High-Dimensional-Covariance-Estimation.R

hecto

2020-02-01

AR1matrix simple function to generate an AR1 matrix of size pxp with given correlation @param correlation a correlation between 0 and 1

@param p a number of features

@return AR1covariance, a p by p covariance matrix

```
AR1matrix=function(correlation,p){
  AR1covariance=matrix(rep(0,p),ncol = p,nrow = p)
  for(i in 1:p){
    for (j in 1:p) {
      AR1covariance[i,j]=correlation^(abs(i-j))
    }
  }
  return(AR1covariance)
}
```

MA1matrix simple function to generate an MA1 matrix of size pxp with given correlation

@param correlation number between 0 and 1

@param p number of features

@return MA1covariance, a p by p covariance matrix

```
MA1matrix=function(correlation,p){
  MA1covariance=matrix(rep(0,p),ncol = p,nrow = p)
  for(i in 1:p){
    for (j in 1:p) {
      MA1covariance[i,j]=correlation*as.integer(abs(i-j)==1)+as.integer(i==j)
    }
  }
  return(MA1covariance)
}
```

getObsesrvations generate n observations from a multivariate normal

@param n number of sample

@param truecovariance a covariance matrix such as MA1covariance

@return observations, a n by p data matrix

```
getObservations=function(n,truecovariance){
  p=dim(truecovariance)[1]
  mu=rep(0,p)
  sigma=truecovariance
  observations=mvrnorm(n,mu,sigma)
  return(observations)
}
```

SampleCovariance compute the sample covariance matrix estimator given some observations

@param observations generated from a multivariate normal distribution

@param p number of features

@param n number of sample

@return a p by p sample covariance matrix estimator

```
SampleCovariance=function(observations,p,n){
  samplecovariance = rep(0,p) %o% rep(0,p)
  for(j in 1:n){samplecovariance = samplecovariance + (observations[j,] %o% observations[j,])}
  return((1/(n))*samplecovariance)}
```

ledoitwolf compute the Ledoit-Wolf estimator given a sample covar and some observations

@param samplecovariance a p by p sample covariance matrix

@param observations a n by data matrix

@param n number of sample

@return the Ledoit Wolf covariance matrix estimator

```
ledoitwolf=function(samplecovariance,observations,n){
  p=dim(samplecovariance)[1]
  identity=diag(1,p,p)
  m=tr(t(samplecovariance))/p
  dsquared=tr((samplecovariance-m*identity)%*%t(samplecovariance-m*identity))/p
  bbarsquared=0
  for(k in 1:n){
    bbarsquared=bbarsquared+tr(((observations[k,]%o%observations[k,])-
                                samplecovariance)%*%
                               t(observations[k,]%o%observations[k,]-
                                   samplecovariance))/p
  }
  bbarsquared=1/(n^2)*bbarsquared
  bsquared=min(c(dsquared,bbarsquared))
  asquared=dsquared-bsquared
  LWestimator=((bsquared/dsquared)*m*identity)+(asquared/dsquared*samplecovariance)
  return(LWestimator)
}
```

HardThresholdingCovar compute the hard thresholding covariance estimator with threshold t @param SampleCovariance a p by p sample covariance matrix

@param t a non negative threshold

@return the sample covariance matrix thresholded at t (Hard threshold covariance estimator)

```
HardThresholdingCovar=function(SampleCovariance,t){
  thresholdedCovar=SampleCovariance*(abs(SampleCovariance)>=t)
  return(thresholdedCovar)
}
```

SoftThresholdingCovar compute the soft thresholding covariance estimator with threshold t @param SampleCovariance a p by p sample covariance matrix

@param t a non-negative threshold t

@return soft thresholded sample covariance matrix at t (soft threshold covariance estimator)

```
SoftThresholdingCovar=function(SampleCovariance,t){
  p=dim(SampleCovariance)[1]
  SoftThresholdedCovar=matrix(0,p,p)
  for (i in 1:p) {
    for (j in 1:p) {
      SoftThresholdedCovar[i,j]=sign(SampleCovariance[i,j])*
        (max(SampleCovariance[i,j],0))
    }
  }
  return(SoftThresholdedCovar)
}
```

someMetrics compute a vector of metrics of an estimator with respect to a true covariance @param estimatorcovariance a p by p covariance estimator (LW, hard threshold,...)

@param truecovar a p by p covariance matrix

@return a vector of norms between the covariance estimator and its population counterpart

```
someMetrics=function(estimatorcovariance,truecovar){
  AbsCosAngleBetweenfirstPC=abs(prcomp(estimatorcovariance)$x[1,]%*%
                                  prcomp(truecovar)$x[1,]/
                                  (Norm(truecovar,p=2)*
                                     Norm(estimatorcovariance,p=2)))
  MSE=MSE(estimatorcovariance,truecovar)
  froebdif=norm(estimatorcovariance-truecovar,"F")
  specdifference=norm(estimatorcovariance-truecovar,"2")
  minEV=minimumEigenValue(estimatorcovariance)
  return(c(AbsCosAngleBetweenfirstPC,MSE,froebdif,specdifference,minEV))
}
```

BIGTABLE generate a table of metrics with respect to 4 estimators and 1 true covariance @param truecovar a p by p covariance matrix such as AR1 ...

@param n sample size

@return a dataframe of metrics (frobenis difference ...) betweeen a covariance matrix and 4 covariance matrix estimator

```
BIGTABLE=function(truecovar,n){
  finallw=finalht=finalst=finalsc=rep(0,5)
  for(i in 1:100){
    observations=getObservations(n,truecovar)
```

```
    p=length(observations[1,])
    samplecov=SampleCovariance(observations,p,n)
    lw=ledoitwolf(samplecov,observations,n)
    ht=HardThresholdingCovar(samplecov,getThresholdValue(observations,10,p))
    st=SoftThresholdingCovar(samplecov,getThresholdValueSoft(observations,10,p))
    finalsc=finalsc+someMetrics(samplecov,truecovar)
    finallw=finallw+someMetrics(lw,truecovar)
    finalht=finalht+someMetrics(ht,truecovar)
    finalst=finalst+someMetrics(st,truecovar)
  }
  df=data.frame(0.01*finalsc,0.01*finallw,0.01*finalht,0.01*finalst)
  names(df)=c("Sample Covariance","Ledoit-Wolf","Hard Thresholding",
              "Soft Thresholding")
  return(df)
}
```

objectiveFunctionRt the objective function to estime the the hard threshold t from method 3.1.2 this function is only ran through an optimization function i.e indirectly @param t

@param observations

@param N

```
objectiveFunctionRt=function(t,observations,N){
  n=length(observations[,1])
  p=length(observations[1,])
  Rt=0
  indexes=1:n
  n1=round(n-(n/log(n)))
  n2=n-n1
  for(v in 1:N){
    indextodelete=sample(indexes,n1,replace = TRUE)
    ##### get the training set and validation set ######
    observationsTRAIN=observations[indextodelete,]
    observationsTEST=observations[-indextodelete,]
    Sigma1v=SampleCovariance(observationsTRAIN,p,n1)
    Sigma2v=SampleCovariance(observationsTEST,p,n2)
    Rt=Rt+norm(HardThresholdingCovar(Sigma1v,t)-Sigma2v,"f")^2
  }
  return((1/N)*Rt)
}
```

objectiveFunctionRtforSoft the objective function to estime the the soft threshold t from method 3.1.2 this function is only ran through an optimization function i.e indirectly @param t

@param observations

@param N

```
objectiveFunctionRtforSoft=function(t,observations,N){
  n=length(observations[,1])
  p=length(observations[1,])
  Rt=0
  indexes=1:n
  n1=round(n-(n/log(n)))
```

```
  n2=n-n1
  for(v in 1:N){
    indextodelete=sample(indexes,n1,replace = TRUE)
    observationsTRAIN=observations[indextodelete,]
    observationsTEST=observations[-indextodelete,]
    Sigma1v=SampleCovariance(observationsTRAIN,p,n1)
    Sigma2v=SampleCovariance(observationsTEST,p,n2)
    Rt=Rt+norm(SoftThresholdingCovar(Sigma1v,t)-Sigma2v,"f")^2
  }
  return((1/N)*Rt)
}

#### function to perform the minimization of method 3.1.2 ####
getThresholdValue=function(observations,N,p){
  o=optimize(objectiveFunctionRt,observations,N,interval = c(0,1))
  t=o$minimum
  return(t)
}
getThresholdValueSoft=function(observations,N,p){
  o=optimize(objectiveFunctionRtforSoft,observations,N,interval = c(0,1))
  t=o$minimum
  return(t)
}
```

```
##### get the minimum eigenvalue of a matrix #####
minimumEigenValue=function(matrix){
  decomposition=eigen(matrix)
  eigenvalues=decomposition$values
  return(min(eigenvalues))}

##### compute some graphs ######
getgraphMinEV=function(samplesize,maxfeatures){
  n=samplesize
  q=maxfeatures
  x_=y_=rep(0,q)
  for(p in 2:q){
    mu=rep(0,p)
    sigma=diag(x=1,p,p)
    observations=mvrnorm(n,mu,sigma)
    samplecovar=SampleCovariance(observations,p,n)
    lw=ledoitwolf(samplecovar,observations,n)
    x_[p]=minimumEigenValue(lw)
    y_[p]=minimumEigenValue(samplecovar)
  }
  plot(y_[2:q],
       xlab="value of p",
       ylab="minimum eigenvalue",
       type="l",
       col="blue")
  lines(x_[2:q], col="red")
  legend("right",
         c("min eigenvals for ledoitwolf","min eigenvals for samplecov"),
         fill=c("red","blue")
```

```
  )
}
getgraphEVspread=function(samplesize,numberoffeatures){
  n=samplesize
  p=numberoffeatures
  mu=rep(0,p)
  sigma=diag(x=1,p,p)
  observations=mvrnorm(n,mu,sigma)
  samplecovar=SampleCovariance(observations,p,n)
  lw=ledoitwolf(samplecovar,observations,n)
  eigenvaluesLedoit=sort(eigen(lw)$values,decreasing = TRUE)
  eigenvaluesSampleCov=sort(eigen(samplecovar)$values,decreasing = TRUE)
  trueeigenvalues=sort(eigen(sigma)$values,decreasing = TRUE)
  plot(eigenvaluesSampleCov,xlab = "index of decreasing ordered eigenvals",
       ylab="eigenvalue",type="l",col="blue")
  lines(trueeigenvalues,lty=20)
  'points(eigenvaluesLedoit,col="red",cex=1.3)'
  legend("topright",c("true eigenvals","ledoit wolf eigenvals",
                      "sample covar eigenvals"),
         fill=c("black","red","blue"))
}
```