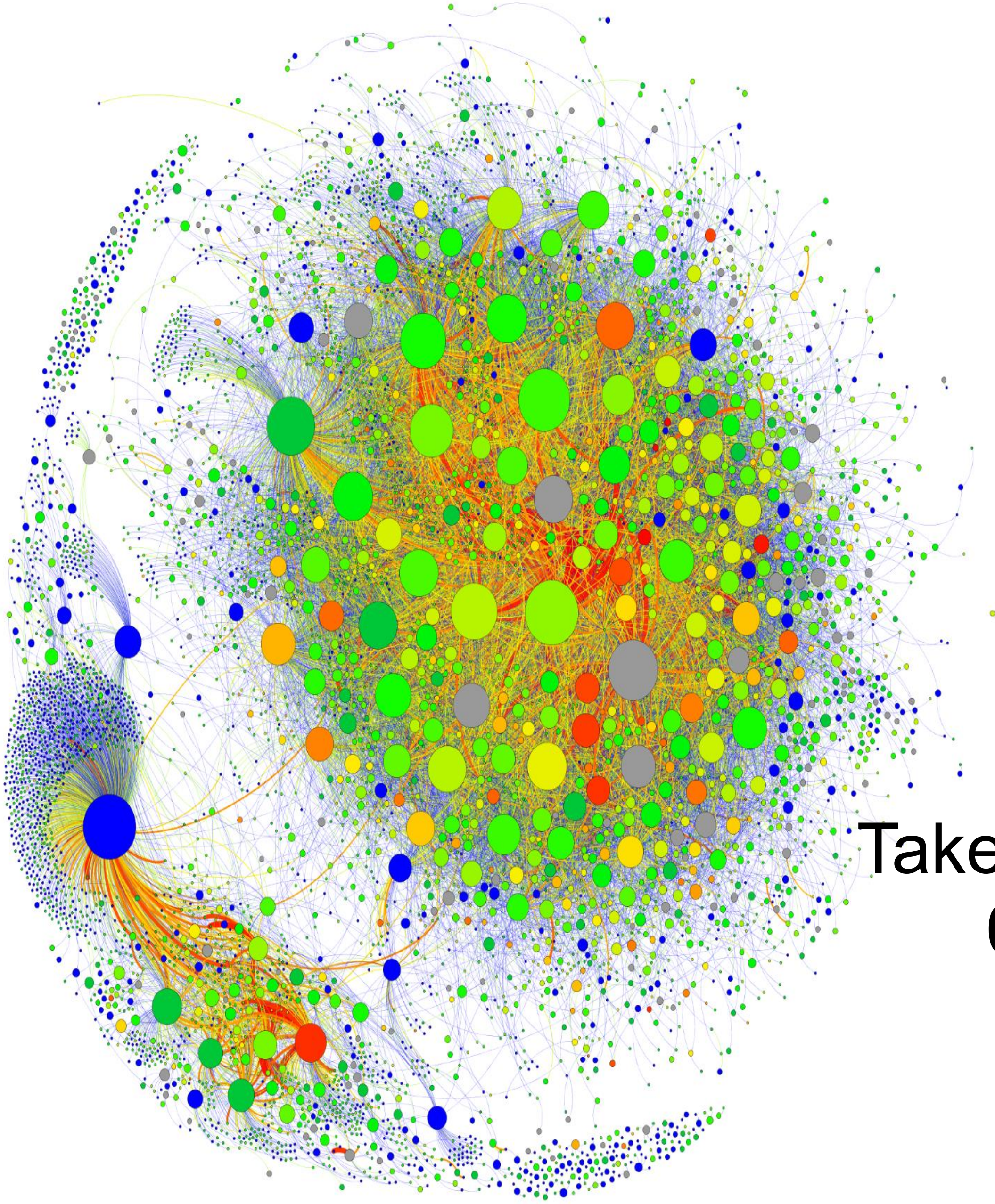


Take-Home exam 2
03/09/2015

Computational Data
Analysis Spring 2015

DATA MANIPULATION

I used the following code to perform the Data manipulation:

```
###################
# Data Manipulation
###################

# Removing completely missing columns
myData1=myData[,colSums(is.na(myData))<nrow(myData)]
# Removing completely missing rows
myData2=myData1[rowSums(is.na(myData1[2:dim(myData1)[2]]))<ncol(
# Removing Columns with constant predictors
myData3=myData2[,apply(myData2,2,function(xx) {
  if(var(xx,na.rm=TRUE)<1e-16){
    return(FALSE)}
    else{return(TRUE)}
  })]
# Removing Collinear columns
CorMat=cov2cor(var(myData3[,2:dim(myData3)[2]],na.rm=TRUE))
myL=matrix(TRUE,1,dim(myData3)[2])
for (i in 1:{dim(CorMat)[2]-1}){
  for (j in {i+1}:dim(CorMat)[2]){
    if(abs(CorMat[i,j]-1)<=1e-4){
      myL[i+1]=FALSE
      if(i<dim(CorMat)[2]-1){
        i=i+1
      }
      }
    }
}
myData4=myData3[,myL]
```

This yields the following results:

- Number of missing columns removed : 3
- Number of missing rows removed : 4
- Number of columns with constant predictors: 0
- Number of collinear columns removed: 17

MULTIPLE IMPUTATION

We use the R library mi to generate the 3 inputed data sets. Then we average the 2 baseline measurements of each v1 texture and compute the difference form average baseline to follow-up for each v1 texture.
This yields a new dataset of **150 new predictors** (75 couples of average of 2 baselines and difference with follow-up). I created a new Data set “AverageDiff” that contains both the response and the new predictors

```
###################
# Multiple Imputation
###################

library(mi)
myInfo=mi.info(myData4)
myInfo=update(myInfo,"type",as.list(rep("predictive-mean-matching",dim(myData4)[2])))
myMIs=mi(myData4,myInfo)
misStar=mi.completed(myMIs)
Data1=mi.data.frame(myMIs, m=1)
Data2=mi.data.frame(myMIs, m=2)
Data3=mi.data.frame(myMIs, m=3)
DataSets=list(Data1,Data2,Data3)
AverageDiffs=vector("list",3)
for (j in 1:3){
        Data=DataSets[[j]]
        FlupIndex=grep("followUp",attributes(Data)$names)
        Bs1Index=grep("baseline1",attributes(Data)$names)
        Bs2Index=grep("baseline2",attributes(Data)$names)
        AverageDiff=Data[1]
        for (i in 1:length(FlupIndex)){
          texture=attributes(Data)$names[FlupIndex[i]]
          Nbchar=nchar(texture)
          texture1=substr(texture, 1,(Nbchar-9))
          indexbs1=Bs1Index[which(grepl(texture1,attributes(Data)$names[Bs1Index]))]
          indexbs2=Bs2Index[which(grepl(texture1,attributes(Data)$names[Bs2Index]))]
          a=b=c=0
          if(length(indexbs1)>0){
            a=Data[,indexbs1[1]]
         }
         if(length(indexbs2)>0){
           b=Data[,indexbs2[1]]
         }
         txt1=paste(texture1,"_Average",sep="")
         txt2=paste(texture1,"_Diff",sep="")
         if(length(a)>1 && length(b)>1){
           AverageDiff[txt1]=(a+b)/2
           AverageDiff[txt2]=Data1[,FlupIndex[i]]-(a+b)/2
         }
         if(length(a)>1 && length(b)<=1){
           AverageDiff[txt1]=a
           AverageDiff[txt2]=Data[,FlupIndex[i]]-a
         }
         if(length(a)<=1 && length(b)>1){
           AverageDiff[txt1]=b
           AverageDiff[txt2]=Data[,FlupIndex[i]]-b
         }
       }
       AverageDiffs[[j]]=AverageDiff
```

LASSO VIA MODIFICATION

Based on the algorithm given in the notes, I create a function **“MyLARS(Data,Type=(lasso,lars)”,** that performs both the original LARS algorithm, and the LASSO modification that removes variables from the active sets of predictors:

```
####################################################
# Lasso via Modification to Least Angle Regression #
####################################################
normV <- function(x) sum(abs(x))

NoInfErr<-function(Response,MU){
  MeanR=mean(Response)
  Nb=length(Response)
  sum0=0
  for (k in 1:Nb){
    Resp=matrix(Response[k],Nb,1)
    sum0=sum0+normV(Resp-MeanR-MU)
  }

  return(sum0/Nb^2)
}

myLARS<-function(DaTa,Type=c("lasso","lars")){
  DaTa=Datx
  Predictors=Data[,2:dim(Data)[2]]
  NbPredictors=dim(Data)[2]-1
  NbPat=dim(Data)[1]
  Beta=matrix(0,NbPredictors,1)
  NbSteps=min(NbPredictors,NbPat-1)
  ActiveSets=matrix(0,NbSteps,1)
  ActiveSets0=matrix(0,NbSteps,1)
  mu=matrix(mean(Data[,1]),NbPat,1)
  CompSet=1:{NbPredictors}
  betas=vector("list")
  tis=c(0)
  AppErr=c()
  NoInfErr=c()
  Response=as.vector(Data[,1])
if (Type=="lasso"){
    Iter=0
    Iter2=1
    AllSets=c()
    jtild=0
    comp=0
    while(Iter<NbSteps){
      Iter2=Iter2+1
      ytild=Data[,1]-mu
      if(jtild==0){
        if(comp==1){
          CompSet=CompSet[which(CompSet!=Jtild)]
          comp=0
        }
        Iter=Iter+1
        correl=apply(Predictors[,CompSet],2,function (xx){
          return(abs(t(xx)%*%ytild))})
        ActiveSets[Iter]=CompSet[which(correl==max(correl))]
        AllSets=c(AllSets,ActiveSets[Iter])
        as=ActiveSets[1:Iter][which(ActiveSets[1:Iter]!=0)]
```

```
}else{
  AllSets=c(AllSets,-as[jtild])
  ActiveSets=c(ActiveSets[which(ActiveSets!=as[jtild])],0)
  as=ActiveSets[1:Iter]
jtild=0
comp=1}

  if(length(as)>1){
    DesignM=apply(Predictors[,as],2,function (xx){
      return(sign(t(xx)%*%ytild)*xx)
    })}else{
      DesignM=sign(t(Predictors[,as])%*%ytild)*Predictors[,as]
    }

  ck=t(Predictors)%*%ytild
  Chat=max(abs(t(DesignM)%*%ytild))
  a=t(Predictors)%*%DesignM%*%solve(t(DesignM)%*%DesignM)%*%matrix(1,length(as),1)
  for (k in 1:length(as)){
    CompSet=CompSet[which(CompSet!=as[k])]
  }
  set=matrix(0,2*length(CompSet),1)
  i=1
  for(index in CompSet){
    set[i]=(Chat-ck[index])/(1-a[index])
    set[i+1]=(Chat+ck[index])/(1+a[index])
    i=i+2
  }
  c=min(set[which(set>0)])
  u=DesignM%*%(solve(t(DesignM)%*%DesignM))%*%matrix(1,length(as),1)
  b=c*(solve(t(DesignM)%*%DesignM))%*%matrix(1,length(as),1)
  btild=solve(t(DesignM)%*%DesignM)%*%matrix(1,length(as),1)
  d=matrix(0,NbPredictors,1)
  cj=matrix(0,length(as),1)
  for(ind in 1:length(as)){
    d[as[ind]]=btild[ind]
    cj[ind]=-Beta[as[ind]]/d[as[ind]]
  }

  cjpos=which(cj>1e-8)
  cjmin=Inf
  if(length(cjpos)>0){
    cjmin=min(cj[cjpos])
  }
  if(cjmin<c){
    mu=mu+cjmin*u
    b=cjmin*(solve(t(DesignM)%*%DesignM))%*%matrix(1,length(as),1)
    Beta[as]=Beta[as]+cjmin*btild[1:length(as)]
    Beta2=Beta
    Beta2[as]=sign(t(Predictors[,as])%*%ytild)*Beta2[as]
    tis=c(tis,sum(abs(Beta2)))
    betas[[Iter2]]=Beta2
    AppErr=c(AppErr,normV(Response-MeanR-Predictors%*%Beta2)/NbPat)
    NoInfErr=c(NoInfErr,NoInfErr(Response,Predictors%*%Beta2))
    jtild=which(cj==cjmin)
    Jtild=as[jtild]
    Iter=Iter-1

  }else{
    Beta[as]=Beta[as]+b[1:length(as)]
    Beta2=Beta
    Beta2[as]=sign(t(Predictors[,as])%*%ytild)*Beta2[as]
    tis=c(tis,sum(abs(Beta2)))
    betas[[Iter2]]=Beta2
    AppErr=c(AppErr,normV(Response-MeanR-Predictors%*%Beta2)/NbPat)
    NoInfErr=c(NoInfErr,NoInfErr(Response,Predictors%*%Beta2))
```

```
            mu=mu+c*u
            }
          CompSet=1:{NbPredictors}
          for (k in 1:length(as)){
            CompSet=CompSet[which(CompSet!=as[k])]
          }

          if(Iter==NbSteps){
            Beta=sign(t(Predictors[,as])%*%ytild)*solve(t(DesignM)%*%DesignM)%*%t(DesignM)%*
            Beta2=matrix(0,NbPredictors,1)
            for(i in 1:length(as)){
              Beta2[as[i]]=Beta[i]
            }

            betas[[Iter2]]=Beta2
            }
        }

    }
    object=list(Betas=betas,Ts=tis,AppErrs=AppErr,NoInfErrs=NoInfErr)
    class(object)="MyLars"
    object
  }
```

The outputs of my function MyLARS are:

- Betas: list of the Coefficient for different iterations $t = \sum_{i=1}^{p}|\beta_i|$
- Ts: All the values of $t = \sum_{i=1}^{p}|\beta_i|$
- AppErr : Apparent mean absolute predictive error Rate for all t
- NoInfoErr: No information Error Rate for all t

Here are the plots of the Lasso path, the Apparent mean absolute predictive error Rate and No information Error rate for the 3 data sets (before performing Bootstrap):

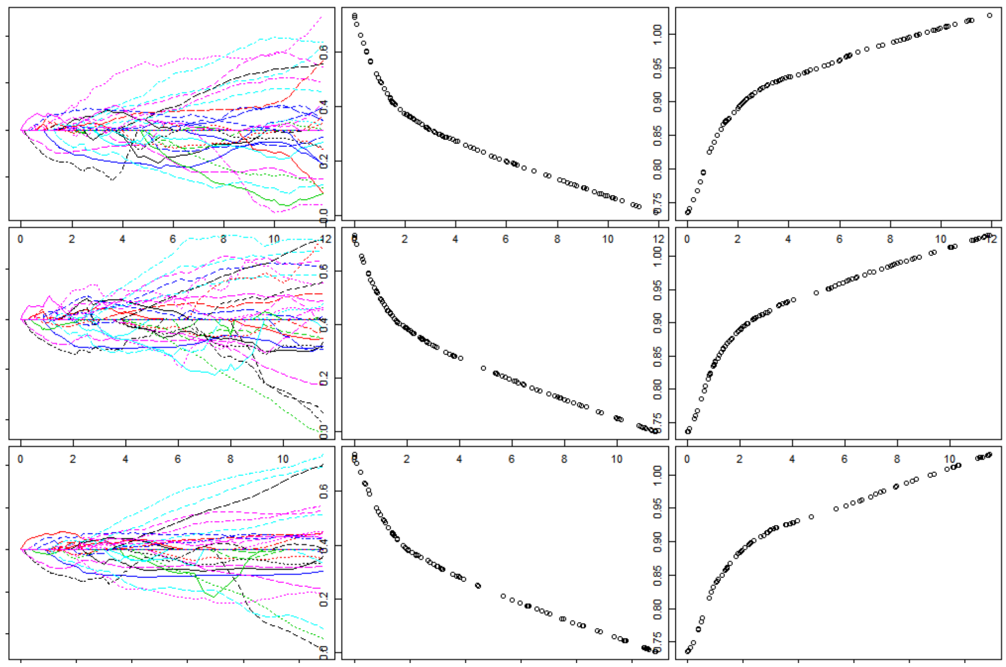
```
#######################
# Lasso Paths plots
######################
dev.new(width=8,height=8)
par(mai=c(0.05,0.05,0.05,0.05),mfrow=c(3,3))
NbTs=29
NbPats=length(AverageDiffs[[1]][,1])
for(j in 1:3){
  Betas=Results[[j]]$Betas
  Tis=Results[[j]]$Ts
  AppEr=Results[[j]]$AppErrs
  NoInfEr=Results[[j]]$NoInfErrs
  #plot(test[[j]])
  Ts=cbind(Tis)[1:NbTs]
  MBETAS=matrix(0,NbTs,NbPredictors)
  for(i in 1:NbTs){
    MBETAS[i,]=t(Betas[[i]])
  }
  matplot(Ts,MBETAS,type="l")
  plot(Tis,AppEr)
  plot(Tis,NoInfEr)
}
```

Lasso Paths Apparent mean Error No Information Error

Data Set 1

Data Set 2

Data Set 3



ACCURACY ASSESSMENT AND MODEL SELECTION

For each multiply inputed dataset, we generate B=25 bootstrap samples. Then we use them to calculate the so-called leave-one-out bootstrap estimate predictive error for every t :

$$\widehat{\mathrm{Err}}^{(1)} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{|C^{-i}|}\sum_{b\in C^{-i}} Q(y_i, \hat{\mu}^b(\mathbf{x}_i)).$$

Since we have different $t = \sum_{i=1}^{p}|\beta_i|$ for every bootstrap. We need to use linear interpolation in order to get the same t values (I use the function “approx” for this purpose

Here is the part of the code to generate 25 boostraps for every data set:

```
###############
# BootStrap #
###############

B=25
Response=Response=as.vector(AverageDiffs[[1]][,1])
Mean=mean(Response)
NbPat=length(AverageDiffs[[1]][,1])
btstraps=vector("list",3)
BSTRPS=vector("list",3)
for(j in 1:3){
  Data=AverageDiffs[[j]]
  Data0=apply(Data[,2:{dim(Data)[2]}],2,function(xx){
    return((xx-mean(xx,na.rm=TRUE))/sd(xx,na.rm=TRUE))
  })
  Data[,2:dim(Data)[2]]=Data0
  Data=as.matrix(Data)
  btstrap=matrix(0,B,NbPat)
  for(i in 1:B){
    btstrap[i,]=sample(NbPat,replace=TRUE)
  }
  btstraps[[j]]=btstrap
  Bstrps=vector("list",B)
  for(k in 1:B){
    NumberIter=length(unique(btstrap[k,]))-1
    #Data2=Data[unique(btstrap[k,]),]
    Data2=Data[btstrap[k,],]
    Data0=apply(Data2[,2:{dim(Data2)[2]}],2,function(xx){
      return((xx-mean(xx,na.rm=TRUE))/sd(xx,na.rm=TRUE))
    })
    Data2[,2:dim(Data2)[2]]=Data0
    Data2=as.matrix(Data2)
    Bstrps[[k]]=myLARS(Data2,"lasso",NumberIter)
  }
  BSTRPS[[j]]=Bstrps
}
```

Here is the code that calculates the leave-one-out bootstrap estimate predictive error for every t :

```
####################################
# Leave-One-Out Bootstrap estimate #
####################################
ERRhat=vector("list",3)
for(j in 1:3){
  Data=AverageDiffs[[j]]
  Data0=apply(Data[,2:{dim(Data)[2]}],2,function(xx){
    return((xx-mean(xx,na.rm=TRUE))/sd(xx,na.rm=TRUE))
  })
  Data[,2:dim(Data)[2]]=Data0
  Data=as.matrix(Data)
  L=c()
  for (bt in 1:25){L=c(L,length(BSTRPS[[j]][[bt]]$Ts))}
  index=which(L==max(L))[1]
  Ts=BSTRPS[[j]][[index]]$Ts
  Err=matrix(0,length(Ts),2)
  for (t in 1:length(Ts)){
    err=0
    N=0
    for(i in 1:NbPat){
      Ci=apply(btstraps[[j]],1,function(xx){return(!any(xx==i))})
      if(sum(Ci)>0){
        bstrp=BSTRPS[[j]][Ci]
        N=N+1
        er=0
        for (btp in bstrp){
          Tsx=btp$Ts
          Bety=as.matrix(btp$Betas)
          Coef=t(sapply(Bety, unlist))
          Coef2=matrix(0,150,1)
        for (c in 1:150){
          Coef2[c]=approx(as.matrix(btp$Ts),as.matrix(Coef[,c]),Ts[t],rule=2)$y
        }
          er=er+abs(Response[i]-Mean-Data[i,2:dim(Data)[2]]%*%Coef2)
        }}}

              er=er/sum(Ci)
            err=err+er
          }
          err=err/N
          Err[t,1]=Ts[t]
          Err[t,2]=err
          }
        ERRhat[[j]]=Err
      }
```

We compute the 0.632+ bootstrap estimate for every data set (function of $t = \sum_{i=1}^{p}|\beta_i|$) based on the relative overfitting rate, no information error rate, apparent error rate and the leave-one-out bootstrap estimate of error:

```
#############################################
#0.632+ bootstrap estimate of expected error#
#############################################

BTErr=vector("list",3)
for(j in 1:3){
  AbsErr=Results[[j]]$AppErrs
  NoInfErr=Results[[j]]$NoInfErrs
  Errhat=ERRhat[[j]]
  Tss=ERRhat[[j]][,1]
  Tss2=Results[[j]]$Ts
  Er=matrix(0,length(Tss),2)
  for (t in 1:length(Tss)){
    errhat=ERRhat[[j]][t,2]
    abserr=approx(as.matrix(Tss2),as.matrix(AbsErr),Tss[t],rule=2)$y
    noinfer=approx(as.matrix(Tss2),as.matrix(NoInfErr),Tss[t],rule=2)$y
    R=(errhat-abserr)/(noinfer-abserr)
    w=0.632/(1-0.368*R)
    Er[t,1]=Tss[t]
    Er[t,2]=(1-w)*abserr+w*errhat
  }
  BTErr[[j]]=Er
}
```

Then we average the three 0.632+ bootstrap estimates of the 3 data sets to estimate the overall expected apparent error rate:

```
#############################################
#overall estimate of expected mean absolute#
#############################################

Ts=BTErr[[1]][,1]
Overall=matrix(0,length(Ts),2)
Overall[,1]=Ts
for(t in 1:length(Ts)){
  erh=BTErr[[1]][t,2]
  for(j in 2:3){
    erh=erh+approx(as.matrix(BTErr[[j]][,1]),as.matrix(BTErr[[j]][,2]),Ts[t],rule=2)$y
  }
  Overall[t,2]=erh/3
}
```

Model Complexity: We choose the value of $t = \sum_{i=1}^{p} |\beta_i|$ that yields a trade-off between the overall error and the number of predictors. We need to have a sufficiently small error with the least number of predictors. With the other 0.632 bootstrap method, I got t=3

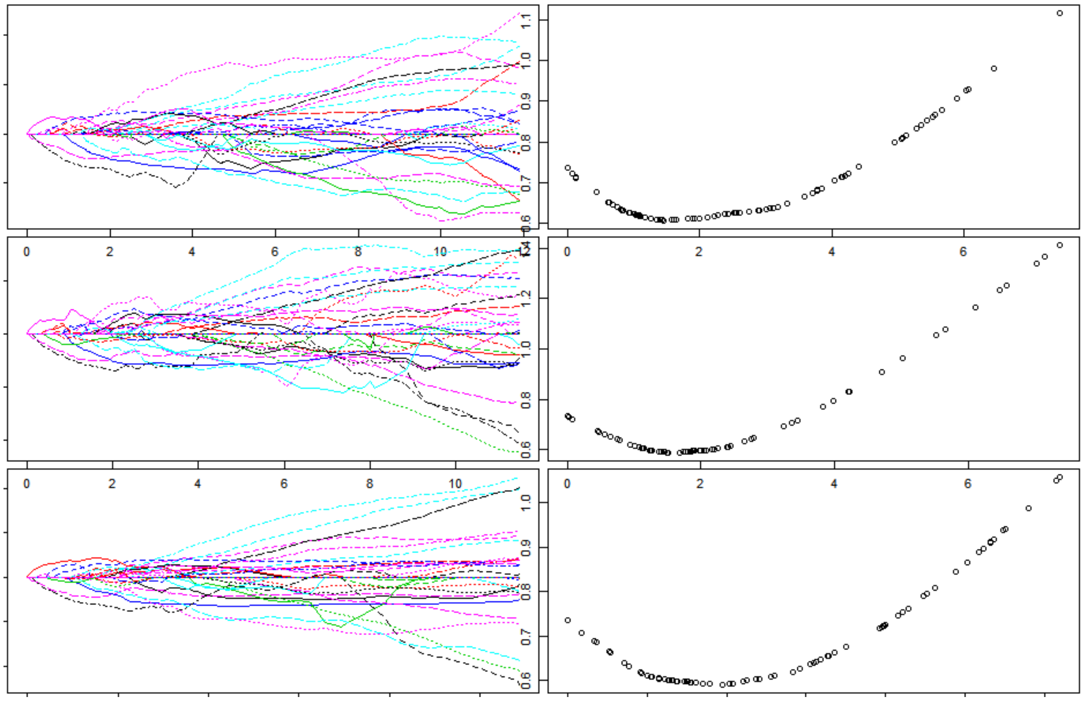
VISUALIZATION

We plot a 3 row by 2 column panel plot: the first column contains the lasso paths for each of the datasets. The second column contains the 0.632 boostrap estimate of expected mean absolute error for the 3 data sets

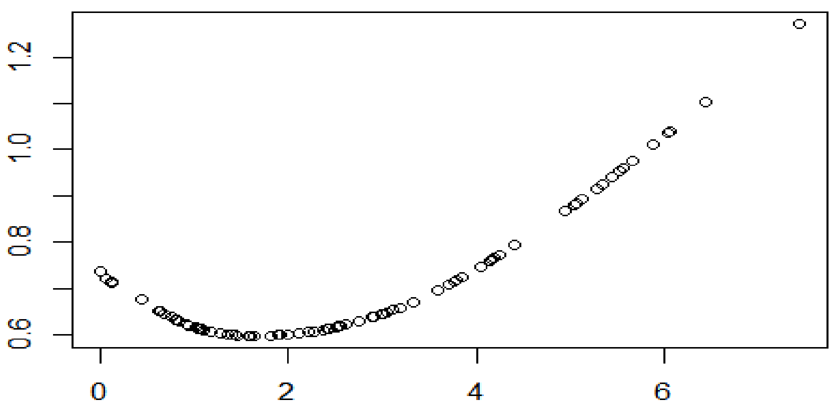
```
############################################
#               Visualization              #
############################################

dev.new(width=5,height=5)
par(mai=c(0.09,0.09,0.09,0.09),mfrow=c(3,2))
NbPats=length(AverageDiffs[[1]][,1])
for(j in 1:3){
  Betas=Results[[j]]$Betas
  Tis=Results[[j]]$Ts
  BtErr=BTErr[[j]]
  MBETAS=t(sapply(Betas, unlist))
  matplot(Tis,MBETAS,type="l")
  plot(BtErr)
}

dev.new(width=10,height=10)
par(mai=c(0.6,0.6,0.6,0.6),mfrow=c(1,1))
plot(Overall2)
```

Since the 0.632 bootstrap with overfitting rate didn’t give good results, I used the orginal definition of the 0.632 bootstrap

$$\widehat{\mathrm{Err}}^{(0.632)} = 0.368 \times \overline{\mathrm{err}} + 0.632 \times \widehat{\mathrm{Err}}^{(1)}$$



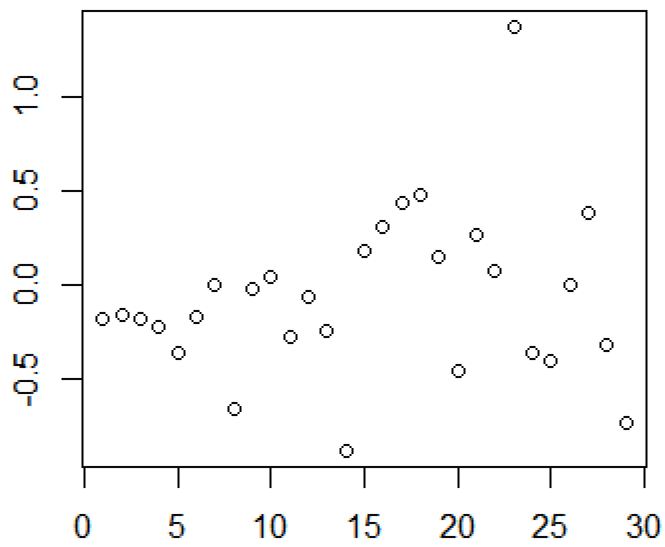
Finally, we plot the overall estimate of expected mean absolute error



INTERPRETATION

Based on the overall estimate of the expected mean absolute error, I chosed a model complexity t=3. Then I averaged the Coefficients of the regression to assess the quality of the fit. I calculated the error between the actual response and the prediction.

```
##############################################
#             Interpretation               #
##############################################
Data=AverageDiffs[[j]]
Data0=apply(Data[,2:{dim(Data)[2]}],2,function(xx){
  return((xx-mean(xx,na.rm=TRUE))/sd(xx,na.rm=TRUE))
})
Data[,2:dim(Data)[2]]=Data0
Data=as.matrix(Data)
Ts=Results[[1]]$Ts
Estimates=matrix(0,150,1)
T=3
for(j in 1:3){
  Coef2=matrix(0,150,1)
  Betas=Results[[j]]$Betas
  Bet=as.matrix(Betas)
  Coef=t(sapply(Bet, unlist))
  for (c in 1:150){
    Coef2[c]=approxExtrap(as.matrix(Ts),as.matrix(Coef[,c]),T)$y}
  Estimates=Estimates+Coef2
}
Estimates=Estimates/3
Error=Response-Data[,2:dim(Data)[2]]%*%Estimates
plot(Error)

# selected variables
  Selected=attributes(Data[,2:{dim(Data)[2]}])[[2]][[2]][which(abs(Estimates)>1e-8)]
# Largest coefficient
  LargeCoef=which(abs(Estimates)==max(abs(Estimates)))
  LCoef=attributes(Data[,2:{dim(Data)[2]}])[[2]][[2]][LargeCoef]
Data2=Data[,2:{dim(Data)[2]}]
Data2[,LargeCoef]=Data[,LargeCoef+1]+1
change=(Data2-Data[,2:dim(Data)[2]])%*%Estimates
```



These are the predictors selected by the model:

```
"v1_Correlation2_Average"
"v1_Information.Measure.of.Correlation1_Average"
"v1_Information.Measure.of.Correlation1_Diff"
"v1_Cluster.Shade.1_Diff"
"v1_Dissimilarity.1_Diff"
"v1_Difference.Entropy.1_Diff"
"v1_Information.Measure.of.Correlation1.1_Diff"
"v1_Inverse.Difference.Moment.Normalized.1_Diff"
"v1_Long.Run.Emphasis_Average"
"v1_Run.Length.Nonuniformity_Diff"
"v1_Coarseness_Average"
"v1_Coarseness_Diff"
"v1_Complexity_Average"
"v1_Texture.Strength_Average"
"v1_Texture.Strength_Diff"
"v1_Maximum.Intensity_Diff"
"v1_Mean.Intensity_Diff"
"v1_Variance_Average"
"v1_Kurtosis_Average"
"v1_Entropy.2_Average"
"v1_Entropy.2_Diff"
"v1_Relative.Dispersion_Average"
"v1_Large.Zone.Emphasis_Average"
"v1_Large.Zone.Emphasis_Diff"
"v1_Intensity.Variability.or.Nonuniformity_Average"
"v1_Zone.Percentage_Average"
"v1_High.Intensity.Zone.Emphasis_Diff"
"v1_High.Intensity.Large.Zone.Emphasis_Average"
"v1_High.Intensity.Large.Zone.Emphasis_Diff"
```

- The majority of them come from the same couples (average, difference of folllow-baseline)
- The observations 14, 23 and 29 weren't fitted very well (largest error). Once again, it is a linear model, we won't expect it to fit the data very well
- The signs of the coefficients are very important. They show which predictors predict an increase or a decrease of the PFS . We should be cautious about the interpretation of the absolute values of the coefficients. In fact, their impact depends on the range of the predictors (a very large coefficient doesn't entail a large change in the PFS, since the corresponding predictor might be very small (we need to check the range of every predictor)
- The predictor "v1_Relative.Dispersion_Average" has the largest coefficie nt 0.321951
- The predictor "v1_Relative.Dispersion_Average" has the largest absolute coeffici ent with value 0.321951
- If we increase the corresponding variable by 1, the PFS will decrease by a factor of exp(-0.321951)=0.72

THE WHOLE R. CODE

```
###############################################
##    Take Home Exam 2-ISyE 6740-Spring 2015 ##
##    By Yassine RIDOUANE                    ##
##    Due 03/09/2015                         ##
###############################################



# Read in texture and response data
myXData=read.csv("Texture.csv",header=TRUE)
myYData=read.csv("TextureResponse.csv",header=TRUE)
# Restrict attention to particular variables
myLogical=sapply(attributes(myXData)$names,function(varname){
  if(varname %in% c("PatID","TimePoint","PFS")){
    return(TRUE)
  }else{
    return(((length(grep(pattern="v1_",varname))>0))&
           (!((length(grep(pattern="Xa",varname))>0)|
              (length(grep(pattern="DeltaX",varname))>0))))
  }
})
myXData=myXData[,myLogical]
```

```
# Rearrange data
myXtemp=data.matrix(myXData[,6:dim(myXData)[2]])
patientIDsTemp=sort(unique(c(myXData$PatID,myYData$Pat.No)))
timePointsTemp=sort(unique(as.character(myXData$TimePoint)))
PFS=matrix(NA,length(patientIDsTemp),1)
predictors=matrix(NA,length(patientIDsTemp),(dim(myXtemp)[2])*length(timePointsTemp))
counter=1
for(pt in patientIDsTemp){
  index=which(myYData$Pat.No==pt)
  if(length(index)==1){
    PFS[counter]=log(myYData$PFS[index])
  }
  for(jj in 1:length(timePointsTemp)){
    index=which((myXData$TimePoint==timePointsTemp[jj])&
                (myXData$PatID==pt))
    if(length(index)==1){
      predictors[counter,((jj-1)*dim(myXtemp)[2]+1):(jj*dim(myXtemp)[2])]=myXtemp[index,]
    }
  }
  counter=counter+1
}
predictorsStar=apply(predictors,2,function(xx){
  if(sum(!is.na(xx))>=2){
    return((xx-mean(xx,na.rm=TRUE))/sd(xx,na.rm=TRUE))
  }else{
    return(xx)
  }
})
myData=data.frame(PFS,predictorsStar)
myData$PFS=myData$PFS-mean(myData$PFS,na.rm=TRUE)
# Label columns
myNames=c("PFS",
          paste(attributes(myXData)$names[6:dim(myXData)[2]],"_baseline1",sep=""),
          paste(attributes(myXData)$names[6:dim(myXData)[2]],"_baseline2",sep=""),
          paste(attributes(myXData)$names[6:dim(myXData)[2]],"_followUp",sep=""))
attributes(myData)$names=myNames

###################
# Data Manipulation
###################

# Removing completely missing columns
myData1=myData[,colSums(is.na(myData))<nrow(myData)]
# Removing completely missing rows
myData2=myData1[rowSums(is.na(myData1[2:dim(myData1)[2]]))<ncol(myData1)-1,]
# Removing Columns with constant predictors
myData3=myData2[,apply(myData2,2,function(xx) {
```

```
  if(var(xx,na.rm=TRUE)<1e-16){
    return(FALSE)}
    else{return(TRUE)}
  })]
# Removing Collinear columns
CorMat=cov2cor(var(myData3[,2:dim(myData3)[2]],na.rm=TRUE))
myL=matrix(TRUE,1,dim(myData3)[2])
for (i in 1:{dim(CorMat)[2]-1}){
  for (j in {i+1}:dim(CorMat)[2]){
    if(abs(CorMat[i,j]-1)<=1e-4){
      myL[i+1]=FALSE
      if(i<dim(CorMat)[2]-1){
        i=i+1
      }
      }
    }
}
myData4=myData3[,myL]



###################
# Multiple Imputation
###################

library(mi)
myInfo=mi.info(myData4)
myInfo=update(myInfo,"type",as.list(rep("predictive-mean-matching",dim(myData4)[2])))
myMIs=mi(myData4,myInfo)
misStar=mi.completed(myMIs)
Data1=mi.data.frame(myMIs, m=1)
Data2=mi.data.frame(myMIs, m=2)
Data3=mi.data.frame(myMIs, m=3)
DataSets=list(Data1,Data2,Data3)
AverageDiffs=vector("list",3)
for (j in 1:3){
        Data=DataSets[[j]]
        FlupIndex=grep("followUp",attributes(Data)$names)
        Bs1Index=grep("baseline1",attributes(Data)$names)
        Bs2Index=grep("baseline2",attributes(Data)$names)
        AverageDiff=Data[1]
        for (i in 1:length(FlupIndex)){
          texture=attributes(Data)$names[FlupIndex[i]]
          Nbchar=nchar(texture)
          texture1=substr(texture, 1,(Nbchar-9))
          indexbs1=Bs1Index[which(grepl(texture1,attributes(Data)$names[Bs1Index]))]
          indexbs2=Bs2Index[which(grepl(texture1,attributes(Data)$names[Bs2Index]))]
```

```
          a=b=c=0
          if(length(indexbs1)>0){
            a=Data[,indexbs1[1]]
          }
          if(length(indexbs2)>0){
            b=Data[,indexbs2[1]]
          }
          txt1=paste(texture1,"_Average",sep="")
          txt2=paste(texture1,"_Diff",sep="")
          if(length(a)>1 && length(b)>1){
            AverageDiff[txt1]=(a+b)/2
            AverageDiff[txt2]=Data1[,FlupIndex[i]]-(a+b)/2
          }
          if(length(a)>1 && length(b)<=1){
            AverageDiff[txt1]=a
            AverageDiff[txt2]=Data[,FlupIndex[i]]-a
          }
          if(length(a)<=1 && length(b)>1){
            AverageDiff[txt1]=b
            AverageDiff[txt2]=Data[,FlupIndex[i]]-b
          }
        }
        AverageDiffs[[j]]=AverageDiff
}


##################################################
# Lasso via Modification to Least Angle Regression #
##################################################
normV <- function(x) sum(abs(x))

NoInfErr<-function(Response,MU){
  MeanR=mean(Response)
  Nb=length(Response)
  sum0=0
  for (k in 1:Nb){
    Resp=matrix(Response[k],Nb,1)
    MeanR=matrix(mean(Response),Nb,1)
    sum0=sum0+normV(Resp-MeanR-MU)
  }

  return(sum0/Nb^2)
}
library(matrixcalc)
myLARS<-function(DaTa,Type=c("lasso","lars"),NumberIter=0){
  Data=DaTa
  Predictors=Data[,2:dim(Data)[2]]
```

```
  NbPredictors=dim(Data)[2]-1
  NbPat=dim(Data)[1]
  Beta=matrix(0,NbPredictors,1)
  NbSteps=min(NbPredictors,NbPat-1)
  if(NumberIter>0){
    NbSteps=min(NbSteps,NumberIter)}
  ActiveSets=matrix(0,NbSteps,1)
  ActiveSets0=matrix(0,NbSteps,1)
  mu=matrix(mean(Data[,1]),NbPat,1)
  CompSet=1:{NbPredictors}
  betas=vector("list")
  tis=c(0)
  AppErr=c()
  NoInfErr=c()
  Response=as.vector(Data[,1])
  MeanR=matrix(mean(Response),NbPat,1)
  betas[[1]]=Beta
  AppErr=c(AppErr,normV(Response-MeanR-Predictors%*%Beta)/NbPat)
  NoInfErr=c(NoInfErr,NoInfErr(Response,Predictors%*%Beta))
  # Initialization:
  if (Type=="lars"){

    for(iter in 1:NbSteps){
      ytild=Data[,1]-mu
      correl=apply(Predictors[,CompSet],2,function (xx){
        return(abs(t(xx)%*%ytild))})
      ActiveSets[iter]=CompSet[which(correl==max(correl))]
      as=ActiveSets[1:iter]
      if(length(as)>1){
        DesignM=apply(Predictors[,as],2,function (xx){
          return(sign(t(xx)%*%ytild)*xx)
        })}
      else{
        DesignM=sign(t(Predictors[,as])%*%ytild)*Predictors[,as]
      }
      ck=t(Predictors)%*%ytild
      Chat=max(abs(t(DesignM)%*%ytild))

a=t(Predictors)%*%DesignM%*%solve(t(DesignM)%*%DesignM)%*%matrix(1,length(as),1)
      for (k in 1:length(as)){
        CompSet=CompSet[which(CompSet!=as[k])]
      }
      set=matrix(0,2*length(CompSet),1)
      i=1
      for(index in CompSet){
        set[i]=(Chat-ck[index])/(1-a[index])
        set[i+1]=(Chat+ck[index])/(1+a[index])
```

```
      i=i+2
     }
     c=min(set[which(set>0)])
     u=c*DesignM%*%(solve(t(DesignM)%*%DesignM))%*%matrix(1,length(as),1)
     b=c*(solve(t(DesignM)%*%DesignM))%*%matrix(1,length(as),1)

     if(iter>1){

Beta[ActiveSets0]=Beta[ActiveSets0]+sign(t(Predictors[,ActiveSets0])%*%ytild)*b[1:length(Acti
veSets0)]
     }
     Beta[ActiveSets[iter]]=sign(t(Predictors[,as[iter]])%*%ytild)*b[iter]
     tis=c(tis,sum(abs(Beta)))
     betas[[iter+1]]=Beta
     AppErr=c(AppErr,normV(Response-MeanR-Predictors%*%Beta)/NbPat)
     NoInfErr=c(NoInfErr,NoInfErr(Response,Predictors%*%Beta))
     mu=mu+u
     ActiveSets0=ActiveSets[1:iter]
     if(iter==NbSteps){

Beta=sign(t(Predictors[,as])%*%ytild)*solve(t(DesignM)%*%DesignM)%*%t(DesignM)%*%(D
ata[,1]-matrix(mean(Data[,1]),NbPat,1))
      Beta2=matrix(0,NbPredictors,1)
      for(i in 1:length(as)){
       Beta2[as[i]]=Beta[i]
      }
      betas[[iter+1]]=Beta2
     }
   }
  }


if (Type=="lasso"){
   Iter=0
   Iter2=1
   AllSets=c()
   jtild=0
   comp=0
   Check=TRUE
   while(Iter<NbSteps && Check){
     Iter2=Iter2+1
     ytild=Data[,1]-mu
     if(jtild==0){
      if(comp==1){
       CompSet=CompSet[which(CompSet!=Jtild)]
       comp=0
      }
```

```
         Iter=Iter+1
         correl=apply(Predictors[,CompSet],2,function (xx){
           return(abs(t(xx)%*%ytild))})
         ActiveSets[Iter]=CompSet[which(correl==max(correl))]
         AllSets=c(AllSets,ActiveSets[Iter])
         as=ActiveSets[1:Iter][which(ActiveSets[1:Iter]!=0)]
        }else{
         AllSets=c(AllSets,-as[jtild])
         ActiveSets=c(ActiveSets[which(ActiveSets!=as[jtild])],0)
         as=ActiveSets[1:Iter]
        jtild=0
        comp=1}

         if(length(as)>1){
           DesignM=apply(Predictors[,as],2,function (xx){
             return(sign(t(xx)%*%ytild)*xx)
           })}else{
             DesignM=sign(t(Predictors[,as])%*%ytild)*Predictors[,as]
           }

         ck=t(Predictors)%*%ytild
         Chat=max(abs(t(DesignM)%*%ytild))
        if(!{rcond(t(DesignM)%*%DesignM)< 1e-08}){

    a=t(Predictors)%*%DesignM%*%solve(t(DesignM)%*%DesignM)%*%matrix(1,length(as),1)
         for (k in 1:length(as)){
           CompSet=CompSet[which(CompSet!=as[k])]
         }
         set=matrix(0,2*length(CompSet),1)
         i=1
         for(index in CompSet){
           set[i]=(Chat-ck[index])/(1-a[index])
           set[i+1]=(Chat+ck[index])/(1+a[index])
           i=i+2
         }
         c=min(set[which(set>0)])
         u=DesignM%*%(solve(t(DesignM)%*%DesignM))%*%matrix(1,length(as),1)
         b=c*(solve(t(DesignM)%*%DesignM))%*%matrix(1,length(as),1)
         btild=solve(t(DesignM)%*%DesignM)%*%matrix(1,length(as),1)
         d=matrix(0,NbPredictors,1)
         cj=matrix(0,length(as),1)

         for(ind in 1:length(as)){
           d[as[ind]]=btild[ind]
           cj[ind]=-Beta[as[ind]]/d[as[ind]]
         }
```

```
      cjpos=which(cj>1e-8)
      cjmin=Inf
      if(length(cjpos)>0){
        cjmin=min(cj[cjpos])
      }
      if(cjmin<c){
        mu=mu+cjmin*u
        b=cjmin*(solve(t(DesignM)%*%DesignM))%*%matrix(1,length(as),1)
        Beta[as]=Beta[as]+cjmin*btild[1:length(as)]
        Beta2=Beta
        Beta2[as]=sign(t(Predictors[,as])%*%ytild)*Beta2[as]
        tis=c(tis,sum(abs(Beta2)))
        betas[[Iter2]]=Beta2
        AppErr=c(AppErr,normV(Response-MeanR-Predictors%*%Beta2)/NbPat)
        NoInfErr=c(NoInfErr,NoInfErr(Response,Predictors%*%Beta2))
        jtild=which(cj==cjmin)
        Jtild=as[jtild]
        Iter=Iter-1

      }else{
        Beta[as]=Beta[as]+b[1:length(as)]
        Beta2=Beta
        Beta2[as]=sign(t(Predictors[,as])%*%ytild)*Beta2[as]
        tis=c(tis,sum(abs(Beta2)))
        betas[[Iter2]]=Beta2
        AppErr=c(AppErr,normV(Response-MeanR-Predictors%*%Beta2)/NbPat)
        NoInfErr=c(NoInfErr,NoInfErr(Response,Predictors%*%Beta2))
        mu=mu+c*u
        }
      CompSet=1:{NbPredictors}
      for (k in 1:length(as)){
        CompSet=CompSet[which(CompSet!=as[k])]
      }

      if(Iter==NbSteps){

Beta=sign(t(Predictors[,as])%*%ytild)*solve(t(DesignM)%*%DesignM)%*%t(DesignM)%*%(D
ata[,1]-matrix(mean(Data[,1]),NbPat,1))
        Beta2=matrix(0,NbPredictors,1)
        for(i in 1:length(as)){
          Beta2[as[i]]=Beta[i]
        }

        betas[[Iter2]]=Beta2
        }
     }else{Check=FALSE}
     }
```

```
  }
  object=list(Betas=betas,Ts=tis,AppErrs=AppErr,NoInfErrs=NoInfErr)
  class(object)="MyLars"
  object
 }
Results=vector("list",3)
test=vector("list",3)
for (j in 1:3){
  Data=AverageDiffs[[j]]
  Data0=apply(Data[,2:{dim(Data)[2]}],2,function(xx){
    return((xx-mean(xx,na.rm=TRUE))/sd(xx,na.rm=TRUE))
  })
  Data[,2:dim(Data)[2]]=Data0
  Data=as.matrix(Data)
  Results[[j]]=myLARS(Data,"lasso")
  Predictors=Data[,2:dim(Data)[2]]
  library(lars)
  test[[j]]=lars(Predictors,Data[,1],type=c("lasso"))
}


#######################
# Lasso Paths plots
######################
dev.new(width=8,height=8)
par(mai=c(0.05,0.05,0.05,0.05),mfrow=c(3,3))
NbTs=29
NbPats=length(AverageDiffs[[1]][,1])
for(j in 1:3){
  Betas=Results[[j]]$Betas
  Tis=Results[[j]]$Ts
  AppEr=Results[[j]]$AppErrs
  NoInfEr=Results[[j]]$NoInfErrs
  #plot(test[[j]])
  MBETAS=t(sapply(Betas, unlist))
  matplot(Tis,MBETAS,type="l")
  plot(Tis,AppEr)
  plot(Tis,NoInfEr)
}


#####################
#BootStrap
###############

B=25
```

```
Response=Response=as.vector(AverageDiffs[[1]][,1])
Mean=mean(Response)
NbPat=length(AverageDiffs[[1]][,1])
btstraps=vector("list",3)
BSTRPS=vector("list",3)
for(j in 1:3){
  Data=AverageDiffs[[j]]
  Data0=apply(Data[,2:{dim(Data)[2]}],2,function(xx){
    return((xx-mean(xx,na.rm=TRUE))/sd(xx,na.rm=TRUE))
  })
  Data[,2:dim(Data)[2]]=Data0
  Data=as.matrix(Data)
  btstrap=matrix(0,B,NbPat)
  for(i in 1:B){
    btstrap[i,]=sample(NbPat,replace=TRUE)
  }
  btstraps[[j]]=btstrap
  Bstrps=vector("list",B)
  for(k in 1:B){
    NumberIter=length(unique(btstrap[k,]))-1
    #Data2=Data[unique(btstrap[k,]),]
    Data2=Data[btstrap[k,],]
    Data0=apply(Data2[,2:{dim(Data2)[2]}],2,function(xx){
      return((xx-mean(xx,na.rm=TRUE))/sd(xx,na.rm=TRUE))
    })
    Data2[,2:dim(Data2)[2]]=Data0
    Data2=as.matrix(Data2)
    Bstrps[[k]]=myLARS(Data2,"lasso",1000)
  }
  BSTRPS[[j]]=Bstrps
}
library(ggplots2)
library(Hmisc)
# Leave-One-Out Bootstrap
ERRhat=vector("list",3)
for(j in 1:3){
  Data=AverageDiffs[[j]]
  Data0=apply(Data[,2:{dim(Data)[2]}],2,function(xx){
    return((xx-mean(xx,na.rm=TRUE))/sd(xx,na.rm=TRUE))
  })
  Data[,2:dim(Data)[2]]=Data0
  Data=as.matrix(Data)
  L=c()
  for (bt in 1:25){L=c(L,length(BSTRPS[[j]][[bt]]$Ts))}
  index=which(L==max(L))[1]
  Ts=BSTRPS[[j]][[index]]$Ts
  Err=matrix(0,length(Ts),2)
```

```
  for (t in 1:length(Ts)){
   err=0
   N=0
   for(i in 1:NbPat){
    Ci=apply(btstraps[[j]],1,function(xx){return(!any(xx==i))})
    if(sum(Ci)>0){
     bstrp=BSTRPS[[j]][Ci]
     N=N+1
     er=0
     for (btp in bstrp){
      Tsx=btp$Ts
      Bety=as.matrix(btp$Betas)
      Coef=t(sapply(Bety, unlist))
      Coef2=matrix(0,150,1)
     for (c in 1:150){
      Coef2[c]=approxExtrap(as.matrix(btp$Ts),as.matrix(Coef[,c]),Ts[t])$y}
      er=er+abs(Response[i]-Mean-Data[i,2:dim(Data)[2]]%*%Coef2)
     }}

     #err=err+1/sum(Ci)*sum(apply(BSTRPS[[j]][Ci],1,function(xx){
      #Coef=approx(as.matrix(xx$Ts),as.matrix(xx$Betas),Ts[t])$y
      #return(abs(Response[i]-Mean-Data[i,2:dim(Data)[2]]%*%Coef))})))}
     er=er/sum(Ci)
    err=err+er
   }
   err=err/N
   Err[t,1]=Ts[t]
   Err[t,2]=err
   }
  ERRhat[[j]]=Err
 }
library(tuneR)
library(audio)
s10<-load.wave("C:/Users/je/Desktop/001.wav")
play(s10)

   ############################################
   #0.632+ bootstrap estimate of expected error#
   ############################################

   BTErr=vector("list",3)
   BTErr2=vector("list",3)
   for(j in 1:3){
    AbsErr=Results[[j]]$AppErrs
    NoInfErr=Results[[j]]$NoInfErrs
    Errhat=ERRhat[[j]]
    Tss=ERRhat[[j]][,1]
```

```
 Tss2=Results[[j]]$Ts
 Er=matrix(0,length(Tss),2)
 Er2=matrix(0,length(Tss),2)
 for (t in 1:length(Tss)){
  errhat=ERRhat[[j]][t,2]
  abserr=approxExtrap(as.matrix(Tss2),as.matrix(AbsErr),Tss[t])$y
  noinfer=approxExtrap(as.matrix(Tss2),as.matrix(NoInfErr),Tss[t])$y
  #abserr=AbsErr[t]
  #errhat=approxExtrap(as.matrix(Tss),as.matrix(ERRhat[[j]]),Tss2[t])$y
  #noinfer=NoInfErr[t]
  R=(errhat-abserr)/(noinfer-abserr)
  w=0.632/(1-0.368*R)
  Er[t,1]=Tss[t]
  Er2[t,1]=Tss[t]
  Er[t,2]=(1-w)*abserr+w*errhat
  Er2[t,2]=0.632*abserr+0.368*errhat
 }
 BTErr[[j]]=Er
 BTErr2[[j]]=Er2
}

############################################
#overall estimate of expected mean absolute#
############################################

Ts=BTErr[[1]][,1]
Overall=matrix(0,length(Ts),2)
Overall2=matrix(0,length(Ts),2)
Overall[,1]=Ts
Overall2[,1]=Ts
for(t in 1:length(Ts)){
 erh=BTErr[[1]][t,2]
 erh2=BTErr2[[1]][t,2]
 for(j in 2:3){
  erh=erh+approxExtrap(as.matrix(BTErr[[j]][,1]),as.matrix(BTErr[[j]][,2]),Ts[t])$y
  erh2=erh2+approxExtrap(as.matrix(BTErr2[[j]][,1]),as.matrix(BTErr2[[j]][,2]),Ts[t])$y
 }
 Overall[t,2]=erh/3
 Overall2[t,2]=erh2/3
}

 ############################################
 #          Visualization              #
 ############################################

 dev.new(width=5,height=5)
 par(mai=c(0.09,0.09,0.09,0.09),mfrow=c(3,2))
```

```
  NbPats=length(AverageDiffs[[1]][,1])
  for(j in 1:3){
    Betas=Results[[j]]$Betas
    Tis=Results[[j]]$Ts
    BtErr=BTErr[[j]]
    MBETAS=t(sapply(Betas, unlist))
    matplot(Tis,MBETAS,type="l")
    plot(BtErr)
  }

  dev.new(width=10,height=10)
  par(mai=c(0.6,0.6,0.6,0.6),mfrow=c(1,1))
  plot(Overall2)

 ############################################
 #          Interpretation             #
 ############################################
 Data=AverageDiffs[[j]]
 Data0=apply(Data[,2:{dim(Data)[2]}],2,function(xx){
   return((xx-mean(xx,na.rm=TRUE))/sd(xx,na.rm=TRUE))
 })
 Data[,2:dim(Data)[2]]=Data0
 Data=as.matrix(Data)
 Ts=Results[[1]]$Ts
 Estimates=matrix(0,150,1)
 T=3
 for(j in 1:3){
   Coef2=matrix(0,150,1)
   Betas=Results[[j]]$Betas
   Bet=as.matrix(Betas)
   Coef=t(sapply(Bet, unlist))
   for (c in 1:150){
     Coef2[c]=approxExtrap(as.matrix(Ts),as.matrix(Coef[,c]),T)$y}
   Estimates=Estimates+Coef2
 }
 Estimates=Estimates/3
 Error=Response-Data[,2:dim(Data)[2]]%*%Estimates
 plot(Error)
# selected variables
 Selected=attributes(Data[,2:{dim(Data)[2]}])[[2]][[2]][which(abs(Estimates)>1e-8)]
# Largest coefficient
 LargeCoef=which(abs(Estimates)==max(abs(Estimates)))
 LCoef=attributes(Data[,2:{dim(Data)[2]}])[[2]][[2]][LargeCoef]
Data2=Data[,2:{dim(Data)[2]}]
Data2[,LargeCoef]=Data[,LargeCoef+1]+1
change=(Data2-Data[,2:dim(Data)[2]])%*%Estimates
```