k-NN, decision trees, and naive Bayes for KDD Cup 2009

# 1. KDD Cup 2009 데이터

개요

* CRM(customer relationship management) 데이터
* <https://github.com/WinVector/zmPDSwR/tree/master/KDD2009> 등 에서 데이터 다운
* 50,000개의 신용카드 계정에 대한 250개의 변수 제공

분석목표

* 카드 해지(churn, 고객이탈)를 예측
* 새로운 제품이나 서비스를 이용하고자 하는 내재적 욕망(appetency)을 예측
* 마케팅에 호의적인 반응(upselling)을 예측

예제에서는 고객이탈을 예측하고자 하며 AUC를 이용하여 모형을 평가하고자 함. 참고로 우승팀의 AUC값은 0.76이었음

# 데이터 설명변수값 읽기  
setwd("D:/Dropbox/PDSwR/KDD2009/")  
d <- read.table('orange\_small\_train.data.gz', header=T,  
 sep='\t', na.strings=c('NA',''))   
# 반응변수 churn 읽어 데이터에 변수 삽입  
churn <- read.table('orange\_small\_train\_churn.labels.txt',  
 header=F, sep='\t')   
d$churn <- churn$V1   
# 반응변수 appetency 읽어 데이터에 변수 삽입  
appetency <- read.table('orange\_small\_train\_appetency.labels.txt',  
 header=F, sep='\t')  
d$appetency <- appetency$V1   
# 반응변수 upselling 읽어 데이터에 변수 삽입  
upselling <- read.table('orange\_small\_train\_upselling.labels.txt',  
 header=F, sep='\t')  
d$upselling <- upselling$V1   
  
# 데이터를 훈련과 시험 데이터로 분할  
set.seed(729375)   
d$rgroup <- runif(dim(d)[[1]])  
dTrainAll <- subset(d, rgroup<=0.9)  
dTest <- subset(d, rgroup>0.9)   
  
  
outcomes=c('churn','appetency','upselling')  
vars <- setdiff(colnames(dTrainAll), c(outcomes,'rgroup'))  
# 범주형 변수  
catVars <- vars[sapply(dTrainAll[,vars],class) %in%  
 c('factor','character')]   
# 숫자형 변수  
numericVars <- vars[sapply(dTrainAll[,vars],class) %in%  
 c('numeric','integer')]   
  
# 불필요한 객체 제거  
rm(list=c('d','churn','appetency','upselling'))   
# churn을 모형화 할것임  
outcome <- 'churn'   
pos <- '1'   
# 훈련데이터를 훈련과 검증으로 분할  
useForCal <- rbinom(n=dim(dTrainAll)[[1]], size=1, prob=0.1)>0   
dCal <- subset(dTrainAll,useForCal)  
dTrain <- subset(dTrainAll,!useForCal)

# 2. 일변수 모형

Var218의 수준별 churn

table218 <- table(  
 Var218=dTrain[,'Var218'],  
 churn=dTrain[,outcome],   
 useNA='ifany') # NA 포함  
print(table218)

## churn  
## Var218 -1 1  
## cJvF 19245 1220  
## UYBR 17860 1618  
## <NA> 423 152

# churn 비율  
print(table218[,2]/(table218[,1]+table218[,2]))

## cJvF UYBR <NA>   
## 0.05961398 0.08306808 0.26434783

NA에서 churn이 많고 원인을 알 수 없으므로 일단 NA를 레벨로 추가

범주형 변수에 대한 일변수 모형

# 함수: 훈련데이터의 결과 outCol, 훈련할 범주형변수 varCol, 예측할 변수 appCol  
# 주어진 경우 outCol과 varCol을 이용하여 일변수 모형을 만들고 appCol을 이용하여 예측  
mkPredC <- function(outCol,varCol,appCol) {   
 pPos <- sum(outCol==pos)/length(outCol)   
 naTab <- table(as.factor(outCol[is.na(varCol)]))  
 # NA에 대하여 positive인 비율  
 pPosWna <- (naTab/sum(naTab))[pos]   
 vTab <- table(as.factor(outCol),varCol)  
 # 레벨에 따라 positive인 비율  
 pPosWv <- (vTab[pos,]+1.0e-3\*pPos)/(colSums(vTab)+1.0e-3)   
 pred <- pPosWv[appCol]   
 pred[is.na(appCol)] <- pPosWna   
 pred[is.na(pred)] <- pPos   
 pred   
}  
  
  
for(v in catVars) {  
 pi <- paste('pred',v,sep='')  
 dTrain[,pi] <- mkPredC(dTrain[,outcome],dTrain[,v],dTrain[,v])  
 dCal[,pi] <- mkPredC(dTrain[,outcome],dTrain[,v],dCal[,v])  
 dTest[,pi] <- mkPredC(dTrain[,outcome],dTrain[,v],dTest[,v])  
}

AUC에 의한 범주형 변수의 스코어링

library('ROCR')

## Loading required package: gplots

##   
## Attaching package: 'gplots'

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

# AUC 계산 함수  
calcAUC <- function(predcol, outcol) {  
 perf <- performance(prediction(predcol, outcol==pos),'auc')  
 as.numeric(perf@y.values)  
 }  
  
for(v in catVars) {  
 pi <- paste('pred',v,sep='')  
 aucTrain <- calcAUC(dTrain[,pi], dTrain[,outcome])  
 aucCal <- calcAUC(dCal[,pi], dCal[,outcome])  
 print(sprintf("%s, trainAUC: %4.3f calibrationAUC: %4.3f",  
 pi, aucTrain, aucCal))  
 }

## [1] "predVar191, trainAUC: 0.505 calibrationAUC: 0.502"  
## [1] "predVar192, trainAUC: 0.619 calibrationAUC: 0.541"  
## [1] "predVar193, trainAUC: 0.557 calibrationAUC: 0.551"  
## [1] "predVar194, trainAUC: 0.513 calibrationAUC: 0.537"  
## [1] "predVar195, trainAUC: 0.511 calibrationAUC: 0.507"  
## [1] "predVar196, trainAUC: 0.502 calibrationAUC: 0.499"  
## [1] "predVar197, trainAUC: 0.576 calibrationAUC: 0.524"  
## [1] "predVar198, trainAUC: 0.749 calibrationAUC: 0.559"  
## [1] "predVar199, trainAUC: 0.770 calibrationAUC: 0.575"  
## [1] "predVar200, trainAUC: 0.830 calibrationAUC: 0.565"  
## [1] "predVar201, trainAUC: 0.513 calibrationAUC: 0.537"  
## [1] "predVar202, trainAUC: 0.827 calibrationAUC: 0.525"  
## [1] "predVar203, trainAUC: 0.507 calibrationAUC: 0.507"  
## [1] "predVar204, trainAUC: 0.572 calibrationAUC: 0.544"  
## [1] "predVar205, trainAUC: 0.541 calibrationAUC: 0.554"  
## [1] "predVar206, trainAUC: 0.587 calibrationAUC: 0.593"  
## [1] "predVar207, trainAUC: 0.550 calibrationAUC: 0.534"  
## [1] "predVar208, trainAUC: 0.504 calibrationAUC: 0.509"  
## [1] "predVar210, trainAUC: 0.519 calibrationAUC: 0.520"  
## [1] "predVar211, trainAUC: 0.526 calibrationAUC: 0.502"  
## [1] "predVar212, trainAUC: 0.581 calibrationAUC: 0.578"  
## [1] "predVar213, trainAUC: 0.506 calibrationAUC: 0.504"  
## [1] "predVar214, trainAUC: 0.830 calibrationAUC: 0.565"  
## [1] "predVar215, trainAUC: 0.501 calibrationAUC: 0.501"  
## [1] "predVar216, trainAUC: 0.680 calibrationAUC: 0.609"  
## [1] "predVar217, trainAUC: 0.897 calibrationAUC: 0.553"  
## [1] "predVar218, trainAUC: 0.561 calibrationAUC: 0.535"  
## [1] "predVar219, trainAUC: 0.512 calibrationAUC: 0.524"  
## [1] "predVar220, trainAUC: 0.749 calibrationAUC: 0.559"  
## [1] "predVar221, trainAUC: 0.538 calibrationAUC: 0.540"  
## [1] "predVar222, trainAUC: 0.749 calibrationAUC: 0.559"  
## [1] "predVar223, trainAUC: 0.505 calibrationAUC: 0.502"  
## [1] "predVar224, trainAUC: 0.502 calibrationAUC: 0.504"  
## [1] "predVar225, trainAUC: 0.554 calibrationAUC: 0.576"  
## [1] "predVar226, trainAUC: 0.553 calibrationAUC: 0.539"  
## [1] "predVar227, trainAUC: 0.548 calibrationAUC: 0.536"  
## [1] "predVar228, trainAUC: 0.567 calibrationAUC: 0.550"  
## [1] "predVar229, trainAUC: 0.556 calibrationAUC: 0.567"

변수 206와 같이 훈련과 검증 AUC의 차이가 많이 나지 않고 검증 AUC가 큰 변수가 좋을 듯

AUC에 의한 수치형변수들의 스코어링

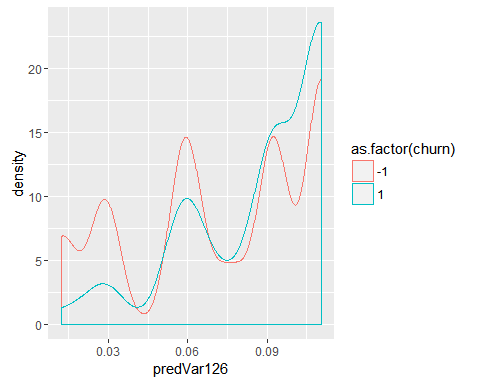
# 0.1간격의 분위수로 범주화하여 예측하는 함수  
mkPredN <- function(outCol,varCol,appCol) {  
 cuts <- unique(as.numeric(quantile(varCol,  
 probs=seq(0, 1, 0.1),na.rm=T)))  
 varC <- cut(varCol,cuts)  
 appC <- cut(appCol,cuts)  
 mkPredC(outCol,varC,appC)  
}  
  
for(v in numericVars) {  
 pi <- paste('pred',v,sep='')  
 dTrain[,pi] <- mkPredN(dTrain[,outcome],dTrain[,v],dTrain[,v])  
 dTest[,pi] <- mkPredN(dTrain[,outcome],dTrain[,v],dTest[,v])  
 dCal[,pi] <- mkPredN(dTrain[,outcome],dTrain[,v],dCal[,v])  
 aucTrain <- calcAUC(dTrain[,pi],dTrain[,outcome])  
 if(aucTrain>=0.55) {  
 aucCal <- calcAUC(dCal[,pi],dCal[,outcome])  
 print(sprintf("%s, trainAUC: %4.3f calibrationAUC: %4.3f",  
 pi,aucTrain,aucCal))  
 }  
 }

## [1] "predVar6, trainAUC: 0.557 calibrationAUC: 0.554"  
## [1] "predVar7, trainAUC: 0.555 calibrationAUC: 0.565"  
## [1] "predVar13, trainAUC: 0.568 calibrationAUC: 0.553"  
## [1] "predVar73, trainAUC: 0.608 calibrationAUC: 0.616"  
## [1] "predVar74, trainAUC: 0.574 calibrationAUC: 0.566"  
## [1] "predVar81, trainAUC: 0.558 calibrationAUC: 0.542"  
## [1] "predVar113, trainAUC: 0.557 calibrationAUC: 0.567"  
## [1] "predVar126, trainAUC: 0.635 calibrationAUC: 0.629"  
## [1] "predVar140, trainAUC: 0.561 calibrationAUC: 0.560"  
## [1] "predVar189, trainAUC: 0.574 calibrationAUC: 0.599"

변수 126의 경우 훈련과 검증 AUC가 0.645와 0.629로 좋음

변수 126의 churn의 값에 따른 확률밀도 추정값 비교

library(ggplot2)  
ggplot(data=dCal) +  
 geom\_density(aes(x=predVar126,color=as.factor(churn)))

 낮은 값에서는 churn이 적게 일어나고 높은 값에서는 많이 일어나는 것으로 보임

오렌지는이탈x, 파랑이 이탈… 즉…126번변수가 커지면 커질수록 이탈율 커짐..

# 3. 다변수 모형

## 3.1. 변수선택

# 로그 가능도 계산 함수  
logLikelyhood <- function(outCol,predCol) {   
 sum(ifelse(outCol==pos,log(predCol),log(1-predCol)))  
}  
  
# deviance에 기반하여 변수 선택  
selVars <- c()  
minStep <- 5  
baseRateCheck <- logLikelyhood(dCal[,outcome],  
 sum(dCal[,outcome]==pos)/length(dCal[,outcome]))  
  
# 범주형   
for(v in catVars) {   
 pi <- paste('pred',v,sep='')  
 liCheck <- 2\*((logLikelyhood(dCal[,outcome],dCal[,pi]) -  
 baseRateCheck))  
 if(liCheck>minStep) {  
 print(sprintf("%s, calibrationScore: %g",  
 pi,liCheck))  
 selVars <- c(selVars,pi)  
 }  
}

## [1] "predVar194, calibrationScore: 5.25759"  
## [1] "predVar201, calibrationScore: 5.25521"  
## [1] "predVar204, calibrationScore: 5.37414"  
## [1] "predVar205, calibrationScore: 24.2323"  
## [1] "predVar206, calibrationScore: 34.4434"  
## [1] "predVar210, calibrationScore: 10.6681"  
## [1] "predVar212, calibrationScore: 6.23409"  
## [1] "predVar218, calibrationScore: 13.2455"  
## [1] "predVar221, calibrationScore: 12.4098"  
## [1] "predVar225, calibrationScore: 22.9074"  
## [1] "predVar226, calibrationScore: 6.68931"  
## [1] "predVar228, calibrationScore: 15.9644"  
## [1] "predVar229, calibrationScore: 24.4946"

# 수치형   
for(v in numericVars) {   
 pi <- paste('pred',v,sep='')  
 liCheck <- 2\*((logLikelyhood(dCal[,outcome],dCal[,pi]) -  
 baseRateCheck))  
 if(liCheck>=minStep) {  
 print(sprintf("%s, calibrationScore: %g",  
 pi,liCheck))  
 selVars <- c(selVars,pi)  
 }  
}

## [1] "predVar6, calibrationScore: 13.2431"  
## [1] "predVar7, calibrationScore: 18.685"  
## [1] "predVar13, calibrationScore: 10.0632"  
## [1] "predVar28, calibrationScore: 11.3864"  
## [1] "predVar65, calibrationScore: 9.96938"  
## [1] "predVar72, calibrationScore: 12.5353"  
## [1] "predVar73, calibrationScore: 48.2524"  
## [1] "predVar74, calibrationScore: 19.6324"  
## [1] "predVar81, calibrationScore: 8.8741"  
## [1] "predVar113, calibrationScore: 23.136"  
## [1] "predVar125, calibrationScore: 6.06029"  
## [1] "predVar126, calibrationScore: 74.9556"  
## [1] "predVar134, calibrationScore: 5.68144"  
## [1] "predVar140, calibrationScore: 16.1816"  
## [1] "predVar144, calibrationScore: 15.9858"  
## [1] "predVar189, calibrationScore: 42.3059"

## 3.2. 의사결정나무

library(rpart)  
  
# 정제되지 않은 모든 원변수를 이용한 의사결정나무  
fV <- paste(outcome,'>0 ~ ',  
 paste(c(catVars,numericVars), collapse=' + '), sep='')  
tmodel <- rpart(fV, data=dTrain)  
print(calcAUC(predict(tmodel,newdata=dTrain),dTrain[,outcome]))

## [1] 0.9241265

print(calcAUC(predict(tmodel,newdata=dTest),dTest[,outcome]))

## [1] 0.5266172

print(calcAUC(predict(tmodel,newdata=dCal),dCal[,outcome]))

## [1] 0.5126917

# bad => NA의 문제?  
  
# 정제된 모든 원변수를 이용한 의사결정나무  
tVars <- paste('pred', c(catVars, numericVars), sep='')  
fV2 <- paste(outcome,'>0 ~ ', paste(tVars, collapse=' + '), sep='')  
tmodel <- rpart(fV2, data=dTrain)  
print(calcAUC(predict(tmodel,newdata=dTrain),dTrain[,outcome]))

## [1] 0.928669

print(calcAUC(predict(tmodel,newdata=dTest),dTest[,outcome]))

## [1] 0.5390648

print(calcAUC(predict(tmodel,newdata=dCal),dCal[,outcome]))

## [1] 0.5384152

# bad => overfitting의 문제?  
  
# 정제된 원변수를 이용한 의사결정나무(가지치기 등 복잡도 조절)  
tmodel <- rpart(fV2, data=dTrain,  
 control=rpart.control(cp=0.001, minsplit=1000,  
 minbucket=1000, maxdepth=5)  
 )  
print(calcAUC(predict(tmodel,newdata=dTrain),dTrain[,outcome]))

## [1] 0.9421195

print(calcAUC(predict(tmodel,newdata=dTest),dTest[,outcome]))

## [1] 0.5794633

print(calcAUC(predict(tmodel,newdata=dCal),dCal[,outcome]))

## [1] 0.547967

# 조금 나아지긴 했어도 bad  
  
# 앞에서 선택된 변수들을 이용하여 모형구성(가지치기 등)  
f <- paste(outcome,'>0 ~ ', paste(selVars, collapse=' + '), sep='')  
tmodel <- rpart(f, data=dTrain,  
 control=rpart.control(cp=0.001, minsplit=1000,  
 minbucket=1000, maxdepth=5)  
 )  
print(calcAUC(predict(tmodel,newdata=dTrain),dTrain[,outcome]))

## [1] 0.6906852

print(calcAUC(predict(tmodel,newdata=dTest),dTest[,outcome]))

## [1] 0.6843595

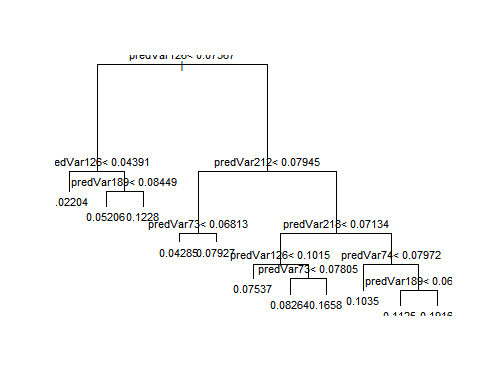
print(calcAUC(predict(tmodel,newdata=dCal),dCal[,outcome]))

## [1] 0.6669301

# 향상된 결과  
  
# 최종 모형 살펴보기  
print(tmodel)

## n= 40518   
##   
## node), split, n, deviance, yval  
## \* denotes terminal node  
##   
## 1) root 40518 2769.3550 0.07379436   
## 2) predVar126< 0.07366888 18188 726.4097 0.04167583   
## 4) predVar126< 0.04391312 8804 189.7251 0.02203544 \*  
## 5) predVar126>=0.04391312 9384 530.1023 0.06010230   
## 10) predVar189< 0.08449448 8317 410.4571 0.05206204 \*  
## 11) predVar189>=0.08449448 1067 114.9166 0.12277410 \*  
## 3) predVar126>=0.07366888 22330 2008.9000 0.09995522   
## 6) predVar212< 0.07944508 8386 484.2499 0.06153112   
## 12) predVar73< 0.06813291 4084 167.5012 0.04285015 \*  
## 13) predVar73>=0.06813291 4302 313.9705 0.07926546 \*  
## 7) predVar212>=0.07944508 13944 1504.8230 0.12306370   
## 14) predVar218< 0.07134103 6728 580.7390 0.09542212   
## 28) predVar126< 0.1015407 3901 271.8426 0.07536529 \*  
## 29) predVar126>=0.1015407 2827 305.1617 0.12309870   
## 58) predVar73< 0.07804522 1452 110.0826 0.08264463 \*  
## 59) predVar73>=0.07804522 1375 190.1935 0.16581820 \*  
## 15) predVar218>=0.07134103 7216 914.1502 0.14883590   
## 30) predVar74< 0.0797246 2579 239.3579 0.10352850 \*  
## 31) predVar74>=0.0797246 4637 666.5538 0.17403490   
## 62) predVar189< 0.06775545 1031 102.9486 0.11251210 \*  
## 63) predVar189>=0.06775545 3606 558.5871 0.19162510 \*

par(cex=0.7)  
plot(tmodel)  
text(tmodel)



## 3.3. k-NN

값?

* churn의 비율이 7%이므로 각 근방마다 10개정도 churn이 나타나려면 10/.07=142

선택된 변수들을 이용하여 k-NN()

library(class)  
nK <- 200  
knnTrain <- dTrain[,selVars] # 선택된 변수만 이용  
knnCl <- dTrain[,outcome]==pos   
# knn 훈련함수  
knnPred <- function(df) {   
 knnDecision <- knn(knnTrain, df, knnCl, k=nK, prob=T)  
 # majority voting 대신 확률값으로   
 ifelse(knnDecision==TRUE,   
 attributes(knnDecision)$prob, 1-(attributes(knnDecision)$prob))  
}  
print(calcAUC(knnPred(dTrain[,selVars]),dTrain[,outcome]))

## [1] 0.7437617

print(calcAUC(knnPred(dCal[,selVars]),dCal[,outcome]))

## [1] 0.7131476

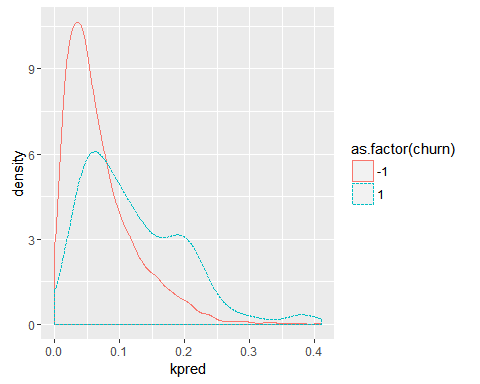
print(calcAUC(knnPred(dTest[,selVars]),dTest[,outcome]))

## [1] 0.7179175

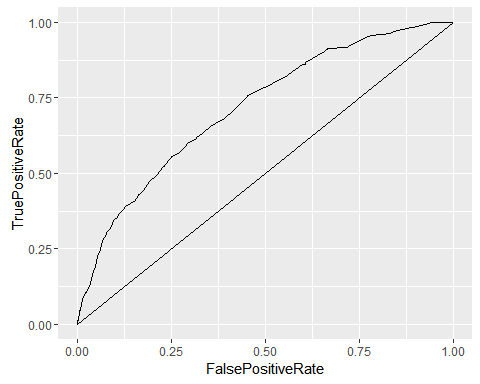
# 결과가 괜찮음

검증 데이터에서의 성능

# 확률밀도함수 비교  
dCal$kpred <- knnPred(dCal[,selVars])  
ggplot(data=dCal) +  
 geom\_density(aes(x=kpred,  
 color=as.factor(churn),linetype=as.factor(churn)))



# ROC 함수  
plotROC <- function(predcol,outcol) {  
 perf <- performance(prediction(predcol,outcol==pos),'tpr','fpr')  
 pf <- data.frame(  
 FalsePositiveRate=perf@x.values[[1]],  
 TruePositiveRate=perf@y.values[[1]])  
 ggplot() +  
 geom\_line(data=pf,aes(x=FalsePositiveRate,y=TruePositiveRate)) +  
 geom\_line(aes(x=c(0,1),y=c(0,1)))  
}  
print(plotROC(knnPred(dTest[,selVars]),dTest[,outcome]))



## 3.4. 단순베이즈 분류

pPos <- sum(dTrain[,outcome]==pos)/length(dTrain[,outcome])  
# 단순베이즈 함수 작성  
nBayes <- function(pPos,pf) { # Note: 1   
 pNeg <- 1 - pPos  
 smoothingEpsilon <- 1.0e-5 # 라플라스 평활량   
 scorePos <- log(pPos + smoothingEpsilon) +   
 rowSums(log(pf/pPos + smoothingEpsilon))   
 scoreNeg <- log(pNeg + smoothingEpsilon) +  
 rowSums(log((1-pf)/(1-pPos) + smoothingEpsilon))   
 m <- pmax(scorePos,scoreNeg)  
 expScorePos <- exp(scorePos-m)  
 expScoreNeg <- exp(scoreNeg-m)   
 expScorePos/(expScorePos+expScoreNeg)   
}  
pVars <- paste('pred', c(numericVars, catVars), sep='')  
dTrain$nbpredl <- nBayes(pPos, dTrain[,pVars])  
dCal$nbpredl <- nBayes(pPos, dCal[,pVars])  
dTest$nbpredl <- nBayes(pPos, dTest[,pVars])   
print(calcAUC(dTrain$nbpredl,dTrain[,outcome]))

## [1] 0.9757348

print(calcAUC(dCal$nbpredl,dCal[,outcome]))

## [1] 0.5995206

print(calcAUC(dTest$nbpredl,dTest[,outcome]))

## [1] 0.5956515

과적합됨

e1071패키지를 이용한 경우

library(e1071)  
lVars <- c(catVars, numericVars)  
ff <- paste('as.factor(',outcome,'>0) ~ ',  
 paste(lVars, collapse=' + '), sep='')  
nbmodel <- naiveBayes(as.formula(ff), data=dTrain)  
dTrain$nbpred <- predict(nbmodel, newdata=dTrain,type='raw')[,'TRUE']  
dCal$nbpred <- predict(nbmodel, newdata=dCal,type='raw')[,'TRUE']  
dTest$nbpred <- predict(nbmodel, newdata=dTest,type='raw')[,'TRUE']  
calcAUC(dTrain$nbpred, dTrain[,outcome])

## [1] 0.4643591

calcAUC(dCal$nbpred, dCal[,outcome])

## [1] 0.5544484

calcAUC(dTest$nbpred, dTest[,outcome])

## [1] 0.5679519

# 연습문제

선택된 변수를 이용하여 단순베이즈 분류를 실시하고 의사결정나무, k-NN, 단순베이즈 분류에서 최적인 모형들의 ROC와 AUC값을 통해 성능을 비교하시오.