Package ‘STPGA’

October 12, 2022

Type Package

Title Selection of Training Populations by Genetic Algorithm

Version 5.2.1

Date 2018-11-21

Author Deniz Akdemir

Maintainer Deniz Akdemir <deniz.akdemir.work@gmail.com>

Description Combining Predictive Analytics and Experimental Design to Optimize Results. To be utilized to select a test data calibrated training population in high dimensional prediction problems and assumes that the explanatory variables are observed for all of the individuals. Once a ``good'' training set is identified, the response variable can be obtained only for this set to build a model for predicting the response in the test set. The algorithms in the package can be tweaked to solve some other subset selection problems.

License GPL-3

Depends R (>= 2.10), AlgDesign, scales, scatterplot3d, emoa, grDevices

Suggests R.rsp, EMMREML, quadprog, UsingR, glmnet, leaps, Matrix

NeedsCompilation no

Repository CRAN

Date/Publication 2018-11-24 17:20:06 UTC

R topics documented:

STPGA-package *Selection of Training Populations by Genetic Algorithm*

Description

This package can be utilized to select a (test data) calibrated training population in high dimensional prediction problems. More specifically, the package contains a genetic algorithm that tries to minimize a design criterion defined for subsets of a certain size selected from a larger set.

Details

Package:	STPGA
Type:	Package
Version:	5.0
Date:	2018-07-20
License:	GPL-3

The package is useful for high dimensional prediction problems where per individual cost of observing / analyzing the response variable is high and therefore a small number of training examples is sought or when the candidate set from which the training set must be chosen (is not representative of the test data set).

The function "GenAlgForSubsetSelection" uses a simple genetic algorithm to identify a training set of a specified size from a larger set of candidates which minimizes an optimization criterion for a known test set. The function "GenAlgForSubsetSelectionNoTest" tries to identify a training set of a specified size from a larger set of candidates which minimizes an optimization criterion.

Author(s)

Maintainer: Deniz Akdemir <deniz.akdemir.work@gmail.com>

References

References: Akdemir, Deniz. "Training population selection for (breeding value) prediction." arXiv preprint arXiv:1401.7953 (2014).

Amat.pieces *Amat.pieces*

Description

This calculates the genomic relationship matrix using the formula in VanRaden (2008)

Usage

```
Amat.pieces(M, pieces=10, mc.cores=1)
```

Arguments

`M`	The matrix of markers rows rorresponding to individuals and columns for markers, the markers scores are coded as -1,0,1 (corresponding to allele counts 0,1,2).
`pieces`	number of chunks to split the markers
`mc.cores`	number of cores to use

Value

a genomic relationship matrix.

Author(s)

Deniz Akdemir

References

VanRaden, Paul M. ”Efficient methods to compute genomic predictions.” Journal of dairy science 91.11 (2008): 4414-4423.

Examples

```
N=50
nmarkers=500
Markers<-c()
for (i in 1:N){
  Markers<-rbind(Markers,sample(-1:1,nmarkers, replace=TRUE))
}

markereffects<-rep(0,nmarkers)
markereffects[sample(1:nmarkers,nmarkers/2)]<-rnorm(nmarkers/2)
Markers[1:5,1:5]

K=Amat.pieces(Markers, pieces=5)
K[1:5,1:5]
```

`CRITERIA` *Optimality Criteria*

Description

These are some default design criteria to be minimized. There is a table in the details section that gives the formula for each design criterion and describes their usage. Note that the inputs for these functions come in 3 syntax flavors, namely Type-X, Type-D and Type-K. Users can define and use their owm design criteria as long as it has the Type-X syntax as shown with the examples.

Usage

```
AOPT(Train, Test, P, lambda = 1e-05, C=NULL)
CDMAX(Train, Test, P, lambda = 1e-05, C=NULL)
CDMAX0(Train, Test, P, lambda = 1e-05, C=NULL)
CDMAX2(Train, Test, P, lambda = 1e-05, C=NULL)
CDMEAN(Train, Test, P, lambda = 1e-05, C=NULL)
CDMEAN0(Train, Test, P, lambda = 1e-05, C=NULL)
CDMEAN2(Train, Test, P, lambda = 1e-05, C=NULL)
CDMEANMM(Train, Test, Kinv,K, lambda = 1e-05, C=NULL, Vg=NULL, Ve=NULL)
DOPT(Train, Test, P, lambda = 1e-05, C=NULL)
EOPT(Train, Test, P, lambda = 1e-05, C=NULL)
GAUSSMEANMM(Train, Test, Kinv, K, lambda = 1e-05, C=NULL, Vg=NULL, Ve=NULL)
GOPTPEV(Train, Test, P, lambda = 1e-05, C=NULL)
GOPTPEV2(Train, Test, P, lambda = 1e-05, C=NULL)
PEVMAX(Train, Test, P, lambda = 1e-05, C=NULL)
PEVMAX0(Train, Test, P, lambda = 1e-05, C=NULL)
PEVMAX2(Train, Test, P, lambda = 1e-05, C=NULL)
PEVMEAN(Train, Test, P, lambda = 1e-05, C=NULL)
PEVMEAN0(Train, Test, P, lambda = 1e-05, C=NULL)
PEVMEAN2(Train, Test, P, lambda = 1e-05, C=NULL)
PEVMEANMM(Train, Test, Kinv,K, lambda = 1e-05, C=NULL, Vg=NULL, Ve=NULL)
dist_to_test(Train, Test, Dst, lambda, C)
dist_to_test2(Train, Test, Dst, lambda, C)
neg_dist_in_train(Train, Test, Dst, lambda, C)
neg_dist_in_train2(Train, Test, Dst, lambda, C)
```

Arguments

`Train`	vector of identifiers for individuals in the training set
`Test`	vector of identifiers for individuals in the test set
`P`	(Only for Type-X) $n \times k$ matrix of the first PCs of the predictor variables. The matrix needs to have union of the identifiers of the candidate and test individuals as rownames.
`Dst`	(Only for Type-D) $n \times n$ symmetric distance matrix with row and column names.
`Kinv`	(Only for Type-K) $n \times n$ symmetric matrix (inverse of the relationship matrix K between n individuals) with row and column names.
`K`	(Only for Type-K) $n \times n$ symmetric matrix (the relationship matrix K between n individuals).
`lambda`	scalar shrinkage parameter ($\lambda > 0$).
`C`	Contrast Matrix.
`Vg`	(Only for PEVMEANMM) covariance matrix between traits generated by the relationship K (multi-trait version).
`Ve`	(Only for PEVMEANMM) residual covariance matrix for the traits (multi-trait version).

Details

criterion name	formula	Type
AOPT	$trace[C(P'_{Train}P_{Train} + lambda * I)^{-1}C']$	X
CDMAX	$max[diag(CP_{Test}(P'_{Train}P_{Train} + lambda * I)^{-1}P'_{Test}C')/ diag(CP_{Test}P'_{Test}C')]$	X
CDMAX0	$max[diag(CP_{Train}(P'_{Train}P_{Train} + lambda * I)^{-1}P'_{Train}C')/ diag(CP_{Train}P'_{Train}C')]$	X
CDMAX2	$max[diag(CP_{Test}(P'_{Train}P_{Train} + lambda * I)^{-1}P'_{Train}P_{Train} (P'_{Train}P_{Train} + lambda * I)^{-1}P'_{Test}C')/diag(CP_{Test}P'_{Test}C')]$	X
CDMEAN	$mean[diag(CP_{Test}(P'_{Train}P_{Train} + lambda * I)^{-1}P'_{Test}C')/ diag(CP_{Test}P'_{Test}C')]$	X
CDMEAN0	$mean[diag(CP_{Train}(P'_{Train}P_{Train} + lambda * I)^{-1}P'_{Train}C')/ diag(CP_{Train}P'_{Train}C')]$	X
CDMEAN2	$mean[diag(CP_{Test}(P'_{Train}P_{Train} + lambda * I)^{-1}P'_{Train}P_{Train} (P'_{Train}P_{Train} + lambda * I)^{-1}P'_{Test}C')/diag(CP_{Test}P'_{Test}C')]$	X
CDMEANMM	$-mean[diag(CZ_{Test}(K - lambda * (Z'_{Train}MZ_{Train} + \lambda * Kinv)^{-1}Z'_{Test}C')/ (diag(CZ_{Test}KZ'_{Test}C'))]$	K
DOPT	$logdet(C(P'_{Train}P_{Train} + lambda * I))^{-1}C'$	X
EOPT	$max(eigenval(C(P'_{Train}P_{Train} + lambda * I))^{-1}C'))$	X
GAUSSMEANMM	$-mean(diag(Z_{Test}KZ'_{Test}- Z_{Test}KZ'_{Train}(Z_{Train}KZ'_{Train} + \lambda * I_{ntrain})^{-1}Z_{Train}KZ'_{Test})$	K
GOPTPEV	$max(eigenval(CP_{Test}(P'_{Train}P_{Train} + \lambda * I_{ntrain})^{-1}P'_{Test}C'))$	X
GOPTPEV2	$mean(eigenval(CP_{Test}(P'_{Train}P_{Train} + \lambda * I_{ntrain})^{-1}P'_{Test}C'))$	X
PEVMAX	$max(diag(CP_{Test}(P'_{Train}P_{Train} + lambda * I)^{-1}P'_{Test}C'))$	X
PEVMAX0	$max(diag(CP_{Train}(P'_{Train}P_{Train} + lambda * I)^{-1}P'_{Train}C'))$	X
PEVMAX2	$max[diag(CP_{Test}(P'_{Train}P_{Train} + lambda * I)^{-1}P'_{Train}P_{Train} (P'_{Train}P_{Train} + lambda * I)^{-1}P'_{Test}C']$	X
PEVMEAN	$mean(diag(CP_{Test}(P'_{Train}P_{Train} + lambda * I)^{-1}P'_{Test}C'))$	X
PEVMEAN0	$mean(diag(CP_{Train}(P'_{Train}P_{Train} + lambda * I)^{-1}P'_{Train}C'))$	X
PEVMEAN2	$mean[diag(CP_{Test}(P'_{Train}P_{Train} + lambda * I)^{-1} P'_{Train}P_{Train}(P'_{Train}P_{Train} + lambda * I)^{-1}P'_{Test}C']$	X
PEVMEANMM	$mean(diag(CZ_{test}(Ztrain'MZtrain + lambda * Kinv)^{-1}Ztest'C')))$	K
dist_to_test	maximum distance from training set to test set based on Dst	D
dist_to_test2	mean distance from training set to test set based on Dst	D
neg_dist_in_train	negative of minimum distance between pairs in the training set based on Dst	D
neg_dist_in_train2	negative of mean distance between distinct pairs in the training set based on Dst	D

Value

value of the criterion.

Author(s)

Deniz Akdemir

Examples

```
## Not run:
#Examples to new criterion:
#1- PEVmax
STPGAUSERFUNC<-function(Train,Test, P, lambda=1e-6, C=NULL){
  PTrain<-P[rownames(P)%in%Train,]
  PTest<-P[rownames(P)%in%Test,]
  if (length(Test)==1){PTest=matrix(PTest, nrow=1)}
  if (!is.null(C)){ PTest<-C%*%PTest}
  PEV<-PTest%*%solve(crossprod(PTrain)+lambda*diag(ncol(PTrain)),t(PTrain))
    PEVmax<-max(diag(tcrossprod(PEV)))
  return(PEVmax)
}




######Here is an example of usage
data(iris)
#We will try to estimate petal width from
#variables sepal length and width and petal length.
X<-as.matrix(iris[,1:4])
distX<-as.matrix(dist(X))
rownames(distX)<-colnames(distX)<-rownames(X)<-paste(iris[,5],rep(1:50,3),sep="_" )
#test data 25 iris plants selected at random from the virginica family,
#candidates are the plants in the  setosa and versicolor families.
candidates<-rownames(X)[1:100]
test<-sample(setdiff(rownames(X),candidates), 25)
#want to select 25 examples using the criterion defined in STPGAUSERFUNC
#Increase niterations and npop substantially for better convergence.
ListTrain<-GenAlgForSubsetSelection(P=distX,Candidates=candidates,
Test=test,ntoselect=25,npop=50,
nelite=5, mutprob=.8, niterations=30,
lambda=1e-5, errorstat="STPGAUSERFUNC", plotiters=TRUE)

## End(Not run)
```

disttoideal *Calculate the distance of solutions from the 'ideal' solution.*

Description

This function calculates the distance of X to the vector of the minimums of columns of X after transforming the variables in X to the interval $[0, 1]$.

Usage

```
disttoideal(X)
```

Arguments

`X`	A matrix of the criteria values. One solution each row, columns correspond to the different criteria.

Value

Vector of distances elements corresponding to each row of X

Author(s)

Deniz Akdemir

GenAlgForSubsetSelection

Genetic algorithm for subset selection

Description

It uses a genetic algorithm to select n_{Train} individuals so that optimality criterion is minimum.

Usage

```
GenAlgForSubsetSelection(P, Candidates, Test, ntoselect, npop = 100, nelite =
                 5, keepbest = TRUE, tabu = TRUE, tabumemsize = 1, mutprob
                 = 0.8, mutintensity = 1, niterations = 500,
                 minitbefstop = 200, niterreg = 5, lambda = 1e-06,
                plotiters = FALSE, plottype=1,errorstat = "PEVMEAN2", C = NULL,
                 mc.cores = 1, InitPop = NULL, tolconv = 1e-07, Vg =
                 NULL, Ve = NULL, Fedorov=FALSE)
```

Arguments

`P`	depending on the criterion this is either a numeric data matrix or a symmetric similarity matrix. When it is a data matrix, the union of the identifiers of the candidate (and test) individuals should be put as rownames (and column names in case of a similarity matrix). For methods using the relationships, this is the inverse of the relationship matrix with row and column names as the the identifiers of the candidate (and test) individuals.

`Candidates` vector of identifiers for the individuals in the candidate set.

`Test` vector of identifiers for the individuals in the test set.

`ntoselect` n_{Train} : number of individuals to select in the training set.

`npop` genetic algorithm parameter, number of solutions at each iteration

`nelite` genetic algorithm parameter, number of solutions selected as elite parents which will generate the next set of solutions.

`keepbest` genetic algorithm parameter, TRUE or FALSE. If TRUE then the best solution is always kept in the next generation of solutions (elitism).

`tabu` genetic algorithm parameter, TRUE or FALSE. If TRUE then the solutions that are saved in tabu memory will not be retried.

`tabumemsize` genetic algorithm parameter, integer>0. Number of generations to hold in tabu memory.

`mutprob` genetic algorithm parameter, probability of mutation for each generated solution.

`mutintensity` mean of the poisson variable that is used to decide the number of mutations for each cross.

`niterations` genetic algorithm parameter, number of iterations.

`minitbefstop` genetic algorithm parameter, number of iterations before stopping if no change is observed in criterion value.

`niterreg` genetic algorithm parameter, number of iterations to use regressions, an integer with minimum value of 1

`lambda` scalar shrinkage parameter ($\lambda > 0$).

`plotiters` plot the convergence: TRUE or FALSE. Default is TRUE.

`plottype` type of plot, default is 1. possible values 1,2,3.

`errorstat` optimality criterion: One of the optimality criterion. Default is "PEVMEAN". It is possible to use user defined functions as shown in the examples.

`mc.cores` number of cores to use.

`InitPop` a list of initial solutions

`tolconv` if the algorithm cannot improve the errorstat more than tolconv for the last minitbefstop iterations it will stop.

`C` Contrast Matrix.

`Vg` covariance matrix between traits generated by the relationship K (only for multi-trait version of PEVMEANMM).

`Ve` residual covariance matrix for the traits (only for multi-trait version of PEVMEANMM).

`Fedorov` Whether the Fedorovs exchange algorithm from `AlgDesign` Package should be used for initial solutions.

Value

A list of length nelite+1. The first nelite elements of the list are optimized training samples of size n_{train} and they are listed in increasing order of the optimization criterion. The last item on the list is a vector that stores the minimum values of the objective function at each iteration.

Note

The GA does not guarantee convergence to globally optimal solutions and it is highly recommended that the algorithm is replicated to obtain ”good” training samples.

Author(s)

Deniz Akdemir

Examples

```
## Not run:
####################################
library(EMMREML)
library(STPGA)
data(WheatData)

svdWheat<-svd(Wheat.K, nu=5, nv=5)
PC50WHeat<-Wheat.K%*%svdWheat$v
plot(PC50WHeat[,1],PC50WHeat[,2])
rownames(PC50WHeat)<-rownames(Wheat.K)
DistWheat<-dist(PC50WHeat)
TreeWheat<-hclust(DistWheat)
TreeWheat<-cutree(TreeWheat, k=4)

Test<-rownames(PC50WHeat)[TreeWheat==4]
length(Test)
Candidates<-setdiff(rownames(PC50WHeat), Test)


###instead of using the algorithm directly using a wrapper to
###implement an for multiple starting points for genetic algorithm.
repeatgenalg<-function(numrepsouter,numrepsinner){
  StartingPopulation2=NULL
  for (i in 1:numrepsouter){
    print("Rep:")
    print(i)
    StartingPopulation<-lapply(1:numrepsinner, function(x){
     GenAlgForSubsetSelection(P=PC50WHeat,Candidates=Candidates,
     Test=Test, ntoselect=50, InitPop=StartingPopulation2,
 npop=50, nelite=5, mutprob=.5, mutintensity = rpois(1,4),
 niterations=10,minitbefstop=5, tabumemsize = 2,plotiters=TRUE,
 lambda=1e-9,errorstat="CDMEAN", mc.cores=1)})
    StartingPopulation2<-vector(mode="list", length = numrepsouter*1)
    ij=1
    for (i in 1:numrepsinner){
      for (j in 1:1){
        StartingPopulation2[[ij]]<-StartingPopulation[[i]][[j]]
        ij=ij+1
      }
    }
  }
  ListTrain<-GenAlgForSubsetSelection(P=PC50WHeat,Candidates=Candidates,
     Test=Test,ntoselect=50, InitPop=StartingPopulation2,npop=100,
     nelite=10, mutprob=.5, mutintensity = 1,niterations=300,
     minitbefstop=100, tabumemsize = 1,plotiters=T,
     lambda=1e-9,errorstat="CDMEAN", mc.cores=1)
  return(ListTrain)
}


ListTrain<-repeatgenalg(20, 3)

###test sample
deptestopt<-Wheat.Y[Wheat.Y$id%in%Test,]

##predictions by optimized sample
deptrainopt<-Wheat.Y[(Wheat.Y$id%in%ListTrain[[1]]),]

Ztrain<-model.matrix(~-1+deptrainopt$id)
Ztest<-model.matrix(~-1+deptestopt$id)

modelopt<-emmreml(y=deptrainopt$plant.height,X=matrix(1, nrow=nrow(deptrainopt), ncol=1),
Z=Ztrain, K=Wheat.K)
predictopt<-Ztest%*%modelopt$uhat

corvecrs<-c()
for (rep in 1:300){
###predictions by a random sample of the same size
  rs<-sample(Candidates, 50)

  deptestrs<-Wheat.Y[Wheat.Y$id%in%Test,]

  deptrainrs<-Wheat.Y[(Wheat.Y$id%in%rs),]

  Ztrain<-model.matrix(~-1+deptrainrs$id)
  Ztest<-model.matrix(~-1+deptestrs$id)

  library(EMMREML)
  modelrs<-emmreml(y=deptrainrs$plant.height,X=matrix(1, nrow=nrow(deptrainrs), ncol=1),
  Z=Ztrain, K=Wheat.K)
  predictrs<-Ztest%*%modelrs$uhat
corvecrs<-c(corvecrs,cor(predictrs, deptestrs$plant.height))

}
mean(corvecrs)
cor(predictopt, deptestopt$plant.height)


plot(PC50WHeat[,1],PC50WHeat[,2], col=rownames(PC50WHeat)%in%ListTrain[[1]]+1,
pch=2*rownames(PC50WHeat)%in%Test+1, xlab="pc1", ylab="pc2")

## End(Not run)
```

GenAlgForSubsetSelectionMO

Genetic algorithm for subset selection no given test with multiple criteria for Multi Objective Optimized Experimantal Design.

Description

It uses a nondominated selection genetic algorithm to find the solutions on the frontier that optimizes several design criteria at the same time. The test set is taken as the complement of the training individuals.

Usage

```
GenAlgForSubsetSelectionMO(Pcs = NULL, Dist = NULL, Kernel = NULL, Candidates, Test,
                 ntoselect, selectionstats, selectionstatstypes,
                 plotdirections, npopGA = 100, mutprob = 0.8,
                 mutintensity = 1, nitGA = 500, lambda = 1e-06,
                 plotiters = FALSE, mc.cores = 1, InitPop = NULL, C =
                 NULL, axes.labels = NULL)
```

Arguments

`Pcs` Principal components matrix for the individuals

`Dist` Distance matrix for the individuals

`Kernel` Kernel matrix for the individuals

`Candidates` The set of individuals from which the training set is selected.

`Test` The set of individuals for which the predictions based on the model built using the training set are needed.

`ntoselect` number of individuals to select in the training set.

`selectionstats` a vector of design optimization criteria

`selectionstatstypes` a vector describing the type of optimality criteria used in `selectionstats`.

`plotdirections` A vector that is used to change the sign of statistics while plotting. This doesnt effect the optimization.

`npopGA` genetic algorithm parameter, number of solutions at each iteration

`mutprob` genetic algorithm parameter, probability of mutation for each generated solution.

`mutintensity` genetic algorithm parameter, mean of the poisson variable that is used to decide the number of mutations for each cross.

`nitGA` genetic algorithm parameter, number of iterations.

`lambda` scalar shrinkage parameter ($\lambda > 0$).

`plotiters` plot the convergence: TRUE or FALSE. Default is TRUE.

`mc.cores`	number of cores to use.
`InitPop`	a list of initial solutions
`C`	Contrast Matrix.
`axes.labels`	Labels for the axes for plotting iterations

Value

A list of length 2. The first item in the list is the list of solutions found by the algorithm. The second item is a matrix of criteria values for the solutions in the first list.

Author(s)

Deniz Akdemir

Examples

```
## Not run:
library(STPGA)
library(GenomicMating)

data(WheatData)


Msvd<-svd(scale(Wheat.M, scale=F, center=T), nu=50, nv=50)
Dgeno<-as.matrix(dist(scale(Wheat.M, scale=F, center=T)))^2
P<-Wheat.M%*%Msvd$v
dim(P)
rownames(Dgeno)<-colnames(Dgeno)<-rownames(P)<-rownames(Wheat.M)
test<-sample(rownames(P), 25)
candidates<-setdiff(rownames(P), test)
outnewprog<-GenAlgForSubsetSelectionMO(Pcs=P,Dist=Dgeno,
Candidates=candidates,Test=test,ntoselect=75,
selectionstats=list("DOPT", "neg_dist_in_train2", "dist_to_test2"),
selectionstatstypes=c("Pcs", "Dist", "Dist"),
plotdirections=c(1,1,1),
npopGA=300,  mutprob=1, mutintensity=2,
nitGA=100, plotiters=TRUE, mc.cores=1, InitPop=NULL)

#####Best solution according to ideal solution concept
outnewprog[[1]][[which.min(disttoideal(outnewprog[[2]]))]]

## End(Not run)
```

GenAlgForSubsetSelectionMONoTest

Genetic algorithm for subset selection no given test with multiple criteria for Multi Objective Optimized Experimental Design.

Description

It uses a nondominated selection genetic algorithm to find the solutions on the frontier that optimizes several design criteria at the same time. The test set is taken as the complement of the training individuals.

Usage

```
GenAlgForSubsetSelectionMONoTest(Pcs = NULL, Dist = NULL, Kernel = NULL, Candidates,
                 ntoselect, selectionstats, selectionstatstypes,
                 plotdirections, npopGA = 100, mutprob = 0.8,
                 mutintensity = 1, nitGA = 500, lambda = 1e-06,
                 plotiters = FALSE, mc.cores = 1, InitPop = NULL, C =
                 NULL, axes.labels = NULL)
```

Arguments

`Pcs` Principal components matrix for the individuals

`Dist` Distance matrix for the individuals

`Kernel` Kernel matrix for the individuals

`Candidates` The set of individuals from which the training set is selected.

`ntoselect` number of individuals to select in the training set.

`selectionstats` a vector of design optimization criteria

`selectionstatstypes` a vector describing the type of optimality criteria used in `selectionstats`.

`plotdirections` A vector that is used to change the sign of statistics while plotting. This doesnt effect the optimization.

`npopGA` genetic algorithm parameter, number of solutions at each iteration

`mutprob` genetic algorithm parameter, probability of mutation for each generated solution.

`mutintensity` genetic algorithm parameter, mean of the poisson variable that is used to decide the number of mutations for each cross.

`nitGA` genetic algorithm parameter, number of iterations.

`lambda` scalar shrinkage parameter ($\lambda > 0$).

`plotiters` plot the convergence: TRUE or FALSE. Default is TRUE.

`mc.cores` number of cores to use.

`InitPop` a list of initial solutions

`C` Contrast Matrix.

`axes.labels` Labels for the axes for plotting iterations

Value

A list of length 2. The first item in the list is the list of solutions found by the algorithm. The second item is a matrix of criteria values for the solutions in the first list.

Author(s)

Deniz Akdemir

Examples

```
## Not run:
library(STPGA)
library(GenomicMating)

data(WheatData)


Msvd<-svd(scale(Wheat.M, scale=F, center=T), nu=50, nv=50)
Dgeno<-as.matrix(dist(scale(Wheat.M, scale=F, center=T)))^2
P<-Wheat.M%*%Msvd$v
dim(P)
rownames(Dgeno)<-colnames(Dgeno)<-rownames(P)<-rownames(Wheat.M)
test<-sample(rownames(P), 25)
candidates<-setdiff(rownames(P), test)
outnewprog<-GenAlgForSubsetSelectionMONoTest(Pcs=P,Dist=Dgeno,
Candidates=candidates,ntoselect=75,
selectionstats=list("DOPT", "neg_dist_in_train2", "dist_to_test2"),
selectionstatstypes=c("Pcs", "Dist", "Dist"),
plotdirections=c(1,1,1),npopGA=300,
mutprob=1, mutintensity=2, nitGA=100,
plotiters=TRUE, mc.cores=1, InitPop=NULL)

#####Best solution according to ideal solution concept
outnewprog[[1]][[which.min(disttoideal(outnewprog[[2]]))]]


## End(Not run)
```

`GenAlgForSubsetSelectionNoTest`

Genetic algorithm for subset selection no given test

Description

It uses a genetic algorithm to select n_{Train} individuals so that optimality criterion is minimum. The test set is taken as the complement of the training individuals.

Usage

```
GenAlgForSubsetSelectionNoTest(P, ntoselect, npop = 100, nelite = 5, keepbest = TRUE,
tabu = TRUE, tabumemsize = 1, mutprob=.8, mutintensity = 1,
niterations = 500, minitbefstop = 200, niterreg = 5,
lambda = 1e-06, plotiters = FALSE, plottype=1, errorstat =
 "PEVMEAN2", C = NULL, mc.cores = 1, InitPop = NULL,
 tolconv = 1e-07, Vg = NULL, Ve = NULL, Fedorov=FALSE)
```

Arguments

P depending on the criterion this is either a numeric data matrix or a symmetric similarity matrix. When it is a data matrix, the union of the identifiers of the candidate individuals should be put as rownames (and column names in case of a similarity matrix). For methods using the relationships, this is the inverse of the relationship matrix with row and column names as the the identifiers of the candidate individuals.

ntoselect n_{Train} : number of individuals to select in the training set.

npop genetic algorithm parameter, number of solutions at each iteration

nelite genetic algorithm parameter, number of solutions selected as elite parents which will generate the next set of solutions.

keepbest genetic algorithm parameter, TRUE or FALSE. If TRUE then the best solution is always kept in the next generation of solutions (elitism).

tabu genetic algorithm parameter, TRUE or FALSE. If TRUE then the solutions that are saved in tabu memory will not be retried.

tabumemsize genetic algorithm parameter, integer>0. Number of generations to hold in tabu memory.

mutprob genetic algorithm parameter, probability of mutation for each generated solution.

mutintensity mean of the poisson variable that is used to decide the number of mutations for each cross.

niterations genetic algorithm parameter, number of iterations.

minitbefstop genetic algorithm parameter, number of iterations before stopping if no change is observed in criterion value.

niterreg genetic algorithm parameter, number of iterations to use regressions, an integer with minimum value of 1

lambda scalar shrinkage parameter ($\lambda > 0$).

plotiters plot the convergence: TRUE or FALSE. Default is TRUE.

plottype type of plot, default is 1. possible values 1,2,3.

errorstat optimality criterion: One of the optimality criterion. Default is "PEVMEAN". It is possible to use user defined functions as shown in the examples.

mc.cores number of cores to use.

InitPop a list of initial solutions

tolconv if the algorithm cannot improve the errorstat more than tolconv for the last minitbefstop iterations it will stop.

C Contrast Matrix.

Vg covariance matrix between traits generated by the relationship K (only for multi-trait version of PEVMEANMM).

Ve residual covariance matrix for the traits (only for multi-trait version of PEVMEANMM).

Fedorov Whether the Fedorovs exchange algorithm from AlgDesign Package should be used for initial solutions.

Value

A list of length nelite+1. The first nelite elements of the list are optimized training samples of size n_{train} and they are listed in increasing order of the optimization criterion. The last item on the list is a vector that stores the minimum values of the objective function at each iteration.

Note

The GA does not guarantee convergence to globally optimal solutions and it is highly recommended that the algorithm is replicated to obtain ”good” training samples.

Author(s)

Deniz Akdemir

Examples

```
## Not run:
###################### Example for three level designs for the
#second order model in two factors with a square design region
X<-matrix(0,nrow=3^2,ncol=5)
ij=0

for (i in -1:1){
  for (j in -1:1){
    ij=ij+1
    X[ij,]<-c(i,j, i^2,j^2, i*j)

  }
}
X<-cbind(1,X)
D<-as.matrix(dist(X))
K<-tcrossprod(X)
rownames(K)<-colnames(K)<-rownames(D)<-colnames(D)<-rownames(X)<-paste(”x”,1:3^2, sep=””)
X
library(STPGA)
ListTrain1<-GenAlgForSubsetSelectionNoTest(P=X,ntoselect=4, InitPop=NULL,
             npop=100, nelite=5, mutprob=.5, mutintensity = 1,
             niterations=200,minitbefstop=20, tabu=F,tabumemsize = 0,plotiters=F,
             lambda=1e-9,errorstat=”DOPT”, mc.cores=1)


ListTrain2<-GenAlgForSubsetSelectionNoTest(P=solve(K+1e-6*diag(ncol(K))),ntoselect=4, InitPop=NULL,
            npop=100, nelite=5, mutprob=.5, mutintensity = 1,
            niterations=200,minitbefstop=20, tabu=F,tabumemsize = 0,plotiters=F,
            lambda=1,errorstat=”CDMEANMM”, mc.cores=1)


  par(mfrow=c(1,2),mar=c(1,1,1,1))
  labelling1<-rownames(X)%in%ListTrain1[[1]]+1
  plot(X[,2], X[,3], col=labelling1, pch=2*labelling1,cex=2*(labelling1-1),
   xlab="", ylab="", main=”DOPT”, cex.main=.7,xaxt='n',yaxt='n')
    for (i in -1:1){
```

```
  abline(v=i, lty=2)
  abline(h=i,lty=2)
    }
    labelling2<-rownames(X)%in%ListTrain2[[1]]+1
  plot(X[,2], X[,3], col=labelling2, pch=2*labelling2,cex=2*(labelling2-1),
   xlab="", ylab="", main="CDMEANMM", cex.main=.7,xaxt='n',yaxt='n')
    for (i in -1:1){
  abline(v=i, lty=2)
  abline(h=i,lty=2)
    }

########################Dopt design three level designs for the second order
#model in two factors with a square design region

par(mfrow=c(2,2),mar=c(1,1,1,1))
 for (ntoselect in c(5,6,7,8)){
  ListTrain<-GenAlgForSubsetSelectionNoTest(P=X,ntoselect=ntoselect, InitPop=NULL,
             npop=10, nelite=3, mutprob=.5, mutintensity = 1,
             niterations=200,minitbefstop=200, tabu=F,tabumemsize = 0,plotiters=F,
             lambda=1e-9,errorstat="DOPT", mc.cores=1)

  labelling<-rownames(X)%in%ListTrain[[1]]+1
 plot(as.numeric(X[,2]), as.numeric(X[,3]), col=labelling, pch=2*labelling,cex=2*(labelling-1),
   xlab="", ylab="", main="DOPT", cex.main=.7,xaxt='n',yaxt='n')
    for (i in -1:1){
  abline(v=i, lty=2)
  abline(h=i,lty=2)
    }

}

par(mfrow=c(1,1))


## End(Not run)
```

GenerateCrossesfromElites

Generate crosses from elites

Description

Given a list of elite training sets, list of candidates the function makes npop new solutions by using crossover and mutation operators.

Usage

```
GenerateCrossesfromElites(Elites, Candidates, npop, mutprob, mc.cores = 1,
                   mutintensity = 1, memoryfortabu = NULL)
```

Arguments

`Elites` a list of elite training sets

`Candidates` a vector of identifiers of the individuals in the candidate set.

`npop` number of training sets to generate.

`mutprob` point mutation probability for each individual generated. Only one mutation per solution is allowed.

`mc.cores` number of cores to use.

`mutintensity` mean of the poisson variable that is used to decide the number of mutations for each cross.

`memoryfortabu` tabu memory

Value

A list of npop training sets.

Author(s)

Deniz Akdemir

`makeonecross` *Make a cross from two solutions and mutate.*

Description

Given two training sets, identifiers for candidates, this function makes a new solution using crossover and one point mutation with probability mutprob. Only one mutation is allowed.

Usage

```
makeonecross(x1, x2, Candidates, mutprob, mutintensity=2)
```

Arguments

`x1` a vector of identifiers selected from the candidate set.

`x2` a vector of identifiers selected from the candidate set that has the same length as x1.

`Candidates` vector of identifiers for individuals in the candidate set.

`mutprob` point mutation probability for each individual generated. Only one mutation per solution is allowed.

`mutintensity` mean of the poisson variable that is used to decide the number of mutations for each cross.

Author(s)

Deniz Akdemir

WheatData *Adult plant height (estimated genetic values) for 1182 elite wheat lines*

Description

Containing the phenotypic observations ’Wheat.Y’, markers ’Wheat.M’ and genetic relationships ’Wheat.K’. The 4670 markers available for these 200 genotypes were pre-porecessed for missingness and minor ellele frequencies, coded numerically as -1, 0, and 1; the relationship genomic relationship matrix was calculated from these markers.

Source

This dataset was obtained from `https://triticeaetoolbox.org/`.

Index