Package ‘VHDClassification’

February 19, 2015

Type Package

Title Discrimination/Classification in very high dimension with linear and quadratic rules.

Version 0.3

Date 2013-12-11

Author Robin Girard

Maintainer Robin Girard <robin.girard@mines-paristech.fr>

Description This package provides an implementation of Linear discriminant analysis and quadratic discriminant analysis that works fine in very high dimension (when there are many more variables than observations).

License GPL-2

LazyLoad yes

Depends methods, e1071, lattice, stats

Suggests pamr,randomForest

NeedsCompilation no

Repository CRAN

Date/Publication 2013-12-18 14:13:37

R topics documented:

VHDClassification-package

Discrimination-Classification in very high dimension with linear and quadratic rules.

Description

This package provides an implementation of Linear discriminant analysis and quadratic discriminant analysis that works fine in very high dimension (when there are many more variables than observations).

Details

Package:	VHDClassification
Type:	Package
Version:	0.1
Date:	2010-04-15
License:	GPL-2
LazyLoad:	yes
Depends:	methods, e1071, lattice, stats

This package provides learning procedure for classification in very high dimension. Binary learning is done with `learnBinaryRule` while K-class (K>=2) learning is done with function `learnPartitionWithLLR`.

`learnBinaryRule` can return an object `LinearRule-class` or an object `QuadraticRule-class` depending whether type='linear' or 'quadratic'. `learnPartitionWithLLR` basically returns a set of binary rules which is represented by the class `PartitionWithLLR-class`. The used procedure for the learning are described in the papers cited below. The

The method predict (predict-methods) is implemented for class `LinearRule-class` `QuadraticRule-class` `learnPartitionWithLLR`. It predicts the class of a new observation.

Author(s)

Maintainer-author: Robin Girard <robin.girard@mines-paristech.fr>

References

Fast rate of convergence in high dimensional linear discriminant analysis. R. Girard To appear in Journal of Nonparametric Statistics.\ Very high dimensional discriminant analysis with thresholding estimation. R. Girard. Submitted.

Examples

```
############ Tests 2 classes when the true rule should be quadratic
#library(VHDClassification)
```

```
p=500; n=50 ; mu=array(0,c(p,2)) ; C=array(c(1,20),c(p,2)); C[c(1,3,5),1]=40
x=NULL; y=NULL;
for (k in 1:2)
{
  M=matrix(rnorm(p*n),nrow=p,ncol=n)
  tmp=array(C[,k]^(1/2),c(p,n))*(M)+array(mu[,k],c(p,n))
  x=rbind(x,t(tmp));
  y=c(y,array(k,n))
  }
#Learning
LearnedQuadBinaryRule=learnBinaryRule(x,y,type='quadratic')
LearnedLinBinaryRule=learnBinaryRule(x,y) # default is linear type
# for comparison with SVM
# require(e1071)
# svmRule=best.tune('svm',
#              train.x=x,
#              train.y=factor(y),
#              ranges=list(gamma=c(2^(-4:4),
#              cost = 2^(-2:2))))
# for comparison with randomForest
require(randomForest)
RF <- best.tune('randomForest',x,factor(y),ranges=list(ntree = c(100,500)))
# for comparison with nearest chrunken centroid
#require(pamr)
#myTrainingdata=list(x=t(x),y=y)
#mytrain <- pamr.train(myTrainingdata)
#mycv <- pamr.cv(mytrain,myTrainingdata)
#thresh=try(mycv$threshold[which.min(mycv$error)],silent = TRUE)




#Testing Set
x=NULL; y=NULL;
for (k in 1:2){
    M=matrix(rnorm(p*n),nrow=p,ncol=n)
    x=rbind(x,t(array(C[,k]^(1/2),c(p,n))*(M)+array(mu[,k],c(p,n))));
    y=c(y,array(k,n))
}
#Testing
myTestingdata=list(x=x,y=y)
QDAScore=mean((y!=predict(LearnedQuadBinaryRule,myTestingdata$x))) ;
LDAScore=mean((y!=predict(LearnedLinBinaryRule,myTestingdata$x))) ;
RFScore=mean((y!=predict(RF,myTestingdata$x))) ;
#SVMScore=mean((y!=predict(svmRule,x))) ;
#comparison with nearest chrunken centroid
myTestingdata=list(x=t(x),y=y)
#V=as.numeric(pamr.predict(mytrain, myTestingdata$x,threshold=thresh,type="class"))
#SCScore=mean((myTestingdata$y!=V))
cat('\n')
cat('What does it cost to use type=linear  when the rule  is quadratic ? ','\n',
'Score of the linear rule: ',LDAScore,'\n',
'Score of the quadratic rule: ',QDAScore,'\n',
```

```
#'Score of the nearest shrunken centroid rule: ',SCScore,'\n',
'Score of the random forest rule: ',RFScore,'\n',
#'Score of the support vector machine rule: ',SVMScore,'\n',
'Note: These scores should be average for a large number of experiment or interpreted carefully \n')
plotClassifRule(LearnedQuadBinaryRule)

############ Tests 2 classes quadratic and linear. when the true is linear
#library(VHDClassification)
#p=100; n=50 ; mu=array(0,c(p,2)); mu[1:10,1]=1 ;C=array(c(1,20),p)
#x=NULL; y=NULL;
#for (k in 1:2){
# M=matrix(rnorm(p*n),nrow=p,ncol=n)
# x=rbind(x,t(array(C^(1/2),c(p,n))*(M)+array(mu[,k],c(p,n))));
#    y=c(y,array(k,n))}
#Learning
#LearnedQuadBinaryRule=learnBinaryRule(x,y,type='quadratic')
#LearnedLinBinaryRule=learnBinaryRule(x,y) # default is linear type
#comparison with nearest chrunken centroid
#require(pamr)
#myTrainingdata=list(x=t(x),y=y)
#mytrain <- pamr.train(myTrainingdata)
#mycv <- pamr.cv(mytrain,myTrainingdata)
#thresh=try(mycv$threshold[which.min(mycv$error)],silent = TRUE)


#Testing Set
#x=NULL; y=NULL;
#for (k in 1:2){
# M=matrix(rnorm(p*n),nrow=p,ncol=n)
#    x=rbind(x,t(array(C^(1/2),c(p,n))*(M)+array(mu[,k],c(p,n))));
#    y=c(y,array(k,n))
#}
#Testing
#myTestingdata=list(x=x,y=y)
#QDAScore=mean((y!=predict(LearnedQuadBinaryRule,myTestingdata$x))) ;
#LDAScore=mean((y!=predict(LearnedLinBinaryRule,myTestingdata$x))) ;
#comparison with nearest shrunken centroid
#myTestingdata=list(x=t(x),y=y)
#tmp=as.numeric(pamr.predict(mytrain,threshold=thresh,
# myTestingdata$x,type="class"))
#SCScore=mean((myTestingdata$y!=tmp))
#cat('\n',
#'What does it cost to use type=
# quadratic rule when the true optimal rule is linear ? ','\n',
#'Score of the linear rule: ',LDAScore,'\n',
#'Score of the  rule with type=quadratic : ',QDAScore,'\n',
# 'it detects that the true rule is linear?\n',
#'Score of the nearest shrunken centroid rule: ',SCScore,'\n')

#plotClassifRule(LearnedQuadBinaryRule)

############ Tests 3 classes
#library(VHDClassification)
```

```
#p=1000; n=40 ; mu=array(0,c(p,3)); mu[1:10,1]=4; C=array(c(1,20),p)

#x=NULL; y=NULL;
#for (k in 1:3){
#    if (k<3){
#     M=matrix(rnorm(p*n),nrow=p,ncol=n)
#    x=rbind(x,t(array(C^(1/2),c(p,n))*(M)+array(mu[,k],c(p,n))));
#    y=c(y,array(k,n))}
#    else
#    {
#    tildeC=C; tildeC[1:10]=40;
#     M=matrix(rnorm(p*n),nrow=p,ncol=n)
#    x=rbind(x,t(array(tildeC^(1/2),c(p,n))*(M)+array(mu[,k],c(p,n))));
#    y=c(y,array(k,n))
#    }
#    }
#Learning
#LearnedLinearPartitionWithLLR=learnPartitionWithLLR(x,y,type='linear')
#LearnedQuadraticPartitionWithLLR=learnPartitionWithLLR(x,y,type='quadratic')
#plotClassifRule(LearnedQuadraticPartitionWithLLR)
#require(randomForest)
#RF <- best.tune('randomForest',x,factor(y),ranges=list(ntree = c(500)))

#Testing Set
#x=NULL; y=NULL;
#for (k in 1:3){
#    if (k<3){
#     M=matrix(rnorm(p*n),nrow=p,ncol=n)
#    x=rbind(x,t(array(C^(1/2),c(p,n))*(M)+array(mu[,k],c(p,n))));
#    y=c(y,array(k,n))}
#    else
#    {
#    tildeC=C; tildeC[1:10]=40;
#     M=matrix(rnorm(p*n),nrow=p,ncol=n)
#    x=rbind(x,t(array(tildeC^(1/2),c(p,n))*(M)+array(mu[,k],c(p,n))));
#    y=c(y,array(k,n))
#    }
#    }
#Testing
#myTestingdata=list(x=x,y=y)
#LDAScore=mean((y!=factor(predict(LearnedLinearPartitionWithLLR,myTestingdata$x)))) ;
#QDAScore=mean((y!=factor(predict(LearnedQuadraticPartitionWithLLR,myTestingdata$x)))) ;
#RFScore=mean((y!=predict(RF,myTestingdata$x))) ;

#cat('Score of the quadratic rule: ',QDAScore,'\n',
#'Score of the linear rule: ',LDAScore,'\n',
#'Score of the random Forest Rule: ',RFScore,'\n')
```

`.EvaluateLogLikeRatio-methods` *~~ Methods for Function .EvaluateLogLikeRatio ~~*

Description

~~ Methods for function `.EvaluateLogLikeRatio` ~~

Methods

`signature(x = "numeric", object = "LinearRule")` hidden.

`getBinaryRule` *Getter set of binary rules (object PartitionWithLLR)*

Description

This function returns the binary rule for discrimination between data from class k and data from class l

Usage

```
getBinaryRule(object, k, l)
```

Arguments

`object`	An object of class PartitionWithLLR as returned by learnPartitionWithLLR
`k`	an existing label
`l`	an existing label

Value

A binary classification rule. Can either be an object of class LinearRule or an object of class QuadraticRule

Author(s)

Robin Girard

References

Fast rate of convergence in high dimensional linear discriminant analysis. R. Girard To appear in Journal of Nonparametric Statistics.\ Very high dimensional discriminant analysis with thresholding estimation. R. Girard. Submitted.

See Also

`getLogLikeRatio`

Examples

```
#try p=1000 , 5000, ...
p=100; n=20 ; mu=array(0,c(p,4)); mu[1:10,1]=2 ;mu[11:20,2]=2;C=array(c(1,20),p)
mu[21:30,3]=2
x=NULL; y=NULL;
for (k in 1:4){
   x=rbind(x,t(array(C^(1/2),c(p,n))*(matrix(rnorm(p*n),nrow=p,ncol=n))+array(mu[,k],c(p,n))));
    y=c(y,array(k,n))}
#Learning
LearnedLinearPartitionWithLLR=learnPartitionWithLLR(x,y,procedure='FDRThresh')
Rule=getBinaryRule(LearnedLinearPartitionWithLLR,1,2)
show(Rule)
```

`getBinaryRule-methods` *~~ Methods for Function getBinaryRule ~~*

Description

~~ Methods for function `getBinaryRule` ~~

Methods

`signature(object = "PartitionWithLLR")` see `getBinaryRule`

`getLogLikeRatio` *Get the log-likelihood ratio from a binary rule (QuadraticRule or LinearRule)*

Description

Binary rules can be expressed

Usage

```
getLogLikeRatio(object)
```

Arguments

`object`	an object of type LinearRule or QuadraticRule.

Details

Get everything that defines a log likelihood ratio between two gaussian measures.

Value

A list, see `getLogLikeRatio-methods`

Author(s)

Robin Girard

References

Fast rate of convergence in high dimensional linear discriminant analysis. R. Girard To appear in Journal of Nonparametric Statistics.\ Very high dimensional discriminant analysis with thresholding estimation. R. Girard. Submitted.

Examples

```
p=100; n=20 ; mu=array(0,c(p,4)); mu[1:10,1]=2 ;mu[11:20,2]=2;C=array(c(1,20),p)
mu[21:30,3]=2
x=NULL; y=NULL;
for (k in 1:4){
   x=rbind(x,t(array(C^(1/2),c(p,n))*(matrix(rnorm(p*n),nrow=p,ncol=n))+array(mu[,k],c(p,n))));
    y=c(y,array(k,n))}
#Learning
LearnedLinearPartitionWithLLR=learnPartitionWithLLR(x,y,procedure='FDRThresh')

Rule=getBinaryRule(LearnedLinearPartitionWithLLR,1,2)
LLR=getLogLikeRatio(Rule)
print(LLR)
```

`getLogLikeRatio-methods`

~~ Methods for Function getLogLikeRatio ~~

Description

~~ Methods for function `getLogLikeRatio` ~~

Methods

`signature(object = "LinearRule")` Returns a list with NormalVector and CenterVector. The loglikelihood ratio on x can be evaluated by L(x)=1/2<NormalVector,x-CenterVector>.

`signature(object = "QuadraticRule")` returns a list with a NormalVector, CenterVector, FormVector (3 vectors) and a numeric constant Constant. The loglikelihood ratio on x can be evaluated by L(x)=-1/2<diag(FormVector)(x-CenterVector),x-CenterVector>+<NormalVector,x-CenterVector> -Constant `learnBinaryRule` *Function to learn a binary classification rule*

Description

Function to learn a binary classification rule. For more than two class, use `learnPartitionWithLLR` instead. The learned rule can be linear or quadratic. There are reduction dimension methods (accessible via argument procedure) to make the procedure efficient when the number of features is larger than the number of observations

Usage

```
learnBinaryRule(x, y,type='linear', procedure = "FDRThresh",
covariance = "diagonal", ql = NULL, qq = NULL,prior=FALSE)
```

Arguments

`x`	The Matrix with input data of size pxn (p feature space dimension, and n number of observations)
`y`	A vector of n factors with 2 LEVELS (labels) associated to observations (can also be numeric)
`type`	'quadratic' or 'linear' are valid types.
`procedure`	Procedure gives the used procedure to reduce the dimensionality of the estimated NormalVector and FormVector. use 'noThresh' for no dimensionality reduction. UnivTresh is the universal threshold and FDRThresh is an FDR thresolding procedure. When type=='linear' 'FANThresh' and 'FDRstudent' are also available. For type linear, the thresholding procedures are fully described in the Paper "Fast rate of convergence in high dimensional linear discriminant analysis"
`covariance`	Unused argument ... further development comming soon
`ql`	The parameter associated to the thresholding procedure for the estimation of NormalVector. If a vector of values is given a 10 fold cross validation is performed
`qq`	The parameter associated to the thresholding procedure for the estimation of FormVector (only when type='quadratic'). If a vector of values is given a 10 fold cross validation is performed
`prior`	Do we put a prior on y (taking into account the proportion of the different class in the learning set to build the classification rule

Value

A classification rule of class LinearRule if type='linear' and of class QuadraticRule if type='quadratic'.

Author(s)

Robin Girard

References

Fast rate of convergence in high dimensional linear discriminant analysis. R. Girard To appear in Journal of Nonparametric Statistics.\ Very high dimensional discriminant analysis with thresholding estimation. R. Girard. Submitted.

See Also

learnPartitionWithLLR

Examples

```
p=100; n=50 ; mu=array(0,c(p,2)); mu[1:10,1]=1 ;C=array(c(1,20),p)
x=NULL; y=NULL;

for (k in 1:2){
  M=matrix(rnorm(p*n),nrow=p,ncol=n)
  x=rbind(x,t(array(C^(1/2),c(p,n))*(M)+array(mu[,k],c(p,n))));
    y=c(y,array(k,n))    }
#Learning
LearnedBinaryRule=learnBinaryRule(x,y)
show(LearnedBinaryRule)
```

---

learnPartitionWithLLR *A function to learn a rule in case of 2 classes or more*

Description

A function to learn a rule in case of 2 classes or more. There are reduction dimension methods (accessible via argument procedure) to make the procedure efficient when the number of features is larger than the number of observations

Usage

```
learnPartitionWithLLR(x, y, type = "linear", procedure = "FDRThresh",
ql = NULL, qq = NULL, BinaryLearningProcedure = NULL,prior=FALSE)
```

Arguments

	The Argument are exactly the same as in learnBinaryRule except that y may have more than 2 levels
	see learnBinaryRule
x y	vector of factors with two or more levels
type	
procedure	
ql	
qq	

BinaryLearningProcedure

`prior` Do we put a prior on y (taking into account the proportion of the different class in the learning set to build the classification rule

---

LinearRule-class *Class "LinearRule" ~~~*

Description

~~ A concise (1-5 lines) description of what the class is. ~~

Objects from the Class

Objects can be created by calls of the form `new("LinearRule", ...)`. ~~ describe objects here ~~

Slots

`labels`: Object of class `"factor"` ~~

`normalVector`: Object of class `"numeric"` ~~

`normalIndex`: Object of class `"numeric"` ~~

`centerVector`: Object of class `"numeric"` ~~

`prior`: Object of class `"logical"` ~~

`proportions`: Object of class `"numeric"` ~~

Methods

.EvaluateLogLikeRatio `signature(x = "numeric", object = "LinearRule")`: ...

getLogLikeRatio `signature(object = "LinearRule")`: ...

plotClassifRule `signature(object = "LinearRule")`: ...

predict `signature(object = "LinearRule")`: ...

show `signature(object = "LinearRule")`: ...

Author(s)

Robin Girard

Examples

```
showClass("LinearRule")
```

PartitionWithLLR-class

Class "PartitionWithLLR" ~~~

Description

~~ A concise (1-5 lines) description of what the class is. ~~

Objects from the Class

Objects can be created by calls of the form `new("PartitionWithLLR", ...)`. ~~ describe objects here ~~

Slots

`LogLikeRatio`**:** Object of class `"list"` ~~

`labels`**:** Object of class `"ordered"` ~~

`ThreshProc`**:** Object of class `"character"` ~~

`ql`**:** Object of class `"numeric"` ~~

`qq`**:** Object of class `"numeric"` ~~

Methods

getBinaryRule `signature(object = "PartitionWithLLR")`: ...

plotClassifRule `signature(object = "PartitionWithLLR")`: ...

predict `signature(object = "PartitionWithLLR")`: ...

show `signature(object = "PartitionWithLLR")`: ...

Author(s)

Robin Girard

Examples

```
showClass("PartitionWithLLR")
```

`plotClassifRule` *A plot function for classification rules (binary or not, quadratic or linear)*

Description

plot function for classification rules (binary or not, quadratic or linear). Essentially a wrapper to xyplot.

Usage

```
plotClassifRule(object, ...)
```

Arguments

`object`	
`...`	other argument that can be passed through xyplot

Author(s)

Robin Girard

`QuadraticRule-class` *Class "QuadraticRule" ~~~*

Description

This class implements a high dimensional binary quadratic classification rule

Objects from the Class

Objects can be created by calls of `learnBinaryRule(x,y,type='quadratic')` see `learnBinaryRule`.

Slots

`formVector`: Object of class `"numeric"` ~~

`formIndex`: Object of class `"numeric"` ~~

`constant`: Object of class `"numeric"` ~~

`normalVector`: Object of class `"numeric"` ~~

`normalIndex`: Object of class `"numeric"` ~~

`centerVector`: Object of class `"numeric"` ~~

Extends

Class `"LinearRule"`, directly.

Methods

getLogLikeRatio `signature(object = "QuadraticRule")`: ...

plotClassifRule `signature(object = "QuadraticRule")`: ...

predict `signature(object = "QuadraticRule")`: ...

show `signature(object = "QuadraticRule")`: ...

Author(s)

robin girard

References

See my preprint Preprint

Examples

```
showClass("QuadraticRule")
```

Index