STSCI 4740 Final Project Group 10 – Appendix (R Output)

```
> setwd("C:/Users/14014/Documents/Cornell_Fall_2019/STS
CI_4740/STSCI_4740_FinalProject")
> divorce <- read.csv('divorce.csv', header=TRUE)
> #fix(divorce)
> attach(divorce)
The following object is masked _by_ .GlobalEnv:

    Class

The following objects are masked from divorce (pos = 3)
:

    Atr1, Atr10, Atr11, Atr12, Atr13, Atr14,
    Atr15, Atr16, Atr17, Atr18, Atr19, Atr2,
    Atr20, Atr21, Atr22, Atr23, Atr24, Atr25,
    Atr26, Atr27, Atr28, Atr29, Atr3, Atr30,
    Atr31, Atr32, Atr33, Atr34, Atr35, Atr36,
    Atr37, Atr38, Atr39, Atr4, Atr40, Atr41,
    Atr42, Atr43, Atr44, Atr45, Atr46, Atr47,
    Atr48, Atr49, Atr5, Atr50, Atr51, Atr52,
    Atr53, Atr54, Atr6, Atr7, Atr8, Atr9,
    Class

The following objects are masked from divorce (pos = 4)
:

    Atr1, Atr10, Atr11, Atr12, Atr13, Atr14,
    Atr15, Atr16, Atr17, Atr18, Atr19, Atr2,
    Atr20, Atr21, Atr22, Atr23, Atr24, Atr25,
    Atr26, Atr27, Atr28, Atr29, Atr3, Atr30,
    Atr31, Atr32, Atr33, Atr34, Atr35, Atr36,
    Atr37, Atr38, Atr39, Atr4, Atr40, Atr41,
    Atr42, Atr43, Atr44, Atr45, Atr46, Atr47,
    Atr48, Atr49, Atr5, Atr50, Atr51, Atr52,
    Atr53, Atr54, Atr6, Atr7, Atr8, Atr9,
    Class

The following objects are masked from divorce (pos = 14
):

    Atr1, Atr10, Atr11, Atr12, Atr13, Atr14,
    Atr15, Atr16, Atr17, Atr18, Atr19, Atr2,
    Atr20, Atr21, Atr22, Atr23, Atr24, Atr25,
    Atr26, Atr27, Atr28, Atr29, Atr3, Atr30,
    Atr31, Atr32, Atr33, Atr34, Atr35, Atr36,
    Atr37, Atr38, Atr39, Atr4, Atr40, Atr41,
    Atr42, Atr43, Atr44, Atr45, Atr46, Atr47,
    Atr48, Atr49, Atr5, Atr50, Atr51, Atr52,
    Atr53, Atr54, Atr6, Atr7, Atr8, Atr9,
    Class

> #install.packages("Hmisc")
```

```
> #library(Hmisc)
> #describe(divorce)
> Class <- factor(Class)
> is.factor(Class)
[1] TRUE
> ### Quick correlation matrix output to see which para
meters are good
> p <- 54
> k <- 5
> folds <- sample (1:k,nrow(divorce),replace=TRUE)
> for(j in 1:k){
+   set.seed(1)
+   train = divorce[folds!=j,]
+
+   corr_matrix <- matrix(c(rep(1,p),rep(1,p)), ncol=2)
+   colnames(corr_matrix) <- c("Atr","Correlation")
+
+   for (i in 1:54){
+     x <- c("Atr", i)
+     y <- paste(x, collapse= "")
+     correl = cor(train[ , i], train[ , 55])
+     corr_matrix[i,2] <- correl
+     corr_matrix[i,1] <- y
+   }
+   corr_matrix_sort <- corr_matrix[order(corr_matrix[,
2]),]
+   print(tail(corr_matrix_sort))
+ }
      Atr     Correlation
[49,] "Atr9"  "0.918764421686489"
[50,] "Atr11" "0.921040320729143"
[51,] "Atr18" "0.928324737691998"
[52,] "Atr40" "0.93176697034468"
[53,] "Atr19" "0.938507619834182"
[54,] "Atr17" "0.941020734208694"
      Atr     Correlation
[49,] "Atr38" "0.908224823593169"
[50,] "Atr11" "0.91909019979494"
[51,] "Atr18" "0.920965126287096"
[52,] "Atr17" "0.921998332657598"
[53,] "Atr19" "0.922263641646883"
[54,] "Atr40" "0.948874474286447"
      Atr     Correlation
[49,] "Atr18" "0.919625583000681"
[50,] "Atr11" "0.921042759515559"
[51,] "Atr9"  "0.924236721302092"
[52,] "Atr17" "0.924903246391796"
[53,] "Atr40" "0.934627647404616"
[54,] "Atr19" "0.936738580233108"
      Atr     Correlation
[49,] "Atr15" "0.90485644723485"
[50,] "Atr11" "0.911740217681647"
[51,] "Atr19" "0.918678103080277"
[52,] "Atr18" "0.921457252524313"
[53,] "Atr17" "0.928342822102871"
[54,] "Atr40" "0.932726086069826"
```

```
      Atr     Correlation
[49,] "Atr41" "0.915393046745463"
[50,] "Atr11" "0.919512598526148"
[51,] "Atr18" "0.926739097887571"
[52,] "Atr19" "0.927495398189489"
[53,] "Atr17" "0.931827344566063"
[54,] "Atr40" "0.945048764279963"
> plot(Atr40, Class, main="Scatterplot of Atr40 vs. Cla
ss",
+      xlab="Atr40", ylab="Class", pch=19)
> set.seed(1)
> # linear regression #
> #1) Perform 5-fold CV to get mse
> lin.mod.mse_vector = rep(0,k)
> for(j in 1:k){
+   set.seed(1)
+
+   train = divorce[folds!=j,]
+   test = divorce[folds ==j,]
+
+   lin.mod = lm(Class~., data=train)
+   lin.mod_predict = predict(lin.mod, newdata = test)
+
+   error = mean((test$Class - lin.mod_predict)^2)
+   lin.mod.mse_vector[j]=error
+ }
> lin.mod.mse_vector
[1] 0.056688434 0.031478861 0.046281356 0.006454491
[5] 0.043046155
> mean(lin.mod.mse_vector)
[1] 0.03678986
> # Linear regression - all parameters / classification
threshold
> lin.mod.class.mse_vector = rep(0,k)
> for(j in 1:k){
+   set.seed(1)
+
+   train = divorce[folds!=j,]
+   test = divorce[folds ==j,]
+
+   lin.mod = lm(Class~., data=train)
+   lin.mod_predict = predict(lin.mod, newdata = test)
+   for(i in 1:length(lin.mod_predict)){
+     if(lin.mod_predict[i]>=0.5){
+       lin.mod_predict[i]=1
+     } else{
+       lin.mod_predict[i]=0
+     }
+   }
+
+   error = mean((test$Class - lin.mod_predict)^2)
+   lin.mod.class.mse_vector[j]=error
+ }
> lin.mod.class.mse_vector
[1] 0.05128205 0.00000000 0.03225806 0.00000000
[5] 0.03030303
```

```
> mean(lin.mod.class.mse_vector)
[1] 0.02276863
> # Basic polynomial linear regression with value of 2
- all parameters
> log.mod.mse_vector = rep(0,k)
> for(j in 1:k){
+   set.seed(1)
+
+   train = divorce[folds!=j,]
+   test = divorce[folds ==j,]
+
+   xnam=paste("Atr",2:54,sep="")
+   fmla=as.formula(paste("Class~poly(Atr1+",paste(xnam
, collapse="+"),paste(",2,raw=T)")))
+   log.mod=lm(fmla,data=train) #use raw polynomial x,
x^2,x^3,...
+   log.mod_predict = predict(log.mod, newdata = test)
+   error = mean((test$Class - log.mod_predict)^2)
+   log.mod.mse_vector[j]=error
+ }
> log.mod.mse_vector
[1] 0.020820286 0.003320101 0.022064738 0.008646856
[5] 0.023494827
> mean(log.mod.mse_vector)
[1] 0.01566936
> # Polynomial - value of 2 - all parameters / include
classification threshold
> log.mod.class.mse_vector = rep(0,k)
> for(j in 1:k){
+   set.seed(1)
+
+   train = divorce[folds!=j,]
+   test = divorce[folds ==j,]
+
+   xnam=paste("Atr",2:54,sep="")
+   fmla=as.formula(paste("Class~poly(Atr1+",paste(xnam
, collapse="+"),paste(",2,raw=T)")))
+   log.mod=lm(fmla,data=train)
+   log.mod_predict = predict(log.mod, newdata = test)
+   for(i in 1:length(log.mod_predict)){
+     if(log.mod_predict[i]>=0.5){
+       log.mod_predict[i]=1
+     } else{
+       log.mod_predict[i]=0
+     }
+   }
+   error = mean((test$Class - log.mod_predict)^2)
+   log.mod.class.mse_vector[j]=error
+ }
> log.mod.class.mse_vector
[1] 0.02564103 0.00000000 0.03225806 0.00000000
[5] 0.03030303
> mean(log.mod.class.mse_vector)
[1] 0.01764042
> # polynomial logistic regression #
> log.mod.mse_vector1 = rep(0,k)
```

```
> for(j in 1:k){
+   set.seed(1)
+
+   train = divorce[folds!=j,]
+   test = divorce[folds ==j,]
+
+   xnam=paste("Atr",2:54,sep="")
+   fmla=as.formula(paste("Class~poly(Atr1+",paste(xnam
, collapse="+"),paste(",2,raw=T)")))
+   log.mod=lm(fmla,data=train,family=binomial) #only d
ifference: family=binomial
+   log.mod_predict = predict(log.mod, newdata = test)
+   error = mean((test$Class - log.mod_predict)^2)
+   log.mod.mse_vector1[j]=error
+ }
Warning messages:
1: In lm.fit(x, y, offset = offset, singular.ok = singu
lar.ok, ...) :
 extra argument 'family' will be disregarded
2: In lm.fit(x, y, offset = offset, singular.ok = singu
lar.ok, ...) :
 extra argument 'family' will be disregarded
3: In lm.fit(x, y, offset = offset, singular.ok = singu
lar.ok, ...) :
 extra argument 'family' will be disregarded
4: In lm.fit(x, y, offset = offset, singular.ok = singu
lar.ok, ...) :
 extra argument 'family' will be disregarded
5: In lm.fit(x, y, offset = offset, singular.ok = singu
lar.ok, ...) :
 extra argument 'family' will be disregarded
> log.mod.mse_vector1
[1] 0.020820286 0.003320101 0.022064738 0.008646856
[5] 0.023494827
> mean(log.mod.mse_vector1)
[1] 0.01566936
> ### Perform basic LDA with all parameters
> library(MASS)
> lda.error=rep(0,k)
> for (j in 1:k){
+   train = divorce[folds!=j,]
+   test = divorce[folds==j,]
+   class.test = Class[folds==j]
+   lda.fit=lda(as.factor(Class)~., data=train)
+   lda.fit
+
+   lda.pred=predict(lda.fit, test)
+   lda.class=lda.pred$class
+   #print(table(lda.class,class.test))
+   mean(lda.class==class.test)
+
+   topleft <- table(lda.class,class.test)[1]
+   bottomleft <- table(lda.class,class.test)[2]
+   topright <- table(lda.class,class.test)[3]
+   bottomright <- table(lda.class,class.test)[4]
+
```

```
+   lda.error[j]= (topright + bottomleft)/(topleft + bo
ttomright + topright + bottomleft)
+ }
> lda.error
[1] 0.05128205 0.00000000 0.03225806 0.00000000
[5] 0.03030303
> mean(lda.error)
[1] 0.02276863
> ### Perform CV for k in KNN with all parameters
> library(class)
> knn.error=rep(0,k)
> for (i in 1:k){
+   train = divorce[folds!=i,]
+   test = divorce[folds==i,]
+   class.test = Class[folds==i]
+   train.X = cbind(train[,c(1:54)])
+   test.X = cbind(test[,c(1:54)])
+   class.train = Class[folds!=i]
+
+   k.error=rep(0,100)
+
+   for(j in 1:100){
+     knn.pred=knn(train.X,test.X,class.train,k=j)
+     knn.class <- knn.pred
+     #print(table(knn.class,class.test))
+
+     topleft <- table(knn.class,class.test)[1]
+     bottomleft <- table(knn.class,class.test)[2]
+     topright <- table(knn.class,class.test)[3]
+     bottomright <- table(knn.class,class.test)[4]
+
+     error_rate <- (topright + bottomleft)/(topleft +
bottomright + topright + bottomleft)
+     k.error[j] <- error_rate
+   }
+   k.error
+   min <- which.min(k.error)
+
+   knn.pred=knn(train.X,test.X,class.train,k=min)
+   knn.class <- knn.pred
+   table(knn.class,class.test)
+
+   error_rate <- (topright + bottomleft)/(topleft + bo
ttomright + topright + bottomleft)
+   knn.error[i] <- error_rate
+
+ }
> knn.error
[1] 0.05128205 0.00000000 0.03225806 0.00000000
[5] 0.03030303
> mean(knn.error)
[1] 0.02276863

>

> ### Basic Decision Tree (all 54 parameters)
```

```
> library(tree)
> tree.error=rep(0,k)
> for (j in 1:k){
+   set.seed(1)
+   train = divorce[folds!=j,]
+   test = divorce[folds==j,]
+   class.test = Class[folds==j]
+
+   tree.divorce=tree(as.factor(Class)~.,train)
+   plot(tree.divorce)
+   text(tree.divorce)
+   tree.pred=predict(tree.divorce,test,type="class")
+   tree.table=table(tree.pred,class.test)
+
+   topleft <- tree.table[1]
+   bottomleft <- tree.table[2]
+   topright <- tree.table[3]
+   bottomright <- tree.table[4]
+
+   tree.error[j]= (topright + bottomleft)/(topleft + b
ottomright + topright + bottomleft)
+ }
> tree.error
[1] 0.05128205 0.00000000 0.03225806 0.03030303
[5] 0.03030303
> mean(tree.error)
[1] 0.02882924
> ### Re-run LDA with only top 5 parameters
> lda5.error=rep(0,k)
> for (j in 1:k){
+   set.seed(1)
+   train = divorce[folds!=j,]
+   test = divorce[folds==j,]
+   class.test = Class[folds==j]
+   lda.fit=lda(as.factor(Class)~Atr40+Atr17+Atr19+Atr1
8+Atr11, data=train)
+   lda.fit
+
+   lda.pred=predict(lda.fit, test)
+   lda.class=lda.pred$class
+   table(lda.class,class.test)
+   mean(lda.class==class.test)
+
+   topleft <- table(lda.class,class.test)[1]
+   bottomleft <- table(lda.class,class.test)[2]
+   topright <- table(lda.class,class.test)[3]
+   bottomright <- table(lda.class,class.test)[4]
+
+   lda5.error[j]= (topright + bottomleft)/(topleft + b
ottomright + topright + bottomleft)
+ }
> lda5.error
[1] 0.05128205 0.00000000 0.03225806 0.00000000
[5] 0.03030303
> mean(lda5.error)
[1] 0.02276863
```

```
> ### Re-run QDA with only top 5 parameters - should wo
rk now
> qda5.error=rep(0,k)
> for (j in 1:k){
+   set.seed(1)
+   train = divorce[folds!=j,]
+   test = divorce[folds==j,]
+   class.test = Class[folds==j]
+
+   qda.fit=qda(as.factor(Class)~Atr40+Atr17+Atr19+Atr1
8+Atr11, data=train)
+   qda.class=predict(qda.fit,test)$class
+   print(table(qda.class,class.test))
+   mean(qda.class==class.test)
+
+   topleft <- table(qda.class,class.test)[1]
+   bottomleft <- table(qda.class,class.test)[2]
+   topright <- table(qda.class,class.test)[3]
+   bottomright <- table(qda.class,class.test)[4]
+
+   qda5.error[j]= (topright + bottomleft)/(topleft + b
ottomright + topright + bottomleft)
+ }
         class.test
qda.class  0  1
        0 20  1
        1  0 18
         class.test
qda.class  0  1
        0 13  0
        1  0 21
         class.test
qda.class  0  1
        0 18  1
        1  0 12
         class.test
qda.class  0  1
        0 18  0
        1  0 15
         class.test
qda.class  0  1
        0 17  1
        1  0 15
> qda5.error
[1] 0.02564103 0.00000000 0.03225806 0.00000000
[5] 0.03030303
> mean(qda5.error)
[1] 0.01764042
> ### CV for k with KNN with only top 5 parameters
> knn5.error=rep(0,k)
> for (i in 1:k){
+   set.seed(1)
+   train = divorce[folds!=i,]
+   test = divorce[folds==i,]
+   class.test = Class[folds==i]
+   train.X = cbind(train[,c(11,17,18,19,40)])
```

```
+   test.X = cbind(test[,c(11,17,18,19,40)])
+   class.train = Class[folds!=i]
+
+   k.error=rep(0,100)
+
+   for(j in 1:100){
+     knn.pred=knn(train.X,test.X,class.train,k=j)
+     knn.class <- knn.pred
+     table(knn.class,class.test)
+
+     topleft <- table(knn.class,class.test)[1]
+     bottomleft <- table(knn.class,class.test)[2]
+     topright <- table(knn.class,class.test)[3]
+     bottomright <- table(knn.class,class.test)[4]
+
+     error_rate <- (topright + bottomleft)/(topleft +
bottomright + topright + bottomleft)
+     k.error[j] <- error_rate
+   }
+   k.error
+   min <- which.min(k.error)
+   print(min)
+
+   knn.pred=knn(train.X,test.X,class.train,k=min)
+   knn.class <- knn.pred
+   table(knn.class,class.test)
+
+   error_rate <- (topright + bottomleft)/(topleft + bo
ttomright + topright + bottomleft)
+   knn5.error[i] <- error_rate
+
+ }
[1] 1
[1] 1
[1] 1
[1] 1
[1] 1
> knn5.error
[1] 0.48717949 0.61764706 0.03225806 0.00000000
[5] 0.48484848
> mean(knn5.error)
[1] 0.3243866
> ### Decision Tree with 5 parameters only
> tree5.error=rep(0,k)
> for (j in 1:k){
+   set.seed(1)
+   train = divorce[folds!=j,]
+   test = divorce[folds==j,]
+   class.test = Class[folds==j]
+
+   tree5.divorce=tree(as.factor(Class)~.,train)
+   plot(tree5.divorce)
+   text(tree5.divorce)
+   tree5.pred=predict(tree5.divorce,test,type="class")
+   tree5.table=table(tree5.pred,class.test)
+
```

```
+   topleft <- tree.table[1]
+   bottomleft <- tree.table[2]
+   topright <- tree.table[3]
+   bottomright <- tree.table[4]
+
+   tree5.error[j]= (topright + bottomleft)/(topleft +
bottomright + topright + bottomleft)
+ }
> tree5.error
[1] 0.03030303 0.03030303 0.03030303 0.03030303
[5] 0.03030303
> mean(tree5.error)
[1] 0.03030303
> xnam=paste("Atr",1:54,sep="")
> fmla=as.formula(paste("Class~",paste(xnam, collapse="
+")))
> fmla
Class ~ Atr1 + Atr2 + Atr3 + Atr4 + Atr5 + Atr6 + Atr7
+ Atr8 +
    Atr9 + Atr10 + Atr11 + Atr12 + Atr13 + Atr14 + Atr1
5 + Atr16 +
    Atr17 + Atr18 + Atr19 + Atr20 + Atr21 + Atr22 + Atr
23 + Atr24 +
    Atr25 + Atr26 + Atr27 + Atr28 + Atr29 + Atr30 + Atr
31 + Atr32 +
    Atr33 + Atr34 + Atr35 + Atr36 + Atr37 + Atr38 + Atr
39 + Atr40 +
    Atr41 + Atr42 + Atr43 + Atr44 + Atr45 + Atr46 + Atr
47 + Atr48 +
    Atr49 + Atr50 + Atr51 + Atr52 + Atr53 + Atr54
> # First Step:
> #fitting the model without using any subset selection
s
> library(boot)
> set.seed(1)
> cv.logi_error1.5=rep(0,5)
> for (i in 1:5){
+   logi.fit=glm(Class~.,data=divorce,family='binomial'
)
+   cv.logi_error1.5[i]=cv.glm(divorce,logi.fit, K=5)$d
elta[1]
+ }
There were 50 or more warnings (use warnings() to see t
he first 50)
> cv.logi_error1.5
[1] 0.03449928 0.05066133 0.02453584 0.04774724
[5] 0.03648031
> mean(cv.logi_error1.5)
[1] 0.0387848
> cv.logiWC_error1.5=rep(0,5)
> for (i in 1:5){
+   logiWC.fit=glm(Class~., data=divorce, family='binom
ial',control=list(maxit=1000))
+   cv.logiWC_error1.5[i]=cv.glm(divorce,logiWC.fit, K=
5)$delta[1]
+ }
```

```
There were 30 warnings (use warnings() to see them)
> cv.logiWC_error1.5
[1] 0.03220952 0.03154547 0.02935895 0.02339867
[5] 0.04088782
> mean(cv.logiWC_error1.5)
[1] 0.03148009
> # ****** GAM Setup *******
> library(gam)
> newx=paste("s(Atr",1:54,',df=k)',sep="")
> fmla2=as.formula(paste('I(Class==1)~',paste(newx, col
lapse="+")))
> fmla2
I(Class == 1) ~ s(Atr1, df = k) + s(Atr2, df = k) + s(A
tr3, df = k) +
    s(Atr4, df = k) + s(Atr5, df = k) + s(Atr6, df = k)
+ s(Atr7,
    df = k) + s(Atr8, df = k) + s(Atr9, df = k) + s(Atr
10, df = k) +
    s(Atr11, df = k) + s(Atr12, df = k) + s(Atr13, df =
k) +
    s(Atr14, df = k) + s(Atr15, df = k) + s(Atr16, df =
k) +
    s(Atr17, df = k) + s(Atr18, df = k) + s(Atr19, df =
k) +
    s(Atr20, df = k) + s(Atr21, df = k) + s(Atr22, df =
k) +
    s(Atr23, df = k) + s(Atr24, df = k) + s(Atr25, df =
k) +
    s(Atr26, df = k) + s(Atr27, df = k) + s(Atr28, df =
k) +
    s(Atr29, df = k) + s(Atr30, df = k) + s(Atr31, df =
k) +
    s(Atr32, df = k) + s(Atr33, df = k) + s(Atr34, df =
k) +
    s(Atr35, df = k) + s(Atr36, df = k) + s(Atr37, df =
k) +
    s(Atr38, df = k) + s(Atr39, df = k) + s(Atr40, df =
k) +
    s(Atr41, df = k) + s(Atr42, df = k) + s(Atr43, df =
k) +
    s(Atr44, df = k) + s(Atr45, df = k) + s(Atr46, df =
k) +
    s(Atr47, df = k) + s(Atr48, df = k) + s(Atr49, df =
k) +
    s(Atr50, df = k) + s(Atr51, df = k) + s(Atr52, df =
k) +
    s(Atr53, df = k) + s(Atr54, df = k)
> gam.fit=gam(fmla2,family=binomial,data=divorce)
Warning messages:
1: In model.matrix.default(mt, mf, contrasts) :
  non-list contrasts argument ignored
2: In gam(fmla2, family = binomial, data = divorce) :
  Residual degrees of freedom are negative or zero.  Th
is occurs when the sum of the parametric and nonparamet
ric degrees of freedom exceeds the number of observatio
```

```
ns.  The model is probably too complex for the amount o
f data available.
> summary(gam.fit)

Call: gam(formula = fmla2, family = binomial, data = di
vorce)

(Dispersion Parameter for binomial family taken to be 1
)

    Null Deviance: 235.6465 on 169 degrees of freedom
Residual Deviance: 0 on -47 degrees of freedom
AIC: 434.0001

Number of Local Scoring Iterations: 24

Anova for Parametric Effects
                  Df     Sum Sq    Mean Sq
s(Atr1, df = k)    1 1.6181e-06 1.6181e-06
s(Atr2, df = k)    1 8.3070e-08 8.3070e-08
s(Atr3, df = k)    1 1.4720e-08 1.4720e-08
s(Atr4, df = k)    1 2.2880e-08 2.2880e-08
s(Atr5, df = k)    1 1.1392e-07 1.1392e-07
s(Atr6, df = k)    1 9.9140e-08 9.9140e-08
s(Atr7, df = k)    1 1.0320e-08 1.0320e-08
s(Atr8, df = k)    1 6.1670e-08 6.1670e-08
s(Atr9, df = k)    1 1.9430e-08 1.9430e-08
s(Atr10, df = k)   1 3.2000e-10 3.2000e-10
s(Atr11, df = k)   1 2.9860e-08 2.9860e-08
s(Atr12, df = k)   1 9.9100e-09 9.9100e-09
s(Atr13, df = k)   1 5.9200e-09 5.9200e-09
s(Atr14, df = k)   1 9.4000e-10 9.4000e-10
s(Atr15, df = k)   1 2.6900e-09 2.6900e-09
s(Atr16, df = k)   1 7.2000e-10 7.2000e-10
s(Atr17, df = k)   1 3.0200e-09 3.0200e-09
s(Atr18, df = k)   1 2.7700e-09 2.7700e-09
s(Atr19, df = k)   1 1.6000e-10 1.6000e-10
s(Atr20, df = k)   1 1.3000e-10 1.3000e-10
s(Atr21, df = k)   1 0.0000e+00 0.0000e+00
s(Atr22, df = k)   1 1.0750e-08 1.0750e-08
s(Atr23, df = k)   1 4.0200e-09 4.0200e-09
s(Atr24, df = k)   1 2.9100e-09 2.9100e-09
s(Atr25, df = k)   1 8.8600e-09 8.8600e-09
s(Atr26, df = k)   1 2.0880e-08 2.0880e-08
s(Atr27, df = k)   1 4.2200e-09 4.2200e-09
s(Atr28, df = k)   1 7.4000e-09 7.4000e-09
s(Atr29, df = k)   1 7.9000e-10 7.9000e-10
s(Atr30, df = k)   1 2.2200e-09 2.2200e-09
s(Atr31, df = k)   1 7.4500e-09 7.4500e-09
s(Atr32, df = k)   1 4.1000e-10 4.1000e-10
s(Atr33, df = k)   1 2.6000e-10 2.6000e-10
s(Atr34, df = k)   1 2.0100e-08 2.0100e-08
s(Atr35, df = k)   1 5.7200e-09 5.7200e-09
s(Atr36, df = k)   1 6.6300e-09 6.6300e-09
s(Atr37, df = k)   1 0.0000e+00 0.0000e+00
s(Atr38, df = k)   1 4.5000e-10 4.5000e-10
```

```
s(Atr39, df = k)   1 2.6400e-09 2.6400e-09
s(Atr40, df = k)   1 1.1890e-08 1.1890e-08
s(Atr41, df = k)   1 4.0000e-11 4.0000e-11
s(Atr42, df = k)   1 2.1000e-10 2.1000e-10
s(Atr43, df = k)   1 5.2000e-10 5.2000e-10
s(Atr44, df = k)   1 8.0000e-10 8.0000e-10
s(Atr45, df = k)   1 1.0000e-10 1.0000e-10
s(Atr46, df = k)   1 7.0000e-11 7.0000e-11
s(Atr47, df = k)   1 3.2400e-09 3.2400e-09
s(Atr48, df = k)   1 7.4000e-10 7.4000e-10
s(Atr49, df = k)   1 1.4800e-09 1.4800e-09
s(Atr50, df = k)   1 1.0000e-11 1.0000e-11
s(Atr51, df = k)   1 1.8100e-09 1.8100e-09
s(Atr52, df = k)   1 1.4950e-08 1.4950e-08
s(Atr53, df = k)   1 1.5000e-10 1.5000e-10
s(Atr54, df = k)   1 2.0000e-11 2.0000e-11
Residuals        -47 0.0000e+00 0.0000e+00
                     F value Pr(>F)
s(Atr1, df = k)  -2.9554e+08
s(Atr2, df = k)  -1.5173e+07
s(Atr3, df = k)  -2.6880e+06
s(Atr4, df = k)  -4.1780e+06
s(Atr5, df = k)  -2.0807e+07
s(Atr6, df = k)  -1.8107e+07
s(Atr7, df = k)  -1.8848e+06
s(Atr8, df = k)  -1.1264e+07
s(Atr9, df = k)  -3.5482e+06
s(Atr10, df = k) -5.8766e+04
s(Atr11, df = k) -5.4540e+06
s(Atr12, df = k) -1.8096e+06
s(Atr13, df = k) -1.0816e+06
s(Atr14, df = k) -1.7119e+05
s(Atr15, df = k) -4.9207e+05
s(Atr16, df = k) -1.3084e+05
s(Atr17, df = k) -5.5226e+05
s(Atr18, df = k) -5.0630e+05
s(Atr19, df = k) -2.9870e+04
s(Atr20, df = k) -2.2940e+04
s(Atr21, df = k) -5.3413e+02
s(Atr22, df = k) -1.9637e+06
s(Atr23, df = k) -7.3419e+05
s(Atr24, df = k) -5.3062e+05
s(Atr25, df = k) -1.6182e+06
s(Atr26, df = k) -3.8128e+06
s(Atr27, df = k) -7.7026e+05
s(Atr28, df = k) -1.3508e+06
s(Atr29, df = k) -1.4409e+05
s(Atr30, df = k) -4.0557e+05
s(Atr31, df = k) -1.3611e+06
s(Atr32, df = k) -7.4995e+04
s(Atr33, df = k) -4.6908e+04
s(Atr34, df = k) -3.6717e+06
s(Atr35, df = k) -1.0453e+06
s(Atr36, df = k) -1.2111e+06
s(Atr37, df = k) -8.4973e+02
s(Atr38, df = k) -8.1957e+04
```

```
s(Atr39, df = k) -4.8309e+05
s(Atr40, df = k) -2.1717e+06
s(Atr41, df = k) -7.9474e+03
s(Atr42, df = k) -3.8962e+04
s(Atr43, df = k) -9.5392e+04
s(Atr44, df = k) -1.4645e+05
s(Atr45, df = k) -1.8223e+04
s(Atr46, df = k) -1.2564e+04
s(Atr47, df = k) -5.9165e+05
s(Atr48, df = k) -1.3523e+05
s(Atr49, df = k) -2.7010e+05
s(Atr50, df = k) -1.7300e+03
s(Atr51, df = k) -3.2995e+05
s(Atr52, df = k) -2.7301e+06
s(Atr53, df = k) -2.7463e+04
s(Atr54, df = k) -3.8505e+03
Residuals

Anova for Nonparametric Effects
                 Npar Df Npar Chisq P(Chi)
(Intercept)
s(Atr1, df = k)        3 1.0202e-09      1
s(Atr2, df = k)        3 1.2342e-09      1
s(Atr3, df = k)        3 3.1302e-09      1
s(Atr4, df = k)        3 2.8132e-09      1
s(Atr5, df = k)        3 3.1820e-10      1
s(Atr6, df = k)        3 4.3560e-10      1
s(Atr7, df = k)        3 3.3831e-09      1
s(Atr8, df = k)        3 1.2689e-09      1
s(Atr9, df = k)        3 4.3280e-09      1
s(Atr10, df = k)       3 7.6840e-10      1
s(Atr11, df = k)       3 1.5846e-09      1
s(Atr12, df = k)       3 4.2030e-10      1
s(Atr13, df = k)       3 7.8200e-11      1
s(Atr14, df = k)       3 1.0869e-09      1
s(Atr15, df = k)       3 9.0400e-10      1
s(Atr16, df = k)       3 7.8420e-10      1
s(Atr17, df = k)       3 2.8769e-09      1
s(Atr18, df = k)       3 2.4578e-09      1
s(Atr19, df = k)       3 5.7110e-10      1
s(Atr20, df = k)       3 1.5749e-09      1
s(Atr21, df = k)       3 4.0171e-09      1
s(Atr22, df = k)       3 6.2650e-10      1
s(Atr23, df = k)       3 1.3182e-09      1
s(Atr24, df = k)       3 3.8708e-09      1
s(Atr25, df = k)       3 4.2460e-10      1
s(Atr26, df = k)       3 3.9860e-09      1
s(Atr27, df = k)       3 9.9800e-11      1
s(Atr28, df = k)       3 2.7089e-09      1
s(Atr29, df = k)       3 5.9623e-09      1
s(Atr30, df = k)       3 1.3760e-10      1
s(Atr31, df = k)       3 3.5530e-09      1
s(Atr32, df = k)       3 5.4540e-10      1
s(Atr33, df = k)       3 1.6051e-09      1
s(Atr34, df = k)       3 5.1030e-10      1
s(Atr35, df = k)       3 6.3010e-10      1
```

```
s(Atr36, df = k)        3 2.7341e-09       1
s(Atr37, df = k)        3 4.1581e-09       1
s(Atr38, df = k)        3 3.6130e-10       1
s(Atr39, df = k)        3 7.3450e-10       1
s(Atr40, df = k)        3 3.4084e-09       1
s(Atr41, df = k)        3 4.6420e-10       1
s(Atr42, df = k)        3 4.9250e-10       1
s(Atr43, df = k)        3 9.7910e-10       1
s(Atr44, df = k)        3 1.2413e-09       1
s(Atr45, df = k)        3 2.8395e-09       1
s(Atr46, df = k)        3 2.4580e-10       1
s(Atr47, df = k)        3 3.4914e-09       1
s(Atr48, df = k)        3 9.4170e-10       1
s(Atr49, df = k)        3 2.5830e-10       1
s(Atr50, df = k)        3 5.0480e-10       1
s(Atr51, df = k)        3 4.3760e-10       1
s(Atr52, df = k)        3 3.8671e-09       1
s(Atr53, df = k)        3 1.9148e-09       1
s(Atr54, df = k)        3 2.6122e-09       1
Warning messages:
1: In pf(f, df, dfr, lower.tail = FALSE) : NaNs produce
d
2: In summary.Gam(gam.fit) :
  Residual degrees of freedom are negative or zero.  Th
is occurs when the sum of the parametric and nonparamet
ric degrees of freedom exceeds the number of observatio
ns.  The model is probably too complex for the amount o
f data available.
> gam.error=rep(0,5)
> # ************** GAM with Smoothing Splines with degr
ee of freedom=5 ***************
> for (j in 1:5){
+   set.seed(1)
+   train = divorce[folds!=j,]
+   test = divorce[folds==j,]
+   class.test = Class[folds==j]
+
+   gam.fit=gam(fmla2,family=binomial,data=train)
+   gam.pred=predict(gam.fit,test,type='response')
+   conf_gam= table(gam.pred>.5,class.test)
+
+   print(conf_gam)
+
+   #Apply it to the test data and get the confusion ma
trix and error rate.
+   gam.error[j]=1-sum(diag(conf_gam))/sum(conf_gam)
+ }
       class.test
         0  1
  FALSE 20  2
  TRUE   0 17
       class.test
         0  1
  FALSE 13  1
  TRUE   0 20
       class.test
```

```
         0  1
  FALSE 18  1
  TRUE   0 12
       class.test
         0  1
  FALSE 18  0
  TRUE   0 15
       class.test
         0  1
  FALSE 17  1
  TRUE   0 15
Warning messages:
1: In model.matrix.default(mt, mf, contrasts) :
  non-list contrasts argument ignored
2: In gam(fmla2, family = binomial, data = train) :
  Residual degrees of freedom are negative or zero.  Th
is occurs when the sum of the parametric and nonparamet
ric degrees of freedom exceeds the number of observatio
ns.  The model is probably too complex for the amount o
f data available.
3: In model.matrix.default(mt, mf, contrasts) :
  non-list contrasts argument ignored
4: In gam(fmla2, family = binomial, data = train) :
  Residual degrees of freedom are negative or zero.  Th
is occurs when the sum of the parametric and nonparamet
ric degrees of freedom exceeds the number of observatio
ns.  The model is probably too complex for the amount o
f data available.
5: In model.matrix.default(mt, mf, contrasts) :
  non-list contrasts argument ignored
6: In gam(fmla2, family = binomial, data = train) :
  Residual degrees of freedom are negative or zero.  Th
is occurs when the sum of the parametric and nonparamet
ric degrees of freedom exceeds the number of observatio
ns.  The model is probably too complex for the amount o
f data available.
7: In model.matrix.default(mt, mf, contrasts) :
  non-list contrasts argument ignored
8: In gam(fmla2, family = binomial, data = train) :
  Residual degrees of freedom are negative or zero.  Th
is occurs when the sum of the parametric and nonparamet
ric degrees of freedom exceeds the number of observatio
ns.  The model is probably too complex for the amount o
f data available.
9: In model.matrix.default(mt, mf, contrasts) :
  non-list contrasts argument ignored
10: In gam(fmla2, family = binomial, data = train) :
  Residual degrees of freedom are negative or zero.  Th
is occurs when the sum of the parametric and nonparamet
ric degrees of freedom exceeds the number of observatio
ns.  The model is probably too complex for the amount o
f data available.
> gam.error
[1] 0.05128205 0.02941176 0.03225806 0.00000000
[5] 0.03030303
> mean(gam.error)
```

```
[1] 0.02865098
> # **************** GAM with natural splines with degr
ee of freedom = 5 **************
> library(splines)
> newx_ns=paste("ns(Atr",1:54,',df=k)',sep="")
> fmla_ns=as.formula(paste('Class~',paste(newx_ns, coll
apse="+")))
> fmla_ns
Class ~ ns(Atr1, df = k) + ns(Atr2, df = k) + ns(Atr3,
df = k) +
    ns(Atr4, df = k) + ns(Atr5, df = k) + ns(Atr6, df =
k) +
    ns(Atr7, df = k) + ns(Atr8, df = k) + ns(Atr9, df =
k) +
    ns(Atr10, df = k) + ns(Atr11, df = k) + ns(Atr12, d
f = k) +
    ns(Atr13, df = k) + ns(Atr14, df = k) + ns(Atr15, d
f = k) +
    ns(Atr16, df = k) + ns(Atr17, df = k) + ns(Atr18, d
f = k) +
    ns(Atr19, df = k) + ns(Atr20, df = k) + ns(Atr21, d
f = k) +
    ns(Atr22, df = k) + ns(Atr23, df = k) + ns(Atr24, d
f = k) +
    ns(Atr25, df = k) + ns(Atr26, df = k) + ns(Atr27, d
f = k) +
    ns(Atr28, df = k) + ns(Atr29, df = k) + ns(Atr30, d
f = k) +
    ns(Atr31, df = k) + ns(Atr32, df = k) + ns(Atr33, d
f = k) +
    ns(Atr34, df = k) + ns(Atr35, df = k) + ns(Atr36, d
f = k) +
    ns(Atr37, df = k) + ns(Atr38, df = k) + ns(Atr39, d
f = k) +
    ns(Atr40, df = k) + ns(Atr41, df = k) + ns(Atr42, d
f = k) +
    ns(Atr43, df = k) + ns(Atr44, df = k) + ns(Atr45, d
f = k) +
    ns(Atr46, df = k) + ns(Atr47, df = k) + ns(Atr48, d
f = k) +
    ns(Atr49, df = k) + ns(Atr50, df = k) + ns(Atr51, d
f = k) +
    ns(Atr52, df = k) + ns(Atr53, df = k) + ns(Atr54, d
f = k)
> ## Code for best subset selection within k-fold for l
inear regression ##
> set.seed(1)

>

> val <- 8 #max preds for bestsubset
> best.lin.cv.errors = matrix (NA,k,val, dimnames =list
(NULL , paste (1:val) ))
> #install.packages("leaps")
> library(leaps)
> # predict function for best subset selection #
```

```
> predict.regsubsets = function(object,newdata,id,...){
+   form=as.formula (object$call [[2]])
+   mat=model.matrix(form ,newdata )
+   coefi=coef(object ,id=id)
+   xvars=names(coefi)
+   mat[,xvars]%*%coefi
+ }
> for(j in 1:k){
+   set.seed(1)
+   train = divorce[folds!=j,]
+   test = divorce[folds ==j,]
+
+   best.fit=regsubsets(Class~.,data=train,nvmax=val,re
ally.big=T)
+   for(i in 1:val){
+     pred=predict(best.fit,test,id=i)
+     for(m in 1:length(pred)){
+       if(pred[m]>=0.5){
+         pred[m]=1
+       } else{
+         pred[m]=0
+       }
+     }
+     best.lin.cv.errors[j,i]= mean((test$Class-pred)^2
)
+   }
+ }
> best.lin.cv.errors
              1          2          3          4
[1,] 0.05128205 0.05128205 0.02564103 0.02564103
[2,] 0.05882353 0.02941176 0.00000000 0.00000000
[3,] 0.03225806 0.03225806 0.00000000 0.03225806
[4,] 0.03030303 0.00000000 0.00000000 0.00000000
[5,] 0.06060606 0.03030303 0.03030303 0.03030303
              5          6          7          8
[1,] 0.02564103 0.02564103 0.02564103 0.02564103
[2,] 0.02941176 0.00000000 0.02941176 0.00000000
[3,] 0.03225806 0.03225806 0.03225806 0.03225806
[4,] 0.00000000 0.00000000 0.00000000 0.00000000
[5,] 0.03030303 0.03030303 0.03030303 0.03030303
> mean.best.lin.cv.errors=apply(best.lin.cv.errors ,2,
mean)
> mean.best.lin.cv.errors
         1          2          3          4
0.04665455 0.02865098 0.01118881 0.01764042
         5          6          7          8
0.02352278 0.01764042 0.02352278 0.01764042
> x = which.min(mean.best.lin.cv.errors)
> reg.best=regsubsets(Class~.,data=divorce , nvmax=val,
really.big=T)
> coef(reg.best,x)
(Intercept)        Atr6       Atr18       Atr40
-0.04022481  0.07411876  0.14921847  0.13498984
> ## Code for forward selection within k-fold for linea
r regression ##
> set.seed(1)
```

```
> val <- 54 #max preds for bestsubset
> fwd.lin.cv.errors = matrix (NA,k,val, dimnames =list(
NULL , paste (1:val) ))
> #install.packages("leaps")
> library(leaps)
> for(j in 1:k){
+   set.seed(1)
+   train = divorce[folds!=j,]
+   test = divorce[folds ==j,]
+
+   regfit.fwd.fit=regsubsets(Class~.,data=train,nvmax=
val,method="forward")
+   for(i in 1:val){
+     pred=predict(regfit.fwd.fit,test,id=i)
+
+     for(m in 1:length(pred)){
+       if(pred[m]>=0.5){
+         pred[m]=1.0
+       } else{
+         pred[m]=0.0
+       }
+     }
+
+     fwd.lin.cv.errors[j,i]= mean((test$Class-pred)^2)
+   }
+ }
> fwd.lin.cv.errors
              1          2          3          4
[1,] 0.05128205 0.05128205 0.05128205 0.02564103
[2,] 0.05882353 0.02941176 0.00000000 0.00000000
[3,] 0.03225806 0.03225806 0.03225806 0.03225806
[4,] 0.03030303 0.00000000 0.00000000 0.00000000
[5,] 0.06060606 0.03030303 0.03030303 0.03030303
              5          6          7          8
[1,] 0.02564103 0.02564103 0.02564103 0.02564103
[2,] 0.02941176 0.00000000 0.02941176 0.02941176
[3,] 0.03225806 0.03225806 0.03225806 0.03225806
[4,] 0.00000000 0.00000000 0.00000000 0.00000000
[5,] 0.03030303 0.03030303 0.03030303 0.03030303
              9         10         11         12
[1,] 0.02564103 0.02564103 0.02564103 0.02564103
[2,] 0.02941176 0.02941176 0.00000000 0.00000000
[3,] 0.03225806 0.03225806 0.03225806 0.03225806
[4,] 0.00000000 0.00000000 0.00000000 0.00000000
[5,] 0.03030303 0.03030303 0.03030303 0.03030303
             13         14         15         16
[1,] 0.02564103 0.02564103 0.02564103 0.02564103
[2,] 0.00000000 0.00000000 0.00000000 0.00000000
[3,] 0.03225806 0.03225806 0.03225806 0.03225806
[4,] 0.00000000 0.00000000 0.00000000 0.00000000
[5,] 0.03030303 0.03030303 0.03030303 0.03030303
             17         18         19         20
[1,] 0.02564103 0.02564103 0.02564103 0.02564103
[2,] 0.00000000 0.00000000 0.00000000 0.00000000
[3,] 0.03225806 0.03225806 0.03225806 0.03225806
[4,] 0.00000000 0.00000000 0.00000000 0.00000000
```

```
[5,] 0.03030303 0.03030303 0.03030303 0.03030303
             21         22         23         24
[1,] 0.02564103 0.02564103 0.02564103 0.02564103
[2,] 0.00000000 0.00000000 0.00000000 0.00000000
[3,] 0.03225806 0.03225806 0.03225806 0.03225806
[4,] 0.00000000 0.00000000 0.00000000 0.00000000
[5,] 0.03030303 0.03030303 0.03030303 0.03030303
             25         26         27         28
[1,] 0.02564103 0.02564103 0.02564103 0.02564103
[2,] 0.00000000 0.00000000 0.00000000 0.00000000
[3,] 0.03225806 0.03225806 0.03225806 0.03225806
[4,] 0.00000000 0.00000000 0.00000000 0.00000000
[5,] 0.03030303 0.03030303 0.03030303 0.03030303
             29         30         31         32
[1,] 0.02564103 0.02564103 0.02564103 0.02564103
[2,] 0.00000000 0.00000000 0.00000000 0.00000000
[3,] 0.03225806 0.03225806 0.03225806 0.03225806
[4,] 0.00000000 0.00000000 0.00000000 0.00000000
[5,] 0.03030303 0.03030303 0.03030303 0.03030303
             33         34         35         36
[1,] 0.02564103 0.02564103 0.02564103 0.05128205
[2,] 0.00000000 0.00000000 0.00000000 0.00000000
[3,] 0.03225806 0.03225806 0.03225806 0.03225806
[4,] 0.00000000 0.00000000 0.00000000 0.00000000
[5,] 0.03030303 0.03030303 0.03030303 0.03030303
             37         38         39         40
[1,] 0.02564103 0.05128205 0.05128205 0.05128205
[2,] 0.00000000 0.00000000 0.00000000 0.00000000
[3,] 0.03225806 0.03225806 0.03225806 0.03225806
[4,] 0.00000000 0.00000000 0.00000000 0.00000000
[5,] 0.03030303 0.03030303 0.03030303 0.03030303
             41         42         43         44
[1,] 0.05128205 0.05128205 0.05128205 0.05128205
[2,] 0.00000000 0.00000000 0.00000000 0.00000000
[3,] 0.03225806 0.03225806 0.03225806 0.03225806
[4,] 0.00000000 0.00000000 0.00000000 0.00000000
[5,] 0.03030303 0.03030303 0.03030303 0.03030303
             45         46         47         48
[1,] 0.05128205 0.05128205 0.05128205 0.05128205
[2,] 0.00000000 0.00000000 0.00000000 0.00000000
[3,] 0.03225806 0.03225806 0.03225806 0.03225806
[4,] 0.00000000 0.00000000 0.00000000 0.00000000
[5,] 0.03030303 0.03030303 0.03030303 0.03030303
             49         50         51         52
[1,] 0.05128205 0.05128205 0.05128205 0.05128205
[2,] 0.00000000 0.00000000 0.00000000 0.00000000
[3,] 0.03225806 0.03225806 0.03225806 0.03225806
[4,] 0.00000000 0.00000000 0.00000000 0.00000000
[5,] 0.03030303 0.03030303 0.03030303 0.03030303
             53         54
[1,] 0.05128205 0.05128205
[2,] 0.00000000 0.00000000
[3,] 0.03225806 0.03225806
[4,] 0.00000000 0.00000000
[5,] 0.03030303 0.03030303
```

```
> mean.fwd.lin.cv.errors=apply(fwd.lin.cv.errors ,2, me
an)
> mean.fwd.lin.cv.errors
         1          2          3          4 
0.04665455 0.02865098 0.02276863 0.01764042 
         5          6          7          8 
0.02352278 0.01764042 0.02352278 0.02352278 
         9         10         11         12 
0.02352278 0.02352278 0.01764042 0.01764042 
        13         14         15         16 
0.01764042 0.01764042 0.01764042 0.01764042 
        17         18         19         20 
0.01764042 0.01764042 0.01764042 0.01764042 
        21         22         23         24 
0.01764042 0.01764042 0.01764042 0.01764042 
        25         26         27         28 
0.01764042 0.01764042 0.01764042 0.01764042 
        29         30         31         32 
0.01764042 0.01764042 0.01764042 0.01764042 
        33         34         35         36 
0.01764042 0.01764042 0.01764042 0.02276863 
        37         38         39         40 
0.01764042 0.02276863 0.02276863 0.02276863 
        41         42         43         44 
0.02276863 0.02276863 0.02276863 0.02276863 
        45         46         47         48 
0.02276863 0.02276863 0.02276863 0.02276863 
        49         50         51         52 
0.02276863 0.02276863 0.02276863 0.02276863 
        53         54 
0.02276863 0.02276863 
> mean(mean.fwd.lin.cv.errors)
[1] 0.02073065
> x = which.min(mean.fwd.lin.cv.errors)
> reg.best=regsubsets(Class~.,data=divorce , nvmax=val,
really.big=T,method="forward")
> coef(reg.best,x)
(Intercept)        Atr6       Atr18       Atr29 
-0.03992621  0.07017789  0.12168790  0.04892915 
      Atr40 
 0.11965844 
> ## Code for backward selection within k-fold for line
ar regression ##
> set.seed(1)
> val <- 54 #max preds for bestsubset
> bwd.lin.cv.errors = matrix (NA,k,val, dimnames =list(
NULL , paste (1:val) ))
> #install.packages("leaps")
> library(leaps)
> for(j in 1:k){
+   set.seed(1)
+   train = divorce[folds!=j,]
+   test = divorce[folds ==j,]
+ 
+   regfit.fwd.fit=regsubsets(Class~.,data=train,nvmax=
val,method="backward")
```

```
+   for(i in 1:val){
+     pred=predict(regfit.fwd.fit,test,id=i)
+     for(m in 1:length(pred)){
+       if(pred[m]>=0.5){
+         pred[m]=1
+       } else{
+         pred[m]=0
+       }
+     }
+     bwd.lin.cv.errors[j,i]= mean((test$Class-pred)^2)
+   }
+ }
> bwd.lin.cv.errors
              1          2          3          4
[1,] 0.05128205 0.05128205 0.05128205 0.05128205
[2,] 0.05882353 0.02941176 0.00000000 0.00000000
[3,] 0.00000000 0.03225806 0.03225806 0.03225806
[4,] 0.03030303 0.00000000 0.00000000 0.00000000
[5,] 0.06060606 0.03030303 0.03030303 0.03030303
              5          6          7          8
[1,] 0.05128205 0.05128205 0.05128205 0.05128205
[2,] 0.00000000 0.00000000 0.00000000 0.00000000
[3,] 0.03225806 0.03225806 0.03225806 0.03225806
[4,] 0.00000000 0.00000000 0.00000000 0.00000000
[5,] 0.03030303 0.03030303 0.03030303 0.03030303
              9         10         11         12
[1,] 0.05128205 0.05128205 0.05128205 0.05128205
[2,] 0.00000000 0.00000000 0.00000000 0.00000000
[3,] 0.03225806 0.03225806 0.03225806 0.03225806
[4,] 0.00000000 0.00000000 0.00000000 0.00000000
[5,] 0.03030303 0.03030303 0.03030303 0.03030303
             13         14         15         16
[1,] 0.05128205 0.05128205 0.05128205 0.05128205
[2,] 0.00000000 0.00000000 0.00000000 0.00000000
[3,] 0.03225806 0.03225806 0.03225806 0.03225806
[4,] 0.00000000 0.00000000 0.00000000 0.00000000
[5,] 0.03030303 0.03030303 0.03030303 0.03030303
             17         18         19         20
[1,] 0.05128205 0.05128205 0.05128205 0.05128205
[2,] 0.00000000 0.00000000 0.00000000 0.00000000
[3,] 0.03225806 0.03225806 0.03225806 0.03225806
[4,] 0.00000000 0.00000000 0.00000000 0.00000000
[5,] 0.03030303 0.03030303 0.03030303 0.03030303
             21         22         23         24
[1,] 0.05128205 0.05128205 0.05128205 0.05128205
[2,] 0.00000000 0.00000000 0.00000000 0.00000000
[3,] 0.03225806 0.03225806 0.03225806 0.03225806
[4,] 0.00000000 0.00000000 0.00000000 0.00000000
[5,] 0.03030303 0.03030303 0.03030303 0.03030303
             25         26         27         28
[1,] 0.05128205 0.05128205 0.05128205 0.05128205
[2,] 0.00000000 0.00000000 0.00000000 0.00000000
[3,] 0.03225806 0.03225806 0.03225806 0.03225806
[4,] 0.00000000 0.00000000 0.00000000 0.00000000
[5,] 0.03030303 0.03030303 0.03030303 0.03030303
             29         30         31         32
```

```
[1,] 0.05128205 0.05128205 0.05128205 0.05128205
[2,] 0.00000000 0.00000000 0.00000000 0.00000000
[3,] 0.03225806 0.03225806 0.03225806 0.03225806
[4,] 0.00000000 0.00000000 0.00000000 0.00000000
[5,] 0.03030303 0.03030303 0.03030303 0.03030303
             33         34         35         36
[1,] 0.05128205 0.05128205 0.05128205 0.05128205
[2,] 0.00000000 0.00000000 0.00000000 0.00000000
[3,] 0.03225806 0.03225806 0.03225806 0.03225806
[4,] 0.00000000 0.00000000 0.00000000 0.00000000
[5,] 0.03030303 0.03030303 0.03030303 0.03030303
             37         38         39         40
[1,] 0.05128205 0.05128205 0.05128205 0.05128205
[2,] 0.00000000 0.00000000 0.00000000 0.00000000
[3,] 0.03225806 0.03225806 0.03225806 0.03225806
[4,] 0.00000000 0.00000000 0.00000000 0.00000000
[5,] 0.03030303 0.03030303 0.03030303 0.03030303
             41         42         43         44
[1,] 0.05128205 0.05128205 0.05128205 0.05128205
[2,] 0.00000000 0.00000000 0.00000000 0.00000000
[3,] 0.03225806 0.03225806 0.03225806 0.03225806
[4,] 0.00000000 0.00000000 0.00000000 0.00000000
[5,] 0.03030303 0.03030303 0.03030303 0.03030303
             45         46         47         48
[1,] 0.05128205 0.05128205 0.05128205 0.05128205
[2,] 0.00000000 0.00000000 0.00000000 0.00000000
[3,] 0.03225806 0.03225806 0.03225806 0.03225806
[4,] 0.00000000 0.00000000 0.00000000 0.00000000
[5,] 0.03030303 0.03030303 0.03030303 0.03030303
             49         50         51         52
[1,] 0.05128205 0.05128205 0.05128205 0.05128205
[2,] 0.00000000 0.00000000 0.00000000 0.00000000
[3,] 0.03225806 0.03225806 0.03225806 0.03225806
[4,] 0.00000000 0.00000000 0.00000000 0.00000000
[5,] 0.03030303 0.03030303 0.03030303 0.03030303
             53         54
[1,] 0.05128205 0.05128205
[2,] 0.00000000 0.00000000
[3,] 0.03225806 0.03225806
[4,] 0.00000000 0.00000000
[5,] 0.03030303 0.03030303
> mean.bwd.lin.cv.errors=apply(bwd.lin.cv.errors ,2, me
an)
> mean.bwd.lin.cv.errors
         1          2          3          4 
0.04020293 0.02865098 0.02276863 0.02276863 
         5          6          7          8 
0.02276863 0.02276863 0.02276863 0.02276863 
         9         10         11         12 
0.02276863 0.02276863 0.02276863 0.02276863 
        13         14         15         16 
0.02276863 0.02276863 0.02276863 0.02276863 
        17         18         19         20 
0.02276863 0.02276863 0.02276863 0.02276863 
        21         22         23         24 
0.02276863 0.02276863 0.02276863 0.02276863 
```

```
        25         26         27         28
0.02276863 0.02276863 0.02276863 0.02276863
        29         30         31         32
0.02276863 0.02276863 0.02276863 0.02276863
        33         34         35         36
0.02276863 0.02276863 0.02276863 0.02276863
        37         38         39         40
0.02276863 0.02276863 0.02276863 0.02276863
        41         42         43         44
0.02276863 0.02276863 0.02276863 0.02276863
        45         46         47         48
0.02276863 0.02276863 0.02276863 0.02276863
        49         50         51         52
0.02276863 0.02276863 0.02276863 0.02276863
        53         54
0.02276863 0.02276863
> mean(mean.bwd.lin.cv.errors)
[1] 0.02320042
> x = which.min(mean.bwd.lin.cv.errors)
> reg.best=regsubsets(Class~.,data=divorce , nvmax=val,
really.big=T,method="backward")
> coef(reg.best,x)
(Intercept)       Atr17       Atr26       Atr40
-0.02461195  0.10758431  0.05624488  0.13749335

>

> ## Code for lasso with linear ##
> library(glmnet)
> lasso.lin.cv.errors = rep(NA,k)
> for(j in 1:k){
+   set.seed(1)
+   train = divorce[folds!=j,]
+   test = divorce[folds ==j,]
+
+   x=model.matrix(Class~.,train)[,-1]  #remove the int
ercept (training)
+   y=train[,dim(divorce)[2]] #response Class (training
)
+   cv.out=cv.glmnet(x,y,alpha=1)
+   bestlam=cv.out$lambda.min #get optimal tuning param
eter (lambda)
+
+   lasso.mod=glmnet(x,y,alpha=1,lambda=bestlam)
+   newx = model.matrix(Class~.,test )[,-1] #remove the
intercept (test)
+   newy = test[,dim(divorce)[2]] #response Class (test
)
+   pred.lasso = predict(lasso.mod, s = bestlam, newx =
newx) #predict
+
+   for(m in 1:length(pred.lasso)){
+     if(pred.lasso[m]>=0.5){
+       pred.lasso[m]=1
+     } else{
+       pred.lasso[m]=0
```

```
+     }
+   }
+
+   #find MSE of lasso on this kth fold
+   error = mean((newy - pred.lasso)^2)
+   error
+
+   #append
+   lasso.lin.cv.errors[j] = error
+
+   #to see the coefs of the kth-fold's lowest MSE
+   lasso.coef=coef(lasso.mod)[,1]
+   print(lasso.coef[lasso.coef!=0])
+
+
+ }
  (Intercept)          Atr2          Atr4
-6.398876e-02  2.944569e-05  3.606098e-03
         Atr6          Atr7         Atr12
 5.682331e-02  2.682467e-02  2.666280e-03
        Atr15         Atr17         Atr18
 4.045280e-02  3.960235e-02  4.058390e-02
        Atr19         Atr26         Atr28
 2.600655e-02  3.248353e-02  1.552487e-02
        Atr31         Atr33         Atr36
 7.258096e-03  5.864019e-03  7.877197e-03
        Atr38         Atr40         Atr46
 1.350457e-02  5.623974e-02 -3.187309e-03
        Atr49         Atr52
 9.024248e-03  1.183193e-03
 (Intercept)         Atr3         Atr6        Atr11
-0.053560046  0.010992800  0.046967209  0.014623062
       Atr17        Atr18        Atr26        Atr28
 0.023043379  0.059092640  0.027553308  0.024959684
       Atr31        Atr40        Atr49        Atr52
 0.002165845  0.120198830  0.011739784  0.003335036
 (Intercept)         Atr1         Atr2         Atr3
-0.090281679  0.010927643  0.009827424  0.016670085
        Atr6         Atr7         Atr9        Atr11
 0.048197398  0.029782146  0.014645164  0.007879829
       Atr18        Atr19        Atr28        Atr29
 0.057756373  0.022514579  0.018123875  0.014030075
       Atr32        Atr39        Atr40        Atr49
 0.002612150  0.022668912  0.077307020  0.009077201
       Atr52
 0.016467939
  (Intercept)          Atr2          Atr3
-0.0700811666  0.0054913956  0.0103762518
         Atr6          Atr7         Atr11
 0.0605669369  0.0099366964  0.0093607290
        Atr15         Atr17         Atr18
 0.0263838139  0.0316957435  0.0547311309
        Atr26         Atr28         Atr40
 0.0236364301  0.0076000893  0.1015792432
        Atr46         Atr48         Atr49
-0.0079609062 -0.0006996802  0.0189200665
```

```
        Atr52
 0.0164566133
 (Intercept)         Atr3         Atr6         Atr7
-0.069396665  0.010130010  0.047650860  0.015954611
        Atr9        Atr11        Atr17        Atr18
 0.017736409  0.032499411  0.020232770  0.048387391
       Atr26        Atr30        Atr40        Atr41
 0.017096396  0.004577569  0.087597236  0.022358047
       Atr44        Atr49        Atr52
 0.019212873  0.005078399  0.009978129
> lasso.lin.cv.errors
[1] 0.05128205 0.00000000 0.03225806 0.00000000
[5] 0.03030303
> ## Code for lasso with logistic##
> lasso.log.cv.errors = rep(NA,k)
> for(j in 1:k){
+   set.seed(1)
+   train = divorce[folds!=j,]
+   test = divorce[folds ==j,]
+
+   x=model.matrix(Class~.,train)[,-1]  #remove the int
ercept (training)
+   y=train[,dim(divorce)[2]] #response Class (training
)
+   cv.out=cv.glmnet(x,y,alpha=1)
+   bestlam=cv.out$lambda.min #get optimal tuning param
eter (lambda)
+
+   lasso.mod=glmnet(x,y,alpha=1,lambda=bestlam, family
="binomial")
+   newx = model.matrix(Class~.,test)[,-1] #remove the
intercept (test)
+   newy = test[,dim(divorce)[2]] #response Class (test
)
+   pred.lasso = predict(lasso.mod, s = bestlam, newx =
newx, type = "response") #predict
+   pred.lasso[pred.lasso>=.5]=1
+   pred.lasso[pred.lasso<.5]=0
+   print(pred.lasso)
+   print(newy)
+
+   #find MSE of lasso on this kth fold
+   error = mean((newy != pred.lasso)^2)
+   error
+
+   #append
+   lasso.log.cv.errors[j] = error
+
+   #to see the coefs of the kth-fold's lowest MSE
+   lasso.coef=coef(lasso.mod)[,1]
+   print(lasso.coef[lasso.coef!=0])
+
+
+ }
    1
1   0
```

```
3   1
10  0
15  1
18  1
19  1
24  1
26  1
36  1
37  1
39  1
46  1
51  1
54  1
55  1
63  1
66  1
67  1
78  1
89  0
94  0
98  0
101 0
107 0
108 0
111 0
115 0
121 0
123 0
131 0
139 0
146 0
154 0
156 0
157 0
160 0
165 0
166 0
169 0
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
[25] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
(Intercept)        Atr6       Atr15       Atr17
 -5.8557004   0.5374038   0.6329337   0.2120006
      Atr19       Atr26       Atr31       Atr40
  0.7012349   1.0152343   0.1373878   0.7805287
      Atr49
  0.1622556
    1
4   1
7   1
13  1
14  1
22  1
23  1
29  1
30  1
34  1
```

```
40  1
42  1
43  1
45  1
69  1
70  1
72  1
76  1
77  1
81  1
82  1
84  1
91  0
110 0
112 0
120 0
125 0
127 0
128 0
129 0
132 0
140 0
142 0
143 0
153 0
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
[25] 0 0 0 0 0 0 0 0 0 0
(Intercept)        Atr3        Atr6       Atr18
 -5.1470283   0.1582206   0.4146123   0.3008922
      Atr26       Atr28       Atr40       Atr49
  0.6989765   0.1289404   1.4401921   0.2986299
    1
6   0
8   1
9   1
28  1
41  1
47  1
48  1
50  1
68  1
71  1
74  1
79  1
80  1
90  0
92  0
93  0
102 0
103 0
104 0
105 0
113 0
116 0
118 0
130 0
```

```
133 0
134 0
135 0
136 0
138 0
155 0
161 0
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
[25] 0 0 0 0 0 0 0
(Intercept)        Atr1        Atr2        Atr3 
 -5.7155865   0.1406552   0.1078151   0.2189390 
       Atr6       Atr18       Atr19       Atr28 
  0.4251436   0.3822256   0.4000094   0.2659912 
      Atr39       Atr40       Atr49       Atr52 
  0.3082291   0.9439608   0.2677671   0.1953335 
    1
2   1
25  1
27  1
31  1
32  1
33  1
35  1
38  1
49  1
52  1
56  1
59  1
61  1
62  1
73  1
86  0
88  0
96  0
97  0
106 0
109 0
114 0
119 0
122 0
124 0
141 0
144 0
150 0
151 0
152 0
158 0
162 0
164 0
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
[25] 0 0 0 0 0 0 0 0 0
(Intercept)        Atr3        Atr6       Atr15 
-6.18616911  0.27657522  0.64870110  0.05288914 
      Atr18       Atr19       Atr26       Atr40 
 0.28630154  0.10886982  0.85841547  1.46415897 
      Atr49       Atr52 
```

```
 0.46538645  0.13650625
    1
5   0
11  1
12  1
16  1
17  1
20  1
21  1
44  1
53  1
57  1
58  1
60  1
64  1
65  1
75  1
83  1
85  0
87  0
95  0
99  0
100 0
117 0
126 0
137 0
145 0
147 0
148 0
149 0
159 0
163 0
167 0
168 0
170 0
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
[25] 0 0 0 0 0 0 0 0 0
(Intercept)        Atr3        Atr6       Atr11
-5.36235545  0.30329654  0.37871943  0.29916399
      Atr17       Atr18       Atr26       Atr40
 0.24353393  0.24338547  0.34377845  1.20978857
      Atr44       Atr52
 0.42292508  0.08235102
> lasso.log.cv.errors
[1] 0.05128205 0.00000000 0.03225806 0.00000000
[5] 0.03030303
> # Linear regression with all 54 parameters - RSS
> mean(lin.mod.mse_vector)
[1] 0.03678986
> # Linear regression with all 54 parameters - MSE / Cl
ass Error
> mean(lin.mod.class.mse_vector)
[1] 0.02276863
> # Polynomial regression (exponent of 2) with all 54 p
arameters - RSS
> mean(log.mod.mse_vector)
```

```
[1] 0.01566936
> # Polynomial regression (exponent of 2) with all 54 p
arameters - MSE / Class Error
> mean(log.mod.class.mse_vector)
[1] 0.01764042
> # Polynomial logistic regression (exponent of 2) with
all 54 parameters
> mean(log.mod.mse_vector1)
[1] 0.01566936
> # LDA with all 54 parameters
> mean(lda.error)
[1] 0.02276863
> # KNN with all 54 parameters (CV for value of K in 5-
fold)
> mean(knn.error)
[1] 0.02276863
> # Decision tree with all 54 parameters
> mean(tree.error)
[1] 0.02882924
> # LDA with only top 5 correlated parameters
> mean(lda5.error)
[1] 0.02276863
> # QDA with only top 5 correlated parameters
> mean(qda5.error)
[1] 0.01764042
> # KNN with only top 5 correlated parameters (CV for v
alue of k in 5-fold)
> mean(knn5.error)
[1] 0.3243866
> # Decision tree with only top 5 correlated parameters
> mean(tree5.error)
[1] 0.03030303
> # Logistic regression with all 54 parameters
> mean(cv.logi_error1.5)
[1] 0.0387848
> # Logistic regression (with control) with all 54 para
meters
> mean(cv.logiWC_error1.5)
[1] 0.03148009
> # GAM w/ Smoothing Splines and 5 df
> mean(gam.error)
[1] 0.02865098
> # Linear regression with best subset selection
> mean(mean.best.lin.cv.errors)
[1] 0.02330765
> # Linear regression with forward selection
> mean(mean.fwd.lin.cv.errors)
[1] 0.02073065
> # Linear regression with backward selection
> mean(mean.bwd.lin.cv.errors)
[1] 0.02320042
> # Linear regression with lasso
> mean(lasso.lin.cv.errors)
[1] 0.02276863
> # Logistic regression with lasso
> mean(lasso.log.cv.errors)
```

```
[1] 0.02276863
> divorce_test <- read.csv('C:/Users/14014/Documents/Co
rnell_Fall_2019/STSCI_4740/STSCI_4740_FinalProject/divo
rce_fun_test.csv', header=TRUE)
> attach(divorce_test)
The following objects are masked from divorce (pos = 4)
:

    Atr1, Atr10, Atr11, Atr12, Atr13, Atr14,
    Atr15, Atr16, Atr17, Atr18, Atr19, Atr2,
    Atr20, Atr21, Atr22, Atr23, Atr24, Atr25,
    Atr26, Atr27, Atr28, Atr29, Atr3, Atr30,
    Atr31, Atr32, Atr33, Atr34, Atr35, Atr36,
    Atr37, Atr38, Atr39, Atr4, Atr40, Atr41,
    Atr42, Atr43, Atr44, Atr45, Atr46, Atr47,
    Atr48, Atr49, Atr5, Atr50, Atr51, Atr52,
    Atr53, Atr54, Atr6, Atr7, Atr8, Atr9

The following objects are masked from divorce (pos = 5)
:

    Atr1, Atr10, Atr11, Atr12, Atr13, Atr14,
    Atr15, Atr16, Atr17, Atr18, Atr19, Atr2,
    Atr20, Atr21, Atr22, Atr23, Atr24, Atr25,
    Atr26, Atr27, Atr28, Atr29, Atr3, Atr30,
    Atr31, Atr32, Atr33, Atr34, Atr35, Atr36,
    Atr37, Atr38, Atr39, Atr4, Atr40, Atr41,
    Atr42, Atr43, Atr44, Atr45, Atr46, Atr47,
    Atr48, Atr49, Atr5, Atr50, Atr51, Atr52,
    Atr53, Atr54, Atr6, Atr7, Atr8, Atr9

The following objects are masked from divorce (pos = 6)
:

    Atr1, Atr10, Atr11, Atr12, Atr13, Atr14,
    Atr15, Atr16, Atr17, Atr18, Atr19, Atr2,
    Atr20, Atr21, Atr22, Atr23, Atr24, Atr25,
    Atr26, Atr27, Atr28, Atr29, Atr3, Atr30,
    Atr31, Atr32, Atr33, Atr34, Atr35, Atr36,
    Atr37, Atr38, Atr39, Atr4, Atr40, Atr41,
    Atr42, Atr43, Atr44, Atr45, Atr46, Atr47,
    Atr48, Atr49, Atr5, Atr50, Atr51, Atr52,
    Atr53, Atr54, Atr6, Atr7, Atr8, Atr9

The following objects are masked from divorce (pos = 16
):

    Atr1, Atr10, Atr11, Atr12, Atr13, Atr14,
    Atr15, Atr16, Atr17, Atr18, Atr19, Atr2,
    Atr20, Atr21, Atr22, Atr23, Atr24, Atr25,
    Atr26, Atr27, Atr28, Atr29, Atr3, Atr30,
    Atr31, Atr32, Atr33, Atr34, Atr35, Atr36,
    Atr37, Atr38, Atr39, Atr4, Atr40, Atr41,
    Atr42, Atr43, Atr44, Atr45, Atr46, Atr47,
    Atr48, Atr49, Atr5, Atr50, Atr51, Atr52,
    Atr53, Atr54, Atr6, Atr7, Atr8, Atr9
```

```
> # With QDA and 5 parameters only
> for (j in 1:k){
+   set.seed(1)
+   train = divorce[folds!=j,]
+   test = divorce_test
+
+   qda.fit=qda(as.factor(Class)~Atr40+Atr17+Atr19+Atr1
8+Atr11, data=train)
+   qda.class=predict(qda.fit,test)$class
+   print(qda.class)
+ }
[1] 0 0
Levels: 0 1
[1] 0 0
Levels: 0 1
[1] 0 0
Levels: 0 1
[1] 0 0
Levels: 0 1
[1] 0 0
Levels: 0 1
> # With Decision Tree with only 5 parameters
> for (j in 1:k){
+   set.seed(1)
+   train = divorce[folds!=j,]
+   test = divorce_test
+
+   tree5.divorce=tree(as.factor(Class)~.,train)
+   tree5.pred=predict(tree5.divorce,test,type="class")
+   print(tree5.pred)
+ }
[1] 0 0
Levels: 0 1
[1] 0 0
Levels: 0 1
[1] 0 0
Levels: 0 1
[1] 0 0
Levels: 0 1
[1] 0 0
Levels: 0 1
>

>
```