Chapter 17 Interactive Notebook for Students

Ram Gopal, Dan Philps, and Tillman Weyde

2022

Contents

Loading the required packages and the Wine data file.

```
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

```
library(rattle)
```

```
## Loading required package: tibble

## Loading required package: bitops

## Rattle: A free graphical interface for data science with R.
## Version 5.5.1 Copyright (c) 2006-2021 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```r
print(names(wine))
```

```
##  [1] "Type"            "Alcohol"         "Malic"           "Ash"
##  [5] "Alcalinity"      "Magnesium"       "Phenols"         "Flavanoids"
##  [9] "Nonflavanoids"   "Proanthocyanins" "Color"           "Hue"
## [13] "Dilution"        "Proline"
```

```r
print(nrow(wine))
```

```
## [1] 178
```

Training and Test Data set creation

```r
index = createDataPartition(wine$Type,p = 0.6,list = F)
train_wi <- wine[index,]
test_wi <- wine[-index,]
```

Run DT algorithm

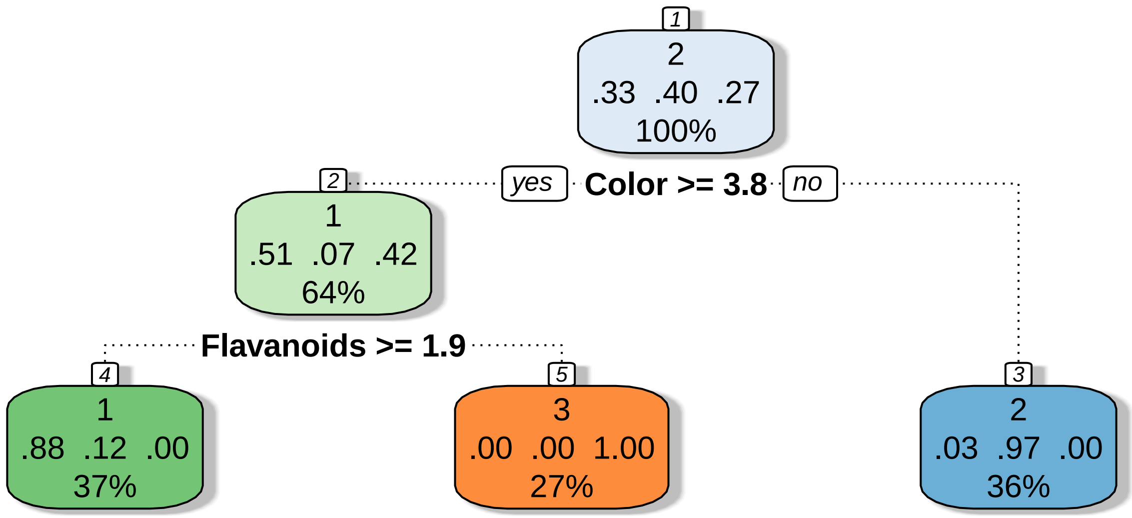
- Train the model

```r
tree = train(Type ~ ., data=train_wi, method='rpart')
fitted <- predict(tree)
table(train_wi$Type,fitted)
```

```
##    fitted
##      1  2  3
##   1 35  1  0
##   2  5 38  0
##   3  0  0 29
```

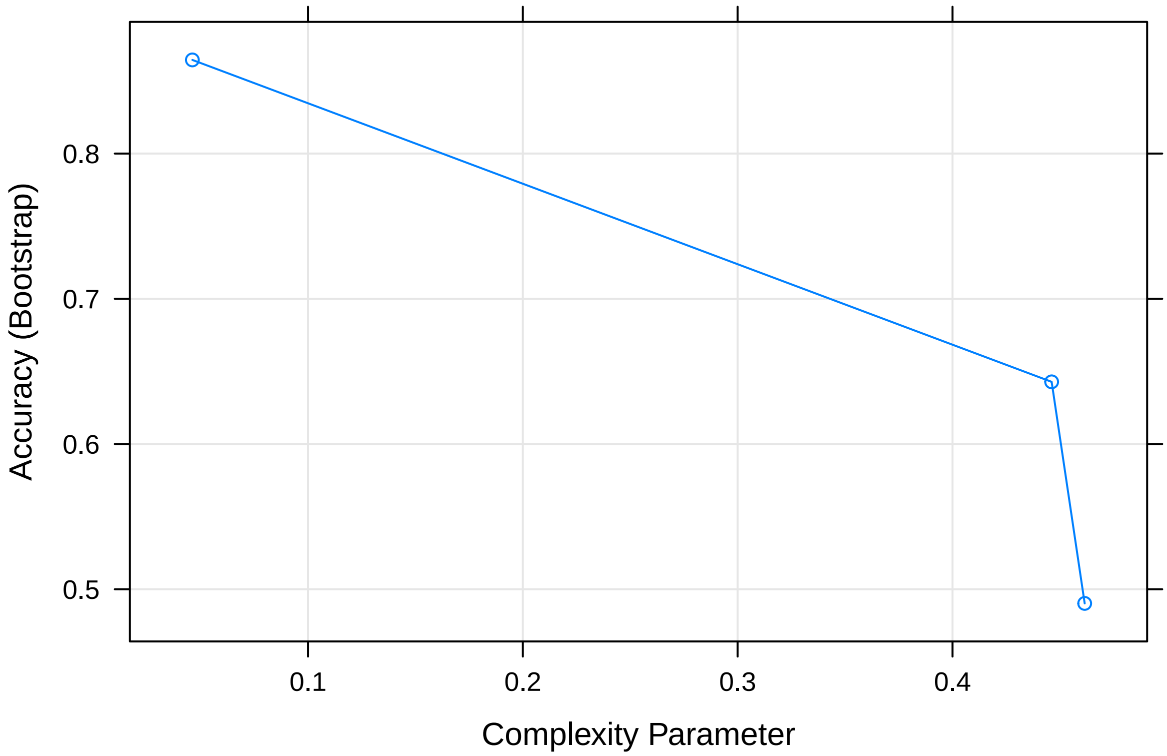
- Create a fancy tree plot with the rattle package

```r
fancyRpartPlot(tree$finalModel,sub = "")
```



- Plotting the model shows how the various iterations of hyper-parameter search was performed. In this case the hyper-parameter is cp (complexity parameter)

```
plot(tree)
```



- Predict on the test data.

```
predicted = predict(tree,newdata = test_wi)
table(test_wi$Type,predicted)
```

```
##    predicted
##      1  2  3
##   1 20  3  0
##   2  4 22  2
##   3  0  0 19
```

- Create the confusion matrix for the training and test data

```
# Performance on Test Data
confusionMatrix(reference = test_wi$Type, data = predicted, mode='everything', positive='MM')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3
##          1 20  4  0
##          2  3 22  0
##          3  0  2 19
##
```

```
## Overall Statistics
##
##                Accuracy : 0.8714
##                  95% CI : (0.7699, 0.9395)
##     No Information Rate : 0.4
##     P-Value [Acc > NIR] : 3.853e-16
##
##                   Kappa : 0.8061
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            0.8696   0.7857   1.0000
## Specificity            0.9149   0.9286   0.9608
## Pos Pred Value         0.8333   0.8800   0.9048
## Neg Pred Value         0.9348   0.8667   1.0000
## Precision              0.8333   0.8800   0.9048
## Recall                 0.8696   0.7857   1.0000
## F1                     0.8511   0.8302   0.9500
## Prevalence             0.3286   0.4000   0.2714
## Detection Rate         0.2857   0.3143   0.2714
## Detection Prevalence   0.3429   0.3571   0.3000
## Balanced Accuracy      0.8922   0.8571   0.9804
```

```
# Performance on Training Data
confusionMatrix(reference = train_wi$Type, data = fitted, mode='everything', positive='MM')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3
##          1 35  5  0
##          2  1 38  0
##          3  0  0 29
##
## Overall Statistics
##
##                Accuracy : 0.9444
##                  95% CI : (0.883, 0.9793)
##     No Information Rate : 0.3981
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9159
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            0.9722   0.8837   1.0000
## Specificity            0.9306   0.9846   1.0000
## Pos Pred Value         0.8750   0.9744   1.0000
```

```
## Neg Pred Value         0.9853   0.9275   1.0000
## Precision              0.8750   0.9744   1.0000
## Recall                 0.9722   0.8837   1.0000
## F1                     0.9211   0.9268   1.0000
## Prevalence             0.3333   0.3981   0.2685
## Detection Rate         0.3241   0.3519   0.2685
## Detection Prevalence   0.3704   0.3611   0.2685
## Balanced Accuracy      0.9514   0.9342   1.0000
```

Run KNN algorithm

- Check the hyper-parameter for KNN

```
modelLookup('knn')
```

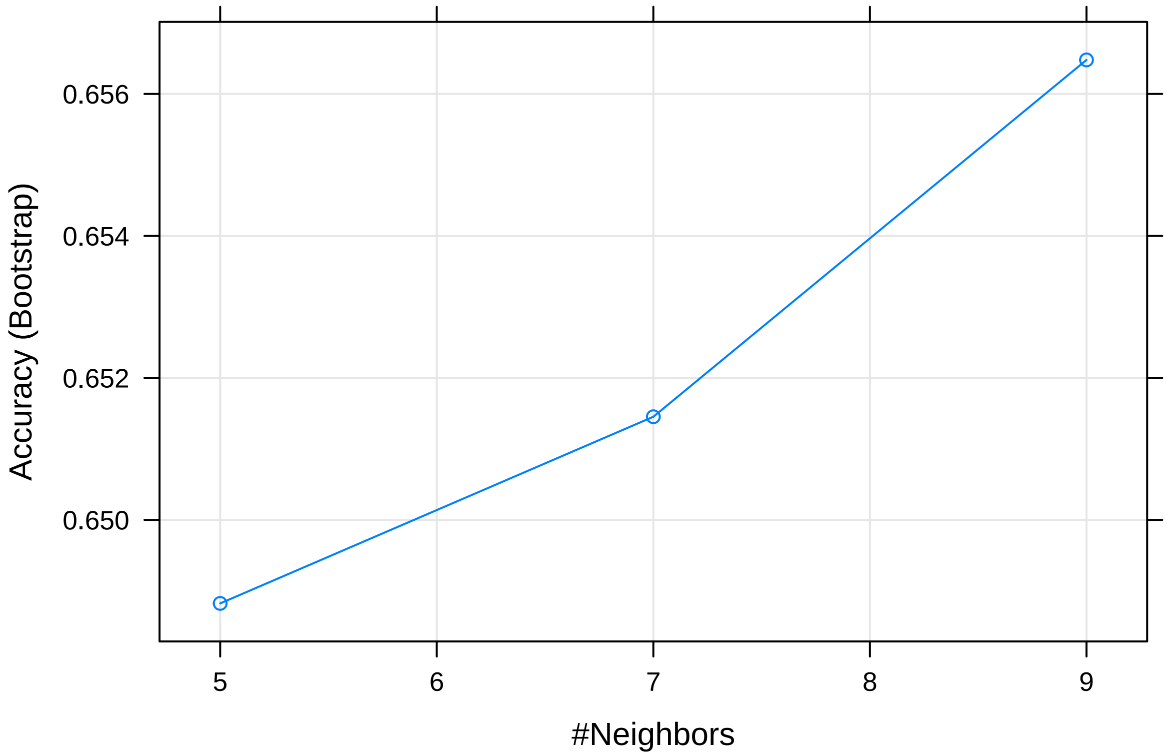
```
##   model parameter      label forReg forClass probModel
## 1   knn         k #Neighbors   TRUE     TRUE      TRUE
```

- Train and plot the KNN model. The plot indicates the best number of neighbors to use.

```
model_knn= train(Type ~ ., data=train_wi, method='knn')
model_knn
```

```
## k-Nearest Neighbors
##
## 108 samples
##  13 predictor
##   3 classes: '1', '2', '3'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 108, 108, 108, 108, 108, 108, ...
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   5  0.6488239  0.4666208
##   7  0.6514530  0.4695061
##   9  0.6564791  0.4785992
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.
```

```
plot(model_knn)
```



- Create the confusion matrix for the training and test data

```
fitted <- predict(model_knn)
predicted <- predict(model_knn, test_wi)
# Performance on Test Data
confusionMatrix(reference = test_wi$Type, data = predicted)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3
##          1 18  0  0
##          2  0 23  5
##          3  5  5 14
##
## Overall Statistics
##
##                Accuracy : 0.7857
##                  95% CI : (0.6713, 0.8748)
##     No Information Rate : 0.4
##     P-Value [Acc > NIR] : 5.341e-11
##
##                   Kappa : 0.6765
##
##  Mcnemar's Test P-Value : NA
```

```
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            0.7826   0.8214   0.7368
## Specificity            1.0000   0.8810   0.8039
## Pos Pred Value         1.0000   0.8214   0.5833
## Neg Pred Value         0.9038   0.8810   0.8913
## Prevalence             0.3286   0.4000   0.2714
## Detection Rate         0.2571   0.3286   0.2000
## Detection Prevalence   0.2571   0.4000   0.3429
## Balanced Accuracy      0.8913   0.8512   0.7704
```

```
# Performance on Training Data
confusionMatrix(reference = train_wi$Type, data = fitted)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3
##          1 30  2  0
##          2  1 28  9
##          3  5 13 20
##
## Overall Statistics
##
##                Accuracy : 0.7222
##                  95% CI : (0.6278, 0.8041)
##     No Information Rate : 0.3981
##     P-Value [Acc > NIR] : 9.039e-12
##
##                   Kappa : 0.5833
##
##  Mcnemar's Test P-Value : 0.1087
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            0.8333   0.6512   0.6897
## Specificity            0.9722   0.8462   0.7722
## Pos Pred Value         0.9375   0.7368   0.5263
## Neg Pred Value         0.9211   0.7857   0.8714
## Prevalence             0.3333   0.3981   0.2685
## Detection Rate         0.2778   0.2593   0.1852
## Detection Prevalence   0.2963   0.3519   0.3519
## Balanced Accuracy      0.9028   0.7487   0.7309
```

- Get the overall model accuracy

```
paste("KNN Test Accuracy = ",confusionMatrix(reference = test_wi$Type, data = predicted)$overall[1])
```

```
## [1] "KNN Test Accuracy =  0.785714285714286"
```

```r
predicted = predict(tree,newdata = test_wi)
paste("DT Test Accuracy = ",confusionMatrix(reference = test_wi$Type, data = predicted)$overall[1])
```

```
## [1] "DT Test Accuracy =  0.871428571428571"
```

Cross-Validation

- 10-fold

```r
model_knn_cv_10= train(Type ~ ., data=wine, method='knn',metric="Accuracy",
                    trControl=trainControl(method = 'cv',number = 10))
paste("Mean of accuracy 10 fold = ",mean(model_knn_cv_10$resample$Accuracy))
```

```
## [1] "Mean of accuracy 10 fold =  0.701633986928105"
```

```r
paste("sd of accuracy 10 fold = ",sd(model_knn_cv_10$resample$Accuracy))
```

```
## [1] "sd of accuracy 10 fold =  0.100632018172329"
```

- 20-fold

```r
model_knn_cv_20= train(Type ~ ., data=wine, method='knn',metric="Accuracy",
                    trControl=trainControl(method = 'cv',number = 20))
paste("Mean of accuracy 20 fold = ",mean(model_knn_cv_20$resample$Accuracy))
```

```
## [1] "Mean of accuracy 20 fold =  0.703710317460317"
```

```r
paste("sd of accuracy 20 fold = ",sd(model_knn_cv_20$resample$Accuracy))
```

```
## [1] "sd of accuracy 20 fold =  0.158669854616182"
```

- LOOCV

```r
model_knn_LOOCV= train(Type ~ ., data=wine, method='knn',metric="Accuracy",
                    trControl=trainControl(method = 'cv',number = nrow(wine)))
```

```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :
## There were missing values in resampled performance measures.
```

```r
paste("Mean of accuracy LOOCV = ",mean(model_knn_LOOCV$resample$Accuracy,na.rm = T))
```

```
## [1] "Mean of accuracy LOOCV =  0.719101123595506"
```

```r
paste("sd of accuracy LOOCV = ",sd(model_knn_LOOCV$resample$Accuracy,na.rm = T))
```

```
## [1] "sd of accuracy LOOCV =  0.450706013516793"
```

- Stratified Sampling

```r
folds = createMultiFolds(wine$Type, k = 10)
model_knn_strat= train(Type ~ ., data=wine, method='knn',metric="Accuracy",
                   trControl=trainControl(index = folds))
paste("Mean for stratified sample = ",mean(model_knn_strat$resample$Accuracy))
```

```
## [1] "Mean for stratified sample =  0.699834881320949"
```

```r
paste("sde for stratified sample = ",sd(model_knn_strat$resample$Accuracy))
```

```
## [1] "sde for stratified sample =  0.107235979605779"
```

Signifance of Model Differences

- Compare 20-fold DT model with KNN model

```r
model_dt_cv_20= train(Type ~ ., data=wine, method='rpart',metric="Accuracy",
                   trControl=trainControl(method = 'cv',number = 20))
paste("Mean accuracy of KNN= ",mean(model_knn_cv_20$resample$Accuracy))
```

```
## [1] "Mean accuracy of KNN=  0.703710317460317"
```

```r
paste("Mean accuracy of DT = ",mean(model_dt_cv_20$resample$Accuracy))
```

```
## [1] "Mean accuracy of DT =  0.842916666666667"
```

```r
wilcox.test(model_dt_cv_20$resample$Accuracy,model_knn_cv_20$resample$Accuracy)
```

```
## Warning in wilcox.test.default(model_dt_cv_20$resample$Accuracy,
## model_knn_cv_20$resample$Accuracy): cannot compute exact p-value with ties
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  model_dt_cv_20$resample$Accuracy and model_knn_cv_20$resample$Accuracy
## W = 299.5, p-value = 0.006825
## alternative hypothesis: true location shift is not equal to 0
```

- Compare Stratified CV DT model with KNN model

```r
folds = createMultiFolds(wine$Type, k = 20)
model_knn_strat= train(Type ~ ., data=wine, method='knn',metric="Accuracy",
                   trControl=trainControl(index = folds))
model_dt_strat= train(Type ~ ., data=wine, method='rpart',metric="Accuracy",
                   trControl=trainControl(index = folds))
paste("Mean accuracy KNN = ",mean(model_knn_strat$resample$Accuracy))
```

```
## [1] "Mean accuracy KNN =  0.71052380952381"
```

```r
paste("Mean accuracy DT = ",mean(model_dt_strat$resample$Accuracy))
```

```
## [1] "Mean accuracy DT =  0.840809523809524"
```

```r
paste("sd accuracy KNN = ",sd(model_knn_strat$resample$Accuracy))
```

```
## [1] "sd accuracy KNN =  0.138029182405176"
```

```r
paste("sd accuracy DT = ",sd(model_dt_strat$resample$Accuracy))
```

```
## [1] "sd accuracy DT =  0.109911579178145"
```

```r
wilcox.test(model_knn_strat$resample$Accuracy,model_dt_strat$resample$Accuracy)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  model_knn_strat$resample$Accuracy and model_dt_strat$resample$Accuracy
## W = 2429.5, p-value = 2.803e-10
## alternative hypothesis: true location shift is not equal to 0
```

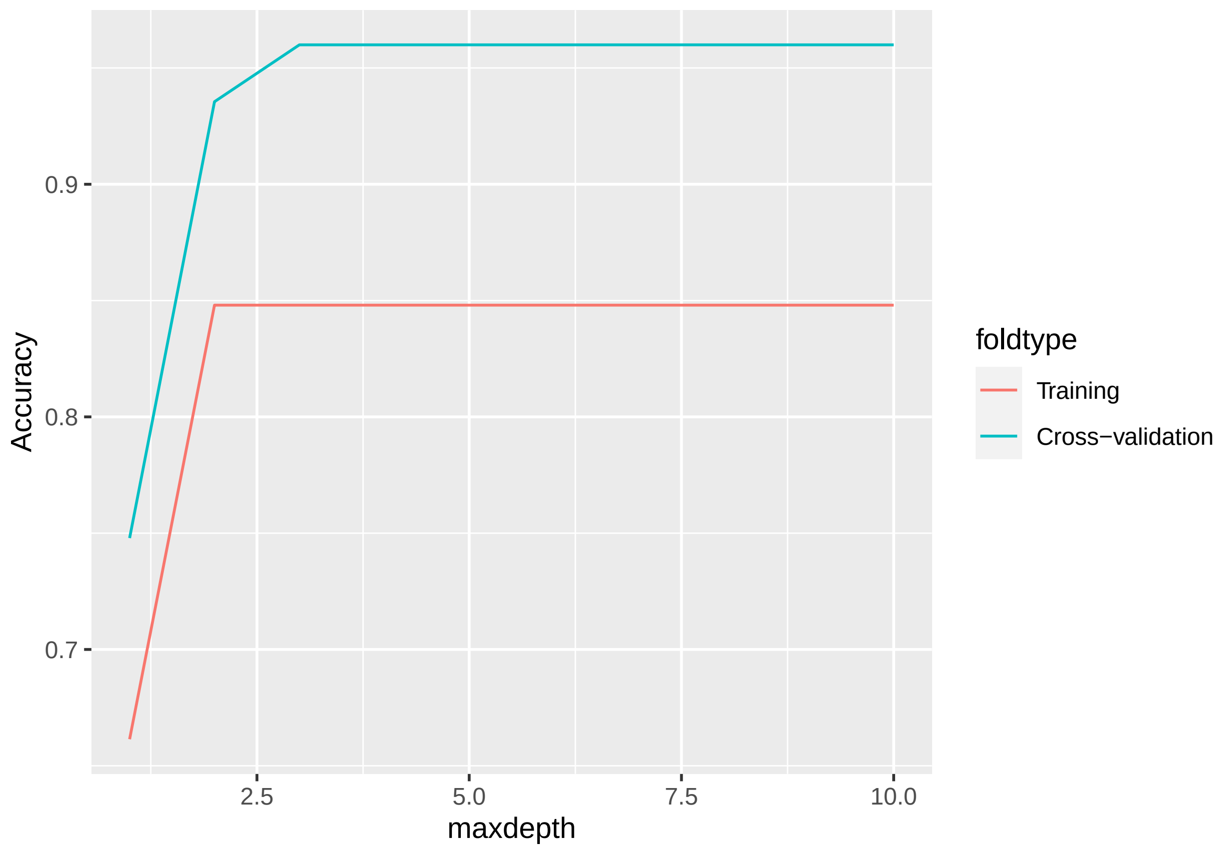
Hyperparameter tuning

- Decision Tree

```r
folds = createMultiFolds(train_wi$Type, k = 20)
model_dt_strat_fold= train(Type ~ ., data=train_wi, method='rpart2',metric="Accuracy",
                    trControl=trainControl(index = folds),
                    tuneGrid = expand.grid(maxdepth = 1:10))
folds = createMultiFolds(train_wi$Type, k = 2)
model_dt_strat_nofold= train(Type ~ ., data=train_wi, method='rpart2',metric="Accuracy",
                    trControl=trainControl(index = folds),
                    tuneGrid = expand.grid(maxdepth = 1:10))
```

- Plot

```r
df1 = model_dt_strat_nofold$results[,c(1,2)]
df1$foldtype = as.factor("Training")
df2 = model_dt_strat_fold$results[,c(1,2)]
df2$foldtype = as.factor("Cross-validation")
df = rbind(df1,df2)
ggplot(data=df,aes(x=maxdepth,y=Accuracy,color=foldtype))+
  geom_line()
```

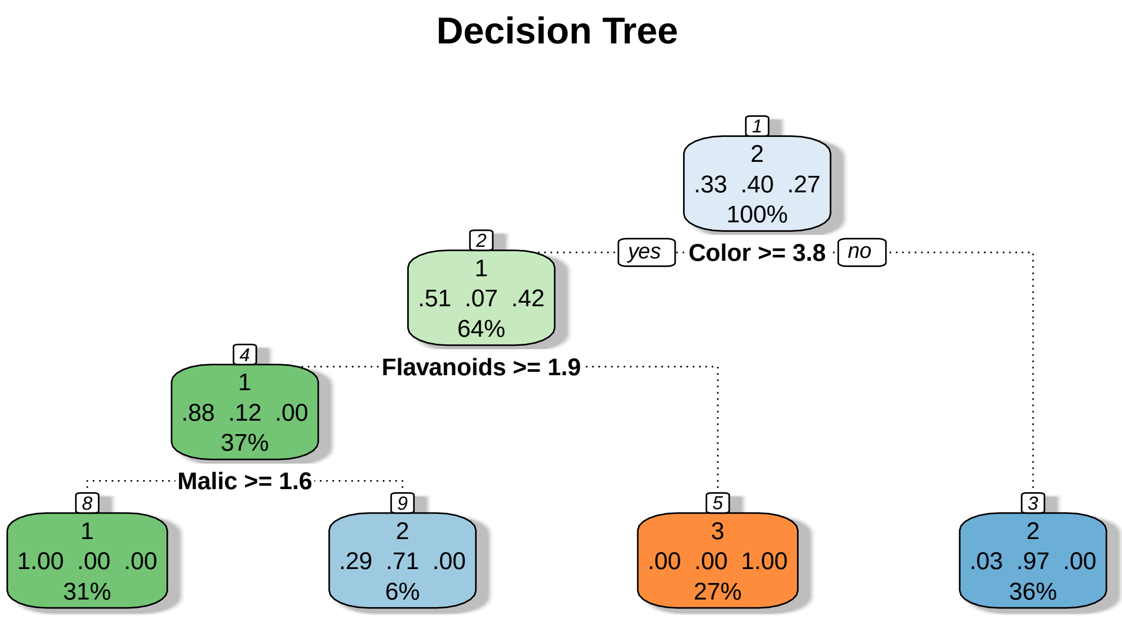


- See and plot the final tree

```
model_dt_strat_fold$finalModel
```

```
## n= 108
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 108 65 2 (0.33333333 0.39814815 0.26851852)
##   2) Color>=3.82 69 34 1 (0.50724638 0.07246377 0.42028986)
##     4) Flavanoids>=1.88 40  5 1 (0.87500000 0.12500000 0.00000000)
##       8) Malic>=1.55 33  0 1 (1.00000000 0.00000000 0.00000000) *
##       9) Malic< 1.55 7  2 2 (0.28571429 0.71428571 0.00000000) *
##     5) Flavanoids< 1.88 29  0 3 (0.00000000 0.00000000 1.00000000) *
##   3) Color< 3.82 39  1 2 (0.02564103 0.97435897 0.00000000) *
```

```
fancyRpartPlot(model_dt_strat_fold$finalModel,main = "Decision Tree",sub = "")
```



Estimating the Model Performance

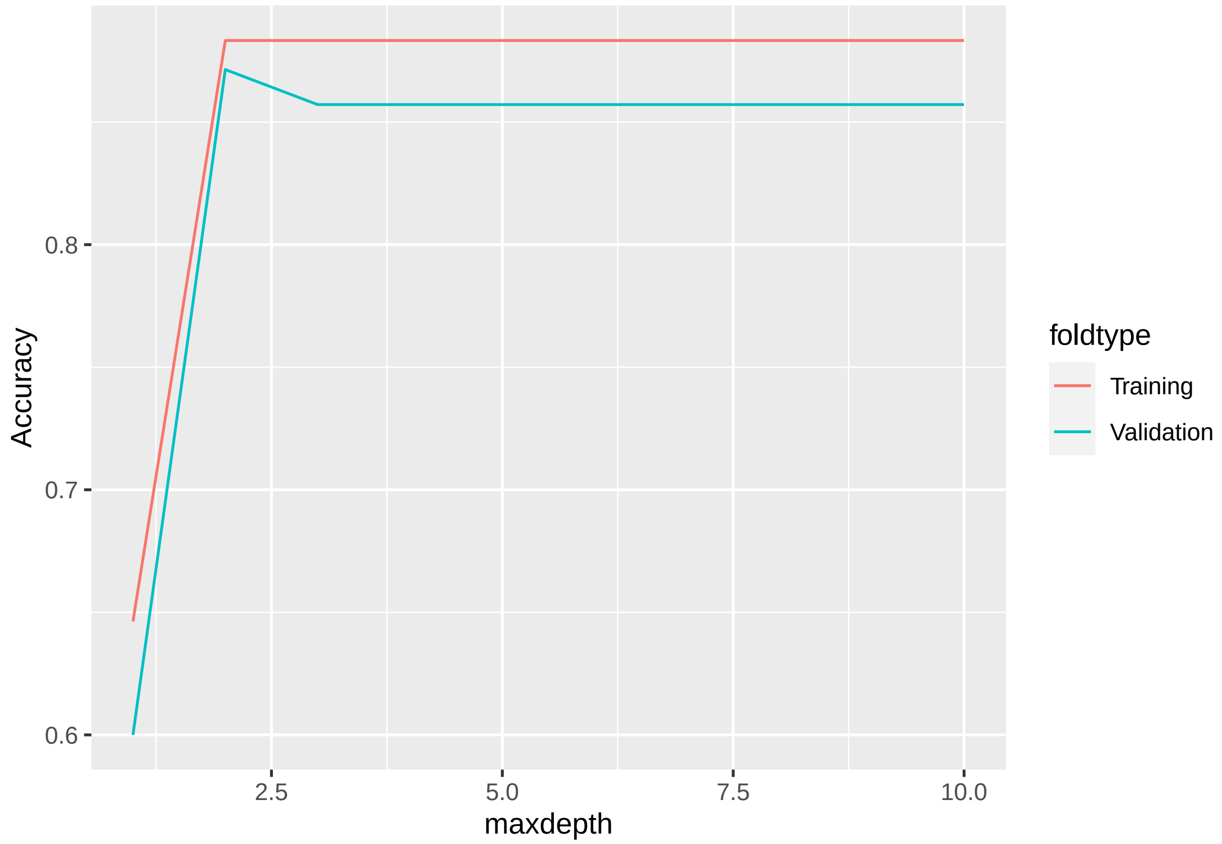
- For 2 folds

```
df = data.frame()
folds = createMultiFolds(train_wi$Type, k = 2)
for (md in 1:10){
model_dt_strat_nofold= train(Type ~ ., data=train_wi, method='rpart2',metric="Accuracy",trControl=train(
                    tuneGrid = expand.grid(maxdepth = md))
df = rbind(df,c(md,"Training",model_dt_strat_nofold$results$Accuracy))
pred = predict(model_dt_strat_nofold,newdata = test_wi)
t1 = table(test_wi$Type,pred)
accu = sum(t1[1,1]+t1[2,2]+t1[3,3])/nrow(test_wi)
df = rbind(df,c(md,"Validation",accu))
}
colnames(df) = c("maxdepth","foldtype","Accuracy")
```

- Plot

```
df$maxdepth = as.numeric(df$maxdepth)
df$Accuracy = as.numeric(df$Accuracy)
```

```
ggplot(data=df,aes(x=maxdepth,y=Accuracy,color=foldtype))+
  geom_line()
```



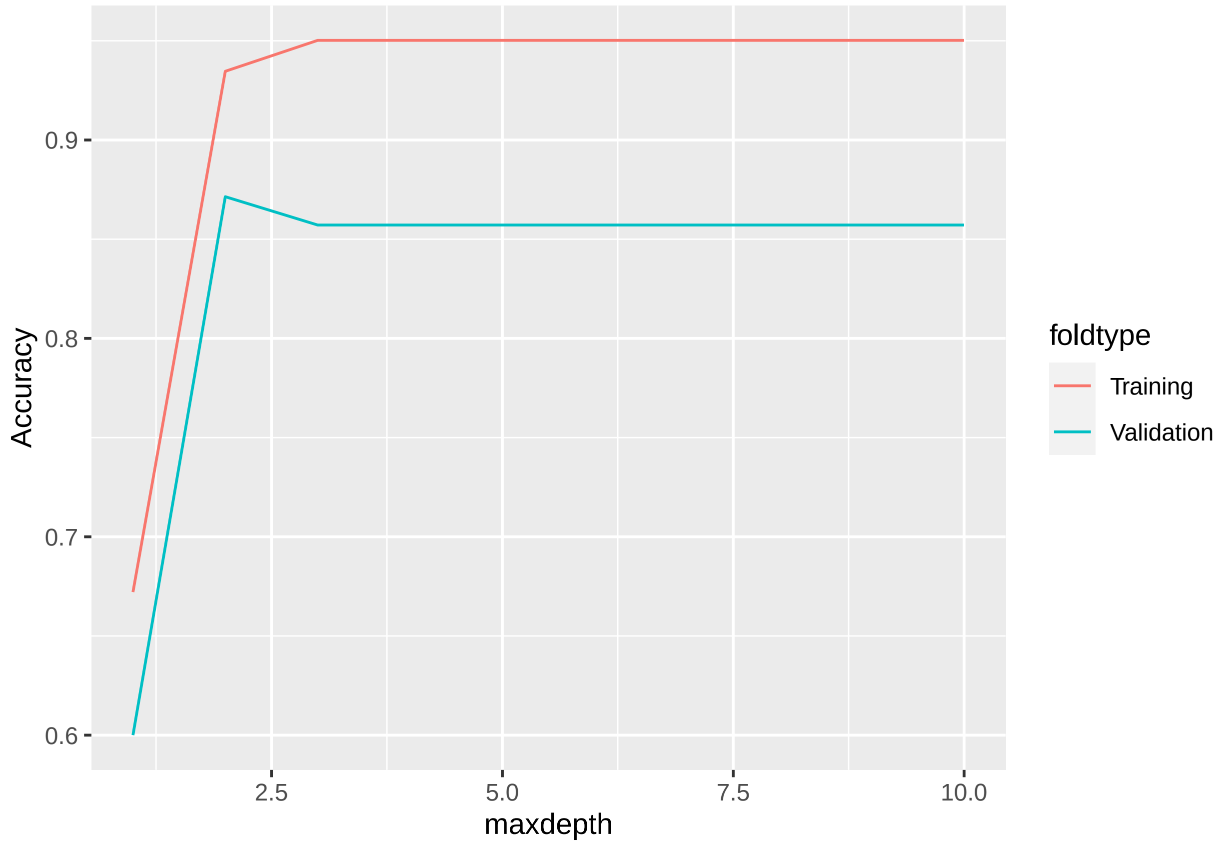
- For 10 folds

```
df = data.frame()
folds = createMultiFolds(train_wi$Type, k = 20)
for (md in 1:10){
  model_dt_strat_fold= train(Type ~ ., data=train_wi,  method='rpart2',metric="Accuracy",trControl=trai
                    tuneGrid = expand.grid(maxdepth = md))
df = rbind(df,c(md,"Training",model_dt_strat_fold$results$Accuracy))
pred = predict(model_dt_strat_fold,newdata = test_wi)
t1 = table(test_wi$Type,pred)
accu = sum(t1[1,1]+t1[2,2]+t1[3,3])/nrow(test_wi)
df = rbind(df,c(md,"Validation",accu))
}
colnames(df) = c("maxdepth","foldtype","Accuracy")
```

- Plot

```
df$maxdepth = as.numeric(df$maxdepth)
df$Accuracy = as.numeric(df$Accuracy)

ggplot(data=df,aes(x=maxdepth,y=Accuracy,color=foldtype))+
  geom_line()
```



Final Test of Performance

- Test accuracy for 2 folds

```
folds = createMultiFolds(train_wi$Type, k = 2)
model_dt_strat_nofold= train(Type ~ ., data=train_wi, method='rpart2',metric="Accuracy",trControl=trainC
                     tuneGrid = expand.grid(maxdepth = 3))
pred = predict(model_dt_strat_nofold,newdata = test_wi)
t1 = table(test_wi$Type,pred)
accu = sum(t1[1,1]+t1[2,2]+t1[3,3])/nrow(test_wi)
paste("Accuracy = ", accu)
```

```
## [1] "Accuracy =  0.857142857142857"
```

- Performance of stratified CV

```
folds = createMultiFolds(wine$Type, k = 20)
model_dt_strat_fold= train(Type ~ ., data=wine,  method='rpart2',metric="Accuracy",trControl=trainContro
                     tuneGrid = expand.grid(maxdepth = 3))
paste("CV Accuracy - Mean = ",mean(model_dt_strat_fold$resample$Accuracy))
```

```
## [1] "CV Accuracy - Mean =  0.868333333333333"
```

```r
paste("CV Accuracy - sd = ",sd(model_dt_strat_fold$resample$Accuracy))
```

```
## [1] "CV Accuracy - sd =  0.0945739193051908"
```