Lecture 4: Prediction methods

Alan Lee

Department of Statistics
STATS 769 Lecture 4

July 29, 2015

Outline

Today's agenda

In this lecture we continue our discussion of prediction methods Some of the ideas should be familiar from STATS 760. Today we will cover

- Trees
- Neural networks
- Boosting and Bagging, particularly random forests
- Use of the `caret` package

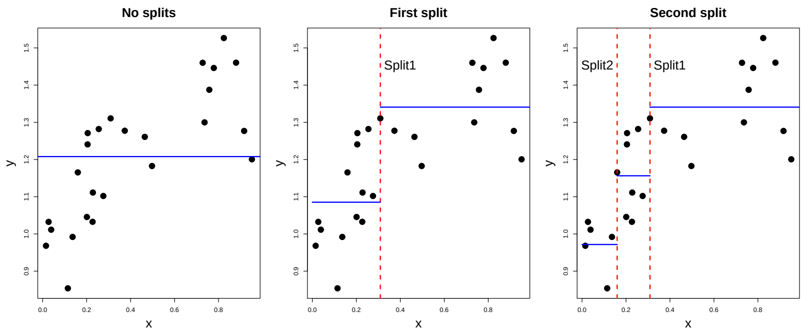
We will use the Boston housing data from *Introduction to Statistical Learning* as a running example. Note: References to V&R are to Venables and Ripley, *Modern Applied Statistics with S*, and to HTF to Hastie, Tibshirani and Friedman, *The Elements of Statistical Learning, 2nd Ed*

Regression trees (CART)

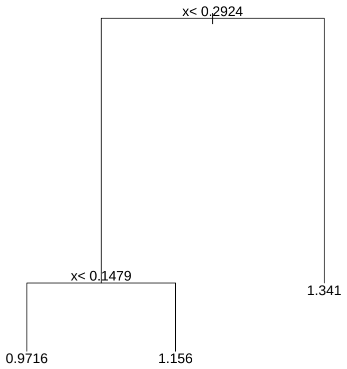
- This class of functions consists of multi-dimensional step functions, constant on regions of X-space.
- They can be defined by means of a tree structure, that enables a nice explanation of how the function behaves.
- We illustrate with a simple example involving a an output y and a single input x.

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

Regression trees (cont)

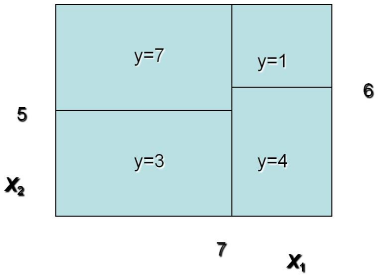


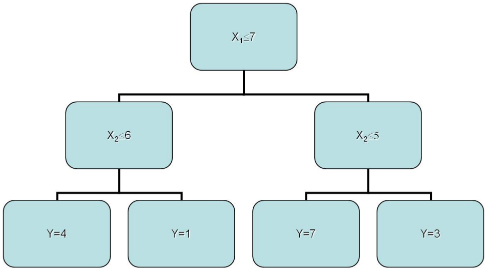
Regression trees (cont)



Regression trees (cont)

In two dimensions:





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. To add nodes:

- Split the data contained in a node 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).
- Work out the reduction in sum of squares:

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

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

Regression trees (cont)

- Choose the split that gives the biggest reduction. The two new nodes correspond to the two subsets S and S'.
- Carry on this process until nodes contain fewer that a fixed number of observations (say 5).
- The terminal nodes correspond to regions of X-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 node.

Pruning

The process on the previous slide will 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`

Neural networks

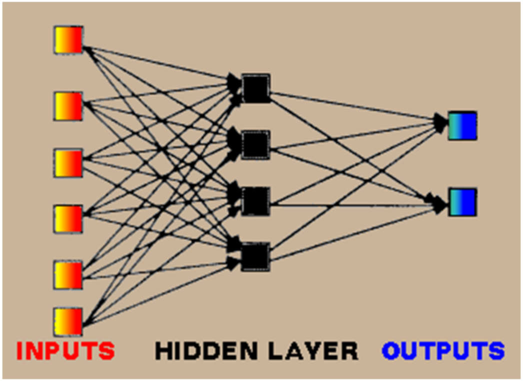
Let $\sigma(x) = exp(x)/(1 + exp(x))$. This is called the sigmoid function. The neural network function is made up of non-linear transformations (using the sigmoid function) of linear combinations of the predictors:

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

Ref: V&R p243, 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`

Complexity

- For linear predictors, complexity is the number of variables
- For Neural nets, measured by the number of units in the hidden layer
- For trees, by the number of terminal nodes

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, number of terminal nodes)

Boosting

The basic idea here is

1. Fit a model to the data. Then repeat
   1.1 Modify the data in some way
   1.2 Fit a model to the modified data
2. Average the results

Ref for Boosting: ISLR p321, HTF Chapter 10.

Application to regression

Forward stagewise modelling: (HTF p 341)
Many regression models fit an additive combination of "basis functions" i.e. express f as

$$f(x) = \sum_{m=1}^{M} \beta_m b(x, \gamma_m)$$

where the β_m are regression coefficients and $b(x, \gamma)$ is a "basis function" which depends on a parameter γ. Examples:

- Linear: $b(x, \gamma) = x_j$
- Trees: $b(x, \gamma) = I(x \in R_m)$
- Neural nets: $b(x, \gamma) = \sigma(\gamma_0 + \gamma^T x)$

FSM Algorithm

Let $L(y, \eta)$ be a loss function, e..g. $L(y, \eta) = (y - \eta)^2$.

1. Set $f_0(x) = 0$.
2. For m=1,2. . . ,M
    - 2.1 Compute

$$(\hat{\beta}\hat{\gamma}) = \text{argmin}_{\beta,\gamma} \sum_{i=1}^{n} L(y_i, f_{m-1}(x_i) + \beta b(x, \gamma))$$

    - 2.2 Set $f_m(x) = f_{m-1}(x) + \hat{\beta} b(x, \hat{\gamma})$.

Squared error loss, trees

If $L(y, \eta) = (y - \eta)^2$, then at each stage we are fitting a simple model involving one β and γ to the residuals from the previous fit. Note that to predict at x, we add up the predictions for each model.

Modification for trees: We replace 2.1 by

$$\text{argmin}_{\beta,\theta} \sum_{i=1}^{n} L(y_i, f_{m-1}(x_i) + T(x_i, \theta))$$

where T is a tree which can be represented by a sum of basis functions:

$$T(x, \theta) = \sum_{j=1}^{J} \beta_j I(x \in R_j)$$

Points to note

- Trees are usually small (2 -5 terminal nodes)
- M is large, say 00's. We are averaging many small noisy trees.
- Implemented in R by the `gbm` and `mboost` packages - see documentation in the R help files.
- We need to choose M (number of boosting steps) and J (tree size, number of terminal nodes).

Bagging

Bagging stands for *B*ootstrap *Agg*regation. The idea is: to make a prediction at x, we

- Draw B bootstrap samples
- Fit a model f to each sample
- Average the predictions from each model

Often done with trees, with a small tweek in the above. This results in Random Forests. Random Forests were invented by Leo Breiman, a Berkeley professor, and further developed by Adele Cutler (an Auckland graduate) Ref for Bagging: HTF Chapter 8 (section 8.7). Ref for Random Forests ISLR p 316, HTF Chapter 15, see also
`http://www.stat.berkeley.edu/ breiman/RandomForests/`

Random Forests

We apply the algorithm above to trees, modifying it as follows. To predict the response at x:

- Draw B bootstrap samples.
- For each sample, fit a tree of some specified depth. At each split, select the splits using only a randomly chosen subset of m variables, rather than choosing from all of them.
- Average the predictions $f(x)$ using only the OOB samples (OOB = "out-of-bag" samples, those not containing x.) Overfitting does not arise since we are using OOB samples only. As in boosting, we use many small trees, rather than a few big ones.

Example

We illustrate the use of these methods with the Boston housing data. These consist of data on 506 neigbourhoods around Boston. The output is the median house value, and the inputs are various socio-economic indices calculated for the neigbourhood. Type

```
library(MASS)
?Boston
```

to get a full description of the variables. In the handout we illustrate the R code to fit several different models and estimate the prediction error for each model. For example, for random forests:

Use of the caret package

You will have seen how the various methods must be “tuned”, adjusting various parameterss that regulate the complexity of the fitted model. For example, in neural networks we must specify the number of units in the hidden layer, in random forests, the number of variables to select for each tree, and the depth of the trees. The parameters are chosen by examining various combinations of tuning parameters and picking the combination that has the smallest PE.

This can be tiresome, but fortunately there is an R package `caret` which automates the process. `caret` = ”classification and regression training”. We illustrate its use with some sample code on the next slide.

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.

```
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 = 1,
    trControl = trainControl(method="cv", number=5,
       repeats=100))
```