IEMS 304 Lecture 6: Trees and Neural Networks

Yiping Lu
yiping.lu@northwestern.edu

Industrial Engineering & Management Sciences
Northwestern University

NORTHWESTERN
UNIVERSITY

Today

① We need more parameters when #data ↑

② We can approximate any smooth function using a parametric form

example. polynomials (the degree can be ↑ when #data ↑)

Universal Approximation Theorem.

Non-parametric Models can also have parameters

Recall. a statistical procedure is of a nonparametric type if it has properties which are satisfied to a reasonable approximation when some assumptions that are at least of a moderately general nature hold.

The assumption we made is "the function is smooth"

if x and y are similar, then f(x) and f(y) are also similar.

Neural Network

⇒ multinomial regression.

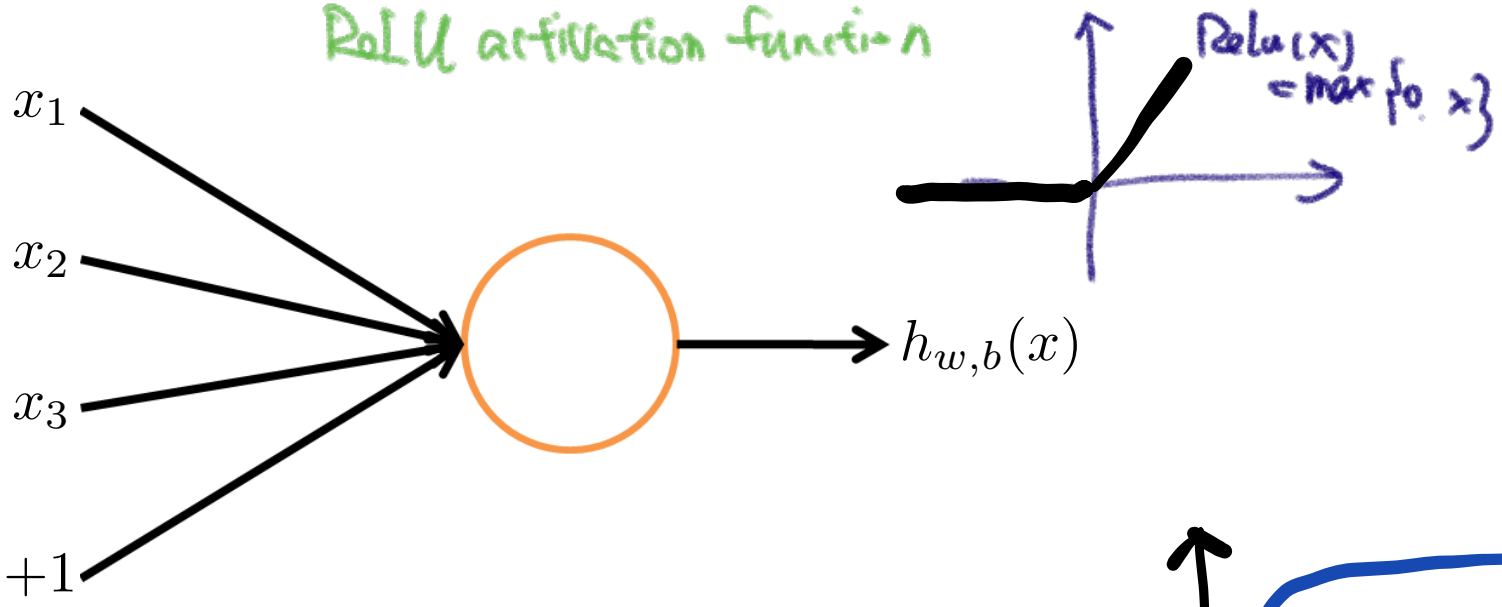
$$f(x) = f(\theta, x)$$

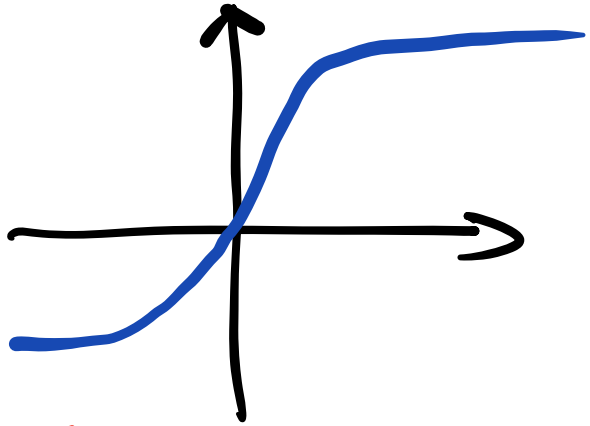
↑ parameter

Neural Networks need Computational Resource

- Clever original idea and memorable name — became very popular in the 1980s and 1990s.
- They have evolved to have less resemblance to how the human brain processes information (but better effectiveness at modeling nonlinear relationships in complicated data sets).
- To fit a neural network model (and all of the other black-box models), the training data must be available in the same format as for linear/logistic regression:
  - A 2D array of observations.
  - Each column is a different variable; each row a different case.
  - One column is the response variable and the other columns are any number of predictor variables.
  - The neural network hidden variables (h's) are internal variables that you do not enter or even care about.

Basic Building Block: Neuron





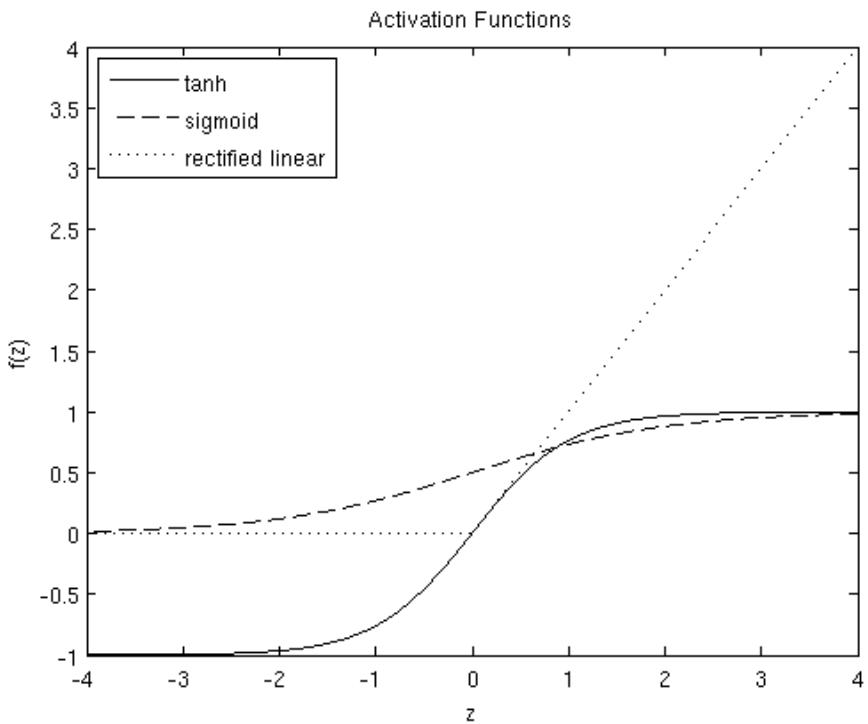
- Input: $\mathrm{x} = [x_1, x_2, x_3, +1]^\top$
- Output: $h_{\mathrm{w},b}(x) = \sigma(\mathrm{w}^\top \mathrm{x}) = \sigma(\sum_{j=1}^{3} w_j x_j + b)$
- Activation Function $\sigma : \mathbb{R} \to \mathbb{R}$

choose any activation function like this

(first NN use the activation function but is not popular now.

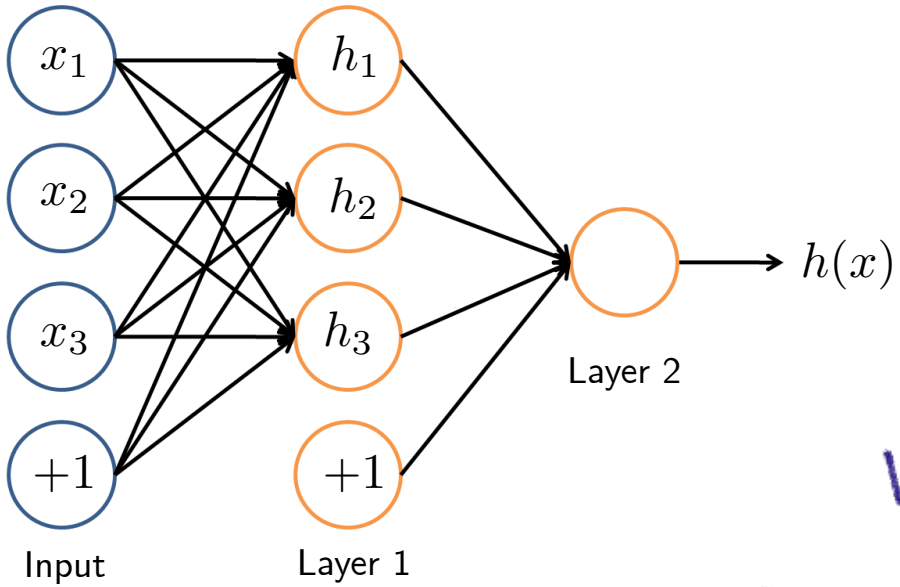
Activation Function

- Sigmoid (logistic) function: $\sigma(z) = \frac{1}{1+\exp(-z)}$
- tanh function: $\sigma(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$
- ReLU function (most commonly used in deep learning): $\sigma(z) = \max\{0, z\}$



Two-Layer Neural Network

Consider supervised Learning: $\mathrm{x} = (x_1, x_2, x_3, +1) \rightarrow y$



$$Wx + b$$

$$W_e \begin{bmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \\ W_{31} & W_{32} & W_{33} \end{bmatrix}$$

- Hidden Units:

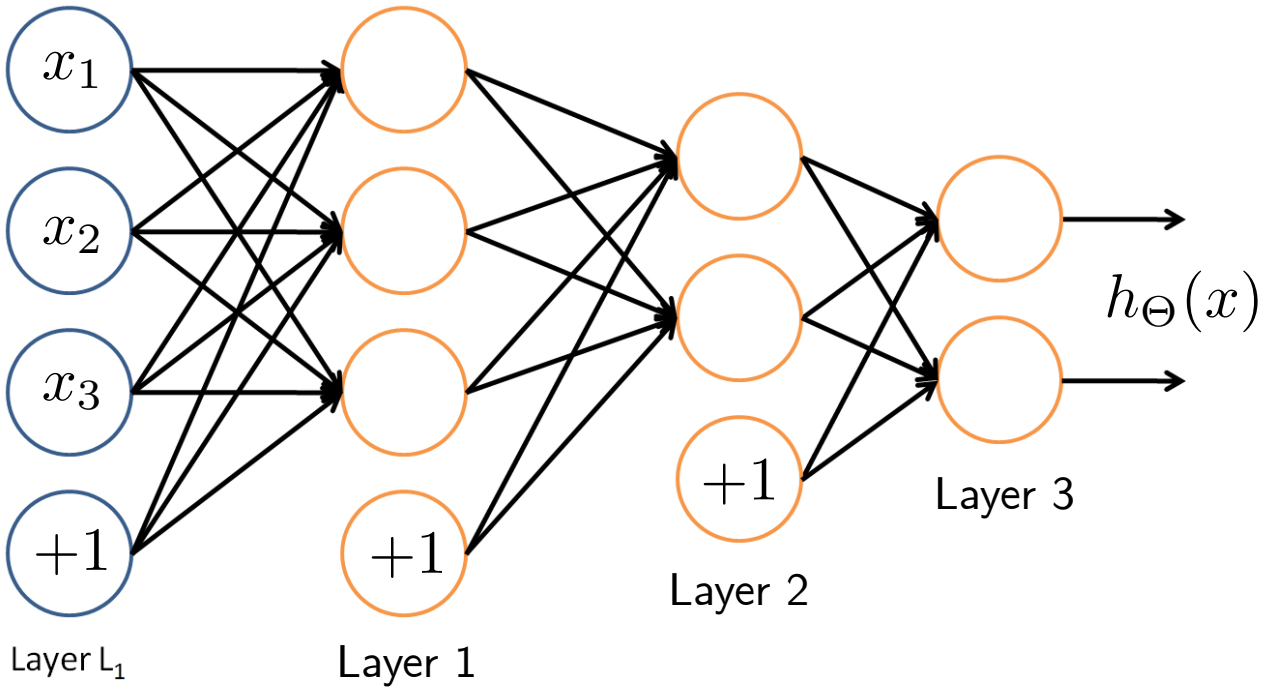
$$h_1 = \sigma(w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b_{11})$$

$$h_2 = \sigma(w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + b_{12})$$

$$h_3 = \sigma(w_{31}x_1 + w_{32}x_2 + w_{33}x_3 + b_{13})$$

- Output: $h(\mathrm{x}) = \sigma(v_1 h_1 + v_2 h_2 + v_3 h_3 + b_2)$

In a Matrix Form



$$h_\theta(x) = W_3\sigma(W_2\sigma(W_1x + b_1) + b_2) + b_3$$

$\theta =$ $W_1, b_1, W_2, b_2, W_3, b_3$,

Fitting A Neural Network Model

- Standardize predictors via

$$x_{ij} = \frac{x_{ij} - \bar{x}_j}{s_{x_j}},$$

  where $\bar{x}_j$ and s_{x_j} are sample mean and standard deviation of the j-th predictor variable.

- Also standardize the response. Or instead, if using sigmoid output activation function, scale response to interval $[0, 1]$ via

$$y_i = \frac{y_i - y_{\text{min}}}{y_{\text{max}} - y_{\text{min}}}.$$

  Why do we need to do this rescaling for a sigmoid output activation function?

Fitting A Neural Network Model Cont'd

- Choose network architecture
  - number of hidden layers,
  - number of neurons in each hidden layer,
  - output activation function (usually linear or logistic),
  - other options and tuning parameters.
- Software estimates parameters to minimize (nonlinear LS with shrinkage):

$$\sum_{i=1}^{n}(y_i - h_{\boldsymbol{\theta}}(\mathrm{x}_i))^2 + \lambda \cdot \texttt{regularization}(\boldsymbol{\theta}).$$

  Regularization is often based on heuristics and λ is a user-chosen parameter.

How to find the best parameter?

Gradient Descent for training neural networks!

We don't use Newton Methods

(at least 2025)

Chain Rule Recap

ex. $g(x) = 2x$ $\quad f(g(x)) = \sin(2x)$

$f(x) = \sin(x)$

Chain Rule Reminder

For a composite function $f(g(x))$, the chain rule states that

$$\frac{df}{dx} = \frac{df}{dg} \cdot \frac{dg}{dx}.$$

the second layer — first layer

In neural networks, this principle is applied layer by layer during backpropagation.

Forward Pass in a Simple Neuron

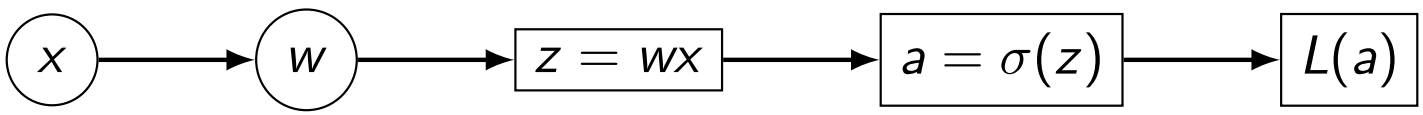
Model Description:

Consider a neuron with the following computations:

$$z = wx, \quad a = \sigma(z), \quad L = L(a),$$

where $\sigma(z)$ is the activation function.

Diagram of the Forward Pass:



Backpropagation: Computing Gradients

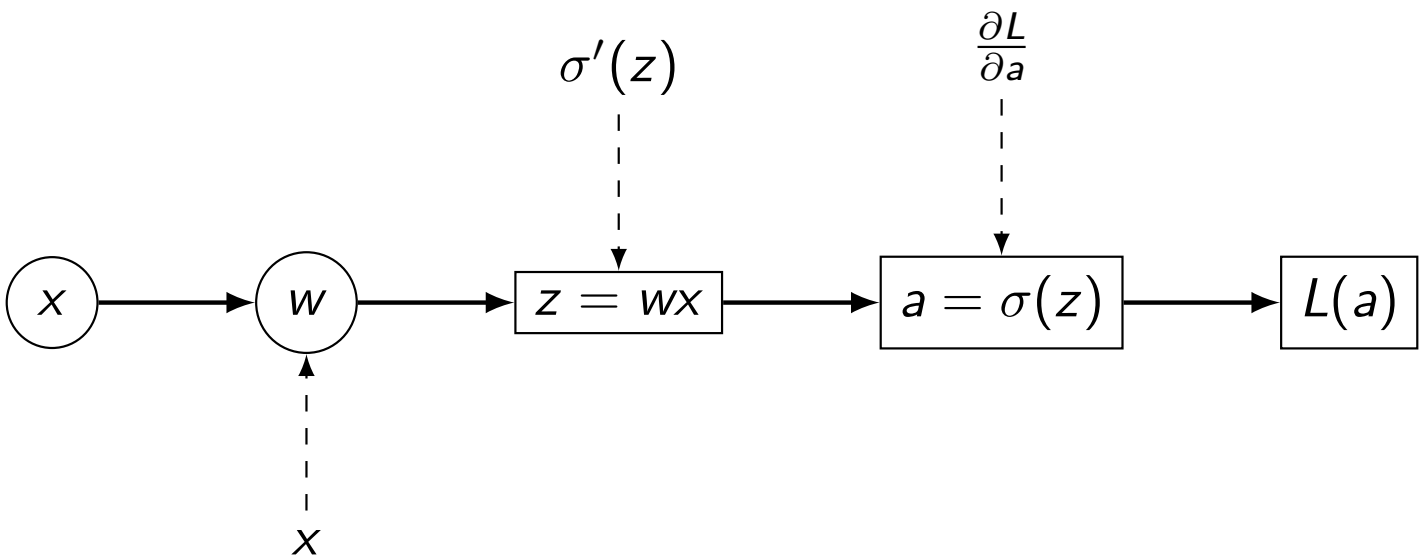
Gradient Computation:

To update the weight w, the gradient is computed as:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w}.$$

Illustrative Diagram:

The dashed arrows represent the flow of gradients during backpropagation.



Backpropagation in Multiple Layers

Multi-layer Neural Network:

Consider a network with one input layer, two hidden layers, and one output layer:

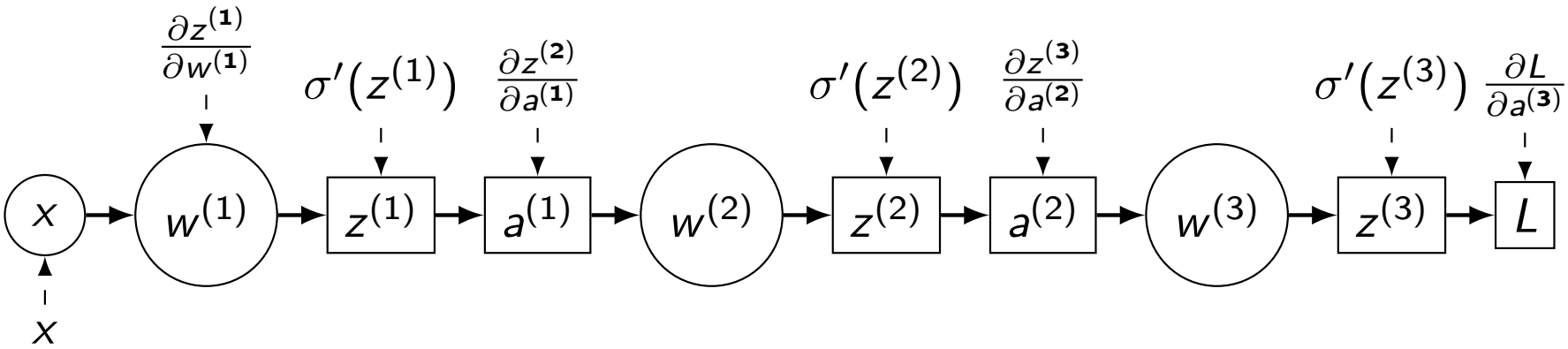
$$x \to z^{(1)} \to a^{(1)} \to z^{(2)} \to a^{(2)} \to z^{(3)} \to a^{(3)} = L.$$

Chain Rule Application:

For instance, the gradient for weight $w^{(2)}$ in the second layer is:

$$\frac{\partial L}{\partial w^{(2)}} = \frac{\partial L}{\partial a^{(3)}} \cdot \frac{\partial a^{(3)}}{\partial z^{(3)}} \cdot \frac{\partial z^{(3)}}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial w^{(2)}}.$$

Backpropagation



Gradient Vanishing Problem

(Exploding)

What is Gradient Vanishing?

In deep neural networks, the gradients are computed by multiplying many small derivatives. When these values are less than 1, their product may shrink exponentially as they propagate backwards through the layers.

If grad > 1, then it's exploding

If grad < 1, then it's Vanishing

Mitigation Techniques:

- Use activation functions such as ReLU.
- Apply normalization methods (e.g., batch normalization).
- Use network architectures like residual networks (ResNets) to improve gradient flow.

Layer Normalization

Layer normalization normalizes the inputs across the features of a single data sample.

For a vector of activations $\mathrm{x} = (x_1, x_2, \ldots, x_n)$, it is computed as:

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}},$$

where μ and σ^2 are the mean and variance of x, and ϵ is a small constant for numerical stability.

Why this helps mitigate back propogation?

Residual Connections in Neural Networks

Residual connections allow a neural network to learn an identity mapping by adding the input directly to the output of a set of layers.

Motivation: They help mitigate the vanishing gradient problem, enabling the training of very deep networks.

Key Idea: Instead of learning a direct mapping, the network learns the residual:

$$\mathrm{y} = \mathcal{F}(\mathrm{x}, \{W_i\}) + \mathrm{x}$$

where:

- x is the input,
- $\mathcal{F}(\mathrm{x}, \{W_i\})$ represents the residual mapping.

Why NN works and Why it is non-parametric ?

$y = f(x) + \varepsilon$, we make no assumption on y!

f is a smooth function.

Universial Approximation Theorem

If f is a smooth function. ε is arbitrary Constant >0 → approximation accuracy you want to achieve.

Then there exists a large enough (two-layer) Neural Network can approximate function f to ε-accuracy

Examples. Fact 1 Neural Network can be more powerful than linear regression.

$$\underbrace{\text{relu}(-w \cdot x)}_{\text{Neural}} + \underbrace{\text{relu}(w \cdot x)}_{\text{Neuron 2}} = w \cdot x$$

Approximation Power (or the functions that NN can represent) $\Rightarrow$ NN

Regression Tree

Structure of a Regression Tree

A final fitted regression tree model divides the predictor (x) space by successively splitting into rectangular regions and modeling the response (Y) as constant over each region.

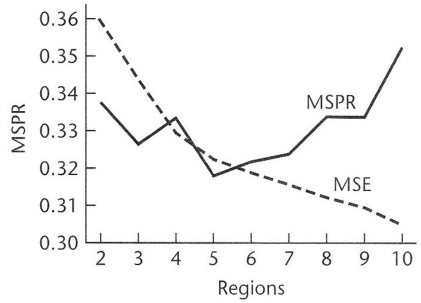
This can be schematically represented as a "tree":

- each interior node of the tree indicates on which predictor variable you split and where you split.
- each terminal node (a.k.a. leaf) represents one region and indicates the value of the predicted response in that region.

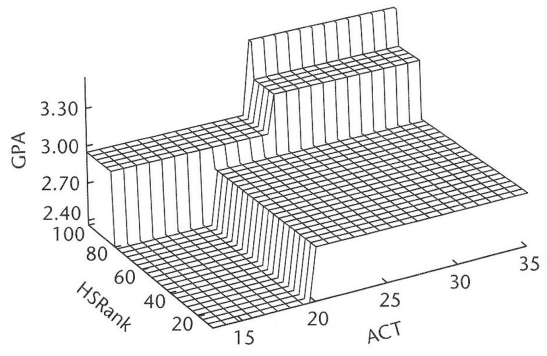
To use a fitted regression tree to predict a new case, you start at the root node and follow the splitting rules down to a leaf.

Illustration of Regression Tree

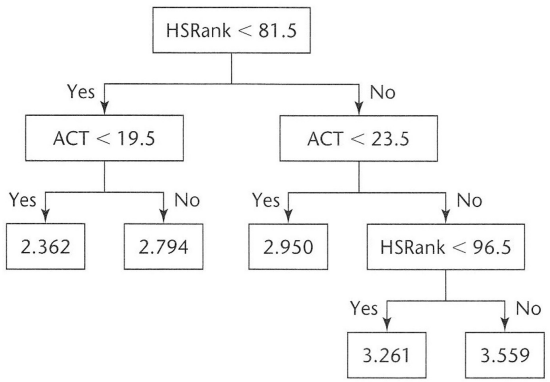
The following slide illustrates a fitted tree model for an example, in which the objective is to predict college GPA (the response) as a function of highschool rank and ACT score (two predictors).



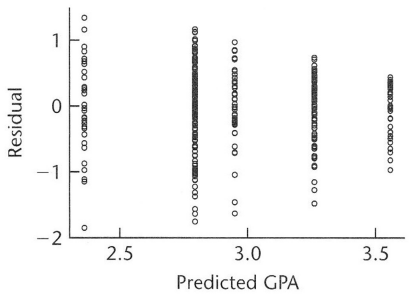
(a)



(b)



(c)



(d)

Mathematical Representation of Regression Tree

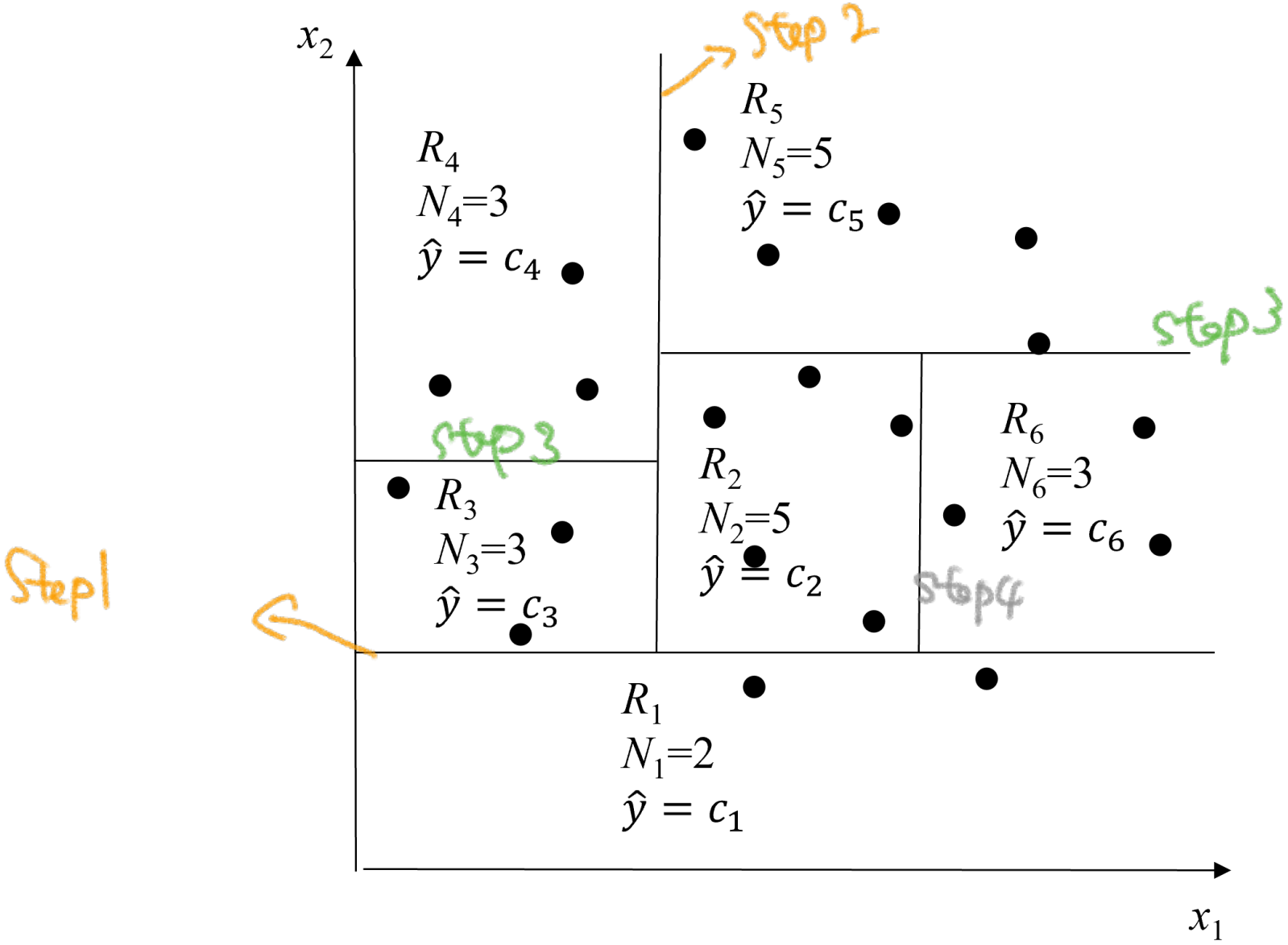
We can denote a tree model as

$$y = \sum_{m=1}^{M} c_m \mathbb{1}\{x \in R_m\},$$

where

- M: total number of regions.
- R_m: m-th region.
- $\mathbb{1}\{x \in R_m\}$: indicator function $= \begin{cases} 1 & x \in R_m \\ 0 & x \notin R_m \end{cases}$.
- c_m: constant value over R_m.

Regression Tree



Questions and Discussions

piece-wise constant

- What kind of functional $x \to Y$ relationships can you capture with a regression tree model structure?
- Can a regression tree represent a linear relationship? Can it represent a linear relationship as efficiently as a neural network?

can not represent a linear function exactly, but can approximate only if.

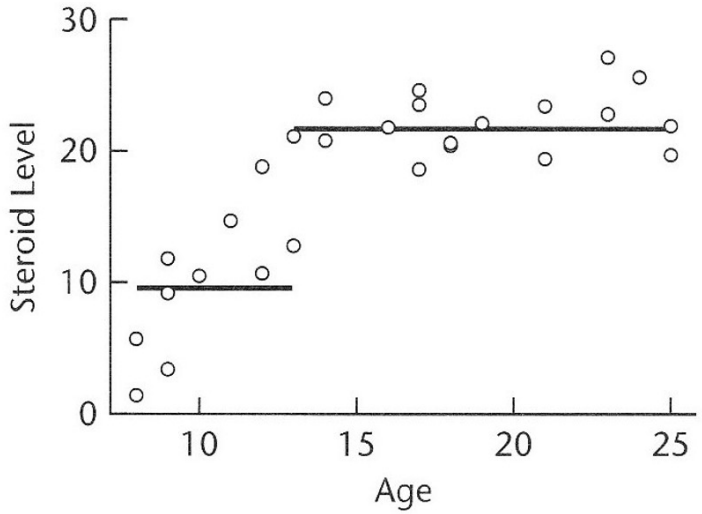
- Which type of model — neural network or regression tree —
  - is more interpretable?
  - is easier to fit?
- Given a specified set of regions, how would you estimate the coefficients $\{c_m : m = 1, 2, \ldots, M\}$?

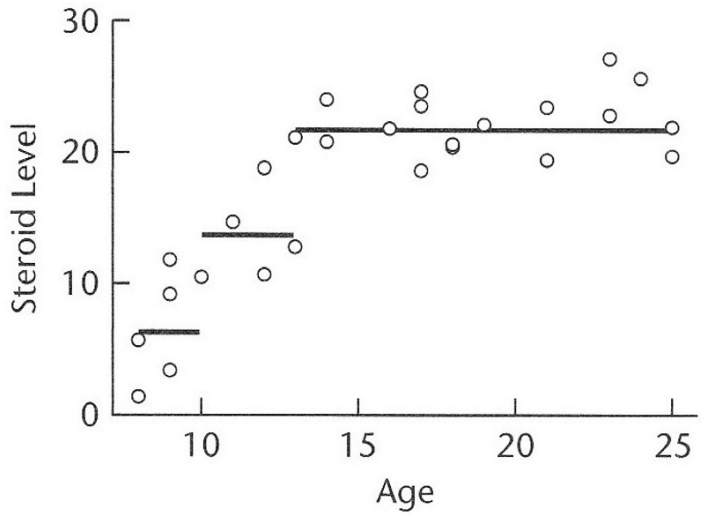
the mean of the data lies in region R_m.

No! NN can

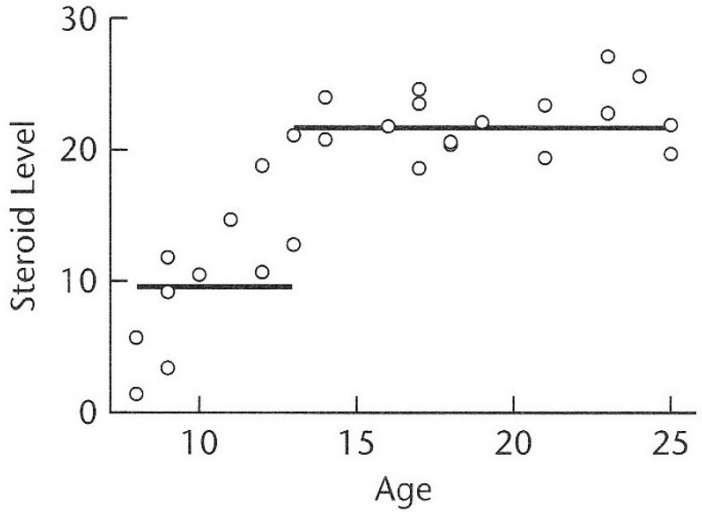
Fitting a Regression Tree

- Fitting the model entails growing the tree one node at a time.
  - At each step, the single best next split (which predictor and where to split) is the one that gives the biggest reduction in SSE.
  - The fitted or predicted response over any region is simply the average response over that region. The errors used to calculate the SSE are the response values minus the fitted values.
  - Stop splitting when reduction in SSE with the next split is below a specified threshold, all node sizes are below a threshold, etc.
  - Most algorithms overfit then prune back branches.

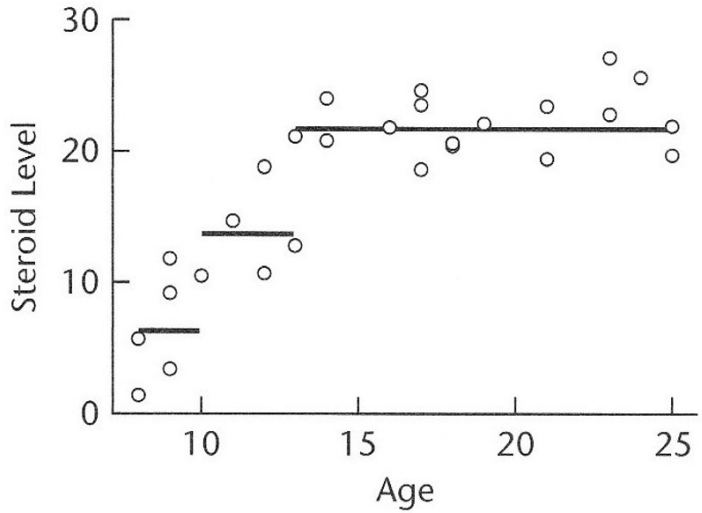




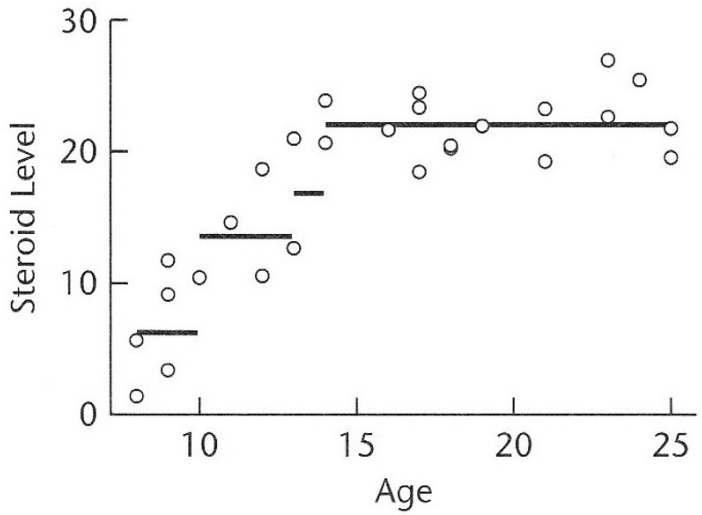
Example of Constructing a Tree



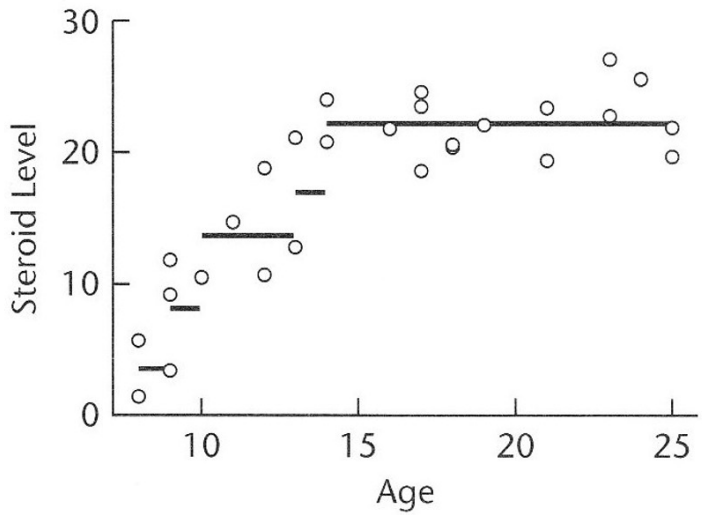
(a)



(b)

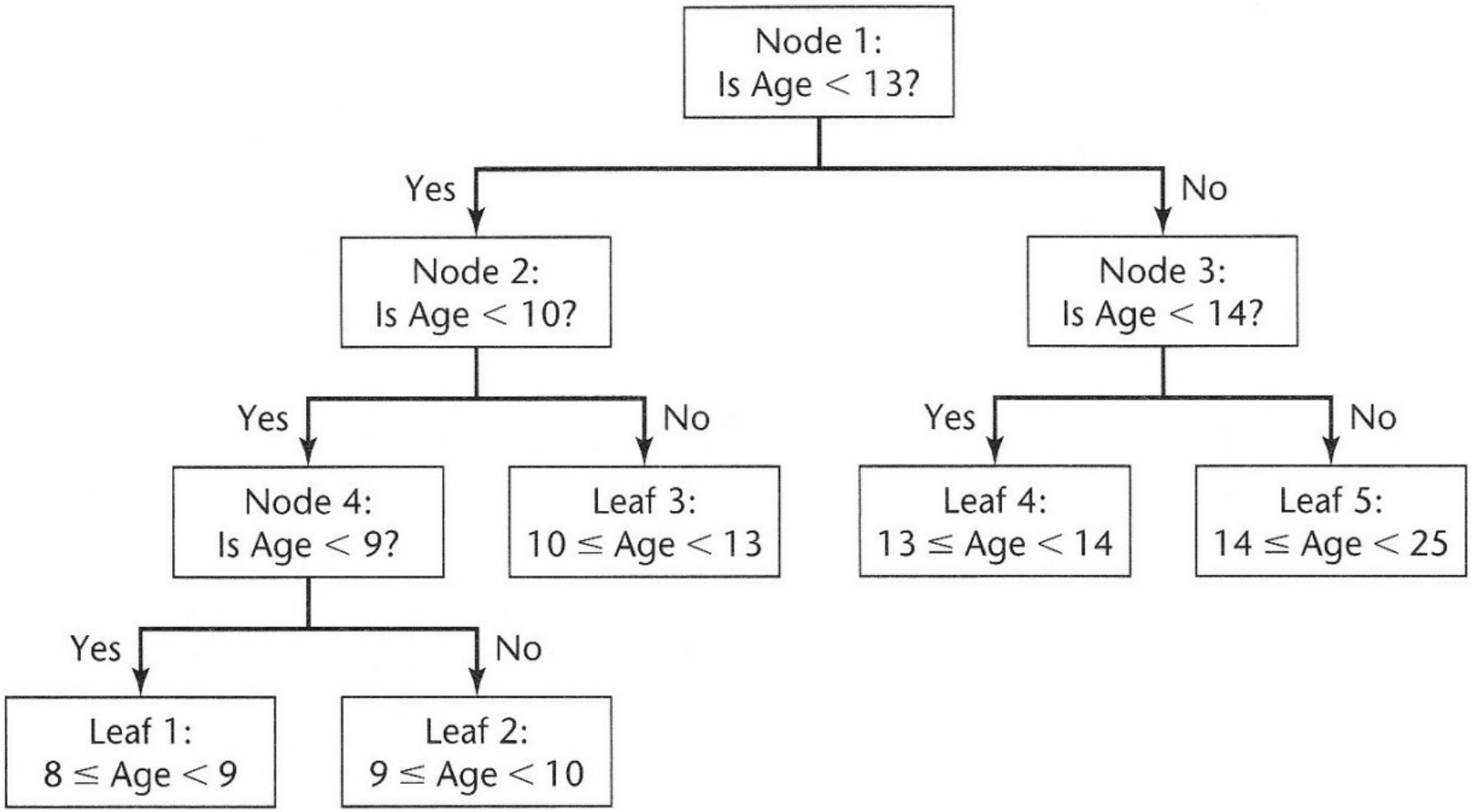


(c)



(d)

The Constructed Regression Tree



How to Calculate SSE

- For a given set of splits $R_1, \ldots, R_M$, we have

$$\hat{c}_m = \text{average}\{y_i | \mathrm{x}_i \in R_m\} = \frac{1}{N_m} \sum_{\mathrm{x}_i \in R_m} y_i,$$

$$N_m = \text{number of points } \mathrm{x}_i \in R_m,$$

$$\text{SSE} = \sum_{i=1}^{n} (y_i - \hat{y}_i) = \sum_{m=1}^{M} \sum_{\mathrm{x}_i \in R_m} (y_i - \hat{c}_m)^2.$$

- Note that for $\mathrm{x}_i \in R_j$, $\hat{y}_i = \hat{c}_j$.

Pruning Branches (Model Selection)

- Pruning a branch means that we collapse one of the internal nodes into a single terminal node.
- Pruning the tree means that we prune a number of branches.
- Pruning algorithms in software will usually optimally prune back a tree in a manner that minimizes $\mathrm{SSE} + \lambda M$, where M (complexity measure) and SSE are for the pruned tree. The best value for λ is determined via CV.
  > If the compleixty measure is the number of leaves, the tree's depth is d, then what is the M?

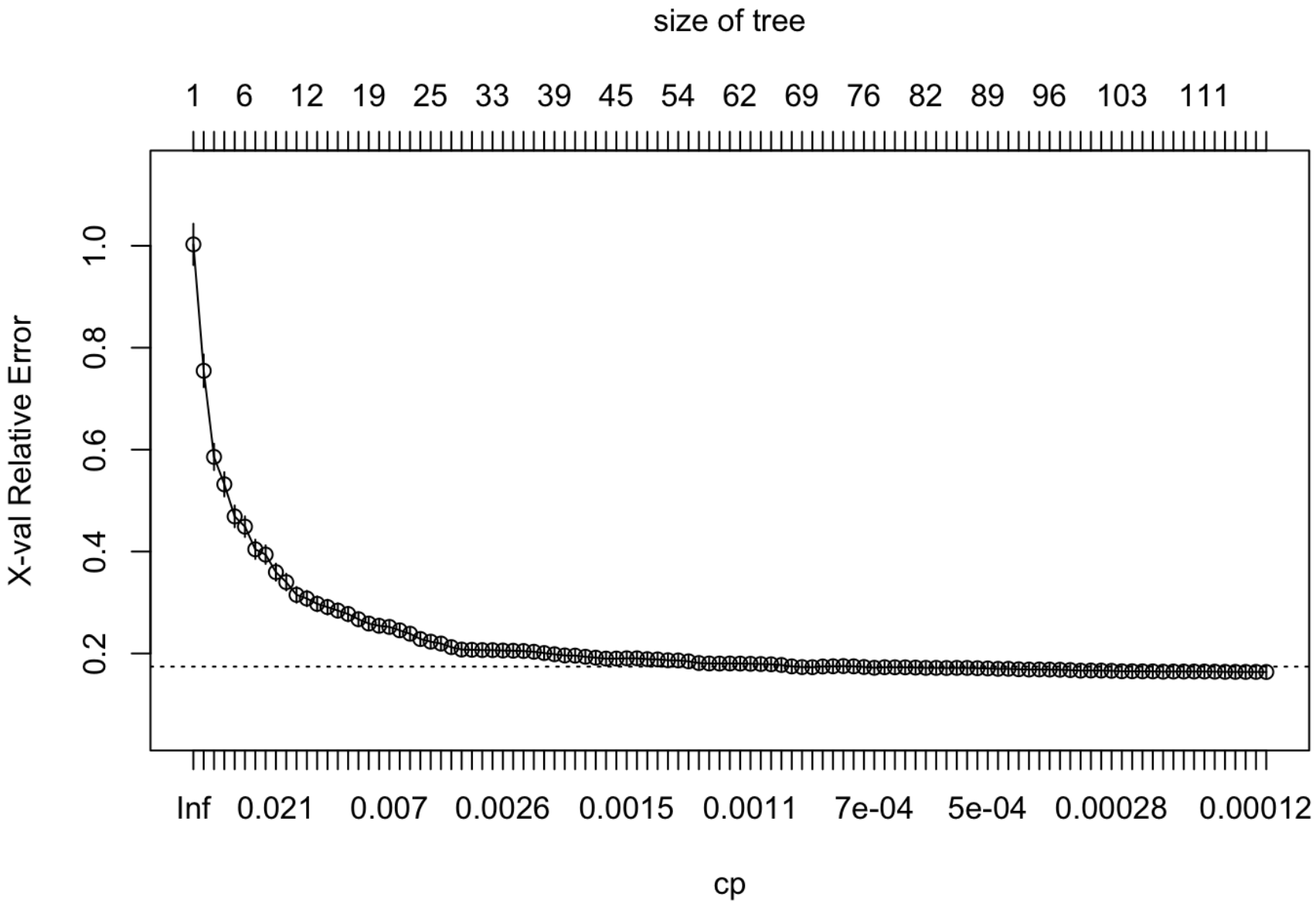
  the number of leaves
- There is a nice computational trick ("weakest link pruning") that allows this optimal pruning to be done very fast.

Example: Predicting Strength in Concrete Data

- Implementing a regression tree in R uses rpart library. Useful commands are rpart.control(), rpart(), prune(), etc.

```
library(rpart)
control<-rpart.control(minbucket = 5, cp = 0.0001,
                       maxsurrogate = 0, usesurrogate = 0, xval = 10)
CRT.tr<-rpart(Strength ~ .,CRT, method = "anova", control = control)
plotcp(CRT.tr)  #plot of CV r^2 vs. size
printcp(CRT.tr)  #same info is in CRT.tr$cptable
#prune back to optimal size, according to plot of CV 1-r^2
CRT.tr1<-prune(CRT.tr, cp=0.0015)  #approximately the best size pruned
```

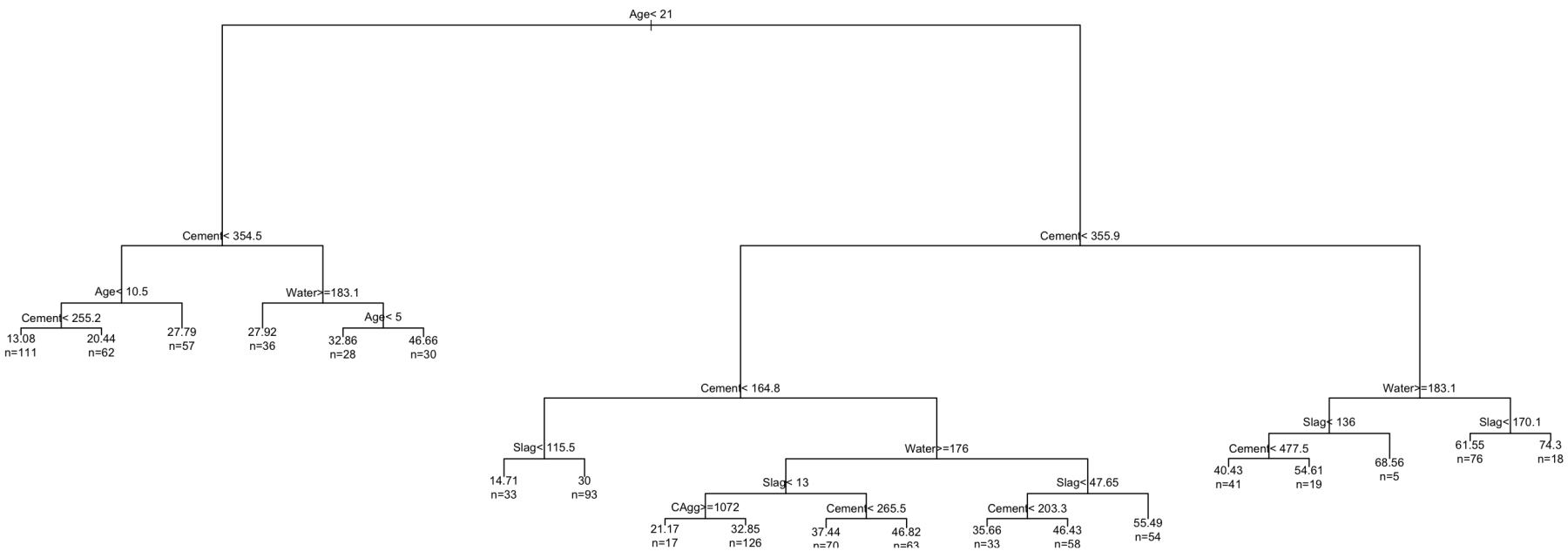
Example: Error v.s. Size of Tree



Example: Interpreting Error v.s. Tree Size

- The horizontal broken line is the lowest $1 - r^2_{\text{cv}}$ plus one SE.
- A common strategy is to choose the best size of the tree, or equivalently the best complexity parameter cp, as the left-most value below the horizontal broken line (around size = 50 in the previous slide).
- Or simply choosing the best size or cp as the one with minimum $1 - r^2_{\text{cv}}$ usually gives the model with the best predictive power (albeit larger than using the rule above).

Example: The Pruned Tree



Numerical Assessment of Variable Importance

- For a visual assessment of the importance of each predictor in a tree, inspect the tree graph. The importance of x_j is reflected by how many times it appears in internal nodes, how close they are to the root node, and the length of the branch for that split.
- For a numerical measure of the importance of x_j, sum the reductions in deviance for each internal node for which the split is based on the same predictor x_j.
- The "deviance" is $-2\log(\texttt{likelihood})$. For a nonlinear regression model with normal errors, the deviance is SSE. Thus, we can get the deviance from the \$frame object of the rpart output or from a textual print of the tree.

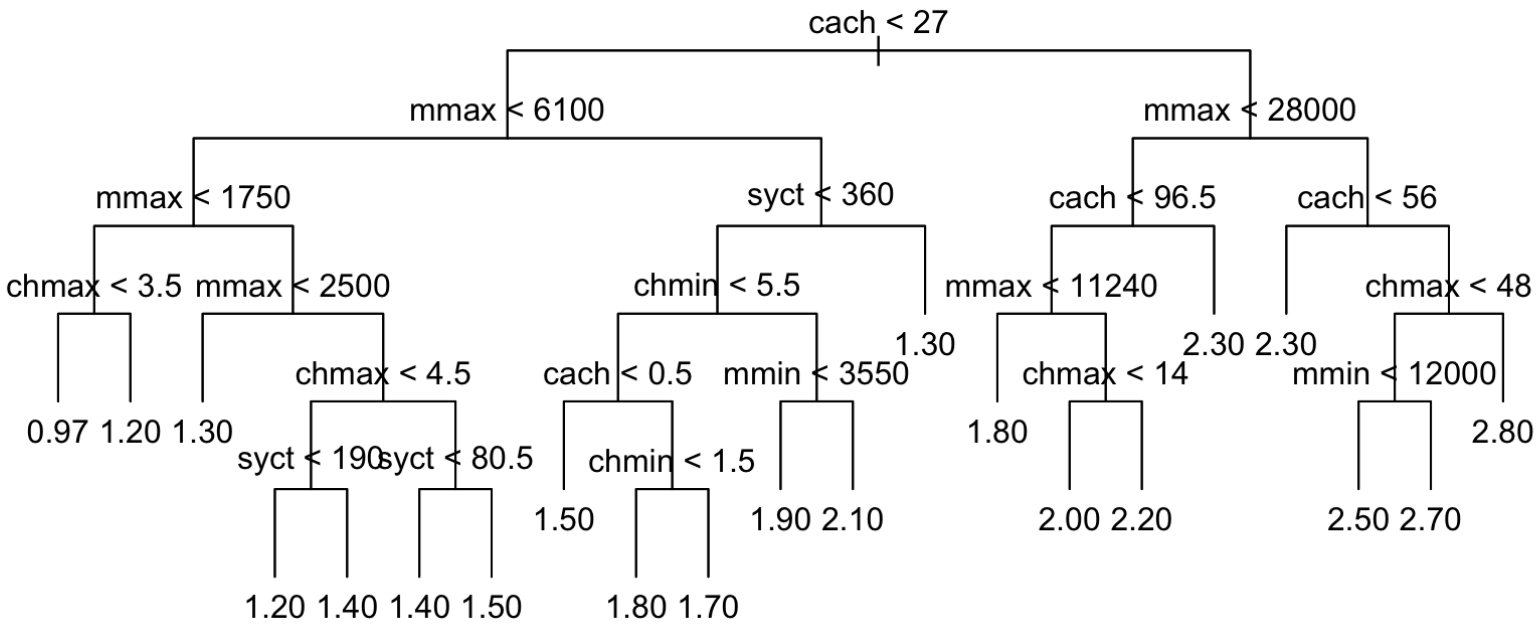
Alternative R Package

- We can also use tree package to fit a regression tree.
- Works very similar to rpart() command.

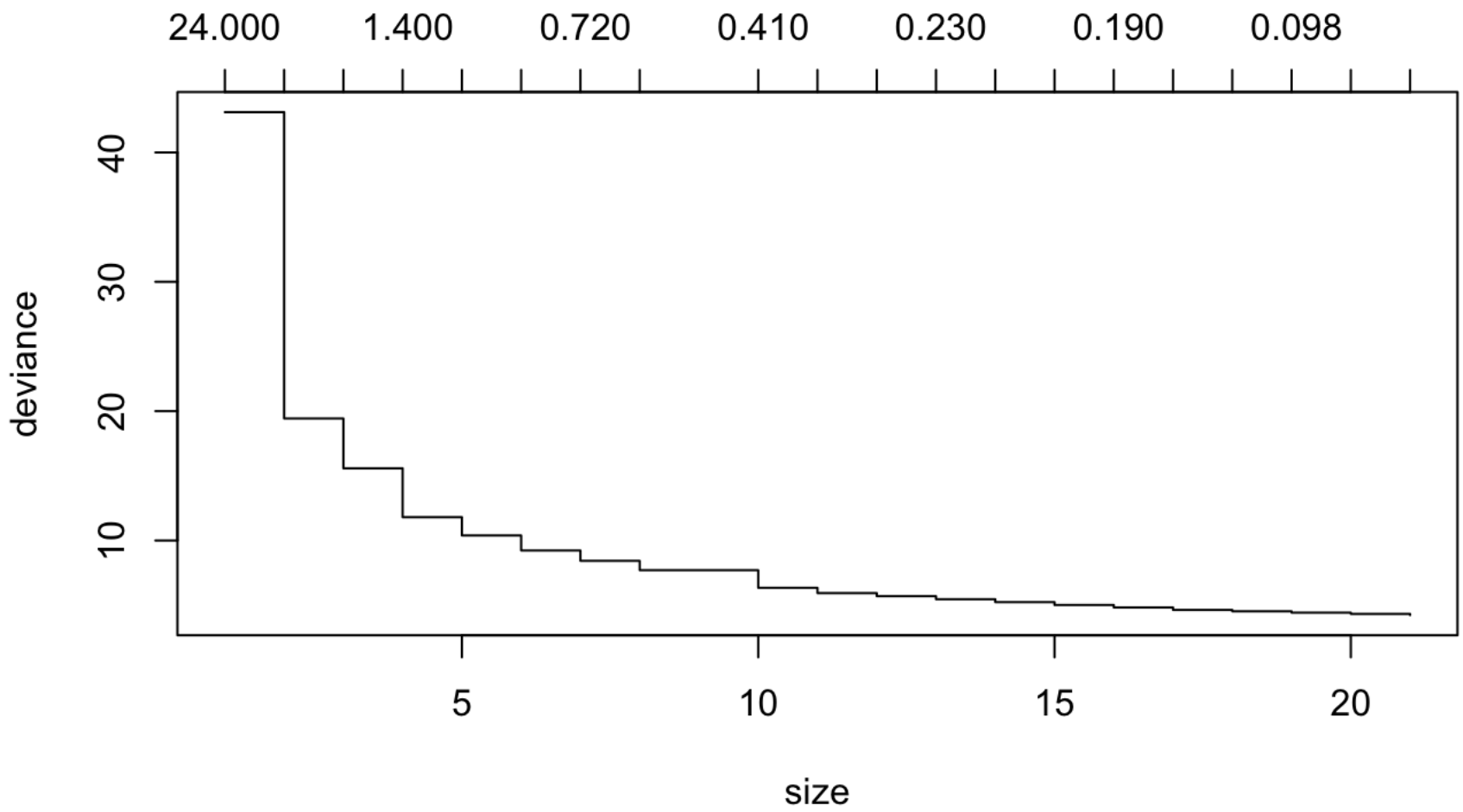
```
library(tree)
control = tree.control(nobs=nrow(CPUS), mincut = 5,
                       minsize = 10, mindev = 0.002)
#default is mindev = 0.01, which only gives a 10-node tree
cpus.tr <- tree(log10(perf) ~ .,CPUS[2:8],control=control)
cpus.tr
summary(cpus.tr)
```

Example: Predicting CPU Performance

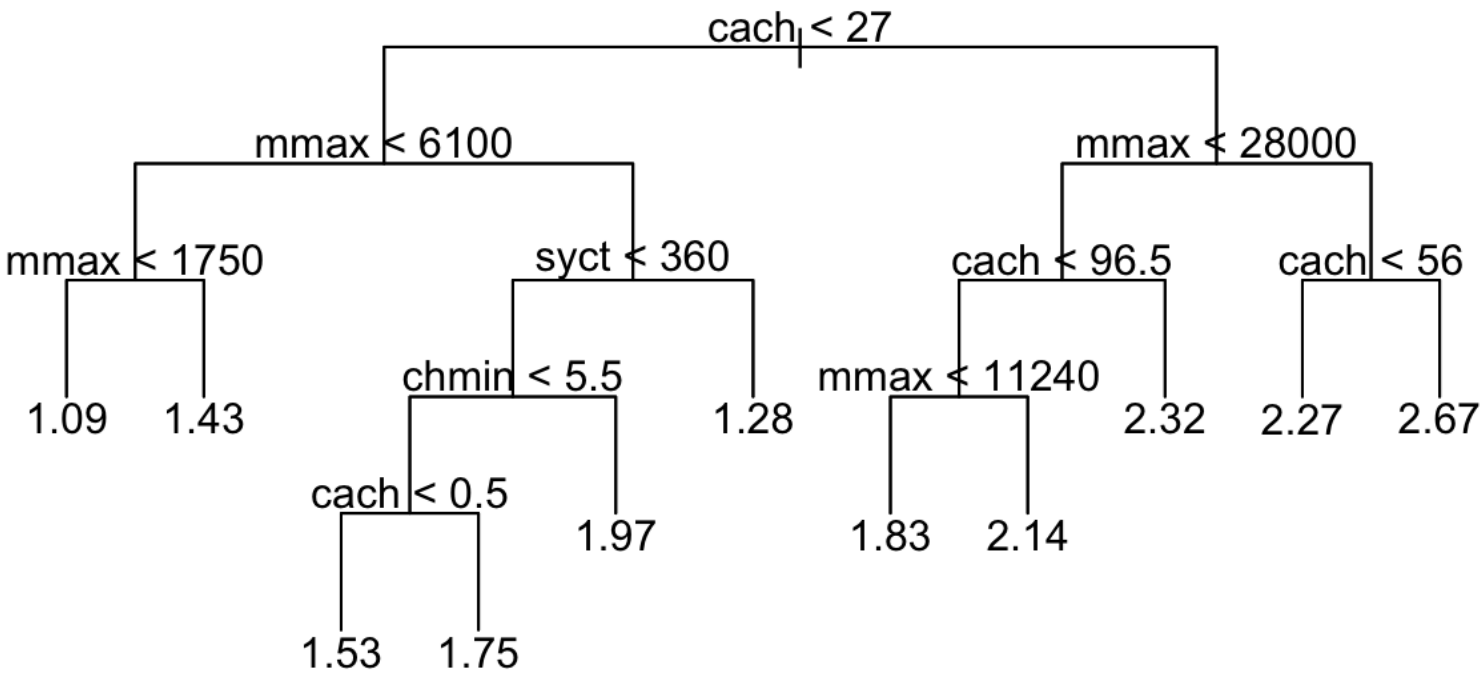
- We use 7 predictor variables to predict the performance measure of CPUs.
- An "overly large" tree is plotted as follows.



Example: Prune the Tree



Example: Best Tree



Questions and Discussions

- What is the best size tree for the CPUS example?
- Provide an interpretation of which predictor variables are most important.
- Do there appear to be any interactions between mmax and cach?
- Why must minsize be at least twice mincut?
- To grow the initial tree larger (it is better to overgrow the initial tree, then prune it back) you can decrease minsize, mincut and/or mindev.

Classification Tree

Classification Tree Overview

- Fitting and using classification trees with a K-category response is similar to fitting and using regression trees.
- For classification trees, we model $p_m(\mathrm{x}) = \mathbb{P}[Y = m|\mathrm{x}](m = 1, 2, \ldots, M)$ as constant over each region. (Majority Vote)
- Compare to regression trees, for which we model $\mathbb{E}[Y|\mathrm{x}]$ as constant over each region.
- At each step in the fitting algorithm, the best next split is the one that most reduces some criterion measuring the "impurity" within the regions.

Some Technical Details

- In the region R_m, the fitted class probabilities and best class prediction are

$$\hat{p}_{m,k} = \frac{1}{N_m} \sum_{x_i \in R_m} \mathbb{1}(y_i = k),$$

$$k_m = \arg\max_k \hat{p}_{k,m}.$$

- Some common impurity measures:
  - misclassification error: $\sum_{m=1}^{M} \sum_{x_i \in R_m} \mathbb{1}(y_i \neq k_m)$;
  - Gini index: $\sum_{m=1}^{M} N_m \sum_{k=1}^{K} \hat{p}_{m,k}(1 - \hat{p}_{m,k})$;
  - deviance: $-2 \sum_{m=1}^{M} N_m \sum_{k=1}^{K} \hat{p}_{m,k} \log(\hat{p}_{m,k})$.

CART (Classification and Regression Tree) Algorithm

Call R_0 the entire space. Consider the first step:

- Define regions R_1, R_2 from splitting on variable j at value s:

$$R_1 = \{X \in \mathbb{R}^p : X_j \leq s\}, \quad R_2 = \{X \in \mathbb{R}^p : X_j > s\}$$

- We **greedily** choose j, s by minimizing the misclassification error

$$\arg\min_{j,s}([1 - \widehat{p}_{c_1}(R_1)] + [1 - \widehat{p}_{c_2}(R_2)]),$$

  where c_1, c_2 are the most common class in R_1, R_2 respectively.

The general algorithm then repeats steps (1) and (2) for every new region.

Example to Illustrate the Notations

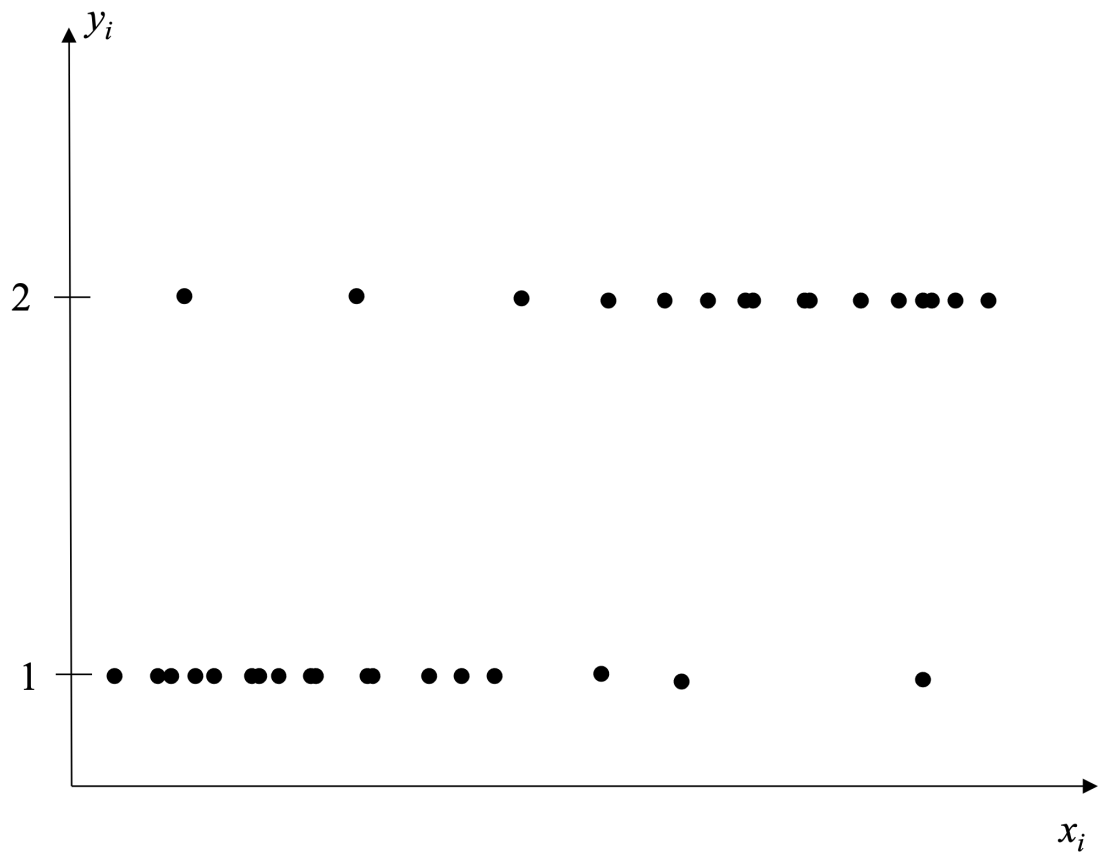
- Suppose we have $K = 4$ classes, and the predictors for $N_m = 100$ training cases fall into a particular region R_m. For those 100 cases, suppose we have the following breakdown of the number of cases with response value that fell into the four categories.

Class, k	**# obsvns with Y in Class k**	$\hat{p}_{m,k}$
1	10	
2	20	
3	65	
4	5	

- What is $\hat{p}_{m,k}$ for $k = 1, \ldots, 4$?
- What is k_m?

Another Example for Illustration

- Suppose we only have $K = 2$ classes and the training data is as follows. Where would the first split that minimizes the misclassification rate be?

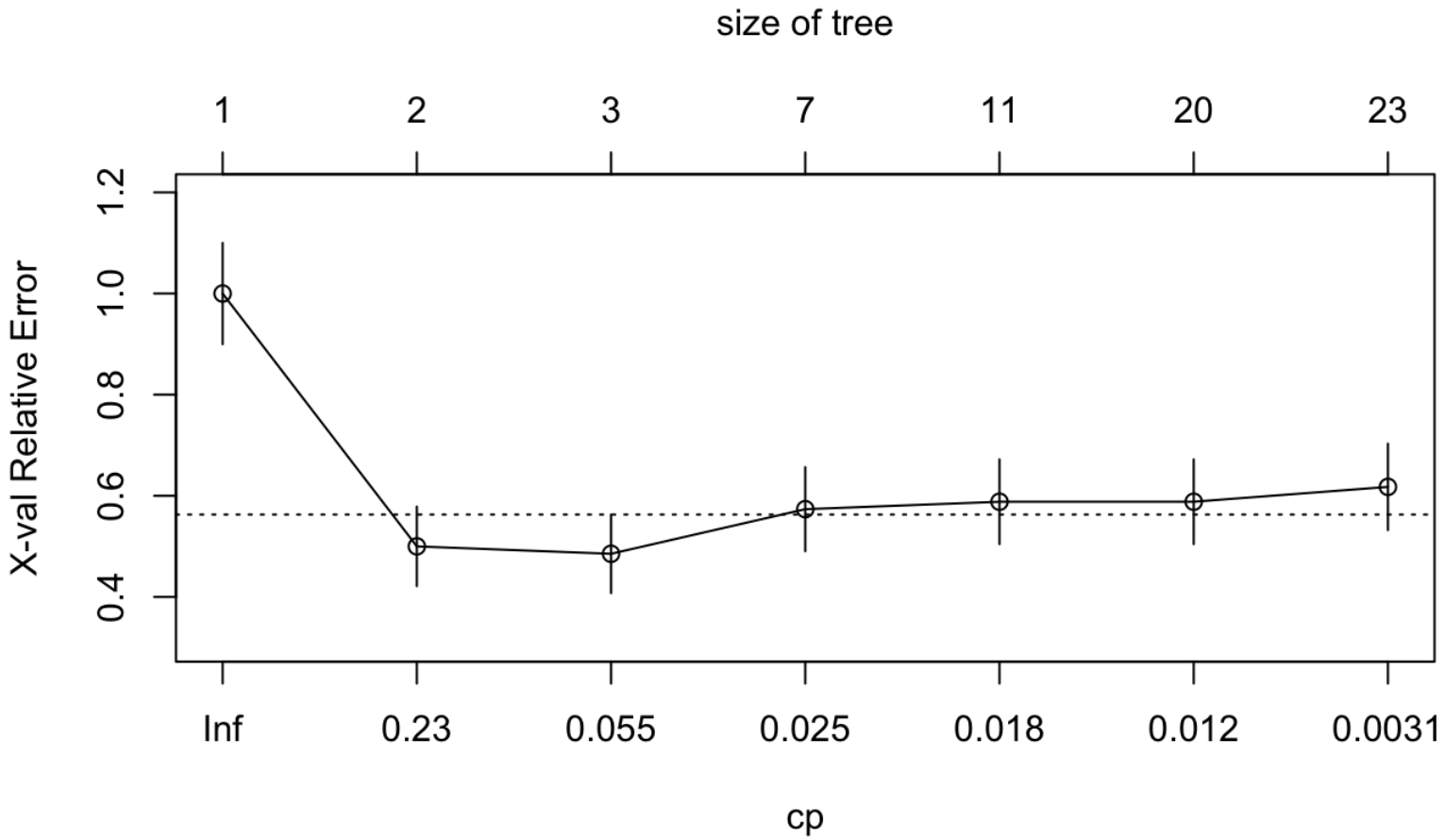


Example: Predicting Glass Type

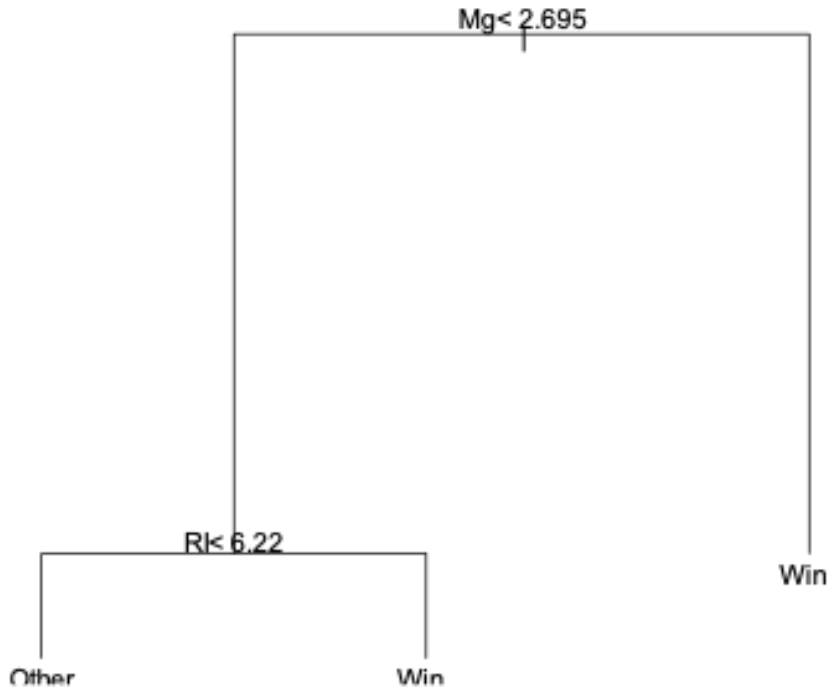
- Data in `fgl.txt` are the same as the FGL data in the MASS package.
- There are 214 cases, with 9 predictor variables and a categorical response.
- Each row contains the results of an analysis of a fragment of glass
- "type" is the response, one of six different glass types: window float glass (WinF: 70 rows), window non-float glass (WinNF: 76 rows), vehicle window glass (Veh: 17 rows), containers (Con: 13 rows), tableware (Tabl: 9 rows) and vehicle headlamps (Head: 29 rows).
- Eight of the predictors are the chemical composition of the fragment, and the ninth (RI) is the refractive index.
- The objective is to train a predictive model to predict the glass type based on a fragment of the glass, for forensic purposes.

Example: Try a Binary Classification First

- There are originally 6 categories in the glass type. We consider WinF and WinNF as one type and all the others as another type.



Example: Pruned Binary Classification Tree



- What is the best tree size?
- Which predictors appear to be the most important?

Pros and Cons of Tree Models

- Pros:
  - almost as flexible as neural networks
  - highly interpretable (at least with simple trees)
  - built-in variable importance measure for each predictor
  - automatically discards irrelevant predictors
  - insensitive to monotonic transformation or outliers in predictors
  - computationally efficient to fit
  - can easily handle response variable of more than 2 categories
  - can handle missing predictor values
- Cons:
  - poor at representing linear behavior; results in non-smooth response surface; and predictive power usually not as good as neural networks
  - high variance/instability of fitted tree