R-Ladies Workshop: Classification using Caret

Ashley Thompson (@ABTresearch) & Maya Alsheh Ali ([maya.alsheh.ali@ki.se](mailto:maya.alsheh.ali@ki.se))

Feburary 2019

## Packages Needed

library(caret)  
library(ggplot2)  
library(car)  
library(ROCR)

## Data Preparation

**Load data**  
*Information on data can be found here:* <https://archive.ics.uci.edu/ml/datasets/Wine%2BQuality>

df = read.csv( 'https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-white.csv' , sep=';' , header=T )  
  
#examine data  
str(df)

## 'data.frame': 4898 obs. of 12 variables:  
## $ fixed.acidity : num 7 6.3 8.1 7.2 7.2 8.1 6.2 7 6.3 8.1 ...  
## $ volatile.acidity : num 0.27 0.3 0.28 0.23 0.23 0.28 0.32 0.27 0.3 0.22 ...  
## $ citric.acid : num 0.36 0.34 0.4 0.32 0.32 0.4 0.16 0.36 0.34 0.43 ...  
## $ residual.sugar : num 20.7 1.6 6.9 8.5 8.5 6.9 7 20.7 1.6 1.5 ...  
## $ chlorides : num 0.045 0.049 0.05 0.058 0.058 0.05 0.045 0.045 0.049 0.044 ...  
## $ free.sulfur.dioxide : num 45 14 30 47 47 30 30 45 14 28 ...  
## $ total.sulfur.dioxide: num 170 132 97 186 186 97 136 170 132 129 ...  
## $ density : num 1.001 0.994 0.995 0.996 0.996 ...  
## $ pH : num 3 3.3 3.26 3.19 3.19 3.26 3.18 3 3.3 3.22 ...  
## $ sulphates : num 0.45 0.49 0.44 0.4 0.4 0.44 0.47 0.45 0.49 0.45 ...  
## $ alcohol : num 8.8 9.5 10.1 9.9 9.9 10.1 9.6 8.8 9.5 11 ...  
## $ quality : int 6 6 6 6 6 6 6 6 6 6 ...

**Modify Outcome variable**  
*Our outcome, quality, is a continuous integer and this classification task requires the outcome to be a binary factor. I’ve then renamed the labels to “bad” & “good” for readability and because “0” & “1” label names create issues in some packages.*

#create new binary variable  
df$quality.bi = NA  
df$quality.bi= car::recode(df$quality, "1:5= 0; 5:10= 1")  
  
#change to factor & change label  
df$quality.bi = as.factor(df$quality.bi)  
levels(df$quality.bi) = c("bad", "good")  
  
#check variable  
table(df$quality.bi)  
  
#remove quality  
df = df[ -c(12)]

**Check for missing values**  
*We must have complete cases for analysis*

row.has.na = apply(df, 1, function(x){any(is.na(x))})  
sum(row.has.na)

**Check variance**

nearZeroVar(df, saveMetrics = TRUE, names=TRUE)

*We see that there are no missing values nor variables with near zero variance. In the case that there were missing values, we could use multiple imputation with the package mice (use CART option) or simply remove the row. In the case that there are variables with near or at zero variance, you should consider removing or combining similar variables to increase power. Ex: combining pollen allegery and animal fur allergy into one allergy.*

## Check Power

**Build Learning Curve**

learning.Curve = learing\_curve\_dat(dat = df,  
 outcome = 'quality.bi',  
 test\_prop = .30, #the % of data you want in testset  
 method = "rf", #random Forest  
 metric = "ROC",  
 trControl = trainControl(classProbs = TRUE,  
 summaryFunction = twoClassSummary))

**Plot**

curve.plot = ggplot(learning.Curve, aes(x = Training\_Size, y = ROC, color = Data)) +  
 geom\_smooth(method = loess, span = .8) +  
 theme\_bw()  
  
curve.plot

*The learning curve is used to determine power estimates and can give information on the bias vs variance trade off. Because the blue line stays at it’s maximum regardless of training size, the data is likely to overfit; we are likely underpowered and will need to use less complex models*

## Data Partition

#Set the fractions of the dataframe you want to split into training,   
# validation, and test.   
fractionTraining = 0.60  
fractionValidation = 0.10  
fractionTest = 0.30  
  
# Compute sample sizes.  
sampleSizeTraining = floor(fractionTraining \* nrow(df))  
sampleSizeValidation = floor(fractionValidation \* nrow(df))  
sampleSizeTest = floor(fractionTest \* nrow(df))  
  
#forloop here   
folds = createFolds(nrow(df), 20) #20 folds  
for(i in unique(folds)) {  
   
 # Create the randomly-sampled indices for the dataframe. Use setdiff() to  
 # avoid overlapping subsets of indices.  
 indicesTraining = sort(sample(seq\_len(nrow(df)), size=sampleSizeTraining))  
 indicesNotTraining = setdiff(seq\_len(nrow(df)), indicesTraining)  
 indicesValidation = sort(sample(indicesNotTraining, size=sampleSizeValidation))  
 indicesTest = setdiff(indicesNotTraining, indicesValidation)  
   
 # Finally, output the three dataframes for training, Validation and test.  
 Trainset = df[indicesTraining, ]  
 Validationset = df[indicesValidation, ]  
 Testset = df[indicesTest, ]  
}

## Build Random Forest Model

**Build Model**

rf.model <- caret::train(quality.bi ~ ., data = Trainset, method = "rf")

**Plot variable importance**

VIrf= varImp(rf.model)  
plot(VIrf)

*This shows how much the model relied certain variables. To understand if a variable was positively or negatively associated with the outcome a partial plot can be created.*

**Test Model on Training set**

train.pred <- predict(rf.model, Trainset)  
confusionMatrix(train.pred, Trainset$quality.bi)

*We see that this model is highly accurate, however this is not surprising since this is the same data the model learned from. Let’s see how it does on an “new” dataset.*

**Test Model on Validation set**

Validation.pred1 = predict(rf.model, Validationset)  
  
#examine prediction, outputting only a table  
Validation.pred.table = table(observed = Validationset$quality.bi, predicted = Validation.pred1)  
Validation.pred.table

**Plot AUC using ROCR package**

#determine performance with ROCR  
Validationpredict <- ROCR::prediction(as.numeric(Validationpred1), as.numeric(Validationset$quality.bi))  
Validationpredict.perf = ROCR::performance(Validationpredict,"tpr","fpr")  
  
#plot  
plot(Validationpredict.perf,main="ROC Curve for Random Forest",col=2,lwd=2, print.auc = TRUE)  
abline(a=0,b=1,lwd=2,lty=2,col="gray")  
  
#print  
Validationrfauc <- ROCR::performance(Validationpredict, measure = "auc")@y.values[[1]]  
Validationrfauc

*The performance dropped significantly! Let’s see if we can increase model performance.*

## Tune Model

*There are several ways to choose optimal parameters in your model: manually selecting them (time consuming), random search, and grid search. According to the No Free Lunch Theorem, on average grid search and random search have equal performance. Today we’ll be using Random Search.*

**Set up random search**

#this code sets the method to repeated cross validation, 10 folds, 3 times.  
control <- trainControl(method="repeatedcv", number=10, repeats=3, search="random")

**Create new model with random search**  
*The random search will rerun the model to find optimal parameters with repeated cross validation*  
*note: this may take a few minutes to run*

rf.random.model <- caret::train(quality.bi ~ ., data=Trainset, method="rf", tuneLength=15, trControl=control)  
print(rf.random.model) #examine parameter performance

**Plot Variable Importance**

VIrfrandom= varImp(rf.random.model)  
plot(VIrfrandom)

*Did the variable importance change between the models?*

**Predict Validation set using the new model**

Validation.pred2 = predict(rf.random.model, Validationset)  
  
#examine prediction  
Validation.pred.table2 = table(observed = Validationset$quality.bi, predicted = Validation.pred2)  
Validation.pred.table2

**Plot AUC**

#Determine performance  
Validation.predict2 <- ROCR::prediction(as.numeric(Validation.pred2), as.numeric(Validationset$quality.bi))  
Validation.predict.perf2 = ROCR::performance(Validation.predict2,"tpr","fpr")  
  
#plot  
plot(Validation.predict.perf2,main="ROC Curve for Random Forest",col=2,lwd=2, print.auc = TRUE)  
abline(a=0,b=1,lwd=2,lty=2,col="gray")  
  
#print   
Validation.rf.auc2 <- ROCR::performance(Validation.predict2, measure = "auc")@y.values[[1]]  
Validation.rf.auc2

*An AUC of ~.8! A fairly good AUC, let’s put it to the test set. You can repeat the model tuning set step as many times as it takes you to get a model you’re satisfied with.*

## Test model on Test set

*Remember you can only do this step one time*

test.pred= predict(rf.model, newdata = Testset)

**Examine**

confusionMatrix(test.pred, Testset$quality.bi)

**Plot AUC**

test.predict <- ROCR::prediction(as.numeric(test.pred), as.numeric(Testset$quality.bi))  
test.predict.perf = ROCR::performance(test.predict,"tpr","fpr")  
plot(test.predict.perf,main="ROC Curve for Random Forest",col=2,lwd=2, print.auc = TRUE)  
abline(a=0,b=1,lwd=2,lty=2,col="gray")  
#print Final AUC  
final.auc <- ROCR::performance(test.predict, measure = "auc")@y.values[[1]]  
final.auc

*All finished! Now try on your own with the red wine data from the same link.*