Comparison of model using Lift Chart

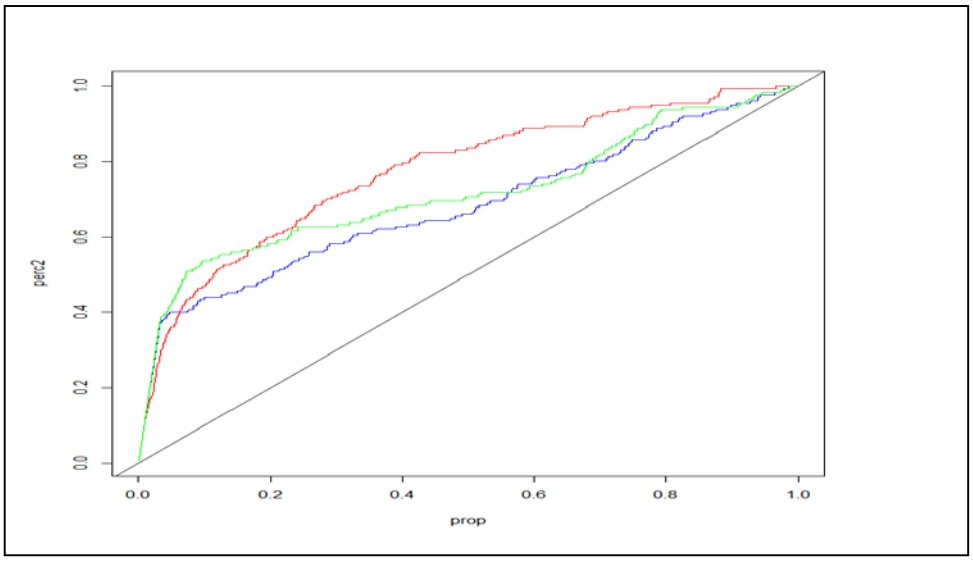
In Chapter 4, we introduce the Lift Chart to evaluate the performance of the logistic regression model. In fact, we can plot the Lift Charts of different models in the same graph for comparison. Let us rewrite the codes for producing Lift Chart in a R function.

```
# Produce lift chart
# input y=binary target, p=Prob{Y=1}
# version ver=1 or 2 (default),
# use add=T if add Lift Chart to existing plot
# color=col (default='black')

LChart<-function(y,p,ver=2,add=F,col='black') {
   ysort<-y[order(p,decreasing=T)]              # sort y according to p
   n<-length(y)                                 # get legnth of y
   ny<-sum(y)                                   # compute no. of 1 in y
   if (ver==1) {                                # version 1
      perc1<-cumsum(ysort)/(1:n)
      if (add==F) {                             # add=F, create a new plot
         plot(perc1,type='l',col=col)           # plot
         abline(h=ny/n) }                       # add reference line
      else                                      # add to existing plot
         lines(perc1,type='l',col=col) }

   else                                         # version 2
      perc2<-cumsum(ysort)/ny
      prop<-(1:n)/n
      if (add==F) {                             # add=F, create a new plot
         plot(prop,perc2,type='l',col=col)      # plot
         abline(a=0,b=1) }                      # add reference line
      else                                      # add to existing plot
         lines(prop,perc2,type='l',col=col)
}
```

The above function is save in a file “LC.r”. Let us use the hmeq1.csv data as an example, built the classification tree, logistic regression and ANN model respectively. Then plot the corresponding Lift Chart on the same graph.

```
# hmeq data
source("LChart.r")            # load LChart() function
d<-read.csv("hmeq1.csv")      # read in dataset
(n<-nrow(d))                  # get no. of row and display
names(d)                      # display var names

set.seed(123)                 # set the random seed
r<-2/3                        # set sampling ratio
id<-sample(1:n,size=round(r*n),replace=F)  # sample r*n random integers from 1 to
n
d1<-d[id,]                    # training dataset
d2<-d[-id,]                   # testing dataset
dim(d1)
dim(d2)
names(d)
y<-d1$BAD                     # create target y
```

```
#### CTREE #####
library(rpart)                          # load rpart library
# use minsplit and maxdepth to control the CTREE
ctree<-rpart(y~.,data=d1[,2:13],method="class",minsplit=40,maxdepth=4)

pr<-predict(ctree)                      # prediction on training dataset d1
cl<-max.col(pr)                         # find col. index of max in each row of pr
table(cl,d1$BAD)                        # classification table

ps=pr[,2]                               # Pr{Y=1}
LChart(y,ps,col='blue')                 # Lift chart
readline("Hit <Return> to continue:")

### logsitic regression ####
lreg<-glm(BAD~YOJ+DEROG+DELINQ+CLAGE+NINQ+DEBTINC,data=d1,binomial)
pr<-lreg$fit>0.5                        # pr=T if prob>0.5
table(pr,d1$BAD)                        # classification table

ps<-lreg$fit                            # Pr{Y=1}
LChart(y,ps,col='red',add=T)            # Lift chart
readline("Hit <Return> to continue:")

#### ANN ####
source('ann.r')                                 # use improved nnet
y1<-as.factor(d1[,1])                           # convert into factor
y2<-as.factor(d2[,1])

source("scale.r")
z<-scale.con(d[,c(8,9,13)])                     # using scale
z1<-z[id,]
z2<-z[-id,]

hmeq.nn<-ann(z1,y1,size=3,maxit=500,try=30)     # use logistic output
hmeq.nn$value                                   # display best value
summary(hmeq.nn)                                # summary of output
pred<-(hmeq.nn$fitted.values>0.5)               # prediction
table(pred,y1)                                  # classification table

ps<-hmeq.nn$fit                                 # Pr{Y=1}
LChart(y,ps,col='green',add=T)                  # Lift chart
```



Resampling methods

In Chapter 2, we introduced the random forest, which is used to assess the true performance of the classification tree. Note that this only gives a better assessment of the model rather than building a new improved model. Random forest is based on the bootstrap sample which is a resampling method. We have other similar approach, known as **Cross-validation (CV)**. The idea is simple but computational intensive. Let us introduce two commonly used CV methods, namely, Leave out one CV (LOOCV) and k-fold CV method.

1. LOOCV (Leave out one Cross-Validation)

Let us illustrate this method by logistic regression. This method is summarized as follow:

1. We leave out the first observation and use the remaining n-1 observation to build a logistic regression.
2. Apply the logistic regression in step 1 to classify the first observation.
3. Leave out the second observation instead of the first observation in step 1 and repeat step 1 and 2.
4. Continue to leave out one observation at a time, apply the logistic regression to classify the omitted observation, up to the last observation.

The following codes is taken from Chapter 4, arrived at a final model with backward elimination.

```
d<-read.csv("hmeq1.csv")          # read in dataset
(n<-nrow(d))                      # get and display sample size
names(d)                          # display names of d

set.seed(123)                     # set random seed
r<-2/3
id<-sample(1:n,size=round(r*n),replace=F)
d1<-d[id,]                        # training dataset
d2<-d[-id,]                       # testing dataset

# logistic regression
lreg<-glm(BAD~YOJ+DEROG+DELINQ+CLAGE+NINQ+DEBTINC,data=d1,binomial)

# using predict() function
pr<-predict(lreg,d2)              # use predict() on the testind data d2
prob<-exp(pr)/(1+exp(pr))         # compute pr using (4.3)
cl<-prob>0.5                      # create label for prediction
(t<-table(cl,d2$BAD))             # classification table
cl              0     1
   FALSE 912    68
   TRUE        9    33

1-sum(diag(t))/sum(t)             # testing error
[1] 0.07534247
```

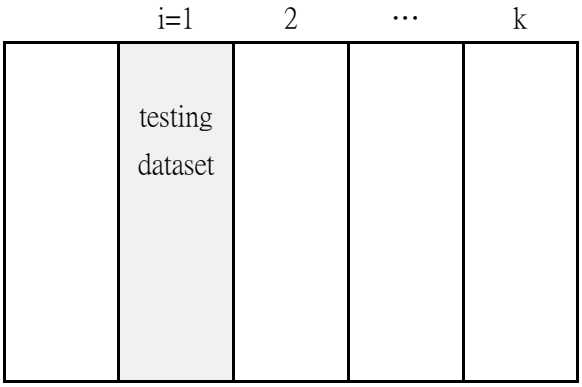
Now we implement the LOOCV as follow:

```
# LOOCV leave out one CV
pred<-rep(0,n)                          # initialize pred
for (i in 1:n) {                        # loop
   x1<-d[-i,]                           # leave out i as training
   x2<-d[i,]                            # obs i as testing data
   lreg<-glm(BAD~YOJ+DEROG+DELINQ+CLAGE+NINQ+DEBTINC,data=x1,binomial)
   pr<-predict(lreg,x2)                 # compute x'b
   prob<-exp(pr)/(1+exp(pr))            # compute prob
   pred[i]<-(prob>0.5)+0                # save pred
}

(t<-table(pred,d$BAD))                  # classification table
pred       0       1
     0 2768    199
     1      21      79

1-sum(diag(t))/sum(t)                   # LOOCV error
[1] 0.07173133
```

2. K-fold CV

An important different between CV and training-testing approach is that all data will have chance to build the model. However, LOOCV rather computational intensive especially when n is large. Instead of using LOOCV, we may partition the whole dataset into k random blocks and use one block (at a time) as testing dataset and the rest as training dataset. The basic idea is as follow:

k random partitions



1. We randomly partition the whole dataset into k (usually k=10 or 5) blocks.
2. Use the first block (i=1) as testing dataset and the rest as the training dataset, build a classification model. Then compute the compute the testing error, say err(1).
3. Repeat step 2 with i=2,…k. We should have k testing errors, err(1),…,err(k).
4. Compute the average of err(1),…, err(k) as the true performance. Usually the variance of these k testing errors is also computed as the accuracy of this k-fold CV.

Now we implement the k-fold CV as follow:

```
# k-fold CV
k<-10                              # no. of fold
set.seed(123)                      # set random seed

idx<-sample(1:n,size=n)            # create random vector idx
r<-n%%k                            # remainder of n/k

if (r==0) {                        # if n divisible by k
      mid<-matrix(idx,ncol=k)      # arrange idx into matrix of k columns
} else                             # else
      mid<-matrix(idx,ncol=(k+1))  # arrange idx into matrix of k+1 columns

nr<-nrow(mid)                      # no of row = size of testing dataset
err<-NULL                          # initialize err

for (i in 1:k) {
   id<-mid[,i]                     # extract mid[,i] as id
   x1<-d[-id,]                     # leave out id as training dataset
   x2<-d[id,]                      # testing dataset
   lreg<-glm(BAD~YOJ+DEROG+DELINQ+CLAGE+NINQ+DEBTINC,data=x1,binomial)

   pr<-predict(lreg,x2)            # compute x'b
   prob<-exp(pr)/(1+exp(pr))       # compute prob
   pred<-(prob>0.5)+0              # save pred
   t<-table(pred,x2$BAD)           # classification table
   err<-c(err,1-sum(diag(t))/sum(t))  # kfCV error
}

mean(err)                  # average k-fold errors
[1] 0.07168459

var(err)                   # variance of k-fold errors
[1] 0.0003625624
```

Note that k-fold CV requires much less computational time than LOOCV. Also, LOOCV is a special case of k-fold CV when k=n. In general, when k is large, we need more computing time; while k is small, the variance (accuracy) of these k testing errors may be large. A common choice for k is 10 or 5.

Finally, from this hmeq1 example, the testing error=0.07534247. Using LOOCV and k-fold CV, the true performance of this logistic regression is in fact a bit better, with LOOCV error=0.07173133 and k-fold CV error=0.07168459.