Demography Informal Methods Seminar - Classification Trees

Corey Sparks, Ph.D. - UTSA Department of Demography

6/30/2020

Contents

Classification models

I would suggest you read section 5.1 of Introduction to Statistical Learning to get a full treatment of this topic

In classification methods, we are typically interested in using some observed characteristics of a case to predict a binary categorical outcome. This can be extended to a multi-category outcome, but the largest number of applications involve a 1/0 outcome.

In these examples, we will use the Demographic and Health Survey Model Data. These are based on the DHS survey, but are publicly available and are used to practice using the DHS data sets, but don't represent a real country.

In this example, we will use the outcome of contraceptive choice (modern vs other/none) as our outcome.

```
library(haven)
dat<-url("https://github.com/coreysparks/data/blob/master/ZZIR62FL.DTA?raw=true")
model.dat<-read_dta(dat)
```

Here we recode some of our variables and limit our data to those women who are not currently pregnant and who are sexually active.

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
model.dat2<-model.dat%>%
  mutate(region = v024,
         modcontra= ifelse(v364 ==1,1, 0),
         age = cut(v012, breaks = 5),
         livchildren=v218,
         educ = v106,
         currpreg=v213,
         wealth = as.factor(v190),
         partnered = ifelse(v701<=1, 0, 1),
         work = ifelse(v731%in%c(0,1), 0, 1),
         knowmodern=ifelse(v301==3, 1, 0),
         age2=v012^2,
         rural = ifelse(v025==2, 1,0),
         wantmore = ifelse(v605%in%c(1,2), 1, 0))%>%
  filter(currpreg==0, v536>0, v701!=9)%>% #notpreg, sex active
  dplyr::select(caseid, region, modcontra,age, age2,livchildren, educ, knowmodern, rural, wantmore, par
```

```
knitr::kable(head(model.dat2))
```

caseid	region	modcontra	age	age2	livchildren	educ	knowmodern	rural	wantmore	partnered	wealth	work
1 1 2	2	0	(28.6,35.4]	900	4	0	1	1	1	0	1	1
1 4 2	2	0	(35.4,42.2]	1764	2	0	1	1	0	0	3	1
1 4 3	2	0	(21.8,28.6]	625	3	1	1	1	0	0	3	1
1 5 1	2	0	(21.8,28.6]	625	2	2	1	1	1	0	2	1
1 6 2	2	0	(35.4,42.2]	1369	2	0	1	1	1	0	3	1
1 7 2	2	0	(15,21.8]	441	1	0	1	1	1	0	1	1

Cross-validation of predictive models

The term cross-validation refers to fitting a model on a subset of data and then testing it on another subset of the data. Typically this process is repeated several times.

The simplest way of doing this is to leave out a single observation, refit the model without it in the data, then predict its value using the rest of the data. This is called **hold out** cross-validation.

K-fold cross-validation is a process where you leave out a "group" of observations, it is as follows:

1. Randomize the data
2. Split the data into k groups, where k is an integer
3. For each of the k groups,
    - Take one of the groups as a hold out test set
    - Use the other k-1 groups as training data
    - Fit a model using the data on the k-1 groups, and test it on the hold out group

- Measure predictive accuracy of that model, and **throw the model away!**

4. Summarize the model accuracy over the measured model accuracy metrics

A further method is called **leave one out, or LOO** cross-validation. This combines hold out and k-fold cross-validation.

Why?

By doing this, we can see how model accuracy is affected by particular individuals, and overall allows for model accuracy to be measured repeatedly so we can assess things such as model **tuning parameters**.

If you remember from last time, the Lasso analysis depended upon us choosing a good value for the **penalty term** λ. In a cross-validation analysis, we can use the various resamplings of the data to examine the model's accuracy sensitivity to alternative values of this parameter.

This evaluation can either be done systematically, along a grid, or using a random search.

Alternative accuracy measures

We talked last time about using model accuracy as a measure of overall fit. This was calculated using the observed and predicted values of our outcome. For classification model, another commonly used metric of model predictive power is the Receiver Operating Characteristics (**ROC**) curve. This is a probability curve, and is often accompanied by the area under the curve (**AUC**) measure, which summarizes the separability of the classes. Together they tell you how capable the model is of determining difference between the classes in the data. The higher the values of these, the better, and they are both bound on (0,1).

A nice description of these are found here.

Regression trees

Regression trees are a common technique used in classification problems. Regression or classification trees attempt to find optimal splits in the data so that the best classification of observations can be found. Chapter 8 of Introduction to Statistical Learning is a good place to start with this.

Regression trees generate a set of splitting rules, which classify the data into a set of classes, based on combina-

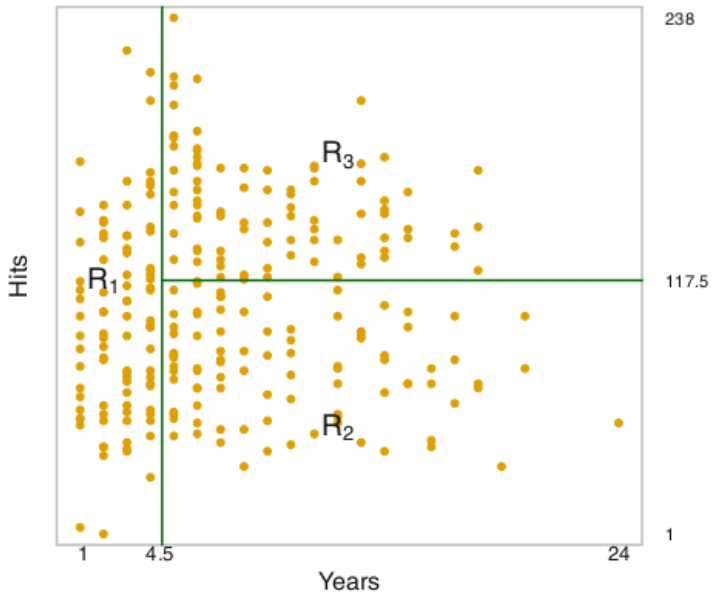


FIGURE 8.2. *The three-region partition for the* Hitters *data set from the regression tree illustrated in Figure 8.1.*

tions of the predictors.

This example, from the text, shows a 3 region partition of data on baseball hitter data. The outcome here is salary in dollars. Region 1 is players who've played less than 4.5 years, they typically have lower salary. The other 2 regions consist of players who've played longer than 4.5 years, and who have either less than 117.5 or greater than 117.5 hits. Those with more hits have higher salary than those with lower hits.

The regions can be thought of as nodes (or leaves) on a tree.

Here is a regression tree for these data. The Nodes are the mean salary (in thousands) for players in that region. For example, if a player has less than 4.5 years experiences, and have less than 39.5 hits, their average salary is 676.5 thousand dollars.

```
library(tree)
data(Hitters, package = "ISLR")
head(Hitters)
```

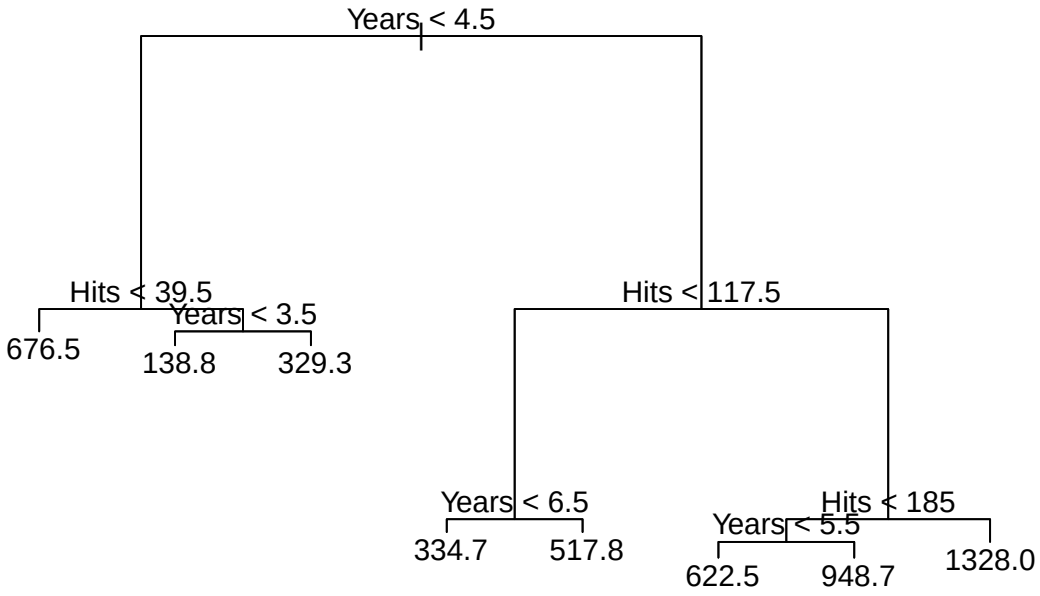
```
##                   AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun
## -Andy Allanson      293   66     1   30  29    14     1    293    66      1
## -Alan Ashby         315   81     7   24  38    39    14   3449   835     69
## -Alvin Davis        479  130    18   66  72    76     3   1624   457     63
## -Andre Dawson       496  141    20   65  78    37    11   5628  1575    225
## -Andres Galarraga   321   87    10   39  42    30     2    396   101     12
## -Alfredo Griffin    594  169     4   74  51    35    11   4408  1133     19
##                   CRuns CRBI CWalks League Division PutOuts Assists Errors
## -Andy Allanson       30   29     14      A        E     446      33     20
## -Alan Ashby         321  414    375      N        W     632      43     10
## -Alvin Davis        224  266    263      A        W     880      82     14
## -Andre Dawson       828  838    354      N        E     200      11      3
## -Andres Galarraga    48   46     33      N        E     805      40      4
## -Alfredo Griffin    501  336    194      A        W     282     421     25
##                   Salary NewLeague
```

```
## -Andy Allanson        NA         A
## -Alan Ashby        475.0         N
## -Alvin Davis       480.0         A
## -Andre Dawson      500.0         N
## -Andres Galarraga   91.5         N
## -Alfredo Griffin   750.0         A
```

```
fit1<-tree(Salary ~ Years+Hits, data=Hitters)
fit1
```

```
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 263 53320000  535.9
##    2) Years < 4.5 90  6769000  225.8
##      4) Hits < 39.5 5  3131000  676.5 *
##      5) Hits > 39.5 85  2564000  199.3
##       10) Years < 3.5 58   309400  138.8 *
##       11) Years > 3.5 27  1586000  329.3 *
##    3) Years > 4.5 173 33390000  697.2
##      6) Hits < 117.5 90  5312000  464.9
##       12) Years < 6.5 26   644100  334.7 *
##       13) Years > 6.5 64  4048000  517.8 *
##      7) Hits > 117.5 83 17960000  949.2
##       14) Hits < 185 76 13290000  914.3
##         28) Years < 5.5 8    82790  622.5 *
##         29) Years > 5.5 68 12450000  948.7 *
##       15) Hits > 185 7  3571000 1328.0 *
```

```
plot(fit1); text(fit1, pretty=1)
```



The cut points are decided by minimizing the residual sums of squares for a particular region. So we identify regions of the predictor space, $R_1, R_2, \ldots, R_j$ so that

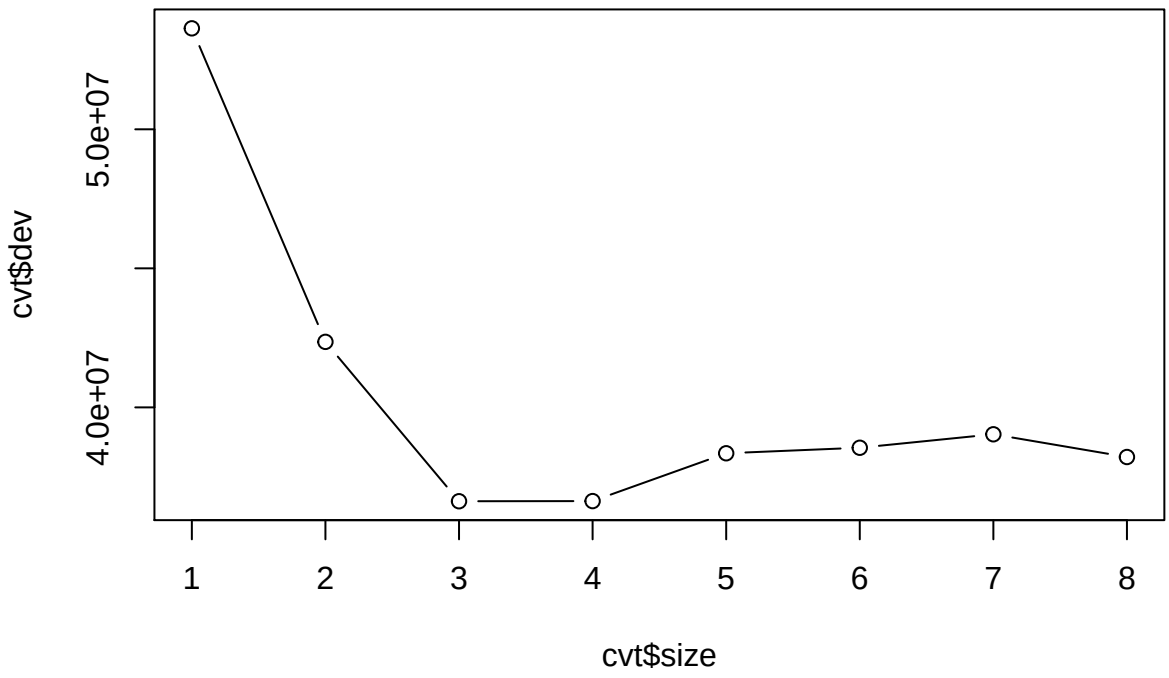
$$\sum_j \sum_{\in R_j} \left(y_i - \hat{y_{R_j}}\right)^2$$

where $\hat{y_{R_j}}$ is the mean for a particular region j.

Often this process may over-fit the data, meaning it creates too complicated of a tree (too many terminal nodes). It's possible to *prune* the tree to arrive at a simpler tree split that may be easier to interpret.

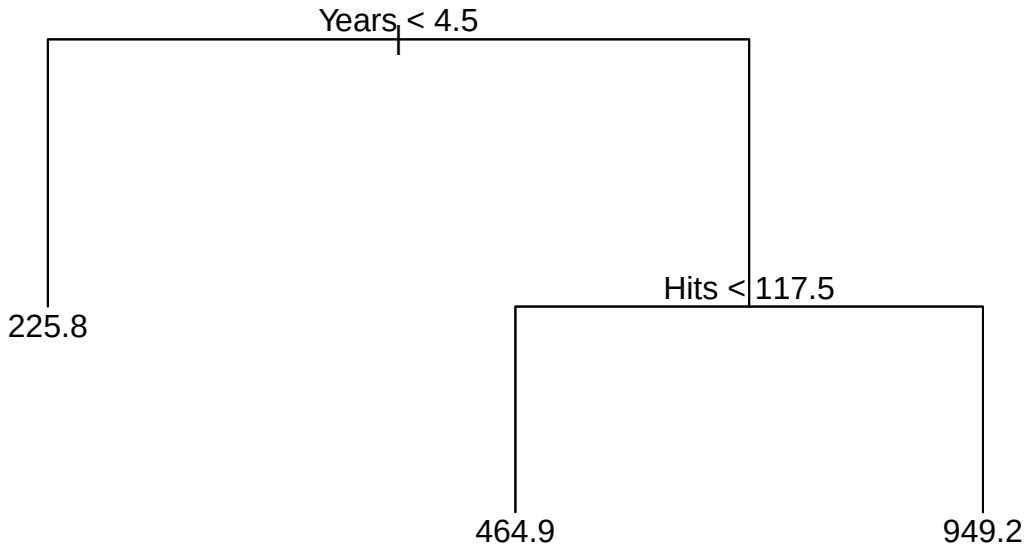
We can tune the tree depth parameter by cross-validation of the data, across different tree depths. In this case a depth of 3 is optimal.

```
cvt<-cv.tree(fit1)
plot(cvt$size, cvt$dev, type="b")
```



Then, we can prune the tree, to basically get the tree version of the figure from above

```
tree2<-prune.tree(fit1, best=3)
plot(tree2); text(tree2, pretty=1)
```




```
# plot(x=Hitters$Years, y=Hitters$Hits)
# abline(v=4.5,  col=3, lwd=3)
# abline(h=117.5, col=4, lwd=3)
```

Prediction works by assigning the mean value from a region to an observation who matches the decision rule. For example, let's make up a player who has 6 years experience and 200 hits

```
new<-data.frame(Hits=200, Years=6)

pred<-predict(fit1, newdata = new)
pred
```

```
##      1
## 1327.5
```

Classification trees

If our outcome is categorical, or binary, the tree will be a *classification tree*. Instead of the mean of a particular value being predicted, the classification tree predicts the value of the *most common class* at a particular terminal node. So in addition to the tree predicting the class at each node, it also gives the class proportions at each node. The *classification error rate* is the percent of of observations at a node that do not belong to the most common class.

$$Error = 1 - max(\hat{p}_{mk})$$

This is not a good method for growing trees, and instead either the Gini index or the entropy is measured at each node:

$$Gini = \sum_k \hat{p}_{mk}(1 - \hat{p}_{mk})$$

The Gini index is used as a measure of *node purity*, if a node only contains 1 class, it is considered *pure*

$$Entropy = D = -\sum_k \hat{p}_{mk} \log \hat{p}_{mk}$$

Bagging and Random Forests

The example above is a single "tree", if we did this type of analysis a large number of times, then we would end up with a *forest* of such trees.

Bagging is short for *bootstrap aggragation*. This is a general purpose procedure for reducing the variance in a statistical test, but it is also commonly used in regression tree contexts. How this works in this setting is the data are bootstrapped into a large number of training sets, each of the same size. The regression tree is fit to each of these large number of trees and not pruned. By averaging these bootstrapped trees, the accuracy is actually higher than for a single tree alone.

Random forests not only bag the trees, but at each iteration a different set of predictors is chosen from the data, so not only do we arrive at a more accurate bagged tree, but we can also get an idea of how important any particular variable is, based on its averaged Gini impurity across all the trees considered.

```
## Warning: Number of logged events: 654
```

simple example using PRB data - Regression tree

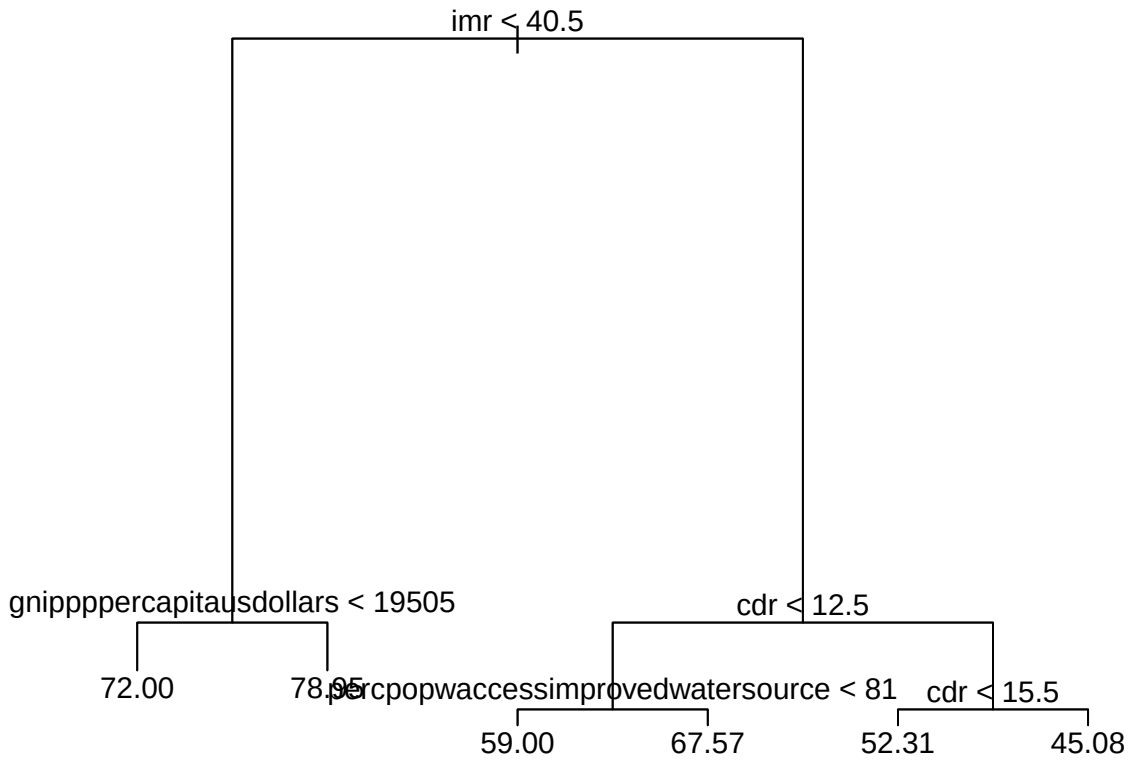
```
#set up training set identifier
train1<-sample(1:dim(prb2)[1], size = .75*dim(prb2)[1], replace=T)

fit<-tree(e0total~., data=prb2[train1,])
```

```
## Warning in tree(e0total ~ ., data = prb2[train1, ]): NAs introduced by coercion
```

```
fit
```

```
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 156 18790.0 67.60
##    2) imr < 40.5 101  1741.0 74.54
##      4) gnippppercapitausdollars < 19505 64   438.0 72.00 *
##      5) gnippppercapitausdollars > 19505 37   171.9 78.95 *
##    3) imr > 40.5 55  3243.0 54.85
##      6) cdr < 12.5 26   593.5 61.31
##       12) percpopwaccessimprovedwatersource < 81 19    94.0 59.00 *
##       13) percpopwaccessimprovedwatersource > 81 7   123.7 67.57 *
##      7) cdr > 12.5 29   595.9 49.07
##       14) cdr < 15.5 16   115.4 52.31 *
##       15) cdr > 15.5 13   104.9 45.08 *
```

```
plot(fit); text(fit, pretty=1)
```

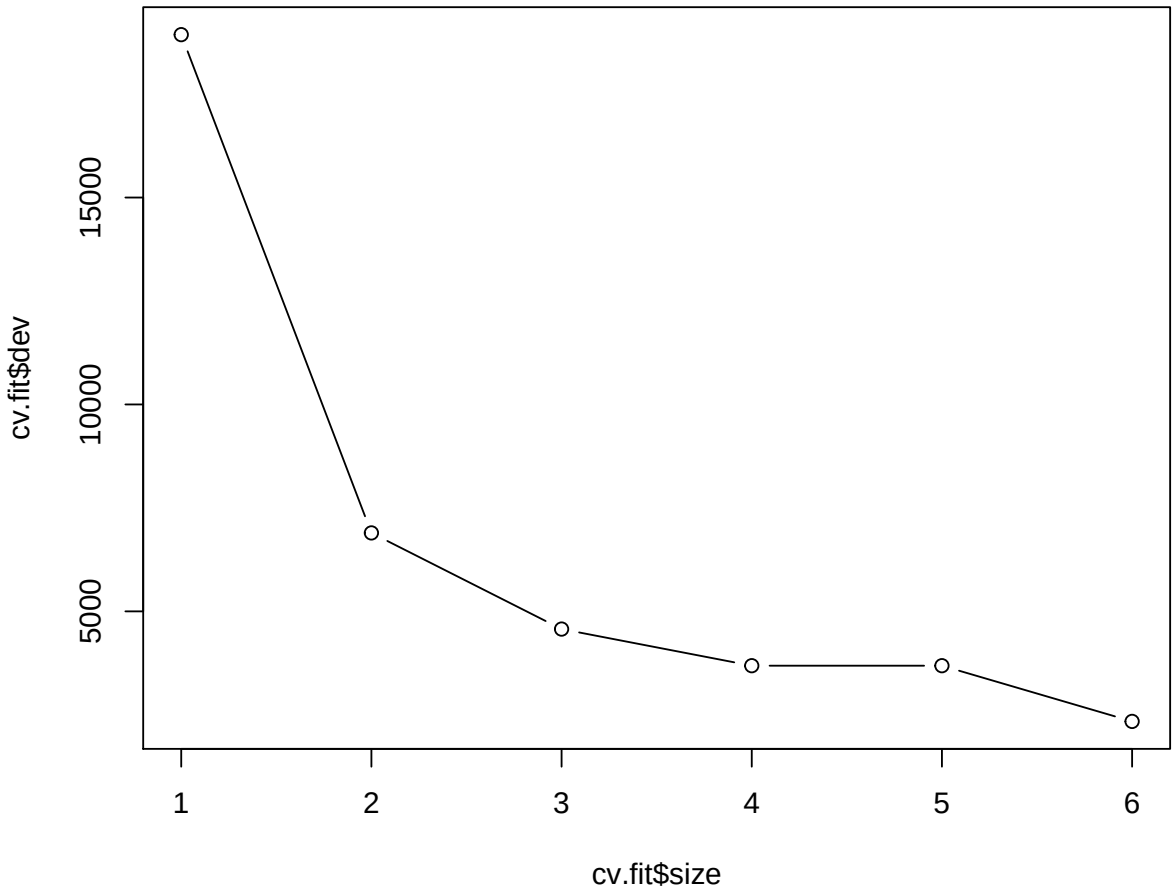



```
cv.fit<-cv.tree(fit)
```

```
## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion
```

```
## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion
```

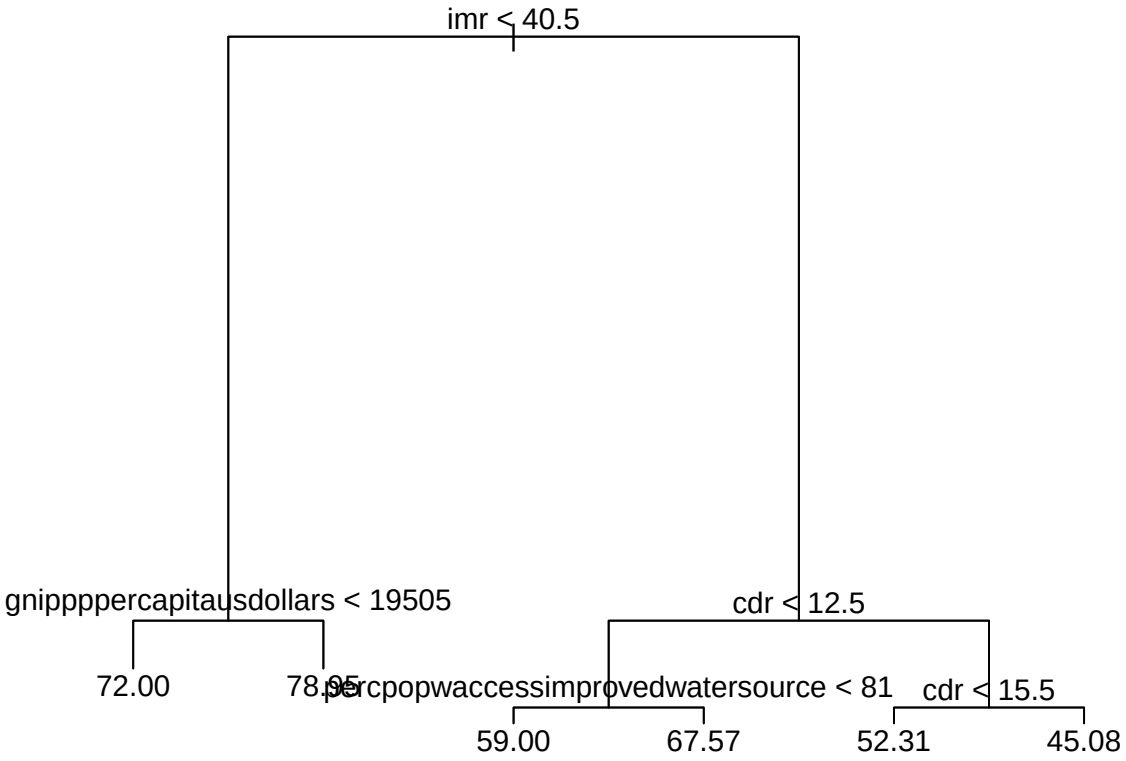
```r
plot(cv.fit$size, cv.fit$dev, type="b")
```




```
pt1<-prune.tree(fit, best=7)
```

```
## Warning in prune.tree(fit, best = 7): best is bigger than tree size
```

```
plot(pt1); text(pt1, pretty=1)
```



Bagged regression tree from PRB data - 100 trees - 3 variables each

```
library(randomForest)
```

```
## randomForest 4.6-14
```

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

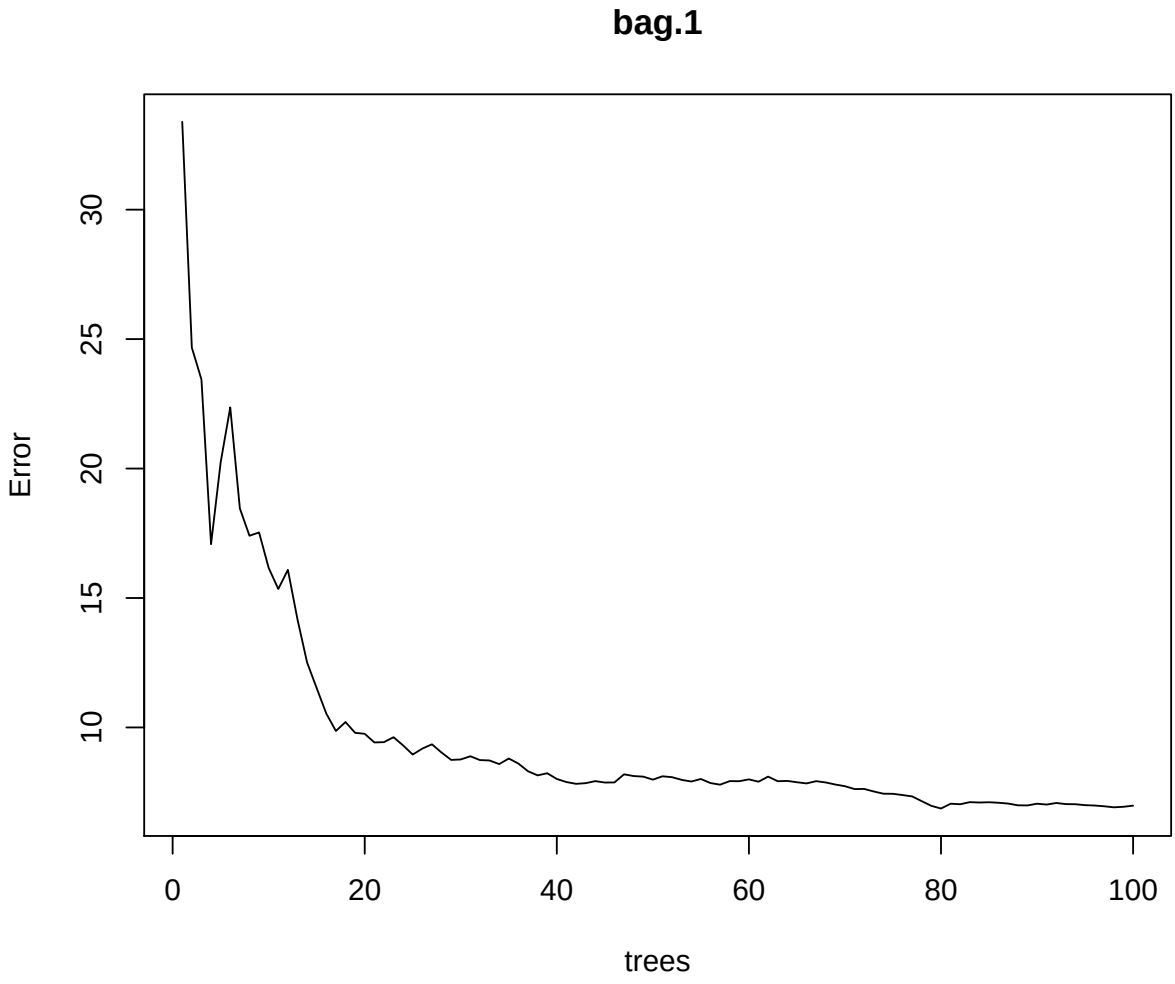
```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
set.seed(1115)
bag.1<-randomForest(e0total~., data=prb2[train1,], mtry=3, ntree=100,importance=T) #mtry = 3; choose 3
bag.1
```

```
##
## Call:
##  randomForest(formula = e0total ~ ., data = prb2[train1, ], mtry = 3,      ntree = 100, importance =
##                Type of random forest: regression
##                      Number of trees: 100
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 6.979873
##                     % Var explained: 94.2
```

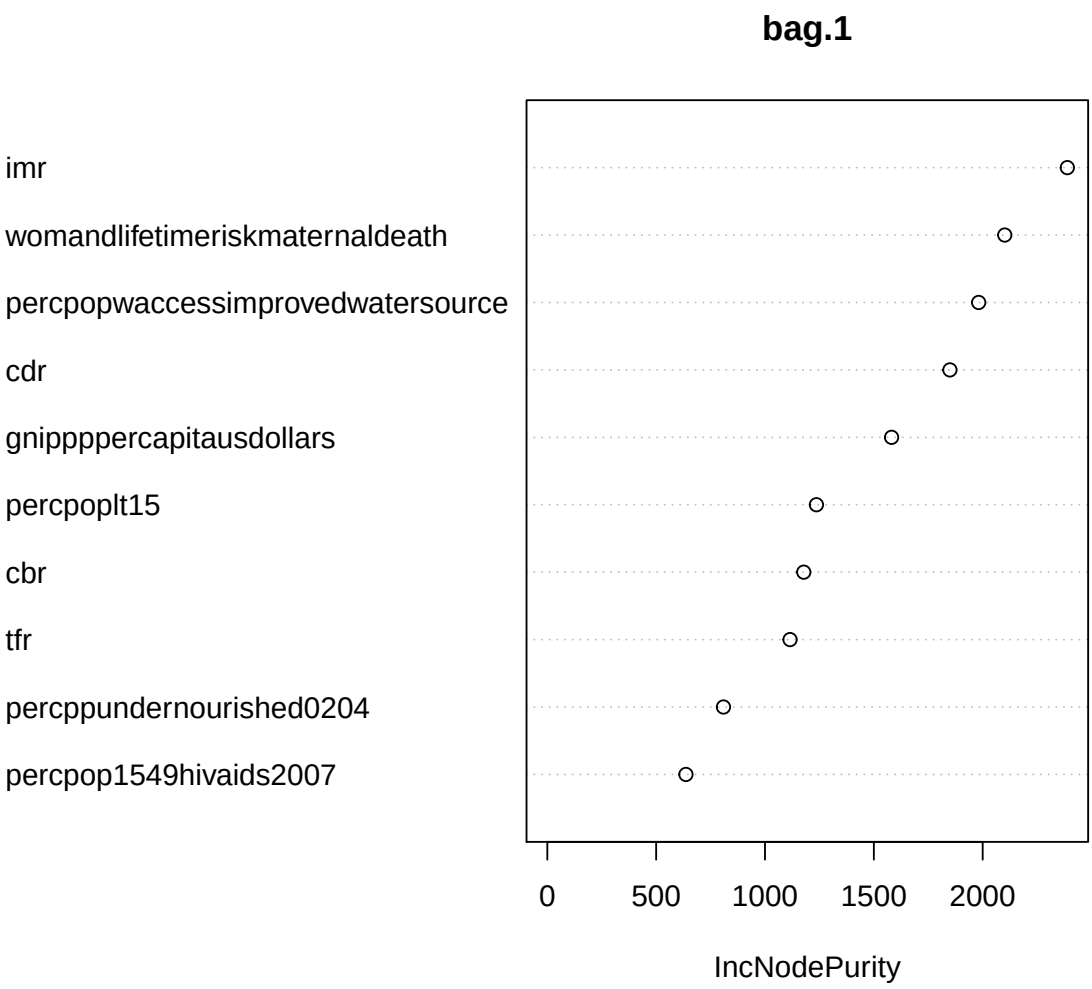
```r
plot(bag.1)
```



```r
importance(bag.1)
```

```
##                                    %IncMSE IncNodePurity
## continent                         2.813376      176.6611
## population.                       3.534477      253.8825
## cbr                               3.823816     1178.1116
## cdr                               6.944041     1849.5647
## rate.of.natural.increase          4.520790      514.3506
## net.migration.rate                2.135764      173.9132
## imr                               5.421584     2388.9598
## womandlifetimeriskmaternaldeath   6.375722     2101.4516
## tfr                               5.171558     1115.0652
## percpoplt15                       4.708656     1236.2414
## percpopgt65                       2.994488      553.2614
## percurban                         1.549603      109.5392
## percpopinurbangt750k              3.136486      118.5853
## percpop1549hivaids2007            6.709671      636.7492
## percmarwomcontraall               4.160640      494.8039
## percmarwomcontramodern            4.080922      437.4198
## percppundernourished0204          5.156677      809.1502
## motorvehper1000pop0005            4.395351      525.0908
## percpopwaccessimprovedwatersource 5.101440     1981.6837
## gnippppercapitausdollars          6.124880     1581.5010
## popdenspersqkm                    4.048486      164.6494
```

```r
varImpPlot(bag.1, n.var = 10, type=2)
```



Classification tree for life expectancy - low or high

```
prb2$lowe0<-as.factor(ifelse(prb2$e0total<median(prb2$e0total), "low", "high"))

fit<-tree(lowe0~., data=prb2[train1,-12])
```

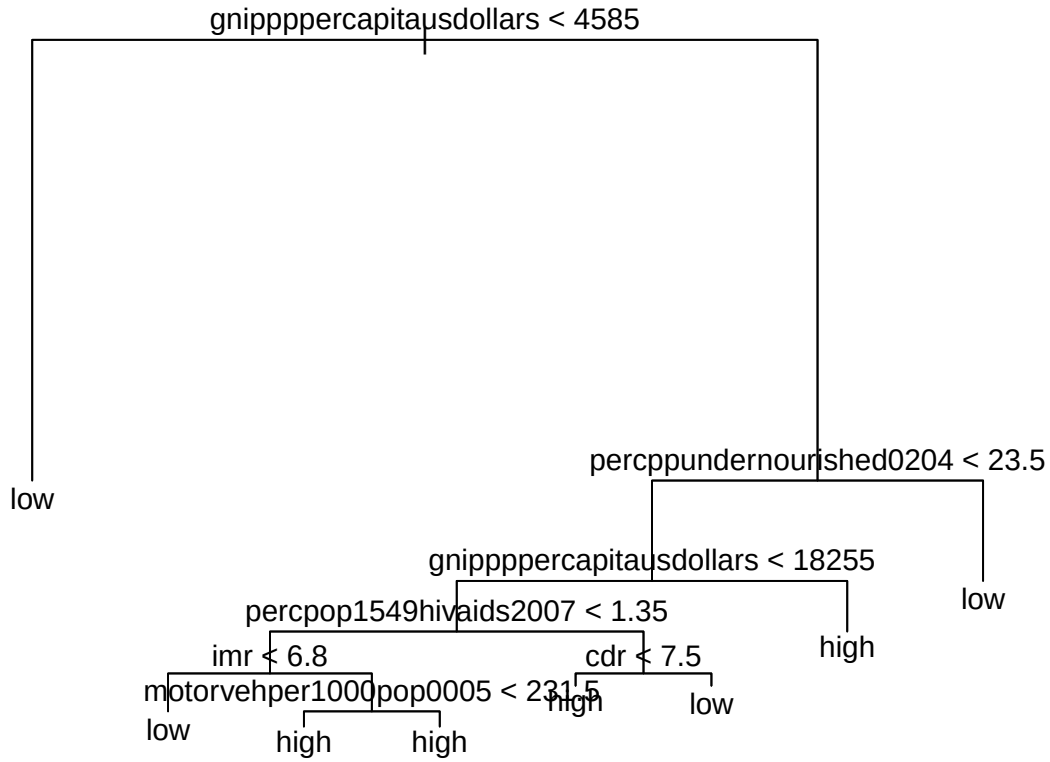
```
## Warning in tree(lowe0 ~ ., data = prb2[train1, -12]): NAs introduced by coercion
```

```
fit
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 156 216.200 high ( 0.50641 0.49359 )
##    2) gnippppercapitausdollars < 4585 57   0.000 low ( 0.00000 1.00000 ) *
##    3) gnippppercapitausdollars > 4585 99  99.630 high ( 0.79798 0.20202 )
##      6) percppundernourished0204 < 23.5 89  66.580 high ( 0.87640 0.12360 )
```

```
##       12) gnippppercapitausdollars < 18255 51  53.180 high ( 0.78431 0.21569 )
##         24) percpop1549hivaids2007 < 1.35 39  25.790 high ( 0.89744 0.10256 )
##           48) imr < 6.8 5   6.730 low ( 0.40000 0.60000 ) *
##           49) imr > 6.8 34   9.023 high ( 0.97059 0.02941 )
##             98) motorvehper1000pop0005 < 231.5 29   0.000 high ( 1.00000 0.00000 ) *
##             99) motorvehper1000pop0005 > 231.5 5   5.004 high ( 0.80000 0.20000 ) *
##         25) percpop1549hivaids2007 > 1.35 12  16.300 low ( 0.41667 0.58333 )
##           50) cdr < 7.5 6   7.638 high ( 0.66667 0.33333 ) *
##           51) cdr > 7.5 6   5.407 low ( 0.16667 0.83333 ) *
##       13) gnippppercapitausdollars > 18255 38   0.000 high ( 1.00000 0.00000 ) *
##      7) percppundernourished0204 > 23.5 10   6.502 low ( 0.10000 0.90000 ) *
```

```
plot(fit); text(fit, pretty=1)
```




```
cv.fit<-cv.tree(fit)
```

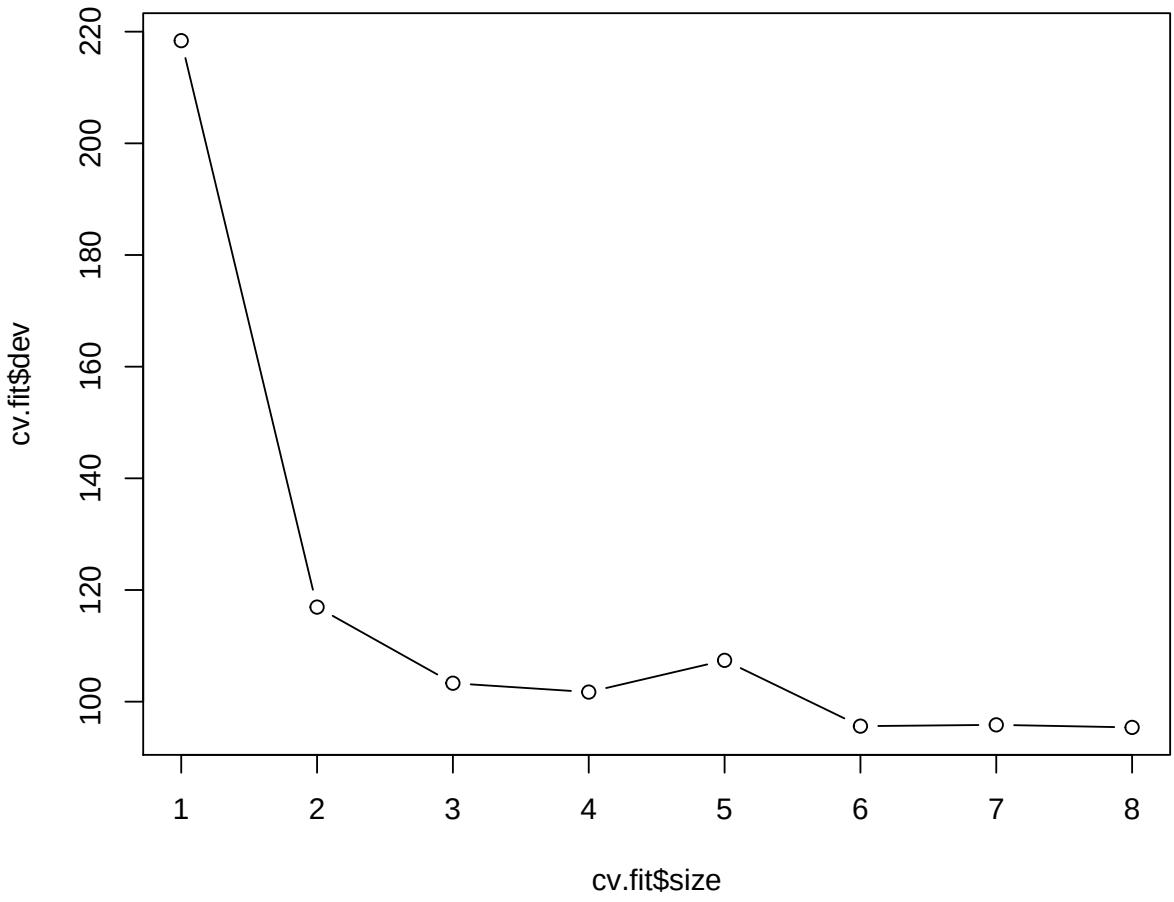
```
## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion
```

```
## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion

## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion

## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion
```

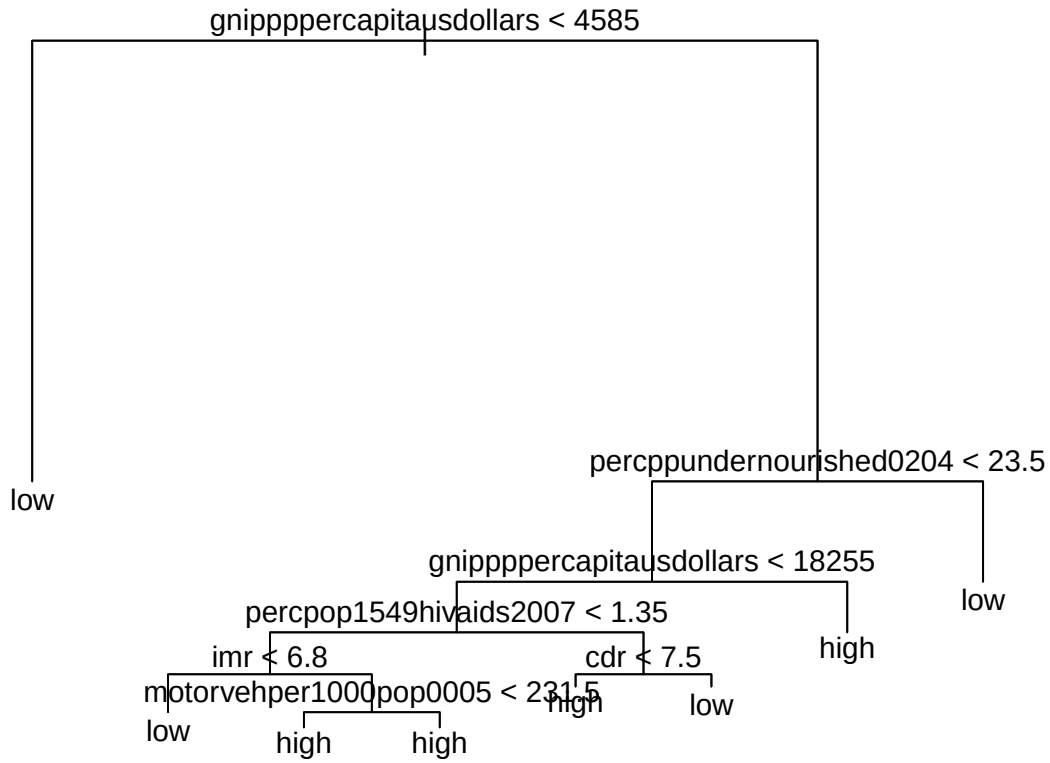
```
cv.fit
```

```
## $size
## [1] 8 7 6 5 4 3 2 1
##
## $dev
## [1]  95.39024  95.84717  95.61339 107.39927 101.71617 103.30371 116.93675
## [8] 218.38979
##
## $k
## [1]       -Inf   3.255734   4.018992  10.039956  11.088435  13.394839  26.553018
## [8] 116.604600
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
plot(cv.fit$size, cv.fit$dev, type="b")
```



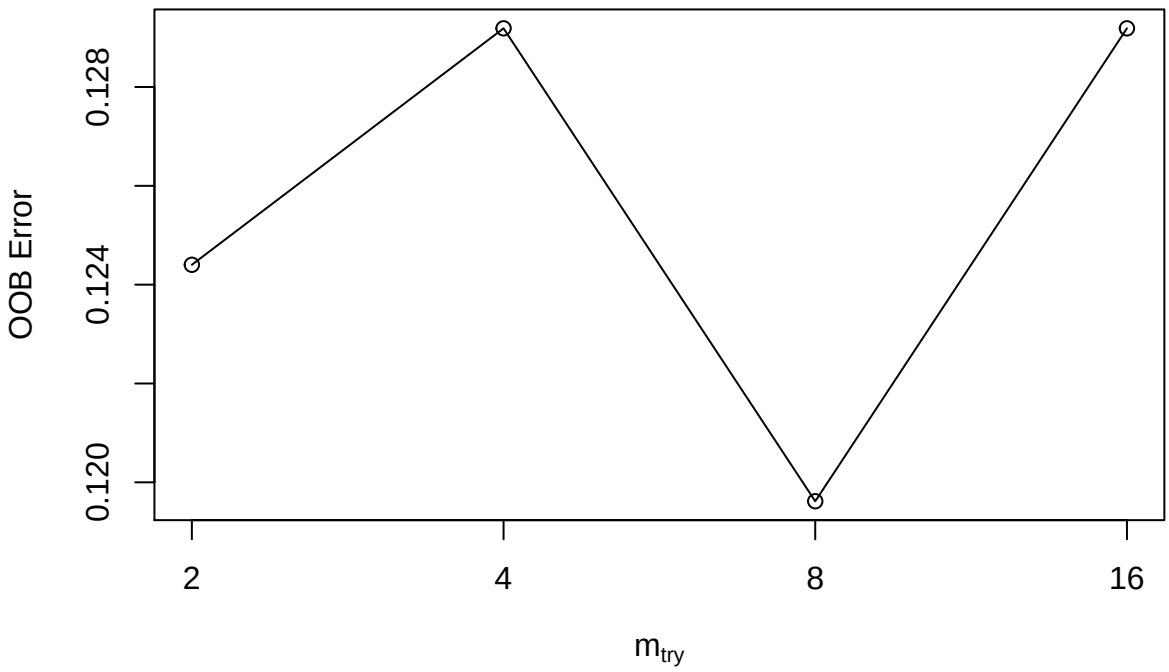

```
pt1<-prune.tree(fit, best=cv.fit$size[which.min(cv.fit$dev)])
plot(pt1); text(pt1, pretty=1)
```



Random forest tree for PRB low life expectancy

```
#Tune to find best number of variables to try
t1<-tuneRF(y=prb2$lowe0, x=prb2[,c(-12,-23)], trace=T, stepFactor = 2, ntreeTry = 1000, plot=T)
```

```
## mtry = 4  OOB error = 12.92%
## Searching left ...
## mtry = 2     OOB error = 12.44%
## 0.03703704 0.05
## Searching right ...
## mtry = 8     OOB error = 11.96%
## 0.07407407 0.05
## mtry = 16    OOB error = 12.92%
## -0.08 0.05
```



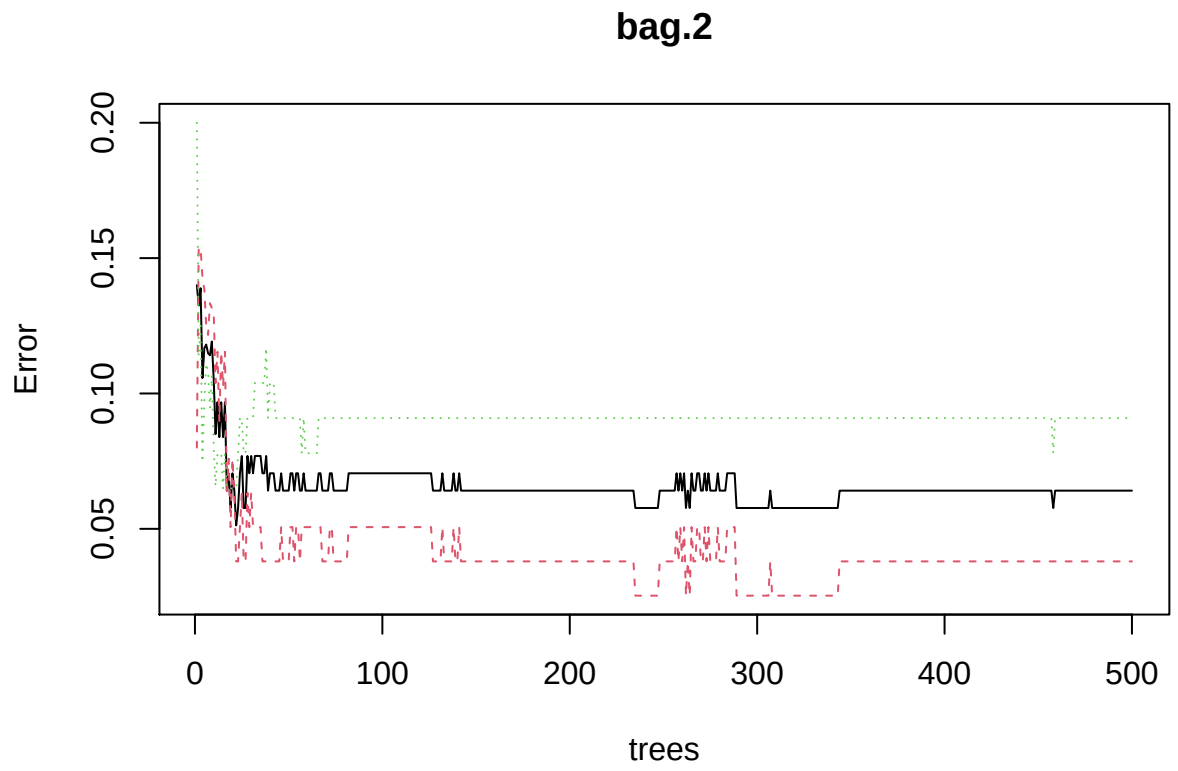

```
t1
```

```
##        mtry  OOBError
## 2.OOB     2 0.1244019
## 4.OOB     4 0.1291866
## 8.OOB     8 0.1196172
## 16.OOB   16 0.1291866
```

```
bag.2<-randomForest(lowe0~., data=prb2[train1,-12], mtry=4, ntree=500,importance=T)
bag.2
```

```
##
## Call:
##  randomForest(formula = lowe0 ~ ., data = prb2[train1, -12], mtry = 4,      ntree = 500, importance =
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 6.41%
## Confusion matrix:
##      high low class.error
## high   76   3  0.03797468
## low     7  70  0.09090909
```

```r
plot(bag.2)
```

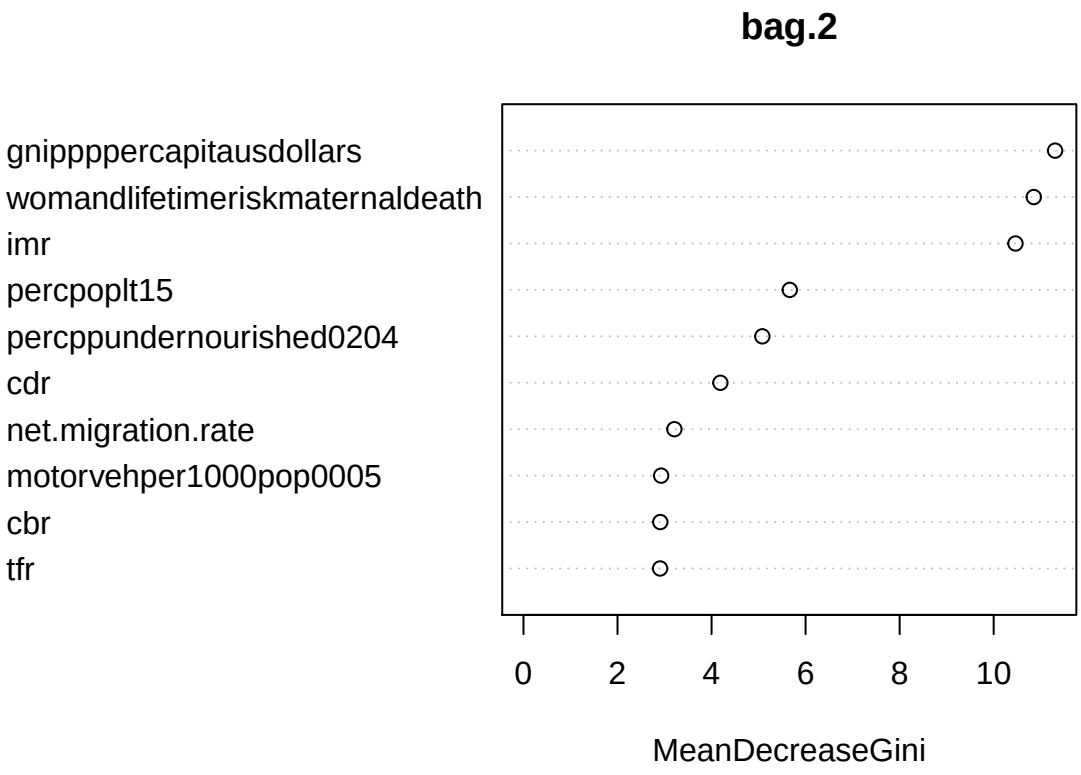



```r
importance(bag.2,scale = T )
```

```
##                                        high       low MeanDecreaseAccuracy
## continent                          1.609506  3.318029             3.291314
## population.                        2.872325  3.454993             4.540342
## cbr                                5.843009  5.192916             7.796013
## cdr                               14.861541 11.797154            16.504268
## rate.of.natural.increase           5.910821  7.739404             8.787250
## net.migration.rate                10.050915 11.093097            13.228865
## imr                               18.002051 10.200624            18.699508
## womandlifetimeriskmaternaldeath   15.244983 11.161658            17.184879
## tfr                                5.828383  6.048005             7.959735
## percpoplt15                        8.329239  5.392705             9.103211
## percpopgt65                        5.986648  4.708002             7.042130
## percurban                          8.698041  7.629242            10.983650
## percpopinurbangt750k               4.357403  6.950224             7.144037
## percpop1549hivaids2007             6.789327  4.270188             7.504811
## percmarwomcontraall                6.258846  5.120535             7.496084
## percmarwomcontramodern             5.742191  4.948921             7.027564
## percppundernourished0204          11.501196  6.849468            12.698909
## motorvehper1000pop0005             6.638286  6.956651             9.403270
## percpopwaccessimprovedwatersource  6.051919  7.408459             9.086937
## gnippppercapitausdollars          18.826763 10.843126            19.439941
```

```
## popdenspersqkm                     5.242944  7.553589            7.925682
##                                  MeanDecreaseGini
## continent                               0.4265788
## population.                             1.0654517
## cbr                                     2.9099166
## cdr                                     4.1879376
## rate.of.natural.increase                2.8684182
## net.migration.rate                      3.2092506
## imr                                    10.4639648
## womandlifetimeriskmaternaldeath        10.8528937
## tfr                                     2.9057764
## percpoplt15                             5.6634735
## percpopgt65                             1.4152021
## percurban                               2.5952542
## percpopinurbangt750k                    1.0900905
## percpop1549hivaids2007                  1.1555316
## percmarwomcontraall                     1.4426142
## percmarwomcontramodern                  1.2690258
## percppundernourished0204                5.0803136
## motorvehper1000pop0005                  2.9279292
## percpopwaccessimprovedwatersource       2.8453461
## gnippppercapitausdollars               11.3063073
## popdenspersqkm                          1.7679285
```

```
varImpPlot(bag.2, n.var = 10, type=2)
```



```
pred<-predict(bag.2, newdata=prb2[-train1,])
table(pred, prb2[-train1, "lowe0"])
```

```
##
## pred   high low
##   high   41   6
##   low     3  43
```

```
mean(pred==prb2[-train1, "lowe0"]) #accuracy
```

```
## [1] 0.9032258
```

More complicated example

using caret to create training and test sets.

We use an 80% training fraction

```
library(caret)
```

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

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

```
##
## Attaching package: 'ggplot2'
```

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

```
set.seed(1115)
train<- createDataPartition(y = model.dat2$modcontra , p = .80, list=F)

dtrain<-model.dat2[train,]
```

```
## Warning: The `i` argument of ``[`()` can't be a matrix as of tibble 3.0.0.
## Convert to a vector.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

```
dtest<-model.dat2[-train,]
```

Create design matrix

If we have a mixture of factor variables and continuous predictors in our analysis, it is best to set up the design matrix for our models before we run them. Many methods within `caret` won't use factor variables correctly unless we set up the dummy variable representations first.

```
y<-dtrain$modcontra
y<-as.factor(ifelse(y==1, "mod", "notmod"))
x<-model.matrix(~factor(region)+factor(age)+livchildren+factor(rural)+factor(wantmore)+factor(educ)+par
x<-data.frame(x)[,-1]


table(y)
```

```
## y
##    mod notmod
##    719   3410
```

```
prop.table(table(y))
```

```
## y
##       mod    notmod
## 0.1741342 0.8258658
```

```
xtest<-model.matrix(~factor(region)+factor(age)+livchildren+factor(rural)+factor(wantmore)+factor(educ)
xtest<-xtest[,-1]
xtest<-data.frame(xtest)

yt<-dtest$modcontra
yt<-as.factor(ifelse(yt==1, "mod", "notmod"))
prop.table(table(yt))
```

```
## yt
##       mod    notmod
## 0.2102713 0.7897287
```

Set up caret for repeated 10 fold cross-validation

To set up the training controls for a caret model, we typically have to specify the type of re-sampling method, the number of resamplings, the number of repeats (if you're doing repeated sampling). Here we will do a 10 fold cross-validation, 10 is often recommended as a choice for k based on experimental sensitivity analysis.

The other things we specify are:

- repeats - These are the number of times we wish to repeat the cross-validation, typically 3 or more is used
- classProbs = TRUE - this is necessary to assess accuracy in the confusion matrix
- search = "random" is used if you want to randomly search along the values of the tuning parameter
- sampling - Here we can specify alternative sampling methods to account for unbalanced outcomes
- SummaryFunction=twoClassSummary - keeps information on the two classes of the outcome
- savePredictions = T - have the process save all the predicted values throughout the process, we need this for the ROC curves

```
fitctrl <- trainControl(method="repeatedcv",
                        number=10,
                        repeats=5,
```

```
                        classProbs = TRUE,
                       search="random", #randomly search on different values of the tuning parameters
                       sampling = "down", #optional, but good for unbalanced outcomes like this one
                       summaryFunction=twoClassSummary,
                       savePredictions = "all")
```

Train regression classification models using caret

Here we fit a basic regression classification tree using the `rpart()` function

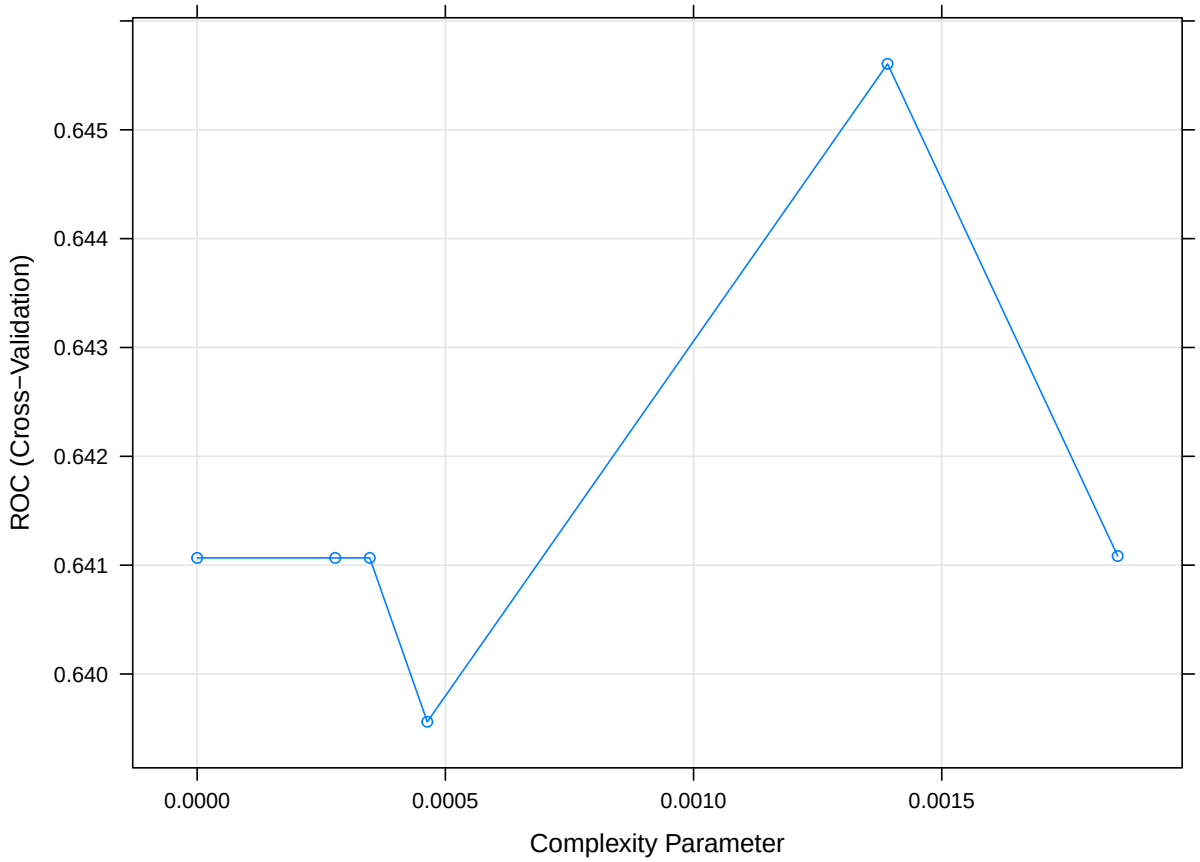
```
fitctrl <- trainControl(method="cv",
                        number=10,
                        #repeats=5,
                        classProbs = TRUE,
                       search="random",
                       sampling = "down",
                       summaryFunction=twoClassSummary,
                       savePredictions = "all")

rp1<-caret::train(y=y, x=x,
           metric="ROC",
           method ="rpart",
          tuneLength=20, #try 20 random values of the tuning parameters
           trControl=fitctrl,
          preProcess=c("center", "scale"))

rp1
```

```
## CART
##
## 4129 samples
##   19 predictor
##    2 classes: 'mod', 'notmod'
##
## Pre-processing: centered (19), scaled (19)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 3716, 3716, 3716, 3716, 3717, 3716, ...
## Addtional sampling using down-sampling prior to pre-processing
##
## Resampling results across tuning parameters:
##
##   cp            ROC        Sens       Spec
##   0.0000000000  0.6410664  0.6077856  0.6073314
##   0.0002781641  0.6410664  0.6077856  0.6073314
##   0.0003477051  0.6410664  0.6077856  0.6073314
##   0.0004636069  0.6395609  0.6077856  0.6067449
##   0.0013908206  0.6456062  0.6119327  0.6219941
##   0.0018544274  0.6410840  0.6147105  0.6225806
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.001390821.
```

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

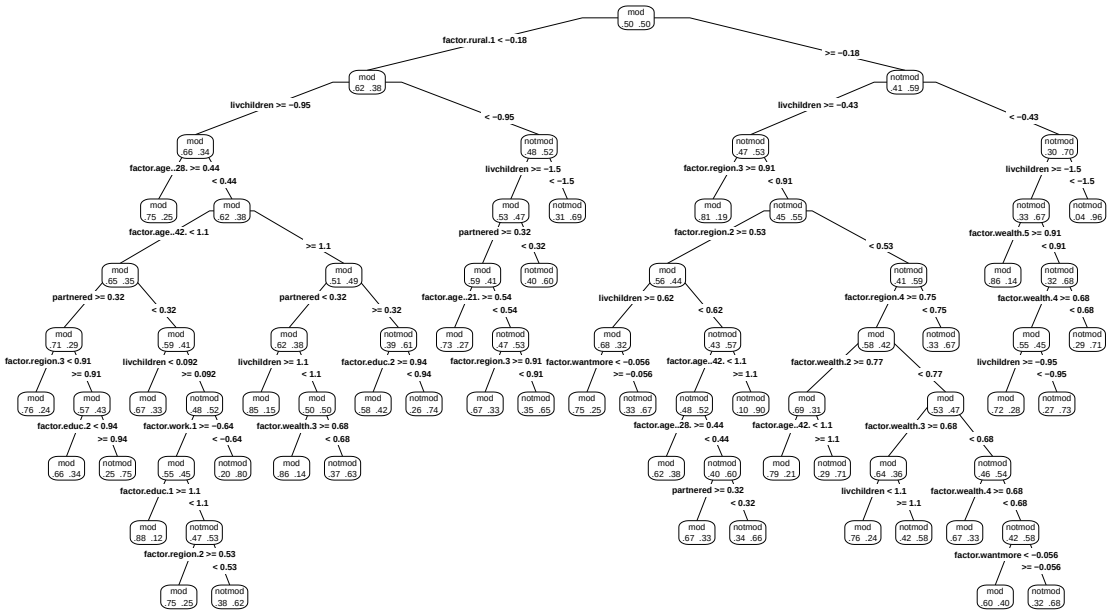
```
## Loading required package: rpart
```

```
plot(rp1)
```




```
#plot(rp1$finalModel)
prp(rp1$finalModel,type=4, extra = 4,
    main="Classification tree for using modern contraception")
```

Classification tree for using modern contraception

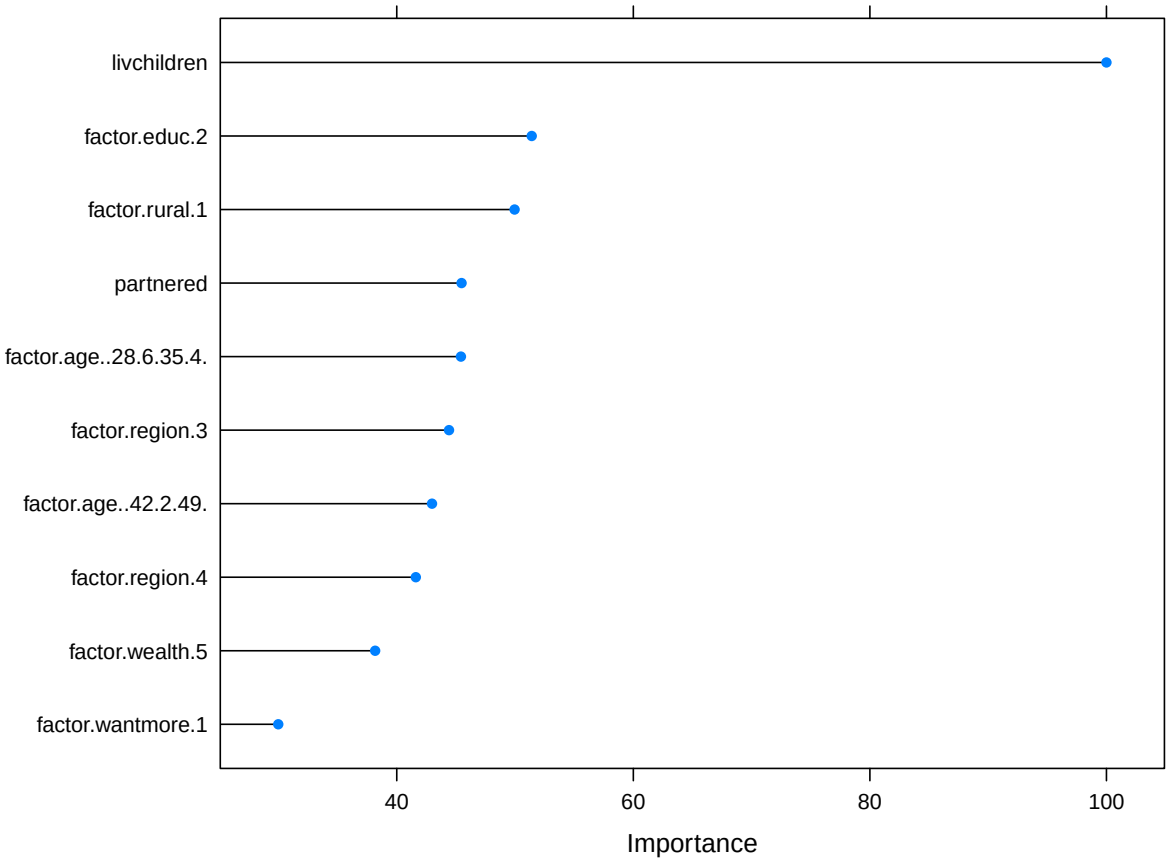



```
varImp(rp1)
```

```
## rpart variable importance
##
##                       Overall
## livchildren           100.000
## factor.educ.2          51.409
## factor.rural.1         49.962
## partnered              45.479
## factor.age..28.6.35.4. 45.425
## factor.region.3        44.418
## factor.age..42.2.49.   42.981
## factor.region.4        41.612
## factor.wealth.5        38.174
## factor.wantmore.1      29.981
## factor.region.2        28.628
## factor.wealth.4        22.629
## factor.age..21.8.28.6. 18.083
## factor.work.1          17.900
## factor.wealth.3        12.914
## factor.age..35.4.42.2.  9.608
## factor.educ.1           5.464
## factor.wealth.2         2.217
```

```
## factor.educ.3            0.000
```

```
plot(varImp(rp1), top=10)
```




```
##Accuracy on training set
pred1<-predict(rp1, newdata=x)
confusionMatrix(data = pred1,reference = y, positive = "mod" )
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  mod notmod
##     mod     479   1113
##     notmod  240   2297
##
##                Accuracy : 0.6723
##                  95% CI : (0.6578, 0.6866)
##     No Information Rate : 0.8259
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2297
##
##  Mcnemar's Test P-Value : <2e-16
```

```
##
##             Sensitivity : 0.6662
##             Specificity : 0.6736
##          Pos Pred Value : 0.3009
##          Neg Pred Value : 0.9054
##              Prevalence : 0.1741
##          Detection Rate : 0.1160
##    Detection Prevalence : 0.3856
##       Balanced Accuracy : 0.6699
##
##        'Positive' Class : mod
##
```

```
predt1<-predict(rp1, newdata=xtest)
confusionMatrix(data = predt1, yt, positive = "mod" )
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction mod notmod
##     mod    129    271
##     notmod  88    544
##
##                Accuracy : 0.6521
##                  95% CI : (0.6222, 0.6812)
##     No Information Rate : 0.7897
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2001
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.5945
##             Specificity : 0.6675
##          Pos Pred Value : 0.3225
##          Neg Pred Value : 0.8608
##              Prevalence : 0.2103
##          Detection Rate : 0.1250
##    Detection Prevalence : 0.3876
##       Balanced Accuracy : 0.6310
##
##        'Positive' Class : mod
##
```

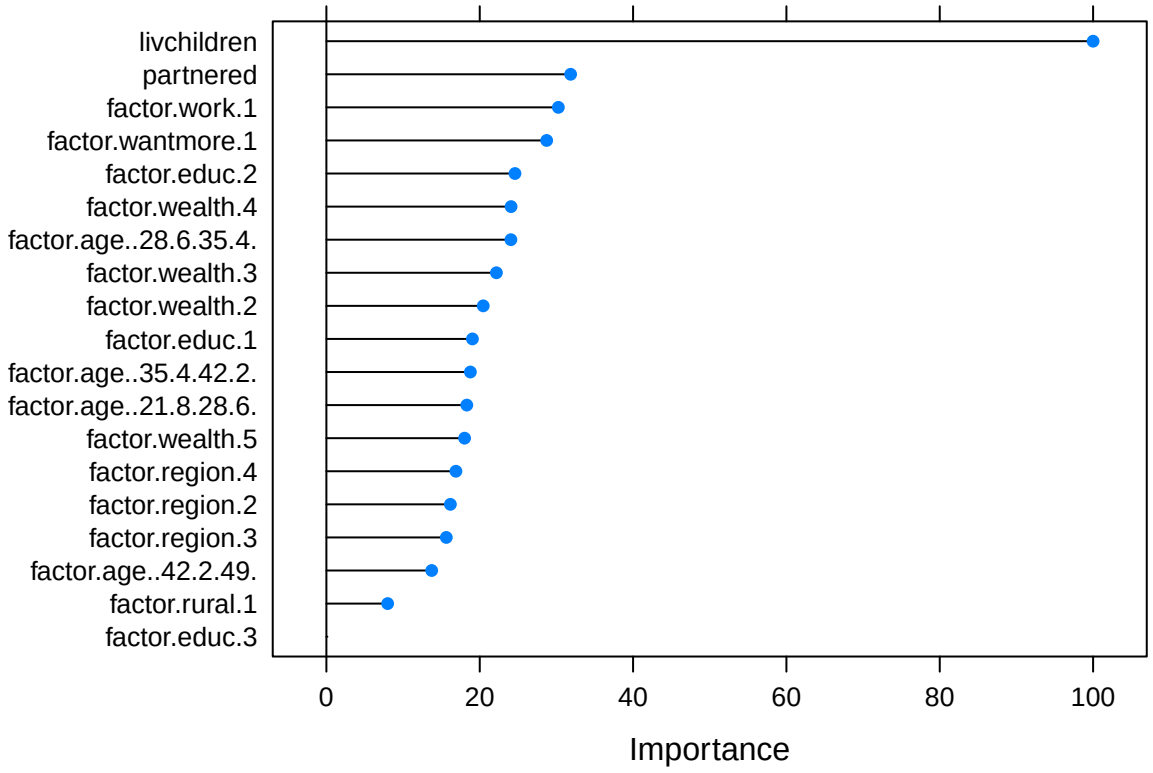
Bagged tree model

```
fitctrl <- trainControl(method="cv",
                        number=10,
                        #repeats=5,
                        classProbs = TRUE,
                       search="random",
```

```
                     sampling = "down",
                     summaryFunction=twoClassSummary,
                     savePredictions = "all")

bt1<-caret::train(y=y, x=x,
           metric="ROC",
           method ="treebag",
          tuneLength=20, #try 20 random values of the tuning parameters
           trControl=fitctrl,
          preProcess=c("center", "scale"))

print(bt1)
```

```
## Bagged CART
##
## 4129 samples
##   19 predictor
##    2 classes: 'mod', 'notmod'
##
## Pre-processing: centered (19), scaled (19)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 3717, 3716, 3716, 3716, 3716, 3716, ...
## Addtional sampling using down-sampling prior to pre-processing
##
## Resampling results:
##
##   ROC        Sens       Spec
##   0.6242268  0.6008803  0.5788856
```

```
plot(varImp(bt1))
```




```
##Accuracy on training set
pred1<-predict(bt1, newdata=x)
confusionMatrix(data = pred1,reference = y, positive = "mod" )
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  mod notmod
##     mod     654   1202
##     notmod   65   2208
##
##                Accuracy : 0.6931
##                  95% CI : (0.6788, 0.7072)
##     No Information Rate : 0.8259
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3431
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9096
##             Specificity : 0.6475
##          Pos Pred Value : 0.3524
##          Neg Pred Value : 0.9714
##              Prevalence : 0.1741
```

```
##          Detection Rate : 0.1584
##    Detection Prevalence : 0.4495
##       Balanced Accuracy : 0.7786
##
##        'Positive' Class : mod
##
```

```
predt1<-predict(bt1, newdata=xtest)
confusionMatrix(data = predt1,yt, positive = "mod" )
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction mod notmod
##     mod    132    360
##     notmod  85    455
##
##                Accuracy : 0.5688
##                  95% CI : (0.5379, 0.5993)
##     No Information Rate : 0.7897
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1137
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.6083
##             Specificity : 0.5583
##          Pos Pred Value : 0.2683
##          Neg Pred Value : 0.8426
##              Prevalence : 0.2103
##          Detection Rate : 0.1279
##    Detection Prevalence : 0.4767
##       Balanced Accuracy : 0.5833
##
##        'Positive' Class : mod
##
```

Random forest model using caret

```
library(rpart)
rf1<-caret::train(y=y, x=x,
           data=dtrain,
           metric="ROC",
           method ="rf",
          tuneLength=20, #try 20 random values of the tuning parameters
           trControl=fitctrl,
          preProcess=c("center", "scale"))

rf1
```

```
## Random Forest
```

```
##
## 4129 samples
##   19 predictor
##    2 classes: 'mod', 'notmod'
##
## Pre-processing: centered (19), scaled (19)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 3716, 3716, 3716, 3716, 3716, 3717, ...
## Addtional sampling using down-sampling prior to pre-processing
##
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##    1    0.6743208  0.5313576  0.7149560
##    2    0.6803158  0.5912559  0.6744868
##    4    0.6665729  0.6260172  0.6258065
##    5    0.6596331  0.6259977  0.6067449
##    6    0.6433240  0.5981808  0.5879765
##    7    0.6491962  0.6218310  0.5976540
##   11    0.6321789  0.6300861  0.5656891
##   12    0.6262939  0.6357394  0.5674487
##   13    0.6270354  0.6092332  0.5651026
##   14    0.6376019  0.6301056  0.5777126
##   15    0.6182209  0.6301056  0.5501466
##   18    0.6354484  0.6370892  0.5633431
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
##Accuracy on training set
predrf1<-predict(rf1, newdata=x)
confusionMatrix(data = predrf1,y, positive = "mod" )
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  mod notmod
##     mod     474   1046
##     notmod  245   2364
##
##                Accuracy : 0.6873
##                  95% CI : (0.6729, 0.7015)
##     No Information Rate : 0.8259
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.2449
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.6592
##             Specificity : 0.6933
##          Pos Pred Value : 0.3118
##          Neg Pred Value : 0.9061
##              Prevalence : 0.1741
```

```
##          Detection Rate : 0.1148
##    Detection Prevalence : 0.3681
##       Balanced Accuracy : 0.6763
##
##        'Positive' Class : mod
##
```

```
predgl1<-predict(rf1, newdata=xtest)
confusionMatrix(data = predgl1,yt, positive = "mod" )
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction mod notmod
##     mod    126    284
##     notmod  91    531
##
##                Accuracy : 0.6366
##                  95% CI : (0.6064, 0.666)
##     No Information Rate : 0.7897
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1751
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.5806
##             Specificity : 0.6515
##          Pos Pred Value : 0.3073
##          Neg Pred Value : 0.8537
##              Prevalence : 0.2103
##          Detection Rate : 0.1221
##    Detection Prevalence : 0.3973
##       Balanced Accuracy : 0.6161
##
##        'Positive' Class : mod
##
```

We see that by down sampling the more common level of the outcome, we end up with much more balanced accuracy in terms of specificity and sensitivity.

You see that the best fitting model is much more complicated than the previous one. Each node box displays the classification, the probability of each class at that node (i.e. the probability of the class conditioned on the node) and the percentage of observations used at that node. From here.

ROC curve

The ROC curve can be shown for the model:

```
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

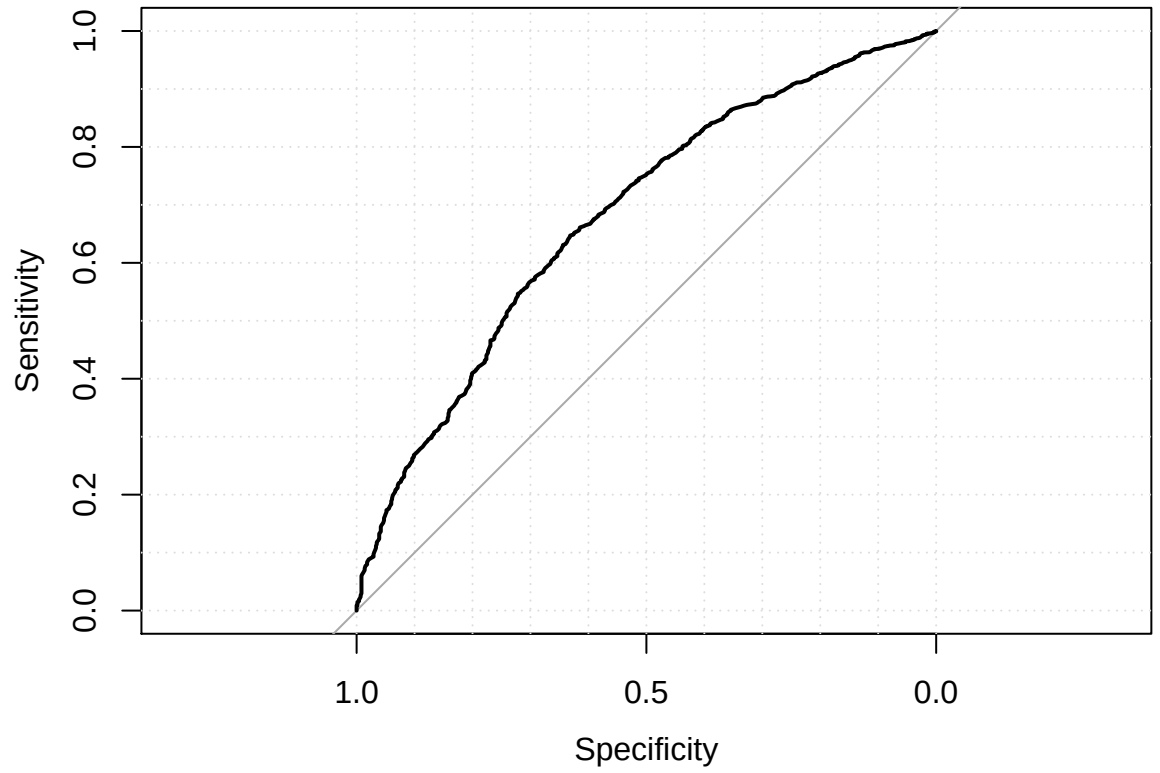
```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
# Select a parameter setting
mycp<-rf1$pred$mtry==rf1$bestTune$mtry
selectedIndices <- mycp==T
# Plot:
plot.roc(rf1$pred$obs[selectedIndices], rf1$pred$mod[selectedIndices], grid=T)
```

```
## Setting levels: control = mod, case = notmod
```

```
## Setting direction: controls > cases
```




```
#Value of ROC and AUC
roc(rf1$pred$obs[selectedIndices],  rf1$pred$mod[selectedIndices])
```

```
## Setting levels: control = mod, case = notmod
## Setting direction: controls > cases
```

```
##
## Call:
## roc.default(response = rf1$pred$obs[selectedIndices], predictor = rf1$pred$mod[selectedIndices])
##
## Data: rf1$pred$mod[selectedIndices] in 719 controls (rf1$pred$obs[selectedIndices] mod) > 3410 cases
## Area under the curve: 0.6794
```

```
auc(rf1$pred$obs[selectedIndices],  rf1$pred$mod[selectedIndices])
```

```
## Setting levels: control = mod, case = notmod
## Setting direction: controls > cases

## Area under the curve: 0.6794
```