Week 6

R Packages

caret, RandomForest, mlbench, adabag, ROCR, and doSNOW.

Model Specification & Selection: Questions to Ask

1. What's my intended goal/outcome?

2. What data do I have?

3. Which predictors are significant?
   4. What's the expected performance of a model?
   5. How do I improve a model's performance?

We will focus on Questions 4 & 5 tonight.

Assessing Model Performance

The best way to measure performance is to know the **true error rate**. The true error rate is calculated by comparing the model's predictions against actual outcomes in the **entire population**. In reality, we usually are not working with the whole population. We are working with one or more samples from the population.

Naive Approach

A *naive* way to estimate the true error rate is to apply our model to the entire sample (i.e. training dataset) and then calculate the error rate. The naive approach has several drawbacks:

- Final model will overfit the training data. The problem is magnified when a model has a large number of parameters.

- Estimated error rate is likely to be lower than the true error rate.

A better approach than the naive method is resampling.

Resampling

Resampling refers to drawing repeated samples from the sample(s) we have. The goal of resampling is to gauge model performance. We will discuss four resampling methods.

Validation Set

We discussed validation set approach in Week 2 when we covered decision trees. In particular, the validation set approach involves randomly dividing the available observations into two subgroups: a) a training set and b) a validation (or hold out) set. We fit our model with the training set and then tests the model's performance on the validation set. Common splits include 60-40 (60% training set and 40% test set), 70-30, and 80-20.

Opioid Prescriber dataset redux!

```
setwd("C:/Users/corylowe/OneDrive/Code/R Practice Code/Applied Data Mining_Portfolio/Week 6")
prescribers<-read.csv("prescribers.csv")

#View(prescribers)

prescribers<-prescribers[,c(241,1:240,242:331)] #Rearranging the columns so that our target variable is

dim(prescribers)
```

```
## [1] 25000   331
```

```
names(prescribers)
```

```
##   [1] "Opioid.Prescriber"
##   [2] "Gender"
##   [3] "ABILIFY"
##   [4] "ACYCLOVIR"
##   [5] "ADVAIR.DISKUS"
##   [6] "AGGRENOX"
##   [7] "ALENDRONATE.SODIUM"
##   [8] "ALLOPURINOL"
##   [9] "ALPRAZOLAM"
##  [10] "AMIODARONE.HCL"
##  [11] "AMITRIPTYLINE.HCL"
##  [12] "AMLODIPINE.BESYLATE"
##  [13] "AMLODIPINE.BESYLATE.BENAZEPRIL"
##  [14] "AMOXICILLIN"
##  [15] "AMOX.TR.POTASSIUM.CLAVULANATE"
##  [16] "AMPHETAMINE.SALT.COMBO"
##  [17] "ATENOLOL"
##  [18] "ATORVASTATIN.CALCIUM"
##  [19] "AVODART"
##  [20] "AZITHROMYCIN"
##  [21] "BACLOFEN"
##  [22] "BD.ULTRA.FINE.PEN.NEEDLE"
##  [23] "BENAZEPRIL.HCL"
##  [24] "BENICAR"
##  [25] "BENICAR.HCT"
##  [26] "BENZTROPINE.MESYLATE"
##  [27] "BISOPROLOL.HYDROCHLOROTHIAZIDE"
##  [28] "BRIMONIDINE.TARTRATE"
##  [29] "BUMETANIDE"
##  [30] "BUPROPION.HCL.SR"
```

```
##  [31] "BUPROPION.XL"
##  [32] "BUSPIRONE.HCL"
##  [33] "BYSTOLIC"
##  [34] "CARBAMAZEPINE"
##  [35] "CARBIDOPA.LEVODOPA"
##  [36] "CARISOPRODOL"
##  [37] "CARTIA.XT"
##  [38] "CARVEDILOL"
##  [39] "CEFUROXIME"
##  [40] "CELEBREX"
##  [41] "CEPHALEXIN"
##  [42] "CHLORHEXIDINE.GLUCONATE"
##  [43] "CHLORTHALIDONE"
##  [44] "CILOSTAZOL"
##  [45] "CIPROFLOXACIN.HCL"
##  [46] "CITALOPRAM.HBR"
##  [47] "CLINDAMYCIN.HCL"
##  [48] "CLOBETASOL.PROPIONATE"
##  [49] "CLONAZEPAM"
##  [50] "CLONIDINE.HCL"
##  [51] "CLOPIDOGREL"
##  [52] "CLOTRIMAZOLE.BETAMETHASONE"
##  [53] "COLCRYS"
##  [54] "COMBIVENT.RESPIMAT"
##  [55] "CRESTOR"
##  [56] "CYCLOBENZAPRINE.HCL"
##  [57] "DEXILANT"
##  [58] "DIAZEPAM"
##  [59] "DICLOFENAC.SODIUM"
##  [60] "DICYCLOMINE.HCL"
##  [61] "DIGOX"
##  [62] "DIGOXIN"
##  [63] "DILTIAZEM.24HR.CD"
##  [64] "DILTIAZEM.24HR.ER"
##  [65] "DILTIAZEM.ER"
##  [66] "DILTIAZEM.HCL"
##  [67] "DIOVAN"
##  [68] "DIPHENOXYLATE.ATROPINE"
##  [69] "DIVALPROEX.SODIUM"
##  [70] "DIVALPROEX.SODIUM.ER"
##  [71] "DONEPEZIL.HCL"
##  [72] "DORZOLAMIDE.TIMOLOL"
##  [73] "DOXAZOSIN.MESYLATE"
##  [74] "DOXEPIN.HCL"
##  [75] "DOXYCYCLINE.HYCLATE"
##  [76] "DULOXETINE.HCL"
##  [77] "ENALAPRIL.MALEATE"
##  [78] "ESCITALOPRAM.OXALATE"
##  [79] "ESTRADIOL"
##  [80] "EXELON"
##  [81] "FAMOTIDINE"
##  [82] "FELODIPINE.ER"
##  [83] "FENOFIBRATE"
##  [84] "FINASTERIDE"
```

```
##  [85] "FLOVENT.HFA"
##  [86] "FLUCONAZOLE"
##  [87] "FLUOXETINE.HCL"
##  [88] "FLUTICASONE.PROPIONATE"
##  [89] "FUROSEMIDE"
##  [90] "GABAPENTIN"
##  [91] "GEMFIBROZIL"
##  [92] "GLIMEPIRIDE"
##  [93] "GLIPIZIDE"
##  [94] "GLIPIZIDE.ER"
##  [95] "GLIPIZIDE.XL"
##  [96] "GLYBURIDE"
##  [97] "HALOPERIDOL"
##  [98] "HUMALOG"
##  [99] "HYDRALAZINE.HCL"
## [100] "HYDROCHLOROTHIAZIDE"
## [101] "HYDROCORTISONE"
## [102] "HYDROXYZINE.HCL"
## [103] "IBANDRONATE.SODIUM"
## [104] "IBUPROFEN"
## [105] "INSULIN.SYRINGE"
## [106] "IPRATROPIUM.BROMIDE"
## [107] "IRBESARTAN"
## [108] "ISOSORBIDE.MONONITRATE.ER"
## [109] "JANTOVEN"
## [110] "JANUMET"
## [111] "JANUVIA"
## [112] "KETOCONAZOLE"
## [113] "KLOR.CON.10"
## [114] "KLOR.CON.M10"
## [115] "KLOR.CON.M20"
## [116] "LABETALOL.HCL"
## [117] "LACTULOSE"
## [118] "LAMOTRIGINE"
## [119] "LANSOPRAZOLE"
## [120] "LANTUS"
## [121] "LANTUS.SOLOSTAR"
## [122] "LATANOPROST"
## [123] "LEVEMIR"
## [124] "LEVEMIR.FLEXPEN"
## [125] "LEVETIRACETAM"
## [126] "LEVOFLOXACIN"
## [127] "LEVOTHYROXINE.SODIUM"
## [128] "LIDOCAINE"
## [129] "LISINOPRIL"
## [130] "LISINOPRIL.HYDROCHLOROTHIAZIDE"
## [131] "LITHIUM.CARBONATE"
## [132] "LORAZEPAM"
## [133] "LOSARTAN.HYDROCHLOROTHIAZIDE"
## [134] "LOSARTAN.POTASSIUM"
## [135] "LOVASTATIN"
## [136] "LOVAZA"
## [137] "LUMIGAN"
## [138] "LYRICA"
```

```
## [139] "MECLIZINE.HCL"
## [140] "MELOXICAM"
## [141] "METFORMIN.HCL"
## [142] "METFORMIN.HCL.ER"
## [143] "METHOCARBAMOL"
## [144] "METHOTREXATE"
## [145] "METHYLPREDNISOLONE"
## [146] "METOCLOPRAMIDE.HCL"
## [147] "METOLAZONE"
## [148] "METOPROLOL.SUCCINATE"
## [149] "METOPROLOL.TARTRATE"
## [150] "METRONIDAZOLE"
## [151] "MIRTAZAPINE"
## [152] "MONTELUKAST.SODIUM"
## [153] "MUPIROCIN"
## [154] "NABUMETONE"
## [155] "NAMENDA"
## [156] "NAMENDA.XR"
## [157] "NAPROXEN"
## [158] "NASONEX"
## [159] "NEXIUM"
## [160] "NIACIN.ER"
## [161] "NIFEDICAL.XL"
## [162] "NIFEDIPINE.ER"
## [163] "NITROFURANTOIN.MONO.MACRO"
## [164] "NITROSTAT"
## [165] "NORTRIPTYLINE.HCL"
## [166] "NOVOLOG"
## [167] "NOVOLOG.FLEXPEN"
## [168] "NYSTATIN"
## [169] "OLANZAPINE"
## [170] "OMEPRAZOLE"
## [171] "ONDANSETRON.HCL"
## [172] "ONDANSETRON.ODT"
## [173] "ONGLYZA"
## [174] "OXCARBAZEPINE"
## [175] "OXYBUTYNIN.CHLORIDE"
## [176] "OXYBUTYNIN.CHLORIDE.ER"
## [177] "PANTOPRAZOLE.SODIUM"
## [178] "PAROXETINE.HCL"
## [179] "PHENOBARBITAL"
## [180] "PHENYTOIN.SODIUM.EXTENDED"
## [181] "PIOGLITAZONE.HCL"
## [182] "POLYETHYLENE.GLYCOL.3350"
## [183] "POTASSIUM.CHLORIDE"
## [184] "PRADAXA"
## [185] "PRAMIPEXOLE.DIHYDROCHLORIDE"
## [186] "PRAVASTATIN.SODIUM"
## [187] "PREDNISONE"
## [188] "PREMARIN"
## [189] "PRIMIDONE"
## [190] "PROAIR.HFA"
## [191] "PROMETHAZINE.HCL"
## [192] "PROPRANOLOL.HCL"
```

```
## [193] "PROPRANOLOL.HCL.ER"
## [194] "QUETIAPINE.FUMARATE"
## [195] "QUINAPRIL.HCL"
## [196] "RALOXIFENE.HCL"
## [197] "RAMIPRIL"
## [198] "RANEXA"
## [199] "RANITIDINE.HCL"
## [200] "RESTASIS"
## [201] "RISPERIDONE"
## [202] "ROPINIROLE.HCL"
## [203] "SEROQUEL.XR"
## [204] "SERTRALINE.HCL"
## [205] "SIMVASTATIN"
## [206] "SOTALOL"
## [207] "SPIRIVA"
## [208] "SPIRONOLACTONE"
## [209] "SUCRALFATE"
## [210] "SULFAMETHOXAZOLE.TRIMETHOPRIM"
## [211] "SUMATRIPTAN.SUCCINATE"
## [212] "SYMBICORT"
## [213] "SYNTHROID"
## [214] "TAMSULOSIN.HCL"
## [215] "TEMAZEPAM"
## [216] "TERAZOSIN.HCL"
## [217] "TIMOLOL.MALEATE"
## [218] "TIZANIDINE.HCL"
## [219] "TOLTERODINE.TARTRATE.ER"
## [220] "TOPIRAMATE"
## [221] "TOPROL.XL"
## [222] "TORSEMIDE"
## [223] "TRAVATAN.Z"
## [224] "TRAZODONE.HCL"
## [225] "TRIAMCINOLONE.ACETONIDE"
## [226] "TRIAMTERENE.HYDROCHLOROTHIAZID"
## [227] "VALACYCLOVIR"
## [228] "VALSARTAN"
## [229] "VALSARTAN.HYDROCHLOROTHIAZIDE"
## [230] "VENLAFAXINE.HCL"
## [231] "VENLAFAXINE.HCL.ER"
## [232] "VENTOLIN.HFA"
## [233] "VERAPAMIL.ER"
## [234] "VESICARE"
## [235] "VOLTAREN"
## [236] "VYTORIN"
## [237] "WARFARIN.SODIUM"
## [238] "XARELTO"
## [239] "ZETIA"
## [240] "ZIPRASIDONE.HCL"
## [241] "ZOLPIDEM.TARTRATE"
## [242] "AL"
## [243] "AR"
## [244] "AZ"
## [245] "CA"
## [246] "CO"
```

```
## [247] "CT"
## [248] "DC"
## [249] "DE"
## [250] "FL"
## [251] "GA"
## [252] "HI"
## [253] "IA"
## [254] "ID"
## [255] "IL"
## [256] "IN"
## [257] "KS"
## [258] "KY"
## [259] "LA"
## [260] "MA"
## [261] "MD"
## [262] "ME"
## [263] "MI"
## [264] "MN"
## [265] "MO"
## [266] "MS"
## [267] "MT"
## [268] "NC"
## [269] "ND"
## [270] "NE"
## [271] "NH"
## [272] "NJ"
## [273] "NM"
## [274] "NV"
## [275] "NY"
## [276] "OH"
## [277] "OK"
## [278] "OR"
## [279] "PA"
## [280] "PR"
## [281] "RI"
## [282] "SC"
## [283] "SD"
## [284] "TN"
## [285] "TX"
## [286] "UT"
## [287] "VA"
## [288] "VT"
## [289] "WA"
## [290] "WI"
## [291] "WV"
## [292] "WY"
## [293] "other"
## [294] "Anesthesiology"
## [295] "Cardiology"
## [296] "Certified.Clinical.Nurse.Specialist"
## [297] "Dentist"
## [298] "Dermatology"
## [299] "Emergency.Medicine"
## [300] "Endocrinology"
```

```
## [301] "Family.Practice"
## [302] "Gastroenterology"
## [303] "General.Practice"
## [304] "Geriatric.Medicine"
## [305] "Hematology.Oncology"
## [306] "Infectious.Disease"
## [307] "Internal.Medicine"
## [308] "Medical.Oncology"
## [309] "Nephrology"
## [310] "Neurology"
## [311] "Neuropsychiatry"
## [312] "Nurse.Practitioner"
## [313] "Obstetrics.Gynecology"
## [314] "Ophthalmology"
## [315] "Optometry"
## [316] "Otolaryngology"
## [317] "Pediatric.Medicine"
## [318] "Physical.Medicine.and.Rehabilitation"
## [319] "Physician.Assistant"
## [320] "Podiatry"
## [321] "Psychiatry"
## [322] "Psychiatry...Neurology"
## [323] "Pulmonary.Disease"
## [324] "Radiation.Oncology"
## [325] "Rheumatology"
## [326] "Specialist"
## [327] "Student.in.an.Organized.Health.Care.Education.Training.Program"
## [328] "Urology"
## [329] "Surgeon"
## [330] "other.1"
## [331] "Pain.Management"
```

```
table(prescribers$Opioid.Prescriber)
```

```
##
##    no   yes
## 10312 14688
```

Let's do a training set of 80% and validation set of 20%.

```
set.seed(123) #set a seed to do draws from a random uniform distribution.
prescribers_rand <- prescribers[order(runif(25000)), ]
prescribers_train <- prescribers_rand[1:20000, ] #Training data set; 20000 observations
prescribers_test  <-prescribers_rand[20001:25000, ]
```

Checking the proportions of opioid prescriber in the test and train sets. They are roughly the same. 58.8% in train set are opioid prescribers; 58.6% in test set are opioid prescribers.

```
dim(prescribers_train) #checking the split
```

```
## [1] 20000   331
```

```r
dim(prescribers_test) #checking the split
```

```
## [1] 5000  331
```

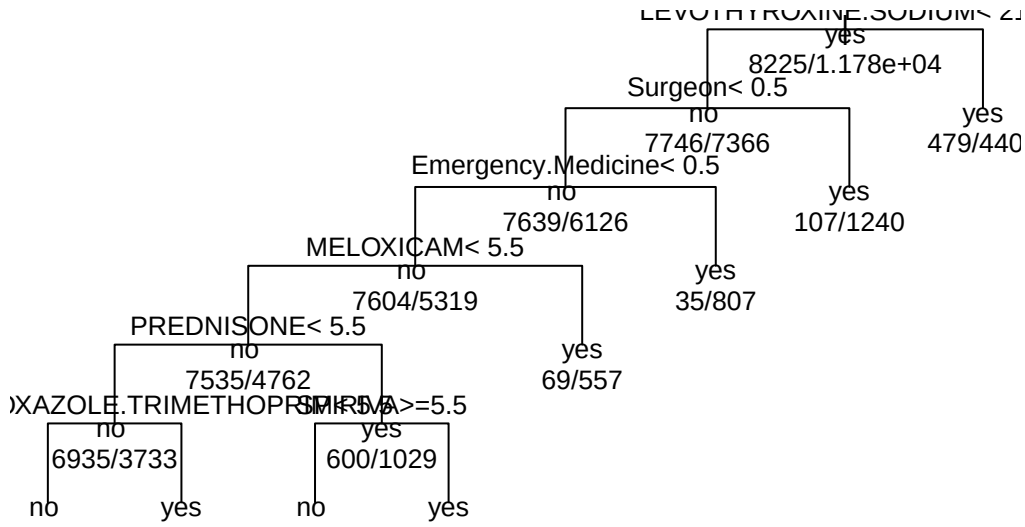
```r
prop.table(table(prescribers_train$Opioid.Prescriber)) #checking to see the class proportions between t
```

```
##
##      no     yes
## 0.41125 0.58875
```

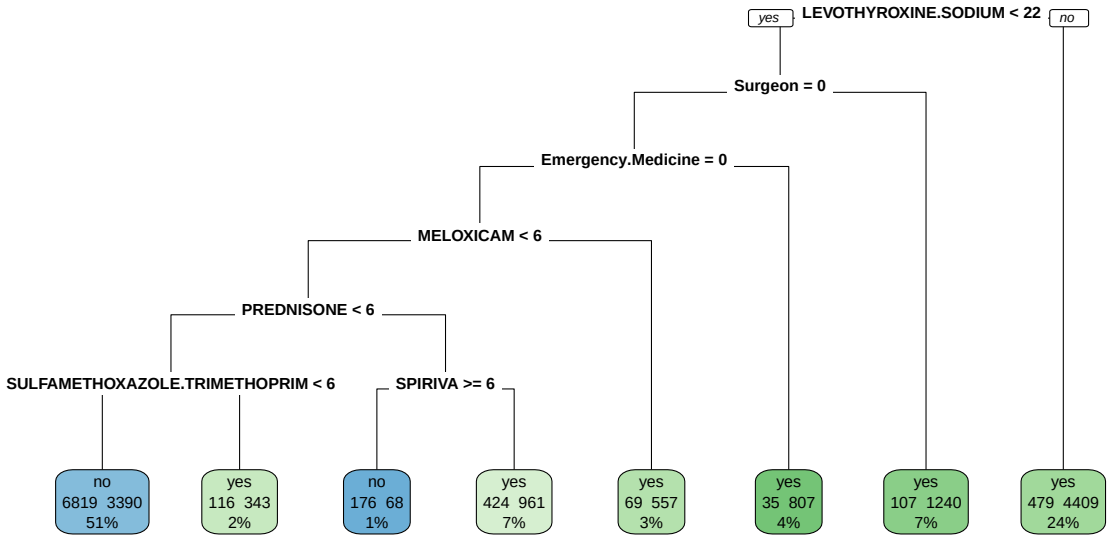
```r
prop.table(table(prescribers_test$Opioid.Prescriber))
```

```
##
##     no    yes
## 0.4174 0.5826
```

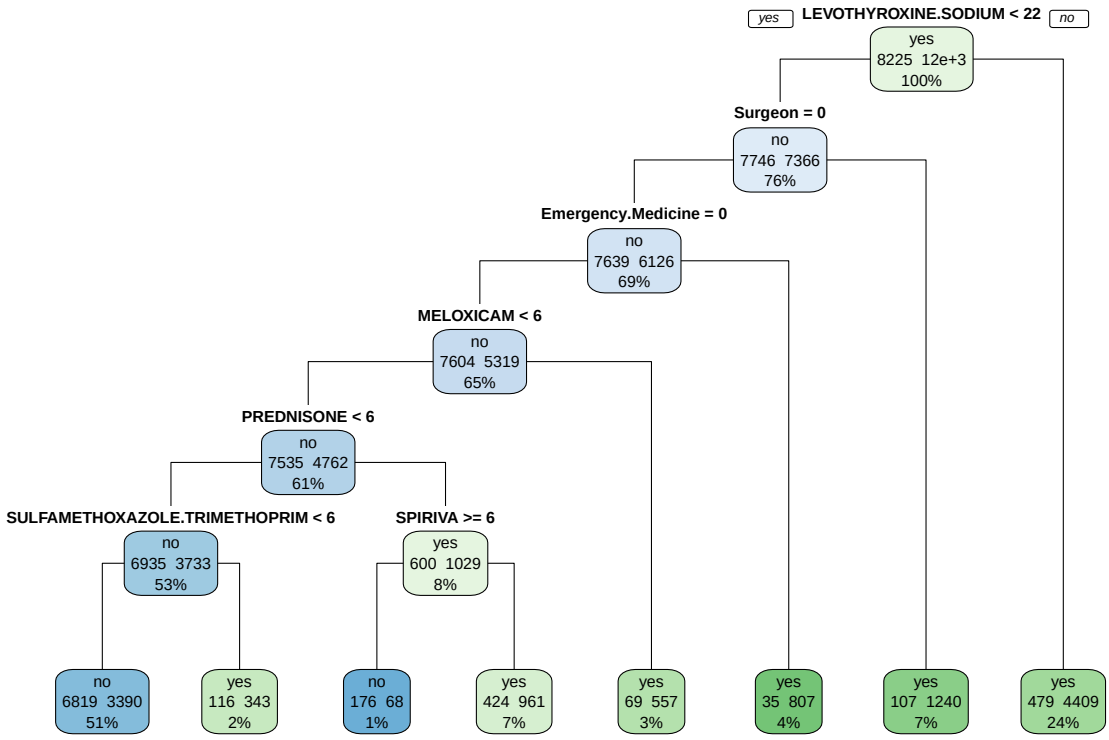
```r
library(rpart)
library(rpart.plot)

prescribers_rpart <- rpart(prescribers_train$Opioid.Prescriber~., method="class", parms = list(split="g

plot(prescribers_rpart, uniform=TRUE, main="Classification Tree for Opioid Prescribers")
text(prescribers_rpart, use.n=TRUE, all=TRUE, cex=0.8)
```



```
# Something a bit fancier
library(rpart.plot)
rpart.plot(prescribers_rpart, type=0, extra=101)
```




```
rpart.plot(prescribers_rpart, type=1, extra=101)
```




```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
actual <- prescribers_test$Opioid.Prescriber
predicted <- predict(prescribers_rpart, prescribers_test, type="class")
results.matrix <- confusionMatrix(predicted, actual, positive="yes")
print(results.matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  1761  877
##        yes  326 2036
##
##                Accuracy : 0.7594
##                  95% CI : (0.7473, 0.7712)
##     No Information Rate : 0.5826
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5231
```

```
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.6989
##             Specificity : 0.8438
##          Pos Pred Value : 0.8620
##          Neg Pred Value : 0.6676
##              Prevalence : 0.5826
##          Detection Rate : 0.4072
##    Detection Prevalence : 0.4724
##       Balanced Accuracy : 0.7714
##
##        'Positive' Class : yes
##
```

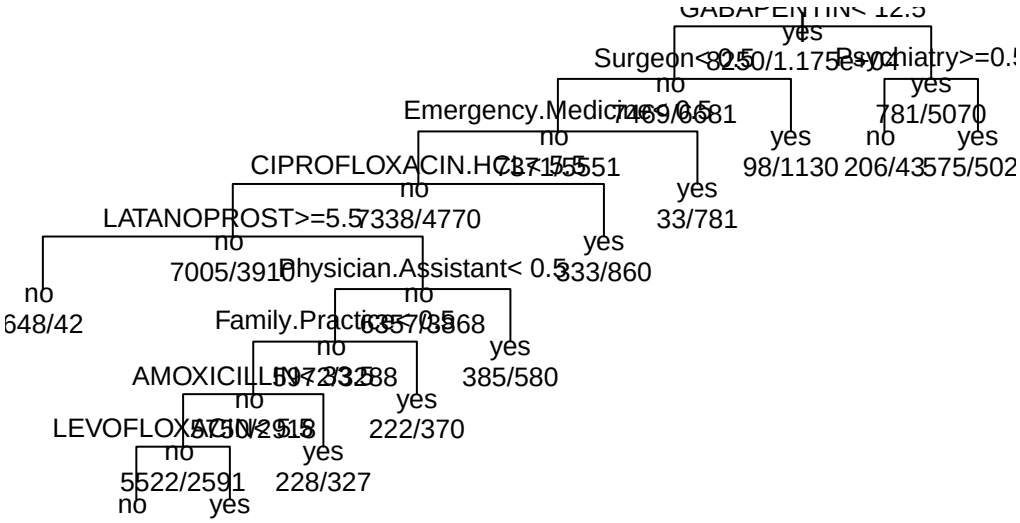
Sensivity = 70%; Specificity = 84%

Our decision tree does a better job of classifying the non-opioid prescribers than the opioid prescribers.

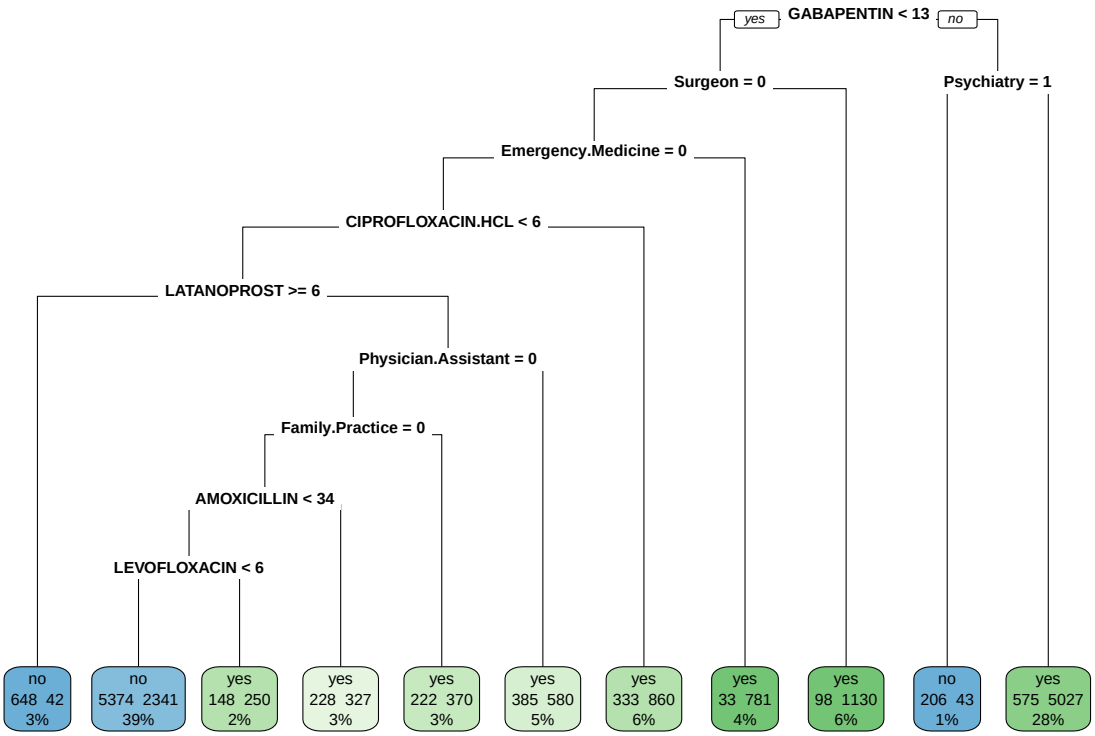
Using the Caret Package to do Train & Test Set Splits

```
set.seed(123)
trainIndex <- createDataPartition(prescribers$Opioid.Prescriber, p = .8,list = FALSE,times = 1)
prescribers_train_caret <- prescribers[ trainIndex,]
prescribers_test_caret <- prescribers[ -trainIndex,]
```

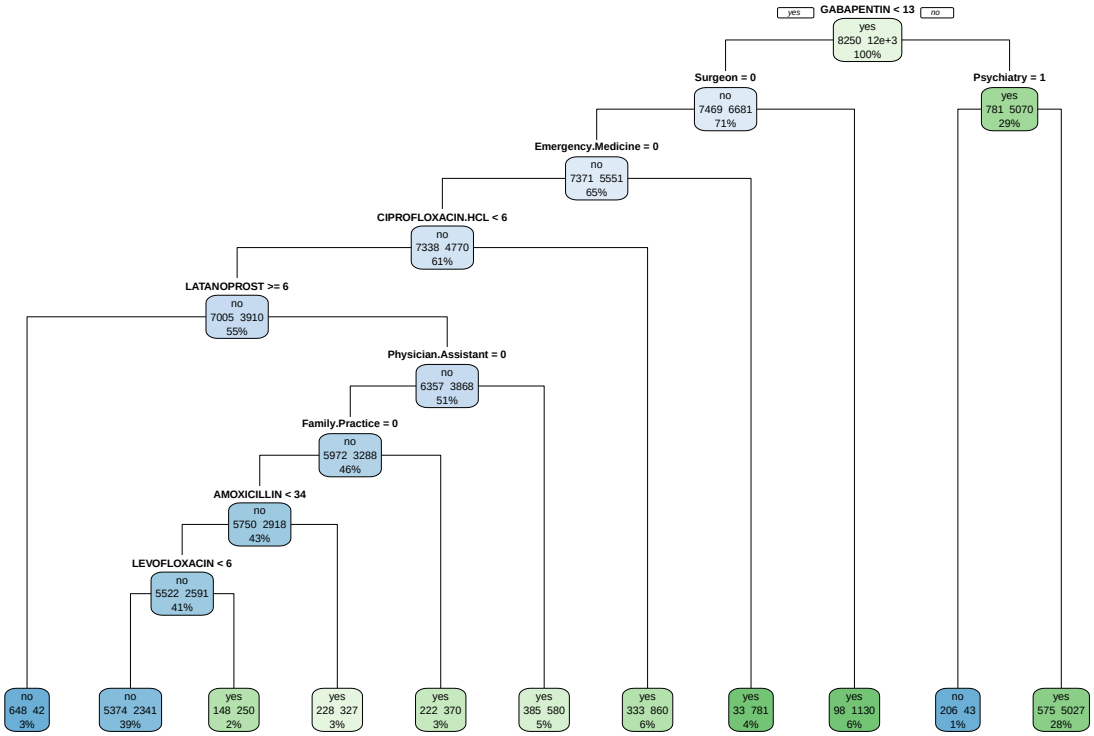
```
prescribers_rpart_caret <- rpart(Opioid.Prescriber~., method="class", parms = list(split="gini"), data=p

plot(prescribers_rpart_caret, uniform=TRUE, main="Classification Tree for Opioid Prescribers")
text(prescribers_rpart_caret, use.n=TRUE, all=TRUE, cex=0.8)
```




```
rpart.plot(prescribers_rpart_caret, type=0, extra=101)
```




```
rpart.plot(prescribers_rpart_caret, type=1, extra=101)
```




```
actual <- prescribers_test_caret$Opioid.Prescriber
predicted <- predict(prescribers_rpart_caret, prescribers_test_caret, type="class")
results.matrix <- confusionMatrix(predicted, actual, positive="yes")
print(results.matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  1528  565
##        yes  534 2372
##
##                Accuracy : 0.7802
##                  95% CI : (0.7684, 0.7916)
##     No Information Rate : 0.5875
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.5474
##  Mcnemar's Test P-Value : 0.3655
##
##             Sensitivity : 0.8076
##             Specificity : 0.7410
##          Pos Pred Value : 0.8162
##          Neg Pred Value : 0.7301
##              Prevalence : 0.5875
```

```
##          Detection Rate : 0.4745
##    Detection Prevalence : 0.5813
##       Balanced Accuracy : 0.7743
##
##        'Positive' Class : yes
##
```

Sensitivity = 79%; Specificity = 75%. The caret approach creates a test set that shows the decision tree is doing slightly better at identifying opioid prescribers. Is there a more stable approach?

k-Fold Cross Validation

k-fold cross validation is a resampling technique that divides the dataset into k groups, or folds, of equal size. Here is how it works:

1. Keep one fold as the validation (hold out) set. Fit the model on the other k-1 folds.
2. Test fitted model on the held out fold. Calculate the mean squared error (MSE) of the held out fold.
3. Repeat Steps 1 & 2 over and over aain so that a different fold is used as a validation set. **The true error rate is estimated as the average error rate of all repetitions.**

Use the **caret** package for this task.

We will divide the data set into 10-folds.

```
fitControl <- trainControl(method="cv", number=10) #10-fold cross validation

set.seed(123)
prescribers_10folds<-train(Opioid.Prescriber~., data=prescribers_train_caret, method="rpart", metric="A
prescribers_10folds
```

```
## CART
##
## 20001 samples
##   330 predictor
##     2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 18001, 18001, 18001, 18001, 18001, 18001, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.06387879  0.7270650  0.4687792
##   0.09066667  0.6993650  0.4238370
##   0.11030303  0.6196168  0.1477631
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.06387879.
```

Now we apply the decision tree on the test set.

```
actual <- prescribers_test_caret$Opioid.Prescriber
predicted <- predict(prescribers_10folds, prescribers_test_caret, type="raw")
results.matrix <- confusionMatrix(predicted, actual, positive="yes")
print(results.matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  1829 1157
##        yes  233 1780
##
##                Accuracy : 0.7219
##                  95% CI : (0.7093, 0.7343)
##     No Information Rate : 0.5875
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4622
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.6061
##             Specificity : 0.8870
##          Pos Pred Value : 0.8843
##          Neg Pred Value : 0.6125
##              Prevalence : 0.5875
##          Detection Rate : 0.3561
##    Detection Prevalence : 0.4027
##       Balanced Accuracy : 0.7465
##
##        'Positive' Class : yes
##
```

Sensitivity = 61%; Specificity = 89%

Kappa?

`Kappa = Pr(a) - Pr(e) / 1 - Pr(e)`

Where,

Pr(a): proportion of actual agreement between the classifier and the true values

Pr(e): proportion of expected agreement between the classifier and the true values

Kappa "adjusts accuracy by accounting for the possibility of a correct prediction by chance alone. Kappa values range to a maximum number of 1, which indicates perfect agreement between the model's predictions and the true values–a rare occurrence. Values less than one indicate imperfect agreement" (Lantz 2013, p. 303)

Repeated k-fold Cross Validation

```r
fitControl <- trainControl(method="cv", number=10, repeats=5) #10-fold cross validation
```

```
## Warning: `repeats` has no meaning for this resampling method.
```

```r
set.seed(123)
prescribers_10folds_rp<-train(Opioid.Prescriber~., data=prescribers_train_caret, method="rpart", metric=
prescribers_10folds_rp
```

```
## CART
##
## 20001 samples
##   330 predictor
##     2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 18001, 18001, 18001, 18001, 18001, 18001, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.06387879  0.7270650  0.4687792
##   0.09066667  0.6993650  0.4238370
##   0.11030303  0.6196168  0.1477631
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.06387879.
```

```r
actual <- prescribers_test_caret$Opioid.Prescriber
predicted <- predict(prescribers_10folds_rp, prescribers_test_caret, type="raw")
results.matrix <- confusionMatrix(predicted, actual, positive="yes")
print(results.matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  1829 1157
##        yes  233 1780
##
##                Accuracy : 0.7219
##                  95% CI : (0.7093, 0.7343)
##     No Information Rate : 0.5875
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4622
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.6061
##             Specificity : 0.8870
##          Pos Pred Value : 0.8843
##          Neg Pred Value : 0.6125
```

```
##              Prevalence : 0.5875
##          Detection Rate : 0.3561
##    Detection Prevalence : 0.4027
##       Balanced Accuracy : 0.7465
##
##        'Positive' Class : yes
##
```

Sensitivity = 61%; Specificity = 89%

Leave-one-out Cross Validation (LOOCV)

Leave one out is a degenerate case of k-fold cross validation, where K is chosen as the total number of observations. LOOCV uses all observations as the training set and leaves one observation out as the test set. The process repeats until all observations have been used as a test set.

Bootstrapping

Bootstrapping is a resampling technique that obtain distinct datasets by repeatedly sampling observations from the original dataset with replacement.

Each boostrapped dataset is created by sampling with replacement and is the same size as the original dataset. Consequently, some observations may appear more than once in a given boostrapped dataset while other observations may not appear at all.

Note: The default method in the train() function in the caret package is the bootstrap.

```
cvCtrl <- trainControl(method="boot", number=10) #10 resampling iterations
set.seed(123)
prescribers_bootstrap<-train(Opioid.Prescriber~., data=prescribers_train_caret, method="rpart", metric=
prescribers_bootstrap
```

```
## CART
##
## 20001 samples
##   330 predictor
##     2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Bootstrapped (10 reps)
## Summary of sample sizes: 20001, 20001, 20001, 20001, 20001, 20001, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.06387879  0.7241786  0.4659578
##   0.09066667  0.6751377  0.3382570
##   0.11030303  0.6247089  0.1573385
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.06387879.
```

```r
actual <- prescribers_test_caret$Opioid.Prescriber
predicted <- predict(prescribers_bootstrap, prescribers_test_caret, type="raw")
results.matrix <- confusionMatrix(predicted, actual, positive="yes")
print(results.matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   no  yes
##        no  1829 1157
##        yes  233 1780
##
##                Accuracy : 0.7219
##                  95% CI : (0.7093, 0.7343)
##     No Information Rate : 0.5875
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4622
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.6061
##             Specificity : 0.8870
##          Pos Pred Value : 0.8843
##          Neg Pred Value : 0.6125
##              Prevalence : 0.5875
##          Detection Rate : 0.3561
##    Detection Prevalence : 0.4027
##       Balanced Accuracy : 0.7465
##
##        'Positive' Class : yes
##
```

Sensitivity = 61%; Specificity = 89%

Last Words on Resampling

A question you may be pondering about is "how many folds should I use?" The answer depends on the size of the dataset. For large datasets, you can use a small number of folds and still get an accurate test error estimate. For smaller datasets, you may have to use LOOCV. You should remember these rules:

Small number of folds = variance of the test error estimate is smaller; test error estimate is more biased; computation time is less.

Large number of folds = variance of the test error estimate is larger; test error estimate is less biased; computation time is greater.

A Second Dataset

For the second part of this class, we will use the Breast Cancer data set from the mlbench package. It's a smaller data set so easier for us to handle in class.

```
library(mlbench)
data("BreastCancer")
str(BreastCancer)
```

```
## 'data.frame':    699 obs. of  11 variables:
##  $ Id             : chr  "1000025" "1002945" "1015425" "1016277" ...
##  $ Cl.thickness   : Ord.factor w/ 10 levels "1"<"2"<"3"<"4"<..: 5 5 3 6 4 8 1 2 2 4 ...
##  $ Cell.size      : Ord.factor w/ 10 levels "1"<"2"<"3"<"4"<..: 1 4 1 8 1 10 1 1 1 2 ...
##  $ Cell.shape     : Ord.factor w/ 10 levels "1"<"2"<"3"<"4"<..: 1 4 1 8 1 10 1 2 1 1 ...
##  $ Marg.adhesion  : Ord.factor w/ 10 levels "1"<"2"<"3"<"4"<..: 1 5 1 1 3 8 1 1 1 1 ...
##  $ Epith.c.size   : Ord.factor w/ 10 levels "1"<"2"<"3"<"4"<..: 2 7 2 3 2 7 2 2 2 2 ...
##  $ Bare.nuclei    : Factor w/ 10 levels "1","2","3","4",..: 1 10 2 4 1 10 10 1 1 1 ...
##  $ Bl.cromatin    : Factor w/ 10 levels "1","2","3","4",..: 3 3 3 3 3 9 3 3 1 2 ...
##  $ Normal.nucleoli: Factor w/ 10 levels "1","2","3","4",..: 1 2 1 7 1 7 1 1 1 1 ...
##  $ Mitoses        : Factor w/ 9 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 5 1 ...
##  $ Class          : Factor w/ 2 levels "benign","malignant": 1 1 1 1 1 2 1 1 1 1 ...
```

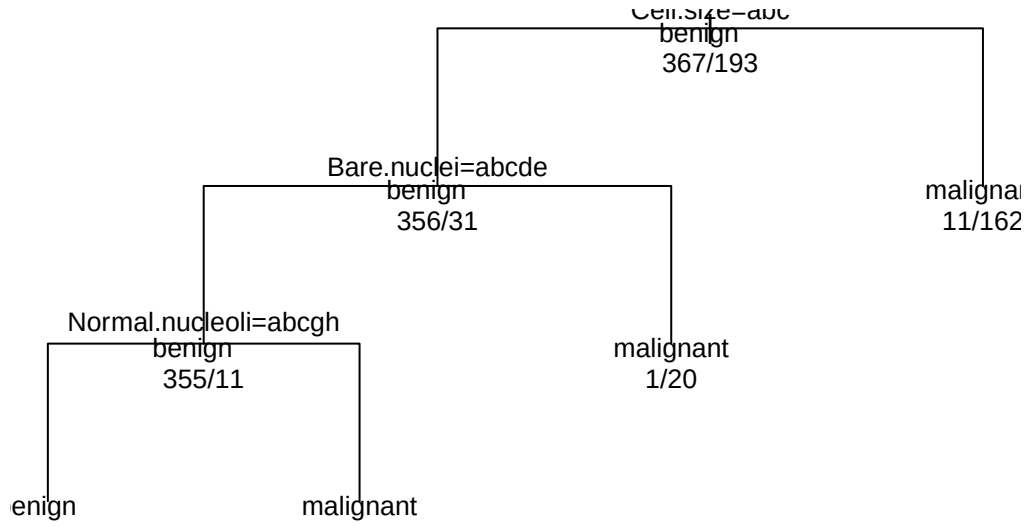
```
BC<-BreastCancer[-1] #removing ID column
```

```
set.seed(123)
trainIndex <- createDataPartition(BC$Class, p = .8,list = FALSE,times = 1)
head(trainIndex)
```

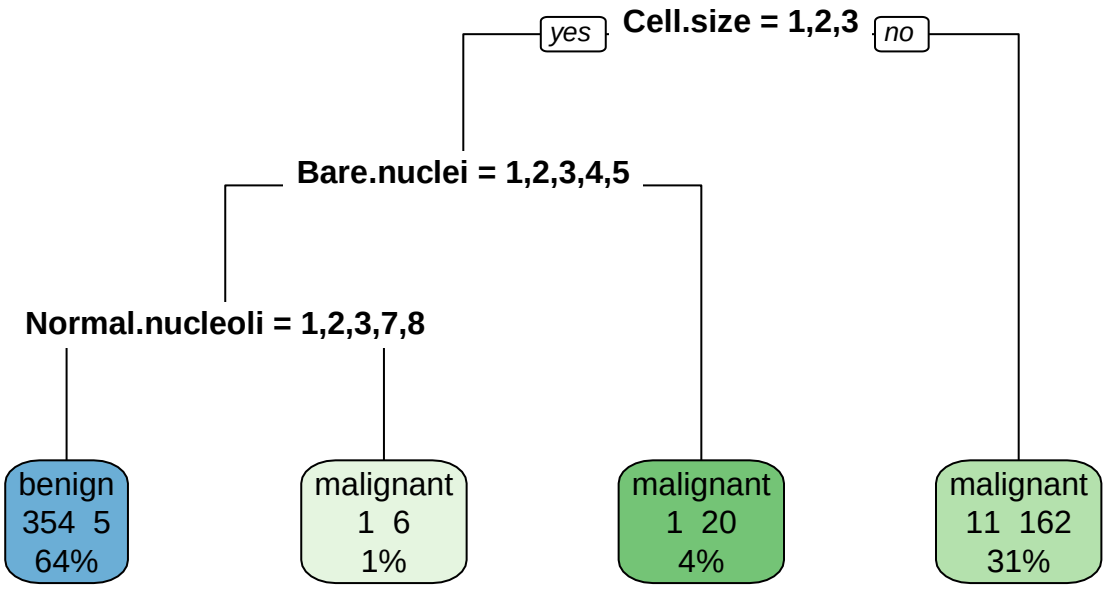
```
##      Resample1
## [1,]         1
## [2,]         2
## [3,]         4
## [4,]         5
## [5,]         6
## [6,]        10
```

```
BC_train_caret <- BC[ trainIndex,]
BC_test_caret <- BC[ trainIndex,]
```

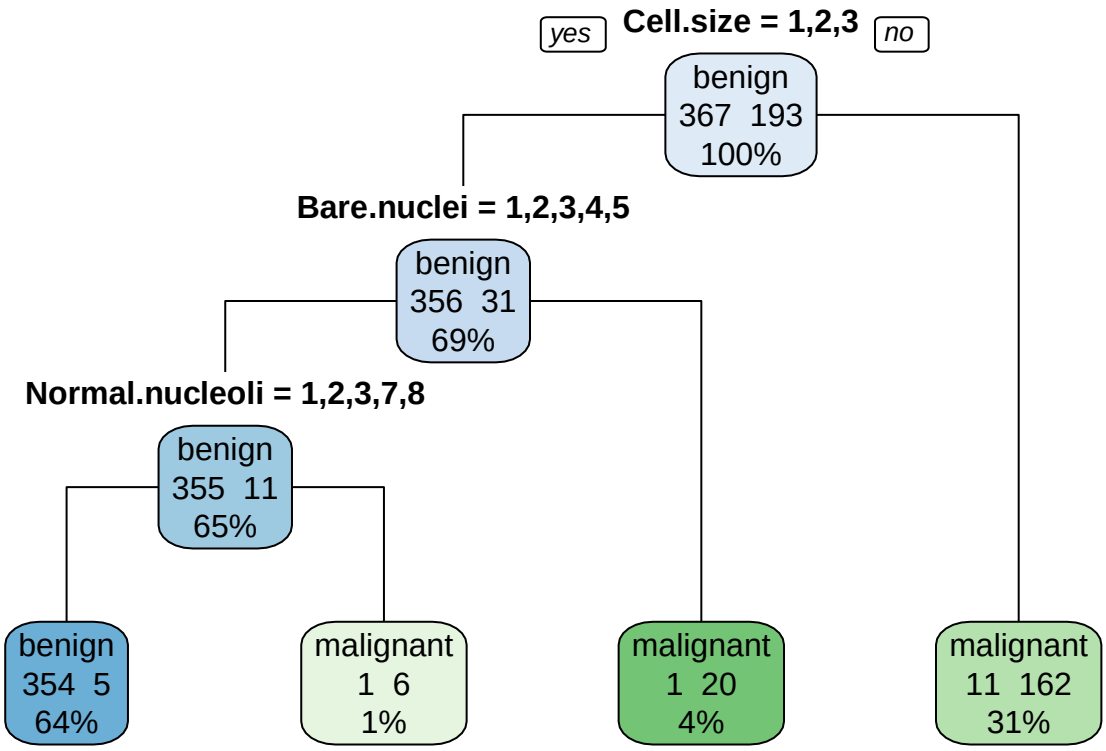
```
library(caret)
BC_rpart_caret <- rpart(BC_train_caret$Class~., method="class", parms = list(split="gini"), data=BC_tra

plot(BC_rpart_caret, uniform=TRUE, main="Classification Tree for Opioid Prescribers")
text(BC_rpart_caret, use.n=TRUE, all=TRUE, cex=0.8)
```




```
rpart.plot(BC_rpart_caret, type=0, extra=101)
```




```
rpart.plot(BC_rpart_caret, type=1, extra=101)
```




```
actual <- BC_test_caret$Class
predicted <- predict(BC_rpart_caret, BC_test_caret, type="class")
results.matrix <- confusionMatrix(predicted, actual, positive="malignant")
print(results.matrix)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  benign malignant
##   benign       354         5
##   malignant     13       188
##
##                Accuracy : 0.9679
##                  95% CI : (0.9497, 0.9808)
##     No Information Rate : 0.6554
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.9295
##  Mcnemar's Test P-Value : 0.09896
##
##             Sensitivity : 0.9741
##             Specificity : 0.9646
##          Pos Pred Value : 0.9353
##          Neg Pred Value : 0.9861
##              Prevalence : 0.3446
```

```
##          Detection Rate : 0.3357
##    Detection Prevalence : 0.3589
##       Balanced Accuracy : 0.9693
##
##        'Positive' Class : malignant
##
```

Our decision tree model does a slightly better job at predicting malignant tumors (true positives) (97%) than benign tumors (96%). This is what we want to see!

Improving Model Performance: Ensemble Models Approach

One decision tree suffers from high variance. The resulting tree depends on the training data. What we want is a procedure with low variance–meaning we should see similar results if the tree is applied repeatedly to distinct datasets. We will examine three ensemble models that are built on the basic decision trees:

1. Bagging (bootstrap aggregation)
2. Random forests (many trees = a forest)
3. Boosting

Bagging

Bagging is a 4 step process:

1. Generate B bootstrap samples from the training set.
2. Construct decision trees for all B bootstrap samples.
3. For each given test observation, we record the class predicted by each of the B trees.
4. The overall prediction is the most commonly occuring class among the B predictions. Majority voting wins.

Bagging averages many trees so it reduces the variance of unstable procedures (such as decision trees!). Bagging leads to improved prediction. The tradeoff is you lose interpretability and the ability to see simple structuree in a tree.

```
library(randomForest)
```

```
## randomForest 4.6-14

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

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
set.seed(123)

#Set mtry to equal all predictors. This means all predictors should be considered at each split. This i

BC.bag <- randomForest(Class~., mtry=9, data=BC_train_caret, na.action=na.omit, importance=TRUE)
```

Out of Bag (OOB) Error

A note on the out-of-bag (OOB) error is warranted. OOB is a measure of the test error popular in tree algorithms that use bootstrapping. Gareth et al. (2013) explained OOB as follows:

"Recall that the key to bagging is that trees are repeatedly fit to bootstrapped subsets of the observations. One that can show that on average, each bagged tree makes use of around two-thirds of the observations. **The remaining one-third of the observations not used to fit a given bagged tree are referred to as out-of-bag (OOB) observations.** We can predict the response for the ith observation using each of the trees in which that observation was OOB. This will yield around B/3 predictions for the ith observation. In order to obtain a single prediction for the ith observation, we take majority vote. This lead to a single OOB prediction for the ith observation. An OOB prediction can be obtained in this way for each of the n observations, from which the overall OOB classification error can be computed. The resulting OOB error is a valid estimate of the test error for the bagged model, since the response for each observation is predicted using only the trees that were not fit using that observation. . . .**It can be shown that with B sufficiently large, OOB error is virtually equivalent to leave-one-out cross-validation error**"(p. 317-318).

```r
print(BC.bag) #note the "out of bag" (OOB) error rate.
```

```
##
## Call:
##  randomForest(formula = Class ~ ., data = BC_train_caret, mtry = 9,      importance = TRUE, na.actio
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 9
##
##         OOB estimate of  error rate: 3.09%
## Confusion matrix:
##           benign malignant class.error
## benign       347        11  0.03072626
## malignant      6       186  0.03125000
```

Look at the mean decrease in accuracy of predictions in the OOB samples, when a given variable is excluded.

```r
importance(BC.bag, type=1)
```

```
##                 MeanDecreaseAccuracy
## Cl.thickness                17.255182
## Cell.size                   24.757200
## Cell.shape                  15.177876
## Marg.adhesion               12.645469
## Epith.c.size                 8.930145
## Bare.nuclei                 38.077858
## Bl.cromatin                 12.022141
## Normal.nucleoli             25.128828
## Mitoses                     12.429564
```

Look at the mean decrease in node impurity resulting from splits over that variable.

```
importance(BC.bag, type=2)
```

```
##                 MeanDecreaseGini
## Cl.thickness            8.494362
## Cell.size             127.499677
## Cell.shape             37.106073
## Marg.adhesion           3.240052
## Epith.c.size            4.671126
## Bare.nuclei            43.597554
## Bl.cromatin             9.421991
## Normal.nucleoli        13.448497
## Mitoses                 1.890239
```

```
actual <- BC_test_caret$Class
BC_predicted <- predict(BC.bag, newdata=BC_test_caret, type="class")
BC_results.matrix.bag <- confusionMatrix(BC_predicted, actual, positive="malignant")
print(BC_results.matrix.bag)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  benign malignant
##   benign       358         0
##   malignant      0       192
##
##                Accuracy : 1
##                  95% CI : (0.9933, 1)
##     No Information Rate : 0.6509
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.3491
##          Detection Rate : 0.3491
##    Detection Prevalence : 0.3491
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : malignant
##
```

Random Forest

Random forests consider only a subset of the predictors at each split. This means the node splits are not dominated by one or a few strong predictors, and, thus, give other (i.e. less strong) predictors more chances to be used. When we average the resulting trees, we get more reliable results since the individual trees are not dominated by a few strong predictors.

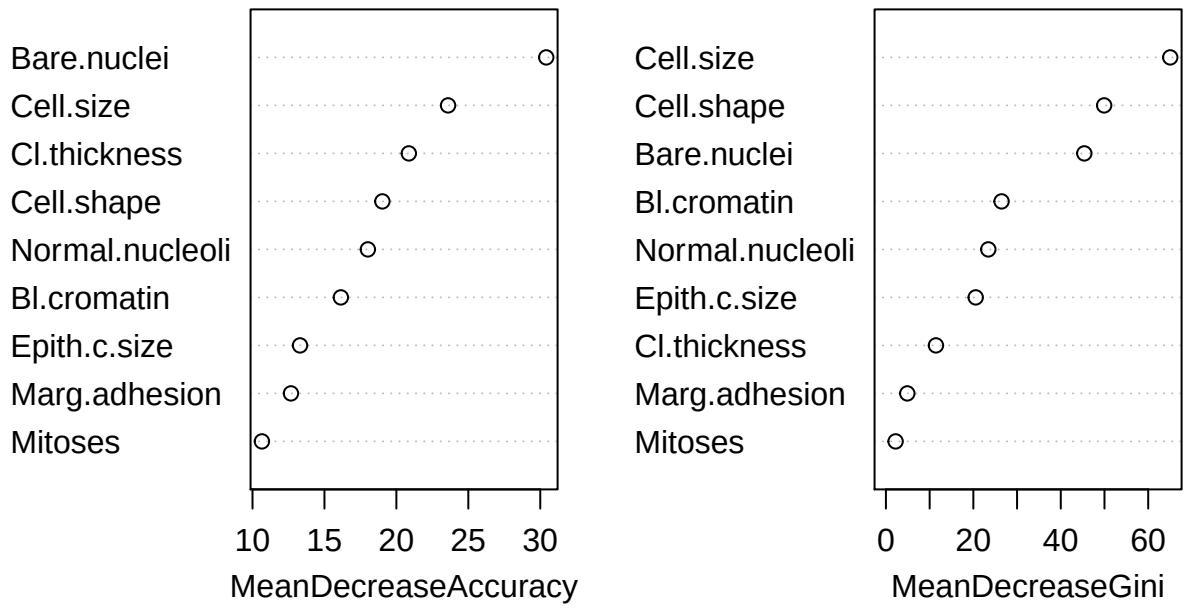
```
BC.RForest <- randomForest(Class ~.,data=BC_train_caret, mtry=3, ntree=600,na.action = na.omit, importa
print(BC.RForest)
```

```
##
## Call:
##  randomForest(formula = Class ~ ., data = BC_train_caret, mtry = 3,      ntree = 600, importance = T
##                Type of random forest: classification
##                      Number of trees: 600
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 2.55%
## Confusion matrix:
##           benign malignant class.error
## benign       348        10  0.02793296
## malignant      4       188  0.02083333
```

```
importance(BC.RForest)
```

```
##                    benign malignant MeanDecreaseAccuracy MeanDecreaseGini
## Cl.thickness    16.036895 19.996101             20.86536        11.440369
## Cell.size       18.060665 15.162733             23.59177        65.038349
## Cell.shape       8.982258 17.202727             19.02595        49.923790
## Marg.adhesion    7.027283 11.221708             12.67245         4.888690
## Epith.c.size     8.819947 10.709152             13.30307        20.539133
## Bare.nuclei     25.309709 26.055144             30.41473        45.382264
## Bl.cromatin     10.678503 13.531367             16.13982        26.454127
## Normal.nucleoli 16.814817  8.449485             18.01777        23.421303
## Mitoses         10.543983  1.670853             10.65552         2.208768
```

```
varImpPlot(BC.RForest)
```




```
actual <- BC_test_caret$Class
BC_predicted <- predict(BC.RForest, newdata=BC_test_caret, type="class")
BC_results.matrix.rf <- confusionMatrix(BC_predicted, actual, positive="malignant")
print(BC_results.matrix.rf)
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  benign malignant
##   benign       358         0
##   malignant      0       192
##
##                Accuracy : 1
##                  95% CI : (0.9933, 1)
##     No Information Rate : 0.6509
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.3491
```

```
##          Detection Rate : 0.3491
##    Detection Prevalence : 0.3491
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : malignant
##
```

Boosting

The boosting model involves:

- We fit a decision tree to the entire training set.
- We "boost" the observations that were misclassified by giving them higher weights. We fit another decision tree for these misclassified cases.
- We add the new tree to the existing tree to update the misclassified cases.

Note that the trees are that built later depend greatly on the trees already built. Learning slowly has shown to improve model accuracy while holding down variability.

```
library(adabag) #a popular boosting algorithm
```

```
## Loading required package: foreach

## Loading required package: doParallel

## Loading required package: iterators

## Loading required package: parallel
```

```
set.seed(123)
BC_adaboost <- boosting.cv(Class ~.,data=BC_train_caret, boos=TRUE, v=10) #.cv is adding cross validati
```

```
## i:  1 Sat Dec 15 11:09:14 2018
## i:  2 Sat Dec 15 11:09:40 2018
## i:  3 Sat Dec 15 11:10:06 2018
## i:  4 Sat Dec 15 11:10:32 2018
## i:  5 Sat Dec 15 11:10:58 2018
## i:  6 Sat Dec 15 11:11:24 2018
## i:  7 Sat Dec 15 11:11:50 2018
## i:  8 Sat Dec 15 11:12:16 2018
## i:  9 Sat Dec 15 11:12:42 2018
## i:  10 Sat Dec 15 11:13:08 2018
```

```
#don't worry about warning message. Also, this take a while to run.
BC_adaboost$confusion #confusion matrix for boosting
```

```
##                Observed Class
## Predicted Class benign malignant
##       benign       355         8
##       malignant     12       185
```

```
BC_adaboost$error #error rate for boosting (OOB)
```

```
## [1] 0.03571429
```

```
1-BC_adaboost$error #accuracy rate for boosting (OOB)
```

```
## [1] 0.9642857
```

ROC Curve: One More Performance Evaluation Metric

The ROC (receiver operating characteristics) curve displays the true positive rate (sensitivity) against the false positive rate (1-specificity). The closer the curve follows the left hand border and then the top left border of the ROC space, the more accurate the model.

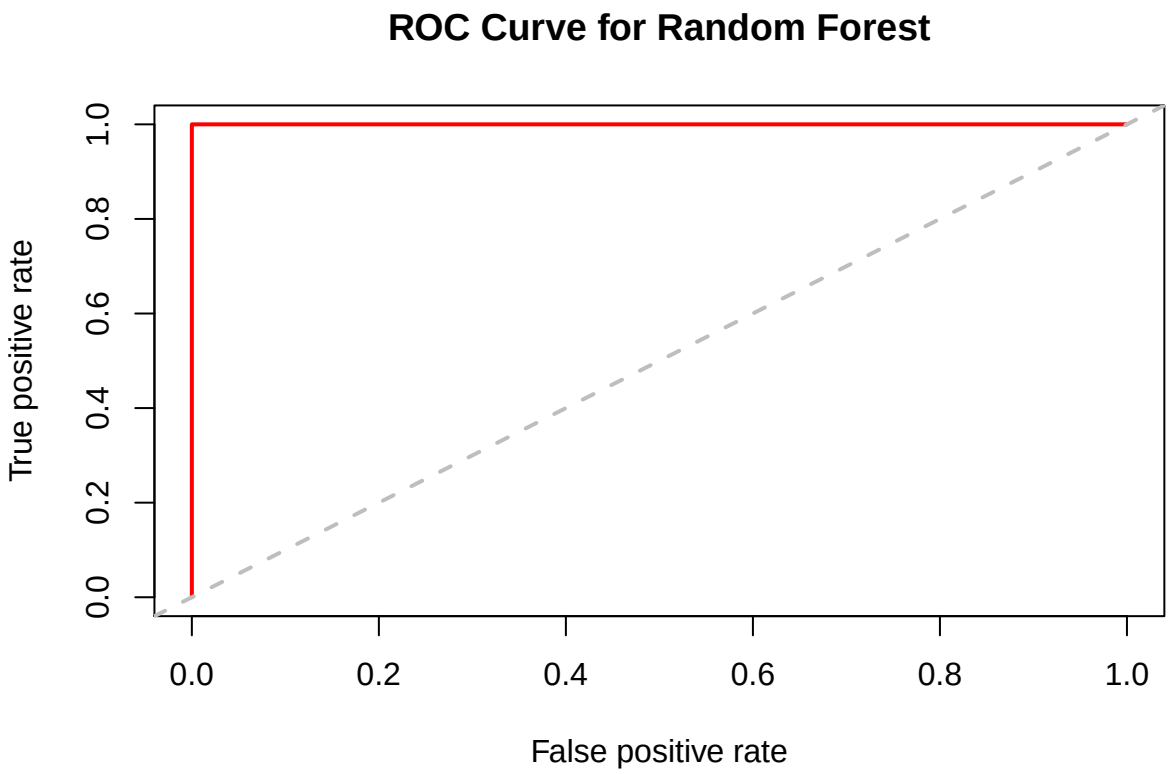
```
#Create a ROC curve
library(ROCR)
```

```
## Loading required package: gplots
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

```
BC.RForest_predict_prob<-predict(BC.RForest, type="prob", BC_test_caret)# same as above for predict, bu
BC.pred = prediction(BC.RForest_predict_prob[,2],BC_test_caret$Class)#use [,2] to pick the malignant cl
BC.RForest.perf = performance(BC.pred,"tpr","fpr") #true pos and false pos
plot(BC.RForest.perf ,main="ROC Curve for Random Forest",col=2,lwd=2)
abline(a=0,b=1,lwd=2,lty=2,col="gray")
```




```
unlist(BC.RForest.perf@y.values) #This is the AUC value (area under the ROC curve)
```

```
##   [1] 0.0000000 0.2135417 0.2812500 0.3437500 0.4010417 0.4322917 0.4531250
##   [8] 0.4895833 0.5208333 0.5520833 0.5729167 0.5885417 0.6041667 0.6145833
##  [15] 0.6302083 0.6510417 0.6770833 0.6979167 0.7135417 0.7187500 0.7239583
##  [22] 0.7343750 0.7447917 0.7500000 0.7604167 0.7708333 0.7812500 0.7864583
##  [29] 0.8020833 0.8125000 0.8177083 0.8229167 0.8333333 0.8385417 0.8541667
##  [36] 0.8645833 0.8697917 0.8750000 0.8802083 0.8854167 0.8906250 0.8958333
##  [43] 0.9010417 0.9062500 0.9166667 0.9270833 0.9322917 0.9375000 0.9427083
##  [50] 0.9479167 0.9531250 0.9583333 0.9635417 0.9687500 0.9739583 0.9791667
##  [57] 0.9843750 0.9895833 0.9947917 1.0000000 1.0000000 1.0000000 1.0000000
##  [64] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##  [71] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##  [78] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##  [85] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##  [92] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
##  [99] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
## [106] 1.0000000 1.0000000 1.0000000 1.0000000
```