Package ‘DTRlearn’

April 6, 2018

Type Package
Title Learning Algorithms for Dynamic Treatment Regimes
Version 1.3
Date 2018-4-05
Author Ying Liu, Yuanjia Wang, Donglin Zeng
Maintainer Ying Liu <summerying1@gmail.com>
Depends kernlab, MASS, glmnet, ggplot2
Description Dynamic treatment regimens (DTRs) are sequential decision rules tailored at each stage by time-varying subject-specific features and intermediate outcomes observed in previous stages. This package implements three methods: O-learning (Zhao et al. 2012, 2014), Q-learning (Murphy et al. 2007; Zhao et al. 2009) and P-learning (Liu et al. 2014, 2015) to estimate the optimal DTRs.
License GPL-2
NeedsCompilation no
Repository CRAN
Date/Publication 2018-04-05 22:03:46 UTC

R topics documented:

DTRlearn-package ......................................................... 2
make_2classification .................................................. 3
make_classification ..................................................... 4
Olearning ................................................................. 6
Olearning_Single ......................................................... 7
Plearning ................................................................. 9
plot.linearcl ............................................................. 11
plot.qlearn ............................................................. 12
predict.linearcl ......................................................... 13
predict.qlearn .......................................................... 14
predict.rbfcl ........................................................... 15
Qlearning ............................................................... 16
Qlearning_Single ....................................................... 17
wsvm ................................................................. 18
Description

Dynamic treatment regimens (DTRs) are sequential decision rules tailored at each stage by time-varying subject-specific features and intermediate outcomes observed in previous stages. For many complex chronic disorders, DTRs operationalize the sequential process of medical decision-making. Sequential Multiple Assignment Randomized Trials (SMARTs) are proposed to best construct DTRs which offer a causal interpretation of their comparisons through randomization at each critical decision point. Machine learning methods such as O-learning (Zhao et al. 2012, 2014), Q-learning (Murphy et al. 2007, Zhao et al. 2009) and P-learning (Liu et al. 2014, 2015) have been proposed to estimate the optimal individualized treatment from a SMART.

This package implements these 3 main types of algorithms to estimate the optimal DTR. The algorithms consider a continuous outcome (the larger indicates better clinical results), and allow two treatment choices for each stage coded by 1 and -1.

Details

Package: DTRlearn
Type: Package
Version: 1.1
Date: 2015-10-26
License: GPL-2

Author(s)

Ying Liu
Yuanjia Wang
Donglin Zeng

References


---

**Data Simulation for 2 stages**

**Description**

It generates simulated dataset to test multiple stage learning algorithms. The outcomes are generated based on a pattern mixture model using a latent variable with 4 categories. For each category, X has a multivariate normal distribution and each category is assigned a vector of optimal treatments V. Specifically, we generate centroids of the classes from a multivariate normal distribution mean 0 and std 5. We add the centroids to the first pinfo dimension of the vectors of feature variables X simulated from multivariate normal distribution with pinfo+pnoise dimensions.

Then we assign optimal treatments $y = (A_1^*, A_2^*)$ from (1,1),(-1,1),(1,-1),(-1,-1) to each latent category. The observed treatment assignments $A = (A_1, A_2)$ are completely random to be 1 and -1 with probability 0.5, and the outcomes are generated as: $R_1=0, R_2= A' y + N(0,1)$. Therefore the mean optimal outcome $R_1+R_2$ is 2 when the treatment assignments are equal to the optimal treatment for a given a latent group in both stages.

**Usage**

`make_2classification(n_cluster, pinfo, pnoise, n_sample, centroids = 0)`

**Arguments**

- **n_cluster**: number of clusters.
- **pinfo**: number of informative variables, dimensions of the centroids related to the latent class of the sample.
- **pnoise**: number of noise variable.
- **n_sample**: sample size.
- **centroids**: For a training set, do not assign centroids, the centroids are generated randomly by the function. For a testing set, ones want to assign the same set of centroids as the training set. It is a matrix of dimension n_cluster by p.
make_classification

**Value**

- **X**: Feature variable matrix, it is a n_sample by pinfo+pnoise matrix generated from multivariate normal distribution. Where the noises are with mean 0 and std 1. The informative variables are shifted to centered at the randomly generate centroids.

- **A**: List of 2, A[[1]] and A[[2]] are the treatment assignment vectors for stage 1 and 2.

- **y**: List of 2, y[[1]] and y[[2]] are the true optimal treatment vectors for stage 1 and 2.

- **R**: List of 2, R[[1]] is vector of n_sample zeros (this is the simplified case where the intermediate outcomes are 0), R[[2]] is the final outcomes vector.

**See Also**

- make_classification

**Examples**

```r
n_cluster=5
pinfo=10
pnoise=10
n_sample=50
example2=make_2classification(n_cluster,pinfo,pnoise,n_sample)
pi=list()
pi[[2]]=pi[[1]]=rep(1,n_sample)
set.seed(3)
model0=0learning(example2$X,example2$A,example2$R,n_sample,2,pi)
modelP=Plearning(example2$X,example2$A,example2$R,n_sample,2,pi)
modelQ=Qlearning(example2$X,example2$A,example2$R,2)
```

**Description**

It generates simulated datasets to test single stage DTR learning algorithms. The outcomes are generated based on a pattern mixture model using a latent variable with 2 categories. Category 1 has the optimal treatment y=1, and category 2 has y=-1. The feature variables X has a multivariate normal distribution. Specifically, we generate centroids of the classes from a multivariate normal distribution mean 0 and std 5. We add the centroids to the first pinfo dimension of the vectors of feature variables X simulated from multivariate normal distribution with pinfo+pnoise dimensions. The observed treatment assignments A are completely random to be 1 and -1 with probability 0.5, and the outcomes are generated as: $R_1 = 0, R_2 = 1.5A * y + N(0, 1)$. 

```
**Usage**

```
make_classification(n_cluster, pinfo, pnoise, n_sample, centroids = 0)
```

**Arguments**

- `n_cluster` number of clusters.
- `pinfo` number of informative variables, dimensions of the centroids related to the latent class of the sample.
- `pnoise` number of noise variables.
- `n_sample` sample size
- `centroids` For a training set, do not assign centroids, the centroids are generated randomly by the function. For a testing set, one wants to assign the same set of centroids as the training set. It is a matrix of dimension n_cluster by p.

**Value**

- `X` The feature variable matrix, it is a n_sample by pinfo+pnoise matrix generated from multivariate normal distribution. Where the noises are with mean 0 and std 1. The informative variables are shifted to centered at the randomly generated centroids.
- `A` The treatment assignment vector
- `y` The true optimal treatment vector
- `R` Outcomes vector
- `centroids` are from pinfo dimensional multivariate normal distribution.

**Author(s)**

Ying Liu yl2802@cumn.columbia.edu

**References**

This function borrows idea from a python comparable function make_classification in scikit_learn


**See Also**

- `make_2classification` for generating simulation data for 2 stages

**Examples**

```
n_cluster=10
pinfo=10
pnoise=20
example1=make_classification(n_cluster,pinfo,pnoise,100)
test=make_classification(n_cluster,pinfo,pnoise,100,example1$centroids)
model1=Olearning_Single(example1$X,example1$A,example1$R)
```
Atp=predict(model1,test$X)
V1=mean(test$R[Atp==test$A])

model2=wsvm(example$X,example$A,example$Y,'rbf',0.05)
Atp=predict(model2,test$X)
V2=mean(test$R[Atp==test$A])

# in this very non-linear case, one can compare V1 with V2 (the empirical value on testing set),
# and can see the better of model2 using rbf kernel
# to model1 using linear kernel.
# the true optimal value here is 1.5

---

**Olearning**

**Multiple stage Improved Olearning**

**Description**

This function implements multiple stage O-learning (with improved single stage O-learning) to find optimal DTR by backward induction.

**Usage**

```r
Olearning(X,AA,RR,n,K,pi, pentype="lasso",kernel="linear",
  sigma=c(0.03,0.05,0.07),clinear = 2^(-2:2), m = 4, e = 1e-05)
```

**Arguments**

- **X** is either a matrix shared among all stages; or list of feature matrices, where feature matrices from different stages can have different dimensions.
- **AA** a list of K, each element A[[i]] is the treatment assignment vector for stage i.
- **RR** a list of K, each element R[[i]] is the outcome vector for stage i.
- **n** sample size
- **K** number of stages
- **pi** a list of K, the i’th element is the randomization probability at stage i
- **pentype** the type of regression used to take residual, ‘lasso’ is the default, using lasso regression; ‘LSE’ is the ordinary least square regression. as in the function `wsvm`
- **kernel** The choice of kernel for Improved O-learning, default is ‘linear’, can also be ‘rbf’
- **sigma** if kernel=’rbf’, sigma is the grid of tuning parameter for ’rbf’ kernel to run cross validation to choose from, the default is (0.03, 0.05, 0.07)
- **clinear** is grid of tuning parameter for `wsvm`, which cross validation was run to choose from. the default is 2^(-2:2)
- **m** number of folds in cross validation for `Olearning_Single`
- **e** The rounding error for computing bias in `wsvm`
**Olearning_Single**

**Value**

models a list of models of class ‘linearcl’

**References**


**See Also**

Olearning_Single

**Examples**

```r
n_cluster=10
pinfo=10
pnoise=20
e=x=make_2classification(n_cluster,pinfo,pnoise,200)
test=make_2classification(n_cluster,pinfo,pnoise,200,example2$centroids)
pi=list()
pi[[0]]=pi[[1]]=rep(1,200)
modelO=Olearning(example2$x,example2$A,example2$R,200,2,pi)
```

---

**Olearning_Single**  
*Improved single stage O-learning with cross validation*

**Description**

Improved outcome weighted learning, first take residuals; and then use cross validation to choose best tuning parameter for `wsvm`. Return the O-learning models with best tuning parameters. Improving from Zhao 2012, the improved outcome weighted learning first take main effect out by regression; the weights are absolute value of the residual; more details can be found in Liu et al. (2015).

**Usage**

```r
Olearning_Single(H,A,R2,pi=rep(1, n),pentype ="lasso",kernel="linear",
sigma = c(0.03, 0.05, 0.07),clinear = 2^(-2:2), m = 4,e = 1e-05)
```
Arguments

H  
\[ \text{a n by p matrix, n is the sample size, p is the number of feature variables.} \]

A  
\[ \text{a vector of n entries coded 1 and -1 for the treatment assignments} \]

R2  
\[ \text{a vector of outcome variable, larger is more desirable.} \]

pi  
\[ \text{a vector of randomization probability } P(A|X), \text{ or the estimated observed probability.} \]

pentype  
\[ \text{the type of regression used to take residual, 'lasso' is the default (lasso regression); 'LSE' is the ordinary least square regression.} \]

kernel  
\[ \text{kernel function for weighted SVM, can be 'linear' or 'rbf' (radial basis kernel), default is 'linear'. When 'rbf' is specified, one can specify the sigma parameter for radial basis kernel.} \]

sigma  
\[ \text{a grid of tuning parameter sigma for 'rbf' kernel for cross validation, when kernel='rbf', the default is } c(0.03, 0.05, 0.07) \]

clinear  
\[ \text{a grid of tuning parameter C for cross validation, the default is } 2^{(-2 : 2)}. \text{ C is tuning parameter as defined in wsvm} \]

m  
\[ \text{folds of cross validation for choosing tuning parameters C and sigma. If 'lasso' is specified for 'pentype', m is also the folds CV for cv.glmnet in the step of taking residual.} \]

e  
\[ \text{the rounding error when computing bias in wsvm} \]

Value

It returns model estimated from wsvm with the best tuning parameters picked by cross validation. If kernel 'linear' is specified, it returns an object of class 'linearcl', and it is a list include the following elements:

alpha1  
\[ \text{the scaled solution for the dual problem: } alpha1_i = \alpha_i A_i wR_i \]

bias  
\[ \text{the intercept } \beta_0 \text{ in } f(X) = \beta_0 + X\beta. \]

fit  
\[ \text{a vector of estimated values for } f(x) \text{ in training data, } fit = bias + X\beta = bias + X' \cdot alpha1. \]

beta  
\[ \text{The coefficients } \beta \text{ for linear SVM, } f(X) = bias + X\beta. \]

If kernel 'rbf' is specified, it returns an object of class 'rbfcl', and it is a list include the following elements:

alpha1  
\[ \text{the scaled solution for the dual problem: } alpha1_i = \alpha_i A_i wR_i \text{ and } X\beta = K(X, X) \cdot alpha1 \]

bias  
\[ \text{the intercept } \beta_0 \text{ in } f(X) = \beta_0 + h(X)\beta. \]

fit  
\[ \text{a vector of estimated values for } f(x) \text{ in training data, } fit = \beta_0 + h(X)\beta = bias + K(X, X) \cdot alpha1. \]

sigma  
\[ \text{the bandwidth parameter for the rbf kernel} \]

X  
\[ \text{the matrix of training feature variable} \]
This is the Plearning to estimate optimal multistage DTR. It implements improved Olearning to estimate optimal treatment rules for each stage backwardly. And it also borrows idea from Q-learning to utilize the estimated optimal outcomes for later stages.

Usage

Plearning(X, AA, RR, n, K, pi, petype = "lasso", kernel = "linear", sigma = c(0.03, 0.05, 0.07), clinear = 2^(-2:2), m = 4, e = 1e-05)
### Arguments

- **X**
  - is either a matrix shared among all stages; or list of feature matrices, where feature matrices from different stages can have different dimensions.
- **AA**
  - a list of K, each element A[[i]] is the treatment assignment vector for stage i.
- **RR**
  - a list of K, each element R[[i]] is the outcome vector for stage i.
- **n**
  - sample size
- **K**
  - number of stages
- **pi**
  - a list of K, the i’th element is the randomization probability at stage i
- **pentype**
  - the type of regression used to take residual, 'lasso' is the default, using lasso regression; 'LSE' is the ordinary least square regression. as in the function `wsvm`
- **clinear**
  - is grid of tuning parameter for `wsvm`, and we use cross validation to choose the tuning parameter here.
- **m**
  - number of folds in cross validation for `olearning_single`
- **e**
  - The rounding error for to compute bias in `wsvm`
- **kernel**
  - The choice of kernel for Improved O-learning, default is 'linear', can also be 'rbf'
- **sigma**
  - if 'rbf' is chosen for kernel, the grid of sigma to search by cross validation.

### Value

- **models**
  - a list of models of class 'linearcl'

### Author(s)

- Ying Liu yl2802@cumc.columbia.edu [http://www.columbia.edu/~yl2802/](http://www.columbia.edu/~yl2802/)

### References


### See Also

- `olearning_single`, `qlearning_single`

### Examples

```r
n_cluster=10
pinfo=10
pnoise=20
example2=make_2classification(n_cluster,pinfo,pnoise,100)
pi=list()
```
Description

The function visualize the contribution of each feature variable by plotting a selected subset of standardized coefficients from SVM with linear kernel, the coefficients are standardized dividing by the L2 norm of the subvector.

Usage

```r
## S3 method for class 'linearcl'
plot(x, index = NULL, names = NULL, ylab = "std coefficients", xlab = "", col = "gray", ...)
```

Arguments

- `x`: a model with class 'linearcl'.
- `index`: is the vector of indices of variables to plot.
- `names`: is the vector of names of each variables to appear in the plot. If not specified, the names would be V1-Vp.
- `ylab`: The label for the y axis. Default is "std coefficients".
- `xlab`: The label for the x axis.
- `col`: color to fill the bars in the plot.
- `...`: further arguments passed to or from other methods.

Author(s)

Ying Liu yl2802@cumc.columbia.edu [http://www.columbia.edu/~yl2802/](http://www.columbia.edu/~yl2802/)

See Also

- `Olearning_Single`
- `wsvm`

Examples

```r
n=200
A=2*runif(n,1,5)-1
p=20
mu=numeric(p)
Sigma=as.matrix(diag(p))
X=mvnrnorm(n,mu,Sigma)
R=X[,1:3]*X[,1,-2]+X[,3:5]*X[,1,-2]*A+rnorm(n)
model1=Olearning_Single(X,A,R)
plot(model1)
```
plot.qlearn  

Plot the linear coefficients of interaction

Description

The function plots the standardized coefficients from linear regression, i.e. it divides the coefficients for a selected set of variables by the L2 norm.

Usage

```r
## S3 method for class 'qlearn'
plot(x, index = NULL, names = NULL,
     ylab = "std coefficients", xlab = ", col = "gray", ...)
```

Arguments

- `x`: a model of class 'qlearn'.
- `index`: is the vector of indices of variables to plot.
- `names`: is the vector of names of each variables to appear in the plot. If not specified, the names would be V1-Vp.
- `ylab`: The label for the y axis. Default is "std coefficients".
- `xlab`: The label for the x axis.
- `col`: color to fill the bars in the plot.
- `...`: further arguments passed to or from other methods.

Author(s)

Ying Liu yl2802@cumc.columbia.edu [http://www.columbia.edu/~ylRXPRO](http://www.columbia.edu/~ylRXPRO)

See Also

`Qlearning_Single`

Examples

```r
n=200
A=2*rbinom(n,1,0.5)-1
p=20
mu=numeric(p)
Sigma=diag(p)
X=mvnorm(n,mu,Sigma)
R=X[,1:3]%*%c(1,1,-2)+X[,3:5]%*%c(1,1,-2)*A+rnorm(n)
modelQ=Qlearning_Single(X,A,R)
plot(modelQ)
```
**predict.linearcl**  

**Description**  
This function predicts the optimal treatments with model of class 'linearcl', which is estimated by *wsvm* with 'linear' kernel.

**Usage**  
```r  
## S3 method for class 'linearcl'  
predict(object, x, ...)  
```

**Arguments**  
- `object`: model of class 'linearcl'
- `x`: a matrix of feature variables, n by p
- `...`: further arguments passed to or from other methods.

**Value**  
a vector of optimal treatments, each entry is for a row in x, the matrix of new feature variables.

**Author(s)**  
Ying Liu

**See Also**  
*wsvm*

**Examples**

```r  
n=200  
A=2*rbinom(n,1,0.5)-1  
p=20  
mu=numeric(p)  
Sigma=diag(p)  
# feature variable is multi variate normal  
X=mvrnorm(n,mu,Sigma)  
# the outcome is generated where the true optimal treatment  
is sign of the interaction term(of treatment and feature)  
R=X[,1:3]*X[,1:3]+X[,3:5]*X[,3:5]+(1,1,-2)*A+rnorm(n)  

# linear SVM  
model1=wsvm(X,A,R)  
m=100  
Xtest=mvrnorm(m,mu,Sigma)  
predict(model1,Xtest)  
```
predict.qlearn

Predict optimal treatment by Qlearning

Description

This function predicts optimal treatments for a given Q-learning model of class 'qlearn'.

Usage

## S3 method for class 'qlearn'
predict(object, x,...)

Arguments

- object: a model with class 'qlearn'
- x: a matrix of feature variables, has p columns that is p dimensions of feature variables
- ...: further arguments passed to or from other methods.

Value

A vector of optimal treatments, each entry is for a row in x, the matrix of new feature variables.

Author(s)

Ying Liu

See Also

- Qlearning_Single

Examples

```r
n=200
A=2*rbinom(n,1,0.5)-1
p=10
mu=numeric(p)
Sigma=diag(p)
X=rmvnorm(n,mu,Sigma)
R=X[,1:3]*c(1,1,-2)+X[,3:5]*c(1,1,-2)*A+rnorm(n)

model1=Qlearning_Single(X,A,R)
m=100
Xtest=rmvnorm(m,mu,Sigma)
predict(model1,Xtest)
```
Description

It predicts the optimal treatments with model of class 'rbfcl', which is estimated by \texttt{wsvm} with 'rbf' kernel.

Usage

\begin{verbatim}
## S3 method for class 'rbfcl'
predict(object, x,...)
\end{verbatim}

Arguments

- \texttt{object} model of class 'rbfcl'
- \texttt{x} a matrix of feature variables, n by p, p is the dimension of feature variables
- \texttt{...} further arguments passed to or from other methods.

Value

a vector of optimal treatments, each entry is for a row in \texttt{x}, the matrix of new feature variables.

Author(s)

Ying Liu yl2802@cumc.columbia.edu \url{http://www.columbia.edu/~ylRXPRO}

See Also

\texttt{wsvm}, \texttt{Olearning_Single}

Examples

\begin{verbatim}
n_cluster=5
pinfo=10
pnoise=10
n=50
m=20
example1=make_classification(n_cluster,pinfo,pnoise,n)
test=make_classification(n_cluster,pinfo,pnoise,m,example1$centroids)
model1=Olearning_Single(example1$X,example1$A,example1$R)
Atp=predict(model1,test$X)
V1=mean(test$R[Atp==test$A])

model2=wsvm(example1$X,example1$A,example1$R,'rbf',0.05)
model2=Olearning_Single(example1$X,example1$A,example1$R,kernel='rbf',m=3)
Atp=predict(model2,test$X)
\end{verbatim}
Qlearning

Description
This function implements multiple stage Q-learning.

Usage
Qlearning(X, AA, RR, K, pentype="lasso", m=4)

Arguments
X is either a matrix shared among all stages; or a list of feature matrices, where feature matrices from different stages can have different dimensions.
AA a list of K, each element A[[i]] is the vector of treatment assignments for stage i.
RR a list of K, each element R[[i]] is the outcome vector for stage i.
K number of stages
pentype the type of regression implemented in Q-learning, the default is 'lasso', another choice is 'LSE'
m number of folds of cross validation for in cv.glmnet in regression model when 'lasso' is selected

Value
It returns a list of K models with class 'qlearn'.

Author(s)
Ying Liu

References


Qlearning_Single

See Also
- Qlearning_Single

Examples

```r
n_cluster=10
pinfo=10
pnoise=20
example2=make_2classification(n_cluster,pinfo,pnoise,200)
test=make_2classification(n_cluster,pinfo,pnoise,200,example2$centroids)
pi=list()
pi[[2]]=pi[[1]]=rep(1,200)
modelQ=Qlearning(example2$X,example2$A,example2$R,2)
```

---

Qlearning_Single  
**Single Stage Q learning**

Description

It implements single stage Q-learning. Q-learning estimates optimal treatment option by fitting a regression model with treatment, feature variable and their interactions. The optimal treatment option is the the sign of the interaction term which maximize the predicted value from the regression model.

Usage

```r
Qlearning_Single(H, A, R, pentype = "lasso",m=4)
```

Arguments

- **H**: a n by p matrix, n is the sample size, p is the number of feature variables.
- **A**: a vector of treatment assignments coded 1 and -1.
- **R**: a vector of outcomes, larger is more desirable.
- **pentype**: The type of regression in Q-learning. 'lasso' is the default lasso regression; 'LSE' is the ordinary least square.
- **m**: needed when pentype='lasso', the number of folds in cross validation for picking tuning parameter for lasso in `cv.glmnet`

Value

It returns a class of 'qlearn', that consists of two components:

- **co**: the coefficient of the regression model, it is a 2p+2 vector. The design matrix is \(X=(\text{Intercept}, H, A, \text{diag}(A)*H)\)
- **Q**: The predicted optimal outcome from the regression model
Author(s)
Ying Liu yl2802@cumc.columbia.edu http://www.columbia.edu/~ylRXPRO

References


See Also
qlearning

Examples
n=200
A=2*rbinom(n,1,0.5)-1
p=20
mu=numeric(p)
Sigma=diag(p)
X=mvrnorm(n,mu,Sigma)
R=X[1:3]%*%c(1,1,-2)+X[3:5]%*%c(1,1,-2)*A+rnorm(n)
modelQ=Qlearning_Single(X,A,R)

Description
This function transforms a weighted SVM problem into its dual form, and solves it by the quadratic programing applying ipop in package kernlab. This is the core step in the improved single stage outcome weighted learning (Liu et.al.2015), and now it can takes positive and negative outcomes as an improvement from Zhao et.al.2012. The function wsvm can implement weighted SVM with gaussian or linear kernel. O-learning target at maximizing the expected value function by transforming it into a classification problem, mapping the feature variables X to the optimal treatment choice, which is \( \text{sign}(f(x)) = \text{sign}(h(x)\beta + \beta_0) \). The original problem weighted SVM problem is

\[
\text{min} \left\{ \frac{1}{2} \|\beta\|^2 + C \sum_{i=1}^{N} |w_i| \right\} \text{subject to } \xi_i \geq 0, \text{sign}(w_i A_i)(X_i\beta + \beta_0) \geq 1 - \xi_i
\]

The transformed dual problem is

\[
\text{min}_a 0.5 \sum_i \sum_j \alpha_i w_i A_i X_i^T X_j A_j w_j \alpha_j - \sum_i |w_i| \alpha_i \text{ subject to } 0 \leq \alpha_i \leq C, \text{ and } \sum_i \alpha_i w_i R_i A_i = 0.
\]
Usage

\texttt{wsvm(X, A, wR, kernel = "linear", sigma = 0.05, C = 1, e = 1e-05)}

Arguments

- \texttt{X} a \( n \) by \( p \) matrix, \( n \) is the sample size, \( p \) is the number of feature variables.
- \texttt{A} a vector of the treatment assignments coded by 1 and -1.
- \texttt{wR} a vector of weighted outcomes computed before hand, it is the outcome \( R_i \) weighted by inverse randomization or observational probability. \( wR_i = R_i / \pi_i \).
- \texttt{kernel} the kernel function for weighted SVM, can be 'linear' or 'rbf' (radial basis kernel), default is 'linear'. When 'rbf' is specified, one can specify the sigma parameter for radial basis kernel.
- \texttt{sigma} the tuning parameter for 'rbf' kernel, this is from \texttt{rbfdot} function in package \texttt{kernlab}, \( Kernel(x, y) = \exp(-\text{sigma} \cdot (x-y)^2) \).
- \texttt{C} the tuning parameter for weighted SVM in the primal problem \( \min \frac{1}{2} ||\beta||^2 + \sum_{i=1}^{N} wR_i |\xi_i| \) subject to \( \xi_i \geq 0 \), \( \text{sign}(wR_i)A_i(X_i\beta + \beta_0 \geq 1 - \xi_i) \).
- \texttt{e} the rounding error when computing the bias, for the varaibles \( \alpha_i \)'s in the dual problem, if \( |\alpha_i| < e \), we consider \( \alpha = 0 \).

Value

If kernel 'linear' is specified, it returns an object of class 'linearcl', and it is a list include the following elements:

- \texttt{alpha1} the scaled solution for the dual problem: \( alpha1_i = \alpha_iA_iwR_i \).
- \texttt{bias} the intercept \( \beta_0 \) in \( f(X) = \beta_0 + X\beta \).
- \texttt{fit} a vector of estimated values for \( \hat{f}(x) \) in training data, \( \text{fit} = \text{bias} + X\beta = \text{bias} + X \ast X' \ast \text{alpha1} \).
- \texttt{beta} The coefficients \( \beta \) for linear SVM, \( f(X) = \text{bias} + X\beta \).

If kernel 'rbf' is specified, it returns an object of class 'rbfcl', and it is a list include the following elements:

- \texttt{alpha1} the scaled solution for the dual problem: \( alpha1_i = \alpha_iA_iwR_i \) and \( X\beta = K(X, X) \ast alpha1 \).
- \texttt{bias} the intercept \( \beta_0 \) in \( f(X) = \beta_0 + h(X)\beta \).
- \texttt{fit} a vector of estimated values for \( \hat{f}(x) \) in training data, \( \text{fit} = \beta_0 + h(X)\beta = \text{bias} + K(X, X) \ast \text{alpha1} \).
- \texttt{sigma} the bandwidth parameter for the rbf kernel
- \texttt{X} the matrix of training feature variable

Author(s)

Ying Liu yl2802@cumc.columbia.edu \url{http://www.columbia.edu/~yl2802/}
References


See Also

`plot.linearcl`, `predict.linearcl`, `predict.rbfcl`

Examples

```r
# generating random assigned treatment vector A
n=200
A=2*runif(n,1,0.5)-1
p=20
mu=numeric(p)
Sigma=diag(p)
# feature variable is multivariate normal distributed
X=mvrnorm(n,mu,Sigma)
# the outcome is generated where the true optimal treatment
# is sign of the interaction term of treatment and feature
R=X[,1:3]*X(1,1,-2)+X[,3:5]*X(1,1,-2)*A+rnorm(n)

# linear SVM
model1=wsvm(X,A,R)
# Check the total number that agree with the true optimal treatment among n=200 patients
sum(sign(model1$fit)==sign(X[,3:5]*X(1,1,-2)))

# SVM with rbf kernel and sigma=0.05
model2=wsvm(X,A,R,'rbf',0.05)
# Check the total number that agree with the true optimal treatment among n=200 patients
sum(sign(model2$fit)==sign(X[,3:5]*X(1,1,-2)))
```
Index

+Topic models
  Olearning, 6
  Qlearning, 16

cv.glmnet, 8, 16, 17

DTRlearn (DTRlearn-package), 2
DTRlearn-package, 2

ipop, 18

make_2classification, 3, 5
make_classification, 4, 4

Olearning, 6, 9
Olearning_Single, 6, 7, 7, 10, 11, 15

Plearning, 9
plot.linearcl, 11, 20
plot.qlearn, 12
predict.linearcl, 13, 20
predict.qlearn, 14
predict.rbfcl, 15, 20

Qlearning, 16, 18
Qlearning_Single, 10, 12, 14, 17, 17

rbfdot, 19

wsvm, 6–11, 13, 15, 18