wineTestAgain3.r

## File: WineTestAgain3.r  
## Author: A. Breitzman  
## Date: 2/25/2017 modified 5/23/2018  
## Description: k-nearest neighbors; bagging on the wine set

library(e1071)  
library("neuralnet")  
library("plyr")  
setwd("C:/Users/Tony/Dropbox/Rowan/DM2/Lecture4")  
whiteWineData <-read.csv("winequality-white.csv",header = TRUE, sep = ";", stringsAsFactors = TRUE)  
redWineData <-read.csv("winequality-red.csv",header = TRUE, sep = ";", stringsAsFactors = TRUE)  
  
maxs <- apply(whiteWineData, 2, max)  
mins <- apply(whiteWineData, 2, min)  
scaledWhite <- as.data.frame(scale(whiteWineData, center = mins, scale = maxs - mins))  
scaledRed <- as.data.frame(scale(redWineData, center=mins, scale=maxs-mins))  
names(scaledWhite)

## [1] "fixed.acidity" "volatile.acidity" "citric.acid"   
## [4] "residual.sugar" "chlorides" "free.sulfur.dioxide"   
## [7] "total.sulfur.dioxide" "density" "pH"   
## [10] "sulphates" "alcohol" "quality"

## Unscale quality first  
scaledWhite$quality <- 3+6\*scaledWhite$quality  
count(scaledWhite$quality)

## x freq  
## 1 3 20  
## 2 4 163  
## 3 5 1457  
## 4 6 2198  
## 5 7 880  
## 6 8 175  
## 7 9 5

set.seed(2)  
train <- sample(1:nrow(scaledWhite),nrow(scaledWhite)\*(8/10))  
test<- -train  
  
trainingData<-scaledWhite[train,]  
testingData<-scaledWhite[test,]  
  
  
## We'll build the formula the right way instead of the lazy  
## way I've been doing it by renaming all the names to ai  
n<-names(scaledWhite)  
formula <- as.formula(paste("as.factor(quality) ~", paste(n[!n %in% "quality"], collapse = " + ")))  
formula

## as.factor(quality) ~ fixed.acidity + volatile.acidity + citric.acid +   
## residual.sugar + chlorides + free.sulfur.dioxide + total.sulfur.dioxide +   
## density + pH + sulphates + alcohol

## in some libraries the periods wil give us trouble. We'll see if they do and  
## change later if we have to.  
  
trainingData<-scaledWhite[train,]  
testingData<-scaledWhite[test,]  
  
  
svm\_model <- svm(formula, data=trainingData)  
summary(svm\_model)

##   
## Call:  
## svm(formula = formula, data = trainingData)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 1   
## gamma: 0.09090909   
##   
## Number of Support Vectors: 3521  
##   
## ( 127 1021 1525 140 691 14 3 )  
##   
##   
## Number of Classes: 7   
##   
## Levels:   
## 3 4 5 6 7 8 9

testingData$result <- predict(svm\_model,testingData)  
trainingData$result <- predict(svm\_model,trainingData)

## Create multiple summaries  
errorSummary<- function(testingData){  
 temp <- count(testingData,c("quality","result"))  
 correct<-0  
 wrong<-0  
 correct3 <- 0  
 wrong3 <- 0  
 for (i in 1:dim(temp)[1]){  
 t <- temp[i,]  
 ## Count 7 class correctness  
 if (t$quality==t$result)  
 correct <- correct + t$freq  
 else  
 wrong <- wrong + t$freq  
 ##Count 3 class correctness  
  
 if (t$quality == 3 || t$quality == 4){  
 if (t$result == 3 || t$result == 4)  
 correct3 <- correct3 + t$freq  
 else  
 wrong3 <- wrong3 + t$freq  
 }else{  
 if (t$quality ==5 || t$quality ==6){  
 if (t$result == 5 || t$result == 6)  
 correct3 <- correct3 + t$freq  
 else  
 wrong3 <- wrong3 + t$freq  
 }else{  
 if (t$quality == 7 || t$quality == 8 || t$quality == 9 ){  
 if (t$result == 7 || t$result == 8 || t$result ==9)  
 correct3 <- correct3 + t$freq  
 else  
 wrong3 <- wrong3 + t$freq  
 }  
 }  
 }}  
 print(noquote("7 class summary: 3,4,5,6,7,8,9"))  
 print(noquote(paste("Number Correct:",correct)))  
 print(noquote(paste("Number Wrong:",wrong)))  
 print(noquote(paste("Percent Error:",(100\*wrong)/(correct+wrong))))  
 print(noquote("3 class summary: (bad=3,4, OK=5,6, Good=7,8,9)"))  
 print(noquote(paste("Number Correct:",correct3)))  
 print(noquote(paste("Number Wrong:",wrong3)))  
 print(noquote(paste("Percent Error:",(100\*wrong3)/(correct3+wrong3))))   
}

errorSummary(testingData)

## [1] 7 class summary: 3,4,5,6,7,8,9  
## [1] Number Correct: 563  
## [1] Number Wrong: 417  
## [1] Percent Error: 42.5510204081633  
## [1] 3 class summary: (bad=3,4, OK=5,6, Good=7,8,9)  
## [1] Number Correct: 773  
## [1] Number Wrong: 207  
## [1] Percent Error: 21.1224489795918

## 42.5% Error when trying to pick exact class  
## 21% error when trying to pick bad, ok, good  
  
  
## Let's see if we get a better error rate if we just try directly  
## to compute a 3 class SVM rather than build it like above  
scaledWhite$Q2 <- apply(scaledWhite,MARGIN=1,function(x){  
 if (x["quality"] <= 4) return("bad")  
 if (x["quality"] <= 6) return("ok")  
 return("good")  
})   
count(scaledWhite,c("quality","Q2"))

## quality Q2 freq  
## 1 3 bad 20  
## 2 4 bad 163  
## 3 5 ok 1457  
## 4 6 ok 2198  
## 5 7 good 880  
## 6 8 good 175  
## 7 9 good 5

##Let's Try again  
formula <- as.formula(paste("as.factor(Q2) ~", paste(n[!n %in% "quality"], collapse = " + ")))  
formula

## as.factor(Q2) ~ fixed.acidity + volatile.acidity + citric.acid +   
## residual.sugar + chlorides + free.sulfur.dioxide + total.sulfur.dioxide +   
## density + pH + sulphates + alcohol

trainingData<-scaledWhite[train,]  
testingData<-scaledWhite[test,]  
  
svm\_model <- svm(formula, data=trainingData)

summary(svm\_model)

##   
## Call:  
## svm(formula = formula, data = trainingData)  
##   
##   
## Parameters:  
## SVM-Type: C-classification   
## SVM-Kernel: radial   
## cost: 1   
## gamma: 0.09090909   
##   
## Number of Support Vectors: 2135  
##   
## ( 141 1211 783 )  
##   
##   
## Number of Classes: 3   
##   
## Levels:   
## bad good ok

testingData$result <- predict(svm\_model,testingData)  
trainingData$result <- predict(svm\_model,trainingData)  
  
count(testingData,c("Q2","result"))

## Q2 result freq  
## 1 bad bad 1  
## 2 bad good 1  
## 3 bad ok 40  
## 4 good good 65  
## 5 good ok 133  
## 6 ok good 39  
## 7 ok ok 701

correct = 1 + 65 + 701  
wrong = 1 + 40 + 133 + 39  
wrong/(wrong + correct)

## [1] 0.2173469

## Same error rate that we got above  
  
  
## Now let's do a Neural Net  
  
scaledWhite$Q4 <- (scaledWhite$quality-3)/6  
trainingData<-scaledWhite[train,]  
testingData<-scaledWhite[test,]  
formula <- as.formula(paste("Q4 ~", paste(n[!n %in% "quality"], collapse = " + ")))  
formula

## Q4 ~ fixed.acidity + volatile.acidity + citric.acid + residual.sugar +   
## chlorides + free.sulfur.dioxide + total.sulfur.dioxide +   
## density + pH + sulphates + alcohol

nnet<- neuralnet(formula, trainingData, hidden = c(12,6), threshold = 0.1)  
result <- compute(nnet, testingData[,1:11])  
  
testingData$result <- sapply(result$net.result, function(b) {  
 return(floor(3.5+(b\*6)))  
})

errorSummary(testingData)

## [1] 7 class summary: 3,4,5,6,7,8,9  
## [1] Number Correct: 512  
## [1] Number Wrong: 468  
## [1] Percent Error: 47.7551020408163  
## [1] 3 class summary: (bad=3,4, OK=5,6, Good=7,8,9)  
## [1] Number Correct: 746  
## [1] Number Wrong: 234  
## [1] Percent Error: 23.8775510204082

## Worse then the SVM  
  
## Let's try a random forest  
library(randomForest)

## randomForest 4.6-12

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

formula <- as.formula(paste("as.factor(quality) ~", paste(n[!n %in% "quality"], collapse = " + ")))  
formula

## as.factor(quality) ~ fixed.acidity + volatile.acidity + citric.acid +   
## residual.sugar + chlorides + free.sulfur.dioxide + total.sulfur.dioxide +   
## density + pH + sulphates + alcohol

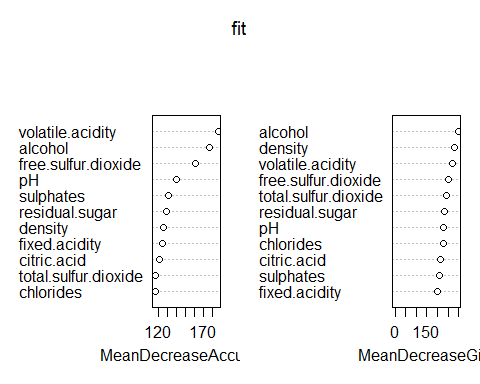
fit <- randomForest(formula,  
 data=trainingData,   
 importance=TRUE,   
 ntree=2000)  
  
  
  
varImpPlot(fit)  
  
##install.packages("caret")  
library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

##   
## Attaching package: 'ggplot2'

## The following object is masked from 'package:randomForest':  
##   
## margin



testingData$result <- predict(fit, testingData)  
errorSummary(testingData)

## [1] 7 class summary: 3,4,5,6,7,8,9  
## [1] Number Correct: 677  
## [1] Number Wrong: 303  
## [1] Percent Error: 30.9183673469388  
## [1] 3 class summary: (bad=3,4, OK=5,6, Good=7,8,9)  
## [1] Number Correct: 837  
## [1] Number Wrong: 143  
## [1] Percent Error: 14.5918367346939

## Much much better  
  
## Let's see how it does predicting red wine quality  
scaledRed$quality <- 3+(6\*scaledRed$quality)  
  
scaledRed$result <- predict(fit, scaledRed)  
errorSummary(scaledRed)

## [1] 7 class summary: 3,4,5,6,7,8,9  
## [1] Number Correct: 712  
## [1] Number Wrong: 887  
## [1] Percent Error: 55.4721701063164  
## [1] 3 class summary: (bad=3,4, OK=5,6, Good=7,8,9)  
## [1] Number Correct: 1059  
## [1] Number Wrong: 540  
## [1] Percent Error: 33.7711069418386

## Awful  
  
## Let's try a single tree  
library(rpart)  
treeFit <- rpart(formula,  
 data=trainingData,method="class")  
  
testingData$result <- predict(treeFit, testingData,type="class")  
errorSummary(testingData)

## [1] 7 class summary: 3,4,5,6,7,8,9  
## [1] Number Correct: 528  
## [1] Number Wrong: 452  
## [1] Percent Error: 46.1224489795918  
## [1] 3 class summary: (bad=3,4, OK=5,6, Good=7,8,9)  
## [1] Number Correct: 753  
## [1] Number Wrong: 227  
## [1] Percent Error: 23.1632653061224

## Worse than all methods. This suggests the bagging in the Random forest  
## makes a big difference  
  
## Let's try K-Nearest Neighbors  
## install.packages("FNN")  
library(FNN)  
testingData$result <- knn(trainingData[,1:11],testingData[,1:11],trainingData[,12],k=1)  
  
errorSummary(testingData)

## [1] 7 class summary: 3,4,5,6,7,8,9  
## [1] Number Correct: 631  
## [1] Number Wrong: 349  
## [1] Percent Error: 35.6122448979592  
## [1] 3 class summary: (bad=3,4, OK=5,6, Good=7,8,9)  
## [1] Number Correct: 796  
## [1] Number Wrong: 184  
## [1] Percent Error: 18.7755102040816  
count(testingData,c("quality","result"))

## quality result freq  
## 1 3 5 3  
## 2 3 6 3  
## 3 4 4 16  
## 4 4 5 11  
## 5 4 6 5  
## 6 4 7 3  
## 7 4 8 1  
## 8 5 3 1  
## 9 5 4 4  
## 10 5 5 199  
## 11 5 6 78  
## 12 5 7 9  
## 13 5 8 2  
## 14 6 3 1  
## 15 6 4 7  
## 16 6 5 69  
## 17 6 6 308  
## 18 6 7 55  
## 19 6 8 7  
## 20 7 4 1  
## 21 7 5 11  
## 22 7 6 45  
## 23 7 7 97  
## 24 7 8 7  
## 25 8 5 1  
## 26 8 6 13  
## 27 8 7 10  
## 28 8 8 11  
## 29 9 6 1  
## 30 9 7 1

## Pretty good. Let's see how it works with k=3  
testingData$result <- knn(trainingData[,1:11],testingData[,1:11],trainingData[,12],k=3)  
errorSummary(testingData)

## [1] 7 class summary: 3,4,5,6,7,8,9  
## [1] Number Correct: 543  
## [1] Number Wrong: 437  
## [1] Percent Error: 44.5918367346939  
## [1] 3 class summary: (bad=3,4, OK=5,6, Good=7,8,9)  
## [1] Number Correct: 752  
## [1] Number Wrong: 228  
## [1] Percent Error: 23.265306122449

## Not as good. Which makes sense because there is a small number of 3's and 9's  
count(testingData,c("quality","result"))

## quality result freq  
## 1 3 5 3  
## 2 3 6 3  
## 3 4 4 10  
## 4 4 5 13  
## 5 4 6 10  
## 6 4 7 2  
## 7 4 8 1  
## 8 5 3 1  
## 9 5 4 10  
## 10 5 5 173  
## 11 5 6 100  
## 12 5 7 9  
## 13 6 3 1  
## 14 6 4 11  
## 15 6 5 91  
## 16 6 6 278  
## 17 6 7 61  
## 18 6 8 5  
## 19 7 4 1  
## 20 7 5 19  
## 21 7 6 62  
## 22 7 7 74  
## 23 7 8 5  
## 24 8 4 2  
## 25 8 6 13  
## 26 8 7 12  
## 27 8 8 8  
## 28 9 6 1  
## 29 9 7 1

## Let's try k=5  
testingData$result <- knn(trainingData[,1:11],testingData[,1:11],trainingData[,12],k=3)  
errorSummary(testingData)

## [1] 7 class summary: 3,4,5,6,7,8,9  
## [1] Number Correct: 543  
## [1] Number Wrong: 437  
## [1] Percent Error: 44.5918367346939  
## [1] 3 class summary: (bad=3,4, OK=5,6, Good=7,8,9)  
## [1] Number Correct: 752  
## [1] Number Wrong: 228  
## [1] Percent Error: 23.265306122449

count(testingData,c("quality","result"))

## quality result freq  
## 1 3 5 3  
## 2 3 6 3  
## 3 4 4 10  
## 4 4 5 13  
## 5 4 6 10  
## 6 4 7 2  
## 7 4 8 1  
## 8 5 3 1  
## 9 5 4 10  
## 10 5 5 173  
## 11 5 6 100  
## 12 5 7 9  
## 13 6 3 1  
## 14 6 4 11  
## 15 6 5 91  
## 16 6 6 278  
## 17 6 7 61  
## 18 6 8 5  
## 19 7 4 1  
## 20 7 5 19  
## 21 7 6 62  
## 22 7 7 74  
## 23 7 8 5  
## 24 8 4 2  
## 25 8 6 13  
## 26 8 7 12  
## 27 8 8 8  
## 28 9 6 1  
## 29 9 7 1

## Going in the wrong direction

## Let's try the ownn function which implements bagging  
temp <- ownn(trainingData[,1:11], testingData[,1:11],trainingData[,12],k=1)  
## default knn  
testingData$result<-temp$knnpred  
errorSummary(testingData)

## [1] 7 class summary: 3,4,5,6,7,8,9  
## [1] Number Correct: 631  
## [1] Number Wrong: 349  
## [1] Percent Error: 35.6122448979592  
## [1] 3 class summary: (bad=3,4, OK=5,6, Good=7,8,9)  
## [1] Number Correct: 796  
## [1] Number Wrong: 184  
## [1] Percent Error: 18.7755102040816

## bagging knn  
testingData$result<-temp$bnnpred  
errorSummary(testingData)

## [1] 7 class summary: 3,4,5,6,7,8,9  
## [1] Number Correct: 486  
## [1] Number Wrong: 494  
## [1] Percent Error: 50.4081632653061  
## [1] 3 class summary: (bad=3,4, OK=5,6, Good=7,8,9)  
## [1] Number Correct: 704  
## [1] Number Wrong: 276  
## [1] Percent Error: 28.1632653061224

## optimal weighted knn  
testingData$result<-temp$ownnpred  
errorSummary(testingData)

## [1] 7 class summary: 3,4,5,6,7,8,9  
## [1] Number Correct: 631  
## [1] Number Wrong: 349  
## [1] Percent Error: 35.6122448979592  
## [1] 3 class summary: (bad=3,4, OK=5,6, Good=7,8,9)  
## [1] Number Correct: 796  
## [1] Number Wrong: 184  
## [1] Percent Error: 18.7755102040816

## Let's try with k = 3  
temp <- ownn(trainingData[,1:11], testingData[,1:11],trainingData[,12],k=3)  
## default knn  
testingData$result<-temp$knnpred  
errorSummary(testingData)

## [1] 7 class summary: 3,4,5,6,7,8,9  
## [1] Number Correct: 543  
## [1] Number Wrong: 437  
## [1] Percent Error: 44.5918367346939  
## [1] 3 class summary: (bad=3,4, OK=5,6, Good=7,8,9)  
## [1] Number Correct: 752  
## [1] Number Wrong: 228  
## [1] Percent Error: 23.265306122449

## bagging knn  
testingData$result<-temp$bnnpred  
errorSummary(testingData)

## [1] 7 class summary: 3,4,5,6,7,8,9  
## [1] Number Correct: 631  
## [1] Number Wrong: 349  
## [1] Percent Error: 35.6122448979592  
## [1] 3 class summary: (bad=3,4, OK=5,6, Good=7,8,9)  
## [1] Number Correct: 796  
## [1] Number Wrong: 184  
## [1] Percent Error: 18.7755102040816

## optimal weighted knn  
testingData$result<-temp$ownnpred  
errorSummary(testingData)

## [1] 7 class summary: 3,4,5,6,7,8,9  
## [1] Number Correct: 619  
## [1] Number Wrong: 361  
## [1] Percent Error: 36.8367346938776  
## [1] 3 class summary: (bad=3,4, OK=5,6, Good=7,8,9)  
## [1] Number Correct: 796  
## [1] Number Wrong: 184  
## [1] Percent Error: 18.7755102040816

## Let's try with k = 5  
temp <- ownn(trainingData[,1:11], testingData[,1:11],trainingData[,12],k=5)  
## default knn  
testingData$result<-temp$knnpred  
errorSummary(testingData)

## [1] 7 class summary: 3,4,5,6,7,8,9  
## [1] Number Correct: 548  
## [1] Number Wrong: 432  
## [1] Percent Error: 44.0816326530612  
## [1] 3 class summary: (bad=3,4, OK=5,6, Good=7,8,9)  
## [1] Number Correct: 768  
## [1] Number Wrong: 212  
## [1] Percent Error: 21.6326530612245

## bagging knn  
testingData$result<-temp$bnnpred  
errorSummary(testingData)

## [1] 7 class summary: 3,4,5,6,7,8,9  
## [1] Number Correct: 613  
## [1] Number Wrong: 367  
## [1] Percent Error: 37.4489795918367  
## [1] 3 class summary: (bad=3,4, OK=5,6, Good=7,8,9)  
## [1] Number Correct: 790  
## [1] Number Wrong: 190  
## [1] Percent Error: 19.3877551020408

## optimal weighted knn  
testingData$result<-temp$ownnpred  
errorSummary(testingData)

## [1] 7 class summary: 3,4,5,6,7,8,9  
## [1] Number Correct: 599  
## [1] Number Wrong: 381  
## [1] Percent Error: 38.8775510204082  
## [1] 3 class summary: (bad=3,4, OK=5,6, Good=7,8,9)  
## [1] Number Correct: 785  
## [1] Number Wrong: 195  
## [1] Percent Error: 19.8979591836735

## Let's try with k = 7  
temp <- ownn(trainingData[,1:11], testingData[,1:11],trainingData[,12],k=7)  
## default knn  
testingData$result<-temp$knnpred  
errorSummary(testingData)

## [1] 7 class summary: 3,4,5,6,7,8,9  
## [1] Number Correct: 553  
## [1] Number Wrong: 427  
## [1] Percent Error: 43.5714285714286  
## [1] 3 class summary: (bad=3,4, OK=5,6, Good=7,8,9)  
## [1] Number Correct: 769  
## [1] Number Wrong: 211  
## [1] Percent Error: 21.530612244898

## bagging knn  
testingData$result<-temp$bnnpred  
errorSummary(testingData)

## [1] 7 class summary: 3,4,5,6,7,8,9  
## [1] Number Correct: 594  
## [1] Number Wrong: 386  
## [1] Percent Error: 39.3877551020408  
## [1] 3 class summary: (bad=3,4, OK=5,6, Good=7,8,9)  
## [1] Number Correct: 781  
## [1] Number Wrong: 199  
## [1] Percent Error: 20.3061224489796

## optimal weighted knn  
testingData$result<-temp$ownnpred  
errorSummary(testingData)

## [1] 7 class summary: 3,4,5,6,7,8,9  
## [1] Number Correct: 598  
## [1] Number Wrong: 382  
## [1] Percent Error: 38.9795918367347  
## [1] 3 class summary: (bad=3,4, OK=5,6, Good=7,8,9)  
## [1] Number Correct: 786  
## [1] Number Wrong: 194  
## [1] Percent Error: 19.7959183673469

## So k=3 is the best. Let's go back to the original question of how does   
## the white wine model do with red wine.  
  
temp <- ownn(trainingData[,1:11], scaledRed[,1:11],trainingData[,12],k=3)  
## default knn  
scaledRed$result<-temp$knnpred

errorSummary(scaledRed)

## [1] 7 class summary: 3,4,5,6,7,8,9  
## [1] Number Correct: 477  
## [1] Number Wrong: 1122  
## [1] Percent Error: 70.1688555347092  
## [1] 3 class summary: (bad=3,4, OK=5,6, Good=7,8,9)  
## [1] Number Correct: 761  
## [1] Number Wrong: 838  
## [1] Percent Error: 52.4077548467792

## bagging knn  
scaledRed$result<-temp$bnnpred  
errorSummary(scaledRed)

## [1] 7 class summary: 3,4,5,6,7,8,9  
## [1] Number Correct: 487  
## [1] Number Wrong: 1112  
## [1] Percent Error: 69.5434646654159  
## [1] 3 class summary: (bad=3,4, OK=5,6, Good=7,8,9)  
## [1] Number Correct: 850  
## [1] Number Wrong: 749  
## [1] Percent Error: 46.8417761100688

## optimal weighted knn  
scaledRed$result<-temp$ownnpred  
errorSummary(scaledRed)

## [1] 7 class summary: 3,4,5,6,7,8,9  
## [1] Number Correct: 504  
## [1] Number Wrong: 1095  
## [1] Percent Error: 68.4803001876173  
## [1] 3 class summary: (bad=3,4, OK=5,6, Good=7,8,9)  
## [1] Number Correct: 857  
## [1] Number Wrong: 742  
## [1] Percent Error: 46.4040025015635

## Probably not a good idea to use a white wine model to predict red quality

## Go to results table in ppt