Chapter 17 Interactive Notebook for Instructors

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.
```

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

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

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

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

Training and Test Data set creation

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

Models offered in the Caret package

- It will be useful to show students all the models that the caret package supports. One can also easily see what hyper-parameters can be set for a model.

```
# All the models supported by the caret package
modelnames <- paste(names(getModelInfo()), collapse=',  ')
modelnames
```

```
## [1] "ada,  AdaBag,  AdaBoost.M1,  adaboost,  amdai,  ANFIS,  avNNet,  awnb,  awtan,  bag,  bagEarth,
```

```
modelLookup(list('rpart'))
```

```
##   model parameter                label forReg forClass probModel
## 1 rpart        cp Complexity Parameter   TRUE     TRUE      TRUE
```

Run DT algorithm

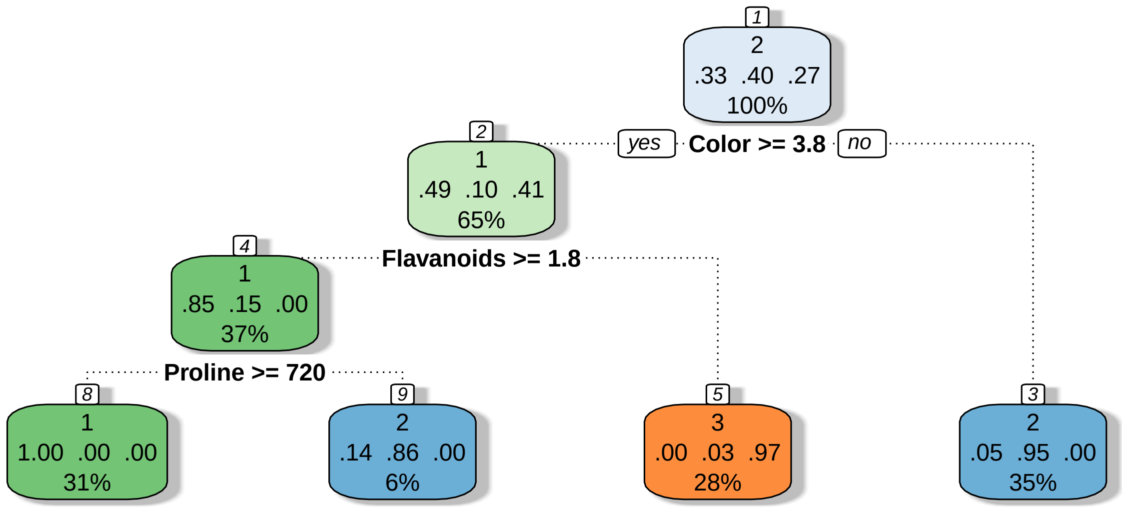
- Train the model

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

```
##    fitted
##      1  2  3
##   1 33  3  0
##   2  0 42  1
##   3  0  0 29
```

- Create a fancy tree plot with the rattle package

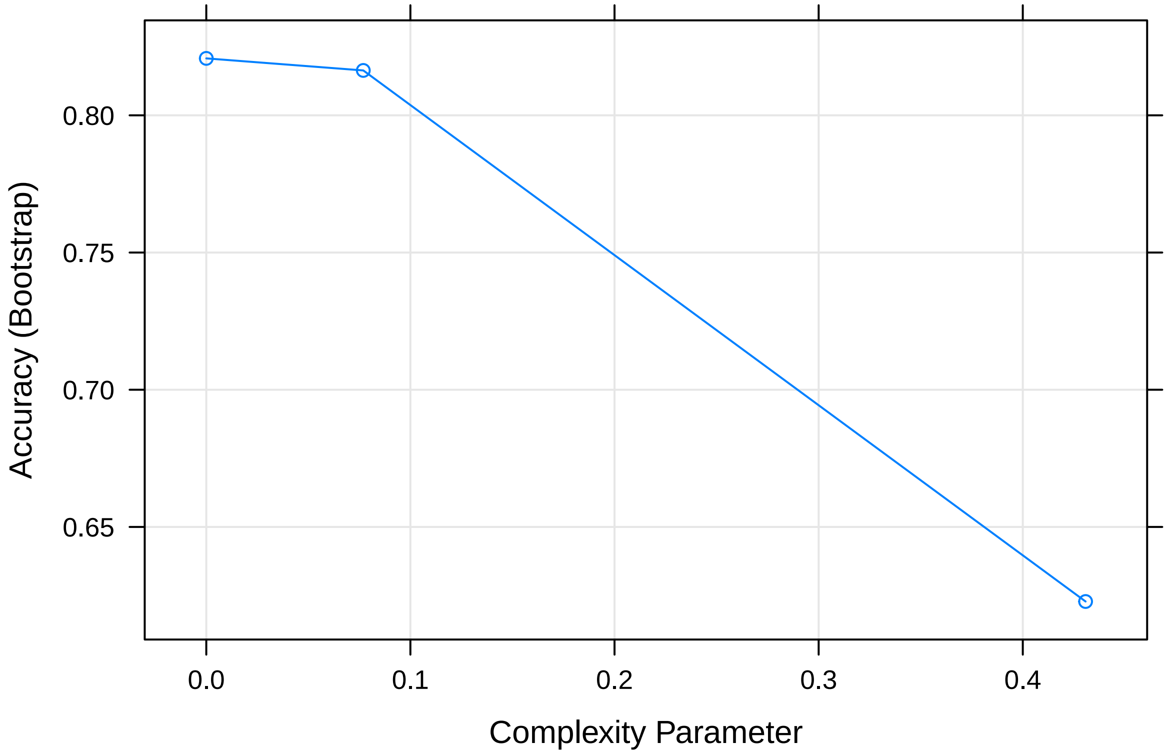
```
fancyRpartPlot(tree$finalModel)
```



Rattle 2022−Aug−27 16:47:39 rgopal

- 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 21  2  0
##   2  0 27  1
##   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 21  0  0
##          2  2 27  0
##          3  0  1 19
##
```

```
## Overall Statistics
##
##                Accuracy : 0.9571
##                  95% CI : (0.8798, 0.9911)
##     No Information Rate : 0.4
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9349
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            0.9130   0.9643   1.0000
## Specificity            1.0000   0.9524   0.9804
## Pos Pred Value         1.0000   0.9310   0.9500
## Neg Pred Value         0.9592   0.9756   1.0000
## Precision              1.0000   0.9310   0.9500
## Recall                 0.9130   0.9643   1.0000
## F1                     0.9545   0.9474   0.9744
## Prevalence             0.3286   0.4000   0.2714
## Detection Rate         0.3000   0.3857   0.2714
## Detection Prevalence   0.3000   0.4143   0.2857
## Balanced Accuracy      0.9565   0.9583   0.9902
```

```
# 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 33  0  0
##          2  3 42  0
##          3  0  1 29
##
## Overall Statistics
##
##                Accuracy : 0.963
##                  95% CI : (0.9079, 0.9898)
##     No Information Rate : 0.3981
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9437
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            0.9167   0.9767   1.0000
## Specificity            1.0000   0.9538   0.9873
## Pos Pred Value         1.0000   0.9333   0.9667
```

```
## Neg Pred Value         0.9600   0.9841   1.0000
## Precision              1.0000   0.9333   0.9667
## Recall                 0.9167   0.9767   1.0000
## F1                     0.9565   0.9545   0.9831
## Prevalence             0.3333   0.3981   0.2685
## Detection Rate         0.3056   0.3889   0.2685
## Detection Prevalence   0.3056   0.4167   0.2778
## Balanced Accuracy      0.9583   0.9653   0.9937
```

Run KNN algorithm

- Check the hyper-parameter for KNN

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

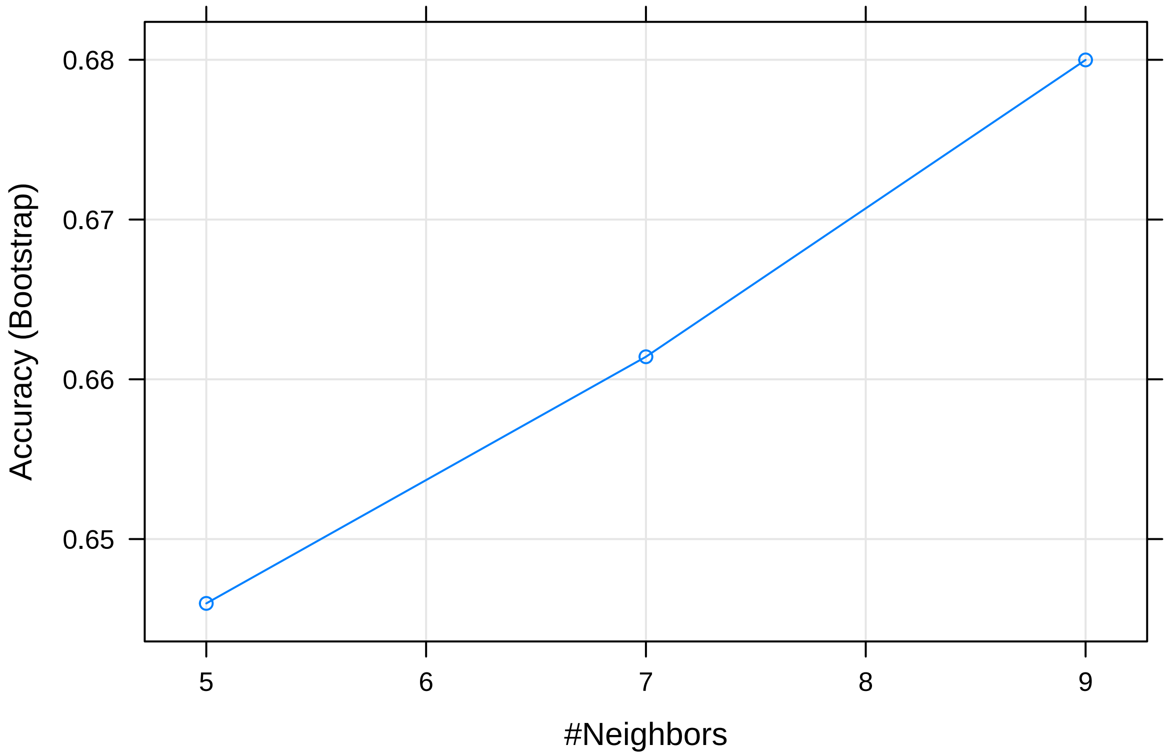
```
##   model parameter      label forReg forClass probModel
## 1   knn         k #Neighbors   TRUE     TRUE      TRUE
```

- 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  0.6459726  0.4603604
##   7  0.6614112  0.4825981
##   9  0.6799933  0.5117634
##
## 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

```r
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 19  1  1
##          2  0 18  9
##          3  4  9  9
##
## Overall Statistics
##
##                Accuracy : 0.6571
##                  95% CI : (0.534, 0.7665)
##     No Information Rate : 0.4
##     P-Value [Acc > NIR] : 1.23e-05
##
##                   Kappa : 0.482
##
##  Mcnemar's Test P-Value : 0.4235
```

```
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            0.8261   0.6429   0.4737
## Specificity            0.9574   0.7857   0.7451
## Pos Pred Value         0.9048   0.6667   0.4091
## Neg Pred Value         0.9184   0.7674   0.7917
## Prevalence             0.3286   0.4000   0.2714
## Detection Rate         0.2714   0.2571   0.1286
## Detection Prevalence   0.3000   0.3857   0.3143
## Balanced Accuracy      0.8918   0.7143   0.6094
```

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

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  1  2  3
##          1 32  2  2
##          2  0 30  7
##          3  4 11 20
##
## Overall Statistics
##
##                Accuracy : 0.7593
##                  95% CI : (0.6675, 0.8363)
##     No Information Rate : 0.3981
##     P-Value [Acc > NIR] : 2.566e-14
##
##                   Kappa : 0.6382
##
##  Mcnemar's Test P-Value : 0.3136
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3
## Sensitivity            0.8889   0.6977   0.6897
## Specificity            0.9444   0.8923   0.8101
## Pos Pred Value         0.8889   0.8108   0.5714
## Neg Pred Value         0.9444   0.8169   0.8767
## Prevalence             0.3333   0.3981   0.2685
## Detection Rate         0.2963   0.2778   0.1852
## Detection Prevalence   0.3333   0.3426   0.3241
## Balanced Accuracy      0.9167   0.7950   0.7499
```

- Get the overall model accuracy

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

```
## [1] "KNN Test Accuracy =  0.657142857142857"
```

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

```
## [1] "DT Test Accuracy =  0.957142857142857"
```

Cross-Validation

- 10-fold

```
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 =  0.712934296525628"
```

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

```
## [1] "sd of accuracy 10 fold =  0.0890832959646867"
```

- 20-fold

```
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 =  0.71375"
```

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

```
## [1] "sd of accuracy 20 fold =  0.162472757250571"
```

- LOOCV

```
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.
```

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

```
## [1] "Mean of accuracy LOOCV =  0.707865168539326"
```

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

```
## [1] "sd of accuracy LOOCV =  0.456026740989417"
```

- 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 =  0.71359133126935"
```

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

```
## [1] "sde for stratified sample =  0.0668273702592139"
```

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=  0.71375"
```

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

```
## [1] "Mean accuracy of DT =  0.832777777777778"
```

```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 = 294.5, p-value = 0.0104
## 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 =  0.719626984126984"
```

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

```
## [1] "Mean accuracy DT =  0.835626984126984"
```

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

```
## [1] "sd accuracy KNN =  0.144762420526761"
```

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

```
## [1] "sd accuracy DT =  0.107772979177569"
```

```
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 = 2588, p-value = 2.952e-09
## alternative hypothesis: true location shift is not equal to 0
```

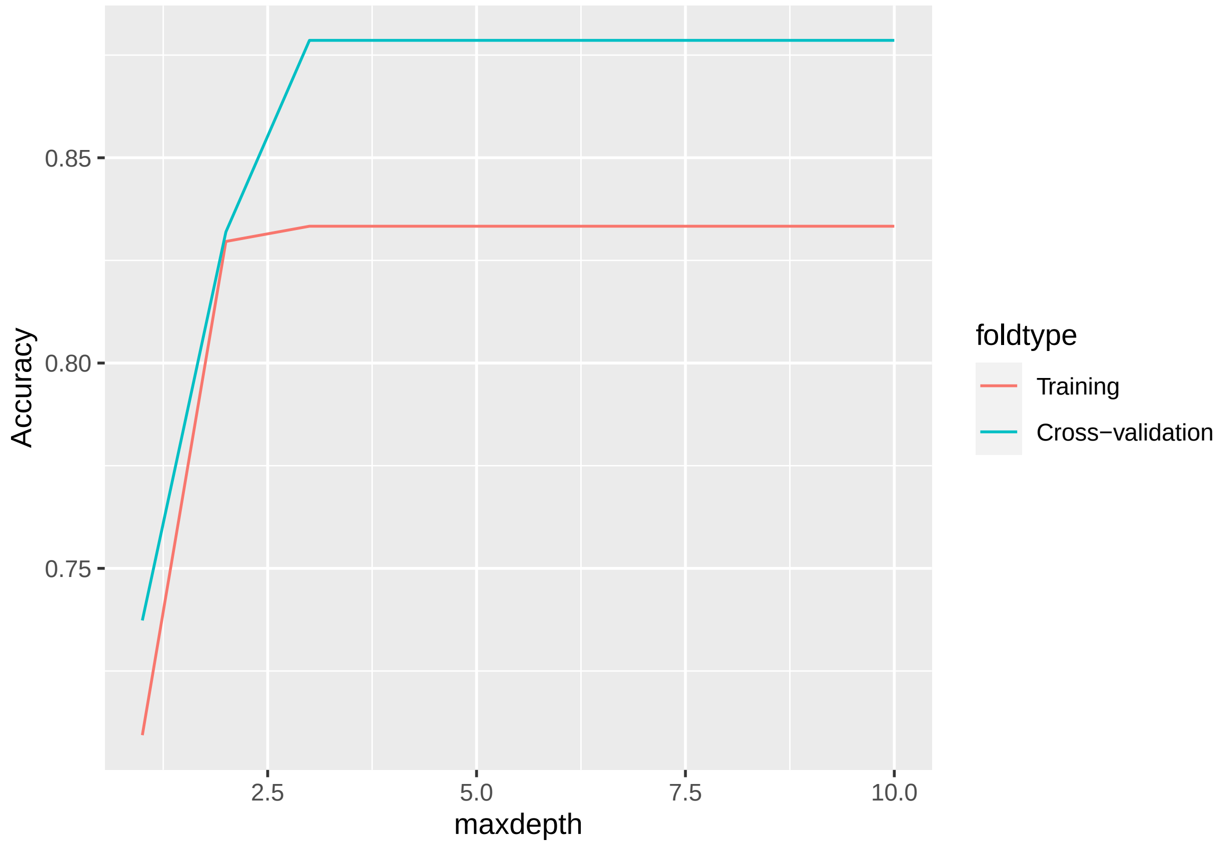
Hyperparameter tuning

- Decision Tree

```
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

```
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()
```

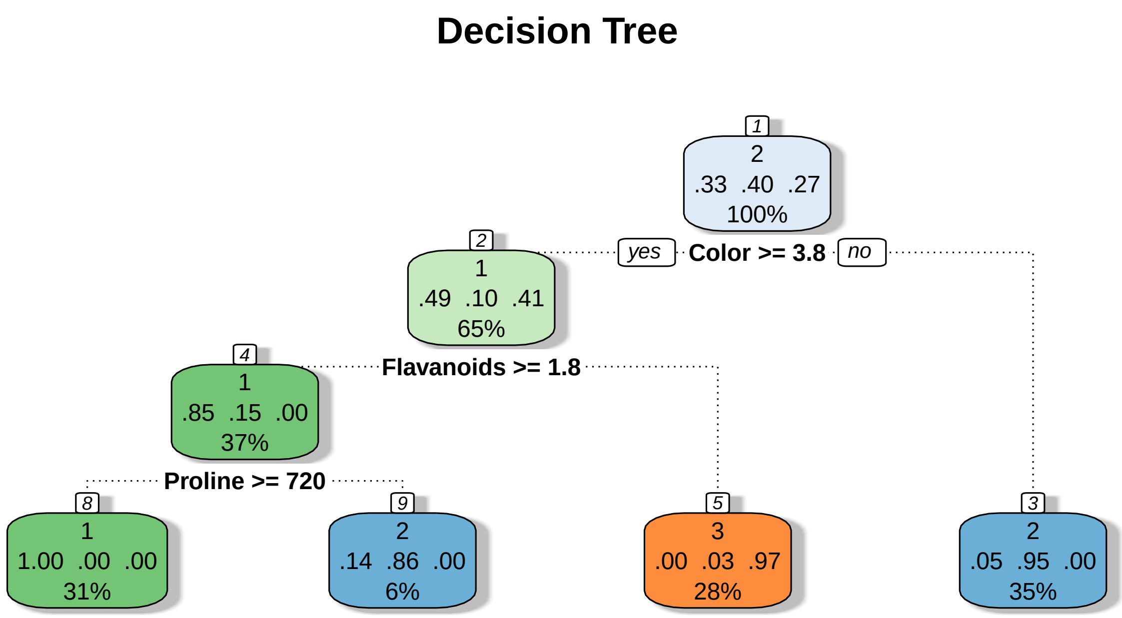


- See and plot the final tree

```
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 70 36 1 (0.48571429 0.10000000 0.41428571)
##     4) Flavanoids>=1.8 40  6 1 (0.85000000 0.15000000 0.00000000)
##       8) Proline>=720 33  0 1 (1.00000000 0.00000000 0.00000000) *
##       9) Proline< 720 7  1 2 (0.14285714 0.85714286 0.00000000) *
##     5) Flavanoids< 1.8 30  1 3 (0.00000000 0.03333333 0.96666667) *
##   3) Color< 3.82 38  2 2 (0.05263158 0.94736842 0.00000000) *
```

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



Estimating the Model Performance

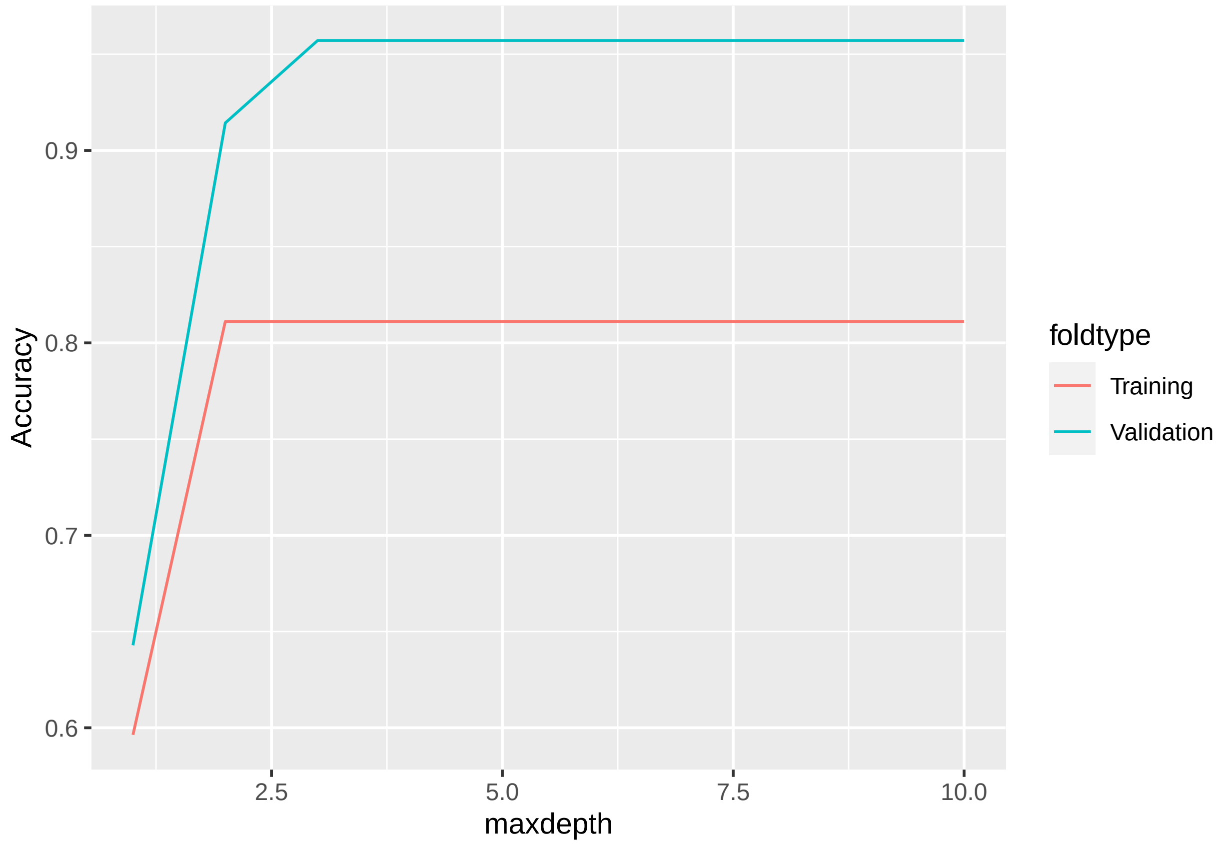
- For 2 folds

```
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)
```

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



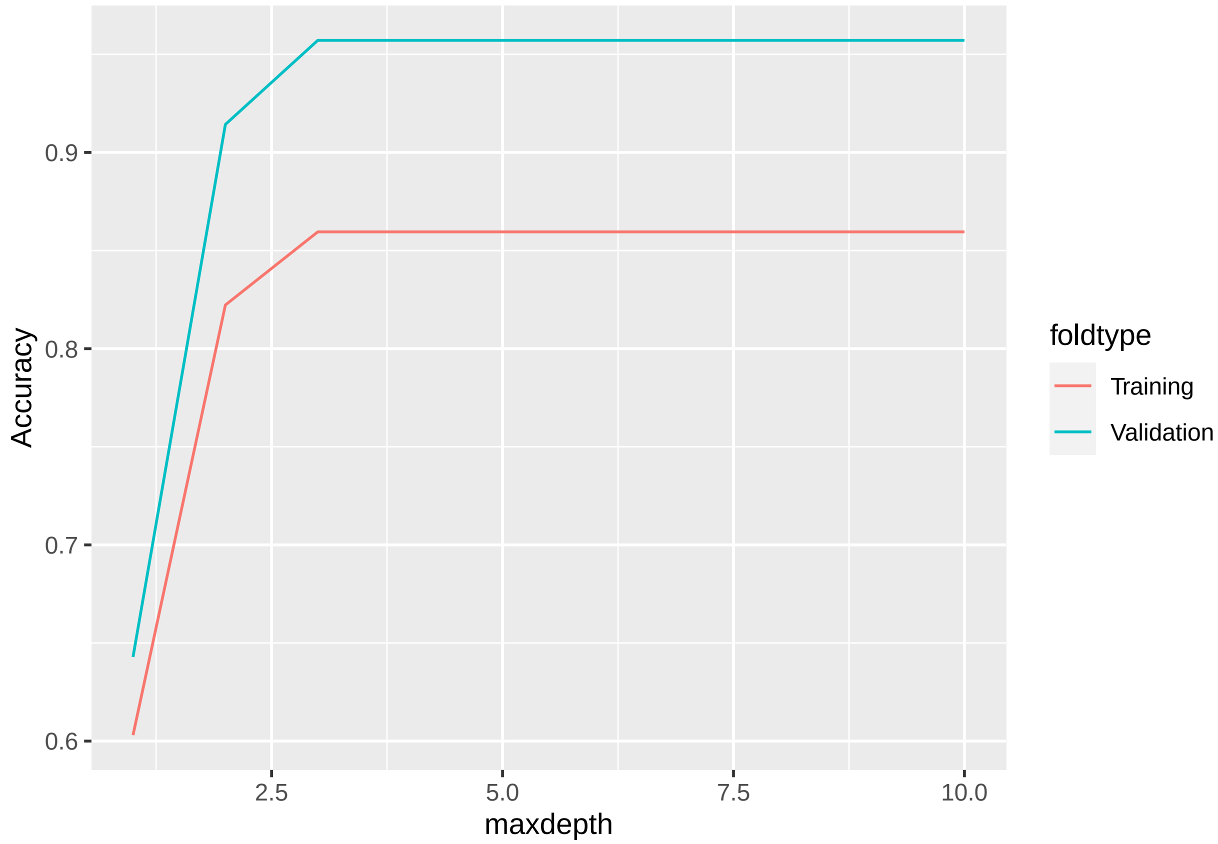
- For 10 folds

```r
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

```r
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 =  0.957142857142857"
```

- 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 =  0.866805555555555"
```

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

```
## [1] "CV Accuracy - sd =  0.106194808806192"
```