K-NN and Decision Tree on IRIS dataset

Abhay Padda

3 March 2018

## k-NN for IRIS dataset. The file content is included at the bottom of this markdown  
source('knn\_functions.R')  
  
str(iris)

## 'data.frame': 150 obs. of 5 variables:  
## $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...  
## $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...  
## $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...  
## $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...  
## $ Species : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...

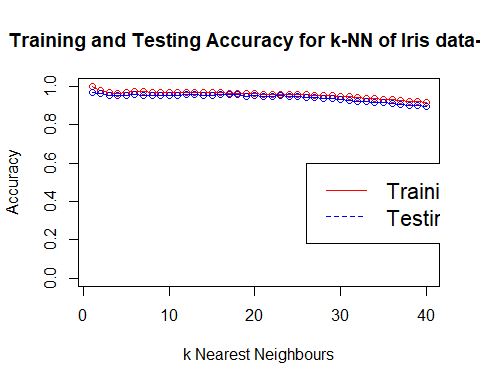
summary(iris)

## Sepal.Length Sepal.Width Petal.Length Petal.Width   
## Min. :4.300 Min. :2.000 Min. :1.000 Min. :0.100   
## 1st Qu.:5.100 1st Qu.:2.800 1st Qu.:1.600 1st Qu.:0.300   
## Median :5.800 Median :3.000 Median :4.350 Median :1.300   
## Mean :5.843 Mean :3.057 Mean :3.758 Mean :1.199   
## 3rd Qu.:6.400 3rd Qu.:3.300 3rd Qu.:5.100 3rd Qu.:1.800   
## Max. :7.900 Max. :4.400 Max. :6.900 Max. :2.500   
## Species   
## setosa :50   
## versicolor:50   
## virginica :50   
##   
##   
##

set.seed(100)  
  
m <- avgTrnTst(iris, 0.7, 5)  
dim(m)

## [1] 40 3

plotFn(m, 'Training and Testing Accuracy for k-NN of Iris data-set')



## Decision Tree for Iris dataset  
library(rpart)  
library(rpart.plot)  
  
v <- iris$Species  
  
table(v)

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

set.seed(522)  
  
# runif function returns a uniform distribution which can be further conditionally split into 75-25 ratio  
iris[, 'train'] <- ifelse(runif(nrow(iris)) < 0.75, 1, 0)  
  
trainSet <- iris[iris$train == 1,]  
testSet <- iris[iris$train == 0, ]  
  
trainColNum <- grep('train', names(trainSet))  
  
trainSet <- trainSet[, -trainColNum]  
testSet <- testSet[, -trainColNum]  
  
treeFit <- rpart(Species~.,data=trainSet,method = 'class')  
print(treeFit)

## n= 111   
##   
## node), split, n, loss, yval, (yprob)  
## \* denotes terminal node  
##   
## 1) root 111 74 setosa (0.33333333 0.33333333 0.33333333)   
## 2) Petal.Length< 2.45 37 0 setosa (1.00000000 0.00000000 0.00000000) \*  
## 3) Petal.Length>=2.45 74 37 versicolor (0.00000000 0.50000000 0.50000000)   
## 6) Petal.Width< 1.75 39 2 versicolor (0.00000000 0.94871795 0.05128205) \*  
## 7) Petal.Width>=1.75 35 0 virginica (0.00000000 0.00000000 1.00000000) \*

rpart.plot(treeFit, box.col=c("red", "green"))  
  
Prediction1 <- predict(treeFit,newdata=testSet[-5],type = 'class')  
  
  
## Print the confusion matrix to check the accuracy and other statistics  
library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

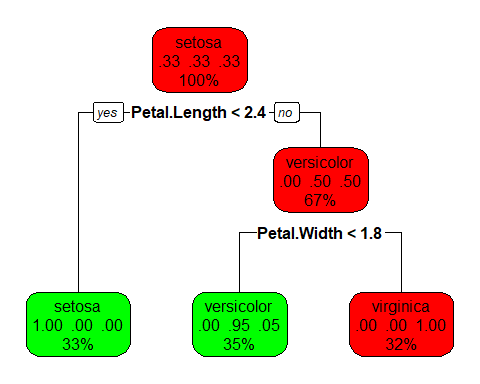
confusionMatrix(Prediction1,testSet$Species)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction setosa versicolor virginica  
## setosa 13 0 0  
## versicolor 0 12 3  
## virginica 0 1 10  
##   
## Overall Statistics  
##   
## Accuracy : 0.8974   
## 95% CI : (0.7578, 0.9713)  
## No Information Rate : 0.3333   
## P-Value [Acc > NIR] : 3.435e-13   
##   
## Kappa : 0.8462   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: setosa Class: versicolor Class: virginica  
## Sensitivity 1.0000 0.9231 0.7692  
## Specificity 1.0000 0.8846 0.9615  
## Pos Pred Value 1.0000 0.8000 0.9091  
## Neg Pred Value 1.0000 0.9583 0.8929  
## Prevalence 0.3333 0.3333 0.3333  
## Detection Rate 0.3333 0.3077 0.2564  
## Detection Prevalence 0.3333 0.3846 0.2821  
## Balanced Accuracy 1.0000 0.9038 0.8654

## Pruning the decision tree  
printcp(treeFit)

##   
## Classification tree:  
## rpart(formula = Species ~ ., data = trainSet, method = "class")  
##   
## Variables actually used in tree construction:  
## [1] Petal.Length Petal.Width   
##   
## Root node error: 74/111 = 0.66667  
##   
## n= 111   
##   
## CP nsplit rel error xerror xstd  
## 1 0.50000 0 1.000000 1.148649 0.060298  
## 2 0.47297 1 0.500000 0.783784 0.071115  
## 3 0.01000 2 0.027027 0.027027 0.018938

opt <- which.min(treeFit$cptable[,'xerror'])  
  
cp <- treeFit$cptable[opt, 'CP']  
pruned\_model <- prune(treeFit,cp)  
rpart.plot(pruned\_model, box.col=c("red", "green"))



rpart\_pruned\_predict <- predict(pruned\_model, newdata=testSet[-5],type = 'class')  
mn2 <- mean(rpart\_pruned\_predict==testSet$Species)  
mn2

## [1] 0.8974359

confusionMatrix(rpart\_pruned\_predict,testSet$Species)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction setosa versicolor virginica  
## setosa 13 0 0  
## versicolor 0 12 3  
## virginica 0 1 10  
##   
## Overall Statistics  
##   
## Accuracy : 0.8974   
## 95% CI : (0.7578, 0.9713)  
## No Information Rate : 0.3333   
## P-Value [Acc > NIR] : 3.435e-13   
##   
## Kappa : 0.8462   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: setosa Class: versicolor Class: virginica  
## Sensitivity 1.0000 0.9231 0.7692  
## Specificity 1.0000 0.8846 0.9615  
## Pos Pred Value 1.0000 0.8000 0.9091  
## Neg Pred Value 1.0000 0.9583 0.8929  
## Prevalence 0.3333 0.3333 0.3333  
## Detection Rate 0.3333 0.3077 0.2564  
## Detection Prevalence 0.3333 0.3846 0.2821  
## Balanced Accuracy 1.0000 0.9038 0.8654

#### knn\_functions.R file content

# 'caTools' package provides us with functions to split dataset uniformly to test and training  
library(caTools)  
  
# Load library 'class' that has the knn() function  
library(class)  
  
# Function to split the dataset randomly  
splitFile <- function(dataset, trProp, classColPos) {  
 # split the dataset  
 sample = sample.split(iris[, classColPos], SplitRatio = trProp)  
   
 # create training and testing dataset  
 train = subset(iris, sample == TRUE)  
 test = subset(iris, sample == FALSE)  
   
 # save the target labels and remove from the train and test dataset  
 trainLabels <- train[, classColPos]  
 testLabels <- test[, classColPos]  
 train <- train[, -classColPos]  
 test <- test[, -classColPos]  
   
 # Nomalize function  
 normalize <- function(x) {  
 return( (x-min(x))/(max(x)-min(x)))  
 }  
 train  
 test  
 # Normalize test and training dataset  
 gtrn <- as.data.frame(lapply(train, normalize))  
 gtsn <- as.data.frame(lapply(test, normalize))  
   
 return(list(trn=gtrn, trL=trainLabels, val=gtsn, tsL=testLabels))  
}  
  
# Function to plot graph  
plotFn <- function(dataSet, graphTitle = '', ylimLo=0) {  
 plot(dataSet[, 1], dataSet[, 2], main = graphTitle, xlab = 'k Nearest Neighbours',  
 ylab = 'Accuracy', ylim = c(ylimLo, 1), type = 'o', col = 'red')  
 lines(dataSet[, 1], dataSet[, 3], type = 'o', col = 'blue')  
 legend(26, 0.6, legend=c("Training Accuracy", "Testing Accuracy"),  
 col=c("red", "blue"), lty=1:2, cex=1.4)  
}  
  
  
# Function to use k-NN and return training and testing results  
train\_test <- function(trainData,trainLabels,testData,testLabels) {  
 train <- c()  
 test <- c()  
 for (k in 1:40) {  
 knntr <- knn(trainData, trainData, trainLabels, k=k)  
 knnts <- knn(trainData, testData, trainLabels, k=k)  
 trTable <- table(knntr, trainLabels)  
 tsTable <- table(knnts, testLabels)  
 trTable <- prop.table(trTable)  
 tsTable <- prop.table(tsTable)  
 trainAccuracy <- sum(trTable[1,1], trTable[2,2], trTable[3,3])/sum(trTable)  
 testAccuracy <- sum(tsTable[1,1], tsTable[2,2], trTable[3,3])/sum(tsTable)  
 train <- c(train, trainAccuracy)  
 test <- c(test, testAccuracy)  
 }  
 acc <- data.frame('k' = 1:40, 'trAc' = train, 'tsAc' = test)  
 return(acc = acc)  
}  
  
# Single function to split data and then call train\_test function  
avgTrnTst <- function(dataset, trProp, classColPos) {  
 for (i in 1:30) {  
 a <- splitFile(dataset, trProp, classColPos)  
 b <- train\_test(a$trn, a$trL, a$val, a$tsL)  
 if (i==1) acd <- b  
 else acd <- rbind(acd, b)  
 }  
 library(plyr)  
   
 a1 <- ddply(acd,.(k), summarize, meanV = mean(trAc))  
 a2 <- ddply(acd,.(k), summarize, meanV = mean(tsAc))  
 m <- merge(a1,a2,by='k')  
   
 return(m)  
}