Risk Analysis using Data Analysis

George Kuliner

Agenda

1. Introduction
2. The Data Set
3. How we will Score the Models
4. Estimating Accuracy of Predicting the Real Probability of Default using Sorting Smoothing Method
5. Overview of the Results

Introduction

- Not Just prediction of who is going to default
- We want to be able to estimate the true population proportion of defaults with a known accuracy
- Gain insight from the data
- (Learn something new)
- Data was donated to UCI machine learning repository in 2016
- Originally comes from a Taiwanese Bank from mid 2000s
- Used by I-Cheng Yeh and Che-hui Lien in 2009 to test a novel approach of the Sorting Smoothing Method to test estimate and test accuracy of population proportion

The Data Set

- 23 Predictors
  - The Amount of Credit loaned
  - Sex
  - Education: scale 1-4
  - Marital Status
  - Age (year)
  - 6 Months of payment History: 1-9 scale of how late each monthly payment was (-1 was payed on time), What was on the Bill, Amount last payed
- 1 Response: Have they defaulted on payments

A Look at the Correlation

- cor(df[,1:24])[,24]

```
> cor(df[,1:24])[,24]
   LIMIT_BAL          SEX    EDUCATION     MARRIAGE          AGE        PAY_0        PAY_2        PAY_3
-0.153519876 -0.039960578  0.028006078 -0.024339216  0.013889834  0.324793728  0.263551202  0.235252514
       PAY_4        PAY_5        PAY_6    BILL_AMT1    BILL_AMT2    BILL_AMT3    BILL_AMT4    BILL_AMT5
 0.216613637  0.204148914  0.186866362 -0.019644197 -0.014193218 -0.014075518 -0.010156496 -0.006760464
   BILL_AMT6     PAY_AMT1     PAY_AMT2     PAY_AMT3     PAY_AMT4     PAY_AMT5     PAY_AMT6      DEFAULT
-0.005372315 -0.072929488 -0.058578707 -0.056250351 -0.056827401 -0.055123516 -0.053183340  1.000000000
```

Scoring the Models

```
#  George Kuliner
require(caret)
library(ggplot2)

#load data
df = read.csv("ccd.csv", header=TRUE)

#get rid of ID parameter
df=df[,2:25]

#geting sample size
smp_size=floor(.5*nrow(df))

#set see for reproducibility
set.seed(13);

#create random indecies
t_ind= sample(seq_len(nrow(df)) , size =smp_size)

#seperate data

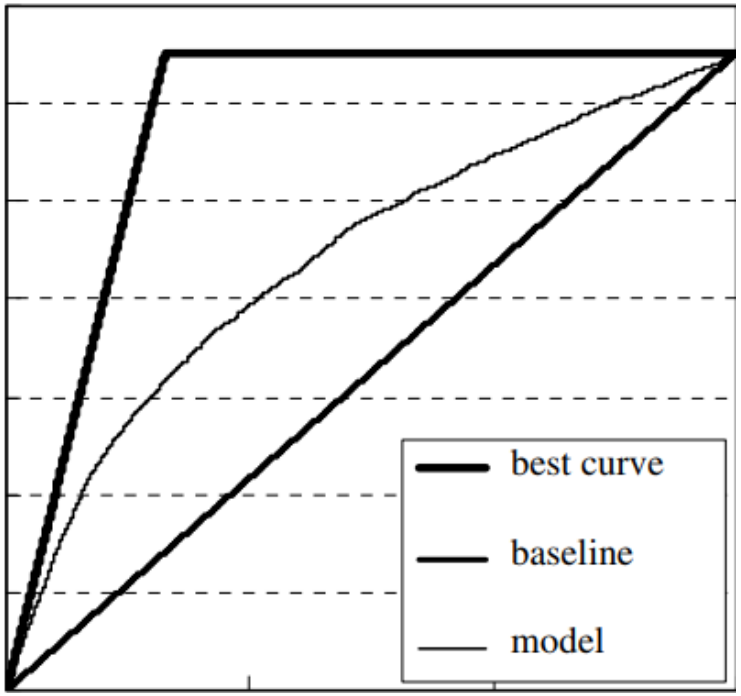
datat=df[t_ind, ]  #train data
datas=df[-t_ind, ]  #test data

#split X and y
X.train=datat[,1:23]
y.train=datat[,24]
Y=as.factor(y.train)
training=cbind(X.train,Y)
X.test=datas[,1:23]
y.test=datas[,24]
Y=as.factor(y.test)
testing=cbind(X.test,Y)
```

- To be able to see how the models performed a test data set was needed
- Normally a misclassification rate and confusion matrix would be very useful, but there is a high proportion of non defaulters
- Another method is needed to help make sense between
- Usually a ROC curve would be used to compare different classification models, but we cant

Scoring the Models

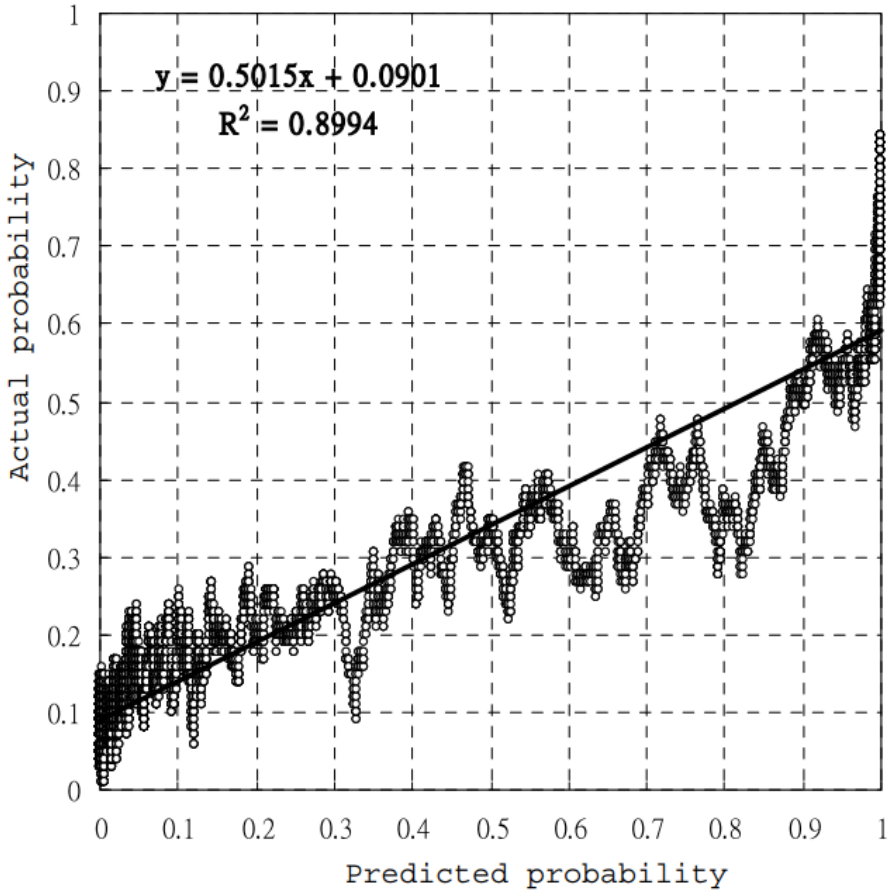
- We will be looking at Cumulative Gains
- Lift measures how good a model is at predicting
- Compares to if no model was used
- The cornered line is the best case, the straight line (baseline) is worst case
- The larger the area between the curve and the baseline the better
- Shows how well of capturing population proportions when adding to the sample




```
lift_results=data.frame(Y=testing$Y)
lift_results$lr=predict(lr.fit,newdata = testing, type='prob')[,1]
head(lift_results)
lift_obj=lift(Y~lr,data=lift_results)
ggplot(lift_obj)+ggtitle("Bayes Generalized Linear Model Lift Plot")+xlim(0,100)+ylim(0,100)

lift_obj
```

Scoring Models, Sorting Smoothing Method



$$P_i = \frac{\sum_{j=i-n}^{i+n} Y}{2n+1}$$

Scoring Models, Sorting Smoothing Method

```
#####Sorting Smoothing Method#########

n=50
m=6000

denom=2*n+1
library(data.table)
ac=0
pr=0
act=c()
pre=c()
probsum=cbind(predict(lr.fit,newdata = testing, type='prob')[,2],as.numeric(predict(lr.fit,newdata = testing))-1,as.numeric(Y)-1)
sortsm=data.table(probsum,key="V1")
for (i in 51:5950) {
  pr=0
  pr=sum(sortsm[(i-n):(i+n),"V2"])/denom
  ac=0
  ac=sum(sortsm[(i-n):(i+n),"V3"])/denom
  act=c(act, ac)
  pre=c(pre, pr)

}

plt=data.table(cbind(pre,act))
fit=lm(act ~ pre, data = plt)

ggplot(plt, aes(x = pre, y = act)) +
  geom_point() +
  stat_smooth(method = "lm", col = "red")
ggplotRegression(fit)
```

Bayes Generalized Logistic Regression

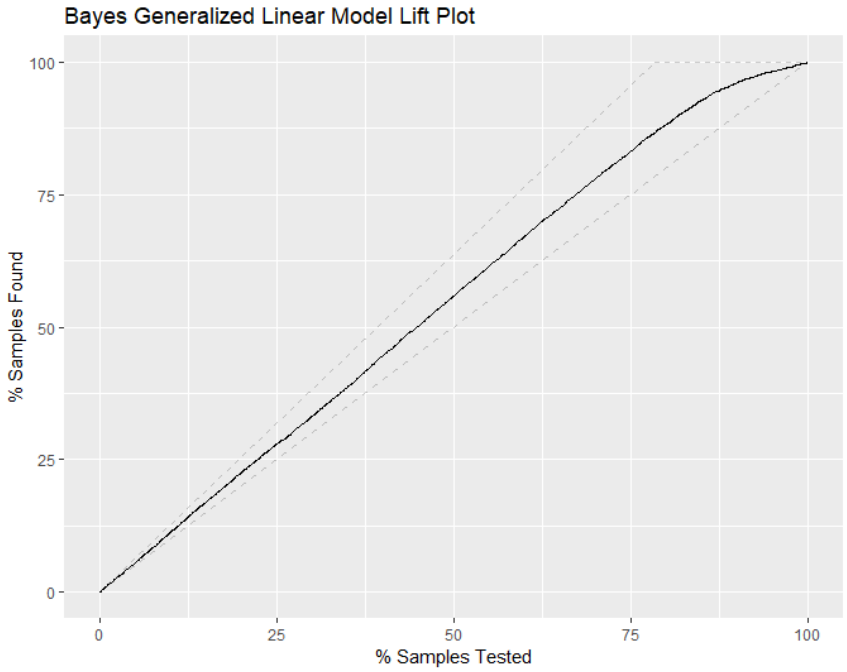
- $\frac{e^{\beta 0+\beta 1X1+\cdots+\beta pXp}}{1+e^{\beta 0+\beta 1X1+\cdots+\beta pXp}} = p(X)$ with X=(X_1,…,X_p) predictors
- Training accuracy =.808
- Testing Accuracy = .817
- Reduction in parameters brought up accuracy

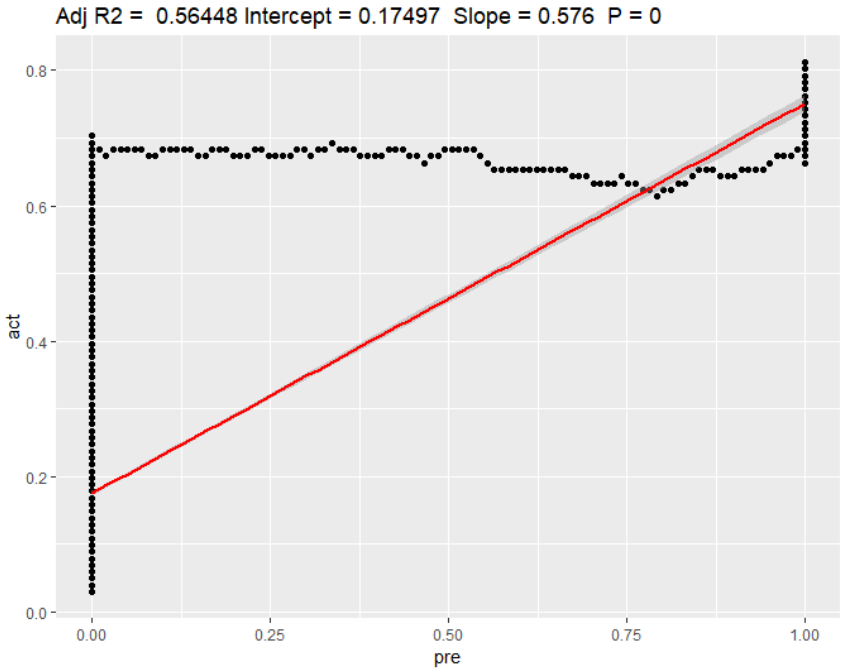
```
############Logistic Regression#########################
################Bayes Generalized Model############
#  method='bayesglm' library(arm)

fitControl=trainControl(##10-fold CV
  method="repeatedcv",
  number = 10,
  ## repeated 10 times
  repeats=10)

set.seed(13)
lr.fit=train(Y1~., data=training, method="bayesglm", trControl=fitControl)
lr.fit
lr.preds=predict(lr.fit,newdata = testing)
#classification rate
mean(lr.preds==Y)
```

Bayes Generalized Logistic Regression

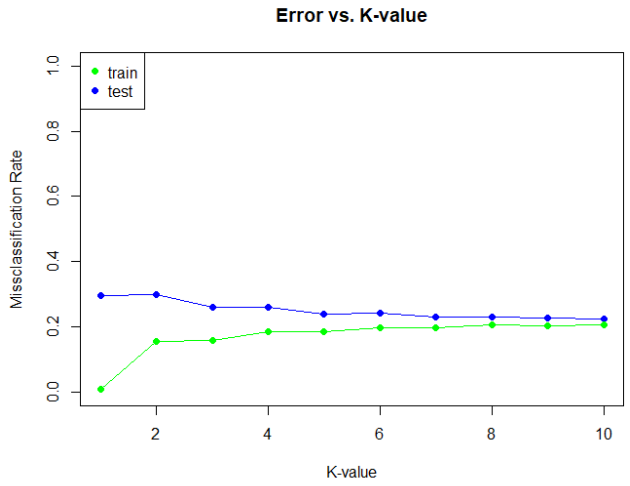




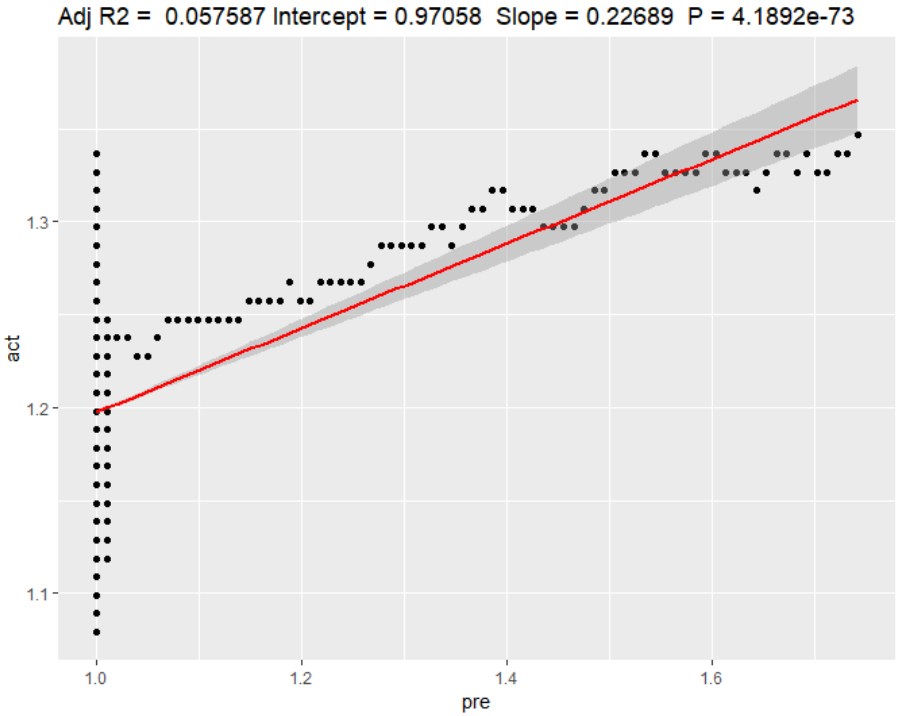
K-Nearest Neighbors

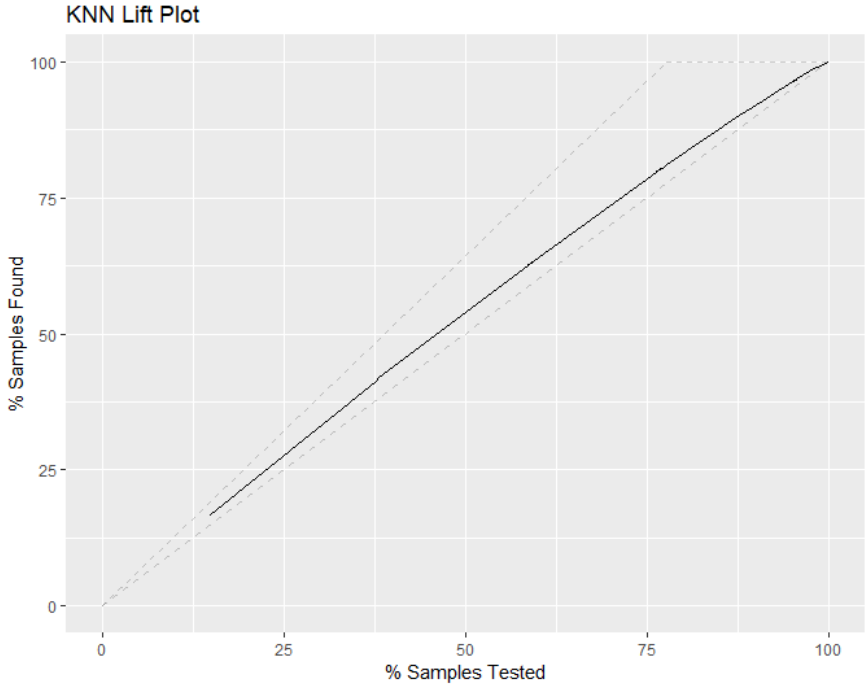
```
k.val=c()
erate.test=c()
erate.train=c()

for( i in 1:10){
  k.val=c(k.val,i)
  set.seed(12)
  knn.pred=knn(X.train,X.test,y.train,k=i)
  e1=1-mean(knn.pred==y.test)
  erate.test=c(erate.test,e1)
  set.seed(12)
  knn.pred=knn(X.train,X.train,y.train,k=i)
  e2=1-mean(knn.pred==y.train)
  erate.train=c(erate.train,e2)
}
plot(k.val,erate.test,col='blue', xlab="K-value",ylab="Missclassification Rate", main="Error vs. K-value",ylim=c(0,1),pch=19)
points(k.val,erate.train,col='green',pch=19)
lines(k.val,erate.test,col='blue')
lines(k.val,erate.train,col='green')
legend("topleft",legend=c('train','test'),col=c('green','blue'),pch=19)
```



K-Nearest Neighbors





Results

- Found higher error rates, this could be due to the selection of a smaller training size compared to I-Cheng Yeh and Che-hui Lien
- Similar results to the paper when using similar sample sizes