Head and Neck Cancer Progression-free Survival Prediction Based on Texture Data Using LASSO (with R code)

In this project, our objective is to build a predictive model for head and neck cancer progressive-free survival (PFS), which is also our respond of interest. Our predictors are textures of fractional intravascular blood volume at baseline measurement or follow–ups.

In this project, the major difficulty is how to select the right variables. The raw data from the cancer texture is very large and most of those predictors are not as significant to our interest. The LASSO (Least Absolute Shrinkage and Selection Operator) is a regression method that involves penalizing the absolute size of the regression coefficients. As a result, we decided to use LASSO to select the variables and developed our own lasso function using R.

Data Manipulation

To begin with, data manipulation is performed before any regression process:

(a) Remove completely missing columns.
(b) Remove rows with completely missing predictors.
(c) Remove columns with constant predictors.
(d) Remove (second or later occurrance of) columns which are collinear (correlation 1) with another.
(e) Calculate “baseline average” and “follow-up differences”

Table I shows the size of predictor matrix after each step of the data manipulation. The tolerance is chosen as 10^-4.

TABLE 1 Predictor Matrix Sizes

	Observation Number	Predictor Number
Raw Data	33	238
Step (a)	33	235
Step (b)	29	235
Step (c)	29	235
Step (d)	29	219
Step (e)	29	150

Multiple Imputation

Multiple imputation is a statistical technique for analyzing incomplete data sets, that is, data sets for which some entries are missing. As a result, multiple imputation allows us to perform complete dataset analyses (which many/most statistical and machine learning procedures require), and obtain corresponding estimates of interest on each complete dataset. In this project, three imputed datasets are created through the multiple imputation process using the R library mi.

Please see the following references for multiple imputation.

- Su, Yu-Sung, et al. "Multiple imputation with diagnostics (mi) in R: Opening windows into the black box." Journal of Statistical Software 45.2 (2011): 1-31;
- Rubin, Donald B. "Multiple imputation after 18+ years." Journal of the American Statistical Association 91.434 (1996): 473-489.

Lasso via Modification to Least Angle Regression

The lasso algorithm proposed by Efron is realized through R (see the Appendix).

- Efron, Bradley, et al. "Least angle regression." The Annals of statistics 32.2 (2004): 407-499.

In order to test my LASSO algorithm, I compared my lasso algorithm actions with the actions in the library lars in R. In TABLE II, "Action 1" is acquired from my algorithm, and "Action 2" is acquired from the library function lars. They are exactly the same, and both take 110 iterations to gain the final results.

TABLE II Action Comparison

Iter.	Action 1	Action 2	Iter.	Action 1	Action 2	Iter.	Action 1	Action 2	Iter.	Action 1	Action 2
1	90	90	31	130	130	61	64	64	91	63	63
2	127	127	32	132	132	62	-63	-63	92	-17	-17
3	96	96	33	-118	-118	63	58	58	93	-63	-63
4	55	55	34	99	99	64	62	62	94	-124	-124
5	86	86	35	131	131	65	124	124	95	-118	-118
6	105	105	36	-79	-79	66	63	63	96	124	124
7	84	84	37	116	116	67	-17	-17	97	50	50
8	-86	-86	38	118	118	68	-105	-105	98	-124	-124
9	56	56	39	79	79	69	78	78	99	16	16
10	146	146	40	-133	-133	70	105	105	100	48	48
11	-84	-84	41	145	145	71	-94	-94	101	3	3
12	75	75	42	96	96	72	-67	-67	102	-15	-15
13	84	84	43	-84	-84	73	-105	-105	103	142	142
14	94	94	44	41	41	74	118	118	104	-44	-44
15	-96	-96	45	-79	-79	75	147	147	105	15	15
16	-56	-56	46	49	49	76	120	120	106	104	104
17	99	99	47	67	67	77	-124	-124	107	-16	-16
18	78	78	48	-145	-145	78	-147	-147	108	118	118
19	-99	-99	49	-41	-41	79	-118	-118	109	-3	-3
20	79	79	50	-78	-78	80	17	17	110	145	145
21	69	69	51	44	44	81	15	15	111		
22	133	133	52	-118	-118	82	-131	-131	112		
23	51	51	53	-55	-55	83	-99	-99	113		
24	54	54	54	118	118	84	144	144	114		
25	62	62	55	76	76	85	117	117	115		
26	137	137	56	-118	-118	86	-127	-127	116		
27	118	118	57	62	62	87	-63	-63	117		
28	126	126	58	17	17	88	124	124	118		
29	-62	-62	59	63	63	89	99	99	119		
30	-137	-137	60	-62	-62	90	118	118			

I also compared the β from my algorithm (β_1) with the β from the function lars in the library (β_2), see TABLE III. They are exactly the same.

TABLE III Action Comparison

Predictor ID	β_1	β_2	Predictor ID	β_1	β_2
1	0.1399669	0.1399669	15	0.3503249	0.3503249
2	-0.15348659	-0.15348659	16	-0.05922694	-0.05922694
3	-1.47835984	-1.47835984	17	-0.03209639	-0.03209639
4	0.25446971	0.25446971	18	0.40883653	0.40883653
5	-0.14958195	-0.14958195	19	0.12781619	0.12781619
6	0.95110139	0.95110139	20	0.03902563	0.03902563
7	1.11528411	1.11528411	21	-0.48478189	-0.48478189
8	-1.32583124	-1.32583124	22	0.3573575	0.3573575
9	0.67937989	0.67937989	23	0.16942768	0.16942768
10	0.2776513	0.2776513	24	0.71598979	0.71598979
11	0.61291334	0.61291334	25	-0.1635122	-0.1635122
12	-0.24589955	-0.24589955	26	0.16354518	0.16354518
13	-0.13537225	-0.13537225	27	-0.0033004	-0.0033004
14	-0.09928269	-0.09928269	28	0.2182681	0.2182681

- Figure 1 shows the sum of absolute beta value (t) vs. iteration
- Figure 2 shows my lasso path
- Figure 3 shows the lasso path from library lars
- Figure 4 shows $\overline{err}$ (the apparent mean absolute predictive error rate)
- Figure 5 γ (the no information error rate)

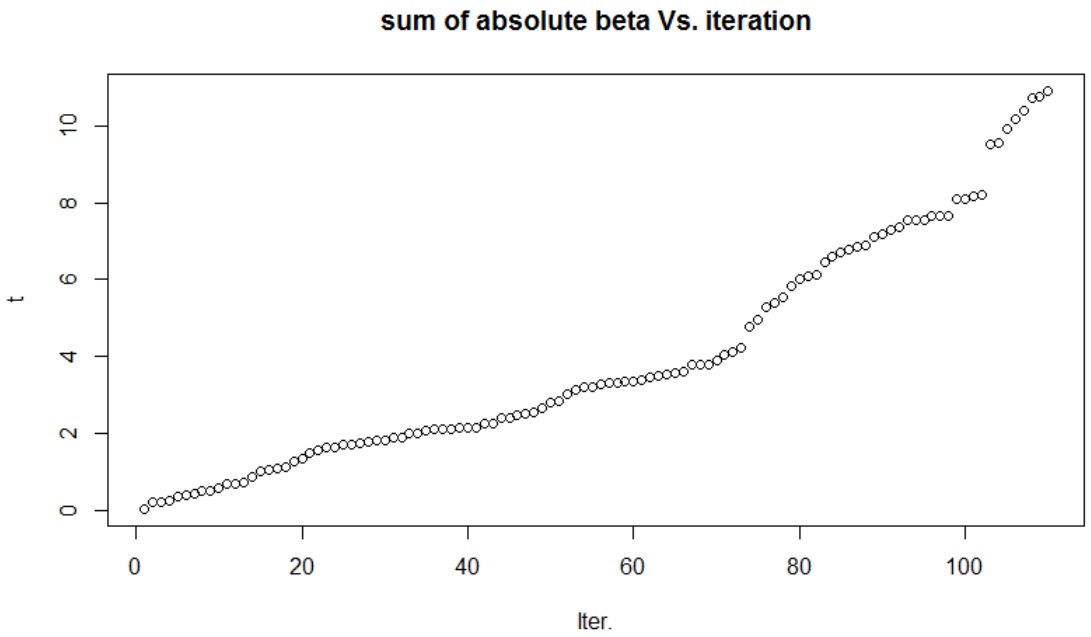

Figure 1 sum of absolute beta Vs. iteration

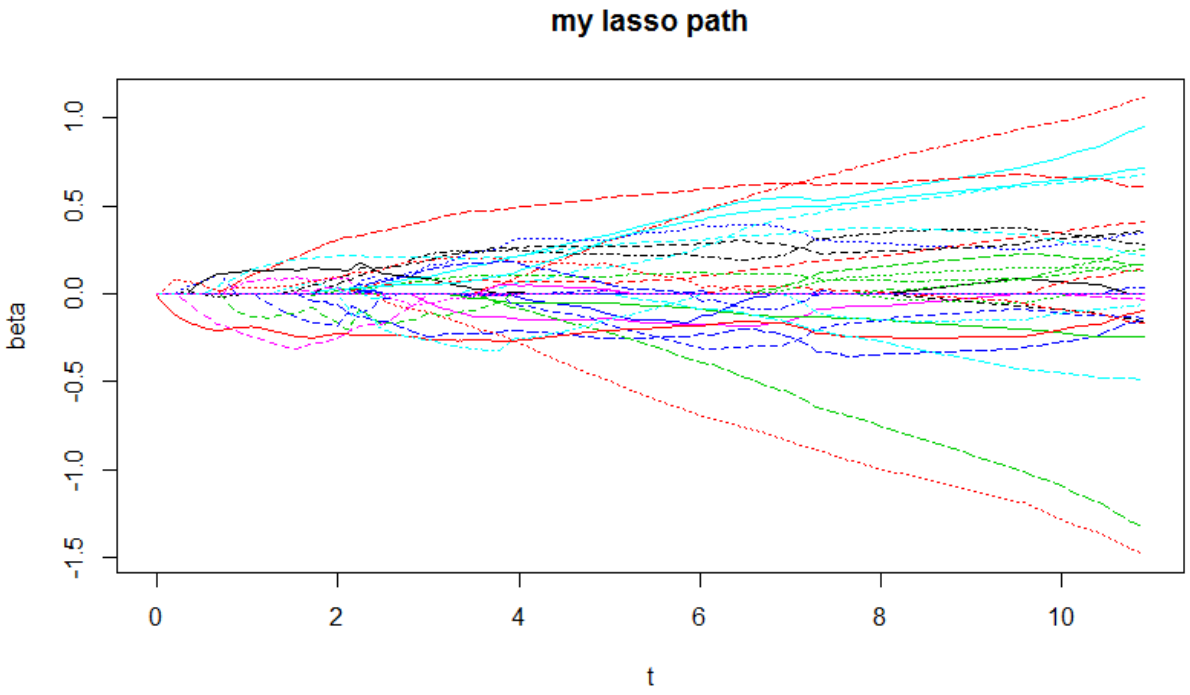

Figure 2 my lasso path

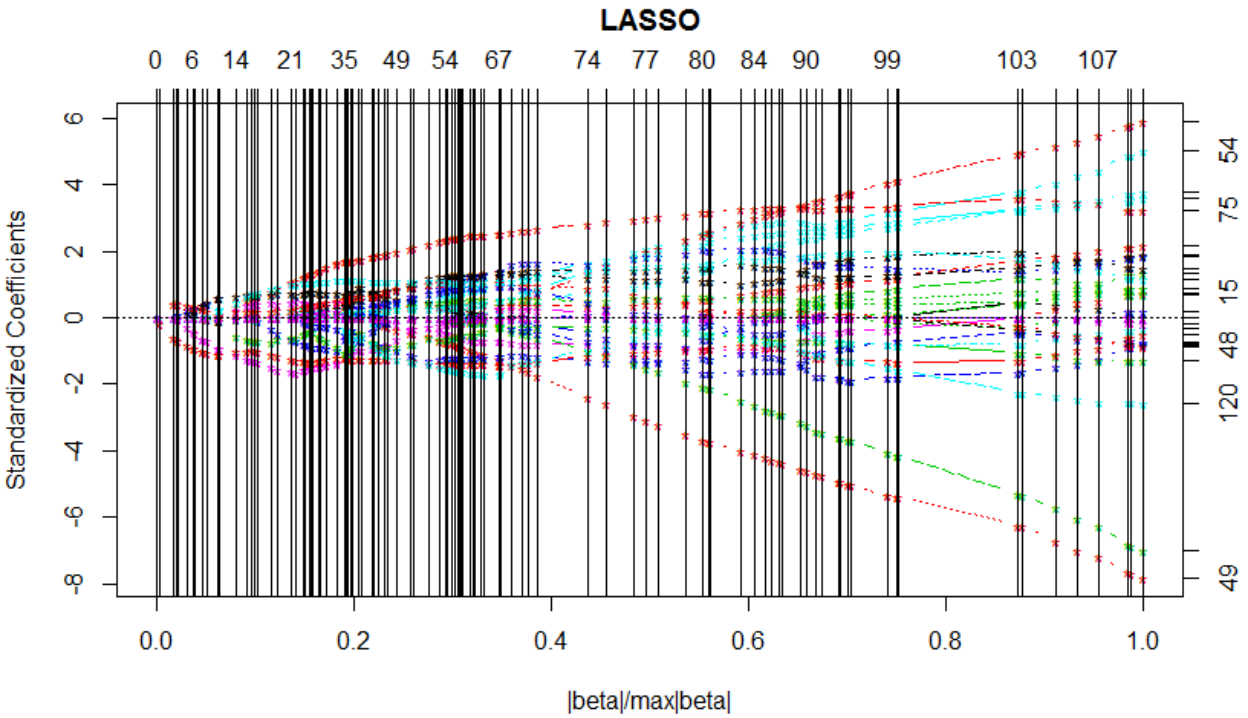

Figure 3 lasso path from lars

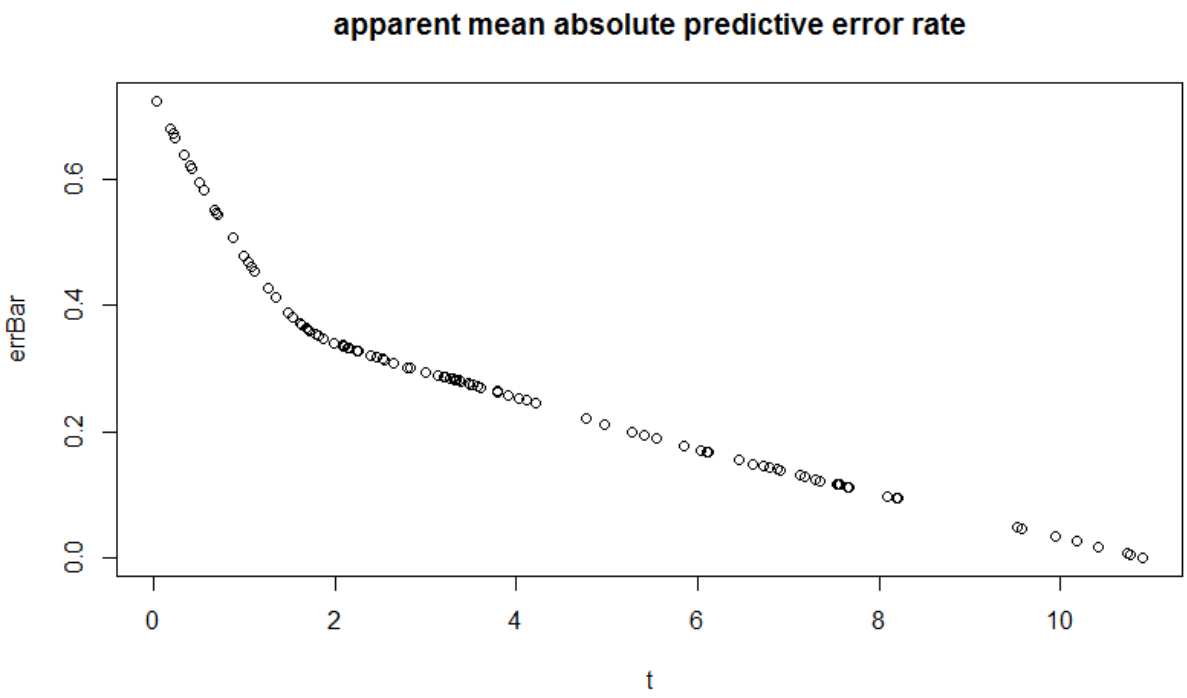

Figure 4 apparent mean absolute predictive error

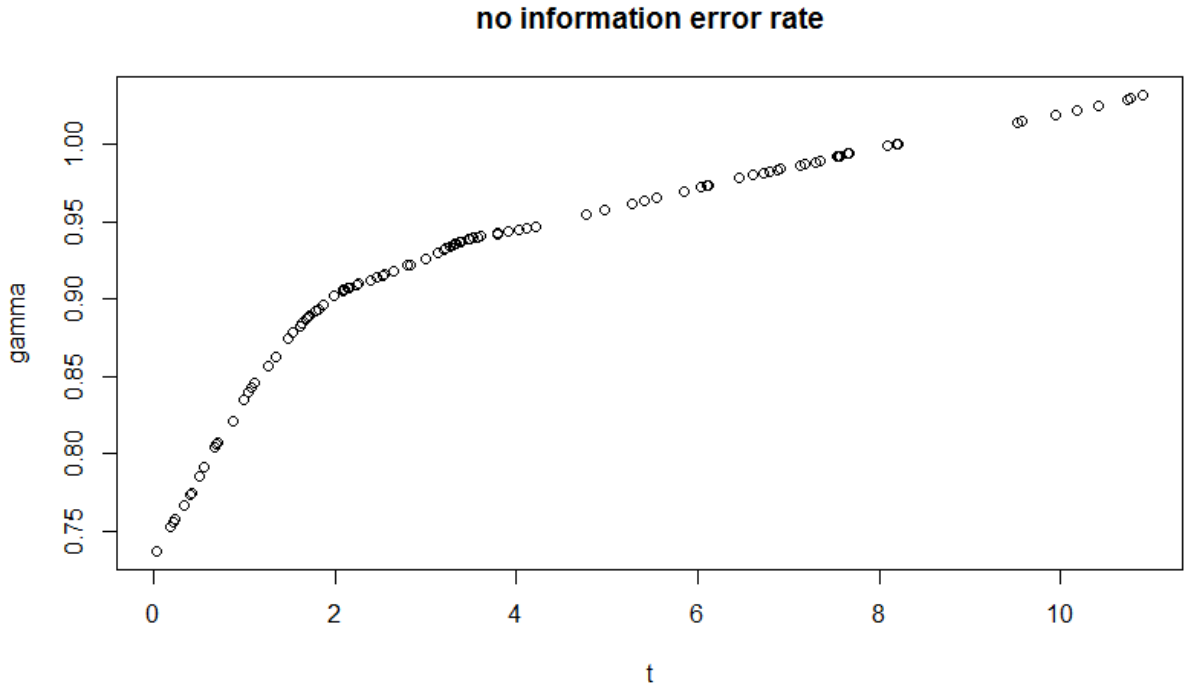

Figure 5 no information error

Lasso via Modification to Least Angle Regression

(a) Calculate the $\widehat{Err}^{(0.632+)}$ using $\overline{err}$, $\widehat{Err}^{(1)}$ and $\hat{\gamma}$.

$$\widehat{Err}^{(0.632+)} = (1 - \hat{w}) \times \overline{err} + \hat{w} \times \widehat{Err}^{(1)}$$

where

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

$$Q\left(y_i, \hat{\mu}^b(x_i)\right) = \left|(y_i - \overline{y^b}) - ((\boldsymbol{x}_i - \overline{\boldsymbol{x}^b})/sd(\boldsymbol{x}^b)) \times \boldsymbol{\beta}^b\right|$$

where $\overline{\boldsymbol{x}^b}$ and $sd(\boldsymbol{x}^b)$ is mean and standard deviation of $\boldsymbol{x}$ in the bootstrap b, and $\overline{y^b}$ is the mean of $\boldsymbol{y}$ in the bootstrap b.

(2) The average $\widehat{Err}^{(0.632+)}$ across three multiple imputation datasets is calculated, see Figure 6.

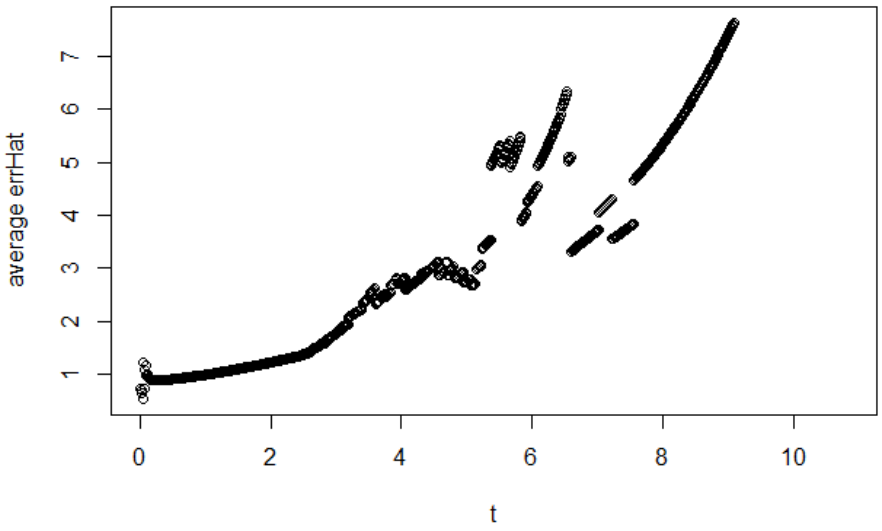

Figure 6 the average errHat

(3) Choose the best t value.

The best t is chosen as the one that minimize the average $\widehat{Err}^{(0.632+)}$. In this example, the best t value is chosen around 0.45, which corresponds to choosing around 7 predictors (depends on the MI dataset).

Problem 5 Visualization

(a) The lasso path and $\widehat{Err}^{(0.632+)}$ are calculated and plotted as the following figures:

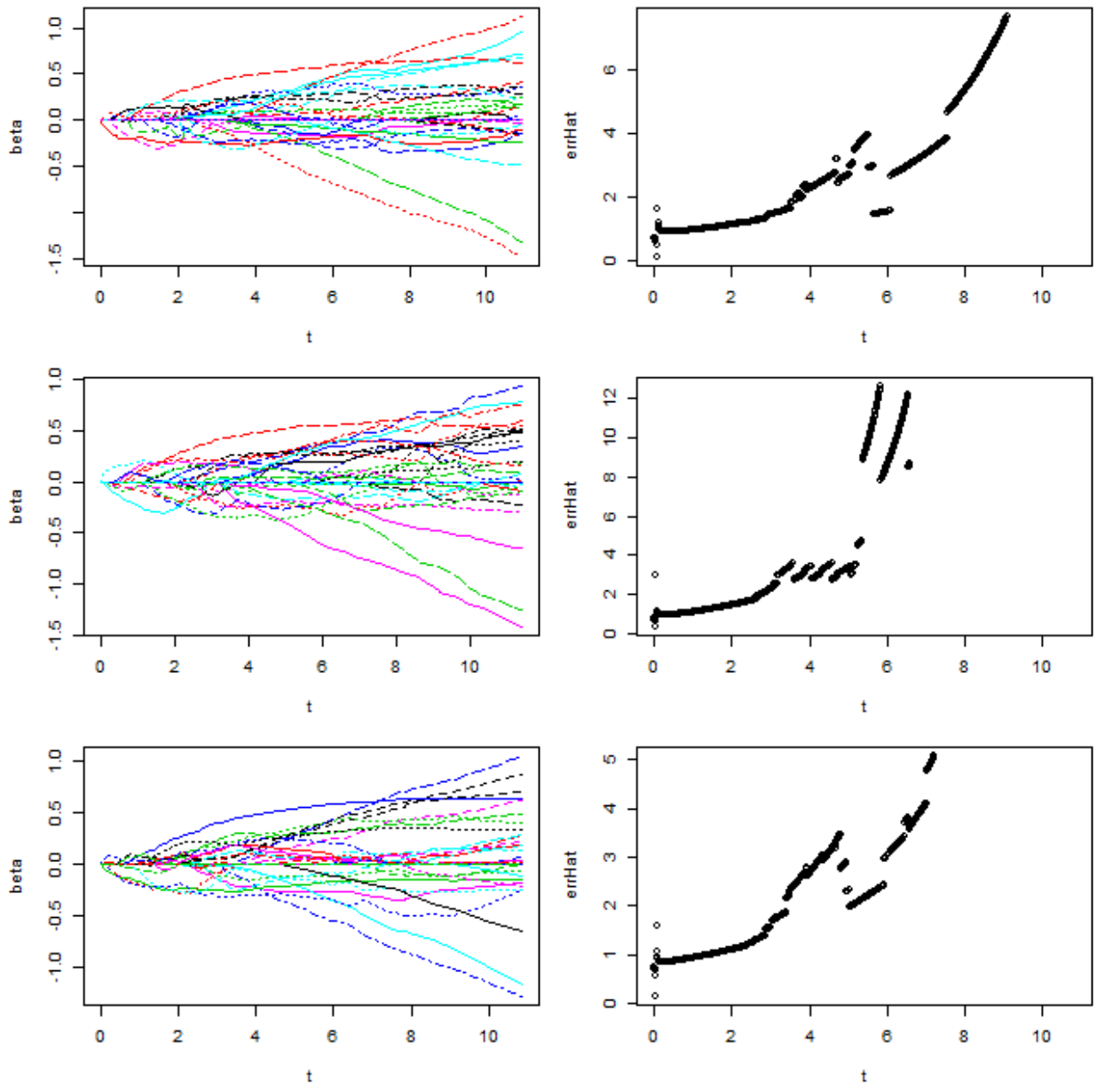

Figure 7 Lasso path and errHat from three MI datasets

(b) The average $\widehat{Err}^{(0.632+)}$ across three multiple imputation datasets is plotted in **Figure 6**.

Interpretation

In this example, we choose t = 0.45. The final model coefficients are calculated by averaging β from three multiple imputation datasets (β_1, β_2 and β_3). In each multiple imputation model i, β_i is interpolated so that the t value is 0.45.

The final model is listed as follows:

Predictor ID	Beta	Predictor Name
55	-0.125843705	v1_Texture.Strength_average
56	-0.022846644	v1_Maximum.Intensity_average
75	0.01346343	v1_High.Intensity.Large.Zone.Emphasis_average
84	0.00146977	v1_Sum.of.Squares.or.Variance_difference
86	0.008499992	v1_Sum.Variance_difference
88	0.060127267	v1_Difference.Variance_difference
90	-0.098805288	v1_Information.Measure.of.Correlation1_difference
96	0.018049183	v1_Contrast.1_difference
105	0.046300367	v1_Sum.of.Squares.or.Variance.1_difference
110	-0.023923559	v1_Information.Measure.of.Correlation1.1_difference
114	-0.062074634	v1_Inverse.Difference.Moment.Normalized.1_difference
127	0.03842584	v1_Contrast.2_difference
148	0.029746621	v1_High.Intensity.Short.Zone.Emphasis_difference

However, the above beta coefficients are based on several normalization processes. To begin with, we standardized the raw data. For example, the mean value of y (PFS) is 4.836. Then we calculate the baseline average and follow up difference as our model predictors. Again, we normalized the data before we could use lasso. Now, based on our normalized data, we ranked the importance of the predictors as follows:

Beta	Predictor Name
-0.1258	v1_Texture.Strength_average
-0.0988	v1_Information.Measure.of.Correlation1_difference
-0.0621	v1_Inverse.Difference.Moment.Normalized.1_difference
0.06013	v1_Difference.Variance_difference
0.0463	v1_Sum.of.Squares.or.Variance.1_difference
0.03843	v1_Contrast.2_difference
0.02975	v1_High.Intensity.Short.Zone.Emphasis_difference
-0.0239	v1_Information.Measure.of.Correlation1.1_difference
-0.0228	v1_Maximum.Intensity_average
0.01805	v1_Contrast.1_difference
0.01346	v1_High.Intensity.Large.Zone.Emphasis_average
0.0085	v1_Sum.Variance_difference
0.00147	v1_Sum.of.Squares.or.Variance_difference

If the coefficient is positive, the increase of the corresponding predictor will increase the PFS, vice versa. And the large the coefficient is the more correlated the predictor is with the PFS. For example, the predictor “v1_Texture.Strength_average” is negative. The increase of 1 unit of the predictor will lead the PFS to decrease 0.1258. Please note that the 1 unit of “v1_Texture.Strength_average” here is 1 unit after the normalization. To go back to the original value, we need to reverse our normalization process.

Appendix I Data Manipulation and Multiple Imputation Example

```
library(mi)
library(lars)

# 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
}
# normalize the predictors
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)
 }
})
```

```
# normalize the PFS mean = zero
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
##################################################

# Remove completely missing columns.
temp1 = logical(length = dim(myData)[2])
for(ii in 1:dim(myData)[2]){
 if(sum(abs(myData[,ii])>matrix(0.0001,dim(myData)[1],1), na.rm = TRUE)>0){
   temp1[ii] = TRUE
 }else{
   temp1[ii] = FALSE # missing columns
 }
}
myData1=myData[,temp1]

# Remove rows with completely missing predictors.
temp = logical(length = dim(myData1)[1])
for(ii in 1:dim(myData1)[1]){
 if(sum(abs(myData1[ii,2:dim(myData1)[2]])>matrix(0.0001,1,dim(myData1)[2]), na.rm = TRUE)>0){
   temp[ii] = TRUE
 }else{
   temp[ii] = FALSE # missing columns
 }
}
myData2=myData1[temp,]

# Remove columns with constant predictors.
temp2 = logical(length = dim(myData2)[2])
tempData = myData2
tempData[is.na(tempData)] = 0
for(ii in 1:dim(myData2)[2]){
 if(abs(abs(max(tempData[,ii]))+abs(min(tempData[,ii])))>0.0001){
   temp2[ii] = TRUE
 }else{
   temp2[ii] = FALSE # missing columns
 }
}
myData3=myData2[,temp2]

# Remove (second or later occurrance of) columns which are collinear (correlation 1) with another.
varmatrix = var(myData3[,-1], use ="pairwise.complete.obs")
```

```
temp = matrix(data = FALSE, nrow = dim(myData3)[2]-1, ncol = dim(myData3)[2]-1)
for(ii in 1:(dim(myData3)[2]-1)){
 for(jj in ii:(dim(myData3)[2]-1)){
  if (varmatrix[ii,jj]>0.9999&ii!=jj){
   temp[ii,jj] = TRUE
  }
 }
}
temp3 = colSums(temp)
temp3 = temp3==0
# sum(temp2)
temp3 = append(temp3,1,after = 0)
myData4 = myData3[,as.logical(temp3)]


############################################ MI Algorithm
#####################################################
myData = myData4
myInfo=mi.info(myData)
myInfo=update(myInfo,"type",as.list(rep("predictive-mean-matching",dim(myData)[2])))
myMIs=mi(myData,myInfo)
misStar=mi.completed(myMIs)

#DeletedNames = myNames[!temp1]
#temp = myNames[temp1]
#DeletedNames = rbind(as.matrix(DeletedNames),as.matrix(temp[as.logical(!temp3)]))

sampleNames = attributes(myXData)$names[6:dim(myXData)[2]]
temp = matrix(NA,dim(myData4)[1],dim(as.matrix(sampleNames))[1])
tempNames=paste(sampleNames,"_average",sep="")
attributes(temp)$names=tempNames
averageData = list(temp,temp,temp)
tempNames=paste(sampleNames,"_difference",sep="")
attributes(temp)$names=tempNames
differenceData = list(temp,temp,temp)

# compute the average and difference data
for (kk in 1:3){
 baseLine1Data = matrix(NA,dim(misStar[[1]])[1],dim(as.matrix(sampleNames))[1])
 baseLine2Data = matrix(NA,dim(misStar[[1]])[1],dim(as.matrix(sampleNames))[1])
 followUpData = matrix(NA,dim(misStar[[1]])[1],dim(as.matrix(sampleNames))[1])
 for(ii in 1:dim(as.matrix(sampleNames))[1]){
  if (length(grep(pattern = paste(sampleNames[ii],"_baseline1",sep=""),attributes(misStar[[1]])$names))!=0){
   baseLine1Data[,ii]=misStar[[kk]][,grep(pattern =
paste(sampleNames[ii],"_baseline1",sep=""),attributes(misStar[[1]])$names)]
  }
  if (length(grep(pattern = paste(sampleNames[ii],"_baseline2",sep=""),attributes(misStar[[1]])$names))!=0){
   baseLine2Data[,ii]=misStar[[kk]][,grep(pattern =
paste(sampleNames[ii],"_baseline2",sep=""),attributes(misStar[[1]])$names)]
  }
  if (length(grep(pattern = paste(sampleNames[ii],"_followUp",sep=""),attributes(misStar[[1]])$names))!=0){
```

```
    followUpData[,ii]=misStar[[kk]][,grep(pattern = paste(sampleNames[ii],"_followUp",sep=""),attributes(misStar[[1]])$names)]
   }
  }
  # compute the average value
  for(ii in 1:dim(as.matrix(sampleNames))[1]){
   averageData[[kk]][,ii] = as.matrix(rowMeans(cbind(baseLine1Data[,ii],baseLine2Data[,ii]),na.rm = TRUE))
  }
  # compute the difference
  differenceData[[kk]] = followUpData-averageData[[kk]]
 }
```

Appendix II Lars Example

```
# Initualization
k = 0
A = matrix(0,dim(myX)[2],dim(myX)[2])
Ac = matrix(1,dim(myX)[2],dim(myX)[2])
beta = matrix(0,dim(myX)[2],dim(myX)[2])
XA = matrix(0,dim(myX)[1],k)
cBar = matrix(0,dim(myX)[2],dim(myX)[2])
u = matrix(0,dim(myX)[1],dim(myX)[2]+1)
betaA = matrix(0,dim(myX)[1],k)
#(min(dim(myX)[1]-1,dim(myX)[2])-1)
for(k in 1:(min(dim(myX)[1]-1,dim(myX)[2]-1))){
 # update cBar (for k-1)
 cBar[k,] = t(myX)%*%(myY-u[,k])
 # find cBar for Ac elements
 cBarc = cBar[k,]
 cBarc[as.logical(A[k,])]=0
 # update XA
 cat(which.max(abs(cBarc)),sep=" ",fill=TRUE)
 A[k:dim(myX)[2],which.max(abs(cBarc))] = 1
 Ac[k:dim(myX)[2],which.max(abs(cBarc))] = 0
 s = sign(cBar[k,as.logical(A[k,])])
 XA = myX[,as.logical(A[k,])]
 XA = XA%*%diag(as.vector(s),length(s))
 # update cBar
 CBar = mean(abs(t(XA)%*%(myY-u[,k])))
 #  cat(abs(t(XA)%*%(myY-u[,k])),sep=" ",fill=TRUE)
 # update angles
 angles =
t(myX)%*%XA%*%solve(t(XA)%*%XA,matrix(1,sum(A[k,]),
1))
 temp = matrix(NA,dim(myX)[2],3)
 temp[,1] = (CBar-cBar[k,])/(1-angles)
 temp[,2] = (CBar+cBar[k,])/(1+angles)
 temp[as.logical(temp[,1]<0),1] = +Inf
 temp[as.logical(temp[,2]<0),2] = +Inf
 temp[,3] = pmin(temp[,1],temp[,2])
 temp[as.logical(A[k,]),3] = +Inf
 c = min(temp[,3])
 # update u
 u[,k+1] =
c*XA%*%solve(t(XA)%*%XA)%*%matrix(1,sum(A[k,]),1)+u[
,k]
 #update betaA
 b = c*solve(t(XA)%*%XA)%*%matrix(1,sum(A[k,]),1)
 j = sum(A[k,1:which.max(abs(cBarc))])
 betaA = betaA+b[-j]
 betaA = append(betaA,b[j],after=j-1)
 beta[k,as.logical(A[k,])] = betaA
}

# ordinary least square solution
s = sign(cBar[k,as.logical(A[k,])])
betaBar = solve(t(XA)%*%XA)%*%t(XA)%*%(myY)
betaBarSign = s*betaBar

# caned function
object = lars(myX, myY, type = c("lar"))
# caned function dim(beta)[1]=dim(X)[1]+1
caned_beta = unique(object$beta[29,])[-1]
diff = cbind(betaBarSign,as.matrix(caned_beta))
cbind(unique(s*beta[k,as.logical(A[k,])]),caned_beta,betaB
arSign)
```

Appendix III Lasso Example

```
################################### Lasso ############################################

# Initualization
k = 0
A = matrix(0,dim(myX)[2],dim(myX)[2])
Ac = matrix(1,dim(myX)[2],dim(myX)[2])
beta = matrix(0,dim(myX)[2],dim(myX)[2])
XA = matrix(0,dim(myX)[1],k)
cBar = matrix(0,dim(myX)[2],dim(myX)[2])
u = matrix(0,dim(myX)[1],dim(myX)[2]+1)
betaA = matrix(0,dim(myX)[1],k)
jj = 0
action = matrix(0,dim(myX)[2],1)
numErr = 10^-12

S = matrix(0,dim(myX)[2],dim(myX)[2]) # record the sign
# run lasso
for(k in 1:dim(myX)[2]){
 # if jj needs to leave the active variable set XA
 if(jj!=0){
  cat(-jj,sep=" ",fill=TRUE)
  action[k,] = -jj
  # exclude jj from the active variable set XA
  A[k:dim(myX)[2],jj] = 0
  Ac[k:dim(myX)[2],jj] = 1
  cBar[k,] = t(myX)%*%(myY-u[,k]) # which one to use u[,k] or u[,k-1]
  s = sign(cBar[k,as.logical(A[k,])])
  XA = myX[,as.logical(A[k,])]
  XA = XA%*%diag(as.vector(s),length(s))
  # update cBar
  CBar = mean(abs(t(XA)%*%(myY-u[,k])))
  #  cat(abs(t(XA)%*%(myY-u[,k])),sep=" ",fill=TRUE)
  # update angles
  angles = t(myX)%*%XA%*%solve(t(XA)%*%XA,matrix(1,sum(A[k,]),1))
  temp = matrix(NA,dim(myX)[2],3)
  temp[,1] = (CBar-cBar[k,])/(1-angles)
  temp[,2] = (CBar+cBar[k,])/(1+angles)
  temp[as.logical(temp[,1]<0),1] = +Inf
  temp[as.logical(temp[,2]<0),2] = +Inf
  temp[,3] = pmin(temp[,1],temp[,2])
  temp[as.logical(A[k,]),3] = +Inf
  c = min(temp[temp[,3]>numErr,3])
  # Lasso Modification
  if(k>1){
   bBar = solve(t(XA)%*%XA)%*%matrix(1,sum(A[k,]),1)
   d = matrix(0,dim(myX)[2],1)
   # d[as.logical(A[k,]),1]=s*bBar
   d[as.logical(A[k,]),1]=bBar
   betaOld = matrix(0,dim(myX)[2],1)
```

```
    # fix betaA delete the zero elements
    betaA = betaA[-sum(A[k-1,1:jj])]
    betaOld[as.logical(A[k,]),1]=betaA
    cLasso = matrix(0,dim(myX)[2],1)
    cLasso[as.logical(A[k,]),1] = -betaOld[as.logical(A[k,]),1]/d[as.logical(A[k,]),1]
    if(sum(cLasso>0)==0){
      cStar = Inf
    }else{
      cStar = min(cLasso[cLasso>0])
    }
    if(cStar<c){
      c = cStar
      jj = which(cLasso==cStar) # the predictor that should leave the active variable set
    }else{
      jj = 0
    }
  }
  # update u
  u[,k+1] = c*XA%*%solve(t(XA)%*%XA)%*%matrix(1,sum(A[k,]),1)+u[,k]
  #update betaA
  b = c*solve(t(XA)%*%XA)%*%matrix(1,sum(A[k,]),1)
  betaA = betaA+b
  beta[k,as.logical(A[k,])]=betaA
 }else{
  # update cBar (for k-1)
  cBar[k,] = t(myX)%*%(myY-u[,k])
  # find cBar for Ac elements: cBarc
  cBarc = cBar[k,]
  cBarc[as.logical(A[k,])]=0
  # update XA
  cat(which.max(abs(cBarc)),sep=" ",fill=TRUE)
  action[k,] = which.max(abs(cBarc))
  A[k:dim(myX)[2],which.max(abs(cBarc))] = 1
  Ac[k:dim(myX)[2],which.max(abs(cBarc))] = 0
  s = sign(cBar[k,as.logical(A[k,])])
  XA = myX[,as.logical(A[k,])]
  XA = XA%*%diag(as.vector(s),length(s))
  # update cBar
  CBar = mean(abs(t(XA)%*%(myY-u[,k])))
  #  cat(abs(t(XA)%*%(myY-u[,k])),sep=" ",fill=TRUE)
  # update angles
  angles = t(myX)%*%XA%*%solve(t(XA)%*%XA,matrix(1,sum(A[k,]),1))
  temp = matrix(NA,dim(myX)[2],3)
  temp[,1] = (CBar-cBar[k,])/(1-angles)
  temp[,2] = (CBar+cBar[k,])/(1+angles)
  temp[as.logical(temp[,1]<0),1] = +Inf
  temp[as.logical(temp[,2]<0),2] = +Inf
  temp[,3] = pmin(temp[,1],temp[,2])
  temp[as.logical(A[k,]),3] = +Inf
  c = min(temp[temp[,3]>numErr,3])
  # Lasso Modification
```

```
  if(k>1){
    bBar = solve(t(XA)%*%XA)%*%matrix(1,sum(A[k,]),1)
    d = matrix(0,dim(myX)[2],1)
    # d[as.logical(A[k,]),1]=s*bBar
    d[as.logical(A[k,]),1]=bBar
    betaOld = matrix(0,dim(myX)[2],1)
    betaOld[as.logical(A[k-1,]),1]=betaA

    ## something is wrong here
    cLasso = matrix(0,dim(myX)[2],1)
    cLasso[as.logical(A[k,]),1] = -betaOld[as.logical(A[k,]),1]/d[as.logical(A[k,]),1]
    if(sum(cLasso>0)==0){
      cStar = Inf
    }else{
      cStar = min(cLasso[cLasso>0])
    }
    if(cStar<c){
      c = cStar
      jj = which(cLasso==cStar) # the predictor that should leave the active variable set
    }else{
      jj = 0
    }
  }
  # update u
  u[,k+1] = c*XA%*%solve(t(XA)%*%XA)%*%matrix(1,sum(A[k,]),1)+u[,k]
  #update betaA
  b = c*solve(t(XA)%*%XA)%*%matrix(1,sum(A[k,]),1)
  j = sum(A[k,1:which.max(abs(cBarc))])
  betaA = betaA+b[-j]
  betaA = append(betaA,b[j],after=j-1)
  beta[k,as.logical(A[k,])]=betaA
 }
 # collect AMAPER and t and NIER
 S[k,] = sign(cBar[k,])
 # stop criteria
 if (dim(XA)[2]==(min(dim(myX)[1]-1,dim(myX)[2]-1)) & jj==0){
  break
 }
}
```

Appendix IV Bootstrap Example

```
KK = 1
myX = cbind(averageData[[KK]],differenceData[[KK]])
myXNames = rbind(as.matrix(paste(sampleNames,"_average",sep="")),as.matrix(paste(sampleNames,"_difference",sep="")))
######  attributes(myX)$names=myXNames
myX = myX[,colSums(is.na(myX))==0]
myPFS = misStar[[KK]][,1]
# check colinearity
# myXInfo=mi.info(myX)
# normalize the predictors
myX=apply(myX,2,function(xx){
  return((xx-mean(xx,na.rm=TRUE))/sd(xx,na.rm=TRUE))
})
# normalize the PFS mean = zero
myY=myPFS-mean(myPFS)



############## run lasso for the full data to get errBar and gamma ###############
# Initualization
k = 0
A = matrix(0,dim(myX)[2],dim(myX)[2])
Ac = matrix(1,dim(myX)[2],dim(myX)[2])
beta = matrix(0,dim(myX)[2],dim(myX)[2])
XA = matrix(0,dim(myX)[1],k)
cBar = matrix(0,dim(myX)[2],dim(myX)[2])
u = matrix(0,dim(myX)[1],dim(myX)[2]+1)
betaA = matrix(0,dim(myX)[1],k)
jj = 0
action = matrix(0,dim(myX)[2],1)
numErr = 10^-12

S = matrix(0,dim(myX)[2],dim(myX)[2]) # record the sign
# run lasso
for(k in 1:dim(myX)[2]){
  # if jj needs to leave the active variable set XA
  if(jj!=0){
    cat(-jj,sep=" ",fill=TRUE)
    action[k,] = -jj
    # exclude jj from the active variable set XA
    A[k:dim(myX)[2],jj] = 0
    Ac[k:dim(myX)[2],jj] = 1
    cBar[k,] = t(myX)%*%(myY-u[,k]) # which one to use u[,k] or u[,k-1]
    s = sign(cBar[k,as.logical(A[k,])])
    XA = myX[,as.logical(A[k,])]
    XA = XA%*%diag(as.vector(s),length(s))
    # update cBar
    CBar = mean(abs(t(XA)%*%(myY-u[,k])))
    #  cat(abs(t(XA)%*%(myY-u[,k])),sep=" ",fill=TRUE)
    # update angles
    angles = t(myX)%*%XA%*%solve(t(XA)%*%XA,matrix(1,sum(A[k,]),1))
```

```
  temp = matrix(NA,dim(myX)[2],3)
  temp[,1] = (CBar-cBar[k,])/(1-angles)
  temp[,2] = (CBar+cBar[k,])/(1+angles)
  temp[as.logical(temp[,1]<0),1] = +Inf
  temp[as.logical(temp[,2]<0),2] = +Inf
  temp[,3] = pmin(temp[,1],temp[,2])
  temp[as.logical(A[k,]),3] = +Inf
  c = min(temp[temp[,3]>numErr,3])
  # Lasso Modification
  if(k>1){
    bBar = solve(t(XA)%*%XA)%*%matrix(1,sum(A[k,]),1)
    d = matrix(0,dim(myX)[2],1)
    # d[as.logical(A[k,]),1]=s*bBar
    d[as.logical(A[k,]),1]=bBar
    betaOld = matrix(0,dim(myX)[2],1)
    # fix betaA delete the zero elements
    betaA = betaA[-sum(A[k-1,1:jj])]
    betaOld[as.logical(A[k,]),1]=betaA
    cLasso = matrix(0,dim(myX)[2],1)
    cLasso[as.logical(A[k,]),1] = -betaOld[as.logical(A[k,]),1]/d[as.logical(A[k,]),1]
    if(sum(cLasso>0)==0){
      cStar = Inf
    }else{
      cStar = min(cLasso[cLasso>0])
    }
    if(cStar<c){
      c = cStar
      jj = which(cLasso==cStar) # the predictor that should leave the active variable set
    }else{
      jj = 0
    }
  }
  # update u
  u[,k+1] = c*XA%*%solve(t(XA)%*%XA)%*%matrix(1,sum(A[k,]),1)+u[,k]
  #update betaA
  b = c*solve(t(XA)%*%XA)%*%matrix(1,sum(A[k,]),1)
  betaA = betaA+b
  beta[k,as.logical(A[k,])]=betaA
 }else{
  # update cBar (for k-1)
  cBar[k,] = t(myX)%*%(myY-u[,k])
  # find cBar for Ac elements: cBarc
  cBarc = cBar[k,]
  cBarc[as.logical(A[k,])]=0
  # update XA
  cat(which.max(abs(cBarc)),sep=" ",fill=TRUE)
  action[k,] = which.max(abs(cBarc))
  A[k:dim(myX)[2],which.max(abs(cBarc))] = 1
  Ac[k:dim(myX)[2],which.max(abs(cBarc))] = 0
  s = sign(cBar[k,as.logical(A[k,])])
  XA = myX[,as.logical(A[k,])]
```

```
 XA = XA%*%diag(as.vector(s),length(s))
 # update cBar
 CBar = mean(abs(t(XA)%*%(myY-u[,k])))
 #  cat(abs(t(XA)%*%(myY-u[,k])),sep=" ",fill=TRUE)
 # update angles
 angles = t(myX)%*%XA%*%solve(t(XA)%*%XA,matrix(1,sum(A[k,]),1))
 temp = matrix(NA,dim(myX)[2],3)
 temp[,1] = (CBar-cBar[k,])/(1-angles)
 temp[,2] = (CBar+cBar[k,])/(1+angles)
 temp[as.logical(temp[,1]<0),1] = +Inf
 temp[as.logical(temp[,2]<0),2] = +Inf
 temp[,3] = pmin(temp[,1],temp[,2])
 temp[as.logical(A[k,]),3] = +Inf
 c = min(temp[temp[,3]>numErr,3])
 # Lasso Modification
 if(k>1){
   bBar = solve(t(XA)%*%XA)%*%matrix(1,sum(A[k,]),1)
   d = matrix(0,dim(myX)[2],1)
   # d[as.logical(A[k,]),1]=s*bBar
   d[as.logical(A[k,]),1]=bBar
   betaOld = matrix(0,dim(myX)[2],1)
   betaOld[as.logical(A[k-1,]),1]=betaA

   ## something is wrong here
   cLasso = matrix(0,dim(myX)[2],1)
   cLasso[as.logical(A[k,]),1] = -betaOld[as.logical(A[k,]),1]/d[as.logical(A[k,]),1]
   if(sum(cLasso>0)==0){
     cStar = Inf
   }else{
     cStar = min(cLasso[cLasso>0])
   }
   if(cStar<c){
     c = cStar
     jj = which(cLasso==cStar) # the predictor that should leave the active variable set
   }else{
     jj = 0
   }
 }
 # update u
 u[,k+1] = c*XA%*%solve(t(XA)%*%XA)%*%matrix(1,sum(A[k,]),1)+u[,k]
 #update betaA
 b = c*solve(t(XA)%*%XA)%*%matrix(1,sum(A[k,]),1)
 j = sum(A[k,1:which.max(abs(cBarc))])
 betaA = betaA+b[-j]
 betaA = append(betaA,b[j],after=j-1)
 beta[k,as.logical(A[k,])]=betaA
}
# collect AMAPER and t and NIER
S[k,] = sign(cBar[k,])
# stop criteria
if (dim(XA)[2]==(min(dim(myX)[1]-1,dim(myX)[2]-1)) & jj==0){
```

```
   break
  }
}
# plot
errBar = matrix(NA,dim(myX)[2],1) # apparent mean absolute predictive error rate
gamma = matrix(NA,dim(myX)[2],1) # no information error rate
tAll = matrix(NA,dim(myX)[2],1)
for(kk in 1:k){
  tAll[kk,1] = sum(abs(beta[kk,]))
  XA = myX[,as.logical(A[kk,])]
  XA = XA%*%diag(as.vector(S[kk,as.logical(A[kk,])]),length(S[kk,as.logical(A[kk,])]))
  errBar[kk,1] = sum(abs(myY-XA%*%beta[kk,as.logical(A[kk,])]))/(dim(myX)[1])
  gamma[kk,1] = sum(abs(matrix(1,dim(myX)[1],1)%*%myY-
XA%*%as.matrix(beta[kk,as.logical(A[kk,])])%*%matrix(1,1,dim(myX)[1])))/(dim(myX)[1])^2
}

plot(tAll[1:k],xlab = "Iter.",ylab = "t")
title(main = "sum of absolute beta Vs. iteration")

plot(tAll[1:k],errBar[1:k],xlab = "t",ylab = "errBar")
title(main = "apparent mean absolute predictive error rate")

plot(tAll[1:k],gamma[1:k],xlab = "t",ylab = "gamma")
title(main = "no information error rate")

betaIndex = colSums(beta[1:k,])!=0
betaPlot = beta[1:k,betaIndex]*S[1:k,betaIndex]
matplot(rbind(0,as.matrix(tAll[1:k,])),rbind(matrix(0,1,dim(betaPlot)[2]),betaPlot),type = "l", xlab ="t", ylab="beta")
title(main = "my lasso path")


###################### bootstrap algorithm ############################

# Perform Bootstrap Sampling
B = 25 # number of bootstraps
n = dim(myX)[1] # number of observations
p = dim(myX)[2] # number of predictors
BI = matrix(sample.int(n,n*B,replace = TRUE),B,n)
UniBI = matrix(0,B,1)
myXB = array(0,dim=c(B,n,p))
myYB = array(0,dim=c(B,n))
BIc = matrix(0,B,n) # the unselected elements are 1 for each bootstrap
for (i in 1:B){
  temp = 1:29
  BIc[i,temp[!(1:29 %in% BI[i,])]] = 1
}
for(i in 1:B){
  myXB[i,,] = myX[BI[i,],]
  myYB[i,] = myY[BI[i,]]
  UniBI[i,] = length(unique(BI[i,]))
}
```

```
# scale the data so that mean(y)=0, mean(x)=0 and var(x)=1
myYBMean = matrix(0,B,1)
myXBsd = matrix(0,B,p)
myXBMean = matrix(0,B,p)
for(i in 1:B){
 # mean(y)=0
 myYBMean[i,] = mean(myYB[i,])
 myYB[i,] = myYB[i,]-mean(myYB[i,])
 # mean(x)=0 and var(x)=1
 for (j in 1:p){
  myXBMean[i,j] = mean(myXB[i,,j])
  myXBsd[i,j] = sd(myXB[i,,j])
  myXB[i,,j] = (myXB[i,,j]-mean(myXB[i,,j]))/sd(myXB[i,,j])
 }
}
# run lesso for each bootstrap sample
betaB = array(NA,dim = c(B,p,p))
SB = array(NA,dim = c(B,p,p))
AcB = array(1,dim = c(B,p,p))
betaB = array(1,dim = c(B,p,p))
cBarB = array(1,dim = c(B,p,p))
tB = array(NA,dim = c(B,p))
errHatB = array(NA,dim = c(B,p,n))

for(bb in 1:B){
 cat(bb,sep=" ",fill=TRUE)
 ######################################## run lasso ########################################
 # Initualization
 k = 0
 A = matrix(0,p,p)
 Ac = matrix(1,p,p)
 beta = matrix(0,p,p)
 XA = matrix(0,n,k)
 cBar = matrix(0,p,p)
 u = matrix(0,n,p+1)
 betaA = matrix(0,n,k)
 jj = 0
 action = matrix(0,p,1)
 numErr = 10^-12
 S = matrix(0,p,p) # record the sign
 # run lasso
 for(k in 1:p){
  # if jj needs to leave the active variable set XA
  if(jj!=0){
   # cat(-jj,sep=" ",fill=TRUE)
   action[k,] = -jj
   # exclude jj from the active variable set XA
   A[k:p,jj] = 0
   Ac[k:p,jj] = 1
   cBar[k,] = t(myXB[bb,,])%*%(myYB[bb,]-u[,k]) # which one to use u[,k] or u[,k-1]
   s = sign(cBar[k,as.logical(A[k,])])
```

```
 XA = myXB[bb,,][,as.logical(A[k,])]
 XA = XA%*%diag(as.vector(s),length(s))
 # update cBar
 CBar = mean(abs(t(XA)%*%(myYB[bb,]-u[,k])))
 #  cat(abs(t(XA)%*%(myYB[bb,]-u[,k])),sep=" ",fill=TRUE)
 # update angles
 angles = t(myXB[bb,,])%*%XA%*%solve(t(XA)%*%XA,matrix(1,sum(A[k,]),1))
 temp = matrix(NA,p,3)
 temp[,1] = (CBar-cBar[k,])/(1-angles)
 temp[,2] = (CBar+cBar[k,])/(1+angles)
 temp[as.logical(temp[,1]<0),1] = +Inf
 temp[as.logical(temp[,2]<0),2] = +Inf
 temp[,3] = pmin(temp[,1],temp[,2])
 temp[as.logical(A[k,]),3] = +Inf
 c = min(temp[temp[,3]>numErr,3])
 # Lasso Modification
 if(k>1){
   bBar = solve(t(XA)%*%XA)%*%matrix(1,sum(A[k,]),1)
   d = matrix(0,p,1)
   # d[as.logical(A[k,]),1]=s*bBar
   d[as.logical(A[k,]),1]=bBar
   betaOld = matrix(0,p,1)
   # fix betaA delete the zero elements
   betaA = betaA[-sum(A[k-1,1:jj])]
   betaOld[as.logical(A[k,]),1]=betaA
   cLasso = matrix(0,p,1)
   cLasso[as.logical(A[k,]),1] = -betaOld[as.logical(A[k,]),1]/d[as.logical(A[k,]),1]
   if(sum(cLasso>0)==0){
     cStar = Inf
   }else{
     cStar = min(cLasso[cLasso>0])
   }
   if(cStar<c){
     c = cStar
     jj = which(cLasso==cStar) # the predictor that should leave the active variable set
   }else{
     jj = 0
   }
 }
 # update u
 u[,k+1] = c*XA%*%solve(t(XA)%*%XA)%*%matrix(1,sum(A[k,]),1)+u[,k]
 #update betaA
 b = c*solve(t(XA)%*%XA)%*%matrix(1,sum(A[k,]),1)
 betaA = betaA+b
 beta[k,as.logical(A[k,])]=betaA
}else{
 # update cBar (for k-1)
 cBar[k,] = t(myXB[bb,,])%*%(myYB[bb,]-u[,k])
 # find cBar for Ac elements: cBarc
 cBarc = cBar[k,]
 cBarc[as.logical(A[k,])]=0
```

```
# update XA
# cat(which.max(abs(cBarc)),sep=" ",fill=TRUE)
action[k,] = which.max(abs(cBarc))
A[k:p,which.max(abs(cBarc))] = 1
Ac[k:p,which.max(abs(cBarc))] = 0
s = sign(cBar[k,as.logical(A[k,])])
XA = myXB[bb,,][,as.logical(A[k,])]
XA = XA%*%diag(as.vector(s),length(s))
# update cBar
CBar = mean(abs(t(XA)%*%(myYB[bb,]-u[,k])))
#  cat(abs(t(XA)%*%(myYB[bb,]-u[,k])),sep=" ",fill=TRUE)
# update angles
angles = t(myXB[bb,,])%*%XA%*%solve(t(XA)%*%XA,matrix(1,sum(A[k,]),1))
temp = matrix(NA,p,3)
temp[,1] = (CBar-cBar[k,])/(1-angles)
temp[,2] = (CBar+cBar[k,])/(1+angles)
temp[as.logical(temp[,1]<0),1] = +Inf
temp[as.logical(temp[,2]<0),2] = +Inf
temp[,3] = pmin(temp[,1],temp[,2])
temp[as.logical(A[k,]),3] = +Inf
c = min(temp[temp[,3]>numErr,3])
# Lasso Modification
if(k>1){
  bBar = solve(t(XA)%*%XA)%*%matrix(1,sum(A[k,]),1)
  d = matrix(0,p,1)
  # d[as.logical(A[k,]),1]=s*bBar
  d[as.logical(A[k,]),1]=bBar
  betaOld = matrix(0,p,1)
  betaOld[as.logical(A[k-1,]),1]=betaA

  ## something is wrong here
  cLasso = matrix(0,p,1)
  cLasso[as.logical(A[k,]),1] = -betaOld[as.logical(A[k,]),1]/d[as.logical(A[k,]),1]
  if(sum(cLasso>0)==0){
    cStar = Inf
  }else{
    cStar = min(cLasso[cLasso>0])
  }
  if(cStar<c){
    c = cStar
    jj = which(cLasso==cStar) # the predictor that should leave the active variable set
  }else{
    jj = 0
  }
}
# update u
u[,k+1] = c*XA%*%solve(t(XA)%*%XA)%*%matrix(1,sum(A[k,]),1)+u[,k]
#update betaA
b = c*solve(t(XA)%*%XA)%*%matrix(1,sum(A[k,]),1)
j = sum(A[k,1:which.max(abs(cBarc))])
betaA = betaA+b[-j]
```

```
    betaA = append(betaA,b[j],after=j-1)
    beta[k,as.logical(A[k,])]=betaA
   }
   S[k,] = sign(cBar[k,])
   # stop criteria
   if (dim(XA)[2]==UniBI[bb,]-1 & jj==0){
    break
   }
  }
  # calculate indices
  errHat = matrix(NA,p,n) # leave-one-our bootstrap error rate
  # calculate errHat: leave-one-our bootstrap error rate
  for(kk in 1:k){
   errHat[kk,as.logical(BIc[bb,])] = abs((myY[as.logical(BIc[bb,])]-myYBMean[bb,1])-as.matrix(((myX[as.logical(BIc[bb,]),]-
myXBMean[bb,])/myXBsd[bb,])[,as.logical(A[kk,])])%*%(S[kk,as.logical(A[kk,])]*beta[kk,as.logical(A[kk,])]))
  }
  tB[bb,k] = sum(abs(beta[kk,]))
  errHatB[bb,,] = errHat
}

# Find the tmax and set the tGrid
tGridR = 1000
tGrid = seq(from = 0, to = max(max(tB,na.rm = TRUE),tAll,na.rm = TRUE), length.out = tGridR)
tBPlot = array(NA,dim = c(B,tGridR))
errBarPlot = array(NA, dim = c(tGridR))
gammaPlot = array(NA, dim = c(tGridR))
errHatBPlot = array(NA,dim = c(B,tGridR,n))
# calculate

# plot errBar
# adding values for t = 0
errBarPlot =
matrix(unlist(approx(rbind(0,as.matrix(tAll)),rbind(sum(abs(myY))/n,as.matrix(errBar)),tGrid,method="linear",rule=1)$y))
# plot(tGrid, errBarPlot)

# plot gamma
# adding values for t = 0
gammaPlot =
matrix(unlist(approx(rbind(0,as.matrix(tAll)),rbind(sum(abs(matrix(1,dim(myX)[1],1)%*%myY))/(dim(myX)[1])^2,as.matrix(gam
ma)),tGrid,method="linear",rule=1)$y))
# plot(tGrid, gammaPlot)

# plot errHat
for(bb in 1:B){
  for(j in 1:n){
    if (is.na(errHatB[bb,1,j])==FALSE){
      # adding values for t = 0
      errHatBPlot[bb,,j] = matrix(unlist(approx(rbind(0,as.matrix(tB[bb,])),rbind(abs(myY[j]-
myYBMean[bb,1]),as.matrix(errHatB[bb,,j])),tGrid,method="linear",rule=1)$y))
    }
  }
```

```
}
# plot(tGrid,errHatBPlot[1,,2])

errHatBPlus = matrix(NA,tGridR,n)
for(i in 1:n){
 cat(sum(BIc[,i]),fill = TRUE)
 # if we the i th observation has not been sampled in a draw
 if(sum(BIc[,i]>0)){
  CIndex = BIc[,i]==1

  if(sum(BIc[,i]>0)==1){
   errHatBPlus[,i] = errHatBPlot[as.logical(CIndex),,i]
  }else{
   errHatBPlus[,i] = colMeans(errHatBPlot[as.logical(CIndex),,i], na.rm = TRUE)
  }
 }
}
errHatBPlusPlot = rowMeans(errHatBPlus, na.rm =TRUE)
# plot(tGrid,errHatBPlusPlot)

# plot RHat
RHat = matrix(NA,tGridR,1)
RHat = (errHatBPlusPlot-errBarPlot)/(gammaPlot-errBarPlot)
# sum(is.na(errHatBPlusPlot))
# sum(is.na(errBarPlot))
# sum(is.na(errHatBPlusPlot-errBarPlot))
# plot(tGrid,RHat)

# plot ErrHatPlus
wHat = 0.632/(1-0.368*RHat)
# plot(tGrid,wHat)
ErrHatPlus = (1-wHat)*errBarPlot+wHat*errHatBPlusPlot
ErrHatPlus[as.logical(ErrHatPlus<0)]=NA
plot(tGrid,ErrHatPlus)

# plotbeta
betaBPlot = array(NA,dim =c(dim(betaPlot)[2],tGridR))
for(i in 1:dim(betaPlot)[2]){
 betaBPlot[i,] =
matrix(unlist(approx(rbind(0,as.matrix(tAll[1:k,])),rbind(0,as.matrix(betaPlot[,i])),tGrid,method="linear",rule=1)$y))
}
matplot(tGrid,t(betaBPlot),type = "l", xlab ="t", ylab="beta")

# plot everything
dev.new(width = 6, hight = 6)
par(mfrow=c(2,2),cex=0.6)
# plot errBar
plot(tGrid, errBarPlot)
# plot gamma
plot(tGrid, gammaPlot)
# plot leave-one-out
```

```
plot(tGrid,errHatBPlusPlot)
# plot err0.632
plot(tGrid,ErrHatPlus)
```