# Package ‘UMA’

May 18, 2022

**Type** Package

**Title** Universal Model Averaging

**Version** 1.0

**Author** Li Wen <wlwendy1008@163.com>, Zhihao Zhao <zhzhao@cueb.edu.cn>, Yuhong Yang <yyang@stat.umn.edu>

**Maintainer** Li Wen <wlwendy1008@163.com>, Zhihao Zhao <zhzhao@cueb.edu.cn>

**Description** UMA provides adaptive model averaging (MA) with both linear and nonparamatric methods. It also allows the use of other averaging methods such as smoothed information criteria and Mallow's MA.

**License** GPL-2

**Imports** SOIL, BMA, randomForest, mboost, gbm, BayesTree, glmnet, ncvreg, mgcv, mvtnorm, Matrix, MASS, ModelMetrics, mda, quadprog, ipred, lars, plyr, stats, parallel, nnet, CatReg, grpreg

**Encoding** UTF-8

**LazyData** true

**NeedsCompilation** no

**Depends** R (>= 3.5.0)

## R topics documented:

---

`gma` *general model averaging for low-dimensional inputs*

---

### Description

gma provides model averaging for linear regrssion with low dimensional inputs (no more than 20 covariates). The MA methods included are SAIC, SBIC, SFIC, ARM, L1-ARM, MMA and JMA.

## Usage

```
gma(x,y,factorID=NULL,method='L1-ARM',candi_models=2,n_train=ceiling(n/2),
    no_rep=50)
```

## Arguments

| | |
|---|---|
| `x` | Matrix of predictors. |
| `y` | Response variable. |
| `factorID` | Indication on whether there are categorical variables among the predictors. If factorID= NULL, the predictors are all continuous or have the identifiable categorical variables; If factorID=`'colnames'` or the location numbers of categorical variables, the name or location of variables provided by the user are treated as categorical variables in the linear model. The default factorID is NULL. |
| `method` | The method for calculating weights. The method= `'SAIC'` is the Smooth-AIC method; the method= `'SBIC'` is the Smooth-BIC method; the method= `'SFIC'` is the Smooth-FIC method; the method= `'ARM'` is the Adaptive Regression by Mixing method; the method= `'L1-ARM'` is the L1 Adaptive Regression by Mixing method; the method= `'MMA'` is the Mallow's Model Averaging (MMA); the method= `'JMA'` is the Jackknife Model Averaging (JMA). The default is `'L1-ARM'`. |
| `candi_models` | Set to 1 for nested subset models in the order given in predictors; set to 2 for all combinations of subsets; input an m*p matrix, where m is the number of models to be combined, and each row of which is a 0/1 indicator vector representing whether each variable is included/excluded in the model. The default is 2. |
| `n_train` | Size of training set when the weight function is `L1-ARM` or `ARM` with `prior=TRUE`. The default value is `n_train=ceiling(n/2)`. |
| `no_rep` | Number of replications when the weight function is `L1-ARM` and `ARM`. The default value is `no_rep=50`. |

## Details

See the paper provided in Reference section.

## Value

A `'gma'` object is retured. The components are:

| | |
|---|---|
| `weight` | The weight for each candidate model. |
| `weight_se` | The standard error of the weights of the candidate models over the data-splittings under the method= `'ARM'` or method=`'L1-ARM'`. |
| `wbetahat` | The weighted estimation of the coefficients. |
| `betahat` | The coefficients matrix estimated by candidate models. |
| `candi_models` | The candidate models. |

## Examples

```
# generate simulation data
n<-50
p<-8
beta<-c(3,1.5,0,0,2,0,0,0)
```

```
b0<-1
x<-matrix(rnorm(n*p,0,1),nrow=n,ncol=p)
e<-rnorm(n,0,3)
y<-x%*%beta+b0+e

# compute weight for candidate models using L1-ARM, JMA and SAIC with nested subsets candidate models
lw<-gma(x,y,factorID=NULL,method='L1-ARM',candi_models=1)$weight
jw<-gma(x,y,factorID=NULL,method='JMA',candi_models=1)$weight
saw<-gma(x,y,factorID=NULL,method='SAIC',candi_models=1)$weight

# output the candidate models used for method L1-ARM
candi_models<-gma(x,y,factorID=NULL,method='L1-ARM',candi_models=1)$candi_models

# simulation with categorical variables
n<-100
x1<-rnorm(n)
x2<-rnorm(n)
x3<-rnorm(n)
x4<-factor(sample(1:5,n,replace=T),levels=c(1:5))
X<-data.frame(x1,x2,x3,x4)
Z<-as.matrix(model.matrix(~.-1,data=as.data.frame(X)))[,-4]
mu<-Z%*%c(0.1,0.3,0.5,1,-2,4,-3)
y<-mu+rnorm(n,0,3)

# compute weight for candidate models using MMA with nested subsets candidate models
mmaw <- gma(X, y, factorID = 'x4', method = 'MMA', candi_models = 1)$weight

# early COVID-19 data in China
data(covid19)
y<-covid19[,1]
x<-covid19[,-1]
n<-length(y)

# the weighted estimation using L1-ARM, MMA and SFIC with all subsets candidate models
Cl<-gma(x,y,factorID=NULL,method='L1-ARM',candi_models=2)$wbetahat
Cm<-gma(x,y,factorID=NULL,method='MMA',candi_models=2)$wbetahat
Csf<-gma(x,y,factorID=NULL,method='SFIC',candi_models=2)$wbetahat
```

---

## gma_h — *general model averaging for high-dimensional inputs*

---

### Description

gma_h provides model averaging for linear regression with high-dimensional inputs. The Model Averaging methods included are SAICp, SBICp, ARM, L1-ARM, PMA, MCV.

### Usage

```
gma_h(x, y, factorID=NULL,candidate='H4',method='L1-ARM', psi=1,
     n_train=ceiling(n/2),no_rep=20, lambda=log(n), alpha=0.05, prior = TRUE)
```

### Arguments

| | |
|---|---|
| x | Matrix of predictors. |

| | |
|---|---|
| `y` | Response variable. |
| `factorID` | Indication on whether there are categorical variables among the predictors. If factorID= NULL, the predictors are all continuous or have the identifiable categorical variables; If factorID='colnames' or the location numbers of categorical variables, the name or location of variables provided by the user are treated as categorical variables in the linear model. The default factorID is NULL. |
| `candidate` | Method for preparing candidate models. The method of candidate selection differs depending on whether it contains categorical variables. If the predictors are all continuous variables, the candidate= 'H4', the candidate models are on solution paths of 4 common methods, which are lasso, adaptive lasso, SCAD and MCP; the candidate= 'H2', the candidate models are on solution paths of 2 common methods, which are lasso, adaptive lasso; the candidate='H1', the candidate models are on solution paths of the lasso. Otherwise, the candidate= 'CH3', the candidate models are on solution paths of 3 group selection methods in categorical regression, which are group lasso, group MCP and group SCAD, treating the categorical variable as the indivual groups. If candidate= 'H0', the candidate model should be input by the user. When the method is MCV, candidate are not required. |
| `candi_models` | This component is used by the user to input the candidate model matrix. It is a matrix of candidate models, each row of which is a 0/1 indicator vector representing whether each variable is included/excluded in the model. For details see example section. |
| `method` | The method for calculating weights.The method= 'SAICp' is the Smooth-AIC method with the penalty term; the method= 'SBICp' is the Smooth-BIC method with the penalty term; the method= 'ARM' is the Adaptive Regression by Mixing method; the method= 'L1-ARM' is the L1 Adaptive Regression by Mixing method; the method= 'PMA' is the Parsimonious Model Averaging (PMA); the method= 'MCV' is the Cross-validation for Model Averaging (MCV). |
| `n_train` | Size of training set when the weight function is `ARM` or `L1-ARM`. The default value is `n_train=ceiling(n/2)`. |
| `no_rep` | Number of replications when the weight function is `ARM` or `L1-ARM`. The default value is `no_rep=20`. |
| `lambda` | It is the tunning parameter in `PMA`. The default is `log(n)`. |
| `alpha` | Threshold value for marginal correlation test in MCV. The default is `0.05`. |
| `psi` | A positive number to control the influence of the prior weight on the models. The default value is 1. |
| `prior` | Whether to use prior in the weighting function when the method= ('ARM','L1-ARM','SAIC','SBIC'). The default is `TRUE`. |

## Details

See the paper provided in Reference section.

## Value

A `'gma_h'` object is retured. The components are:

| | |
|---|---|
| `weight` | The weight for each candidate model. |
| `weight_se` | The standard error of the weights of the candidate models over the data-splittings under the method= 'ARM' or method='L1-ARM'. |

| | |
|---|---|
| `wbetahat` | The weighted estimation of the coefficients. |
| `betahat` | The coefficients matrix estimated by candidate models. |
| `candi_models` | The candidate models. NOTE: The `weights` and `candi_models` of MCV method, see the article for details. its outputs are mainly for prediction. |

## Examples

```
library(mvtnorm)
n=100
p=200
alpha=1
b <- rep(0,len=p)
for (j in 1:10){
  b[j]=(j^(-alpha-0.5))*sqrt(2*alpha)
}
b=5*b/sum(b)
###################cov setting
Sig= matrix(0,p,p)
rho = 0.5
for(i in 1:p)
{
  for(j in 1:p)
  {
    Sig[i,j]=rho^abs(i-j)
  }
}
#################train data
X=matrix(rmvnorm(n,matrix(0,ncol=1,nrow=p),Sig),nrow=n)
mu0=X%*%b
y=mu0+rt(n,df=3)##t distribution
#the calculating for different method.
##################################################
g1=gma_h(x=X, y,factorID=NULL, candidate='H4',method='SAICp',psi=1)
g2=gma_h(x=X, y, factorID=NULL,candidate='H4',method='ARM',n_train=n/2, no_rep=50, psi=1)
g3=gma_h(x=X, y, factorID=NULL,candidate='H4',method='L1-ARM',n_train=n/2, no_rep=50, psi=1)
g4=gma_h(x=X, y, factorID=NULL,candidate='H4',method='PMA',lambda=log(n))
#####weight
weight=cbind(g1$weight,g2$weight,g3$weight,g4$weight)
weight_se=cbind(g2$weight_se,g3$weight_se)
#####coefficients estimation and prediction
wbetahat=cbind(g1$wbetahat,g2$wbetahat,g3$wbetahat,g4$wbetahat)
Xs=cbind(1,X)
pre=cbind(Xs%*%g1$wbetahat,Xs%*%g2$wbetahat,Xs%*%g3$wbetahat,Xs%*%g4$wbetahat)
se=(pre-matrix(mu0,n,1)%*%rep(1,4))^2
colnames(se)=c('SAICp','ARM','L1-ARM','PMA')
boxplot(se)
apply(se,2,mean)

##############categorical case
library(nnet)
library(CatReg)
library(mvtnorm)
n=100
p=200
sigma0=1
######
```

```
b <- rep(0,len=p) #1*p,beta
##decay
for (j in 1:7){
  b[j]=1/j
}
##################cov setting
Sig= matrix(0,p,p)
rho = 0.5
for(i in 1:p)
{
  for(j in 1:p)
  {
    Sig[i,j]=rho^abs(i-j)
  }
}
x0= UniformDesignMatrix(n, 1, 3) ##categorical variable
X_c=class.ind(as.matrix(x0))
X0=matrix(rmvnorm(n,matrix(0,ncol=1,nrow=p),Sig),nrow=n)
mu0=X_c[,-1]%*%c(2,4)+X0%*%b
y=mu0+rnorm(n,0,sigma0) ##normal distribution
X=cbind(x0,X0)### combine dummy covariable

###categorical regression
isarm=gma_h(x=X,y,factorID=1,candidate='CH3',method='ARM',n_train=n/2,
            no_rep=50,psi=1,prior=TRUE)
pma=gma_h(x=X,y,factorID=1,candidate='CH3',method='PMA',lambda=log(n))

weight=cbind(isarm$weight,pma$weight)
wbeta=cbind(isarm$wbetahat,pma$wbetahat)
```

## uarm *Universal Adaptive Regression by Mixing with low-dimensional inputs*

### Description

Universal Adaptive Regression by Mixing (UARM) provides an adaptive model averaging with both linear models and nonparamatric methods considered as candidates. The nonparamatric methods include Generalized Boosted Regression modeling (GBM), L2Boosting (L2B), Random Forests (RF), Bagging (BAG), and Bayesian Additive Regression Trees (BART) on low-dimensional inputs.

### Usage

```
uarm(x,y,factorID=NULL,candi_models,n_train=ceiling(n/2),no_rep=20,psi=0.1,
    method='L1-UARM',prior=TRUE,p0=0.5)
```

### Arguments

| | |
|---|---|
| x | Matrix of predictors. |
| y | Response variable. |

| | |
|---|---|
| `factorID` | Indication on whether there are categorical variables among the predictors. If factorID= NULL, the predictors are all continuous or have the identifiable categorical variables; If factorID=`'colnames'` or the location numbers of categorical variables, the name or location of variables provided by the user are treated as categorical variables in the linear model. The default factorID is NULL. |
| `candi_models` | Set to 1 for nested subset models in the order given in predictors; set to 2 for all combinations of subsets; input an m*p matrix, where m is the number of models to be combined, and each row of which is a 0/1 indicator vector representing whether each variable is included/excluded in the model. The default is 2. |
| `n_train` | Size of training set when the weight function is `L1-UARM` or `UARM`. The default value is `n_train=ceiling(n/2)`. |
| `no_rep` | Number of replications when the weight function is `L1-UARM` and `UARM`. The default value is `no_rep=50`. |
| `psi` | A positive number to control the influence of the prior weight on the models. The default value is 0.1. |
| `prior` | Whether to use prior in the weighting function. The default is `TRUE`. |
| `method` | The method for calculating weights. Users can choose between `'L1-UARM'` and `'UARM'`. The default is `'L1-UARM'`. |
| `p0` | Prior probabilities of parametric methods. The default value is 0.5. |

## Details

See the paper provided in Reference section.

## Value

A `'uarm'` object is retured. The components are:

| | |
|---|---|
| `weight` | The weight for each candidate model. |
| `weight_se` | The standard error of the weights of the candidate models over the data-splittings. |

## Examples

```
# generate simulation data
n<-50
p<-8
beta<-c(3,1.5,0,0,2,0,0,0)
b0<-1
x<-matrix(rnorm(n*p,0,1),nrow=n,ncol=p)
e<-rnorm(n,0,3)
y<-x%*%beta+b0+e



# compute weight and weight_se for candidate models using L1-UARM
#all subsets candidate models
Lu1<-uarm(x,y,factorID=NULL,candi_models=2,n_train=ceiling(n/2),no_rep=50,psi=0.1,
     method = 'L1-UARM', prior = TRUE, p0 = 0.5)
Lw1<-Lu1$weight
Ls1<-Lu1$weight_se

# compute weight and weight_se for candidate models using UARM
```

```
Lu2<-uarm(x,y,factorID=NULL,candi_models=2,n_train=ceiling(n/2),no_rep=50,psi=0.1,
     method = 'UARM', prior = TRUE, p0 = 0.5)
Luw2<-Lu2$weight
Ls2<-Lu2$weight_se

# output the candidate models
candi_models<-Lu2$candi_models

# simulation with categorical variables
n=100
x1<-rnorm(n)
x2<-rnorm(n)
x3<-rnorm(n)
x4<-factor(sample(1:5,n,replace=T),levels=c(1:5))
X<-data.frame(x1,x2,x3,x4)
Z<-as.matrix(model.matrix(~.-1,data=as.data.frame(X)))[,-4]
mu<-Z%*%c(0.1,0.3,0.5,1,-2,4,-3)
y<-mu+rnorm(n,0,3)

# compute weight for candidate models using MMA with nested subsets candidate models
Lu3<-uarm(X,y,factorID='x4',candi_models=2,n_train=ceiling(n/2),no_rep=50,psi=0.1,
     method='UARM',prior=TRUE,p0=0.5)
Luw3<-Lu3$weight
Ls3<-Lu3$weight_se

# early COVID-19 data in China
data(covid19)
y<-covid19[,1]
x<-covid19[,-1]
n<-length(y)

# compute weight and weight_se for model using L1-UARM
# all subsets candidate models
LC<-uarm(x,y,factorID=NULL,candi_models=2,n_train=ceiling(n/2),no_rep=50,psi=0.1,
    method='L1-UARM',prior=TRUE,p0=0.5)
LCw<-LC$weight
LCs<-LC$weight_se

# compute weight and weight_se for candidate models using UARM
LC2<-uarm(x,y,factorID=NULL,candi_models=2,n_train=ceiling(n/2),no_rep=50,psi=0.1,
     method='UARM',prior=TRUE,p0=0.5)
LCw2<-LC2$weight
LCs2<-LC2$weight_se
```

---

`uarm_h` *Universal Adaptive Regression by Mixing with high-dimensional inputs*

---

## Description

High-dimensional Universal Adaptive Regression by Mixing (uarm_h) provides an adaptive model averaging with both linear and nonparamatric methods considered as candidates (Generalized Boosted Regression modeling (GBM), L2Boosting (L2B) and Random Forests (RF)) on high-dimensional inputs.

### Usage

```
uarm_h(x, y, factorId=NULL,candidate='H4', method='L1-UARM', n_train=ceiling(n/2),
      p0=0.5, no_rep=50, psi=1, prior = TRUE)
```

### Arguments

| | |
|---|---|
| `x` | Matrix of predictors. |
| `y` | Response variable. |
| `factorID` | Indication on whether there are categorical variables among the predictors. If factorID= NULL, the predictors are all continuous or have the identifiable categorical variables; If factorID=‘colnames’ or the location numbers of categorical variables, the name or location of variables provided by the user are treated as categorical variables in the linear model. The default factorID is NULL. |
| `candidate` | Method for preparing candidate models. The method of candidate selection differs depending on whether it contains categorical variables. If the predictors are all continuous variables, the candidate= ‘H4’, the candidate models are on solution paths of 4 common methods, which are lasso, adaptive lasso, SCAD and MCP; the candidate= ‘H2’, the candidate models are on solution paths of 2 common methods, which are lasso, adaptive lasso; the candidate=‘H1’, the candidate models are on solution paths of the lasso. Otherwise, the candidate= ‘CH3’, the candidate models are on solution paths of 3 group selection methods in categorical regression, which are group lasso, group MCP and group SCAD, treating the categorical variable as the indivual groups. If candidate= ‘H0’, the candidate model should be input by the user. When the method is MCV, candidate are not required. |
| `method` | The method for calculating weights. The method= ‘UARM’ is the Universal Adaptive Regression by Mixing method ; the method= ‘L1-UARM’ is the L1 Universal Adaptive Regression by Mixing method; the method= ‘UARM.rf’ is the Universal Adaptive Regression by Mixing method using the random forest to estimate the standard deviation of random error for candidate models; the method= ‘L1-UARM.rf’ is the L1 Universal Adaptive Regression by Mixing method using the random forest to estimate the standard deviation of random error for candidate models. |
| `n_train` | Size of training set when the weight function is `UARM` or `L1-UARM`. The default value is `n_train=ceiling(n/2)`. |
| `no_rep` | Number of replications when the weight function is `UARM` and `L1-UARM`. The default value is `no_rep=20`. |
| `psi` | A positive number to control the improvement of the prior weight. The default value is `psi=1`. |
| `p0` | Prior probability of parametric methods. The default value is `p0=0.5`. |
| `candi_models` | This component is used by the user to input the candidate model matrix. It is a matrix of candidate models, each row of which is a 0/1 indicator vector representing whether each variable is included/excluded in the model. For details see example section. |
| `prior` | Whether to use prior in the weighting function. The default is `TRUE`. |

### Details

See the paper provided in reference section.

## Value

A "uarm_h" object is retured. The components are:

| | |
|---|---|
| `weight` | The weight for each candidate model. |
| `weight_se` | The standard error of candidate models over the each data-splitting. |
| `candi_models` | The candidate models. |

## Examples

```
#####linear case
library(mvtnorm)
n=100
p=200
b=rep(0,len=p)
b[1:10]=2*c(0.878,0.636,-0.257,0.320,0.181,0.174,0.795,-0.523,0.228,-0.727)
X=matrix(rnorm(n*p,0,1),nrow=n,ncol=p)
e=rnorm(n,0,0.5^2)
y=X%*%b + e
#compute the weight and prediction
uarm=uarm_h(x=X,y,factorID=NULL,candidate='H4',n_train=n/2,method='UARM',
            p0=0.5,no_rep=50,psi=1, prior = TRUE)
wy=uarm$weight
uarm$weight_se
l1uarm=uarm_h(x=X,y,factorID=NULL,candidate='H4',n_train=n/2,method='L1-UARM',
              p0=0.5,no_rep=50,psi=1, prior = TRUE)
wy1=l1uarm$weight
l1uarm$weight_se

##compare the nonparametric weights and parametric weights,
##the nonparametric weights almost zero. (the first three weights are nonparametric).
weight_U=cbind(sum(uarm$weight[1:3]),sum(uarm$weight[-c(1:3)]))
weight_L1U=cbind(sum(l1uarm$weight[1:3]),sum(l1uarm$weight[-c(1:3)]))


# Nonlinear Case
library(mvtnorm)
n=100
p=400
sigma0=1
###beta and beta0;decay
b=rep(0,p)
alpha=1
for (j in 1:10){
  b[j]=1/j
}
b=b/sum(b)
b0=5.2 ##nonpara degree
###################cov setting
Sig= matrix(0,p,p)
rho = 0.5
for(i in 1:p)
{
  for(j in 1:p)
  {
    Sig[i,j]=rho^abs(i-j)
  }
}
#################train data
X=matrix(rmvnorm(n,matrix(0,ncol=1,nrow=p),Sig),nrow=n)
mu0=X%*%b+b0*X[,1]*X[,2]
y=mu0+rnorm(n,0,sigma0)##normal distribution
#compute the weight and prediction
l1uarm_rf=uarm_h(x=X,y,factorID=NULL,candidate='H4',n_train=n/2,
                method='L1-UARM.rf',p0=0.5,no_rep=50,psi=1, prior = TRUE)
weight=l1uarm_rf$weight
#####comparr the nonparametric and parametric weights,
#####the parametric weights are small.
weight_U=cbind(sum(l1uarm_rf$weight[1:3]),sum(l1uarm_rf$weight[-c(1:3)]))
weight_se=l1uarm_rf$weight_se


#######categorical regression
library(nnet)
library(CatReg)
library(mvtnorm)
n=100
p=200#
sigma0=1
#######1*p,beta
b <- rep(0,len=p)
##decay
for (j in 1:7){
  b[j]=1/j
}
###################cov setting
Sig= matrix(0,p,p)
rho = 0.5
for(i in 1:p)
{
  for(j in 1:p)
  {
    Sig[i,j]=rho^abs(i-j)
  }
}
x0=UniformDesignMatrix(n, 1, 3)
X_c=class.ind(as.matrix(x0))
X0=matrix(rmvnorm(n,matrix(0,ncol=1,nrow=p),Sig),nrow=n)
mu0=X_c[,-1]%*%c(2,4)+X0%*%b
y=mu0+rnorm(n,0,sigma0)##normal distribution
X=cbind(x0,X0)###combine dummy covariable
###categorical regression
isarm=uarm_h(x=X,y,factorID=1,candidate='CH3',method='UARM',n_train=n/2,
            no_rep=50,psi=1,prior=TRUE)
iluarm=uarm_h(x=X,y,factorID=1,candidate='CH3',method='L1-UARM',n_train=n/2,
             no_rep=50,psi=1,prior=TRUE)
weight=cbind(isarm$weight,iluarm$weight)
```

## `uma.predict` *Prediction for Universal Adaptive Regression by Mixing*

## Description

The predictions based on different MA methods, including SAIC (SAICp), SBIC (SBICp), SFIC, ARM, L1-ARM, UARM, L1-UARM, MMA, JMA, PMA, BMA and MCV.

## Usage

```
uma.predict(x,y,factorID=NULL,newdata,candi_models,weight,method,dim)
```

## Arguments

| | |
|---|---|
| `x` | Matrix of predictors. |
| `y` | Response variable. |
| `factorID` | Indication on whether there are categorical variables among the predictors. If factorID= NULL, the predictors are all continuous or have the identifiable categorical variables; If factorID=`'colnames'` or the location numbers of categorical variables, the name or location of variables provided by the user are treated as categorical variables in the linear model. The default factorID is NULL. |
| `candi_models` | The candidate models under specific method, you can be calculated by gma, gma_h, uarm, and uarm_h functions, as shown in the examples. |
| `newdata` | An optional data frame in which to look for variables with which to predict. If omitted, the fitted values are used. |
| `weight` | The weights of candidate models under specific methods, you can be calculated by gma, gma_h, uarm, and uarm_h functions, as shown in the examples |
| `method` | The method= `'UARM'` is the Universal Adaptive Regression by Mixing method; the method= `'L1-UARM'` is the L1 Universal Adaptive Regression by Mixing method; the method= `'SAIC'` is the Smooth-AIC method; the method= `'SBIC'` is the Smooth-BIC method; the method= `'SAICp'` is the Smooth-AIC method with the penalty term; the method= `'SBICp'` is the Smooth-BIC method with the penalty term; the method= `'SFIC'` is the Smooth-FIC method; the method= `'ARM'` is the Adaptive Regression by Mixing method; the method= `'L1-ARM'` is the L1 Adaptive Regression by Mixing method; the method= `'MMA'` is the Mallows Model Averaging (MMA); the method= `'JMA'` is the Jackknife Model Averaging (JMA); the method= `'UARM.rf'` is the Universal Adaptive Regression by Mixing method using the random forest to estimate the standard deviation of random error for candidate models; the method= `'L1-UARM.rf'` is the L1 Universal Adaptive Regression by Mixing method using the random forest to estimate the standard deviation of random error for candidate models; the method= `'PMA'` is the Parsimonious Model Averaging; the method= `'MCV'` is the Cross-validation for Model Averaging (MCV). |
| `dim` | High-dimensional or low-dimensional methods are used for prediction. If dim =`'H'`, high-dimensional methods are used; otherwise, low-dimensional methods are used. |

## Details

See the paper provided in Reference section.

## Value

A `'uma.predict'` object is retured. The components is:

| | |
|---|---|
| `pre_out` | The prediction by given method. |

## Examples

```
### low dimension

# generate simulation data
n<-50
p<-8
beta<-c(3,1.5,0,0,2,0,0,0)
b0<-1
x<-matrix(rnorm(n*p,0,1),nrow=n,ncol=p)
e<-rnorm(n,0,3)
y<-x%*%beta+b0+e

# user supplied candidate models
candi_models<-rbind(c(0,0,0,0,0,0,0,1),
                    c(0,1,0,0,0,0,0,1),
                    c(0,1,1,1,0,0,0,1),
                    c(0,1,1,0,0,0,0,1),
                    c(1,1,0,1,1,0,0,0),
                    c(1,1,0,0,1,0,0,0))

# compute weight for candidate models using L1-UARM
weightL<-uarm(x,y,factorID=NULL,candi_models=candi_models,n_train=ceiling(n/2),
         no_rep=50,psi=1,method='L1-UARM',prior=TRUE,p0=0.5)$weight

# compute the prediction by method L1-UARM
luma.predict<-uma.predict(x,y,factorID=NULL,newdata=x,candi_models=candi_models,
              weight=weightL,method='L1-UARM',dim='L')$pre_out

# early COVID-19 data in China
data(covid19)
y<-covid19[,1]
x<-covid19[,-1]
n<-length(y)

# compute the predicts for L1-UARM, MMA and SFIC
# user supplied all subsets candidate models
Cl1uarmw<-uarm(x,y,factorID=NULL,candi_models=2,n_train=ceiling(n/2),no_rep=50,
          psi=1,method='L1-UARM',prior=TRUE,p0=0.5)$weight
Cmmaw<-gma(x,y,factorID=NULL,method='MMA',candi_models=2)$weight
Csficw<-gma(x,y,factorID=NULL,method='SFIC',candi_models=2)$weight

# compute the prediction by methods L1-UARM, MMA, SFIC and BMA
cl1uarm.predict<-uma.predict(x,y,factorID=NULL,newdata=x,candi_models=2,
                             weight=Cl1uarmw,method='L1-UARM',dim='L')$pre_out
cmma.predict<-uma.predict(x,y,factorID=NULL,newdata=x,candi_models=2,
                          weight=Cmmaw,method='MMA',dim='L')$pre_out
csfic.predict<-uma.predict(x,y,factorID=NULL,newdata=x,candi_models=2,
                           weight=Cmmaw,method='SFIC',dim='L')$pre_out

#The BMA prediction does not depend on candidate models
cbma.predict<-uma.predict(x,y,factorID=NULL,newdata=x,candi_models=2,
                          method='BMA',dim='L')$pre_out



###high dimension
```

```
library(mvtnorm)
n1=100;n2=1000
p=200
sigma0=1
######
b=rep(0,len=p) #1*p,beta
for(j in 1:12){
  b[j]=2/j
}
# cov setting
Sig = matrix(0,p,p)
rho = 0.5
for(i in 1:p)
{
  for(j in 1:p)
  {
    Sig[i,j] = rho^abs(i-j)
  }
}
# new data
X=matrix(rmvnorm(n1,matrix(0,ncol=1,nrow=p),Sig),nrow=n1)
X_test=matrix(rmvnorm(n2,matrix(0,ncol=1,nrow=p),Sig),nrow=n2)
mu0=X%*%b
mu_test=X_test%*%b
y=mu0+rnorm(n1,0,sigma0)##normal distribution
##########the prediction on each methods
g1=gma_h(x=X,y,factorID=NULL,candidate='H4',method='SBICp',psi=1, prior=TRUE)
pre_out1=uma.predict(x=X,y,factorID=NULL,newdata=X_test,method='SBICp',weight=g1$weight,
                    candi_models=g1$candi_models,dim='H')$pre_out

g2=gma_h(x=X,y,factorID=NULL,candidate='H4',method='PMA',lambda=log(n1))
pre_out2=uma.predict(x=X,y,factorID=NULL,newdata=X_test,method='PMA',weight=g2$weight,
                     candi_models=g2$candi_models,dim='H')$pre_out

g3=gma_h(x=X,y,factorID=NULL,method='MCV',alpha = 0.05)
pre_out3=uma.predict(x=X,y,factorID=NULL,newdata=X_test,method='MCV',weight=g3$weight,
                     candi_models=g3$candi_models,dim='H')$pre_out

g4=gma_h(x=X, y, factorID=NULL,candidate='H4',method='ARM',n_train=n1/2, no_rep=50, psi=1)
pre_out4=uma.predict(x=X,y,factorID=NULL,newdata=X_test,method='ARM',weight=g4$weight,
                     candi_models=g4$candi_models,dim='H')$pre_out

g5=gma_h(x=X, y, factorID=NULL,candidate='H4',method='L1-ARM',n_train=n1/2, no_rep=50, psi=1)
pre_out5=uma.predict(x=X,y,factorID=NULL,newdata=X_test,method='L1-ARM',weight=g5$weight,
                     candi_models=g5$candi_models,dim='H')$pre_out

g6=uarm_h(x=X,y,factorID=NULL,candidate='H4',n_train=n1/2,method='UARM',
            no_rep=50,p0=0.5,psi=1, prior = TRUE)
pre_out6=uma.predict(x=X,y,factorID=NULL,newdata=X_test,method='UARM',weight=g6$weight,
                candi_models=g6$candi_models,dim='H')$pre_out

g7=uarm_h(x=X,y,factorID=NULL,candidate='H4',n_train=n1/2,method='L1-UARM',
          no_rep=50,p0=0.5,psi=1, prior = TRUE)
pre_out7=uma.predict(x=X,y,factorID=NULL,newdata=X_test,method='L1-UARM',weight=g7$weight,
                candi_models=g7$candi_models,dim='H')$pre_out

####the performance of different methods
```

```
pre_out=cbind(pre_out1,pre_out2,pre_out3,pre_out4,pre_out5,pre_out6,pre_out7)
se=(pre_out-matrix(mu_test,n2,1)%*%rep(1,7))^2
colnames(se)=c('SBICp','PMA','MCV','ARM','L1-ARM','UARM','L1-UARM')
Pre=apply(se,2,mean)
Pre_se=apply(se,2,sd)
```

# Index