

Package: mpred (via r-universe)

October 21, 2026

Type Package

Title Generic Predictive Margins

Version 0.4

Date 2025-07-09

Author Martin Elff

Maintainer Martin Elff <martin@elff.eu>

Description Provides a function to compute predictive margins.

License GPL-2

LazyLoad Yes

Depends R (>= 3.3.3), stats

Imports parallel, ordinal

Suggests mclogit, magrittr, carData, MASS, ggplot2

Config/roxygen2/version 8.1.0

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

Date/Publication 2026-08-22 20:04:05 UTC

RemoteUrl <https://github.com/melff/mpred>

RemoteRef HEAD

RemoteSha ddd29e9fafed2ad801c06fcac1e19ee97121e6c3

RemoteSubdir pkg

Contents

cibeta	2
cinorm	2
get_cifunc	3
predmarg	3

Index	9
--------------	----------

cibeta	<i>Confidence/prediction interval based on a beta distribution</i>
--------	--

Description

Creates prediction intervals from a beta distribution with given mean and standard deviation. Prediction intervals stay between 0 and 1.

Usage

```
cibeta(mean, sd, level)
```

Arguments

mean	a scalar; the mean of the beta distribution
sd	a scalar; the standard deviation of the beta distribution
level	a scalar; the confidence level

cinorm	<i>Confidence/prediction interval based on a normal distribution</i>
--------	--

Description

Creates prediction intervals from a normal distribution with given mean and standard deviation.

Usage

```
cinorm(mean, sd, level)
```

Arguments

mean	a scalar; the mean parameter of the normal distribution
sd	a scalar; the standard deviation parameter of the normal distribution
level	a scalar; the confidence level

get_cifunc	<i>A generic function that returns functions for confidence/prediction intervals</i>
------------	--

Description

Returns a function for confidence or prediction intervals appropriate for an fitted model object. For linear regression this will be cinorm(), for logistic regression (done by glm) this will be cibeta, etc.

Usage

```
get_cifunc(obj)

## S3 method for class 'lm'
get_cifunc(obj)

## S3 method for class 'glm'
get_cifunc(obj)

## S3 method for class 'mblogit'
get_cifunc(obj)

## S3 method for class 'mclogit'
get_cifunc(obj)

## S3 method for class 'clm'
get_cifunc(obj)

## S3 method for class 'clmm'
get_cifunc(obj)
```

Arguments

obj	an fitted model object
-----	------------------------

predmarg	<i>Generic function to produce predictive margins</i>
----------	---

Description

Generic function to produce predictive margins

Usage

```

predmarg(
  obj,
  settings,
  data,
  subset,
  type = NULL,
  groups = NULL,
  setup = NULL,
  cifunc = get_cifunc(obj),
  level = 0.95,
  parallel = FALSE,
  n = 21,
  trim = 0,
  mc.cores = if (.Platform$OS.type == "windows") 1L else max.cores,
  predict.args = NULL,
  ...
)

```

Arguments

<code>obj</code>	a model object, e.g. returned by <code>lm</code> , <code>glm</code> , etc.
<code>settings</code>	an optional data frame of settings for independent variables or a formula with variables for which settings are to be generated.
<code>data</code>	an optional data frame for which the predictive margins are computed. If omitted, an attempt is made to obtain the data from the model object.
<code>subset</code>	an optional logical vector that defines a subset for which a predictive margin is computed
<code>type</code>	an optional character string that specifies the type of predictions, e.g. probabilities or cumulative probabilities. For future versions only.
<code>groups</code>	a variable that defines groups for which predictive margins are computed. This variable has to have the same number of observations as the data to which the model was fitted.
<code>setup</code>	an optional expression that is evaluated for each setting, i.e. individually for each row of the settings data frame. Can be used to modify independent variables.
<code>cifunc</code>	a function to compute prediction intervals. By default it is the chosen by the function of the same name.
<code>level</code>	level of confidence intervals of predictions.
<code>parallel</code>	logical value that determines whether predictions for individual settings are computed in parallel. (Does not yet work on windows.)
<code>n</code>	an integer number, the number of values generated for an independent variable if settings is a formula. The values form an equally spaced grid that cover the range from the variable in data.
<code>trim</code>	a number between 0 and 1. The proportion of extreme values of variable in data that should be trimmed from the range of a generated set of values of an independent variable.

<code>mc.cores</code>	number of CPU cores used for parallel processing.
<code>predict.args</code>	an optional list with additional arguments to internal calls of predict
<code>...</code>	optional vectors of values of independent variables. These further arguments, if present, are used to create a data frame of settings, using expand.grid .

Details

The generic function `predmarg` computes predictive margins for various settings of the independent variables. It is also possible to provide settings for independent variables that are included in the model, but that are used in the `setup` expression to transform independent variables. See the examples below.

Value

a data frame with the following variables:

<code>pred</code>	the mean prediction for the setting of the independent variables
<code>var.pred</code>	the (estimated) variance of the mean prediction
<code>se.pred</code>	the standard error of prediction, i.e. the square root of the variance of the mean prediction
<code>lower</code>	lower prediction interval computed with <code>qfunc</code>
<code>upper</code>	upper prediction interval computed with <code>qfunc</code>
<code>...</code>	the independent variables for which values are set to create the predictions are also included in the resulting data frame.

Examples

```
library(magrittr)

# Simple linear regression

fm <- lm(weight ~ poly(height, 2), data = women)
pm <- predmarg(fm,
               height=seq(from=58,to=72,
                           length=10))

str(pm)

plot(pred~height,data=pm,
      type="l")

with(women, points(height,weight))
with(pm, lines(height,lower,lty=2))
with(pm, lines(height,upper,lty=2))

# Logistic regression

library(carData)
Chile %<>% within({
```

```

    vote2 <- factor(vote,levels=c("N","Y"))
    vote2 <- as.integer(vote2=="Y")
  })

glm.Chile.1 <- glm(vote2~sex+age+income+education,
                  data=Chile,
                  family=binomial)
summary(glm.Chile.1)

pm.Chile.1.income <- predmarg(glm.Chile.1,
                             income=seq(from=2500,to=200000,length=20))

plot(pred~income,data=pm.Chile.1.income,
     type="l")

# Baseline category logit

library(mcllogit)
library(MASS)

mb.Chile <- mblogit(vote~statusquo,
                   data=Chile)
pm.mb.Chile <- predmarg(mb.Chile,
                      statusquo=seq(from=-2,to=2,length=20))
str(pm.mb.Chile)

library(ggplot2)
(ggplot(pm.mb.Chile,
       aes(x=statusquo,
           y=pred,
           fill=response)
       ) + geom_area())

(ggplot(pm.mb.Chile,
       aes(x=statusquo,
           y=pred,
           ymin=lower,
           ymax=upper)
       ) + geom_line() +geom_ribbon(alpha=.25) + facet_grid(~response))

mb.hs <- mblogit(Sat~Infl+Type+Cont,weights=Freq,
                data=housing)

pm.mb.hs <- predmarg(mb.hs,
                    Infl=levels(Infl),
                    Type=levels(Type))

dodge <- position_dodge(width=.8)
(ggplot(pm.mb.hs)
 +facet_wrap(~Type)

```

```

+geom_bar(
  aes(fill=response,
      x=Infl,
      y=pred),
  stat='identity',position=dodge,width=.8)
+geom_errorbar(
  aes(x=Infl,
      ymin=lower,
      ymax=upper,group=response),
  position=dodge,width=.4))

# The following requires the most current 'mclogit' version on GitHub
# and fails with the CRAN version
# # Baseline category logit with random effects
#
# # Some artificial data
# exadata <- local({
#   B <- cbind(c(-.5,.3),
#             c(.5,-.5))
#   set.seed(42)
#   x <- rnorm(n=60)
#   X <- cbind(1,x)
#   Eta <- X %*% B
#   j <- rep(1:10,6)
#   jf <- as.factor(j)
#   u1 <- rnorm(n=10,sd=.8)
#   u2 <- rnorm(n=10,sd=.8)
#   Eta <- Eta + cbind(u1[j],x*u2[j])
#   expEta <- cbind(1,exp(Eta))
#   sum.expEta <- rowSums(expEta)
#   pi <- expEta/sum.expEta
#   Y <- t(apply(pi,1,rmultinom,n=1,size=300))
#   res <- data.frame(Y,x,j,jf)
#   names(res)[1:3] <- paste0("y",1:3)
#   res
# })
#
# # Baseline logit model with random intercepts and random slopes
# mbrsl <- mblogit(cbind(y1,y2,y3)~x,data=exadata,
#   random = ~1+x|j)
# summary(mbrsl)
#
# # Predictive margins for values of x
# pm.mbrsl <- predmarg(mbrsl,x=seq(from=min(x),to=max(x),length=24))
# (ggplot(pm.mbrsl,
#   aes(x=x,
#       y=pred,
#       fill=response
#     )
#   ) + geom_area())
#
# # Predictive margins for the random effects
# pm.mbrsl.j <- predmarg(mbrsl,j=1:10)

```

```
# (ggplot(pm.mbrsl.j,  
#       aes(x=j,  
#           y=pred,  
#           fill=response  
#       )  
#       ) + geom_bar(position="fill",stat="identity"))  
#  
# pm.mbrsl.jx <- predmarg(mbrsl,  
#       j=1:10,  
#       x=seq(from=min(x),to=max(x),  
#             length=24))  
#  
# (ggplot(pm.mbrsl.jx,  
#       aes(x=x,  
#           y=pred,  
#           fill=response  
#       )  
#       ) + geom_area()  
#       + facet_wrap(~j))
```

Index

cibeta, [2](#)

cinorm, [2](#)

expand.grid, [5](#)

get_cifunc, [3](#)

predict, [5](#)

predmarg, [3](#)