Ensemble for Spambase data

# 1. Spambase 데이터

약 4,600개의 문서와 특정 키워드와 문자의 빈도를 나타내는 57개의 특성치값으로 구성

<https://github.com/WinVector/zmPDSwR/raw/master/Spambase/> 에서 spamD.tsv를 다운 받을 수 있음

setwd("D:/Dropbox/PDSwR/Spambase/")  
  
# 데이터를 읽어서 훈련과 시험 데이터로 분할   
spamD <- read.table('spamD.tsv',header=T,sep='\t')   
spamTrain <- subset(spamD,spamD$rgroup>=10)  
spamTest <- subset(spamD,spamD$rgroup<10)  
  
# spam인 경우 모든 변수를 사용하도록  
spamVars <- setdiff(colnames(spamD),list('rgroup','spam'))  
spamFormula <- as.formula(paste('spam=="spam"',   
 paste(spamVars,collapse=' + '),sep=' ~ '))  
  
# 로그 우도 함수  
loglikelihood <- function(y, py) {   
 pysmooth <- ifelse(py==0, 1e-12,  
 ifelse(py==1, 1-1e-12, py))  
  
 sum(y \* log(pysmooth) + (1-y)\*log(1 - pysmooth))  
}  
  
  
# 정확성 측도 함수  
# 정규화된 deviance, 예측 정확도, fl = precision\*recall  
accuracyMeasures <- function(pred, truth, name="model") {   
 dev.norm <- -2\*loglikelihood(as.numeric(truth), pred)/length(pred)   
 ctable <- table(truth=truth,  
 pred=(pred>0.5))   
 accuracy <- sum(diag(ctable))/sum(ctable)  
 precision <- ctable[2,2]/sum(ctable[,2])  
 recall <- ctable[2,2]/sum(ctable[2,])  
 f1 <- 2\*precision\*recall/(precision+recall)  
 data.frame(model=name, accuracy=accuracy, f1=f1, dev.norm)  
}  
  
## 단일 의사결정나무  
library(rpart)   
treemodel <- rpart(spamFormula, spamTrain)  
accuracyMeasures(predict(treemodel, newdata=spamTrain),   
 spamTrain$spam=="spam",  
 name="tree, training")

## model accuracy f1 dev.norm  
## 1 tree, training 0.9104514 0.88337 0.5618654

accuracyMeasures(predict(treemodel, newdata=spamTest),  
 spamTest$spam=="spam",  
 name="tree, test")

## model accuracy f1 dev.norm  
## 1 tree, test 0.8799127 0.8414986 0.6702857

# 2. 배깅

ntrain <- dim(spamTrain)[1]  
n <- ntrain   
ntree <- 100  
  
# 붓스트랩 표본 추출 반복  
samples <- sapply(1:ntree,   
 FUN = function(iter)  
 {sample(1:ntrain, size=n, replace=T)})  
# 각 붓스트랩 표본에 대하여 의사결정나무 적합  
treelist <-lapply(1:ntree,   
 FUN=function(iter)  
 {samp <- samples[,iter];  
 rpart(spamFormula, spamTrain[samp,])})  
# 배깅에 의한 확률 예측값 함수  
predict.bag <- function(treelist, newdata) {   
 preds <- sapply(1:length(treelist),  
 FUN=function(iter) {  
 predict(treelist[[iter]], newdata=newdata)})  
 predsums <- rowSums(preds)  
 predsums/length(treelist)  
}  
  
# 훈련데이터에서의 평가  
accuracyMeasures(predict.bag(treelist, newdata=spamTrain),   
 spamTrain$spam=="spam",  
 name="bagging, training")

## model accuracy f1 dev.norm  
## 1 bagging, training 0.9213131 0.8971609 0.4719123

# 시험데이터에서의 평가  
accuracyMeasures(predict.bag(treelist, newdata=spamTest),  
 spamTest$spam=="spam",  
 name="bagging, test")

## model accuracy f1 dev.norm  
## 1 bagging, test 0.9126638 0.8816568 0.5329826

단일 의사결정나무보다 예측력 향상

# 3. 랜덤포레스트

library(randomForest)

## randomForest 4.6-12

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

set.seed(5123512)   
  
fmodel <- randomForest(x=spamTrain[,spamVars],   
 y=spamTrain$spam,  
 ntree=100, # 100 trees  
 nodesize=7, # 노드에 최소 7개 데이터   
 importance=T) # 변수의 중요도  
  
accuracyMeasures(predict(fmodel,   
 newdata=spamTrain[,spamVars], type='prob')[,'spam'],  
 spamTrain$spam=="spam",name="random forest, train")

## model accuracy f1 dev.norm  
## 1 random forest, train 0.9884142 0.9851943 0.1428786

accuracyMeasures(predict(fmodel,  
 newdata=spamTest[,spamVars],type='prob')[,'spam'],  
 spamTest$spam=="spam",name="random forest, test")

## model accuracy f1 dev.norm  
## 1 random forest, test 0.9541485 0.9401709 0.3972416

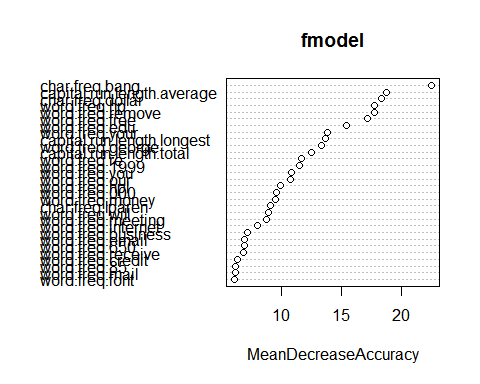
랜덤 포레스트는 배깅보다 예측력 좋음

변수의 중요도

varImp <- importance(fmodel)   
varImp[1:10, ]

## non-spam spam MeanDecreaseAccuracy  
## word.freq.make 2.096811 3.7304353 4.334207  
## word.freq.address 3.603167 3.9967031 4.977452  
## word.freq.all 2.799456 4.9527834 4.924958  
## word.freq.3d 3.000273 0.4125932 2.917972  
## word.freq.our 9.037946 7.9421391 10.731509  
## word.freq.over 5.879377 4.2402613 5.751371  
## word.freq.remove 16.637390 13.9331691 17.753122  
## word.freq.internet 7.301055 4.4458342 7.947515  
## word.freq.order 3.937897 4.3587883 4.866540  
## word.freq.mail 5.022432 3.4701224 6.103929  
## MeanDecreaseGini  
## word.freq.make 5.877954  
## word.freq.address 10.081640  
## word.freq.all 23.524720  
## word.freq.3d 1.550635  
## word.freq.our 52.569163  
## word.freq.over 11.820391  
## word.freq.remove 174.126926  
## word.freq.internet 22.578106  
## word.freq.order 11.809265  
## word.freq.mail 11.127200

varImpPlot(fmodel, type=1)



## 연습문제

변수의 중요도를 기준으로 적절히 변수를 선택한 후 배깅과 부스팅을 실시하고 변수를 선택하기 이전의 결과와 비교해보시오.