R for Health Data Science

Week 10: Intro to Machine Learning - Classification Trees

Sam Stewart

2021-05-07

```
library(Hmisc)
library(kableExtra)
library(tidyverse)
library(ggplot2)
library(boomer)
library(XML)
library(car)
library(rpart)
library(rattle)
library(pROC)

#lot's of ML algorithms have randomness, so it's good practice to
#set the seed at the beginning of your work so it will produce consistent
#results
set.seed(11)

dat = read.csv("data/brfss2013-Connecticut.csv")

#variables under study
vars = c('bphigh4','cvdinfr4','cvdcrhd4','cvdstrk3','asthma3','chcscncr','chcocncr',
         'chccopd1','havarth3','addepev2','chckidny','diabete3','drvisits')

dat %>% tibble %>%
  select(vars)
```

```
## # A tibble: 7,710 x 13
##    bphigh4 cvdinfr4 cvdcrhd4 cvdstrk3 asthma3 chcscncr chcocncr chccopd1
##      <int>    <int>    <int>    <int>   <int>    <int>    <int>    <int>
##  1       1        2        2        2       2        2        2        2
##  2       1        2        7        2       2        2        2        2
##  3       1        2        2        2       2        1        2        2
##  4       3        2        2        2       2        2        2        2
##  5       3        2        2        2       2        1        2        2
##  6       3        2        2        2       2        2        2        2
##  7       1        7        2        2       2        2        2        2
##  8       1        2        2        2       2        1        2        2
##  9       3        2        2        2       2        2        2        2
## 10       3        2        2        2       2        2        2        2
## # ... with 7,700 more rows, and 5 more variables: havarth3 <int>,
## #   addepev2 <int>, chckidny <int>, diabete3 <int>, drvisits <int>
```

```r
#dat02 is a new version of the dataset
dat02 = data.frame(ID=rownames(dat))[,-1]

#car::recode is useful for collapsing levels in a factor
#we need the car:: part because dplyr also has a recode function, but it's mor verbose
dat02$bphigh = car::recode(dat$bphigh4, recodes="1=1;2:4=0;else=NA")
dat02$ha = car::recode(dat$cvdinfr4,recodes="1=1;2=0;else=NA")
dat02$chd = car::recode(dat$cvdcrhd4,recodes="1=1;2=0;else=NA")
dat02$stroke = car::recode(dat$cvdstrk3,recodes="1=1;2=0;else=NA")
dat02$asthma = car::recode(dat$asthma3,recodes="1=1;2=0;else=NA")
dat02$skinCa = car::recode(dat$chcscncr,recodes="1=1;2=0;else=NA")
dat02$otherCa = car::recode(dat$chcocncr,recodes="1=1;2=0;else=NA")
dat02$copd = car::recode(dat$chccopd1,recodes="1=1;2=0;else=NA")
dat02$arthritis = car::recode(dat$havarth3,recodes="1=1;2=0;else=NA")
dat02$depression = car::recode(dat$addepev2,recodes="1=1;2=0;else=NA")
dat02$kidney = car::recode(dat$chckidny,recodes="1=1;2=0;else=NA")
dat02$diabetes = car::recode(dat$diabete3,recodes="1=1;2:4=0;else=NA")
#drvisits is continuous so it doesn't need to be recoded
dat02$drvisits = dat$drvisits
dat02$drvisits[which(dat02$drvisits==88)] = 0
dat02$drvisits[which(dat02$drvisits>76)] = NA

#converting to factors in a new dataset
dat03 = do.call(cbind.data.frame,lapply(dat02,factor,levels=1:0,labels=c("Yes","No")))
dat03$drvisits = dat02$drvisits

#dropping missing values from each dataset
dat03 = na.omit(dat03)
dat02 = na.omit(dat02)

#dat02 is the numeric version of the dataset,
#dat03 is the factor version
```

I've changed our plan for the last lecture - I was going to talk about XML and JSON data, but they're a bit too complex from a programming perspective to explore at this point. If you have a need for working with XML data then let me know and I can share some sample code to get you through.

Instead I want to broach the topic of Machine Learning. R is one of the two premier languages being used in the ML world (the other major choice being Python). As a brief introduction I want to talk about how to create and share classification trees - I've included a tutorial on how classification trees work from a course on Data Mining I taught, take a look through if you're more interested. In this tutorial we'll talk about

1. Training vs Test data
2. Building CART
3. Presenting CART
4. Evaluating CART

1 Training and Test Data

Classification is about trying to accurately predict an outcome from a set of predictors. Unlike our regression methods, in classification we're primarily interested in the accuracy of the prediction - in statistics we tend to be more focused on the *mechanism* of prediction, or the regression coefficients.

In order to have an unbiased evaluation of the model we should evaluate it on data that it was not trained on - evaluating the quality of a prediction based on the data it was trained on would result in *over-fitting*, or a

model that is very good for that specific data and very poor outside of it.

There are many complex methods of creating a training/test split (cross-validation, bootstrapping, bagging and boosting), but I'm going to focus on a simple 70/30 split of the data. 70% of the observations will be in the training data, and 30% in the test.

```
#we'll focus on the factored data
n=dim(dat03)[1]
nSplit = round(n*0.7,0)
train = dat03[1:nSplit,]
test = dat03[(nSplit+1):n,]
```

In our model building we'll fit on the `train` dataset and then build on the `test` dataset.

The data we're working with in this lecture is the Behavioral Risk Factor Surveillance System (BRFSS) - "America's premier system of health-related telephone surveys". We'll be focusing on the Connecticut subset, and will be predicting high blood-pressure (`bphigh`) with several predictors. See the data dictionaries here for more details of the variables: https://www.cdc.gov/brfss/annual_data/annual_2013.html.

2 RPART

The traditional name for trees is *Classification and Regression Trees*, or CART. The original CART algorithm was copyrighted, so some researchers re-created the algorithm, and implemented it in R as RPART, or *Recursive Partitioning and Regression Trees*.

The idea is to create a decision tree from a flowchart like structure. All the observations start at the top, and then are split into child nodes based on a decision rule. Splitting continues until an endpoint (leaf) is reached, at which point a label is assigned in some manner.

Some important components of the algorithm that I won't cover in detail

- How do you decide what splits to consider?
- How do you choose the best split?
- How do you decide to stop splitting?

I'll focus more on the function itself, and direct you toward the important parts for building a tree.

```
rpart(formula, data, weights, subset, na.action = na.rpart, method,
      model = FALSE, x = FALSE, y = TRUE, parms, control, cost, ...)
```

The format is much like in regression - a formula that defines the model, and a dataset.

```
###a simple classification tree
mod.cart01 = rpart(bphigh~.,data=train)
mod.cart01
```

```
## n= 4943
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 4943 1886 No (0.3815497 0.6184503)
##    2) arthritis=Yes 1472  618 Yes (0.5801630 0.4198370) *
##    3) arthritis=No 3471 1032 No (0.2973207 0.7026793)
##      6) diabetes=Yes 256   90 Yes (0.6484375 0.3515625) *
##      7) diabetes=No 3215  866 No (0.2693624 0.7306376)
##       14) chd=Yes 86   22 Yes (0.7441860 0.2558140) *
##       15) chd=No 3129  802 No (0.2563119 0.7436881) *
```

```
summary(mod.cart01)
```

```
## Call:
## rpart(formula = bphigh ~ ., data = train)
##   n= 4943
##
##           CP nsplit rel error    xerror       xstd
## 1 0.12513256      0 1.0000000 1.0000000 0.01810846
## 2 0.04029692      1 0.8748674 0.8748674 0.01757927
## 3 0.02226935      2 0.8345705 0.8345705 0.01736664
## 4 0.01000000      3 0.8123012 0.8123012 0.01723984
##
## Variable importance
## arthritis  diabetes       chd  drvisits      copd        ha    stroke    kidney
##        57        24        14         1         1         1         1         1
##
## Node number 1: 4943 observations,    complexity param=0.1251326
##   predicted class=No   expected loss=0.3815497  P(node) =1
##     class counts:  1886  3057
##    probabilities: 0.382 0.618
##   left son=2 (1472 obs) right son=3 (3471 obs)
##   Primary splits:
##       arthritis splits as  LR,       improve=165.38300, (0 missing)
##       diabetes  splits as  LR,       improve=134.85450, (0 missing)
##       chd       splits as  LR,       improve= 95.19580, (0 missing)
##       drvisits  < 2.5  to the right, improve= 78.04115, (0 missing)
##       ha        splits as  LR,       improve= 63.06548, (0 missing)
##   Surrogate splits:
##       drvisits < 16.5 to the right, agree=0.709, adj=0.024, (0 split)
##       copd     splits as  LR,       agree=0.709, adj=0.022, (0 split)
##       ha       splits as  LR,       agree=0.707, adj=0.018, (0 split)
##       stroke   splits as  LR,       agree=0.705, adj=0.011, (0 split)
##       kidney   splits as  LR,       agree=0.705, adj=0.010, (0 split)
##
## Node number 2: 1472 observations
##   predicted class=Yes  expected loss=0.419837  P(node) =0.2977949
##     class counts:   854   618
##    probabilities: 0.580 0.420
##
## Node number 3: 3471 observations,    complexity param=0.04029692
##   predicted class=No   expected loss=0.2973207  P(node) =0.7022051
##     class counts:  1032  2439
##    probabilities: 0.297 0.703
##   left son=6 (256 obs) right son=7 (3215 obs)
##   Primary splits:
##       diabetes splits as  LR,       improve=68.14703, (0 missing)
##       chd      splits as  LR,       improve=55.78990, (0 missing)
##       drvisits < 1.5  to the right, improve=34.42429, (0 missing)
##       ha       splits as  LR,       improve=33.35102, (0 missing)
##       stroke   splits as  LR,       improve=24.84136, (0 missing)
##
## Node number 6: 256 observations
##   predicted class=Yes  expected loss=0.3515625  P(node) =0.05179041
##     class counts:   166    90
```

```
##    probabilities: 0.648 0.352
##
## Node number 7: 3215 observations,    complexity param=0.02226935
##   predicted class=No   expected loss=0.2693624  P(node) =0.6504147
##     class counts:   866  2349
##    probabilities: 0.269 0.731
##   left son=14 (86 obs) right son=15 (3129 obs)
##   Primary splits:
##       chd      splits as  LR,       improve=39.84452, (0 missing)
##       ha       splits as  LR,       improve=27.31362, (0 missing)
##       drvisits < 1.5  to the right, improve=24.52311, (0 missing)
##       stroke   splits as  LR,       improve=18.31978, (0 missing)
##       copd     splits as  LR,       improve=11.97701, (0 missing)
##   Surrogate splits:
##       ha splits as  LR, agree=0.974, adj=0.012, (0 split)
##
## Node number 14: 86 observations
##   predicted class=Yes  expected loss=0.255814  P(node) =0.01739834
##     class counts:    64    22
##    probabilities: 0.744 0.256
##
## Node number 15: 3129 observations
##   predicted class=No   expected loss=0.2563119  P(node) =0.6330164
##     class counts:   802  2327
##    probabilities: 0.256 0.744
```

You can see that the simple summary is a summary of the tree, and the summary gives a detailed description of each node, including all the splits it considered. We'll look later at plotting.

The important tree controls are passed as a list to the `control` argument.

- `cp` is the "complexity parameter" - it measures how much improvement is needed for a split to be included in the model.
- `minbucket` controls how small a node can be to be created
- `minsplit` controls how small a node can be to be considered for a split

```
#trying to build as large a tree as possible
mod.cart02 = rpart(bphigh~.,data=train,control=list(cp=0.0,minbucket=0,minsplit=0))
mod.cart03 = rpart(bphigh~.,data=train,control=list(cp=0.0041,minbucket=1,minsplit=2))
```

2.1 Getting Optimal Trees

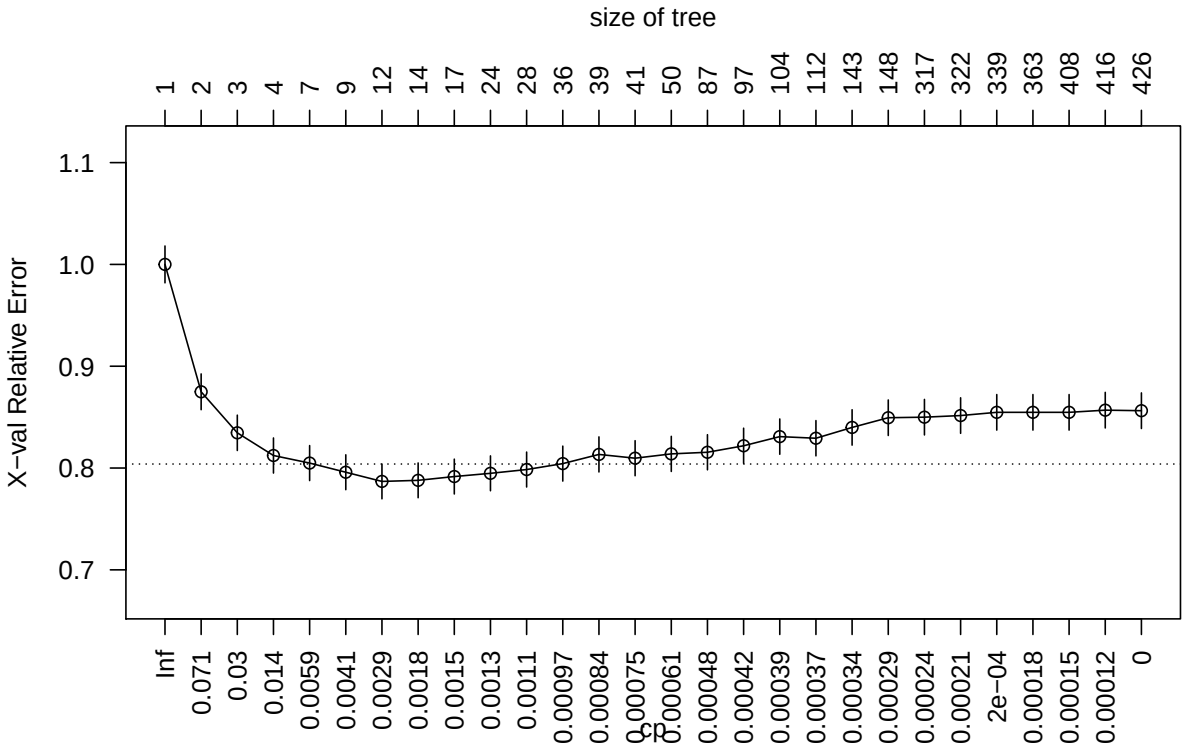
```
#getting optimal CP values
printcp(mod.cart02)
```

```
##
## Classification tree:
## rpart(formula = bphigh ~ ., data = train, control = list(cp = 0,
##     minbucket = 0, minsplit = 0))
##
## Variables actually used in tree construction:
##  [1] arthritis  asthma     chd        copd       depression diabetes
##  [7] drvisits   ha         kidney     otherCa    skinCa     stroke
##
## Root node error: 1886/4943 = 0.38155
##
```

```
## n= 4943
##
##            CP nsplit rel error  xerror     xstd
## 1  0.12513256      0   1.00000 1.00000 0.018108
## 2  0.04029692      1   0.87487 0.87487 0.017579
## 3  0.02226935      2   0.83457 0.83457 0.017367
## 4  0.00830682      3   0.81230 0.81230 0.017240
## 5  0.00424178      6   0.78738 0.80488 0.017196
## 6  0.00397667      8   0.77890 0.79586 0.017142
## 7  0.00212089     11   0.76617 0.78685 0.017087
## 8  0.00159067     13   0.76193 0.78791 0.017093
## 9  0.00132556     16   0.75716 0.79162 0.017116
## 10 0.00119300     23   0.74708 0.79480 0.017135
## 11 0.00106045     27   0.74231 0.79852 0.017158
## 12 0.00088370     35   0.73383 0.80435 0.017193
## 13 0.00079533     38   0.73118 0.81336 0.017246
## 14 0.00070696     40   0.72959 0.80965 0.017224
## 15 0.00053022     49   0.72322 0.81389 0.017249
## 16 0.00044185     86   0.70255 0.81548 0.017258
## 17 0.00039767     96   0.69724 0.82185 0.017295
## 18 0.00037873    103   0.69406 0.83086 0.017346
## 19 0.00035348    111   0.69088 0.82927 0.017337
## 20 0.00031813    142   0.67922 0.83987 0.017396
## 21 0.00026511    147   0.67762 0.84942 0.017447
## 22 0.00021209    316   0.62566 0.84995 0.017450
## 23 0.00019883    321   0.62460 0.85154 0.017459
## 24 0.00019281    338   0.62089 0.85472 0.017476
## 25 0.00017674    362   0.61506 0.85472 0.017476
## 26 0.00013256    407   0.60657 0.85472 0.017476
## 27 0.00010604    415   0.60551 0.85684 0.017487
## 28 0.00000000    425   0.60445 0.85631 0.017484
```

What this table shows is the size of the tree for decreasing values of `cp` - as the tolerance for improvement goes down, the tree size goes up. The column we care about is the `xerror` column - that's a cross-validated measure of error that we want to be as small as possible. The plot below provides a visual representation of that column.

```
plotcp(mod.cart02,las=2)
```



What we do is find the lowest `xerror` value, then calculate a CI around it. The upper bound of that CI is the lowest reasonable `xerror` we should expect, so we pick the smallest tree that meets that value - in this case it's at a CP of 0.0041.

Once we've identified the appropriate CP value we can either re-fit the tree with the appropriate CP value, or we can *prune* the tree by removing splits below that CP threshold.

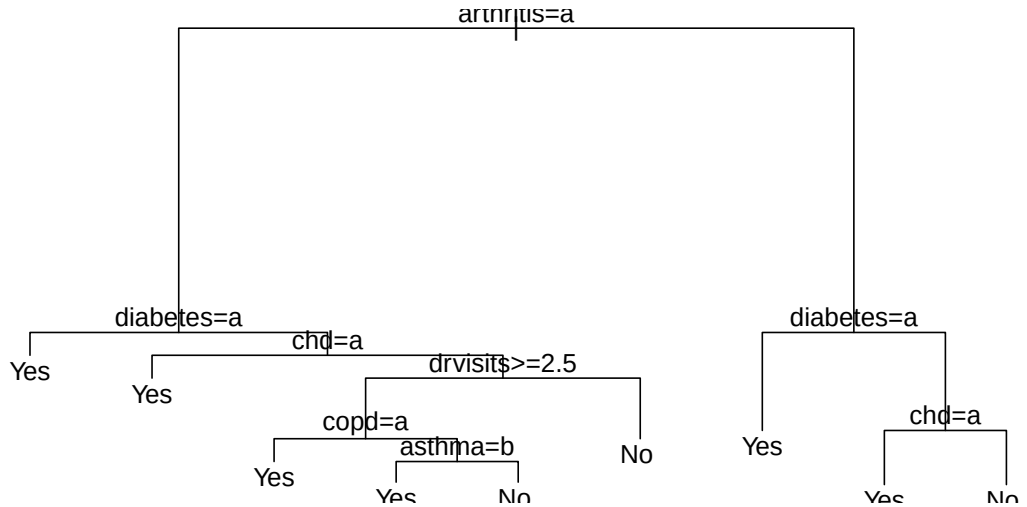
```
#pruning trees is done using prune
mod.cart02a = prune(mod.cart02,cp=0.0005)
mod.cart02b = prune(mod.cart02,cp=0.004)#this should be the same mod.cart03

mod.cart01 = mod.cart02b
```

2.2 Plotting

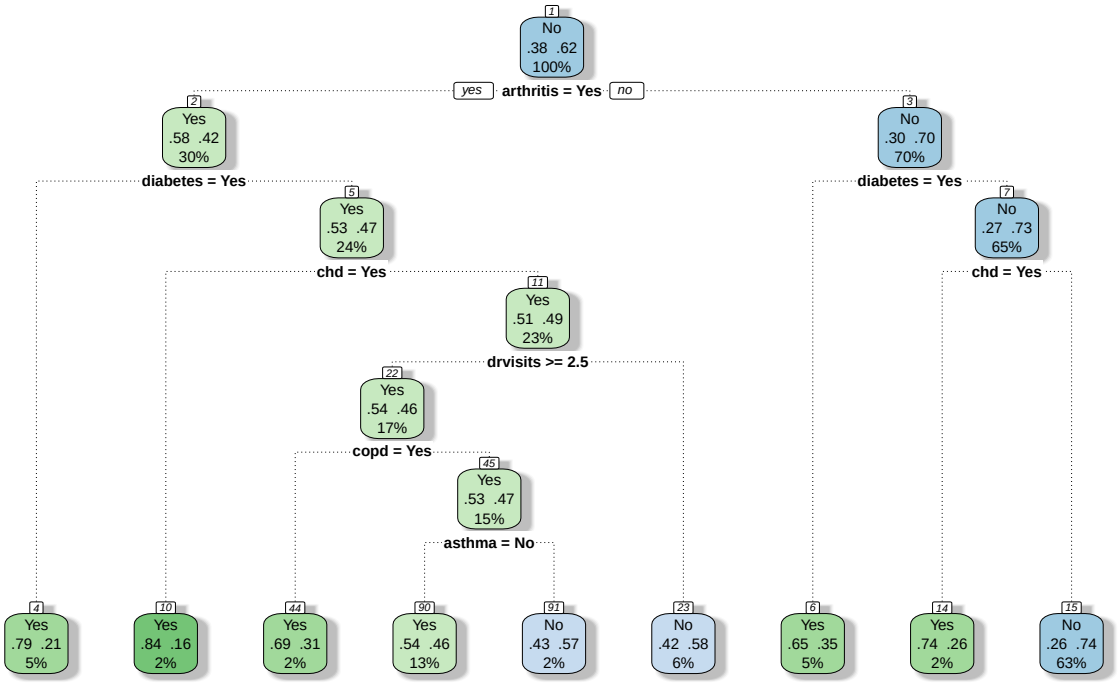
The best thing about CART over other ML algorithms is the ability to understand the decision making process - in that sense it falls somewhere between the explanatory nature of regression and the purely classification goals of Neural Networks and Deep Learning.

Unfortunately the default plotting function for `rpart` objects is terrible, as you can see below. Luckily there is a function `rattle::fancyRpartPlot()` to create usable tree plots.

```
#plotting the trees is best through the rattle library
plot(mod.cart01)#boo :(
text(mod.cart01)
```




```
fancyRpartPlot(mod.cart01)#yay! :)
```



There are three lines to each node by default:

1. The classification of the objects at the node
2. The split between classes at the node
3. The percentage of subjects that reach that node

2.3 Evaluation

To evaluate the success of the model we'll run the test data through it, then compare the predicted results to the actual results. `predict()` will be used to produce the predicted results.

```
#predicting new values from the model
#default is probabilities
pred.cart01 = predict(mod.cart01,newdata=test)
head(pred.cart01,10)
```

```
##            Yes        No
## 5421 0.2563119 0.7436881
## 5422 0.2563119 0.7436881
## 5423 0.2563119 0.7436881
## 5424 0.2563119 0.7436881
## 5426 0.2563119 0.7436881
## 5428 0.2563119 0.7436881
## 5429 0.2563119 0.7436881
## 5430 0.2563119 0.7436881
## 5432 0.2563119 0.7436881
## 5433 0.2563119 0.7436881
```

```
pred.cart01 = pred.cart01[,1]#Just keep the Y probabilities
```

You can adjust what is predicted with the `type` argument, but probabilities are the default and are usually the most useful. Since `rpart` can predict multi-level outcomes it produces a prediction for all classes, but in this case we just need the first column.

2.3.1 ROC Curve

Once we have the predicted probabilities we'll evaluate them using a **R**eceiver **O**perator **C**haracteristic curve. There are two libraries that are commonly used to produce ROC curves in R: `ROCR` and `pROC`. I'll look at `pROC` here, but `ROCR` works fine as well.

```
roc.cart01 = roc(test$bphigh,pred.cart01,ci=TRUE,levels=c("Yes","No"))
roc.cart01
```

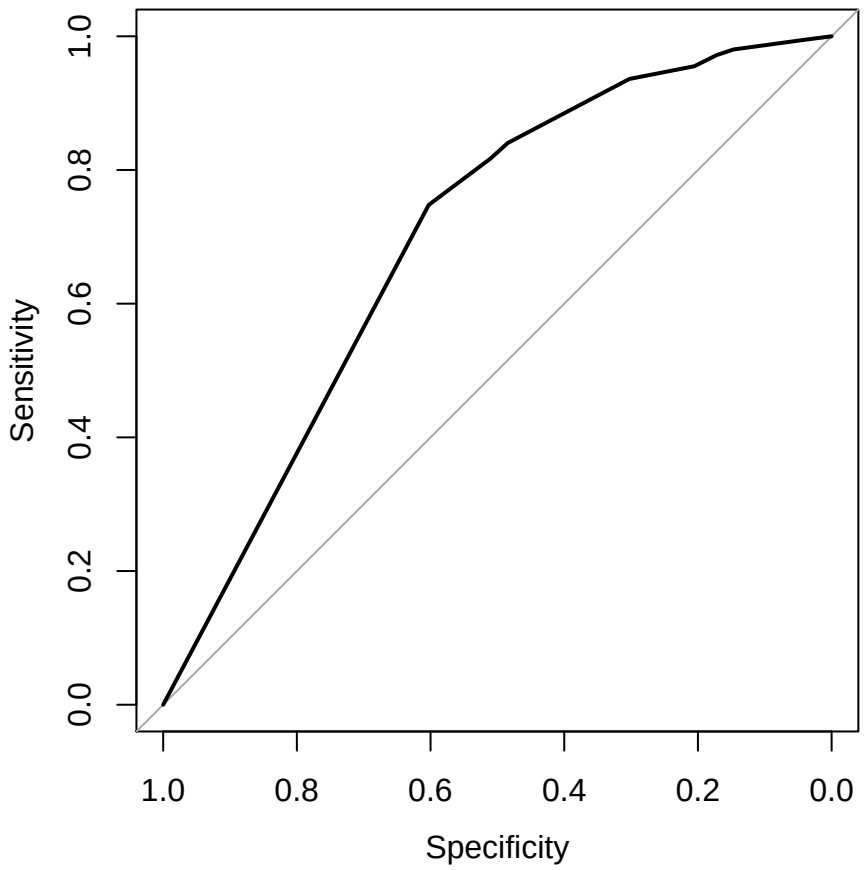
```
##
## Call:
## roc.default(response = test$bphigh, predictor = pred.cart01,     levels = c("Yes", "No"), ci = TRUE)
##
## Data: pred.cart01 in 803 controls (test$bphigh Yes) > 1316 cases (test$bphigh No).
## Area under the curve: 0.6979
## 95% CI: 0.6765-0.7192 (DeLong)
```

```
roc.dat01 = coords(roc.cart01)
roc.dat01
```

```
##    threshold specificity sensitivity
## 1        Inf  0.00000000   1.0000000
## 2  0.8150599  0.04109589   0.9946809
## 3  0.7694314  0.14694894   0.9802432
## 4  0.7178377  0.17185554   0.9718845
## 5  0.6699634  0.20547945   0.9551672
```

```
## 6  0.5960788  0.30261519   0.9361702
## 7  0.4879618  0.48443337   0.8404255
## 8  0.4254443  0.51058531   0.8168693
## 9  0.3374985  0.60273973   0.7477204
## 10      -Inf  1.00000000   0.0000000
```

```
plot(roc.cart01)
```



One nice thing about the `pROC` plot is that it forces the plotting space to be square - ROC curves should always be presented with the x- and y- axes the same length.

We need to convert our predicted probabilities to Y/N values so we can evaluate the model. To do that we'll use the 'best' argument in the `coords()` function to extract the optimal cutpoint - the default is to choose the highest *sens+spec* value, but there are many approaches to this problem.

```
opt.cart01 = coords(roc.cart01,x='best',transpose=TRUE)
pred.cart01.value = pred.cart01 >= opt.cart01[1]
```

Once we have the predicted Y/N values we can compare them to the actual Y/N values in a 2x2 table, and then calculate a variety of accuracy measures on that table. I've included a custom function here that I wrote a while ago called `getAccuracy()` that is a *very fragile* way to get the sens/spec and other numbers - make very sure your table is oriented the right way for the function to work (see the comment).

```
#a function to get the accuracy of the model from the predicted labels, should be of the form
#           pred=Y  pred=N
#label  Yes      a       b
```

```
#        No      c       d
getAccuracy = function(tab){
    if(any(c(colnames(tab)[1],rownames(tab)[1])%in%c("No",0,FALSE)))
        warning("Are you sure that your table is set up the right way?")
    TP = tab[1,1]
    FP = tab[2,1]
    FN = tab[1,2]
    TN = tab[2,2]
    P = rowSums(tab)[1]
    N = rowSums(tab)[2]
    acc=(TP+TN)/(P+N)
    errRate=(FP+FN)/(P+N)
    sens=TP/P
    spec=TN/N
    prec=TP/(TP+FP)
    F1=2*prec*sens/(prec+sens)
    F2=(1+2^2)*prec*sens/(2^2*prec+sens)
    F0.5=(1+0.5^2)*prec*sens/(0.5^2*prec+sens)
    out=c(acc,errRate,sens,spec,prec,F1,F2,F0.5)
    names(out) = c('acc','errRate','sens','spec','prec','F1','F2','F0.5')
    out
}

#Note that I have to flip the columns to make the diagonal line up right
tab.cart01 = table(test$bphigh,pred.cart01.value)[,2:1]
tab.cart01
```

```
##      pred.cart01.value
##       TRUE FALSE
##   Yes  484   319
##   No   332   984
```

```
getAccuracy(tab.cart01)
```

```
##       acc   errRate      sens      spec      prec        F1        F2      F0.5
## 0.6927796 0.3072204 0.6027397 0.7477204 0.5931373 0.5978999 0.6007944 0.5950332
```

3 Breakout Activity

We're going back to the Framingham data to see how well we can predict smoking status. Build a CART model predicting smoking status - use whatever variables you see fit. I'll start everyone off with the same training/test split.

```
fram = read.csv("data/framinghamFirst.csv",header=TRUE,
                na.strings=".",stringsAsFactors=FALSE)
fram = fram %>%
  select(-CIGPDAY) #we obviously need to drop this

#any data formatting you should do here first
set.seed(42)
n=dim(fram)[1]
nSplit = round(n*0.7,0)
train.fram = fram[1:nSplit,]
test.fram = fram[(nSplit+1):n,]
```