knitting

Sankit Gupta

December 10, 2016

#reading and transforming the data  
df1= read.csv(file="QueryResults1.csv",header=TRUE)  
df2<- read.csv(file="QueryResults2.csv",header=TRUE)  
StackOverflow<- rbind(df1,df2)  
#str(StackOverflow)  
#names(StackOverflow)  
nrow(StackOverflow)

## [1] 92388

#replacing NA with 1001  
StackOverflow [is.na(StackOverflow)] <- 1001  
  
#removing column that contain text or are unrelated to this question  
StackOverflowdata<- StackOverflow[,-c(1, 2,3,12,14,15,22,23,25,26,27,28,29,30,31,32,33,34)]  
#str(StackOverflowdata)  
nrow(StackOverflowdata)

## [1] 92388

#to check uniqueness of data  
lapply(StackOverflowdata["RejectionReasonId"], unique)

## $RejectionReasonId  
## [1] 101 1001

StackOverflowdata$IsSpam= ifelse(StackOverflowdata$RejectionReasonId==101,1,0)  
#str(StackOverflowdata)  
#names(StackOverflowdata)  
StackOverflowdata<- StackOverflowdata[,-c(16)]  
lapply(StackOverflowdata["IsSpam"], unique)

## $IsSpam  
## [1] 1 0

#dividing data for training and testing purpose  
smp\_size<- floor(0.75\*nrow(StackOverflowdata))  
set.seed(123)  
train\_ind<- sample(seq\_len(nrow(StackOverflowdata)), size=smp\_size)  
training\_data<- StackOverflowdata[train\_ind,]  
testing\_data<- StackOverflowdata[-train\_ind,]  
testing\_y<- testing\_data$IsSpam  
nrow(training\_data)

## [1] 69291

nrow(testing\_data)

## [1] 23097

#str(training\_data)  
#names(training\_data)  
  
#MOdel 1 - GLM  
logistics\_model<-glm(IsSpam~.,data=training\_data,family="binomial")  
#to discover better model  
#step(logistics\_model, direction = "forward")  
  
logistics\_model = glm(IsSpam ~ PostTypeId + PostScore + post\_length +   
 owner\_reputation + owner\_profile\_summary + owner\_views +   
 owner\_upvotes + owner\_downvotes + editDurationAfterCreation +  
 q\_num\_tags + AnswerCount + CommentCount + has\_code +post\_views +   
 UserId, family = "binomial", data = training\_data)  
summary(logistics\_model)

##   
## Call:  
## glm(formula = IsSpam ~ PostTypeId + PostScore + post\_length +   
## owner\_reputation + owner\_profile\_summary + owner\_views +   
## owner\_upvotes + owner\_downvotes + editDurationAfterCreation +   
## q\_num\_tags + AnswerCount + CommentCount + has\_code + post\_views +   
## UserId, family = "binomial", data = training\_data)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.5776 -0.9496 0.2926 0.9326 2.9590   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 5.426e+01 1.069e+01 5.075 3.88e-07 \*\*\*  
## PostTypeId -5.416e+01 1.072e+01 -5.053 4.36e-07 \*\*\*  
## PostScore -4.833e-05 9.232e-06 -5.235 1.65e-07 \*\*\*  
## post\_length -4.418e-05 5.107e-06 -8.650 < 2e-16 \*\*\*  
## owner\_reputation -5.820e-07 1.700e-07 -3.424 0.000617 \*\*\*  
## owner\_profile\_summary 1.330e-01 7.807e-03 17.041 < 2e-16 \*\*\*  
## owner\_views -8.557e-07 2.489e-07 -3.438 0.000587 \*\*\*  
## owner\_upvotes 8.296e-05 5.829e-06 14.232 < 2e-16 \*\*\*  
## owner\_downvotes 1.254e-05 5.249e-06 2.388 0.016921 \*   
## editDurationAfterCreation -3.634e-07 8.343e-09 -43.561 < 2e-16 \*\*\*  
## q\_num\_tags 4.348e-02 1.085e-02 4.007 6.15e-05 \*\*\*  
## AnswerCount 1.123e-02 1.423e-03 7.886 3.11e-15 \*\*\*  
## CommentCount -4.850e-02 1.495e-03 -32.437 < 2e-16 \*\*\*  
## has\_code -1.859e-01 2.081e-02 -8.931 < 2e-16 \*\*\*  
## post\_views -3.838e-07 3.298e-08 -11.636 < 2e-16 \*\*\*  
## UserId 7.001e-07 1.369e-08 51.152 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 95830 on 69290 degrees of freedom  
## Residual deviance: 78062 on 69275 degrees of freedom  
## AIC: 78094  
##   
## Number of Fisher Scoring iterations: 5

#performing trial modelling  
#step(logistics\_model, direction = "backward")  
  
#final regression model  
logistics\_model = glm(formula = IsSpam ~ PostTypeId + PostScore + post\_length +   
 owner\_reputation + owner\_profile\_summary +   
 owner\_upvotes+ editDurationAfterCreation +   
 q\_num\_tags + AnswerCount + CommentCount+ post\_views, family = "binomial", data = training\_data)  
summary(logistics\_model)

##   
## Call:  
## glm(formula = IsSpam ~ PostTypeId + PostScore + post\_length +   
## owner\_reputation + owner\_profile\_summary + owner\_upvotes +   
## editDurationAfterCreation + q\_num\_tags + AnswerCount + CommentCount +   
## post\_views, family = "binomial", data = training\_data)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.1181 -0.9882 0.6144 0.9149 2.8775   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 8.323e+01 1.028e+01 8.093 5.84e-16 \*\*\*  
## PostTypeId -8.231e+01 1.031e+01 -7.980 1.46e-15 \*\*\*  
## PostScore -6.284e-05 9.381e-06 -6.699 2.10e-11 \*\*\*  
## post\_length -1.492e-05 4.598e-06 -3.244 0.00118 \*\*   
## owner\_reputation -1.232e-06 1.435e-07 -8.585 < 2e-16 \*\*\*  
## owner\_profile\_summary 4.576e-02 7.467e-03 6.129 8.87e-10 \*\*\*  
## owner\_upvotes 6.464e-05 5.587e-06 11.569 < 2e-16 \*\*\*  
## editDurationAfterCreation -5.639e-07 7.742e-09 -72.831 < 2e-16 \*\*\*  
## q\_num\_tags 7.472e-02 1.044e-02 7.158 8.17e-13 \*\*\*  
## AnswerCount 8.073e-03 1.416e-03 5.702 1.19e-08 \*\*\*  
## CommentCount -4.645e-02 1.461e-03 -31.790 < 2e-16 \*\*\*  
## post\_views -4.293e-07 3.382e-08 -12.694 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 95830 on 69290 degrees of freedom  
## Residual deviance: 81598 on 69279 degrees of freedom  
## AIC: 81622  
##   
## Number of Fisher Scoring iterations: 4

#Bayesian information criterion  
BIC(logistics\_model)

## [1] 81732.11

logistics\_probs<-predict(logistics\_model,training\_data,type="response")  
head(logistics\_probs)

## 26569 72830 37784 81578 86885 4209   
## 0.3176955 0.6167894 0.2470509 0.4929076 0.7764361 0.1575586

logistics\_pred\_y=rep(0,length(testing\_y))  
logistics\_pred\_y[logistics\_probs>0.55]=1  
training\_y=training\_data$IsSpam  
table(logistics\_pred\_y,training\_y)

## training\_y  
## logistics\_pred\_y 0 1  
## 0 7509 3948  
## 1 10138 25012

mean(logistics\_pred\_y!=training\_y,na.rm=TRUE)

## [1] 0.3022293

logistics\_probs<- predict(logistics\_model,testing\_data, type="response")  
head(logistics\_probs)

## 1 3 10 13 15 22   
## 0.2402627 0.2402627 0.2402627 0.2402627 0.2402627 0.2402627

logistics\_pred\_y=rep(0,length(testing\_y))  
logistics\_pred\_y[logistics\_probs>0.55]=1  
table(logistics\_pred\_y,testing\_y)

## testing\_y  
## logistics\_pred\_y 0 1  
## 0 7419 3838  
## 1 3269 8571

mean(logistics\_pred\_y!=testing\_y,na.rm=TRUE)

## [1] 0.3077023

#kfold for regression  
library(boot)  
MSE\_10\_Fold\_CV=cv.glm(training\_data,logistics\_model,K=10)$delta[1]  
MSE\_10\_Fold\_CV

## [1] 0.2001554

MSE\_10\_Fold\_CV=NULL  
for(i in 1:10){  
model=glm(IsSpam~poly(PostTypeId + PostScore + post\_length +   
 owner\_reputation + owner\_profile\_summary +   
 owner\_upvotes+ editDurationAfterCreation +   
 q\_num\_tags + AnswerCount + CommentCount+ post\_views),data=training\_data)  
MSE\_10\_Fold\_CV[i]=cv.glm(training\_data,model,K=10)$delta[1]  
}  
#summary(model)  
MSE\_10\_Fold\_CV

## [1] 0.2072214 0.2072230 0.2072172 0.2072193 0.2072205 0.2072235 0.2072220  
## [8] 0.2072256 0.2072207 0.2072290

#ROC logistic regression  
#install.packagesll.packages("ROCR")  
library(ROCR)

## Warning: package 'ROCR' was built under R version 3.3.2

## Loading required package: gplots

##   
## Attaching package: 'gplots'

## The following object is masked from 'package:stats':  
##   
## lowess

ROCRpred=prediction(logistics\_probs,testing\_y)  
ROCRperf=performance(ROCRpred,"tpr","fpr")  
#plot(ROCRperf)  
#plot(ROCRperf,colorize=TRUE)  
#plot(ROCRperf,colorize=TRUE,print.cutoffs.at=seq(0,1,0.05),text.adj=c(-0.2,1.7))  
as.numeric(performance(ROCRpred,"auc")@y.values)

## [1] 0.762237

#Linear Discriminant Analysis  
library(MASS)  
lda.model<- lda(IsSpam~PostTypeId + PostScore + post\_length +  
    owner\_reputation + owner\_profile\_summary +  
    owner\_upvotes+ editDurationAfterCreation +  
    q\_num\_tags + AnswerCount + CommentCount+ post\_views,data=training\_data)  
 #lda.model  
#summary(lda.model)  
lda\_pred<- predict(lda.model,training\_data)  
lda\_class<- lda\_pred$class  
table(lda\_class,training\_data$IsSpam)

##   
## lda\_class 0 1  
## 0 19887 8900  
## 1 12774 27730

mean(lda\_class!=training\_data$IsSpam)

## [1] 0.3127968

lda\_pred<- predict(lda.model,testing\_data)  
lda\_class<- lda\_pred$class  
table(lda\_class,testing\_data$IsSpam)

##   
## lda\_class 0 1  
## 0 6519 2934  
## 1 4169 9475

mean(lda\_class!=testing\_data$IsSpam)

## [1] 0.3075291

#Cross Validation LDA  
library(rpart)   
library(ipred)

## Warning: package 'ipred' was built under R version 3.3.2

ip.lda <- function(object, newdata) predict(object, newdata = newdata)$class  
errorest(factor(training\_data$IsSpam)~ training\_data$PostTypeId + training\_data$PostScore + training\_data$post\_length +  
    training\_data$owner\_reputation + training\_data$owner\_profile\_summary +  
    training\_data$owner\_upvotes+ training\_data$editDurationAfterCreation +  
    training\_data$q\_num\_tags + training\_data$AnswerCount + training\_data$CommentCount+ training\_data$post\_views, data=training\_data, model=lda, estimator="cv",est.para=control.errorest(k=10), predict=ip.lda)$err

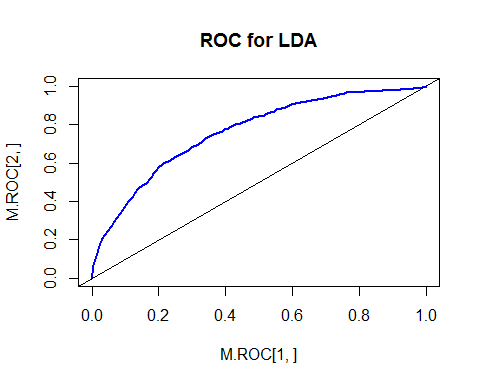
## [1] 0.4942345

#ROC LDA  
S=lda\_pred$posterior[,2]  
roc.curve=function(s,print=FALSE){  
 Ps=(S>s)\*1  
 FP=sum((Ps==1)\*(testing\_data$IsSpam==0))/sum(testing\_data$IsSpam==0)  
TP=sum((Ps==1)\*(testing\_data$IsSpam==1))/sum(testing\_data$IsSpam==1)  
 if(print==TRUE){  
     print(table(Observed=testing\_data$IsSpam,Predicted=Ps))  
  }  
 vect=c(FP,TP)  
 names(vect)=c("FPR","TPR")  
 return(vect)  
}  
threshold=0.53  
roc.curve(threshold,print=TRUE)

## Predicted  
## Observed 0 1  
## 0 7052 3636  
## 1 3388 9021

## FPR TPR   
## 0.3401946 0.7269724

ROC.curve=Vectorize(roc.curve)  
M.ROC=ROC.curve(seq(0,1,by=.01))  
plot(M.ROC[1,],M.ROC[2,],col="blue",lwd=2,type="l",main="ROC for LDA")  
abline(0,1)



#Quadratic Discriminant Analysis  
qda.model<- qda(IsSpam~PostTypeId + PostScore + post\_length +  
    owner\_reputation + owner\_profile\_summary +  
    owner\_upvotes+ editDurationAfterCreation +  
    q\_num\_tags + AnswerCount + CommentCount+ post\_views,data=training\_data)  
#qda.model  
qda\_pred=predict(qda.model,training\_data)  
qda\_class=qda\_pred$class  
table(qda\_class,training\_data$IsSpam)

##   
## qda\_class 0 1  
## 0 12455 4444  
## 1 20206 32186

mean(qda\_class!=training\_data$IsSpam)

## [1] 0.3557461

qda\_pred1=predict(qda.model,testing\_data)  
qda\_class\_n=qda\_pred1$class  
table(qda\_class\_n,testing\_data$IsSpam)

##   
## qda\_class\_n 0 1  
## 0 4114 1525  
## 1 6574 10884

mean(qda\_class\_n!=testing\_data$IsSpam)

## [1] 0.3506516

#Cross Validation QDA  
ip.qda <- function(object, newdata) predict(object, newdata = newdata)$class  
errorest(factor(training\_data$IsSpam)~ training\_data$PostTypeId + training\_data$PostScore + training\_data$post\_length +  
    training\_data$owner\_reputation + training\_data$owner\_profile\_summary +  
    training\_data$owner\_upvotes+ training\_data$editDurationAfterCreation +  
    training\_data$q\_num\_tags + training\_data$AnswerCount + training\_data$CommentCount+ training\_data$post\_views, data=training\_data, model=qda, estimator="cv",est.para=control.errorest(k=10), predict=ip.qda)$err

## [1] 0.483959

#ROC QDA  
qda.S=qda\_pred1$posterior[,2]  
roc.curve=function(s,print=FALSE){  
 Ps=(qda.S>s)\*1  
 FP=sum((Ps==1)\*(testing\_data$IsSpam==0))/sum(testing\_data$IsSpam==0)  
 TP=sum((Ps==1)\*(testing\_data$IsSpam==1))/sum(testing\_data$IsSpam==1)  
 if(print==TRUE){  
    print(table(Observed=testing\_data$IsSpam,Predicted=Ps))  
    }  
 vect=c(FP,TP)  
 names(vect)=c("FPR","TPR")  
 return(vect)  
 }  
threshold=0.85  
roc.curve(threshold,print=TRUE)

## Predicted  
## Observed 0 1  
## 0 6406 4282  
## 1 3037 9372

## FPR TPR   
## 0.4006362 0.7552583

ROC.curve=Vectorize(roc.curve)  
M.ROC=ROC.curve(seq(0,1,by=.01))  
plot(M.ROC[1,],M.ROC[2,],col="blue",lwd=2,type="l",main="ROC for LDA")  
abline(0,1)  
  
#k nearest neighbours  
library(class)  
#install.packagesll.packages("cars")  
#library(cars)  
train.x<- cbind(training\_data$PostTypeId + training\_data$PostScore + training\_data$post\_length + training\_data$owner\_reputation + training\_data$owner\_profile\_summary + training\_data$owner\_upvotes + training\_data$editDurationAfterCreation + training\_data$q\_num\_tags + training\_data$AnswerCount + training\_data$CommentCount + training\_data$post\_views)  
  
test.x<- cbind(testing\_data$PostTypeId + testing\_data$PostScore + testing\_data$post\_length + testing\_data$owner\_reputation + testing\_data$owner\_profile\_summary + testing\_data$owner\_upvotes + testing\_data$editDurationAfterCreation + testing\_data$q\_num\_tags + testing\_data$AnswerCount + testing\_data$CommentCount + testing\_data$post\_views)  
  
train1.x=train.x[!duplicated(train.x),drop=FALSE]  
test1.x=test.x[!duplicated(test.x), drop=FALSE]  
tt<- training\_data$IsSpam[duplicated(train.x)=='FALSE']  
head(tt)

## [1] 0 0 0 0 1 0

length(tt)

## [1] 6402

knn.pred<- knn(data.frame(train1.x),data.frame(test1.x),tt,k=1)  
tt1<- testing\_data$IsSpam[duplicated(test.x)=='FALSE']  
length(tt1)

## [1] 5179

table(knn.pred,tt1)

## tt1  
## knn.pred 0 1  
## 0 1580 38  
## 1 40 3521

mean(knn.pred!=tt1)

## [1] 0.01506082

knn.pred<- knn(data.frame(train1.x),data.frame(test1.x),tt,k=2)  
table(knn.pred,tt1)

## tt1  
## knn.pred 0 1  
## 0 1146 500  
## 1 474 3059

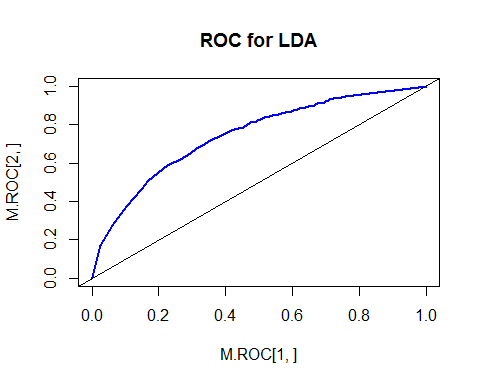
mean(knn.pred!=tt1)

## [1] 0.1880672

#Classification and Regression Trees  
#CART Modeling:  
 #install.packagesll.packages("tree")  
library(tree)

## Warning: package 'tree' was built under R version 3.3.2

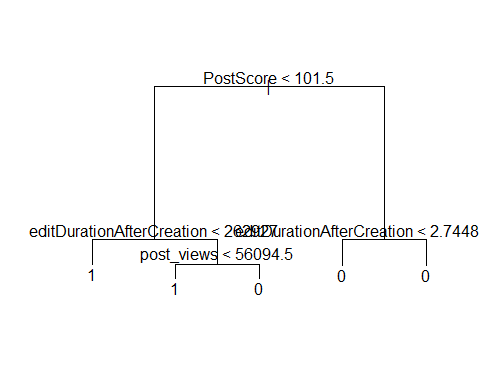
tree.training\_data=tree(as.factor(IsSpam)~PostTypeId + PostScore + post\_length +  
    owner\_reputation + owner\_profile\_summary +  
    owner\_upvotes+ editDurationAfterCreation +  
    q\_num\_tags + AnswerCount + CommentCount+ post\_views,training\_data)  
text(tree.training\_data,pretty=0)



summary(tree.training\_data)

##   
## Classification tree:  
## tree(formula = as.factor(IsSpam) ~ PostTypeId + PostScore + post\_length +   
## owner\_reputation + owner\_profile\_summary + owner\_upvotes +   
## editDurationAfterCreation + q\_num\_tags + AnswerCount + CommentCount +   
## post\_views, data = training\_data)  
## Variables actually used in tree construction:  
## [1] "PostScore" "editDurationAfterCreation"  
## [3] "post\_views"   
## Number of terminal nodes: 5   
## Residual mean deviance: 1.133 = 78490 / 69290   
## Misclassification error rate: 0.2791 = 19339 / 69291

plot(tree.training\_data)  
text(tree.training\_data,pretty=0)



lf<- seq(1,nrow(training\_data))  
tree.training\_data=tree(as.factor(IsSpam)~PostTypeId + PostScore + post\_length +  
    owner\_reputation + owner\_profile\_summary +  
    owner\_upvotes+ editDurationAfterCreation +  
    q\_num\_tags + AnswerCount + CommentCount+ post\_views,training\_data,subset=lf)  
tree.pred=predict(tree.training\_data,testing\_data,type="class")  
table(tree.pred,testing\_y)

## testing\_y  
## tree.pred 0 1  
## 0 7871 3551  
## 1 2817 8858

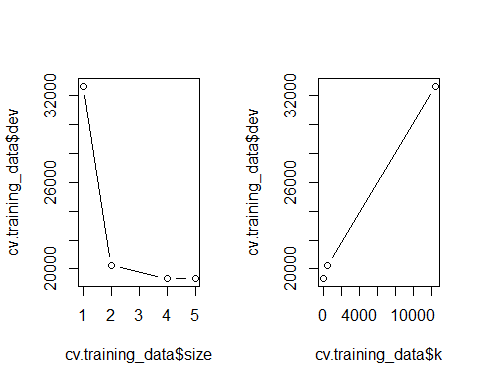
mean(tree.pred!=testing\_data$IsSpam)

## [1] 0.2757068

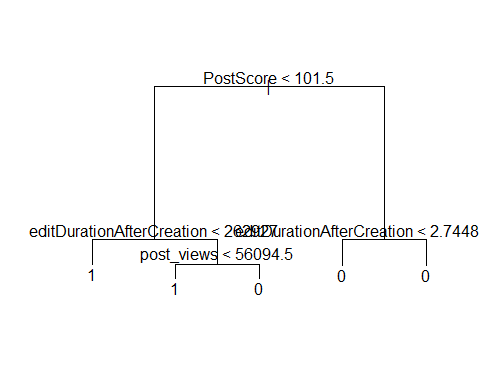
#Cross Validation and Pruning for the Classification Tree:  
cv.training\_data=cv.tree(tree.training\_data,FUN=prune.misclass)  
names(cv.training\_data)

## [1] "size" "dev" "k" "method"

#cv.training\_data  
par(mfrow=c(1,2))  
plot(cv.training\_data$size,cv.training\_data$dev,type="b")  
plot(cv.training\_data$k,cv.training\_data$dev,type="b")



par(mfrow=c(1,1))  
prune.training\_data=prune.misclass(tree.training\_data,best=5)  
plot(prune.training\_data)  
text(prune.training\_data,pretty=0)



tree.pred=predict(prune.training\_data,testing\_data,type="class")  
table(tree.pred,testing\_y)

## testing\_y  
## tree.pred 0 1  
## 0 7871 3551  
## 1 2817 8858

mean(tree.pred!=testing\_data$IsSpam)

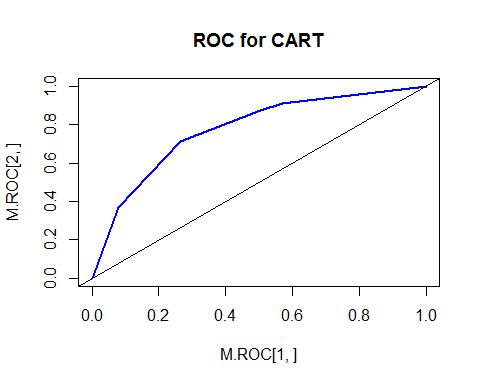
## [1] 0.2757068

#ROC for CART:  
 tree.pred=predict(tree.training\_data,testing\_data,type="vector",prob=TRUE)  
#tree.pred  
tree.S=tree.pred[,2]  
roc.curve=function(s,print=FALSE){  
Ps=(tree.S>s)\*1  
FP=sum((Ps==1)\*(testing\_data$IsSpam==0))/sum(testing\_data$IsSpam==0)  
TP=sum((Ps==1)\*(testing\_data$IsSpam==1))/sum(testing\_data$IsSpam==1)  
if(print==TRUE){  
 print(table(Observed=testing\_data$IsSpam,Predicted=Ps))  
}  
vect=c(FP,TP)  
names(vect)=c("FPR","TPR")  
return(vect)  
}  
 threshold=0.55  
 roc.curve(threshold,print=TRUE)

## Predicted  
## Observed 0 1  
## 0 7871 2817  
## 1 3551 8858

## FPR TPR   
## 0.2635666 0.7138367

ROC.curve=Vectorize(roc.curve)  
 M.ROC=ROC.curve(seq(0,1,by=.01))  
 plot(M.ROC[1,],M.ROC[2,],col="blue",lwd=2,type="l",main="ROC for CART")  
 abline(0,1)



#Random Forest Modeling:  
#install.packagesll.packages("randomForest")  
library(randomForest)

## Warning: package 'randomForest' was built under R version 3.3.2

## randomForest 4.6-12

## Type rfNews() to see new features/changes/bug fixes.

bag.training\_data=randomForest(as.factor(IsSpam)~PostTypeId + PostScore + post\_length +  
    owner\_reputation + owner\_profile\_summary +  
    owner\_upvotes+ editDurationAfterCreation +  
    q\_num\_tags + AnswerCount + CommentCount+ post\_views,data=training\_data,subset=lf,importance=TRUE)  
  
  
  
#bag.training\_data  
xyz = predict(bag.training\_data, newdata = testing\_data)  
table(testing\_y, xyz)

## xyz  
## testing\_y 0 1  
## 0 10562 126  
## 1 226 12183

mean(xyz!=testing\_y)

## [1] 0.01524007

#C5.0  
 #install.packagesll.packages("C50")  
 library(C50)

## Warning: package 'C50' was built under R version 3.3.2

c50\_model <- C5.0(training\_data[-16], as.factor(training\_data$IsSpam))  
c50\_model

##   
## Call:  
## C5.0.default(x = training\_data[-16], y = as.factor(training\_data$IsSpam))  
##   
## Classification Tree  
## Number of samples: 69291   
## Number of predictors: 15   
##   
## Tree size: 1146   
##   
## Non-standard options: attempt to group attributes

#summary(c50\_model)  
  
#testing C50  
c50\_pred <- predict(c50\_model, testing\_data)  
#install.packagesll.packages("gmodels")  
library(gmodels)

## Warning: package 'gmodels' was built under R version 3.3.2

CrossTable(testing\_data$IsSpam, c50\_pred,  
 prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,  
 dnn = c('actual default', 'predicted default'))

##   
##   
## Cell Contents  
## |-------------------------|  
## | N |  
## | N / Table Total |  
## |-------------------------|  
##   
##   
## Total Observations in Table: 23097   
##   
##   
## | predicted default   
## actual default | 0 | 1 | Row Total |   
## ---------------|-----------|-----------|-----------|  
## 0 | 10534 | 154 | 10688 |   
## | 0.456 | 0.007 | |   
## ---------------|-----------|-----------|-----------|  
## 1 | 266 | 12143 | 12409 |   
## | 0.012 | 0.526 | |   
## ---------------|-----------|-----------|-----------|  
## Column Total | 10800 | 12297 | 23097 |   
## ---------------|-----------|-----------|-----------|  
##   
##

#improving C50 with adaptive boosting  
c50\_boost10 <- C5.0(training\_data[-16], as.factor(training\_data$IsSpam),  
 trials = 10)  
c50\_boost10

##   
## Call:  
## C5.0.default(x = training\_data[-16], y =  
## as.factor(training\_data$IsSpam), trials = 10)  
##   
## Classification Tree  
## Number of samples: 69291   
## Number of predictors: 15   
##   
## Number of boosting iterations: 10   
## Average tree size: 711.7   
##   
## Non-standard options: attempt to group attributes

c50\_pred10 <- predict(c50\_boost10, testing\_data)  
CrossTable(testing\_data$IsSpam, c50\_pred10,  
 prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE,  
 dnn = c('actual default', 'predicted default'))

##   
##   
## Cell Contents  
## |-------------------------|  
## | N |  
## | N / Table Total |  
## |-------------------------|  
##   
##   
## Total Observations in Table: 23097   
##   
##   
## | predicted default   
## actual default | 0 | 1 | Row Total |   
## ---------------|-----------|-----------|-----------|  
## 0 | 10381 | 307 | 10688 |   
## | 0.449 | 0.013 | |   
## ---------------|-----------|-----------|-----------|  
## 1 | 220 | 12189 | 12409 |   
## | 0.010 | 0.528 | |   
## ---------------|-----------|-----------|-----------|  
## Column Total | 10601 | 12496 | 23097 |   
## ---------------|-----------|-----------|-----------|  
##   
##

#Dimensional Reduction  
#install.packages("ISLR")  
library(ISLR)

## Warning: package 'ISLR' was built under R version 3.3.2

#fix(StackOverflowdata)  
#names(StackOverflowdata)  
dim(StackOverflowdata)

## [1] 92388 16

sum(is.na(StackOverflowdata$IsSpam))

## [1] 0

StackOverflowdata=na.omit(StackOverflowdata)  
dim(StackOverflowdata)

## [1] 92388 16

sum(is.na(StackOverflowdata))

## [1] 0

#  
#install.packages("pls")  
library(pls)

## Warning: package 'pls' was built under R version 3.3.2

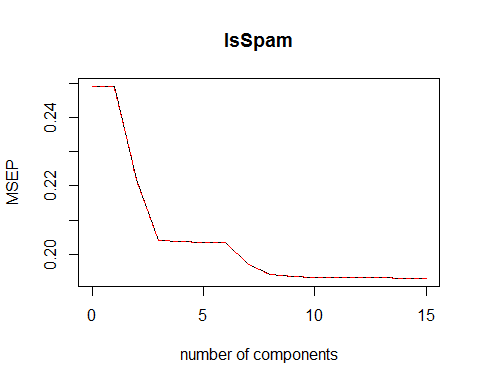
##   
## Attaching package: 'pls'

## The following object is masked from 'package:stats':  
##   
## loadings

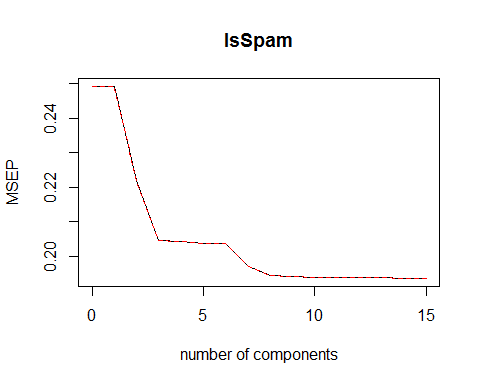
set.seed(2)  
pcr.fit=pcr(IsSpam~., data=StackOverflowdata,scale=TRUE,validation="CV")  
summary(pcr.fit)

## Data: X dimension: 92388 15   
## Y dimension: 92388 1  
## Fit method: svdpc  
## Number of components considered: 15  
##   
## VALIDATION: RMSEP  
## Cross-validated using 10 random segments.  
## (Intercept) 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps  
## CV 0.4991 0.499 0.4709 0.4517 0.4514 0.4509 0.451  
## adjCV 0.4991 0.499 0.4709 0.4517 0.4514 0.4509 0.451  
## 7 comps 8 comps 9 comps 10 comps 11 comps 12 comps 13 comps  
## CV 0.4440 0.4405 0.44 0.4397 0.4397 0.4397 0.4396  
## adjCV 0.4439 0.4405 0.44 0.4397 0.4397 0.4397 0.4396  
## 14 comps 15 comps  
## CV 0.4393 0.4393  
## adjCV 0.4393 0.4393  
##   
## TRAINING: % variance explained  
## 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps 7 comps  
## X 26.30757 45.12 57.47 65.57 71.92 77.24 82.00  
## IsSpam 0.01617 10.95 18.07 18.18 18.36 18.36 20.88  
## 8 comps 9 comps 10 comps 11 comps 12 comps 13 comps 14 comps  
## X 86.41 90.47 93.65 96.34 98.55 100.00 100.00  
## IsSpam 22.10 22.28 22.39 22.40 22.40 22.43 22.52  
## 15 comps  
## X 100.00  
## IsSpam 22.55

validationplot(pcr.fit,val.type="MSEP")



set.seed(1)  
pcr.fit=pcr(IsSpam~., data=training\_data,scale=TRUE, validation="CV")  
validationplot(pcr.fit,val.type="MSEP")



pcr.pred=predict(pcr.fit,testing\_data,ncomp=7)  
mean((pcr.pred-testing\_y)^2)

## [1] 0.1955035

pcr.fit=pcr(IsSpam~.,data=testing\_data, scale=TRUE,ncomp=7)  
mean((pcr.pred-testing\_y)^2)

## [1] 0.1955035

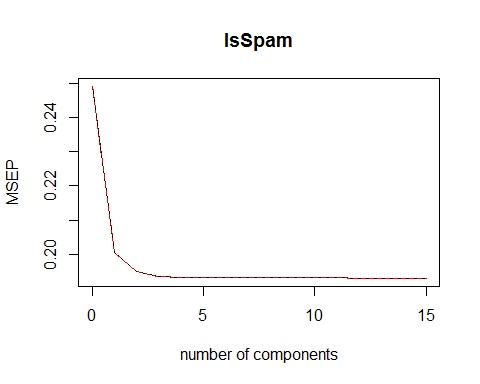
summary(pcr.fit)

## Data: X dimension: 23097 15   
## Y dimension: 23097 1  
## Fit method: svdpc  
## Number of components considered: 7  
## TRAINING: % variance explained  
## 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps 7 comps  
## X 26.333119 45.31 57.77 65.82 72.18 77.50 82.28  
## IsSpam 0.002557 10.98 18.38 18.52 18.75 18.82 20.83

#  
# Partial Least Squares  
set.seed(1)  
pls.fit=plsr(IsSpam~., data=StackOverflowdata,scale=TRUE, validation="CV")  
summary(pls.fit)

## Data: X dimension: 92388 15   
## Y dimension: 92388 1  
## Fit method: kernelpls  
## Number of components considered: 15  
##   
## VALIDATION: RMSEP  
## Cross-validated using 10 random segments.  
## (Intercept) 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps  
## CV 0.4991 0.4477 0.4416 0.4398 0.4397 0.4396 0.4396  
## adjCV 0.4991 0.4477 0.4416 0.4398 0.4397 0.4396 0.4396  
## 7 comps 8 comps 9 comps 10 comps 11 comps 12 comps 13 comps  
## CV 0.4396 0.4396 0.4396 0.4396 0.4396 0.4394 0.4394  
## adjCV 0.4396 0.4396 0.4396 0.4396 0.4396 0.4394 0.4394  
## 14 comps 15 comps  
## CV 0.4394 0.4393  
## adjCV 0.4394 0.4393  
##   
## TRAINING: % variance explained  
## 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps 7 comps  
## X 17.17 28.91 39.21 61.93 67.83 72.43 76.97  
## IsSpam 19.54 21.74 22.35 22.42 22.43 22.43 22.43  
## 8 comps 9 comps 10 comps 11 comps 12 comps 13 comps 14 comps  
## X 81.40 85.47 88.30 91.30 92.30 95.88 100.00  
## IsSpam 22.43 22.43 22.44 22.45 22.52 22.52 22.52  
## 15 comps  
## X 100.00  
## IsSpam 22.55

validationplot(pls.fit,val.type="MSEP")



pls.pred=predict(pls.fit,testing\_data,ncomp=2)  
mean((pls.pred-testing\_y)^2)

## [1] 0.1935006

pls.fit=plsr(IsSpam~., data=StackOverflowdata,scale=TRUE,ncomp=2)  
summary(pls.fit)

## Data: X dimension: 92388 15   
## Y dimension: 92388 1  
## Fit method: kernelpls  
## Number of components considered: 2  
## TRAINING: % variance explained  
## 1 comps 2 comps  
## X 17.17 28.91  
## IsSpam 19.54 21.74

# Subset Selection Methods  
# Best Subset Selection  
#install.packages("ISLR")  
library(ISLR)  
#sum(is.na(StackOverflow$IsSpam))  
lapply(StackOverflowdata["IsSpam"], unique)

## $IsSpam  
## [1] 1 0

#Confirms NO NA data  
#  
#install.packages("leaps")  
library(leaps)

## Warning: package 'leaps' was built under R version 3.3.2

regfit.full=regsubsets(IsSpam~.,StackOverflowdata)  
summary(regfit.full)

## Subset selection object  
## Call: regsubsets.formula(IsSpam ~ ., StackOverflowdata)  
## 15 Variables (and intercept)  
## Forced in Forced out  
## PostTypeId FALSE FALSE  
## PostScore FALSE FALSE  
## post\_length FALSE FALSE  
## owner\_reputation FALSE FALSE  
## owner\_profile\_summary FALSE FALSE  
## owner\_views FALSE FALSE  
## owner\_upvotes FALSE FALSE  
## owner\_downvotes FALSE FALSE  
## editDurationAfterCreation FALSE FALSE  
## q\_num\_tags FALSE FALSE  
## AnswerCount FALSE FALSE  
## CommentCount FALSE FALSE  
## has\_code FALSE FALSE  
## post\_views FALSE FALSE  
## UserId FALSE FALSE  
## 1 subsets of each size up to 8  
## Selection Algorithm: exhaustive  
## PostTypeId PostScore post\_length owner\_reputation  
## 1 ( 1 ) " " " " " " " "   
## 2 ( 1 ) " " " " " " " "   
## 3 ( 1 ) " " " " " " " "   
## 4 ( 1 ) " " " " " " " "   
## 5 ( 1 ) " " " " " " " "   
## 6 ( 1 ) " " " " " " " "   
## 7 ( 1 ) " " " " " " " "   
## 8 ( 1 ) " " " " "\*" " "   
## owner\_profile\_summary owner\_views owner\_upvotes owner\_downvotes  
## 1 ( 1 ) " " " " " " " "   
## 2 ( 1 ) " " " " " " " "   
## 3 ( 1 ) " " " " " " " "   
## 4 ( 1 ) " " " " " " " "   
## 5 ( 1 ) "\*" " " " " " "   
## 6 ( 1 ) "\*" " " " " " "   
## 7 ( 1 ) "\*" " " "\*" " "   
## 8 ( 1 ) "\*" " " "\*" " "   
## editDurationAfterCreation q\_num\_tags AnswerCount CommentCount  
## 1 ( 1 ) "\*" " " " " " "   
## 2 ( 1 ) "\*" " " " " " "   
## 3 ( 1 ) "\*" " " " " "\*"   
## 4 ( 1 ) "\*" " " "\*" "\*"   
## 5 ( 1 ) "\*" " " "\*" "\*"   
## 6 ( 1 ) "\*" " " "\*" "\*"   
## 7 ( 1 ) "\*" " " "\*" "\*"   
## 8 ( 1 ) "\*" " " "\*" "\*"   
## has\_code post\_views UserId  
## 1 ( 1 ) " " " " " "   
## 2 ( 1 ) " " " " "\*"   
## 3 ( 1 ) " " " " "\*"   
## 4 ( 1 ) " " " " "\*"   
## 5 ( 1 ) " " " " "\*"   
## 6 ( 1 ) " " "\*" "\*"   
## 7 ( 1 ) " " "\*" "\*"   
## 8 ( 1 ) " " "\*" "\*"

#  
regfit.full=regsubsets(IsSpam~.,data=StackOverflowdata,nvmax=19)  
reg.summary=summary(regfit.full)  
names(reg.summary)

## [1] "which" "rsq" "rss" "adjr2" "cp" "bic" "outmat" "obj"

#  
reg.summary$rsq

## [1] 0.1644792 0.1909453 0.2003658 0.2119149 0.2165611 0.2190217 0.2206896  
## [8] 0.2225965 0.2234860 0.2241067 0.2246671 0.2249888 0.2252077 0.2253372  
## [15] 0.2254766

#  
par(mfrow=c(2,2))  
plot(reg.summary$rss,xlab="Number of Variables",ylab="RSS",type="l")  
plot(reg.summary$adjr2,xlab="Number of Variables",ylab="Adjusted   
RSq",type="l")  
which.max(reg.summary$adjr2)

## [1] 15

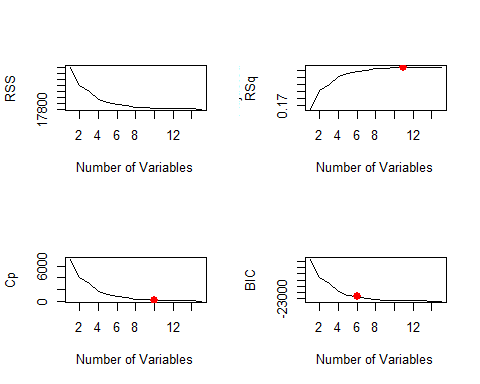
points(11,reg.summary$adjr2[11], col="red",cex=2,pch=20)  
plot(reg.summary$cp,xlab="Number of Variables",ylab="Cp",type='l')  
which.min(reg.summary$cp)

## [1] 15

points(10,reg.summary$cp[10],col="red",cex=2,pch=20)  
which.min(reg.summary$bic)

## [1] 15

plot(reg.summary$bic,xlab="Number of Variables",ylab="BIC",type='l')  
points(6,reg.summary$bic[6],col="red",cex=2,pch=20)



#plot(regfit.full,scale="r2")  
#plot(regfit.full,scale="adjr2")  
#plot(regfit.full,scale="Cp")  
#plot(regfit.full,scale="bic")  
coef(regfit.full,6)

## (Intercept) owner\_profile\_summary   
## 5.889294e-01 2.868317e-02   
## editDurationAfterCreation AnswerCount   
## -9.138459e-08 7.260512e-05   
## CommentCount post\_views   
## -9.128528e-03 -6.525074e-08   
## UserId   
## 9.362390e-08

# Forward and Backward Stepwise Selection  
  
regfit.fwd=regsubsets(IsSpam~.,data=StackOverflowdata,nvmax=19,method="forward")  
summary(regfit.fwd)

## Subset selection object  
## Call: regsubsets.formula(IsSpam ~ ., data = StackOverflowdata, nvmax = 19,   
## method = "forward")  
## 15 Variables (and intercept)  
## Forced in Forced out  
## PostTypeId FALSE FALSE  
## PostScore FALSE FALSE  
## post\_length FALSE FALSE  
## owner\_reputation FALSE FALSE  
## owner\_profile\_summary FALSE FALSE  
## owner\_views FALSE FALSE  
## owner\_upvotes FALSE FALSE  
## owner\_downvotes FALSE FALSE  
## editDurationAfterCreation FALSE FALSE  
## q\_num\_tags FALSE FALSE  
## AnswerCount FALSE FALSE  
## CommentCount FALSE FALSE  
## has\_code FALSE FALSE  
## post\_views FALSE FALSE  
## UserId FALSE FALSE  
## 1 subsets of each size up to 15  
## Selection Algorithm: forward  
## PostTypeId PostScore post\_length owner\_reputation  
## 1 ( 1 ) " " " " " " " "   
## 2 ( 1 ) " " " " " " " "   
## 3 ( 1 ) " " " " " " " "   
## 4 ( 1 ) " " " " " " " "   
## 5 ( 1 ) " " " " " " " "   
## 6 ( 1 ) " " " " " " " "   
## 7 ( 1 ) " " " " " " " "   
## 8 ( 1 ) " " " " "\*" " "   
## 9 ( 1 ) "\*" " " "\*" " "   
## 10 ( 1 ) "\*" " " "\*" " "   
## 11 ( 1 ) "\*" " " "\*" "\*"   
## 12 ( 1 ) "\*" "\*" "\*" "\*"   
## 13 ( 1 ) "\*" "\*" "\*" "\*"   
## 14 ( 1 ) "\*" "\*" "\*" "\*"   
## 15 ( 1 ) "\*" "\*" "\*" "\*"   
## owner\_profile\_summary owner\_views owner\_upvotes owner\_downvotes  
## 1 ( 1 ) " " " " " " " "   
## 2 ( 1 ) " " " " " " " "   
## 3 ( 1 ) " " " " " " " "   
## 4 ( 1 ) " " " " " " " "   
## 5 ( 1 ) "\*" " " " " " "   
## 6 ( 1 ) "\*" " " " " " "   
## 7 ( 1 ) "\*" " " "\*" " "   
## 8 ( 1 ) "\*" " " "\*" " "   
## 9 ( 1 ) "\*" " " "\*" " "   
## 10 ( 1 ) "\*" " " "\*" " "   
## 11 ( 1 ) "\*" " " "\*" " "   
## 12 ( 1 ) "\*" " " "\*" " "   
## 13 ( 1 ) "\*" " " "\*" " "   
## 14 ( 1 ) "\*" "\*" "\*" " "   
## 15 ( 1 ) "\*" "\*" "\*" "\*"   
## editDurationAfterCreation q\_num\_tags AnswerCount CommentCount  
## 1 ( 1 ) "\*" " " " " " "   
## 2 ( 1 ) "\*" " " " " " "   
## 3 ( 1 ) "\*" " " " " "\*"   
## 4 ( 1 ) "\*" " " "\*" "\*"   
## 5 ( 1 ) "\*" " " "\*" "\*"   
## 6 ( 1 ) "\*" " " "\*" "\*"   
## 7 ( 1 ) "\*" " " "\*" "\*"   
## 8 ( 1 ) "\*" " " "\*" "\*"   
## 9 ( 1 ) "\*" " " "\*" "\*"   
## 10 ( 1 ) "\*" " " "\*" "\*"   
## 11 ( 1 ) "\*" " " "\*" "\*"   
## 12 ( 1 ) "\*" " " "\*" "\*"   
## 13 ( 1 ) "\*" "\*" "\*" "\*"   
## 14 ( 1 ) "\*" "\*" "\*" "\*"   
## 15 ( 1 ) "\*" "\*" "\*" "\*"   
## has\_code post\_views UserId  
## 1 ( 1 ) " " " " " "   
## 2 ( 1 ) " " " " "\*"   
## 3 ( 1 ) " " " " "\*"   
## 4 ( 1 ) " " " " "\*"   
## 5 ( 1 ) " " " " "\*"   
## 6 ( 1 ) " " "\*" "\*"   
## 7 ( 1 ) " " "\*" "\*"   
## 8 ( 1 ) " " "\*" "\*"   
## 9 ( 1 ) " " "\*" "\*"   
## 10 ( 1 ) "\*" "\*" "\*"   
## 11 ( 1 ) "\*" "\*" "\*"   
## 12 ( 1 ) "\*" "\*" "\*"   
## 13 ( 1 ) "\*" "\*" "\*"   
## 14 ( 1 ) "\*" "\*" "\*"   
## 15 ( 1 ) "\*" "\*" "\*"

regfit.bwd=regsubsets(IsSpam~.,data=StackOverflowdata,nvmax=19,method="backward"  
)  
summary(regfit.bwd)

## Subset selection object  
## Call: regsubsets.formula(IsSpam ~ ., data = StackOverflowdata, nvmax = 19,   
## method = "backward")  
## 15 Variables (and intercept)  
## Forced in Forced out  
## PostTypeId FALSE FALSE  
## PostScore FALSE FALSE  
## post\_length FALSE FALSE  
## owner\_reputation FALSE FALSE  
## owner\_profile\_summary FALSE FALSE  
## owner\_views FALSE FALSE  
## owner\_upvotes FALSE FALSE  
## owner\_downvotes FALSE FALSE  
## editDurationAfterCreation FALSE FALSE  
## q\_num\_tags FALSE FALSE  
## AnswerCount FALSE FALSE  
## CommentCount FALSE FALSE  
## has\_code FALSE FALSE  
## post\_views FALSE FALSE  
## UserId FALSE FALSE  
## 1 subsets of each size up to 15  
## Selection Algorithm: backward  
## PostTypeId PostScore post\_length owner\_reputation  
## 1 ( 1 ) " " " " " " " "   
## 2 ( 1 ) " " " " " " " "   
## 3 ( 1 ) " " " " " " " "   
## 4 ( 1 ) " " " " " " " "   
## 5 ( 1 ) " " " " " " " "   
## 6 ( 1 ) " " " " " " " "   
## 7 ( 1 ) " " " " " " " "   
## 8 ( 1 ) " " " " "\*" " "   
## 9 ( 1 ) "\*" " " "\*" " "   
## 10 ( 1 ) "\*" " " "\*" " "   
## 11 ( 1 ) "\*" " " "\*" " "   
## 12 ( 1 ) "\*" "\*" "\*" " "   
## 13 ( 1 ) "\*" "\*" "\*" " "   
## 14 ( 1 ) "\*" "\*" "\*" "\*"   
## 15 ( 1 ) "\*" "\*" "\*" "\*"   
## owner\_profile\_summary owner\_views owner\_upvotes owner\_downvotes  
## 1 ( 1 ) " " " " " " " "   
## 2 ( 1 ) " " " " " " " "   
## 3 ( 1 ) " " " " " " " "   
## 4 ( 1 ) " " " " " " " "   
## 5 ( 1 ) "\*" " " " " " "   
## 6 ( 1 ) "\*" " " " " " "   
## 7 ( 1 ) "\*" " " "\*" " "   
## 8 ( 1 ) "\*" " " "\*" " "   
## 9 ( 1 ) "\*" " " "\*" " "   
## 10 ( 1 ) "\*" " " "\*" " "   
## 11 ( 1 ) "\*" "\*" "\*" " "   
## 12 ( 1 ) "\*" "\*" "\*" " "   
## 13 ( 1 ) "\*" "\*" "\*" " "   
## 14 ( 1 ) "\*" "\*" "\*" " "   
## 15 ( 1 ) "\*" "\*" "\*" "\*"   
## editDurationAfterCreation q\_num\_tags AnswerCount CommentCount  
## 1 ( 1 ) "\*" " " " " " "   
## 2 ( 1 ) "\*" " " " " " "   
## 3 ( 1 ) "\*" " " " " "\*"   
## 4 ( 1 ) "\*" " " "\*" "\*"   
## 5 ( 1 ) "\*" " " "\*" "\*"   
## 6 ( 1 ) "\*" " " "\*" "\*"   
## 7 ( 1 ) "\*" " " "\*" "\*"   
## 8 ( 1 ) "\*" " " "\*" "\*"   
## 9 ( 1 ) "\*" " " "\*" "\*"   
## 10 ( 1 ) "\*" " " "\*" "\*"   
## 11 ( 1 ) "\*" " " "\*" "\*"   
## 12 ( 1 ) "\*" " " "\*" "\*"   
## 13 ( 1 ) "\*" "\*" "\*" "\*"   
## 14 ( 1 ) "\*" "\*" "\*" "\*"   
## 15 ( 1 ) "\*" "\*" "\*" "\*"   
## has\_code post\_views UserId  
## 1 ( 1 ) " " " " " "   
## 2 ( 1 ) " " " " "\*"   
## 3 ( 1 ) " " " " "\*"   
## 4 ( 1 ) " " " " "\*"   
## 5 ( 1 ) " " " " "\*"   
## 6 ( 1 ) " " "\*" "\*"   
## 7 ( 1 ) " " "\*" "\*"   
## 8 ( 1 ) " " "\*" "\*"   
## 9 ( 1 ) " " "\*" "\*"   
## 10 ( 1 ) "\*" "\*" "\*"   
## 11 ( 1 ) "\*" "\*" "\*"   
## 12 ( 1 ) "\*" "\*" "\*"   
## 13 ( 1 ) "\*" "\*" "\*"   
## 14 ( 1 ) "\*" "\*" "\*"   
## 15 ( 1 ) "\*" "\*" "\*"

coef(regfit.full,7)

## (Intercept) owner\_profile\_summary   
## 5.871938e-01 2.507065e-02   
## owner\_upvotes editDurationAfterCreation   
## 9.479629e-06 -9.138924e-08   
## AnswerCount CommentCount   
## 6.881578e-05 -9.428608e-03   
## post\_views UserId   
## -6.339023e-08 9.578108e-08

coef(regfit.fwd,7)

## (Intercept) owner\_profile\_summary   
## 5.871938e-01 2.507065e-02   
## owner\_upvotes editDurationAfterCreation   
## 9.479629e-06 -9.138924e-08   
## AnswerCount CommentCount   
## 6.881578e-05 -9.428608e-03   
## post\_views UserId   
## -6.339023e-08 9.578108e-08

coef(regfit.bwd,7)

## (Intercept) owner\_profile\_summary   
## 5.871938e-01 2.507065e-02   
## owner\_upvotes editDurationAfterCreation   
## 9.479629e-06 -9.138924e-08   
## AnswerCount CommentCount   
## 6.881578e-05 -9.428608e-03   
## post\_views UserId   
## -6.339023e-08 9.578108e-08

# Choosing Among Models  
  
set.seed(1)  
train=sample(c(TRUE,FALSE), nrow(StackOverflowdata),rep=TRUE)  
test=(!train)  
regfit.best=regsubsets(IsSpam~.,data=StackOverflowdata[train,],nvmax=19)  
test.mat=model.matrix(IsSpam~.,data=StackOverflowdata[test,])  
val.errors=rep(NA,19)  
for(i in 1:11){  
 coefi=coef(regfit.best,id=i)  
 pred=test.mat[,names(coefi)]%\*%coefi  
 val.errors[i]=mean((StackOverflowdata$IsSpam[test]-pred)^2)  
}  
val.errors

## [1] 0.2080485 0.2016296 0.1990773 0.1962963 0.1952114 0.1946127 0.1942180  
## [8] 0.1938474 0.1935931 0.1934303 0.1933006 NA NA NA  
## [15] NA NA NA NA NA

which.min(val.errors)

## [1] 11

coef(regfit.best,10)

## (Intercept) PostTypeId   
## 2.838404e+00 -2.247997e+00   
## post\_length owner\_profile\_summary   
## -1.048496e-05 2.520294e-02   
## owner\_upvotes editDurationAfterCreation   
## 1.211938e-05 -9.134598e-08   
## AnswerCount CommentCount   
## 2.333916e-03 -8.609658e-03   
## has\_code post\_views   
## -2.890255e-02 -9.042860e-08   
## UserId   
## 1.011327e-07

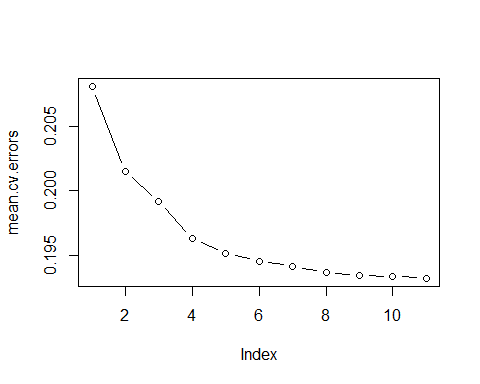
predict.regsubsets=function(object,newdata,id,...){  
 form=as.formula(object$call[[2]])  
 mat=model.matrix(form,newdata)  
 coefi=coef(object,id=id)  
 xvars=names(coefi)  
 mat[,xvars]%\*%coefi  
 }  
regfit.best=regsubsets(IsSpam~.,data=StackOverflowdata,nvmax=19)  
coef(regfit.best,10)

## (Intercept) PostTypeId   
## 3.054017e+00 -2.458100e+00   
## post\_length owner\_profile\_summary   
## -9.375770e-06 2.436814e-02   
## owner\_upvotes editDurationAfterCreation   
## 1.161266e-05 -9.216858e-08   
## AnswerCount CommentCount   
## 2.545020e-03 -8.946950e-03   
## has\_code post\_views   
## -2.969629e-02 -9.136166e-08   
## UserId   
## 9.970572e-08

k=10  
set.seed(1)  
folds=sample(1:k,nrow(StackOverflowdata),replace=TRUE)  
cv.errors=matrix(NA,k,11, dimnames=list(NULL, paste(1:11)))  
for(j in 1:k){  
 best.fit=regsubsets(IsSpam~.,data=StackOverflowdata[folds!=j,],nvmax=11)  
 for(i in 1:11){  
 pred=predict(best.fit,StackOverflowdata[folds==j,],id=i)  
 cv.errors[j,i]=mean( (StackOverflowdata$IsSpam[folds==j]-pred)^2)  
 }  
 }  
mean.cv.errors=apply(cv.errors,2,mean)  
mean.cv.errors

## 1 2 3 4 5 6 7   
## 0.2080934 0.2015016 0.1991608 0.1962864 0.1951342 0.1945307 0.1941220   
## 8 9 10 11   
## 0.1936495 0.1934335 0.1933440 0.1931856

par(mfrow=c(1,1))  
plot(mean.cv.errors,type='b')



reg.best=regsubsets(IsSpam~.,data=StackOverflowdata, nvmax=11)  
coef(reg.best,11)

## (Intercept) PostTypeId   
## 2.935493e+00 -2.341285e+00   
## post\_length owner\_reputation   
## -9.038649e-06 -1.935190e-07   
## owner\_profile\_summary owner\_upvotes   
## 2.466968e-02 1.664120e-05   
## editDurationAfterCreation AnswerCount   
## -9.152383e-08 2.429953e-03   
## CommentCount has\_code   
## -8.782691e-03 -2.977414e-02   
## post\_views UserId   
## -9.007912e-08 9.927786e-08

# Ridge Regression and LASSO  
#install.packages("ISLR")  
library(ISLR)  
fix(Hitters)  
names(Hitters)

## [1] "AtBat" "Hits" "HmRun" "Runs" "RBI"   
## [6] "Walks" "Years" "CAtBat" "CHits" "CHmRun"   
## [11] "CRuns" "CRBI" "CWalks" "League" "Division"   
## [16] "PutOuts" "Assists" "Errors" "Salary" "NewLeague"

dim(Hitters)

## [1] 322 20

sum(is.na(Hitters$Salary))

## [1] 59

Hitters=na.omit(Hitters)  
dim(Hitters)

## [1] 263 20

sum(is.na(Hitters))

## [1] 0

#  
#  
#install.packages("glmnet")  
library(glmnet)

## Loading required package: Matrix

## Loading required package: foreach

## Loaded glmnet 2.0-5

#the package invokes inputs and outputs separately unlike lm and glm  
x=model.matrix(Salary~.,Hitters)[,-1]  
y=Hitters$Salary  
  
# set vector of lambda values to study range from 10^10 to 0.01, total length=100  
grid=10^seq(10,-2,length=100)  
ridge.mod=glmnet(x,y,alpha=0,lambda=grid)  
dim(coef(ridge.mod))

## [1] 20 100

#  
# let us look at a few results here  
  
#first lambda=50  
#  
ridge.mod$lambda[50]

## [1] 11497.57

coef(ridge.mod)[,50]

## (Intercept) AtBat Hits HmRun Runs   
## 407.356050200 0.036957182 0.138180344 0.524629976 0.230701523   
## RBI Walks Years CAtBat CHits   
## 0.239841459 0.289618741 1.107702929 0.003131815 0.011653637   
## CHmRun CRuns CRBI CWalks LeagueN   
## 0.087545670 0.023379882 0.024138320 0.025015421 0.085028114   
## DivisionW PutOuts Assists Errors NewLeagueN   
## -6.215440973 0.016482577 0.002612988 -0.020502690 0.301433531

sqrt(sum(coef(ridge.mod)[-1,50]^2))

## [1] 6.360612

#next, lambda=60  
#  
ridge.mod$lambda[60]

## [1] 705.4802

coef(ridge.mod)[,60]

## (Intercept) AtBat Hits HmRun Runs   
## 54.32519950 0.11211115 0.65622409 1.17980910 0.93769713   
## RBI Walks Years CAtBat CHits   
## 0.84718546 1.31987948 2.59640425 0.01083413 0.04674557   
## CHmRun CRuns CRBI CWalks LeagueN   
## 0.33777318 0.09355528 0.09780402 0.07189612 13.68370191   
## DivisionW PutOuts Assists Errors NewLeagueN   
## -54.65877750 0.11852289 0.01606037 -0.70358655 8.61181213

sqrt(sum(coef(ridge.mod)[-1,60]^2))

## [1] 57.11001

#prediction of the coefficients for lambda=50 (play with this)  
predict(ridge.mod,s=500,type="coefficients")[1:20,]

## (Intercept) AtBat Hits HmRun Runs   
## 31.99399814 0.10192006 0.74275680 0.96443185 1.00120750   
## RBI Walks Years CAtBat CHits   
## 0.87010642 1.45801666 2.17233898 0.01118270 0.05128768   
## CHmRun CRuns CRBI CWalks LeagueN   
## 0.36723403 0.10258701 0.10794255 0.06645486 17.40736327   
## DivisionW PutOuts Assists Errors NewLeagueN   
## -65.72957647 0.14006794 0.02088359 -0.97696746 9.33672191

#prepare for training and validation set testing  
  
set.seed(1)  
train=sample(1:nrow(x), nrow(x)/2)  
test=(-train)  
y.test=y[test]  
  
ridge.mod=glmnet(x[train,],y[train],alpha=0,lambda=grid, thresh=1e-12)  
ridge.pred=predict(ridge.mod,s=4,newx=x[test,])  
#  
  
#evaluate and compare test MSE and the spread of y.test  
mean((ridge.pred-y.test)^2)

## [1] 101036.8

mean((mean(y[train])-y.test)^2)

## [1] 193253.1

#  
#test wth two other lambdas  
ridge.pred=predict(ridge.mod,s=1e10,newx=x[test,])  
mean((ridge.pred-y.test)^2)

## [1] 193253.1

ridge.pred=predict(ridge.mod,s=0,newx=x[test,],exact=T)  
mean((ridge.pred-y.test)^2)

## [1] 114783.1

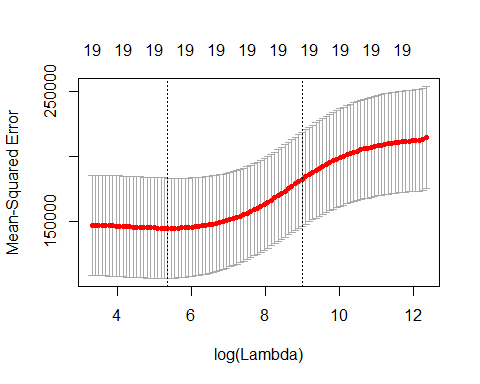
#  
#compare with lm  
# The following two are the same  
#  
lm(y~x, subset=train)

##   
## Call:  
## lm(formula = y ~ x, subset = train)  
##   
## Coefficients:  
## (Intercept) xAtBat xHits xHmRun xRuns   
## 299.42849 -2.54027 8.36682 11.64512 -9.09923   
## xRBI xWalks xYears xCAtBat xCHits   
## 2.44105 9.23440 -22.93673 -0.18154 -0.11598   
## xCHmRun xCRuns xCRBI xCWalks xLeagueN   
## -1.33888 3.32838 0.07536 -1.07841 59.76065   
## xDivisionW xPutOuts xAssists xErrors xNewLeagueN   
## -98.86233 0.34087 0.34165 -0.64207 -0.67442

predict(ridge.mod,s=0,exact=T,type="coefficients")[1:20,]

## (Intercept) AtBat Hits HmRun Runs   
## 299.42883596 -2.54014665 8.36611719 11.64400720 -9.09877719   
## RBI Walks Years CAtBat CHits   
## 2.44152119 9.23403909 -22.93584442 -0.18160843 -0.11561496   
## CHmRun CRuns CRBI CWalks LeagueN   
## -1.33836534 3.32817777 0.07511771 -1.07828647 59.76529059   
## DivisionW PutOuts Assists Errors NewLeagueN   
## -98.85996590 0.34086400 0.34165605 -0.64205839 -0.67606314

#  
#Cross validation to get the best lambda  
set.seed(1)  
cv.out=cv.glmnet(x[train,],y[train],alpha=0)  
plot(cv.out)



bestlam=cv.out$lambda.min  
bestlam

## [1] 211.7416

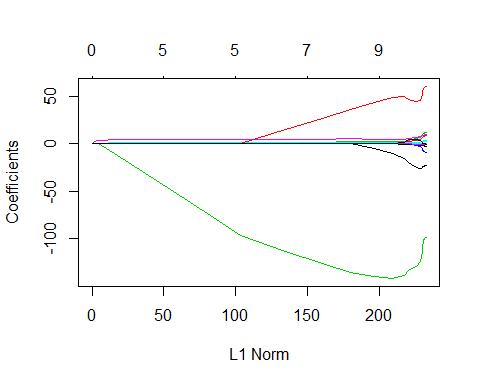
#now predict with the best lambda  
#  
ridge.pred=predict(ridge.mod,s=bestlam,newx=x[test,])  
mean((ridge.pred-y.test)^2)

## [1] 96015.51

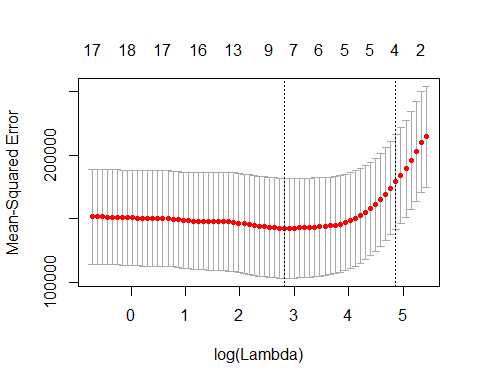
out=glmnet(x,y,alpha=0)  
predict(out,type="coefficients",s=bestlam)[1:20,]

## (Intercept) AtBat Hits HmRun Runs   
## 9.88487157 0.03143991 1.00882875 0.13927624 1.11320781   
## RBI Walks Years CAtBat CHits   
## 0.87318990 1.80410229 0.13074381 0.01113978 0.06489843   
## CHmRun CRuns CRBI CWalks LeagueN   
## 0.45158546 0.12900049 0.13737712 0.02908572 27.18227535   
## DivisionW PutOuts Assists Errors NewLeagueN   
## -91.63411299 0.19149252 0.04254536 -1.81244470 7.21208390

# Lasso  
  
#only difference in model building is to use aloha=1  
lasso.mod=glmnet(x[train,],y[train],alpha=1,lambda=grid)  
plot(lasso.mod)



# use CV to get best lambda  
set.seed(1)  
cv.out=cv.glmnet(x[train,],y[train],alpha=1)  
plot(cv.out)



bestlam=cv.out$lambda.min  
  
  
#use best lambda for prediction  
lasso.pred=predict(lasso.mod,s=bestlam,newx=x[test,])  
mean((lasso.pred-y.test)^2)

## [1] 100743.4

out=glmnet(x,y,alpha=1,lambda=grid)  
lasso.coef=predict(out,type="coefficients",s=bestlam)[1:20,]  
lasso.coef

## (Intercept) AtBat Hits HmRun Runs   
## 18.5394844 0.0000000 1.8735390 0.0000000 0.0000000   
## RBI Walks Years CAtBat CHits   
## 0.0000000 2.2178444 0.0000000 0.0000000 0.0000000   
## CHmRun CRuns CRBI CWalks LeagueN   
## 0.0000000 0.2071252 0.4130132 0.0000000 3.2666677   
## DivisionW PutOuts Assists Errors NewLeagueN   
## -103.4845458 0.2204284 0.0000000 0.0000000 0.0000000

lasso.coef[lasso.coef!=0]

## (Intercept) Hits Walks CRuns CRBI   
## 18.5394844 1.8735390 2.2178444 0.2071252 0.4130132   
## LeagueN DivisionW PutOuts   
## 3.2666677 -103.4845458 0.2204284