Classification and Regression Trees

Cavan Reilly

October 16, 2019

Table of contents

Classification

A common problem is trying to use a set of variables to predict another variable.

For example, if the variable one is trying to predict is continuous, one can use a linear regression model.

However if there are many potential predictor variables one may not be able to fit the regression model as one needs more observations than predictor variables.

If this condition is not met there is no way to find a unique regression model: it is like trying to find a line that goes through a point in 2 dimensional space.

Classification

There are many methods that can be used to classify subjects.

Here we focus on classification and regression trees.

Classification trees are applied to categorical traits.

Regression trees are applied to continuous traits.

Both methods recursively break the data into smaller homogeneous groups based on the predictor variables.

Building a tree

One begins constructing a tree by determining which of the predictors is the most predictive of the trait.

Here our predictors will be genetic variants, so if we find that the presence of a particular allele at a SNP is the most predictive of our variables, we break the data up into 2 groups so that all subjects in one group have at least one copy of this particular allele and the other group has zero copies.

This process is then repeated within the 2 resulting groups.

The partitioning stops when we encounter a *stopping rule*, such as there are less than 5 subjects in some subgroup.

Building a tree

Trees are conveniently displayed with tree-like diagrams where a node represents a group of subjects.

The top node is called the *root node*.

Under the root node are the *daughter nodes*, which we distinguish with the descriptors left and right.

Splitting rules

At each split, we must decide which variable to use for splitting.

This decision is made with reference to a measure of node impurity, or how much variation there is upon choosing a split.

Each split is made by maximizing the node impurity of the current node minus the node impurity of its 2 daughter nodes.

Frequently the resulting tree is usually insensitive to the choice of the measure of node impurity.

Node impurity: binary traits

First we will treat the case where the trait is binary, taking the values 0 and 1 (e.g. case and control).

A simple impurity measure is to just use

$$\min\big(\hat{p}(y=1), 1-\hat{p}(y=1)\big).$$

That is, we compute the proportion of cases in the node and if this exceeds 0.5 we subtract it from 1.

The minimum just makes the definition not depend on the labeling scheme we have used.

This is called the *Bayes error*, *minimum error* or the *misclassification cost*.

Node impurity: binary traits

This is intuitively appealing: if the root node has an equal number of cases and controls then the root node would have maximal node impurity, namely 0.5.

But if there was a variable that aligned perfectly with case and control status the 2 daughter nodes would both have impurity 0 which means the impurity of the root node less the impurity of its daughters would be 0.5 which is the largest possible value for this difference.

Hence we would split using any variable that perfectly aligns with case and control status.

However there are frequently ties so that there is no unique way of splitting up the data.

Node impurity: binary traits

The *Gini index* (also called the nearest neighbor error) is a commonly used impurity measure and is defined by

$$2\hat{p}(y=1)\big(1-\hat{p}(y=1)\big).$$

One problem with this measure is that in practice it often produces groups with very different sizes.

Another popular impurity measure is the *entropy function*, also called the *deviance* and the *information index* and is defined by

$$-\hat{p}(y=0)\log\big(\hat{p}(y=0)\big)-\hat{p}(y=1)\log\big(\hat{p}(y=1)\big).$$

This typically gives results that are similar to the Gini index.

Building trees in R

We will use the HIV mutation data to examine tree building.

We first extract all of the position data then construct a binary trait by comparing the fold change for IDV to NFV.

```
> attach(virco)
> VircoGeno <- data.frame(virco[,substr(names(virco),1,1)=="P"]!="-")
> Trait <- as.factor(IDV.Fold > NFV.Fold)
> library(rpart)
> ClassTree <- rpart(Trait~., method="class",
+   data=VircoGeno)
```

Building trees in R

We then examine the tree.

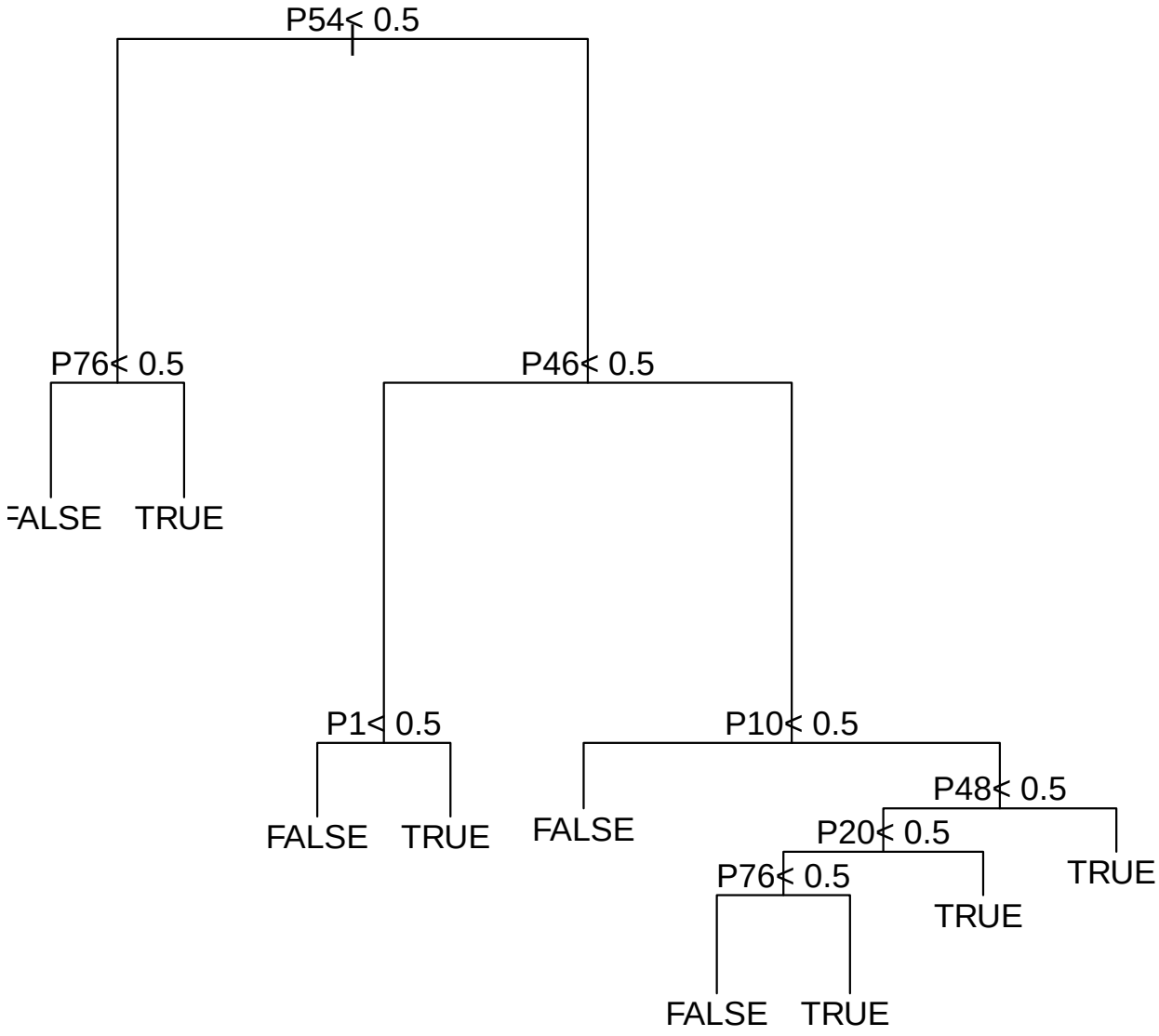
```
> ClassTree
n=976 (90 observations deleted due to missingness)
node), split, n, loss, yval, (yprob)
      * denotes terminal node
  1) root 976 399 FALSE (0.5911885 0.4088115)
    2) P54< 0.5 480 130 FALSE (0.7291667 0.2708333)
      4) P76< 0.5 466 116 FALSE (0.7510730 0.2489270) *
      5) P76>=0.5 14   0 TRUE (0.0000000 1.0000000) *
    3) P54>=0.5 496 227 TRUE (0.4576613 0.5423387)
      6) P46< 0.5 158  57 FALSE (0.6392405 0.3607595)
       12) P1< 0.5 115  31 FALSE (0.7304348 0.2695652) *
       13) P1>=0.5 43  17 TRUE (0.3953488 0.6046512) *
      7) P46>=0.5 338 126 TRUE (0.3727811 0.6272189)
       14) P10< 0.5 22   7 FALSE (0.6818182 0.3181818) *
       15) P10>=0.5 316 111 TRUE (0.3512658 0.6487342)
         30) P48< 0.5 278 106 TRUE (0.3812950 0.6187050)
           60) P20< 0.5 113  55 TRUE (0.4867257 0.5132743)
            120) P76< 0.5 92  40 FALSE (0.5652174 0.4347826) *
            121) P76>=0.5 21   3 TRUE (0.1428571 0.8571429) *
           61) P20>=0.5 165  51 TRUE (0.3090909 0.6909091) *
         31) P48>=0.5 38   5 TRUE (0.1315789 0.8684211) *
```

Building trees in R

We can also look at a plot of the tree

```
> plot(ClassTree)
> text(ClassTree)
```

However the text based representation is more informative.


P54< 0.5
P76< 0.5
P46< 0.5
FALSE
TRUE
P1< 0.5
P10< 0.5
FALSE
TRUE
FALSE
P48< 0.5
P20< 0.5
TRUE
P76< 0.5
TRUE
FALSE
TRUE

Building trees in R

For example, we can see that there are 480 subjects that are wildtype at position 54.

By the `loss` variable we see that of these 480 subjects, 130 do not have the majority trait, which is `FALSE` as indicated by the next value.

Finally, the values in parentheses are the estimates of the probabilities for being in either of the 2 groups.

Building trees in R

There are additional arguments to `rpart` that allow one to control the tree building process.

For example, by using the `parms` argument we can select a different measure of node impurity.

We can also use the `control` argument to specify a minimum number of observations in any terminal node (via `minbucket`) and the minimal number of observations in a node to consider splitting it.

For example in the following we use the entropy as our measure of node impurity.

Building trees in R

```
> rpart(Trait~., method="class", parms=list(split='information'),
+   data=VircoGeno)
n=976 (90 observations deleted due to missingness)
node), split, n, loss, yval, (yprob)
* denotes terminal node
 1) root 976 399 FALSE (0.5911885 0.4088115)
   2) P54< 0.5 480 130 FALSE (0.7291667 0.2708333)
     4) P76< 0.5 466 116 FALSE (0.7510730 0.2489270) *
     5) P76>=0.5 14   0 TRUE (0.0000000 1.0000000) *
   3) P54>=0.5 496 227 TRUE (0.4576613 0.5423387)
     6) P46< 0.5 158  57 FALSE (0.6392405 0.3607595)
      12) P1< 0.5 115  31 FALSE (0.7304348 0.2695652) *
      13) P1>=0.5 43  17 TRUE (0.3953488 0.6046512) *
     7) P46>=0.5 338 126 TRUE (0.3727811 0.6272189)
      14) P48< 0.5 299 120 TRUE (0.4013378 0.5986622)
        28) P20< 0.5 125  62 FALSE (0.5040000 0.4960000)
          56) P76< 0.5 104  44 FALSE (0.5769231 0.4230769) *
          57) P76>=0.5 21   3 TRUE (0.1428571 0.8571429) *
        29) P20>=0.5 174  57 TRUE (0.3275862 0.6724138) *
      15) P48>=0.5 39   6 TRUE (0.1538462 0.8461538) *
```

Regression trees

When the trait is quantitative, the most common measure of node impurity is simply the MLE of the variance within the node (i.e. we use n in the denominator rather the $n - 1$).

As usual, you would want to log transform the data if it is strongly skewed.

We can use the same R function in much the same, except we now specify `method="anova"`.

```
> Trait <- NFV.Fold - IDV.Fold
> Tree <- rpart(Trait~., method="anova",
+   data=VircoGeno)
```

Regression trees

```
> Tree
n=976 (90 observations deleted due to missingness)
node), split, n, deviance, yval
* denotes terminal node
 1) root 976 6437933.00   4.288320
   2) P54>=0.5 496 1247111.00  -3.916935
     4) P46>=0.5 338  343395.20 -10.567160 *
     5) P46< 0.5 158  856789.90  10.309490
      10) P58< 0.5 144  110944.10   2.570139 *
      11) P58>=0.5 14  648503.60  89.914290 *
   3) P54< 0.5 480 5122921.00  12.767080
     6) P73< 0.5 422  145579.90   5.706635 *
     7) P73>=0.5 58 4803244.00  64.137930
      14) P35< 0.5 45   26136.17   8.171111 *
      15) P35>=0.5 13 4148242.00 257.869200 *
```

Regression trees

Note that we get slightly different output.

The `yval` is the mean value of the trait within that node.

The `deviance` is the total sum of squares within the node, so if you divide this by the sample size in each node you get the variance.

Multicategory predictors

Thus far we have assumed that the predictor variables are binary which is useful when we think of indels.

However we typically think of SNPs as taking 3 values: the number of copies of the minor allele, which is 0,1, or 2.

In this case we can define the Gini index to be

$$\sum_{i \neq j} \hat{p}(y = i)\hat{p}(y = j).$$

The entropy also generalizes easily to

$$-\sum_{i} \hat{p}(y = i) \log\big(\hat{p}(y = i)\big).$$

Multicategory predictors

Sometimes it makes sense to treat the number of copies of the minor allele as an ordinal variable and other times it is more sensible to treat it as categorical.

If we know nothing about the action of a variant we may want to allow for any sort of association between the variant and the trait.

In this case we may want to consider every possible way of splitting the 3 levels:

- 0 vs. 1 and 2
- 0 and 1 vs. 2
- 0 and 2 vs. 1

Multicategory predictors

In other cases we may want to use an additive genetic model.

In that case you would only consider 2 possible splits: between 0 and 1 and between 1 and 2.

We treat continuous variables in the same manner as ordinal variables: sort the values and consider splits that occur between the ordered values.

Multicategory predictors in R

Here we consider an example involving the resistin gene and the percent change in arm strength.

```
> attach(fms)
> Trait <- NDRM.CH
> RegTree <- rpart(Trait~resistin_c30t+resistin_c398t+
+          resistin_g540a+resistin_c980g+resistin_c180g+
+          resistin_a537c, method="anova")
```

Multicategory predictors in R

Then here is the resulting tree.

```
> RegTree
n=611 (786 observations deleted due to missingness)
node), split, n, deviance, yval
      * denotes terminal node
1) root 611 665669.4 52.85352
  2) resistin_c980g=CC,CG 510 491113.4 51.23314 *
  3) resistin_c980g=GG 101 166455.3 61.03564 *
```

So being homozygous for G at `resistin_c980g` is associated with a greater increase in muscle strength.

Multicategory predictors in R

By using the `as.numeric` command on the genotype variables we automatically convert them to numerical values.

The value taken by a genotype will be determined by its alphabetical order (so nothing to do with number of minor alleles).

```
> RegTreeOr <- rpart(Trait~as.numeric(resistin_c30t)+
+   as.numeric(resistin_c398t)+as.numeric(resistin_g540a)+
+   as.numeric(resistin_c980g)+as.numeric(resistin_c180g)+
+   as.numeric(resistin_a537c), method="anova")
```

Multicategory predictors in R

```
> RegTreeOr
n=611 (786 observations deleted due to missingness)
node), split, n, deviance, yval
      * denotes terminal node
1) root 611 665669.4 52.85352
  2) as.numeric(resistin_c980g)< 2.5 510 491113.4 51.23314 *
  3) as.numeric(resistin_c980g)>=2.5 101 166455.3 61.03564 *
```

Covariates

Generally one wants to consider potential covariates due to the possibility of confounding and effect modification.

One simple way to deal with covariates is to just include them in the set of potential predictors.

The primary problem with this approach is that the tree may just use the covariates so that you don't learn anything about the association between your genetic loci and trait.

One alternative to this is to first stratify the data set using the potential confounders.

Covariates

The downside to stratification is that you lose power to identify associations due to the reduction in power.

Finally, one could fit a regression model using the trait and the covariates.

After fitting the model one can get the residuals from the model (using the `residuals` command) and use these as the trait data.

Statistical interaction and conditional associations

When statisticians speak of a statistical interaction we mean a model for the trait of the form

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i3} + \gamma x_{i1} x_{i2} + \epsilon_i.$$

Now suppose that x_{i1} is either 0 or 1 for all subjects and explains the most variability in y.

Statistical interaction and conditional associations

Then we have 2 models depending on the value of x_{i1}: if $x_{i1} = 0$

$$y_i = \beta_0 + \beta_2 x_{i2} + \beta_3 x_{i3} + \epsilon,$$

and if $x_{i1} = 1$

$$y_i = \beta_0 + \beta_1 + (\beta_2 + \gamma) x_{i2} + \beta_3 x_{i3} + \epsilon.$$

If $\beta_3 > \beta_2 + \gamma$ then after x_1 is used for splitting, x_3 will be selected as it has the stronger conditional association even though there is an interaction between x_1 and x_2.

Thus classification trees are not the best tool for finding statistical interactions, they uncover conditional associations.

Optimal trees

The trees we have examined so far likely overfit to the sample we have used for tree construction.

This is because we have not considered if the estimated tree is just fitting to random noise.

By chance there will some structure in any data set and if you search hard enough you will find some pattern.

However when you look at new samples this pattern will no longer hold.

Tree error rates

The error rate for an estimated tree is the sum over all terminal nodes of the error rates associated with these terminal nodes.

The error rate for a terminal node is the product of the probability of reaching that terminal node and the error rate at that terminal node.

If we just use the observed error rate using the same data we used to fit the tree we will underestimate the true error rate since the same data is used to build the tree and test the tree.

By the true error rate we mean the error rate one would obtain if one collected an entirely different sample and ran them through the estimated tree.

Cross validation

We can use tenfold cross-validation to get a better estimate of the true error rate (or more generally k-fold cross-validation).

To do this, first randomly split up the data set into 10 groups.

Then for each of the groups, set it aside, fit the tree, then use the excluded data to calculate the error rate.

Then average over all 10 error rates to get the cross-validation error rate.

Pruning via cost complexity

As noted previously, large trees are susceptible to overfitting.

One way to deal with this is to modify the measure of error to include a cost for introducing new nodes into the tree.

For every split made by a given tree, there is a cost so that the subtree that starts with that split is too expensive and should be eliminated.

If we order the splits with respect to this cost, then we will have a nested set of trees so that each of these trees is optimal if the cost takes a given value.

These costs at which the optimal tree changes are the *critical costs*.

Pruning via cost complexity

We then break up the data set into 10 groups, and for each group determine the optimal set of trees using the critical costs from the entire data set.

Then for each subset of the data we compute the error rate using the portion that was set aside. Do this for all subsets of data.

We then use the standard deviation of the errors to construct an interval for the error rate.

One suggested rule is to use the smallest model so that the error is not larger than that given by the best model (which is indicated by the horizontal line).

Here we illustrate this using the virco data, but we don't dichotomize the mutations as we had previously.

```
> VircoGeno <- data.frame(virco[,substr(names(virco),1,1)=="P"])
> Tree <- rpart(APV.Fold~.,data=VircoGeno)
```

Pruning in R

```
> Tree
n=939 (127 observations deleted due to missingness)
node), split, n, deviance, yval
      * denotes terminal node
 1) root 939 356632.300 12.946540
   2) P54=-,A,L,MI,S,T,TI,TS,V,VA,VI,VIM,VL,X 889 237601.200 10.726550
     4) P46=-,ILM,IM,LM,LMI,MI,MIL,ML,V,VIM,X 481  44960.940  4.506653
       8) P54=-,T,TI,TS,VI,X 342   4475.893  1.980702 *
       9) P54=A,L,MI,S,V,VA 139  32934.020 10.721580
        18) P89=-,M,ML 132  24074.510  9.125000
          36) P82=-,A,AT,AV,S,T,TS 122  15185.570  7.560656 *
          37) P82=C,F,M,TA 10   4948.009 28.210000 *
        19) P89=I,V,VL 7   2178.014 40.828570 *
     5) P46=I,L,LI,LIM,VL 408 152093.900 18.059310
      10) P47=- 340 107235.300 15.332650
        20) P84=-,VI 232  56131.660 11.931900
          40) P50=-,L 214  37976.750 10.106540
            80) P33=-,I,LF,M,V 167  15681.720  7.290419 *
            81) P33=F,FL,IL,MIL 47  16264.770 20.112770 *
          41) P50=V 18   8964.740 33.633330 *
```

Pruning in R

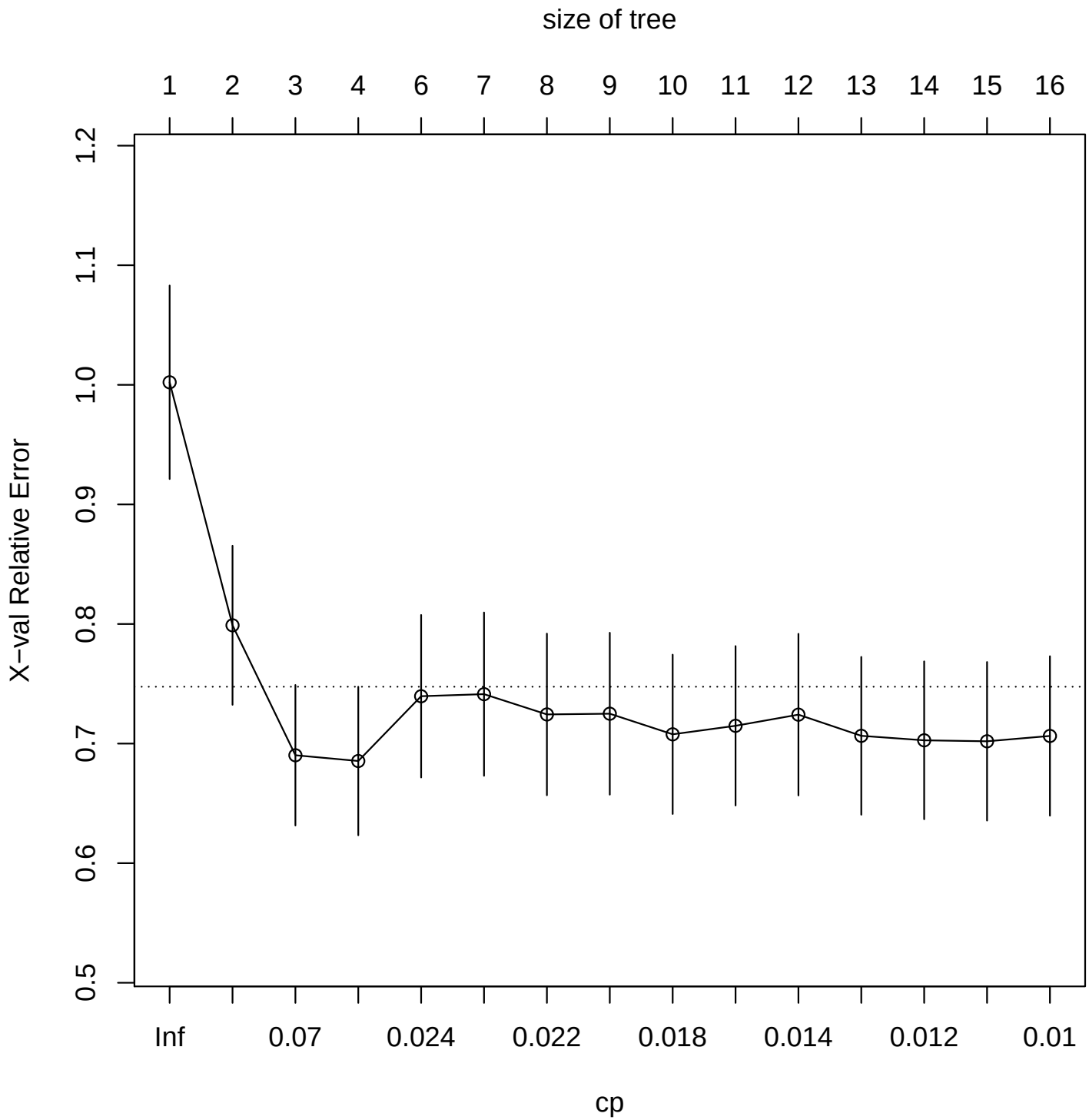
```
      21) P84=A,V,X 108  42656.790 22.637960
        42) P91=-,A,N,ST,Z 101  34293.240 20.867330
          84) P76=- 92  25472.030 18.966300 *
          85) P76=V 9   5090.080 40.300000 *
        43) P91=S,SA 7   3478.089 48.185710 *
    11) P47=A,V,VI 68  29691.790 31.692650
      22) P20=-,M,RK,T,TI,TK,VI 35   7891.207 23.125710 *
      23) P20=I,R,V 33  16507.440 40.778790
        46) P53=L,LF 13   4679.171 26.661540 *
        47) P53=- 20   7553.350 49.955000 *
3) P54=LI,M,MIL,VM 50  36749.950 52.418000
  6) P20=-,M,R,TK,V 33  21075.000 43.024240
   12) P46=- 11   5723.236 20.881820 *
   13) P46=I,IM,V,VI 22   7262.030 54.095450 *
  7) P20=I,QK,RK,T,VI 17   7110.222 70.652940 *
```

Pruning in R

We can make a plot to look at the error

```
> plotcp(Tree)
```

To determine a value for pruning, the help file for the plot command suggests using the left-most value for which the mean is less than the horizontal line.


size of tree
1
2
3
4
6
7
8
9
10
11
12
13
14
15
16
X−val Relative Error
0.5
0.6
0.7
0.8
0.9
1.0
1.1
1.2
Inf
0.07
0.024
0.022
0.018
0.014
0.012
0.01
cp

Pruning in R

```
> printcp(Tree)
Regression tree:
rpart(formula = APV.Fold ~ ., data = VircoGeno)
Variables actually used in tree construction:
 [1] P20 P33 P46 P47 P50 P53 P54 P76 P82 P84 P89 P91
Root node error:  356632/939 = 379.8
n=939 (127 observations deleted due to missingness)
          CP nsplit rel error  xerror     xstd
1  0.230717      0   1.00000 1.00214 0.080908
2  0.113693      1   0.76928 0.79893 0.066504
3  0.042528      2   0.65559 0.69025 0.058739
4  0.024727      3   0.61306 0.68544 0.062107
5  0.024016      5   0.56361 0.73961 0.067964
6  0.022684      6   0.53959 0.74135 0.068287
7  0.021173      7   0.51691 0.72438 0.067585
8  0.018735      8   0.49574 0.72499 0.067711
9  0.016909      9   0.47700 0.70776 0.066616
```

Pruning in R

10	0.014842	10	0.46009	0.71488	0.066662
11	0.013699	11	0.44525	0.72421	0.067596
12	0.011987	12	0.43155	0.70650	0.065984
13	0.011050	13	0.41956	0.70275	0.066009
14	0.010462	14	0.40851	0.70192	0.066298
15	0.010000	15	0.39805	0.70639	0.066684

Pruning in R

Then the text suggests that we prune the tree so that there are 4 terminal nodes.

```
> pruneTree <- prune(Tree, cp=0.03)
> pruneTree
n=939 (127 observations deleted due to missingness)
node), split, n, deviance, yval
      * denotes terminal node
 1) root 939 356632.30 12.946540
   2) P54=-,A,L,MI,S,T,TI,TS,V,VA,VI,VIM,VL,X 889 237601.20 10.726550
     4) P46=-,ILM,IM,LM,LMI,MI,MIL,ML,V,VIM,X 481  44960.94  4.506653 *
     5) P46=I,L,LI,LIM,VL 408 152093.90 18.059310
      10) P47=- 340 107235.30 15.332650 *
      11) P47=A,V,VI 68  29691.79 31.692650 *
   3) P54=LI,M,MIL,VM 50  36749.95 52.418000 *
```

Pruning in R

While if we follow the suggestion from the help file for the plotting function we prune more aggressively.

```
> pruneTree <- prune(Tree, cp=0.07)
> pruneTree
n=939 (127 observations deleted due to missingness)
node), split, n, deviance, yval
      * denotes terminal node
1) root 939 356632.30 12.946540
  2) P54=-,A,L,MI,S,T,TI,TS,V,VA,VI,VIM,VL,X 889 237601.20 10.726550
    4) P46=-,ILM,IM,LM,LMI,MI,MIL,ML,V,VIM,X 481  44960.94  4.506653 *
    5) P46=I,L,LI,LIM,VL 408 152093.90 18.059310 *
  3) P54=LI,M,MIL,VM 50  36749.95 52.418000 *
```