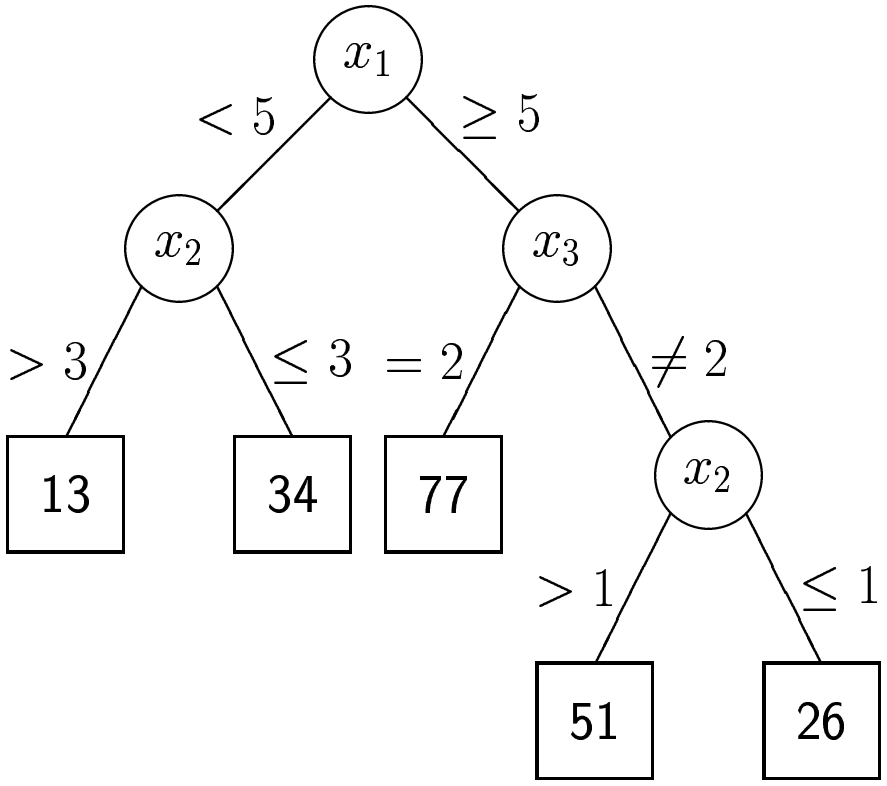
Tree-based models

Suppose we have a scalar outcome, y, and a p-vector of explanatory variables, x.

Regression tree: $y \in \mathcal{R}$

A regression tree partitions x-space into disjoint regions A_k and provides a fitted value $\mathsf{E}(y|x \in A_k)$ within each region.

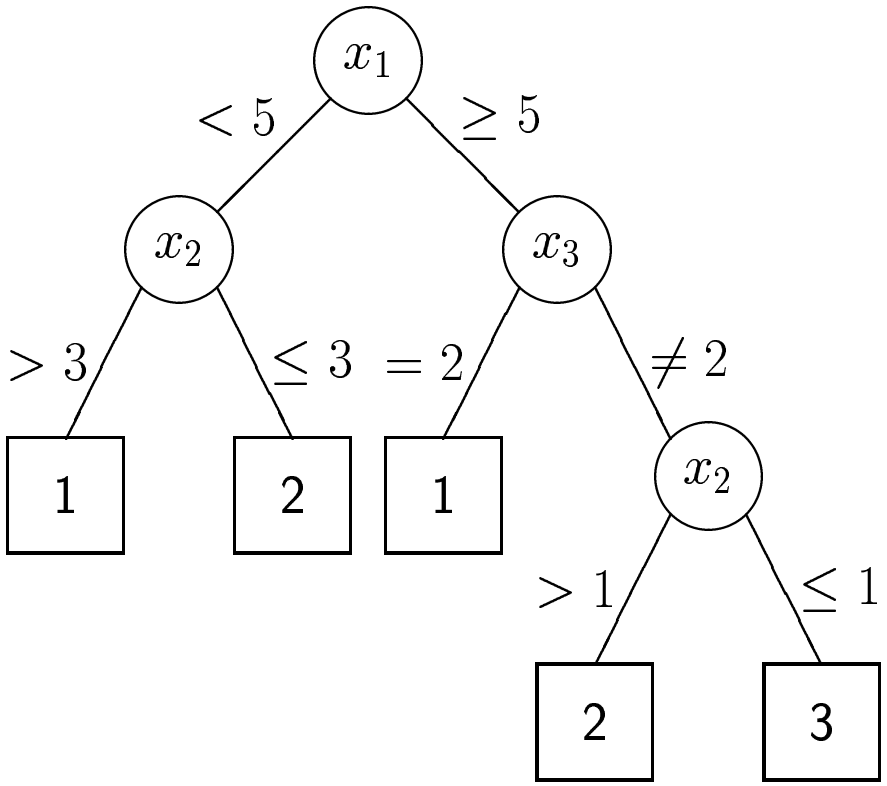


In other words, this is a decision tree where the outcome is a fitted value for y.

Tree-based models (continued)

Classification tree: $y \in \mathcal{K} = \{1, 2, \ldots, k\}$

A classification tree partitions x-space and provides a predicted value, perhaps $\arg\max_s \mathsf{Pr}(y = s | x \in A_k)$ in each region.



In computer science, this business goes under the name *recursive partitioning*.

Ref: Breiman et al (1984) Classification and regression trees. Wadsworth.

General points

- This is most natural when the explanatory variables are categorical (and it is especially nice when they are *binary*).
- There is nothing special about the tree structure...the tree just partitions x-space, with a fitted value in each region.
- **Advantage**: These models go after *interactions* immediately, rather than as an afterthought.
- **Advantage**: Trees can be easy to explain to non-statisticians.
- **Disadvantage**: Tree-space is huge, so we may need *a lot* of data.
- **Disadvantage**: It can be hard to assess uncertainty in inference about trees.
- **Disadvantage**: The results can be quite variable. (Tree selection is not very *stable*.)
- **Disadvantage**: Actual *additivity* becomes a mess in a binary tree. This problem is somewhat alleviated by allowing splits of the form $x_1 + bx_2 < (\geq)\ d$.

Model selection

The big issue here is *model selection*. In my view, the model selection problem consists of four orthogonal components.

1. Select a space of models
2. Search through model space
3. Compare models
   - of the same size
   - of different sizes (penalize complexity)
4. Assess the performance of a procedure

Important points:

- Components 2 and 3 are often confused (e.g., in stepwise regression). That's bad.
- People often forget component 1.
- People almost always ignore component 4; it can be the hardest.
- I've drifted back into statistics (from statistical computing).

Searching through trees

The search through trees is generally performed as follows:

1. **Grow** an overly large tree using forward selection. (At each step, find the *best* split.)

   Grow until all terminal nodes either

   (a) have $< n$ (perhaps $n = 1$) data points

   (b) are "pure" (all points in a node have the same outcome)

2. **Prune** the tree back, creating a nested sequence of trees, decreasing in complexity.

This suffers from the usual problems of forward selection, though things are somewhat better by growing up and pruning back rather than just growing up.

Better trees may be found by doing a one-step "look ahead," but this comes with the cost of a great increase in computation.

Comparing trees

Regression trees

For trees of the same size, we seek to minimize the residual sum of squares (or possibly the sum of absolute values) $= \sum (y - \hat{y})^2$.

For trees of different sizes, one approach is to minimize an AIC/BIC-type criterion: for a tree, γ, with q_γ terminal nodes, consider $\Psi(\gamma) = \log \mathsf{RSS}(\gamma) + q_\gamma D(n)/n$.

$D(n) = 2$ gives the AIC criterion.

$D(n) = \log n$ gives the BIC criterion.

These are both based on asymptotics. One may also look at $D(n) = \delta \log n$, which I call BIC-δ; it has the same asymptotic properties as BIC, but, with $\delta > 1$, puts greater penalty on model complexity and may have better small-sample properties.

The focus is generally on *minimizing prediction error*.

We use our observed data (x, y) to estimate a tree $\hat{\gamma}(x, y)$. We define the prediction error of the tree to be the mean squared error in a new value $(x^\star, y^\star)$: $\mathsf{E}\{[y^\star - \hat{y}(\hat{\gamma}, x^\star)]^2\}$.

The typical method is then to get an *honest* estimate of the prediction error either using set-aside *test* data or m-fold cross-validation.

Comparing trees (continued)

Classification trees

A classification tree defines a decision rule d, providing a predicted value $\hat{y} = d(x)$.

Consider a *loss* function $L(y, d)$, defining the cost associated with (incorrectly) predicting d when the true class is y. We wish to minimize the *risk*: $R(d) = \mathsf{E}\{L[y, d(x)]\}$.

In regression, we use squared error loss: $L[y, d(x)] = \{y - d(x)\}^2$.

In classfication, we create a matrix $L(y, d)$ where the diagonal elements (correct predictions) are 0. (The simplest choice is to make all non-diagonal elements 1, but we have much flexibility here.)

The risk (letting $d_x = d(x)$) is then

$$\begin{aligned} R(d) &= \mathsf{E}\{L(y, d_x)\} \\ &= \mathsf{E}\{\mathsf{E}[L(y, d_x)|x]\} \\ &= \mathsf{E}\{\sum_y L(y, d_x)\mathsf{Pr}(y|x)\} \end{aligned}$$

The re-substitution estimate of $R(d)$ for a tree should be obvious.

Honest estimates of risk

We wish to get an *honest* estimate of the risk associated with trees. We can then use that to pick from the nested sequence of trees that results from the pruning after the initial growing (minimize estimated risk).

Learning and test sets

If you have a lot of data, you might split it into a *learning set* (used to estimate the tree) and *test set* (used to estimate the risk associated with a tree.

"A lot of data" depends on the program. I'd think in the range of 500+ observations. The split need not be half/half: I seem to recall people recommending 2/3 learning, 1/3 test.

Advantage: It's easy
Disadvantage: Loss of efficiency by ignoring data

Cross-validation

1. Estimate trees, dropping k data points.
2. Look at error in predicting the k dropped points.
3. Repeat many times (in a *balanced* way).

Advantage: Make efficient use of data
Disadvantage: Heavy computation

Computing with trees

R: `library(tree)`; `library(rpart)` [MASS, ch 10]

An important issue: Storing trees

Binary trees are composed of nodes (root node, internal nodes and terminal nodes).

Root and internal nodes:

- Splitting rule (variable + what goes to right)
- Link to left and right daughter nodes
- Possibly a link to the parent node (null if this is the root node)

Terminal nodes:

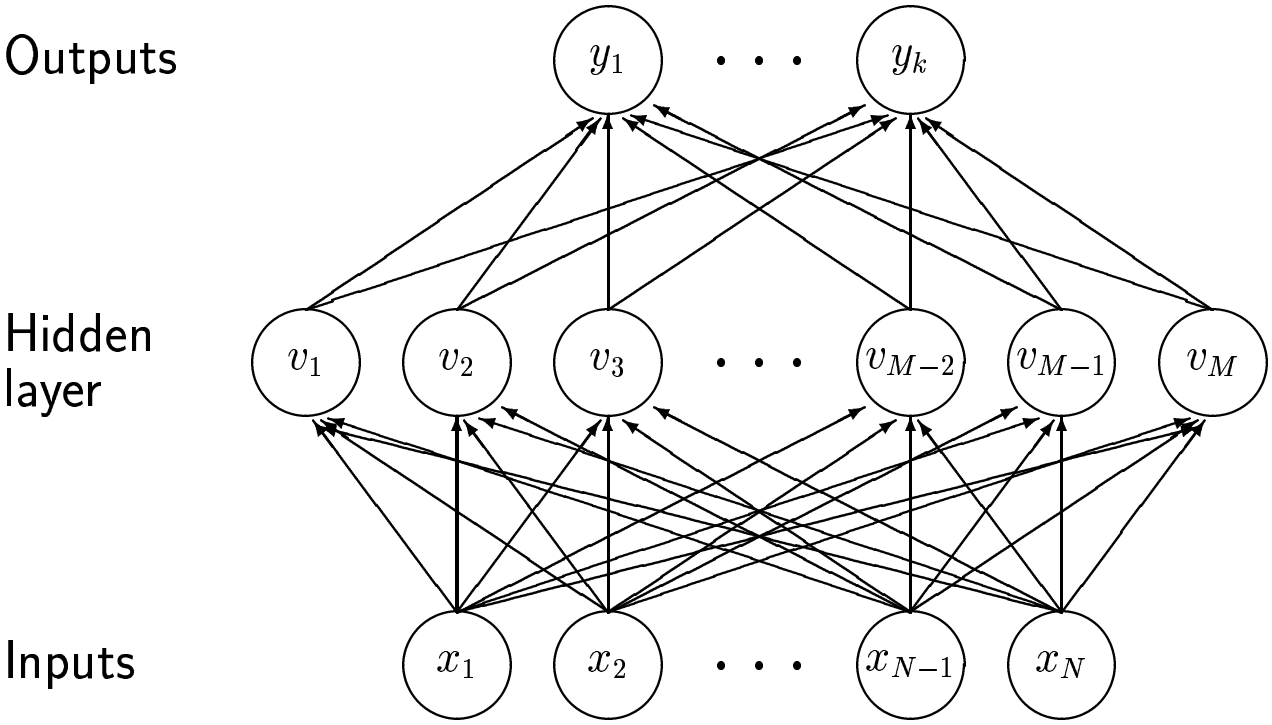
- Fitted value
- Possibly a link to the parent node

C: Use pointers and structures (`struct`)

R: It beats me. Take a look.

Neural networks

A generic feed forward neural network:



The output from each of the hidden units is of the form $v_j = \phi_h(\alpha_j + \sum_j w_{ij}x_i)$. The function ϕ_h is nearly always a logistic: $\phi_h(z) = \exp(z)/[1 + \exp(z)]$.

The outputs, y, are of the form $y_j = \phi_h(\alpha'_j + \sum_j w'_{ij}v_i)$. The function ϕ_0 may be the identity, a logistic or an indicator/threshold function.

We have a total of $M(k + N + 1) + k$ parameters here.

Neural networks (continued)

We seek a good predictor of $y = (y_1, \ldots, y_k)$ given $x = (x_1, \ldots, x_N)$.

The conditional distribution $\Pr(y|x)$ may be hugely complicated, with lots of interactions between the x_j's.

Rather than (as a statistician might do) try to build up a model from little bits, we form a massive, overparameterized, impressively flexible model and use a pile of data to form a good "black box" predictor.

There is theory to show that with a sufficiently large single layer, this sort of feed-forward network can well approximate *any arbitrary function*.

Why do care about these things?

- People use them.
- We might want to use them.
- We might want to make public statements trashing their use.

Refs: MASS §9.4; Cheng and Titterington (1994) Neural networks: A review from a statistical perspective. Statistical Science 9:2–54. [Look especially at the comments from Breiman, Ripley and Tibshirani.]

Neural networks (continued)

Applications: Regression, classification, clustering

- Speech recognition and generation
- Handwriting recognition
- Predict financial indices
- Optimize chemical processes
- Military stuff
- Identify cell abnormalities
- Determine the sex of a face

Classification and regression are forms of *supervised learning* (we have data on (x, y) pairs); clustering is a form of *unsupervised learning* (we have data only on x's).

Note: While neural networks were originally developed to simulate the architecture of the brain (How does the brain work? Can we get a machine to think like a brain?), noone seriously considers that aspect of things anymore. Neural networks are just black-box predictors.

More

What are the issues?

- Form the outputs
- Form the neural network architecture
- Train/estimate/fit the thing
- Estimate prediction error honestly
- Figure out what's going on inside the black box.

Regression

You might take ϕ_0 to be the identity, and use only one output.

Classification

If the outcome set is $\{1, 2, \ldots, k\}$, you might use binary outputs with $j \to y = (0, \ldots, 0, 1, 0, \ldots, 0)$ (i.e., the y corresponding to outcome j has $y_i = \delta_{ij}$).

ϕ_0 may be a threshold function or a logistic.

Note: Some people work hard to create fancy architectures tailored to their particular problem. Others just use the basic form I gave above, playing with the number of hidden nodes and maybe dropping some connections.

Training/Fitting/Estimating

Gather n data points (x, y).

Form some function to optimize, say $\sum_i [y_i - \hat{y}(x_i; \theta)]^2$

Method of fitting: "error backpropagation" (basically a gradient descent algorithm).

The weird bit: Since the model is *way* overparameterized, we don't actually want to minimize the RSS for our data.

We need to either add some sort of penalty or (this is the weird bit), halt the fitting procedure *before* going into a local mode.

The procedure:

- Set aside a test set
- Stop the fitting procedure at various times
- Check out the prediction on the test set at each stopping point
- Pick the stopping point with the minimal prediction error

Another problem: starting points. You can't really try more than one (or two).

Summary

Robert Tibshirani's comment on Cheng and Titterington:

What the statistician can learn from neural network researchers:

1. We should worry less about statistical optimality and more about finding methods that work, especially with large data sets.
2. We should tackle difficult real problems like some of those addressed by neural network researchers.
3. Models with very large numbers of parameters can be useful for prediction, especially for large data sets and problems exhibiting high signal-to-noise ratios.
4. Modelling linear combinations of input variables can be a very effective approach because it provides both feature extraction and dimension reduction.
5. Iterative, nongreedy fitting algorithms can help to avoid overfitting in models with large numbers of parameters.
6. We (statisticians) should sell ourselves more.

What the neural network research can learn from statisticians

1. They should worry more about statistical optimality or at least about the statistical properties of methods.
2. They should spend more effort comparing their methods to simpler statistical approaches. They should not use a complicated model where a simple one will do.