decition\_tree.R

SANGHOOJEFFREY

Thu Jun 28 01:30:34 2018

# 분류함수로 logistic regression외 다른 기계학습방법(데이터마이닝, 빅데이터방법)이 존재  
  
# 1. 의사결정나무 (Decision Tree)  
# 의사결정나무는 지니 불순도(Gini Impurity) 또는 정보이득 (Information Gain) 등의 기준을 사용하여  
# 노드를 재귀적으로 분할하면서 나무 모형을 만든다.   
  
# example  
if(!require(rpart)) install.packages("rpart"); library(rpart)

## Loading required package: rpart

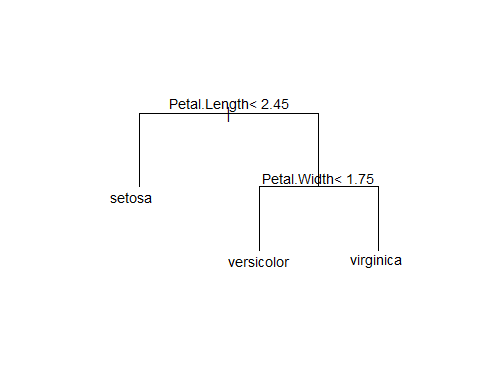
data(iris)  
dt <- rpart(Species ~., data=iris)  
table(iris$Species)

##   
## setosa versicolor virginica   
## 50 50 50

dt

## n= 150   
##   
## node), split, n, loss, yval, (yprob)  
## \* denotes terminal node  
##   
## 1) root 150 100 setosa (0.33333333 0.33333333 0.33333333)   
## 2) Petal.Length< 2.45 50 0 setosa (1.00000000 0.00000000 0.00000000) \*  
## 3) Petal.Length>=2.45 100 50 versicolor (0.00000000 0.50000000 0.50000000)   
## 6) Petal.Width< 1.75 54 5 versicolor (0.00000000 0.90740741 0.09259259) \*  
## 7) Petal.Width>=1.75 46 1 virginica (0.00000000 0.02173913 0.97826087) \*

# 총 150개의 자료이며 기본적으로 각 종의 비율을 1/3, 1/3, 1/3  
# 2) 기준에 의해 즉 Petal.Length<2.45로 setosa가 판별됨  
# setosa 50개 분류  
# 6) Petal.width <1.75를 기준으로 54개의 versicolor라 분류 (분류가 잘못된 자료 5개)  
# 7) 로 verginica 46개 예측 (분류가 잘못된 자료 1개)  
  
# plot(dt)  
plot(dt, compress=TRUE, margin=.2)  
# compress로 나무를 좀 더 조밀하게 그릴 수 있다.  
  
text(dt, cex=0.9)



# cex는 글자의 크기를 의미  
  
pred.dt<-predict(dt, newdat=iris, type="class")   
  
xtabs(~pred.dt+iris$Species)

## iris$Species  
## pred.dt setosa versicolor virginica  
## setosa 50 0 0  
## versicolor 0 49 5  
## virginica 0 1 45

# Petal.Length<2.45, Petal.width<1.75 가 노드(Node)  
# 노드 수 결정   
# 가지치기 : 오분류 가능성이 높아지는 경우 노드를 정지  
# 언제까지 나무모형을 성장시킬 것이냐?  
# 너무 큰 나무는 자료를 과대적합하고, 반대로 너무 작은 나무는 자료를 과소 적합  
# 일반적으로 사용되는 방법은 마디에 속하는 자료가 일정 수 이하일 때 분할 정시 (thumb rule)  
  
# another package  
library(MASS)  
if(!require(tree)) install.packages("tree");library(tree)

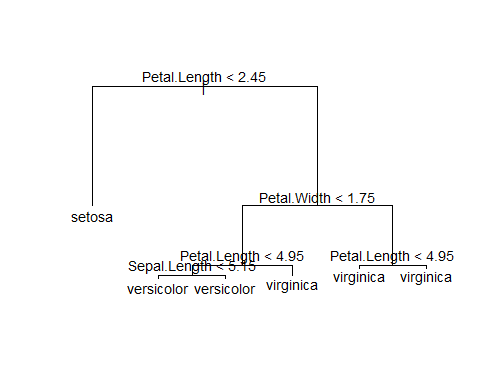
## Loading required package: tree

## Warning: package 'tree' was built under R version 3.4.4

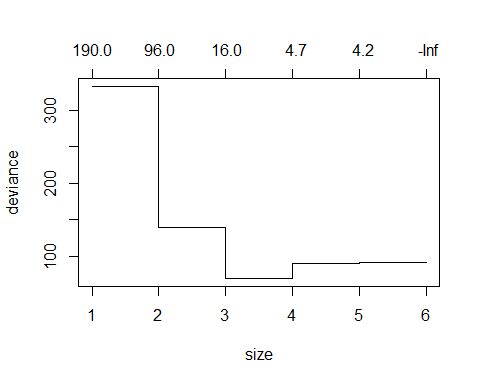
tree.iris <- tree(Species ~., data=iris)  
summary(tree.iris)

##   
## Classification tree:  
## tree(formula = Species ~ ., data = iris)  
## Variables actually used in tree construction:  
## [1] "Petal.Length" "Petal.Width" "Sepal.Length"  
## Number of terminal nodes: 6   
## Residual mean deviance: 0.1253 = 18.05 / 144   
## Misclassification error rate: 0.02667 = 4 / 150

plot(tree.iris)  
text(tree.iris, cex=0.9)



# 과적합화의 문제가 있으므로 가지치기단계가 필요함  
  
cv.tree.out<-cv.tree(tree.iris,FUN=prune.tree, K = 5 )  
plot(cv.tree.out)



tree.iris2<-prune.tree(tree.iris, best=4)  
plot(tree.iris2)  
text(tree.iris2, cex=0.9)  
  
# 의사결정나무의 장단점  
# 장점 : 설명력이 높다.  
# 결과에 대한 근거를 나무가지 형태로 쉽게 추적할 수 있다.  
# 계산이 빠르고 변수 선택 능력이 있다.  
  
# 단점 : 반응변수가 연속형일 때 사용하기 어렵다.  
# 설명변수가 연속형일 때 낮은 예측능력을 보일 수 있다.   
# 자료가 추가되면 나무구조가 바뀔 수 있다.  
# 비선형데이터는 나무모형으로 잘 설명되지 않는다.  
  
# 조건부 추론 나무는 조건부 분포로 분류하는 방법으로 과적합의 문제를 해결해준다.  
  
if(!require(party)) install.packages("party"); library(party)

## Loading required package: party

## Warning: package 'party' was built under R version 3.4.4

## Loading required package: grid

## Loading required package: mvtnorm

## Warning: package 'mvtnorm' was built under R version 3.4.3

## Loading required package: modeltools

## Loading required package: stats4

## Loading required package: strucchange

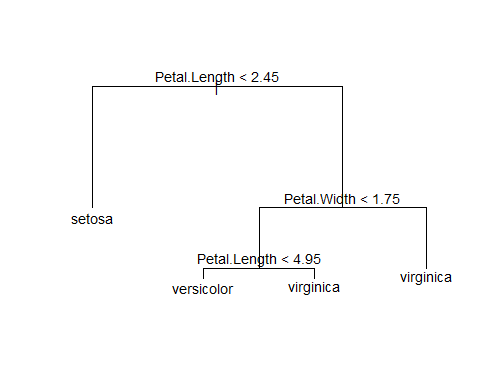
## Loading required package: zoo

## Warning: package 'zoo' was built under R version 3.4.4

##   
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':  
##   
## as.Date, as.Date.numeric

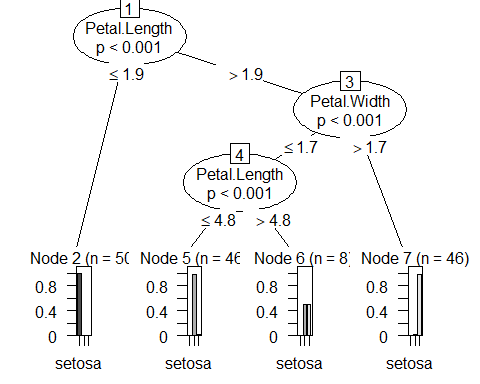
## Loading required package: sandwich



ct <- ctree(Species~., data=iris)  
ct

##   
## Conditional inference tree with 4 terminal nodes  
##   
## Response: Species   
## Inputs: Sepal.Length, Sepal.Width, Petal.Length, Petal.Width   
## Number of observations: 150   
##   
## 1) Petal.Length <= 1.9; criterion = 1, statistic = 140.264  
## 2)\* weights = 50   
## 1) Petal.Length > 1.9  
## 3) Petal.Width <= 1.7; criterion = 1, statistic = 67.894  
## 4) Petal.Length <= 4.8; criterion = 0.999, statistic = 13.865  
## 5)\* weights = 46   
## 4) Petal.Length > 4.8  
## 6)\* weights = 8   
## 3) Petal.Width > 1.7  
## 7)\* weights = 46

plot(ct)



predict(ct, newdata=iris)

## [1] setosa setosa setosa setosa setosa setosa   
## [7] setosa setosa setosa setosa setosa setosa   
## [13] setosa setosa setosa setosa setosa setosa   
## [19] setosa setosa setosa setosa setosa setosa   
## [25] setosa setosa setosa setosa setosa setosa   
## [31] setosa setosa setosa setosa setosa setosa   
## [37] setosa setosa setosa setosa setosa setosa   
## [43] setosa setosa setosa setosa setosa setosa   
## [49] setosa setosa versicolor versicolor versicolor versicolor  
## [55] versicolor versicolor versicolor versicolor versicolor versicolor  
## [61] versicolor versicolor versicolor versicolor versicolor versicolor  
## [67] versicolor versicolor versicolor versicolor virginica versicolor  
## [73] versicolor versicolor versicolor versicolor versicolor versicolor  
## [79] versicolor versicolor versicolor versicolor versicolor versicolor  
## [85] versicolor versicolor versicolor versicolor versicolor versicolor  
## [91] versicolor versicolor versicolor versicolor versicolor versicolor  
## [97] versicolor versicolor versicolor versicolor virginica virginica   
## [103] virginica virginica virginica virginica versicolor virginica   
## [109] virginica virginica virginica virginica virginica virginica   
## [115] virginica virginica virginica virginica virginica versicolor  
## [121] virginica virginica virginica virginica virginica virginica   
## [127] virginica virginica virginica versicolor virginica virginica   
## [133] virginica versicolor versicolor virginica virginica virginica   
## [139] virginica virginica virginica virginica virginica virginica   
## [145] virginica virginica virginica virginica virginica virginica   
## Levels: setosa versicolor virginica

# Randomforest는 앙상블(Ensemble) 학습을 사용한 모델  
# 앙상블 학습은 주어진 데이터로부터 여러개의 모델을 학습한 다음,   
# 예측 시 여러 모델의 예측 결과들을 종합해 정확도를 높이는 기법  
  
# 랜덤 포레스트는 두가지 방법을 사용해 다양한 의사결정 나무를 만듬  
# 첫번째, 데이터의 일부를 복원추출로 사용하여 의사결정나무 생성  
# 두번째 노드 나누는 기준에 사용되는 변수를 일부만 사용  
  
# 랜덤포레스트는 성능이 뛰어난 의사결정나무 하나로 모형을 예측하지 않고  
# 여러개를 사용하여 예측하므로 과적합므이 문제가 피해간다.  
  
if(!require(randomForest)) install.packages("randomForest"); library(randomForest)

## Loading required package: randomForest

## Warning: package 'randomForest' was built under R version 3.4.4

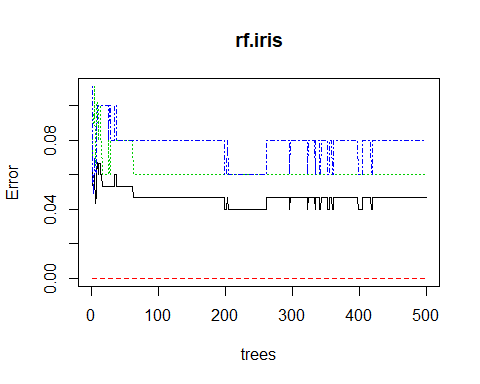
## randomForest 4.6-14

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

rf.iris <- randomForest(Species~., data=iris, importance=TRUE)  
  
rf.iris

##   
## Call:  
## randomForest(formula = Species ~ ., data = iris, importance = TRUE)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 2  
##   
## OOB estimate of error rate: 4.67%  
## Confusion matrix:  
## setosa versicolor virginica class.error  
## setosa 50 0 0 0.00  
## versicolor 0 47 3 0.06  
## virginica 0 4 46 0.08

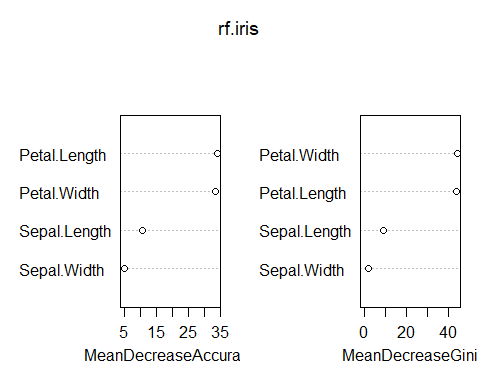
plot(rf.iris) #적절한 ntress를 판단할 수 있다.



importance(rf.iris) #변수별 중요도 확인

## setosa versicolor virginica MeanDecreaseAccuracy  
## Sepal.Length 6.316789 7.143148 8.090678 10.70338  
## Sepal.Width 4.622851 1.462479 4.415429 5.00773  
## Petal.Length 23.064071 32.965867 27.345715 33.95829  
## Petal.Width 22.173873 31.523845 29.390400 33.38757  
## MeanDecreaseGini  
## Sepal.Length 9.265806  
## Sepal.Width 2.141824  
## Petal.Length 43.651369  
## Petal.Width 44.169060

varImpPlot(rf.iris)



# 랜덤포레스트를 개선한 RRF (Regularized Random Forest)도 있음  
# package(RRF)  
  
# 랜덤포레스트 모형개선  
# randomforest()함수에서는 나무개수, 고려할 변수 mtry 등의 파라메터가 있다.   
# 기본적으로 자동자로 잘 부여되지만 모형 성능을 개선하고 싶다면 이 값을 조정할 수 있다.  
  
grid <- expand.grid(ntree=c(10,100,200), mtry=c(2,3,4))  
grid

## ntree mtry  
## 1 10 2  
## 2 100 2  
## 3 200 2  
## 4 10 3  
## 5 100 3  
## 6 200 3  
## 7 10 4  
## 8 100 4  
## 9 200 4

# 최적 조합을 찾기 위해 10개로 분할된 데이터를 이용해보자.  
  
if(!require(cvTools)) install.packages("cvTools"); library(cvTools)

## Loading required package: cvTools

## Warning: package 'cvTools' was built under R version 3.4.4

## Loading required package: lattice

## Loading required package: robustbase

## Warning: package 'robustbase' was built under R version 3.4.2

if(!require(foreach)) install.packages("foreach"); library(foreach)

## Loading required package: foreach

## Warning: package 'foreach' was built under R version 3.4.4

set.seed(3513)  
K=10  
R=3  
  
cv <- cvFolds(nrow(iris), K=K, R=R)  
  
grid <- expand.grid(ntree=c(10,100,200), mtry=c(3,4))  
  
result <- foreach(g=1:nrow(grid), .combine=rbind) %do% {  
 foreach(r=1:R, .combine = rbind) %do% {  
 foreach(k=1:K, .combine=rbind) %do% {  
   
 validation\_idx <- cv$subsets[which(cv$which == k), r]  
   
 train <- iris[-validation\_idx,]  
 validation <- iris[validation\_idx,]  
   
 m <- randomForest(Species~., data=train, ntree=grid[g, "ntree"],  
 mtry=grid[g,"mtry"])  
   
 predicted <- predict(m, newdata=validation)  
   
 precision <- sum(predicted == validation$Species) / length(predicted)  
 return(data.frame(g=g, precision=precision))  
 }  
 }  
}  
  
result

## g precision  
## 1 1 1.0000000  
## 2 1 1.0000000  
## 3 1 0.9333333  
## 4 1 1.0000000  
## 5 1 0.9333333  
## 6 1 0.9333333  
## 7 1 1.0000000  
## 8 1 0.8666667  
## 9 1 1.0000000  
## 10 1 0.9333333  
## 11 1 0.8000000  
## 12 1 0.9333333  
## 13 1 0.9333333  
## 14 1 0.8666667  
## 15 1 1.0000000  
## 16 1 1.0000000  
## 17 1 0.9333333  
## 18 1 1.0000000  
## 19 1 1.0000000  
## 20 1 0.9333333  
## 21 1 0.9333333  
## 22 1 0.8666667  
## 23 1 0.9333333  
## 24 1 1.0000000  
## 25 1 0.9333333  
## 26 1 0.9333333  
## 27 1 1.0000000  
## 28 1 0.9333333  
## 29 1 1.0000000  
## 30 1 0.9333333  
## 31 2 1.0000000  
## 32 2 0.9333333  
## 33 2 0.9333333  
## 34 2 1.0000000  
## 35 2 0.9333333  
## 36 2 0.9333333  
## 37 2 1.0000000  
## 38 2 0.8666667  
## 39 2 1.0000000  
## 40 2 0.9333333  
## 41 2 0.8000000  
## 42 2 0.9333333  
## 43 2 0.9333333  
## 44 2 0.9333333  
## 45 2 1.0000000  
## 46 2 1.0000000  
## 47 2 0.9333333  
## 48 2 1.0000000  
## 49 2 1.0000000  
## 50 2 0.9333333  
## 51 2 0.9333333  
## 52 2 0.8666667  
## 53 2 0.9333333  
## 54 2 1.0000000  
## 55 2 1.0000000  
## 56 2 0.9333333  
## 57 2 1.0000000  
## 58 2 0.9333333  
## 59 2 1.0000000  
## 60 2 0.9333333  
## 61 3 1.0000000  
## 62 3 0.9333333  
## 63 3 0.9333333  
## 64 3 1.0000000  
## 65 3 0.9333333  
## 66 3 0.9333333  
## 67 3 1.0000000  
## 68 3 0.8666667  
## 69 3 1.0000000  
## 70 3 0.9333333  
## 71 3 0.8000000  
## 72 3 0.9333333  
## 73 3 0.9333333  
## 74 3 0.8666667  
## 75 3 1.0000000  
## 76 3 1.0000000  
## 77 3 0.9333333  
## 78 3 1.0000000  
## 79 3 1.0000000  
## 80 3 0.9333333  
## 81 3 0.9333333  
## 82 3 0.8666667  
## 83 3 0.9333333  
## 84 3 1.0000000  
## 85 3 1.0000000  
## 86 3 0.9333333  
## 87 3 1.0000000  
## 88 3 0.9333333  
## 89 3 1.0000000  
## 90 3 0.9333333  
## 91 4 1.0000000  
## 92 4 1.0000000  
## 93 4 0.9333333  
## 94 4 1.0000000  
## 95 4 0.9333333  
## 96 4 0.9333333  
## 97 4 1.0000000  
## 98 4 0.8666667  
## 99 4 1.0000000  
## 100 4 0.8666667  
## 101 4 0.7333333  
## 102 4 0.9333333  
## 103 4 0.9333333  
## 104 4 1.0000000  
## 105 4 1.0000000  
## 106 4 1.0000000  
## 107 4 0.8666667  
## 108 4 1.0000000  
## 109 4 1.0000000  
## 110 4 0.9333333  
## 111 4 0.9333333  
## 112 4 0.9333333  
## 113 4 0.9333333  
## 114 4 1.0000000  
## 115 4 1.0000000  
## 116 4 0.9333333  
## 117 4 1.0000000  
## 118 4 0.9333333  
## 119 4 1.0000000  
## 120 4 0.9333333  
## 121 5 1.0000000  
## 122 5 0.9333333  
## 123 5 0.9333333  
## 124 5 1.0000000  
## 125 5 0.9333333  
## 126 5 0.9333333  
## 127 5 1.0000000  
## 128 5 0.8666667  
## 129 5 1.0000000  
## 130 5 0.9333333  
## 131 5 0.8000000  
## 132 5 0.9333333  
## 133 5 0.9333333  
## 134 5 0.8666667  
## 135 5 1.0000000  
## 136 5 1.0000000  
## 137 5 0.9333333  
## 138 5 1.0000000  
## 139 5 1.0000000  
## 140 5 0.9333333  
## 141 5 0.9333333  
## 142 5 0.8666667  
## 143 5 0.9333333  
## 144 5 1.0000000  
## 145 5 1.0000000  
## 146 5 0.9333333  
## 147 5 1.0000000  
## 148 5 0.9333333  
## 149 5 1.0000000  
## 150 5 0.9333333  
## 151 6 1.0000000  
## 152 6 0.9333333  
## 153 6 0.9333333  
## 154 6 1.0000000  
## 155 6 0.9333333  
## 156 6 0.9333333  
## 157 6 1.0000000  
## 158 6 0.8666667  
## 159 6 1.0000000  
## 160 6 0.9333333  
## 161 6 0.8000000  
## 162 6 0.9333333  
## 163 6 0.9333333  
## 164 6 0.8666667  
## 165 6 1.0000000  
## 166 6 1.0000000  
## 167 6 0.9333333  
## 168 6 1.0000000  
## 169 6 1.0000000  
## 170 6 0.9333333  
## 171 6 0.9333333  
## 172 6 0.9333333  
## 173 6 0.9333333  
## 174 6 1.0000000  
## 175 6 1.0000000  
## 176 6 0.9333333  
## 177 6 1.0000000  
## 178 6 0.9333333  
## 179 6 1.0000000  
## 180 6 0.9333333

if(!require(plyr)) install.packages("plyr"); library(plyr)

## Loading required package: plyr

##   
## Attaching package: 'plyr'

## The following object is masked from 'package:modeltools':  
##   
## empty

#grid에 따른 결과를 살펴보자.  
ddply(result, ~g, summarize, mean\_precision=mean(precision))

## g mean\_precision  
## 1 1 0.9488889  
## 2 2 0.9511111  
## 3 3 0.9488889  
## 4 4 0.9511111  
## 5 5 0.9488889  
## 6 6 0.9511111