, ])
yhat <- predict(xlm, newdata=dfx[ xndx_test, ])
cat("\n\n", "No interaction model RMSE:", sqrt( mean( (y_test - yhat)^2 ) ), "\n")


xlm <- lm(xtrue_formula_use , data=dfx[ xndx_train, ])
yhat <- predict(xlm, newdata=dfx[ xndx_test, ])
cat("\n\n", "'true' model RMSE:", sqrt( mean( (y_test - yhat)^2 ) ), "\n")


cat( "Runtime:", difftime(Sys.time(), xxnow, units="secs"), "\n" )
```

f_fit_gradient_logistic_01 *Fit Logistic Multivariate Regression Model with mactivate Using Gradient Descent*

Description

Use simple gradient descent to locate logistic model parameters, i.e., primary effects, multiplicative effects, and activation parameters, W.

Usage

```
f_fit_gradient_logistic_01(
X,
y,
m_tot,
U = NULL,
m_start = 1,
mact_control = f_control_mactivate(),
verbosity = 2)
```

Arguments

X	Numerical matrix, N x d of model inputs. Do not include intercept term.
y	Integer vector of length N, elements in {0, 1}. Binomial model response, or output.
m_tot	Scalar non-negative integer. Total number of columns of activation layer, W, over which to fit.
U	Numerical matrix, N x d_u of model inputs to send to the activation layer, W. The default, NULL, instructs this function to simply use arg X.
m_start	Currently not used.
mact_control	Named list of class control_mactivate_obj as created by fun f_control_mactivate — fitting hyperparameters.
verbosity	Scalar integer.

Details

Please make sure to read Details in f_dmss_dW help page before using this function.

Value

An unnamed list of class mactivate_fit_gradient_logistic_01 of length m_tot + 1. Each node is a named list containing fitted parameter estimates. The first top-level node of this object contains parameter estimates when fitting 'primary effects' only (W has no columns), the second, parameter estimates for fitting with 1 column of W, and so on.

See Also

See f_fit_gradient_01 or f_fit_gradient_logistic_01 for MLR data (numerical response).

Examples

```
xxnow <- Sys.time()

library(mactivate)

set.seed(777)


d <- 4
N <- 2400

X <- matrix(rnorm(N*d, 0, 1), N, d) ####

colnames(X) <- paste0("x", I(1:d))

############## primary effects
b <- rep_len( c(-1/2, 1/2), d )


xxA <- (X[ , 1]+1/3) * (X[ , 2]-1/3)
xxB <- (X[ , 1]+0/3) * (X[ , 2]-0/3) * (X[ , 4]-1/3)


ystar <-
X %*% b +
2 * xxA -
1 * xxB

xs2 <- "y ~ . "

xtrue_formula <- eval(parse(text=xs2))

xnoint_formula <- eval(parse(text="y ~ . - xxA - xxB"))


ysigmoid <- 1 / (1 + exp(-ystar))

range(ysigmoid)

y <- rbinom(size=1, n=N ,prob=ysigmoid)


Nall <- N

cov(X)

yall <- y
```

```
Xall <- X

### Xall <- X + rnorm(prod(dim(X)), 0, 1/10000) ### add a little noise -- optional

sd(y)


dfx <- data.frame("y"=yall, Xall, xxA, xxB)

tail(dfx)



################### incorrectly fit LM: no interactions
xglm <- glm(xnoint_formula , data=dfx, family=binomial(link="logit"))
summary(xglm)
yhat <- predict(xglm, newdata=dfx, type="response")
mean( f_logit_cost(y=yall, yhat=yhat) )


####### known true
xglm <- glm(xtrue_formula , data=dfx, family=binomial(link="logit"))
summary(xglm)
yhat <- predict(xglm, newdata=dfx, type="response")
mean( f_logit_cost(y=yall, yhat=yhat) )



xxfoldNumber <- rep_len( 1:4, Nall )

ufolds <- sort(unique(xxfoldNumber))

######################

xthis_fold <- ufolds[ 1 ]

xndx_test <- which( xxfoldNumber %in% xthis_fold )

xndx_train <- setdiff( 1:Nall, xndx_test )

##################

X_train <- Xall[ xndx_train, , drop=FALSE ]

X_test <- Xall[ xndx_test, , drop=FALSE ]

y_train <- yall[ xndx_train ]

y_test <- yall[ xndx_test ]
```

```
###################

m_tot <- 4

xcmact_gradient <-
f_control_mactivate(
param_sensitivity = 10^11,
bool_free_w       = FALSE,
w0_seed           = 0.05,
#w_col_search      = "alternate",
w_col_search      = "one",
bool_headStart    = TRUE,
ss_stop           = 10^(-12), ### very small
escape_rate       = 1.02,
step_size         = 1,
Wadj              = 1/1,
force_tries       = 0,
lambda            = 1/1 #### does nothing here
)


Uall <- Xall


X_train <- Xall[ xndx_train, , drop=FALSE ]
y_train <- yall[ xndx_train ]


xxls_out <-
f_fit_gradient_logistic_01(
X = X_train,
y = y_train,
m_tot = m_tot,
U = X_train,
m_start = 1,
mact_control = xcmact_gradient,
verbosity = 0
)


######### check test error

U_test <- Xall[ xndx_test, , drop=FALSE ]
X_test <- Xall[ xndx_test, , drop=FALSE ]
y_test <- yall[ xndx_test ]


yhatTT <- matrix(NA, length(xndx_test), m_tot+1)

for(iimm in 0:m_tot) {
    yhat_fold <- predict(object=xxls_out, X0=X_test, U0=U_test, mcols=iimm )
    yhatTT[ , iimm + 1 ] <- yhat_fold[[ "p0hat" ]]
}
```

```
errs_by_m <- NULL
for(iimm in 1:ncol(yhatTT)) {
    yhatX <- yhatTT[ , iimm]
    errs_by_m[ iimm ] <- mean( f_logit_cost(y=y_test, yhat=yhatX) )
    cat(iimm, "::", errs_by_m[ iimm ])
}

##### plot test Logit vs m

plot(0:(length(errs_by_m)-1), errs_by_m, type="l", xlab="m", ylab="Logit Cost")




################## test off 'correct' model
xtrue_formula_use <- xtrue_formula



xglm <- glm(xnoint_formula , data=dfx[ xndx_train, ], family=binomial(link="logit"))
yhat <- predict(xglm, newdata=dfx[ xndx_test, ], type="response")
cat("\n\n", "No interaction model logit:", mean( f_logit_cost(y=y_test, yhat=yhat) ), "\n")


xglm <- glm(xtrue_formula_use , data=dfx[ xndx_train, ], family=binomial(link="logit"))
yhat <- predict(xglm, newdata=dfx[ xndx_test, ], type="response")
cat("\n\n", "'true' model logit:", mean( f_logit_cost(y=y_test, yhat=yhat) ) , "\n")


cat( "Runtime:", difftime(Sys.time(), xxnow, units="secs"), "\n" )
```

`f_fit_hybrid_01` *Fit Multivariate Regression Model with mactivate Using Hybrid Method*

Description

Use hybrid algorithm (essentially a flavor of EM) to locate model parameters, i.e., primary effects, multiplicative effects, and activation parameters, `W`.

Usage

```
f_fit_hybrid_01(
X,
y,
m_tot,
U = NULL,
```

```
m_start = 1,
mact_control = f_control_mactivate(),
verbosity = 2)
```

Arguments

X	Numerical matrix, N x d of model inputs. Do not include intercept term.
y	Numerical vector of length N. Model response, or output.
m_tot	Scalar non-negative integer. Total number of columns of activation layer, W, over which to fit.
U	Numerical matrix, N x d_u of model inputs to send to the activation layer, W. The default, NULL, instructs this function to simply use arg X.
m_start	Currently not used.
mact_control	Named list of class control_mactivate_obj as created by fun f_control_mactivate — fitting hyperparameters.
verbosity	Scalar integer.

Details

Please make sure to read Details in f_dmss_dW help page before using this function.

Value

An unnamed list of class mactivate_fit_hybrid_01 of length m_tot + 1. Each node is a named list containing fitted parameter estimates. The first top-level node of this object contains parameter estimates when fitting 'primary effects' only (W has no columns), the second, parameter estimates for fitting with 1 column of W, and so on.

See Also

Essentially equivalent to, but likely faster than: f_fit_gradient_01. See f_fit_gradient_logistic_01 for logistic data (binomial response).

Examples

```
xxnow <- Sys.time()

library(mactivate)

set.seed(777)


d <- 4
N <- 2000

X <- matrix(rnorm(N*d, 0, 1), N, d) ####

colnames(X) <- paste0("x", I(1:d))
```

```
############# primary effects
b <- rep_len( c(-1/2, 1/2), d )



###########

xxA <- (X[ , 1]+1/3) * (X[ , 2]-1/3)
#xxA <- (X[ , 1]+0/3) * (X[ , 2]-0/3)


ystar <-
X %*% b +
2 * xxA



#############




xs2 <- "y ~ . "


xtrue_formula <- eval(parse(text=xs2))

xnoint_formula <- eval(parse(text="y ~ . - xxA"))



yerrs <- rnorm(N, 0, 3)

y <- ystar + yerrs

## y <- (y - mean(y)) / sd(y)


########## standardize X
Xall <- t( ( t(X) - apply(X, 2, mean) ) / apply(X, 2, sd) )
yall <- y
Nall <- N


####### fold index
xxfoldNumber <- rep_len(1:2, N)

ufolds <- sort(unique(xxfoldNumber)) ; ufolds


############### predict
```

```
############### predict


dfx <- data.frame("y"=yall, Xall, xxA)

tail(dfx)



################### incorrectly fit LM: no interactions
xlm <- lm(xnoint_formula , data=dfx)
summary(xlm)
yhat <- predict(xlm, newdata=dfx)
sqrt( mean( (yall - yhat)^2 ) )



################### correctly fit LM
xlm <- lm(xtrue_formula, data=dfx)
summary(xlm)
yhat <- predict(xlm, newdata=dfx)
sqrt( mean( (yall - yhat)^2 ) )




################ fit using hybrid m-activation
######

m_tot <- 4

xcmact_hybrid <-
f_control_mactivate(
param_sensitivity = 10^12,
bool_free_w       = TRUE,
w0_seed           = 0.1, ### 0.01
w_col_search      = "alternate",
max_internal_iter = 500, #####
ss_stop           = 10^(-11), ###
escape_rate       = 1.02, ### 1.05
Wadj              = 1/1,
force_tries       = 0,
lambda            = 0/10000, ### hybrid only
tol               = 10^(-11) ### hybrid only
)

#### Fit

Uall <- Xall

head(Uall)
```

```
xthis_fold <- ufolds[ 1 ]

xndx_test <- which( xxfoldNumber %in% xthis_fold )
xndx_train <- setdiff( 1:Nall, xndx_test )

X_train <- Xall[ xndx_train, , drop=FALSE ]
y_train <- yall[ xndx_train ]
U_train <- Uall[ xndx_train, , drop=FALSE ]


xxls_out <-
f_fit_hybrid_01(
X = X_train,
y = y_train,
m_tot = m_tot,
U = U_train,
m_start = 1,
mact_control = xcmact_hybrid,
verbosity = 1
)



######### check test error

U_test <- Uall[ xndx_test, , drop=FALSE ]
X_test <- Xall[ xndx_test, , drop=FALSE ]
y_test <- yall[ xndx_test ]


yhatTT <- matrix(NA, length(xndx_test), m_tot+1)

for(iimm in 0:m_tot) {
    yhat_fold <- predict(object=xxls_out, X0=X_test, U0=U_test, mcols=iimm )
    yhatTT[ , iimm + 1 ] <- yhat_fold
}

errs_by_m <- NULL
for(iimm in 1:ncol(yhatTT)) {
    yhatX <- yhatTT[ , iimm]
    errs_by_m[ iimm ] <- sqrt(mean( (y_test - yhatX)^2 ))
    cat(iimm, "::", errs_by_m[ iimm ])
}



plot(0:(length(errs_by_m)-1), errs_by_m, type="l", xlab="m", ylab="RMSE Cost")
```

```
##################
xtrue_formula_use <- xtrue_formula


xlm <- lm(xnoint_formula , data=dfx[ xndx_train, ])
yhat <- predict(xlm, newdata=dfx[ xndx_test, ])
cat("\n\n", "No interaction model RMSE:", sqrt( mean( (y_test - yhat)^2 ) ), "\n")


xlm <- lm(xtrue_formula_use , data=dfx[ xndx_train, ])
yhat <- predict(xlm, newdata=dfx[ xndx_test, ])
cat("\n\n", "'true' model RMSE:", sqrt( mean( (y_test - yhat)^2 ) ), "\n")


cat( "Runtime:", difftime(Sys.time(), xxnow, units="secs"), "\n" )
```

`f_logit_cost` *Logistic Cost*

Description

Calculate the logistic cost of probability predictions of a dichotomous outcome.

Usage

```
f_logit_cost(y, yhat)
```

Arguments

`y`	Numeric vector. The outcome vector. Must be in $\{0, 1\}$.
`yhat`	Numeric vector. Prediction vector. Should be in $(0, 1)$ – the open unit interval. In an inferential setting, one should probably never make a prediction of zero or one; however, values of zero or one are allowed, provided they are "correct".

Details

This function is included in this library as a convenience.

Value

A numeric vector of length equal to `y` and `yhat`. The logistic cost associated with each corresponding prediction.

See Also

`f_fit_gradient_logistic_01`, `predict.mactivate_fit_gradient_logistic_01`.

Examples

```
y <- c(0, 0, 1, 1)
yhat <- rep(1/2, length(y))

mean( f_logit_cost(y=y, yhat=yhat) )
```

`f_mactivate` *Map Activation Layer and Inputs to Polynomial Model Inputs*

Description

Passes activation inputs, `U` into activation layer, `W`, to obtain new polynomial model inputs.

Usage

```
f_mactivate(U, W)
```

Arguments

`U`	Numeric matrix, `N` x `d_u` of activation inputs.
`W`	Numeric matrix, `d_u` x `m`, the multiplicative activation layer.

Details

This function calculates the multiplicative activations; it maps selected inputs, `U`, back into the input space using the m-activation layer(s). In practice, the arg `W`, will be a fitted value, as created by the fitting functions.

Value

Numeric matrix, `N` x `m`. Referred to as `Xstar` elsewhere in this documentation.

Examples

```
library(mactivate)

set.seed(777)


d <- 7
N <- 15000

X <- matrix(rnorm(N*d, 0, 1), N, d) ####

colnames(X) <- paste0("x", I(1:d))
```

```
############# primary effects
b <- rep_len( c(-1/4, 1/4), d )



###########

xxA <- (X[ , 1]+1/3) * (X[ , 1]-1/3) * (X[ , 3]+1/3)
xxB <- (X[ , 2]+0) * (X[ , 2]+1/3) * (X[ , 3]-0) * (X[ , 3]-1/3)
xxC <- (X[ , 3]+1/3) * (X[ , 3]-1/3)

ystar <-
X %*% b +
1/3 * xxA -
1/2 * xxB +
1/3 * xxC


#############

xs2 <- "y ~ . "

xtrue_formula <- eval(parse(text=xs2))

xnoint_formula <- eval(parse(text="y ~ . - xxA - xxB - xxC"))



yerrs <- rnorm(N, 0, 3)

y <- ystar + yerrs

########## standardize X
Xall <- t( ( t(X) - apply(X, 2, mean) ) / apply(X, 2, sd) )
yall <- y
Nall <- N


####### fold index
xxfoldNumber <- rep_len(1:2, N)

ufolds <- sort(unique(xxfoldNumber)) ; ufolds


############### predict
############### predict


dfx <- data.frame("y"=yall, Xall, xxA, xxB, xxC)

tail(dfx)
```

```
################### incorrectly fit LM: no interactions

xlm <- lm(xnoint_formula , data=dfx)
summary(xlm)
yhat <- predict(xlm, newdata=dfx)
sqrt( mean( (yall - yhat)^2 ) )



################### correctly fit LM
xlm <- lm(xtrue_formula, data=dfx)
summary(xlm)
yhat <- predict(xlm, newdata=dfx)
sqrt( mean( (yall - yhat)^2 ) )




################ fit using hybrid m-activation
###### takes about 2 minutes

xcmact_hybrid <-
f_control_mactivate(
param_sensitivity = 10^12,
bool_free_w       = TRUE,
w0_seed           = 0.1,
w_col_search      = "alternate",
max_internal_iter = 500, #####
ss_stop           = 10^(-14), ###
escape_rate       = 1.005,
Wadj              = 1/1,
force_tries       = 0,
lambda            = 0/10000, ###
tol               = 10^(-14) ###
)



#### Fit

m_tot <- 7

Uall <- cbind(Xall, Xall)
colnames(Uall) <- paste0(rep(c("a_", "b_"), each=d), colnames(Uall))

head(Uall)

xthis_fold <- ufolds[ 1 ]
```

```
xndx_test <- which( xxfoldNumber %in% xthis_fold )
xndx_train <- setdiff( 1:Nall, xndx_test )

X_train <- Xall[ xndx_train, , drop=FALSE ]
y_train <- yall[ xndx_train ]
U_train <- Uall[ xndx_train, , drop=FALSE ]

xxnow <- Sys.time()
xxls_out <-
f_fit_hybrid_01(
X = X_train,
y = y_train,
m_tot = m_tot,
U = U_train,
m_start = 1,
mact_control = xcmact_hybrid,
verbosity = 1
)
cat( difftime(Sys.time(), xxnow, units="mins"), "\n" )



######### check test error

U_test <- Uall[ xndx_test, , drop=FALSE ]
X_test <- Xall[ xndx_test, , drop=FALSE ]
y_test <- yall[ xndx_test ]


yhatTT <- matrix(NA, length(xndx_test), m_tot+1)

for(iimm in 0:m_tot) {
    yhat_fold <- predict(object=xxls_out, X0=X_test, U0=U_test, mcols=iimm )
    yhatTT[ , iimm + 1 ] <- yhat_fold
}

errs_by_m <- NULL
for(iimm in 1:ncol(yhatTT)) {
    yhatX <- yhatTT[ , iimm]
    errs_by_m[ iimm ] <- sqrt(mean( (y_test - yhatX)^2 ))
    cat(iimm, "::", errs_by_m[ iimm ])
}

plot(0:(length(errs_by_m)-1), errs_by_m, type="l", xlab="m", ylab="RMSE Cost")




##################

xthis_fold <- ufolds[ 1 ]
```

```
xndx_test <- which( xxfoldNumber %in% xthis_fold )
xndx_train <- setdiff( 1:Nall, xndx_test )

xlm <- lm(xtrue_formula , data=dfx[ xndx_train, ])
yhat <- predict(xlm, newdata=dfx[ xndx_test, ])

sqrt( mean( (y_test - yhat)^2 ) )


################ hatXstar

X_test <- Xall[ xndx_test, ]
y_test <- yall[ xndx_test ]

Xstar_test <- f_mactivate(U=U_test, W=xxls_out[[ length(xxls_out) ]][[ "What" ]])
Xi <- cbind(X_test, Xstar_test)
xlm <- lm(y_test ~ Xi)

sumxlm <- summary(xlm)
print(sumxlm)

xcoefs <- sumxlm$coefficients
xcoefs <- xcoefs[ (2+d):nrow(xcoefs), ] ; xcoefs

xndox_cu <- which( abs(xcoefs[ , "t value"]) > 3 ) ; xndox_cu


bWhat <- xxls_out[[ length(xxls_out) ]][[ "What" ]][ ,  xndox_cu ]
bWhat

wwmag <- apply(bWhat, 1, function(x) { return(sum(abs(x)))} ) ; wwmag

plot(wwmag, type="h", lwd=4,
ylim=c(0, max(wwmag)),
main="W Coefficient Total Magnitute vs Input Term",
xlab="Column of U",
ylab="Sum of magnitudes in fitted W",
cex.lab=1.3
)
```

`predict.mactivate_fit_gradient_01`

Predict from Fitted Gradient Model

Description

Predict using fitted model returned by f_fit_gradient_01.

Usage

```
## S3 method for class 'mactivate_fit_gradient_01'
predict(object, X0, U0=NULL, mcols, ...)
```

Arguments

object	A list of class 'mactivate_fit_gradient_01' as returned by f_fit_gradient_01().
X0	Numeric matrix, N x d. Model 'primary effect' inputs.
U0	Numeric matrix with N rows. Inputs to pass to activation layer.
mcols	Scalar non-negative integer specifying which first columns of W to use.
...	Nothing else is required for this extension of the predict() function.

Details

If U0 is not provided, X0 will be passed to activation layer.

Value

yhat. Numeric vector of length N.

See Also

f_fit_gradient_01.

Examples

```
####### Please see examples in the fitting functions
```

---

predict.mactivate_fit_gradient_logistic_01

Predict from Fitted Gradient Logistic Model

Description

Predict using fitted model returned by f_fit_gradient_logistic_01.

Usage

```
## S3 method for class 'mactivate_fit_gradient_logistic_01'
predict(object, X0, U0=NULL, mcols, ...)
```

Arguments

object	A list of class 'mactivate_fit_gradient_logistic_01' as returned by f_fit_gradient_logistic_01().
X0	Numeric matrix, N x d. Model 'primary effect' inputs.
U0	Numeric matrix with N rows. Inputs to pass to activation layer.
mcols	Scalar non-negative integer specifying which first columns of W to use.
...	Nothing else is required for this extension of the predict() function.

Details

If U0 is not provided, X0 will be passed to activation layer.

Value

A named list with 2 elements:

y0hat	Vector of length N. Linear predictions
p0hat	Vector of length N. Probability predictions. Similar to setting type='response' when predicting from glm logistic fitted model

See Also

f_fit_gradient_logistic_01.

Examples

```
####### Please see examples in the fitting functions
```

---

predict.mactivate_fit_hybrid_01

Predict from Fitted Hybrid Model

Description

Predict using fitted model returned by f_fit_hybrid_01.

Usage

```
## S3 method for class 'mactivate_fit_hybrid_01'
predict(object, X0, U0=NULL, mcols, ...)
```

Arguments

object	A list of class 'mactivate_fit_hybrid_01' as returned by f_fit_hybrid_01.
X0	Numeric matrix, N x d. Model 'primary effect' inputs.
U0	Numeric matrix with N rows. Inputs to pass to activation layer.
mcols	Scalar non-negative integer specifying which first columns of W to use.
...	Nothing else is required for this extension of the predict() function.

Details

If `U0` is not provided, `X0` will be passed to activation layer.

Value

`yhat`. Numeric vector of length `N`.

See Also

`f_fit_hybrid_01`.

Examples

```
####### Please see examples in the fitting functions
```

Index