Lecture 5: Non-Linear Prediction Methods (Cont)

Alan Lee

Department of Statistics
STATS 784 Lecture 5

August 7, 2017

Outline

Introduction

Projection Pursuit

MARS

Trees

Neural Networks

Tuning

Today's agenda

In this lecture we continue our discussion of prediction methods, introducing some more non-linear methods. Today we will cover

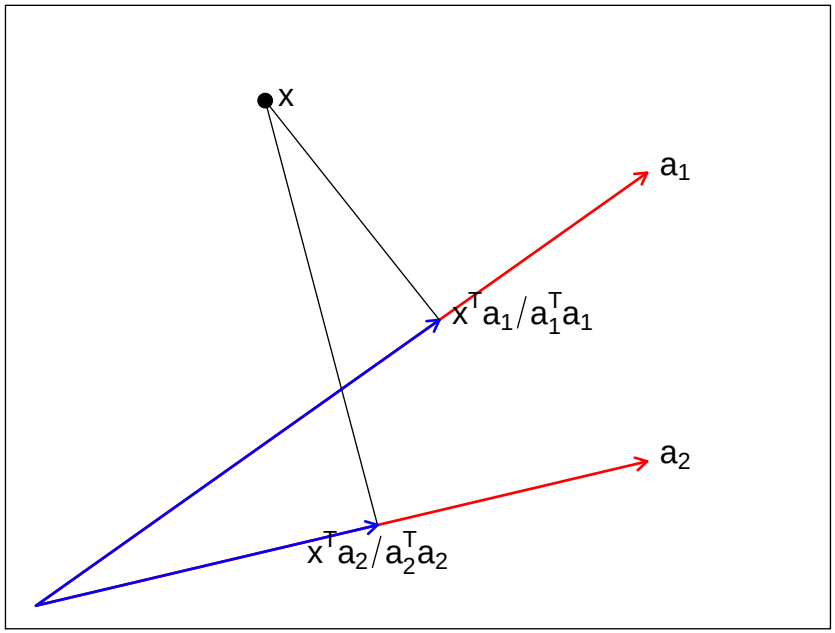
- Projection pursuit
- MARS
- Trees
- Neural networks
- Use of the `caret` package

We will use the Boston housing data from *Introduction to Statistical Learning* as a running example. Note: references to HTF refer to Hastie, Tibshirani and Friedman, *The Elements of Statistical Learning, 2nd Ed*

Projection pursuit

These are a generalized form of additive model: we first project the x-vector onto one of several directions $a_1, \ldots a_M$ in X-space, making new features (derived features) $v_1 = \frac{a_1^T x}{a_1^T a_1}, \ldots, v_M = \frac{a_M^T x}{a_M^T a_M}$

.



Projection pursuit (cont)

We then use an additive model in the new variables:

$$f(x) = \phi_1(v_1) + \cdots \phi_M(v_M),$$

We now have to choose both the directions a and the ϕ's. The ϕ's are smooth functions estimated using the backfitting algorithm. The predictor is

$$\hat{f}(x) = \phi_1(a_1^T x) + \cdots \phi_M(a_M^T x)$$

The ϕ's and the $\hat{a}$'s are functions of the training set. The ϕ's are called ridge functions.

R function: `ppr` Ref: HTF p389 (Ch 9)

Example: Boston housing data

```
library(MASS)
data(Boston)
ppr.fit = ppr(log(medv)~., data=Boston,
       sm.method = "spline", max.terms = 4, nterms = 6)
summary(ppr.fit)
par(mfrow=c(2,2))
plot(ppr.fit)
```

Output of summary function

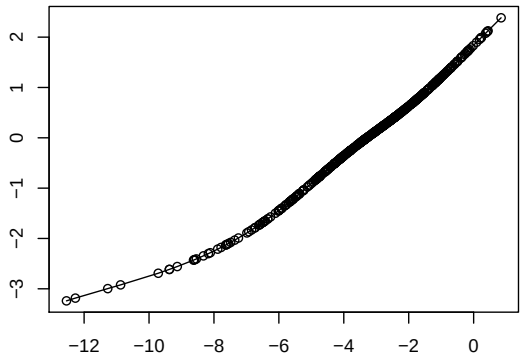
```
Call:
ppr(formula = log(medv) ~ ., data = Boston, sm.method = "spline",
    max.terms = 6, nterms = 4)

Goodness of fit:
 4 terms  5 terms  6 terms
9.387864 8.511150 8.132592

Projection direction vectors:
        term 1       term 2       term 3       term 4
crim    -0.0921213253 -0.0102906139 -0.0243395696  0.0245737567
zn       0.0052038340  0.0276757684  0.0070986183 -0.0017145470
indus    0.0034796721 -0.0928573557 -0.1199240669  0.1656041283
chas     0.1009101123  0.5728053710  0.5807235777 -0.1380548831
nox     -0.7900875794 -0.5121167030 -0.2468754818 -0.4480900748
rm       0.4642344114 -0.4496035938  0.7092952288  0.6750359155
age     -0.0030929265  0.0230042046 -0.0080653738  0.0147565288
```

Output (cont)

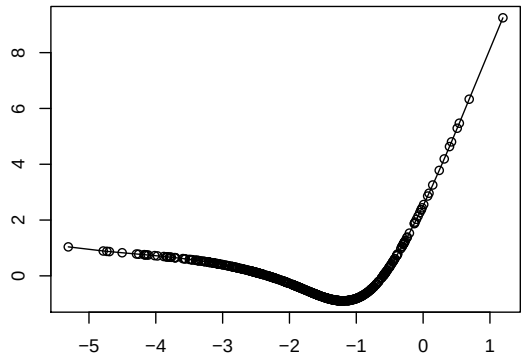
```
age      -0.0030929265  0.0230042046 -0.0080653738  0.0147565288
dis      -0.3056872201 -0.4222035565 -0.2299029019  0.5092284924
rad       0.0341271098  0.0183595612 -0.0610274999  0.0670092951
tax      -0.0023340337  0.0013227690 -0.0017514283  0.0014680500
ptratio -0.1568562836  0.1350668289  0.0692735106  0.0282272552
black     0.0033962754 -0.0023332774  0.0031164110  0.0002259702
lstat    -0.1493314182 -0.0249080780  0.1491038740 -0.1778495657

Coefficients of ridge terms:
    term 1     term 2     term 3     term 4
0.34249326 0.10440649 0.09643623 0.09491871

Equivalent df for ridge terms:
term 1 term 2 term 3 term 4
  5.07   5.10   5.08   5.07
```

Note: the RSS is 9.387864 and the R^2 is 89%.

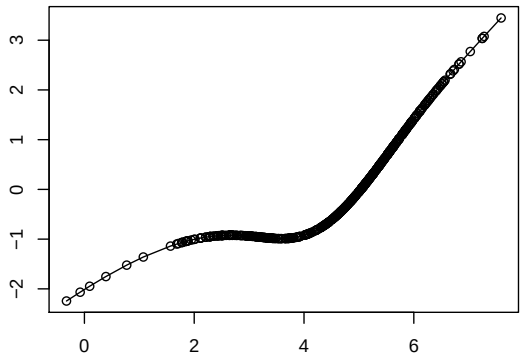
Example: Boston housing data



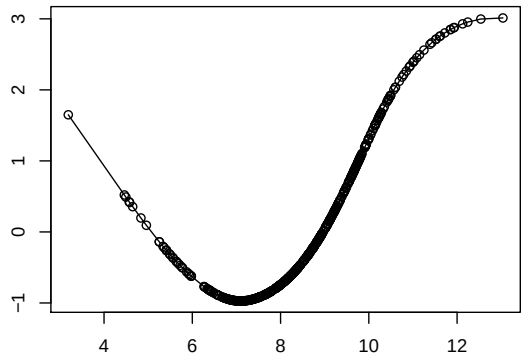
term 1



term 2



term 3



term 4

Example: Boston housing data

Points to note:

- Up to `max.terms` ridge terms are added, then deleted one at a time until `nterms` are left.
- The projection directions are chosen using a Gauss-Newton method which can be implemented as repeated weighted least squares calculations.
- The algorithm alternates between updating *phi*'s (using smoothing) for fixed directions, and new directions (for fixed *phi*'s). Typically the ridge functions are added one at a time and not updated.
- The ridge functions are removed using a least squares criterion, similar to backward elimination.

The original paper introducing ppr is by Jerry Friedman and Walter Stuezle (Friedman and Stuezle, 1981). See HTF p389 for more information.

MARS

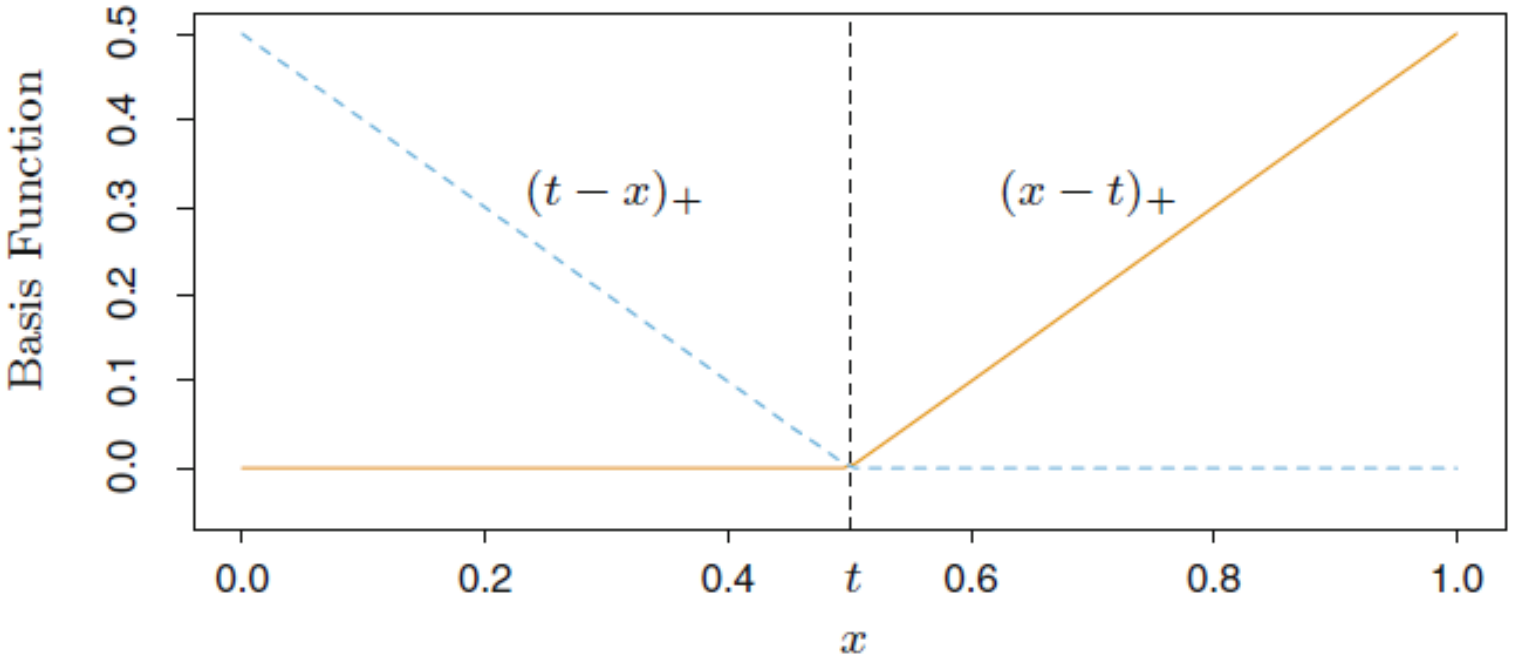
MARS = Multivariate Adaptive Regression Splines

- Consider transformations of the predictors x_j of the form $\max(0, x_j - t)$ and $\max(0, t - x_j)$ where t is one of the data values of variable x_j. These are called hinge functions or linear splines, see picture on next slide.
- Fit a model using forward selection as in linear regression, adding terms of the type

$$\beta_1 \max(0, x_j - t) + \beta_2 \max(0, t - x_j - t)$$

  with products of these terms with terms already in the model, choosing them to minimise the residual sum of squares. (See HTF p 322 for details). Only products of one or two terms are usually considered.
- Stop adding terms when some preset number of terms is reached.
- This model will probably be too complicated (will overfit) so drop some fixed number of terms one at a time, choosing them to give the mimimum increase in the RSS. The fixed number to drop can be chosen by generalised cross-validation (see HTF)
- R function: `mars, earth` Ref: HTF p 321 (Ch9)

The MARS basis functions



Example: Boston housing data

```
library(earth)
data(Boston)
mars.fit = earth(log(medv)~., data=Boston,
                         nk=30, degree=2)
summary(mars.fit)
par(mfrow=c(2,2))
plot(mars.fit)
```

Fitting 30 terms, allowing products of pairs of terms.

Example: Boston housing data

```
library(earth)
data(Boston)
mars.fit = earth(log(medv)~., data=Boston,
                         nk=30, degree=2)
summary(mars.fit)
par(mfrow=c(2,2))
plot(mars.fit)
```

Fitting 30 terms, allowing products of pairs of terms.

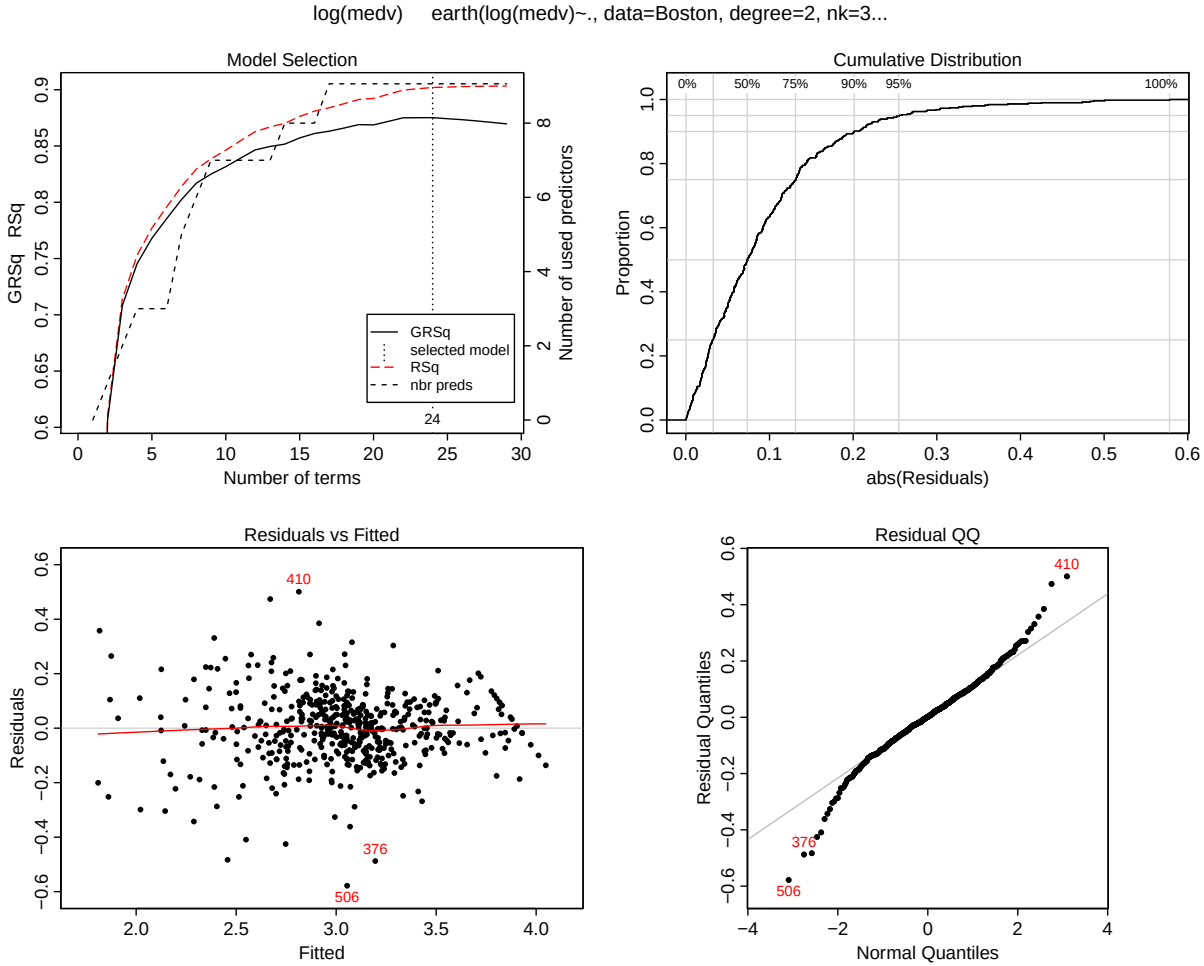
Example: Boston housing data

```
Call: earth(formula=log(medv)~., data=Boston, degree=2, nk=30)

                                  coefficients
(Intercept)                         3.11665848
h(rm-6.405)                         0.28197546
h(307-tax)                          0.00141749
h(tax-307)                          0.00028142
h(19.2-ptratio)                     0.05277268
h(ptratio-19.2)                    -0.03033980
h(172.91-black)                    -0.00089799
h(6.12-lstat)                       0.04406193
h(lstat-6.12)                      -0.04705400
h(7.99248-crim) * h(lstat-6.12)     0.00256829
h(crim-7.99248) * h(lstat-6.12)    -0.00060203
h(18.1-indus) * h(6.405-rm)        -0.01436065
h(indus-18.1) * h(6.405-rm)        -0.07137486
h(0.507-nox) * h(19.2-ptratio)     -0.41489781
h(nox-0.507) * h(19.2-ptratio)     -0.21330718
```

Example: Boston housing data

```
h(0.693-nox) * h(lstat-6.12)      0.16775948
h(nox-0.693) * h(lstat-6.12)      0.13471965
h(rm-6.405) * h(ptratio-19.1)    -0.21945542
h(6.405-rm) * h(lstat-23.79)      0.03854071
h(1.5894-dis) * h(tax-307)        0.00501698
h(dis-1.5894) * h(tax-307)        0.00012695
h(1.6102-dis) * h(lstat-6.12)    -0.07052066
h(dis-1.6102) * h(lstat-6.12)    -0.00683820
h(tax-403) * h(19.2-ptratio)     -0.00408753

Selected 24 of 29 terms, and 9 of 13 predictors
Termination condition: Reached nk 30
Importance: lstat, rm, crim, dis, tax, ptratio, indus, nox, black, ...
Number of terms at each degree of interaction: 1 8 15
GCV 0.02089385    RSS 8.269019    GRSq 0.8751958    RSq 0.9019985
```

Plots

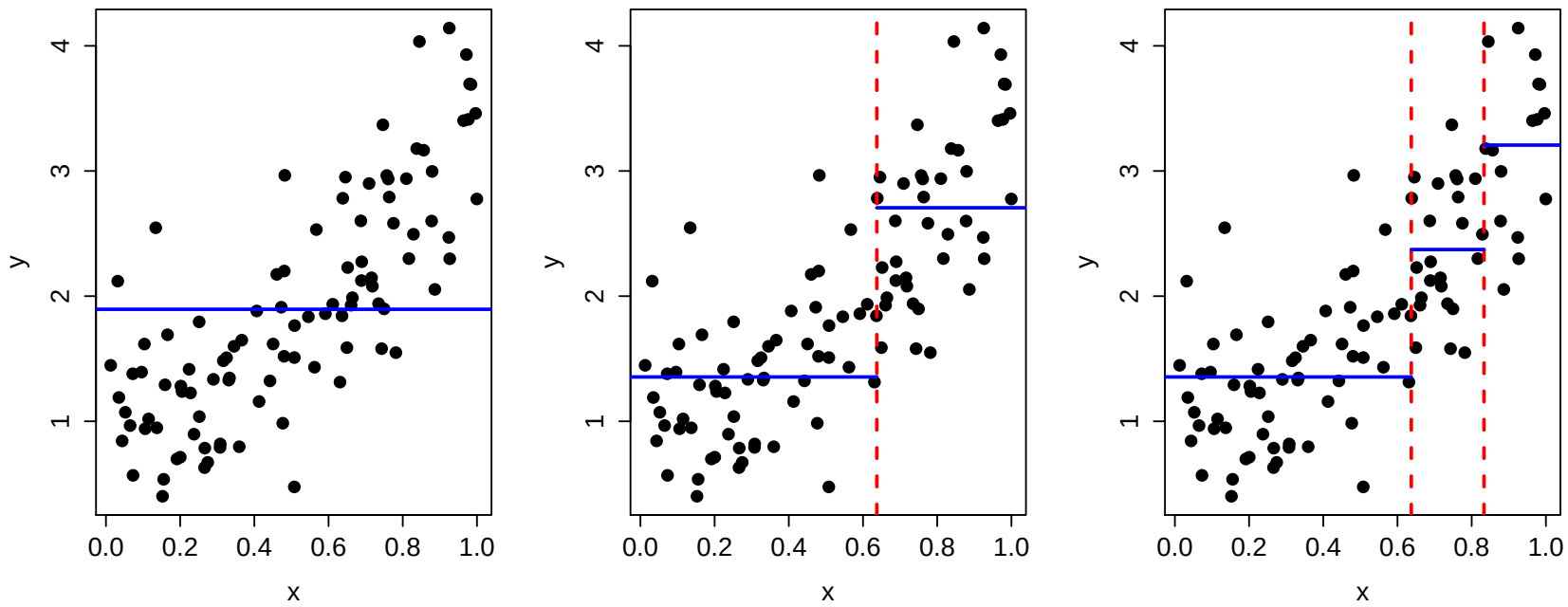

log(medv) earth(log(medv)~., data=Boston, degree=2, nk=3...
Model Selection
GRSq RSq
Number of used predictors
Number of terms
GRSq
selected model
RSq
nbr preds
24
Cumulative Distribution
0% 50% 75% 90% 95% 100%
Proportion
abs(Residuals)
Residuals vs Fitted
Residuals
Fitted
410
376
506
Residual QQ
Residual Quantiles
Normal Quantiles
410
376
506

Regression trees (CART)

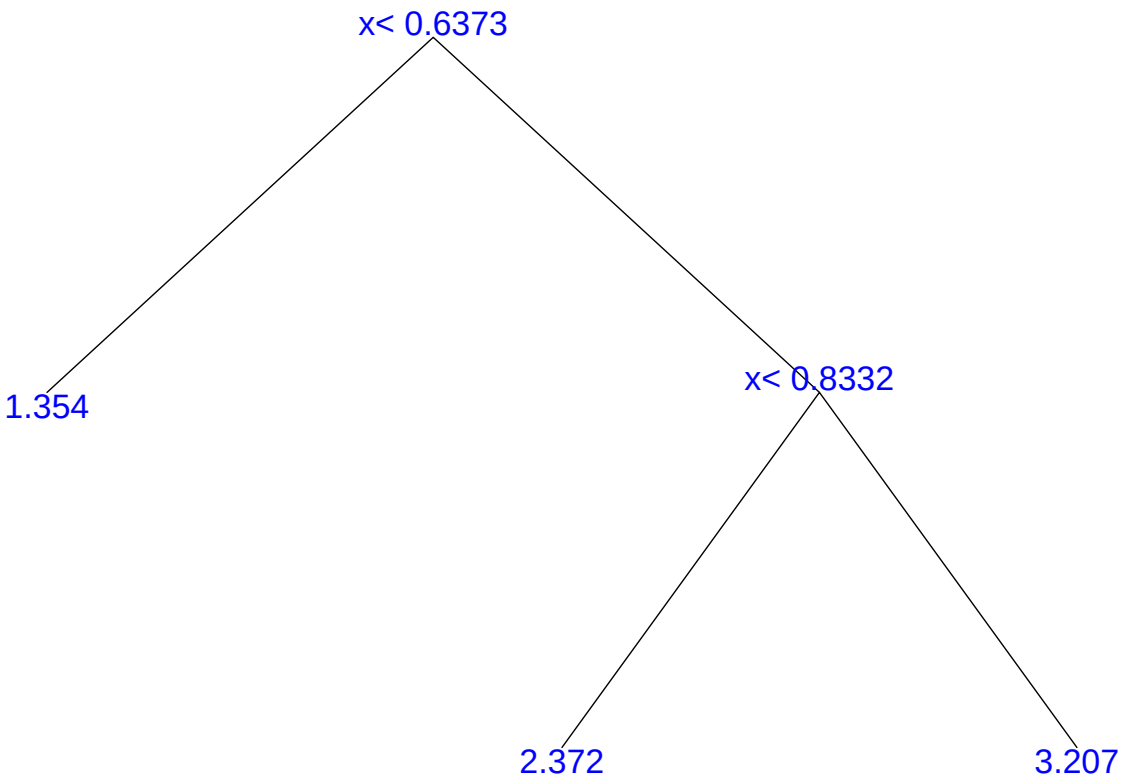
- This class of functions consists of multi-dimensional step functions, constant on regions of feature-space.
- They can be defined by means of a tree structure, that enables a nice explanation of how the function behaves.
- We illustrate with a simple example involving a an output y and a single input x.

Ref: ISLR p302, V&R p251, HTF p305 (Ch9)

Regression trees (cont)



Regression trees (cont)



Regression trees (cont)

To fit a regression tree, we "grow" the tree by adding two "child" nodes to an existing node. We start with the "root" node, containing all the observations. This corresponds to te whole feature space. To add nodes:

- Split the data contained in a node (region) into two complementary subsets S and S', according as $x_j \leq c$ or $x_j > c$. The split will depend on the variable used (x_j) and the threshold (c). In practice only the data values need be used for c.
- Work out the reduction in sum of squares:

$$\text{Residual SS of data at node}(= \sum_i (y - \bar{y})^2)$$

$$-\text{Residual SS of data in S} - \text{Residual SS of data in S}'$$

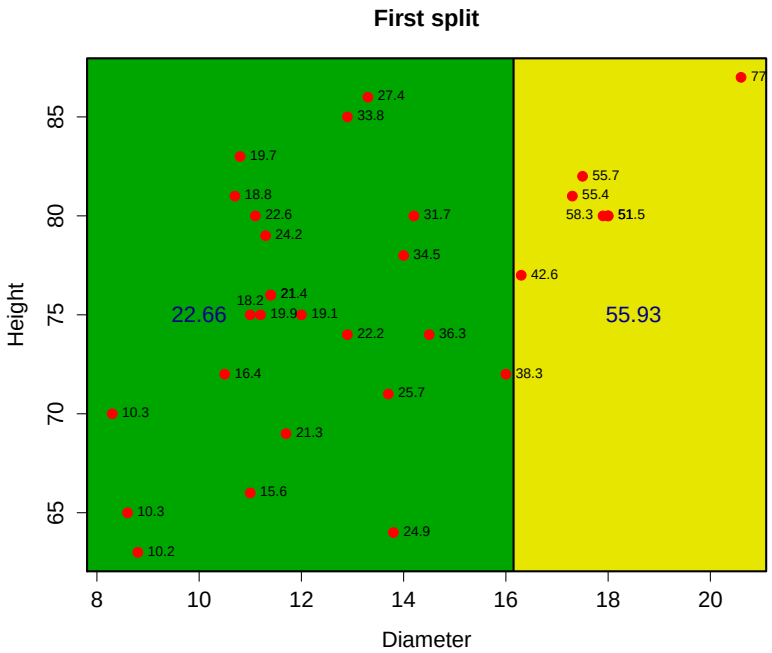
We do this for all possible splits at all the nodes (regions).

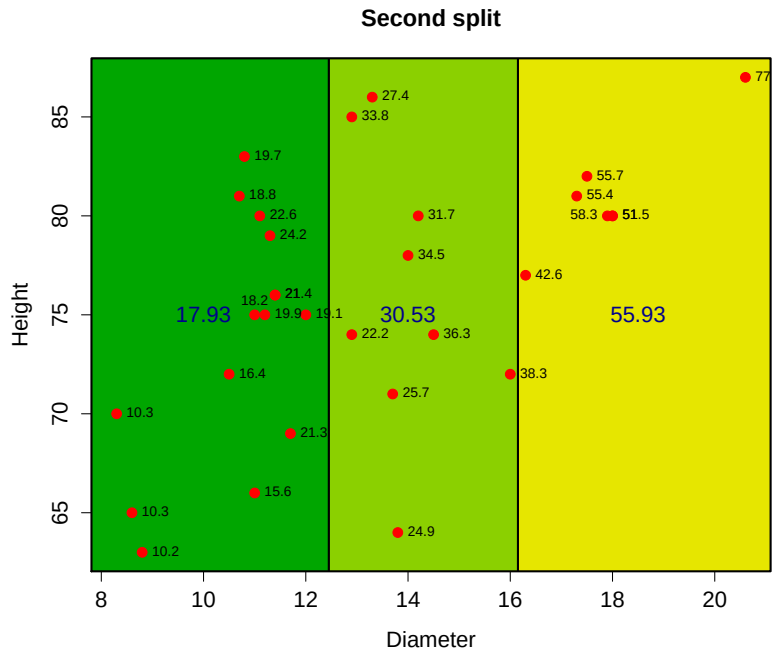
Regression trees (cont)

- Choose the split that gives the biggest reduction in the RSS. The two new nodes correspond to two new regions.
- Carry on this process until some stopping rule is satisfied. One such could be to only split nodes (regions) that contain more that a fixed number of observations (say 5).
- Since each split enlarges the model, the R^2 goes up with each split. We could stop if the R^2 increases by less than a set amount.
- The terminal nodes correspond to regions of feature-space that partition the space. The value of the fitted function $\hat{f}$ on a region is the mean of the y values of the observations contained in that region.
- This algorithm is called **recursive partitioning**. It is a "greedy algorithm" in that it only looks on step ahead, and can't go backwards.

Example: the cherry tree data

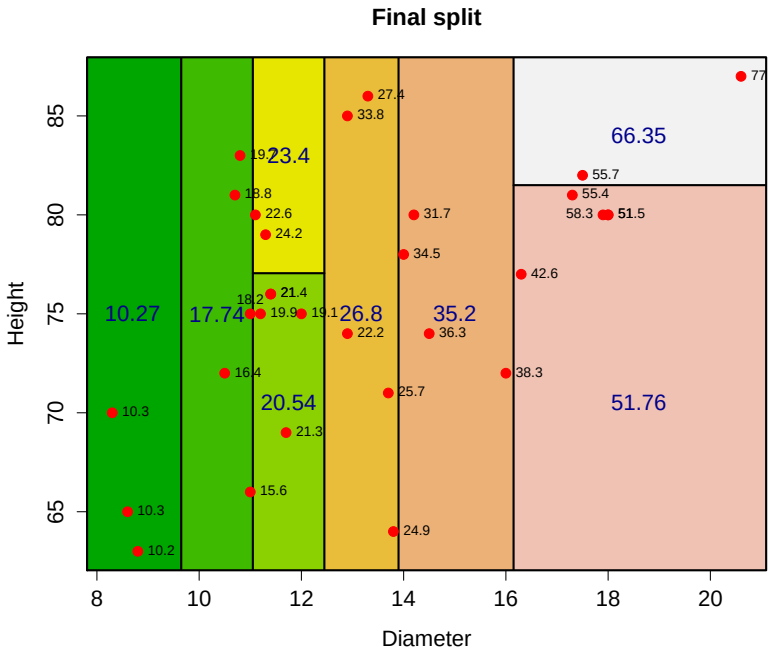
```
mytree = rpart(volume~height + diameter, data=cherry.df,
        method="anova", cp=0.001, minsplit = 5)
```

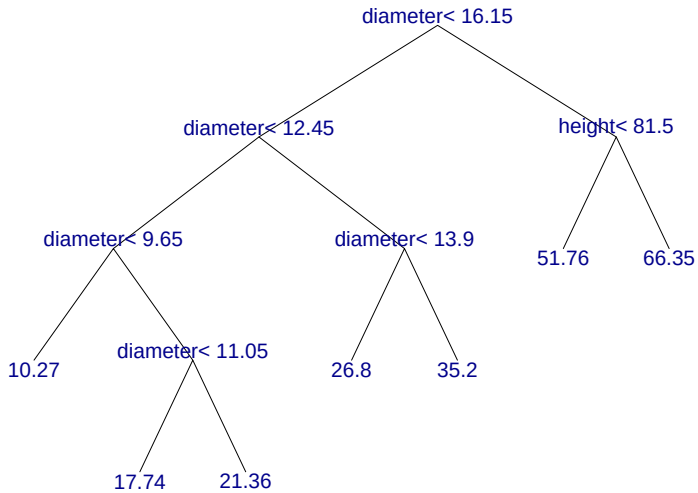




Cherry trees (cont)

```
mytree = rpart(volume~height + diameter, data=cherry.df,
        method="anova", cp=0.001, minsplit = 7)
```





Pruning

If the tree is grown too much, the process can result in a tree that is too complex. To obtain a tree that will predict better, we "prune" the tree. Suppose we delete all nodes that are below some fixed node. This results in a subtree, T say. Define the residual sum of squares for a subtree as the sum of the RSS $= \sum_i (y - \bar{y})^2$ summed over the terminal nodes. Consider the criterion

$$\text{Residual SS of } T + \alpha \times \text{Number of terminal nodes of } T,$$

where α is some fixed number. We choose the tree that minimizes this criterion. The value of α can be chosen by cross-validation.

R function: `rpart`

Pruning: cp

An equivalent method is to stop growing the tree when the best split increases the R^2 by an amount less that some cut-off number, called cp (complexity parameter) in the R rpart function. This is equivalent to the previous method if we set cp $= \alpha/$(RSS of the null model).

A good strategy is to over-grow the tree, and then prune it back. For each subtree, we can work out a cross-validated estimate of the prediction error for that tree, and select the subtree with the best PE. R has a pair of functions that do this nicely: `printcp` and plotcp.

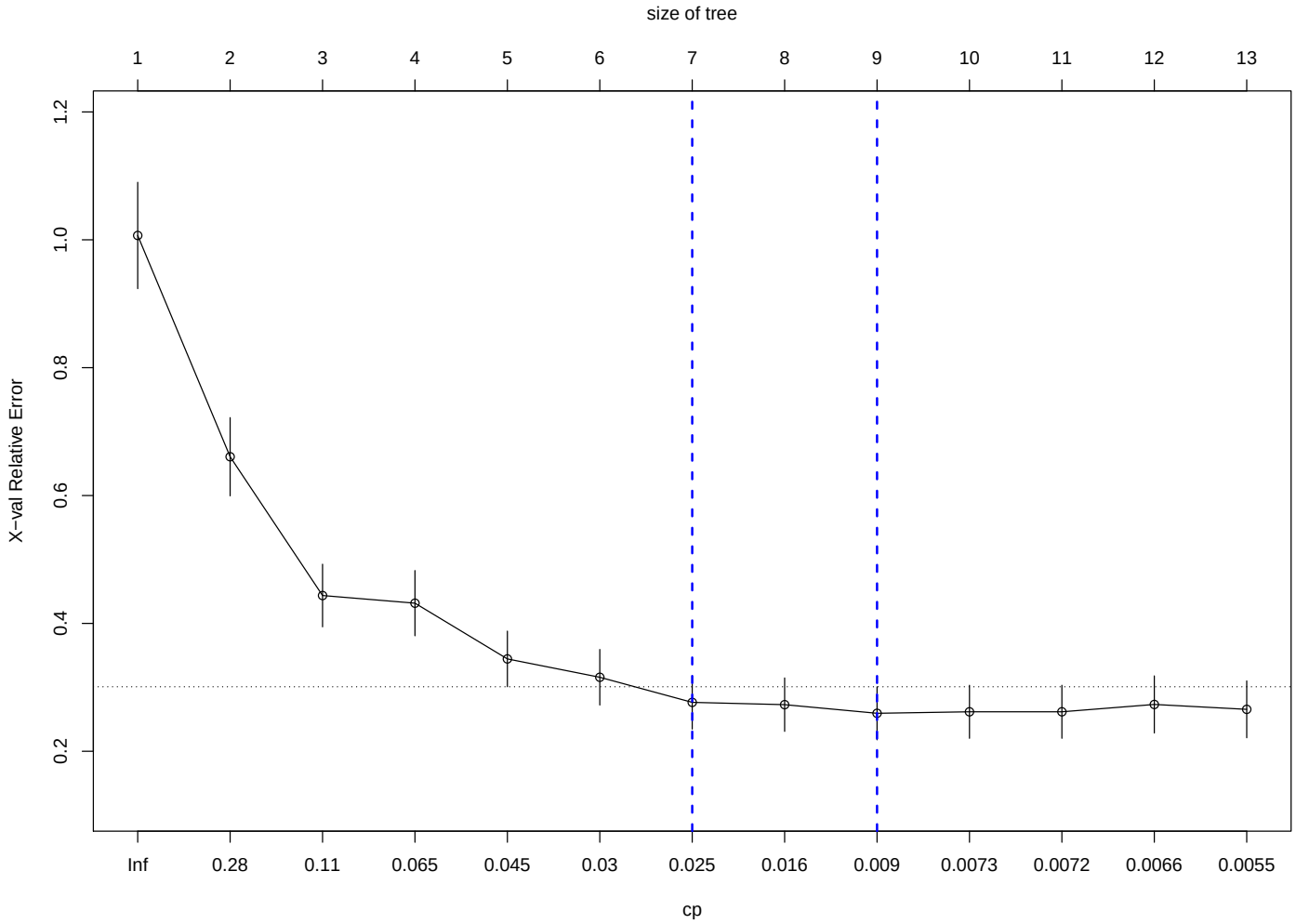
We illustrate the process with the Boston housing data.

Pruning: cp

Lets fit a model with cp=.005, and won't split regions having less than 5 points.

```
> library(MASS)
> data(Boston)
> library(rpart)
> tree.fit = rpart(medv~., data=Boston,
        method="anova", cp=0.005, minsplit=5)
# now the cp plot (see next slide):
> plotcp(tree.fit)
# and the printed equivalent
> printcp(tree.fit)
```

plotcp


size of tree
1
2
3
4
5
6
7
8
9
10
11
12
13
X−val Relative Error
0.2
0.4
0.6
0.8
1.0
1.2
Inf
0.28
0.11
0.065
0.045
0.03
0.025
0.016
0.009
0.0073
0.0072
0.0066
0.0055
cp

printcp

```
Regression tree:
rpart(formula = medv ~ ., data = Boston, cp = 0.005, minsplit = 5)
Variables actually used in tree construction:
[1] crim    dis     lstat   nox     ptratio rm
Root node error: 42716/506 = 84.42, n= 506

           CP nsplit rel error  xerror     xstd
1  0.4527442      0   1.00000 1.00679 0.083185
2  0.1711724      1   0.54726 0.66069 0.061169
3  0.0716578      2   0.37608 0.44361 0.048990
4  0.0590015      3   0.30443 0.43166 0.051017
5  0.0337559      4   0.24542 0.34445 0.043491
6  0.0266130      5   0.21167 0.31577 0.043494
7  0.0235724      6   0.18506 0.27651 0.041791
8  0.0108593      7   0.16148 0.27291 0.041823
9  0.0074304      8   0.15062 0.25940 0.041483
10 0.0072654      9   0.14319 0.26175 0.041486
11 0.0070714     10   0.13593 0.26175 0.041486
12 0.0061263     11   0.12886 0.27324 0.044665
13 0.0050000     12   0.12273 0.26560 0.044499
```

Neural networks

Neural networks were first introduced in the field of artificial intelligence, and as HTF remark (section 11.3 of the *Elements of Statistical Learning*)

> *There has been a great deal of hype surrounding neural networks, making them seem magical and mysterious. As we make clear in this section, they are just nonlinear statistical models, much like the projection pursuit regression model discussed above.*

The neural network function is made up of non-linear transformations (using the sigmoid function) of linear combinations of the predictors:

$$f(x) = \alpha_0 + \sum_h \beta_h \sigma(\alpha_{0h} + \sum_i \alpha_{ih} x_i)$$

where $\sigma(x) = exp(x)/(1 + exp(x))$. (This is called the sigmoid function). Thus, we assume that our target y is releted to our features x by

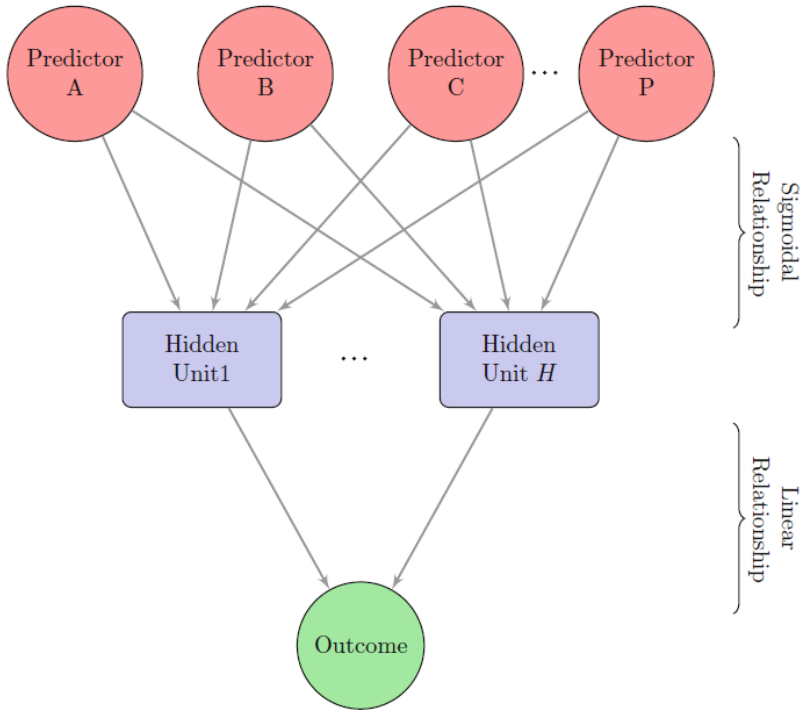
$$y = f(x) + noise$$

where f is the NN function above. The parameters α_0, β_h and α_{0h} are called **weights**. By varying the weights, a very flexible family of functions is obtained.

Ref: HTF p392 (Ch9)

Neural networks: the picture

This function can be thought of as describing the action of a "neural network":



Fitting neural networks

- Since the NN function depends on a finite set of parameters (weights), the function can be fitted by non-linear least squares, using a specialized iterative algorithm (see HTF p 391 for details).
- The choice of starting values is important as the objective function has multiple local minima - several starts should be tried.
- It is advantageous to add a "regularization penalty" (proportional to the sum of the squared weights) to the least squares criterion to assist the optimization process. If this is done the input data should be standardized.

R function: `nnet`

Example: Boston housing data

```
library(nnet)
data(Boston)
nnet.fit = nnet(log(medv)~., data=Boston, size = 6,
    linout=TRUE, decay = 0.01, maxit = 500)
```

Typically the weights are not of too much interest (although they can be seen using the summary function), and the neural network is treated as a black box predictor. The RSS from this fit was 7.956961 and the R^2 ws 91%.

Model complexity and model tuning

- For linear predictors, complexity is the number of variables
- For ppr, measured by the number of ridge functions
- For MARS, the number of terms fitted in the forward selection
- For Trees, the size of the tree, as controlled by cp
- For Neural nets, measured by the number of units in the hidden layer

Caution: Models that are too complex will not be good for prediction - model noise as well as signal

Choose a model with moderate complexity, one that minimizes the PE. This involves searching over the tuning parameters (size of hidden layer, value of cp etc) and selecting the tuning parameters that minimize the PE. We can use the `caret` package for this. `caret` = "classification and regression training". We illustrate its use with some sample code on the next slide. See also the caret website

http://topepo.github.io/caret/index.html

Example: use of caret

Note: The arguments `decay` and `size` (the number of hidden layer units) are parameters used in the `nnet` function we used to fit a neural network. They need to be "tuned" to find the best predictor. The parameters `maxit`, `trace` and `linout` are additional parameters controlling the fitting process, but these do not change.

```
library(caret)
my.grid <- expand.grid(.decay = c(0.001), .size = c(4,6,8))
nn.CV <- train(y~., data = data,
 method = "nnet",
    maxit = 1000,
    tuneGrid = my.grid,
    trace = FALSE,
    linout = TRUE,
    trControl = trainControl(method="cv", number=5,
       repeats=100))
```

Caret output

```
> nn.CV
Neural Network
506 samples  13 predictor
No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 404, 406, 404, 405, 405
Resampling results across tuning parameters:

  decay  size  RMSE       Rsquared   RMSE SD     Rsquared SD
  0.001  4     0.2301332  0.7376485  0.11610168  0.15925283
  0.001  6     0.1751344  0.8220023  0.02078402  0.04066819
  0.001  8     0.1962414  0.7799063  0.02794876  0.05555551
  0.010  4     0.2341524  0.6475049  0.09870711  0.32277666
  0.010  6     0.1701837  0.8343562  0.01381691  0.02319806
  0.010  8     0.1825407  0.8148357  0.02755196  0.03505227

RMSE was used to select the optimal model using  the smallest value.
The final values used for the model were size = 6 and decay = 0.01.
```

Points to note

- The best model has a cross-validated R^2 of 83%, considerably less than the 91% obtained from the training set.
- We now would feel reasonably confident about using this NN model as a predictor.
- We would make our predictions using the predict function as usual.
- In Assignment 2 you will be asked to compare all the methods we have discussed (which will involve tuning all of them). You will need to consult the caret website to get the names of the tuning parameters - usually the same as in the R functions.
- Don't forget the period in `expand.grid(.decay = c(0.001), .size = c(4,6,8))`